

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Інтернет-магазин дизайнерського одягу»

Виконав студент групи ПЗ-22-2
ПЛАТОНОВ Микита Сергійович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

Житомир – 2026

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПЛАТОНОВ Микита Сергійович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«Інтернет-магазин дизайнерського одягу»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: об'єктом дослідження є процес автоматизації продажу товарів та управління контентом у сфері електронної комерції на прикладі магазину дизайнерського одягу. Предметом дослідження є методи, архітектурні підходи та програмні засоби розробки інтернет-магазину дизайнерського одягу.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	01.03.2026р.	01.03.2026р
2	Лобанчикова Н.М.	01.04.2026р	01.04.2026р
3	Лобанчикова Н.М.	01.05.2026р	01.05.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного продукту для продажу дизайнерського одягу та пояснювальної записки. Пояснювальна записка містить 111 сторінок, 43 рисунка, 10 таблиць, 4 додатки та 36 літературних джерел.

Мета роботи – проектування та практична реалізація інтернет-магазину дизайнерського одягу «ELARIUM» на платформі ASP.NET Core MVC з підтримкою онлайн-оплати, управління каталогом та обслуговування клієнтів.

Об'єкт дослідження – процес автоматизації продажу товарів та управління контентом у сфері електронної комерції на прикладі магазину дизайнерського одягу.

Предмет дослідження – методи, архітектурні підходи та програмні засоби розробки інтернет-магазину дизайнерського одягу.

Методами дослідження є системний і порівняльний аналіз вебрішень у fashion-сегменті, об'єктно-орієнтоване проектування, моделювання бази даних, і функціональне тестування розробленого застосунку.

У ході виконання роботи проаналізовано сучасні підходи до побудови дизайнерських інтернет-магазинів, сформульовано вимоги до програмного продукту та виконано порівняння існуючих вебрішень. На цій основі обрано архітектуру вебзастосунку. Визначено варіанти використання системи й побудовано об'єктно-орієнтовану модель даних, що охоплює каталог товарів, замовлення та підсистему автентифікації. Здійснено розробку структури бази даних, реалізацію алгоритмів роботи основних модулів. Усі основні сценарії роботи програмного комплексу пройшли функціональне тестування, результати якого наведено в третьому розділі пояснювальної записки.

КЛЮЧОВІ СЛОВА: ASP.NET CORE, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, MVC, POSTGRESQL.

					ІПЗ.КР.Б – 121 – 26 – ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Платонов М.С.			Інтернет магазин дизайнерського одягу Пояснювальна записка	Літ.	Арк.
							3
Керівник		Лобанчикова Н.М.				Аркушів	
Рецензент		Чижмотря О.В.				115	
Зав. каф.		Вакалюк Т. А.				Житомирська Політехніка Група ІПЗ-22-2	

ABSTRACT

The bachelor's final qualification work consists of a software product for selling designer clothes and an explanatory note. The explanatory note contains 115 pages, 43 figures, 10 tables, 4 appendices, and 35 sources in the list of references.

The purpose of the work is the design and practical implementation of the online store of designer clothes "ELARIUM" on the ASP.NET Core MVC platform with support for online payment, catalog management and customer service.

The object of the study is the process of automating the sale of goods and content management in the field of e-commerce using the example of a designer clothes store.

The subject of the research is methods, architectural approaches and software tools for developing an online designer clothing store.

The research methods are a systematic and comparative analysis of web solutions in the fashion segment, object-oriented design, database modeling, and functional testing of the developed application.

In the course of the work, modern approaches to building designer online stores were analyzed, requirements for the software product were formulated, and existing web solutions were compared. On this basis, the architecture of the web application was selected. Variants of using the system were determined and an object-oriented data model was built, covering the product catalog, orders, and authentication subsystem. The database structure was developed and the algorithms for the main modules were implemented. All the main scenarios of the software complex underwent functional testing, the results of which are given in the third section of the explanatory note.

KEYWORDS: ASP.NET CORE, E-COMMERCE, ONLINE STORE, MVC, POSTGRESQL.

					ИПЗ.КР.Б – 121 – 26 – ИПЗ	Арк.
						4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У СФЕРІ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ДИЗАЙНЕРСЬКОГО ОДЯГУ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	13
1.3 Вибір архітектури веб-орієнтованої системи.....	18
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	21
Висновки до першого розділу.....	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ.....	25
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	25
2.2. Розробка бази даних системи.....	29
2.3. Алгоритми роботи системи та взаємодія модулів	35
2.4. Опис реалізації функціоналу системи	40
2.5. Опис компонентної будови програмного додатку	47
Висновки до другого розділу	50
РОЗДІЛ 3. ВПРОВАДЖЕННЯ ТА ОПИС РОБОТИ ПРОГРАМНОГО КОМПЛЕКСУ.....	52
3.1 Порядок встановлення та налаштування параметрів системи.....	52
3.2 Структура інтерфейсу інтернет-магазину «ELARIUM».....	55
3.3 Тестування роботи програмного комплексу	82
Висновки до третього розділу	90
ВИСНОВКИ.....	92
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	94
ДОДАТКИ.....	98

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних.

ПЗ – Програмне забезпечення.

СКБД – Система керування базами даних (у системі використовується PostgreSQL).

API (Application Programming Interface) – Інтерфейс прикладного програмування для взаємодії між модулями системи.

Auth (Authentication/Authorization) – Система автентифікації та авторизації користувачів і адміністраторів (реалізована на базі ASP.NET Core Identity).

Backend – Серверна частина програмного комплексу (реалізована на C#), що відповідає за бізнес-логіку, обробку замовлень та збереження даних.

CMS (Content Management System) – Система управління контентом.

CRUD (Create, Read, Update, Delete) – Базовий набір операцій управління даними (Створення, Читання, Оновлення, Видалення) в адміністративній панелі.

DI (Dependency Injection) – Впровадження залежностей, патерн проєктування, що використовується в архітектурі системи для зменшення зв'язності коду.

Frontend – Клієнтська частина інтернет-магазину, що відповідає за візуалізацію (HTML, CSS, Bootstrap, JavaScript) та взаємодію з користувачем.

MVC (Model-View-Controller) – Архітектурний патерн (Модель-Представлення-Контролер), що розділяє систему на логічні компоненти.

OAuth (Open Authorization) – Протокол авторизації через Google.

ORM (Object-Relational Mapping) – Об'єктно-реляційне відображення, засіб роботи з базою даних через об'єктну модель (у проєкті використовується Entity Framework Core).

SSR (Server-Side Rendering) – Рендеринг сторінок на стороні сервера (забезпечується технологією Razor у ASP.NET Core) для швидкого завантаження та SEO-оптимізації.

UI (User Interface) – Користувацький інтерфейс сайту.

UX (User Experience) – Досвід користувача під час взаємодії з програмним продуктом.

ВСТУП

Актуальність теми зумовлена тим, що сучасний етап розвитку світової економіки характеризується тотальною цифровізацією та стрімким переходом традиційних форм торгівлі у площину електронної комерції. Глобальні виклики останніх років, зміна споживацької поведінки та розвиток мобільних технологій призвели до того, що наявність функціонального веб-ресурсу стала не просто конкурентною перевагою, а необхідною умовою виживання будь-якого комерційного підприємства. Особливо динамічно ці процеси розвиваються у fashion-індустрії, де якість візуальної презентації виробу, швидкість завантаження сторінок та зручність оформлення замовлення безпосередньо впливають на рішення про покупку. Для брендів дизайнерського одягу наявність власної спеціалізованої вебплатформи поступово перетворилася з конкурентної переваги на необхідну умову повноцінного ведення господарської діяльності.

Локальний бренд дизайнерського одягу «ELARIUM», для якого розробляється програмний продукт, на початковому етапі свого розвитку реалізовував товари переважно через соціальні мережі та месенджери. Із розширенням колекцій такий підхід виявляє низку суттєвих обмежень на прикладі алгоритмічного стиснення фотографій яке не дозволяє повноцінно передати фактуру тканин та якість пошиття, відсутності автоматизованого обліку залишків за розмірами, без можливості реалізувати зручний пошук і фільтрацію товарів, а ручна обробка замовлень у листуванні призводить до затримок та втрати потенційних покупців.

Альтернативою власній розробці є використання універсальних систем управління контентом. Однак такі рішення супроводжуються обмеженою свободою кастомізації дизайну під унікальну айдентику бренду, періодичними платежами за підписку та підвищеними вимогами до ресурсів через значний обсяг службового коду. З огляду на це доцільною є розробка спеціалізованого інтернет-магазину на базі сучасного фреймворку, що дає змогу створити

швидко, безпечно та повністю кероване рішення, адаптоване під конкретні потреби дизайнерського бренду українського ринку.

Метою роботи є проєктування та програмна реалізація інтернет-магазину дизайнерського одягу «ELARIUM» на платформі ASP.NET Core MVC, що забезпечує автоматизацію процесів управління каталогом, презентації колекцій, онлайн-оплати замовлень та обслуговування клієнтів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні тенденції розвитку інформаційних технологій у сфері електронної комерції fashion-індустрії та здійснити порівняльний аналіз існуючих аналогів;
- обґрунтувати вибір архітектури системи та технологічного стеку; сформулювати функціональні та нефункціональні вимоги до програмного продукту;
- розробити фізичну схему бази даних PostgreSQL та реалізувати об'єктно-реляційне відображення засобами Entity Framework Core;
- реалізувати підсистему онлайн-оплати через платіжні шлюзи Stripe та WayForPay із захистом HMAC-підписом;
- впровадити систему автентифікації та авторизації з підтримкою облікових записів Google OAuth 2.0;
- розробити адміністративну панель управління каталогом і замовленнями з аналітичною інформаційною панеллю;
- провести тестування реалізованого програмного комплексу.

Об'єктом дослідження є процес автоматизації продажу товарів та управління контентом у сфері дизайнерського одягу.

Предметом дослідження є методи, архітектурні підходи та програмні засоби розробки інтернет-магазину дизайнерського одягу.

Методами дослідження є методи системного аналізу при дослідженні предметної області та формуванні вимог до системи, методи об'єктно-орієнтованого аналізу та проєктування (OOAD) при побудові архітектури програмного комплексу; методи реляційного моделювання при проєктуванні

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						8

логічної та фізичної структури бази даних; методи функціонального тестування для перевірки сценаріїв взаємодії користувача з системою та коректності роботи модулів системи.

Інформаційною базою роботи є офіційна документація Microsoft з ASP.NET Core MVC [1, 2], Identity [3, 20], Entity Framework Core [4, 21, 22] і механізмів сесій та DI [17, 18]; документація PostgreSQL [6] і Npgsql [7]; матеріали платіжних сервісів Stripe [8, 28] та WayForPay [9]; специфікації Google OAuth 2.0 [10, 19]; а також джерела з Docker [11, 33], Railway [12], UML [13, 35] і рекомендації OWASP щодо веббезпеки [16, 34].

Практичне значення одержаних результатів полягає у створенні повністю працездатного програмного продукту, готового до впровадження у комерційну діяльність бренду дизайнерського одягу «ELARIUM». Розроблений інтернет-магазин дозволяє централізувати управління асортиментом і замовленнями, автоматизувати прийом онлайн-платежів через шлюзи WayForPay та Stripe та забезпечити клієнтам зручний механізм відстеження замовлення за номером і email без реєстрації, а зареєстрованим клієнтам – перегляд історії покупок у розділі «Мої замовлення». Модульна структура програмного коду дозволяє легко розширювати функціонал системи та інтегрувати нові сервіси доставки у майбутньому.

Апробація. За темою кваліфікаційної роботи було опубліковано 2 тези конференції: Платонов М.С., Лобанчикова Н.М. Проєктування інтернет-магазину дизайнерського одягу на базі ASP.NET Core MVC; Застосування сучасних технологій електронної комерції у fashion-ритейлі. Тези Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених «Тенденції розвитку технологій в автоматизації, електроенергетиці, приладобудуванні та робототехніці» (присвяченої Дню науки), м. Житомир, 13-15 травня 2026 р. – Житомир: Житомирська політехніка, 2026. – С. 97. [35, 36].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У СФЕРІ ЕЛЕКТРОННОЇ КОМЕРЦІЇ ДИЗАЙНЕРСЬКОГО ОДЯГУ

1.1 Постановка задачі

Сучасний етап розвитку економіки нерозривно пов'язаний із цифровізацією торгівлі та переходом більшості комерційних процесів у середовище електронної комерції. За даними аналітичних оглядів цифрового ринку, частка онлайн-продажів у загальному обсязі роздрібного товарообігу щороку зростає, а покупці дедалі частіше проходять повний цикл вибору та придбання товару безпосередньо у вебзастосунках, не звертаючись до офлайн-точок продажу. Особливо динамічно ці зміни відбуваються в індустрії моди, де якість візуальної презентації виробу, швидкість завантаження сторінок та зручність оформлення замовлення безпосередньо впливають на рішення про покупку. Для брендів дизайнерського одягу наявність власної спеціалізованої вебплатформи поступово перетворилася з конкурентної переваги на необхідну умову повноцінного ведення господарської діяльності.

Локальний бренд дизайнерського одягу «ELARIUM», для якого розробляється програмний продукт, на початковому етапі свого розвитку реалізовував товари переважно через соціальні мережі та месенджери. Такий підхід виправдовує себе лише за невеликого асортименту та незначної кількості замовлень. Із розширенням колекцій соціальні платформи виявляють низку суттєвих обмежень у вигляді алгоритмічного стиснення фотографій яке не дозволяє повноцінно передати фактуру тканин та якість пошиття, відсутності автоматизованого обліку залишків за розмірами, без можливості реалізувати зручний пошук і фільтрацію товарів, а ручна обробка замовлень у листуванні призводить до затримок, помилок та втрати потенційних покупців. Окремою проблемою є приймання оплати, переказ коштів «навздогін» не дає клієнтові звичного для електронної комерції відчуття захищеної та миттєвої транзакції, а бренду – структурованого обліку продажів.

Альтернативою власній розробці є використання універсальних систем управління контентом, таких як Shopify чи WooCommerce. Однак такі рішення часто супроводжуються надмірною технічною складністю, обмеженою свободою кастомізації дизайну під унікальну айдентику бренду, періодичними платежами за підписку та підвищеними вимогами до ресурсів через значний обсяг службового коду. З огляду на це доцільною є розробка спеціалізованого інтернет-магазину на базі сучасного фреймворку, що дає змогу створити швидке, безпечне та повністю кероване рішення, адаптоване під конкретні потреби дизайнерського бренду.

Виходячи з наведеного, у межах випускної кваліфікаційної роботи бакалавра необхідно спроектувати та розробити повнофункціональний інтернет-магазин дизайнерського одягу, який автоматизує процеси презентації колекцій, управління каталогом, оформлення та оплати замовлень, а також їх подальшого супроводу. Для досягнення зазначеного результату програмний комплекс доцільно розділити на логічно завершені підсистеми, кожна з яких вирішує власне коло завдань.

Перша підсистема – управління каталогом та контентом – призначена для уповноважених представників бренду й охоплює повний набір операцій створення, перегляду, редагування та видалення карток товарів. Адміністратор задає назву, детальний опис, ціну, категорію та статтю цільової аудиторії товару, керує переліком доступних розмірів та залишками за кожним із них, а також завантажує за один крок цілу серію фотографій для формування повноцінної галереї. Окремо реалізується керування промо-контентом – відеоматеріалом показу колекції (дефіле) та його текстовим описом.

Друга підсистема – презентації товарів та взаємодії з відвідувачем – відповідає за клієнтську частину застосунку. Вона забезпечує відображення головної сторінки з відеобанером та добіркою актуальних колекцій, навігацію структурованим каталогом із багатокритеріальною фільтрацією за категорією, статтю, ціновим діапазоном і наявністю конкретного розміру, а також текстовий пошук та сортування результатів. Детальна сторінка товару надає

інтерактивну галерею з масштабуванням зображення, вибір розміру з урахуванням залишків і позначенням розпроданих позицій, а також можливість додати товар до списку бажаного.

Третя підсистема – кошика, оформлення та оплати замовлень – реалізує наскрізний сценарій покупки. Обрані товари накопичуються в сесійному кошику з перевіркою залишку за розміром, після чого клієнт заповнює форму з контактними даними та адресою доставки, що проходить серверну валідацію. Оплата здійснюється онлайн через інтегровані платіжні шлюзи, після успішного підтвердження якої формується замовлення, автоматично зменшуються залишки відповідних розмірів, а товар, усі розміри якого розпродано, переводиться у статус недоступного.

Четверта підсистема – супроводу замовлень та обслуговування користувачів – забезпечує відстеження статусу замовлення. Неавторизований відвідувач може перевірити стан покупки за її номером та електронною поштою, тоді як зареєстрований клієнт отримує доступ до особистого розділу з історією власних замовлень. На боці бренду адміністратор переглядає перелік усіх замовлень, змінює їхній статус за визначеним життєвим циклом та аналізує показники продажів за допомогою інформаційної панелі.

П'ята підсистема – автентифікації, авторизації та безпеки – відповідає за розмежування прав доступу та захист даних. Система підтримує реєстрацію й вхід за допомогою електронної пошти та пароллю, а також через обліковий запис Google, за потреби надсилає листи відновлення пароллю, розрізняє ролі звичайного користувача й адміністратора та автоматично блокує спроби несанкціонованого доступу до службових функцій. Передбачено також захист від типових вебзагроз, валідацію введених даних та перевірку завантажуваних файлів.

Очікуваним результатом вирішення перелічених завдань є єдиний централізований вебресурс, що повністю замінює ручне ведення продажів у соціальних мережах. Його впровадження надасть бренду «ELARIUM» інструмент для системного управління асортиментом, замовленнями та

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						12

контентом, знизить операційне навантаження, а клієнтам забезпечить зручне й безпечне середовище для детального ознайомлення з колекціями та здійснення покупок.

1.2 Аналіз аналогів програмного продукту

Проектування сучасної інформаційної системи передбачає попереднє дослідження ринку та аналіз наявних рішень. У сфері електронної комерції такий аналіз дозволяє визначити галузевий стандарт функціональних можливостей, дослідити усталені патерни взаємодії з користувачем та виявити недоліки, яких слід уникнути у власній розробці. Для порівняння обрано три вебплатформи, що працюють у ніші дизайнерського й вуличного одягу та представляють різні масштаби й підходи до організації онлайн-продажів: міжнародний преміальний бренд «Acne Studios», концептуальний бренд високого цінового сегмента «Rick Owens» та локальну платформу продажу фірмового одягу «Optimus Gang».

Платформа бренду «Acne Studios» (acnestudios.com) є зразком зрілого рішення міжнародного рівня. Вона надає розгалужений каталог із багаторівневою навігацією через випадне «мега-меню», у якому товари згруповано за статтю, типом виробу та колекціями. Сторінка каталогу підтримує комбіновану фільтрацію за кольором, розміром, ціною та категорією, а також сортування результатів. Картка товару виконана в редакторському стилі: вона містить серію великих студійних фотографій із плавним масштабуванням, докладний опис крою та складу матеріалу, інтерактивну розмірну сітку, а також можливість додавання виробу до списку бажаного. Передбачено реєстрацію та особистий кабінет із історією замовлень, локалізований процес оформлення покупки з вибором країни й валюти та кілька способів онлайн-оплати. Сильною стороною платформи є саме повнота функціоналу та якість візуальної подачі, тоді як зворотним боком стає висока ресурсоемність сторінок і надмірна складність каталогу для невеликого бренду.

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						13

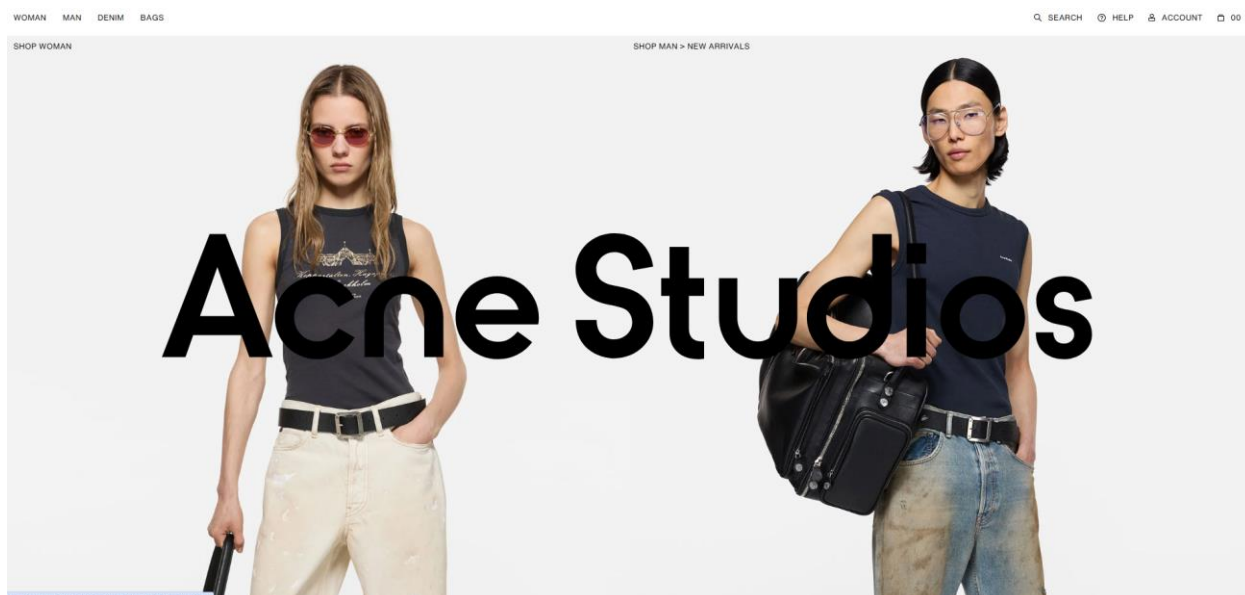


Рис. 1.1 – Головна сторінка вебсайту «Acne Studios»

Онлайн-магазин бренду «Rick Owens» (rickowens.eu) орієнтований на преміальний сегмент і робить акцент на художній, мінімалістичній подачі продукту. Навігація навмисно стримана й підпорядкована візуальному наративу колекції, а каталог побудований навколо великих повноекранних зображень та лукбуків. Картка товару містить серію фотографій високої роздільної здатності з можливістю детального перегляду, лаконічний опис та вибір розміру; функціонал реєстрації, кошика та онлайн-оплати реалізований у класичному для електронної комерції вигляді. Водночас платформа свідомо обмежує можливості фільтрації та пошуку, оскільки робить ставку на послідовний перегляд колекції, а не на швидке знаходження конкретного артикула. Такий підхід забезпечує сильне естетичне враження, проте ускладнює орієнтування за великого асортименту та підвищує вимоги до якості інтернет-з'єднання користувача. Окремої сторінки «Обране» та розширеного пошуку на платформі не передбачено, тому користувач змушений запам'ятовувати цікаві позиції або повертатися до них через історію перегляду браузера. Оформлення замовлення доступне лише після реєстрації, а відстеження статусу покупки потребує входу в особистий кабінет – для разових відвідувачів це додатковий бар'єр порівняно з моделлю «гостьового» замовлення. Платіжна інфраструктура орієнтована насамперед на міжнародні

карткові транзакції в іноземній валюті, що є природним для глобального бренду, але не враховує потреб локального ринку з точки зору звичних українському споживачеві способів оплати.



Рис. 1.2 – Головна сторінка вебсайту «Rick Owens»

Платформа «Optimus Gang» (optimusgang.com) представляє локальний бренд і побудована на базі готової системи управління контентом. Вона має виразну корпоративну айдентику, базовий каталог із поділом на категорії та кошик, реалізований у вигляді бічної панелі, що дозволяє переглядати обрані товари, не залишаючи поточну сторінку. Процес оформлення замовлення сконцентовано на одному екрані з покроковим введенням даних доставки й оплати, а інтеграція з платіжним шлюзом забезпечує приймання коштів у національній валюті. Разом із тим картка товару є інформаційно обмеженою: фотографії мають невисоку деталізацію, опис нерідко зводиться до назви та ціни, відсутні розмірна сітка й розгорнута фільтрація. Використання універсальної системи управління контентом спрощує початковий запуск, але обмежує гнучкість кастомізації та призводить до наявності значного обсягу службового коду, який безпосередньо не використовується. Бічна панель кошика дійсно зручна для швидкого перегляду обраних позицій, проте не дає окремого механізму збереження товарів «на потім» – функція списку бажаного відсутня.

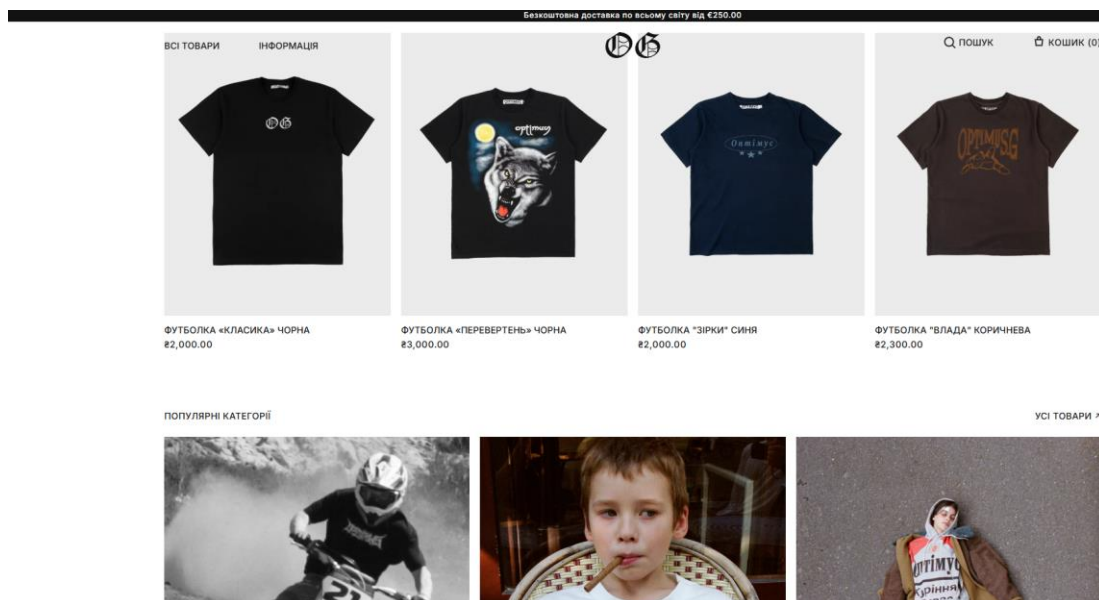


Рис. 1.3 – Головна сторінка вебсайту «Optimus Gang»

Проведений аналіз свідчить, що наявні рішення тяжіють до двох протилежних полюсів. З одного боку, преміальні платформи демонструють бездоганну візуальну подачу та повний набір інструментів електронної комерції, проте є технічно важкими й надмірно складними для молодого бренду. З іншого боку, локальні рішення на базі систем управління контентом запускаються швидко, але поступаються в інформативності карток товару, гнучкості та свободі оформлення. Спільним недоліком значної частини проаналізованих платформ є також відсутність зручного механізму відстеження замовлення без реєстрації та обмежені можливості налаштування саме під специфіку дизайнерського бренду.

Розроблюваний інтернет-магазин «ELARIUM» покликаний поєднати переваги обох підходів, зберігши якісну візуальну презентацію та повноту функціоналу за технічної легкості й повної керованості рішення. Його ключові відмінності полягають у такому. Застосунок розробляється з нуля на фреймворку ASP.NET Core за патерном MVC, завдяки чому в системі відсутній надлишковий службовий код, властивий універсальним платформам. Каталог підтримує багатокритеріальну фільтрацію та облік залишків за окремими розмірами, а детальна сторінка товару доповнена інтерактивною галереєю з масштабуванням та списком бажаного. Підсистема

оплати інтегрує одразу два платіжні шлюзи, зокрема локальний WayForPay, що враховує особливості українського ринку. Передбачено відстеження замовлення гостем за номером та поштою, особистий кабінет із історією покупок, аналітичну панель адміністратора, а також окремий розділ показу колекцій. Узагальнене порівняння функціональних можливостей розглянутих платформ та розроблюваного продукту наведено в табл. 1.1.

Таблиця 1.1

Порівняльна характеристика функціональних можливостей систем
електронної комерції та розроблюваного продукту

Функціональна можливість	Acne Studios	Rick Owens	Optimus Gang	ELARIUM
Управління каталогом (CRUD)	+	+	+	+
Багатокритеріальна фільтрація та пошук	+	–	–	+
Галерея фото з функцією масштабування	+	+	–	+
Облік залишків за розмірами	+	+	–	+
Сесійний кошик користувача	+	+	+	+
Список бажаного (Wishlist)	+	–	–	+
Реєстрація та вхід через Google	+	+	–	+
Онлайн-оплата карткою	+	+	+	+
Локальний платіжний шлюз (WayForPay)	–	–	+	+
Відстеження замовлення гостем	+	–	–	+
Кабінет «Мої замовлення»	+	+	–	+
Аналітична панель адміністратора	+	+	+ (CMS)	+
Окремий розділ показу колекцій	–	–	–	+
Кастомний код без CMS	–	–	–	+

Результати порівняння підтверджують, що розроблюваний інтернет-магазин «ELARIUM» за сукупністю функціональних можливостей не поступається проаналізованим аналогам, а в частині відстеження замовлень гостем, обліку залишків за розмірами та підтримки локального платіжного шлюзу пропонує рішення, повніше адаптоване під потреби українського дизайнерського бренду.

1.3 Вибір архітектури веб-орієнтованої системи

Вибір архітектури є фундаментальним етапом розробки, що визначає здатність системи до масштабування, легкість супроводу та стабільність роботи під навантаженням. Серед поширених архітектурних стилів вебзастосунків розрізняють монолітну архітектуру, за якої всі компоненти системи розгортаються як єдине ціле, та мікросервісну, що передбачає поділ застосунку на незалежні служби з власними процесами життєвого циклу. Мікросервісний підхід доцільний для великих систем зі складними та слабко пов'язаними бізнес-доменами, проте він суттєво підвищує складність розгортання, моніторингу й підтримки, що є невиправданим для проєкту масштабу одного дизайнерського бренду. З огляду на це для інтернет-магазину «ELARIUM» обрано монолітну багаторівневу архітектуру на базі патерну MVC, яка забезпечує оптимальний баланс між простотою розробки та достатнім рівнем структурованості коду [1].

Важливим архітектурним рішенням є також спосіб формування інтерфейсу користувача. На противагу односторінковим застосункам (SPA), де розмітка генерується переважно на боці клієнта, у проєкті застосовано рендеринг на стороні сервера (Server-Side Rendering) засобами рушія Razor. Такий підхід забезпечує швидке первинне завантаження сторінок, спрощує пошукову оптимізацію, що є важливим для торгової платформи, та зменшує обчислювальні вимоги до пристрою користувача, оскільки браузер отримує вже сформований HTML-документ [1].

Патерн MVC поділяє застосунок на три логічні компоненти зі зниженою взаємозалежністю. Модель (Model) представлена класами мови C#, що описують предметні сутності та правила їх обробки. У проєкті «ELARIUM» це, зокрема, класи Product, ProductImage, ProductSizeStock, Order та OrderItem, які засобами Entity Framework Core відображаються на таблиці бази даних. Представлення (View) реалізовано за допомогою Razor-сторінок, що містять розмітку каталогу, детальних сторінок товарів, кошика, форм оформлення та адміністративних розділів. Контролер (Controller) обробляє вхідні HTTP-запити; у системі функціонує низка контролерів, серед яких HomeController відповідає за каталог і детальні сторінки, CartController та WishlistController – за роботу з кошиком і списком бажаного, OrderController – за оформлення, оплати та відстеження замовлень, а ProductsController, OrdersController, AdminController і DefileController – за адміністративні функції.

Для дотримання принципу єдиної відповідальності архітектуру організовано у вигляді кількох логічних рівнів. Рівень доступу до даних базується на Entity Framework Core; його центральним елементом є клас ApplicationDbContext, що успадковує функціональність ASP.NET Core Identity та інкапсулює налаштування підключення до бази даних і опис наборів сутностей. Рівень бізнес-логіки представлений виділеними службами, до яких належать WayForPayService, що формує та підписує дані платіжного запиту, та EmailSender, що відповідає за надсилання електронних листів. Рівень представлення об'єднує контролери, Razor-сторінки та статичні ресурси (каскадні таблиці стилів і клієнтські сценарії), які забезпечують візуалізацію та інтерактивну поведінку інтерфейсу.

Взаємодію компонентів організовано за допомогою вбудованого у платформу .NET контейнера інверсії керування. Залежності, такі як контекст бази даних, служби автентифікації, платіжна служба та компонент надсилання листів, реєструються під час запуску застосунку та автоматично впроваджуються у конструктори контролерів і служб [17]. Такий підхід

знижує зв'язність коду, спрощує заміну реалізацій та підвищує придатність системи до тестування й подальшого розширення.

Обробка кожного запиту відбувається через конвеєр проміжного програмного забезпечення (middleware). До нього входять компоненти перенаправлення на захищений протокол, обслуговування статичних файлів, маршрутизації, керування сесією, автентифікації та авторизації. Окремий проміжний компонент додає до відповідей заголовки безпеки, що обмежують вбудовування сторінок у сторонні ресурси, визначають політику передавання реферера та забороняють доступ до апаратних можливостей пристрою. Стан кошика й списку бажаного зберігається в серверній сесії з обмеженим часом життя, а службові cookie позначаються як недоступні для клієнтських сценаріїв, що зменшує ризик їх компрометації.

Реляційна природа предметних даних зумовила вибір реляційної моделі їх зберігання. Між сутністю товару та його зображеннями, а також між товаром і записами залишків за розмірами встановлено зв'язок «один-до-багатьох»; аналогічно замовлення пов'язане з переліком його позицій. Облікові записи користувачів та ролі зберігаються у стандартних таблицях підсистеми ASP.NET Core Identity. Управління схемою бази даних здійснюється за допомогою механізму міграцій, що дозволяє поступово розвивати структуру сховища без ризику втрати наявних даних.

Особливістю архітектури є її пристосованість до хмарного розгортання. Застосунок працює за зворотним проксі-сервером хмарної платформи, тому для коректного формування захищених адрес перенаправлення опрацьовуються заголовки переадресованого протоколу. Під час запуску система автоматично застосовує наявні міграції до бази даних та за потреби створює початкову роль адміністратора й службовий обліковий запис, що спрощує введення системи в експлуатацію на новому середовищі [12]. Обрана сукупність архітектурних рішень дозволяє створити стабільний, безпечний та зручний у супроводі інтернет-магазин, який відповідає сучасним вимогам до вебзастосунків електронної комерції.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Стабільність, безпека та продуктивність розроблюваного інтернет-магазину безпосередньо залежать від обраного технологічного стеку. Під час добору інструментальних засобів пріоритет надавався сучасним технологіям із відкритою ліцензією, високою продуктивністю та розвиненою екосистемою документації й підтримки.

Базовою мовою програмування обрано С# – об’єктно-орієнтовану мову зі строгою статичною типізацією, що дозволяє виявляти значну частину логічних помилок ще на етапі компіляції, а це особливо важливо для коректності фінансових операцій. Серверну частину реалізовано на платформі .NET 8.0 – сучасній кросплатформній версії фреймворку, яка демонструє високі показники швидкодії в незалежних навантажувальних тестуваннях. Застосування ASP.NET Core MVC забезпечує чіткий поділ відповідальності, автоматичне керування пам’яттю та вбудований захист від поширених вебзагроз.

Для зберігання інформації про товари, користувачів та замовлення обрано об’єктно-реляційну систему керування базами даних PostgreSQL із відкритим кодом, яка вирізняється надійністю, відповідністю принципам цілісності транзакцій та високою продуктивністю під час складних вибірок [6]. Взаємодію коду з базою даних реалізовано через ORM-систему Entity Framework Core за підходом Code First, що дозволяє описувати структуру таблиць безпосередньо класами моделей, формувати запити мовою інтегрованих запитів LINQ та керувати еволюцією схеми за допомогою міграцій [21]. Зв’язок із сервером PostgreSQL забезпечує постачальник Npgsql [7]. Entity Framework Core виступає проміжним шаром між доменними класами та таблицями бази даних, що дозволяє формувати запити мовою LINQ, уникати ручного складання SQL у більшості сценаріїв і централізовано керувати змінами схеми через міграції [21], [22].

Підсистему автентифікації та авторизації побудовано на основі ASP.NET Core Identity, інтегрованої зі сховищем PostgreSQL. Вона забезпечує безпечне зберігання паролів у вигляді хешів, рольову модель розмежування доступу та захист від підробки міжсайтових запитів [3]. Вибір PostgreSQL як сховища даних обґрунтовується не лише відкритою ліцензією та надійністю, а й відповідністю архітектурі хмарного розгортання: платформа Railway надає керований сервіс цієї СУБД із автоматичним формуванням рядка підключення, що спрощує введення системи в експлуатацію [6], [12]. Додатково реалізовано можливість входу через обліковий запис Google за протоколом OAuth 2.0, що спрощує реєстрацію для користувачів [10]. Надсилання листів підтвердження пошти та відновлення паролю здійснюється засобами окремої служби через протокол SMTP.

Приймання онлайн-платежів реалізовано через інтеграцію двох платіжних шлюзів. Шлюз Stripe використовується у режимі попередньо налаштованої платіжної сесії (Checkout) та забезпечує безпечне опрацювання карткових даних на боці провайдера [28]. Платіжний шлюз WayForPay орієнтований на український ринок; взаємодію з ним реалізовано через формування підписаного криптографічним алгоритмом HMAC платіжного запиту та опрацювання серверного зворотного виклику для надійного підтвердження факту оплати [9]. Підтвердження оплати супроводжується ідемпотентним оновленням статусу замовлення та зменшенням залишків відповідних розмірів.

Клієнтську частину побудовано на основі семантичної розмітки HTML5 та каскадних таблиць стилів CSS3 із застосуванням фреймворку Bootstrap для побудови адаптивної сітки інтерфейсу та базових компонентів. Інтерактивну поведінку реалізовано засобами мови JavaScript без залучення важких сторонніх бібліотек: до неї належать ефект плавного згасання відеобанера під час прокручування головної сторінки, масштабування зображень товару під час руху курсора, а також додавання товарів до кошика й списку бажаного без повного перезавантаження сторінки через асинхронні запити.

Розгортання застосунку здійснюється на хмарній платформі Railway з автоматичним складанням контейнера на основі опису Dockerfile та автоматичним повторним розгортанням після надходження змін до репозиторію системи контролю версій Git. Розробка велася в інтегрованому середовищі Visual Studio. Поєднання зазначених засобів забезпечує безперервну інтеграцію та постачання, що скорочує час упровадження оновлень і знижує ймовірність помилок під час розгортання.

Окремо до переліку серверних і клієнтських технологій слід зазначити допоміжні програмні засоби, що забезпечують якість, безпеку та супровід розробки. Система контролю версій Git використовується для зберігання вихідного коду, відстеження змін і координації роботи над проєктом; інтеграція репозиторію з платформою Railway дозволяє автоматизувати цикл збірки та розгортання після кожного оновлення основної гілки. Контейнеризація за допомогою Docker відокремлює застосунок від конфігурації операційної системи хоста й гарантує однакову поведінку програмного продукту в середовищах розробки, тестування та експлуатації. Багатоетапна збірка образу мінімізує розмір кінцевого контейнера шляхом відокремлення етапу компіляції від середовища виконання, що прискорює розгортання на обмежених ресурсах хмарної платформи [11].

Для стабільного функціонування програмного продукту визначено апаратні вимоги. Серверне середовище (та робоча станція розробника) потребує процесора рівня Intel Core i5 або AMD Ryzen 5 із чотирма ядрами, щонайменше 8 ГБ оперативної пам'яті (рекомендовано 16 ГБ для одночасної роботи середовища розробки та сервера бази даних), твердотільного накопичувача обсягом від 20 ГБ вільного простору та стабільного підключення до мережі для роботи з медіаконтентом і платіжними шлюзами. Вимоги до пристрою кінцевого користувача є мінімальними, оскільки основні обчислення виконуються на сервері: достатньо сучасного процесора, 2 ГБ оперативної пам'яті, монітора з роздільною здатністю від 1280×720 пікселів та

актуальної версії одного з браузерів Google Chrome, Mozilla Firefox або Microsoft Edge.

Висновки до першого розділу

У першому розділі здійснено аналіз предметної області та обґрунтовано необхідність автоматизації бізнес-процесів бренду дизайнерського одягу «ELARIUM». Визначено обмеження продажу через соціальні мережі та універсальні системи управління контентом і сформульовано перелік завдань, що мають бути вирішені у вигляді програмного комплексу, поділеного на підсистеми управління каталогом, презентації товарів, оформлення та оплати замовлень, їх супроводу, а також автентифікації й безпеки.

На основі порівняння трьох аналогів – платформ «Acne Studios», «Rick Owens» та «Optimus Gang» – визначено галузевий стандарт функціональних можливостей і виявлено типові недоліки наявних рішень. Це дозволило окреслити відмінності розроблюваного інтернет-магазину, серед яких кастомна реалізація без надлишкового коду, облік залишків за розмірами, відстеження замовлення гостем, підтримка локального платіжного шлюзу та аналітична панель адміністратора.

Обґрунтовано вибір монолітної багаторівневої архітектури на базі патерну MVC із рендерингом на стороні сервера, а також технологічного стеку, що включає мову C#, платформу .NET 8.0, систему керування базами даних PostgreSQL, ORM-систему Entity Framework Core, підсистему ASP.NET Core Identity та інтеграцію платіжних шлюзів. Визначено апаратні вимоги до серверного середовища та пристрою користувача. Сформовані у розділі рішення є основою для проєктування системи, що детально розглядається у наступному розділі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Проектування інтернет-магазину «ELARIUM» розпочинається з формального аналізу вимог до програмного продукту. Цей етап передбачає чітке визначення того, які завдання повинна вирішувати система, яким чином вона взаємодіє з різними категоріями користувачів та які обмеження накладаються на її функціонування. Фундаментом проєктного рішення стала об'єктно-орієнтована модель доменних сутностей, побудована на базі аналізу функціональних і нефункціональних вимог та специфікації варіантів використання.

Функціональні вимоги визначають поведінку системи та специфікують дії, які програмний комплекс повинен виконувати для задоволення потреб різних категорій користувачів. У фінальній версії застосунку «ELARIUM» реалізовано повний набір функцій для трьох ролей, Гість (неавторизований відвідувач), Зареєстрований клієнт та Адміністратор. Функціональні вимоги для ролі Гість охоплюють перегляд головної сторінки з відеобанером та добіркою актуальних колекцій; навігацію каталогом з багатокритеріальною фільтрацією за категорією, статтю, ціновим діапазоном та наявністю конкретного розміру; текстовий пошук за назвою й описом товару; перегляд детальної сторінки товару з інтерактивною галереєю зображень (масштабування 2.5x при наведенні курсора) та вибором розміру з урахуванням залишків; додавання товарів до сесійного кошика; перегляд вмісту кошика, видалення позицій; оформлення замовлення із заповненням контактних даних та адреси доставки; оплату замовлення онлайн через платіжні шлюзи Stripe та WayForPay; відстеження статусу замовлення за його номером та адресою електронної пошти; реєстрацію нового облікового запису [3].

Функціональні вимоги для Зареєстрованого клієнта включають усі можливості Гостя, а також вхід через обліковий запис Google або за допомогою електронної пошти та пароллю; доступ до розділу «Мої замовлення» з повною історією власних замовлень та їхніми статусами; збереження товарів до сесійного списку бажаного (Wishlist) з можливістю подальшого перегляду.

Функціональні вимоги для Адміністратора передбачають захищений вхід через сторінку Login з перевіркою ролі «Admin»; доступ до адміністративної панелі управління; повний набір CRUD-операцій над товарними позиціями; множинне завантаження фотографій галереї за один POST-запит; видалення окремих зображень з галереї з синхронним видаленням фізичного файлу; управління видимістю товарів через прапор IsAvailable без фізичного видалення; перегляд і управління замовленнями з можливістю зміни їхнього статусу за визначеним життєвим циклом (Очікує оплати → Оплачено → В обробці → Відправлено → Доставлено); перегляд аналітичних показників у інформаційній панелі; управління колекцією дефіле (розділ Defile) з відеоматеріалами показу.

Нефункціональні вимоги визначають якісні характеристики системи. Система має генерувати HTML-сторінку на сервері (SSR засобами Razor), час генерації не повинен перевищувати 200–300 мс при звичайному навантаженні; операції з кошиком і Wishlist обробляються через асинхронні AJAX-запити з відгуком до 500 мс без перезавантаження сторінки; підтвердження платежу від шлюзу опрацьовується ідемпотентно.

Паролі зберігаються у вигляді хешів PBKDF2 засобами ASP.NET Core Identity; захист від SQL-ін'єкцій забезпечується через Entity Framework Core з параметризованими запитамі; адміністративні та платіжні форми захищено від CSRF атрибутом [ValidateAntiForgeryToken]; сесійні cookie позначено як HttpOnly та IsEssential; завантажувані файли зображень перевіряються за типом та розміром; підпис платіжних запитів до WayForPay виконується алгоритмом HMAC-MD5 з секретним ключем [9].

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						26

Сесійний кошик і Wishlist зберігаються у Distributed Memory Cache з часом живіння 2 години; каскадне видалення між Products та ProductImages і ProductSizeStock забезпечує цілісність даних; підтвердження оплати від шлюзу та оновлення залишків реалізовано ідемпотентно для уникнення подвійного списання; застосунок автоматично застосовує міграції БД та ініціалізує роль адміністратора при кожному розгортанні.

На основі аналізу реалізованого коду виділено п'ять ключових сценаріїв взаємодії. Сценарій «Перегляд каталогу та деталей товару»: відвідувач звертається до HomeController.Index() з параметрами category, gender, minPrice, maxPrice та searchString. Контролер формує IQueryable-запит з фільтром IsAvailable == true і динамічно додає умови WHERE. При переході на детальну сторінку HomeController.Details() завантажує товар разом із зображеннями галереї та залишками за розмірами через .Include(p => p.Images).Include(p => p.SizeStocks) одним SQL-запитом із JOIN.

Сценарій «Оформлення та оплата замовлення»: клієнт наповнює сесійний кошик, потім переходить до OrderController.Checkout(). Після серверної валідації форми система розраховує загальну суму, зберігає замовлення в БД зі статусом Pending (Очікує оплати) та формує позиції OrderItem. Спосіб оплати (stripe або wayforpay) передається через модель CheckoutViewModel і використовується лише на етапі ініціації платежу; окреме поле PaymentMethod у таблиці Orders не зберігається. Для Stripe в OrderController створюється сесія через Stripe.Checkout.SessionService, ідентифікатор сесії записується в StripeSessionId, після чого клієнт перенаправляється на хостовану сторінку Stripe Checkout. Для WayForPay сервіс WayForPayService.BuildFormFields() формує підписаний набір полів HMAC-MD5, після чого відображається форма перенаправлення на платіжний шлюз. Після успішної оплати для Stripe – метод OrderController.Success() перевіряє session_id і PaymentStatus == "paid"; для WayForPay – WayForPayReturn() (редирект) та/або WayForPayCallback() (server-to-server). У обох випадках спрацьовує

					ІПЗ.КР.Б – 121 – 26 – ІПЗ	Арк.
						27

ідемпотентний метод `MarkOrderPaidAsync()` (або еквівалентна логіка в `Success()`) статус змінюється на `Paid`, зменшуються залишки в `ProductSizeStocks`, очищується сесійний кошик.

Сценарій «Відстеження замовлення» – гість вводить номер замовлення та email на сторінці `OrderController.Track()`. Система перевіряє збіг email із замовленням та відображає його статус і деталі. Зареєстрований клієнт бачить усі свої замовлення в розділі `OrderController.MyOrders()` – фільтрація виконується за полем `CustomerEmail` облікового запису.

Сценарій «Управління каталогом» – адміністратор автентифікується, `middleware` авторизації перевіряє роль «Admin». При створенні товару відбувається двоетапне збереження: спочатку `SaveChangesAsync()` фіксує `Product` і отримує `Id`, потім цикл завантажує файли зображень з унікальними іменами через `Guid.NewGuid()` та зберігає записи `ProductImage` і `ProductSizeStock`.

Сценарій «Управління замовленнями» – адміністратор у панелі `OrdersController.Index()` переглядає весь перелік замовлень з фільтрацією за статусом. Через `OrdersController.UpdateStatus()` змінює статус замовлення, що автоматично показується клієнту у статусі замовлення.

В основі системи «ELARIUM» лежить об'єктна модель предметної області з шести класів. П'ять із них відображаються засобами `Entity Framework Core` за підходом `Code First` на реляційні таблиці `PostgreSQL` [21]; шостий клас описує вміст кошика в серверній сесії і в базі даних не зберігається.

`Product` – центральна сутність каталогу. Вона містить назву, опис, ціну, категорію, стать цільової аудиторії та прапор видимості в каталозі, а також пов'язані колекції зображень і залишків за розмірами. Саме через цей клас система формує картки товарів, фільтрує асортимент і перевіряє наявність обраного розміру.

`ProductImage` – доповнює товар галереєю фотографій. Кожен запис зберігає шлях до файлу зображення і пов'язаний із конкретним товаром

зв'язком «один-до-багатьох»; при видаленні товару відповідні зображення видаляються каскадно.

ProductSizeStock – відповідає за облік залишків у розрізі розмірів. Для кожної пари «товар - розмір» зберігається кількість одиниць на складі; саме ці дані використовуються під час додавання товару до кошика та після підтвердження оплати.

Order – описує оформлене замовлення: контактні дані покупця, місто й адресу доставки, загальну суму, дату створення та поточний статус обробки. Окремого поля для ознаки оплати немає – факт оплати відображається значенням переліку OrderStatus (Pending, Paid, Processing, Shipped, Delivered, Cancelled). Для інтеграції з платіжними сервісами зберігаються ідентифікатор сесії Stripe або код авторизації WayForPay. Зв'язку з таблицеюAspNetUsers через зовнішній ключ немає: замовлення може оформити гість, а для зареєстрованого клієнта історія покупок відбирається за збігом CustomerEmail з email облікового запису.

OrderItem – зберігає окрему позицію замовлення – назву товару, розмір, зображення, ціну та кількість на момент оформлення. Фіксація ціни в момент покупки запобігає зміні суми замовлення після подальшого оновлення каталогу.

CartItem – легка сесійна модель кошика. Вона містить ідентифікатор товару, назву, ціну, зображення, обраний розмір і кількість, зафіксовані на момент додавання до кошика. Завдяки цьому вміст кошика залишається стабільним навіть після змін у каталозі, а сам кошик не навантажує базу даних до моменту оформлення замовлення.

2.2. Розробка бази даних системи

Надійне і продуктивне збереження даних є критичною умовою функціонування будь-якого інтернет-магазину. Для системи «ELARIUM» обрано реляційну СУБД PostgreSQL як сховище даних. Взаємодія програмного коду C# з базою реалізована через ORM-систему Entity

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						29

Framework Core за підходом Code First, що дозволяє описувати структуру таблиць безпосередньо класами моделей та управляти еволюцією схеми за допомогою міграцій без ризику втрати наявних даних [21].

Фізична схема БД складається з двох блоків. Системні таблиці ASP.NET Core Identity, що генеруються автоматично через успадкування ApplicationDbContext від IdentityDbContext<IdentityUser>, та доменні таблиці бізнес-логіки. Нижче наведено опис кожної з доменних таблиць.

Таблиця 2.1

Структура таблиці Products

Поле	Тип (PostgreSQL)	Опис	Обмеження
Id	integer (serial)	Первинний ключ	PK, Identity
Name	text	Назва товару	[Required], [MaxLength(200)]
Description	text	Детальний опис	nullable
Price	numeric(18,2)	Роздрібна ціна	[Required]
Gender	text	Стать цільової аудиторії (Men/Women/Unisex)	nullable
Category	text	Назва категорії рядком	nullable
IsAvailable	boolean	Прапор видимості в каталозі	Default: true

Products – таблиця зберігає основну інформацію про товари каталогу, включаючи назву, ціну, категорію, стать цільової аудиторії та прапор видимості. Прапор IsAvailable дозволяє приховати позицію з каталогу без її фізичного видалення, а зв'язок «один-до-багатьох» із таблицями зображень і залишків забезпечує повну презентацію товару на сайті.

Таблиця 2.2

Структура таблиці ProductImages

Поле	Тип (PostgreSQL)	Опис	Обмеження
Id	integer (serial)	Первинний ключ	PK, Identity
Url	text	Відносний шлях до файлу зображення	Required
ProductId	integer	Зовнішній ключ на таблицю Products	FK, CASCADE

ProductImages – таблиця зберігає шляхи до файлів зображень галереї, пов'язаних із конкретним товаром через зовнішній ключ із каскадним видаленням.

Таблиця 2.3

Структура таблиці ProductSizeStocks

Поле	Тип (PostgreSQL)	Опис	Обмеження
Id	integer (serial)	Первинний ключ	PK, Identity
ProductId	integer	Зовнішній ключ на Products	FK, CASCADE
Size	text	Назва розміру (XS, S, M, L, XL, XXL)	Required
Quantity	integer	Кількість одиниць на складі	Default: 0

ProductSizeStocks - таблиця зберігає залишки товару за кожним розміром окремо, що дозволяє системі відстежувати наявність та автоматично декрементувати кількість після оформлення замовлення.

Таблиця 2.4

Структура таблиці Orders

Поле	Тип (PostgreSQL)	Опис	Обмеження
Id	integer (serial)	Первинний ключ	PK, Identity
CustomerName	text	ПІБ покупця	Required
CustomerEmail	text	Email для зв'язку та «Мої замовлення»	Required
CustomerPhone	text	Номер телефону	Required
City	text	Місто доставки	Required
Address	text	Адреса або відділення НП	Required
TotalAmount	numeric(18,2)	Загальна сума замовлення	Required
Status	integer	Статус (OrderStatus enum)	Default: 0 (Pending)
CreatedAt	timestamp	Дата та час оформлення	Default: UTC now
StripeSessionId	text	ID сесії Stripe Checkout	nullable
PaymentIntentId	text	ID платежу Stripe / authCode WayForPay	nullable

Orders – таблиця зберігає контактні дані покупця, адресу доставки, суму та поточний статус у життєвому циклі замовлення. Спосіб оплати (Stripe / WayForPay) на етапі checkout не персистується в БД. Зв'язок ізAspNetUsers відсутній – ідентифікація клієнта для історії замовлень виконується за CustomerEmail.

Структура таблиці OrderItems

Поле	Тип (PostgreSQL)	Опис	Тип зв'язку	Обмеження
Id	integer (serial)	Первинний ключ	–	PK, Identity
OrderId	integer	FK на Orders	багато-до-одного	FK, CASCADE
ProductId	integer	ID товару в каталозі	–	Required
ProductName	text	Назва на момент покупки	–	Required
ImageUrl	text	Шлях до зображення	–	nullable
Size	text	Обраний розмір	–	Required
Price	numeric(18,2)	Ціна одиниці на момент покупки	–	Required

OrderItems – таблиця зберігає позиції конкретного замовлення з фіксованими на момент оформлення назвою, розміром та ціною товару, що унеможливорює некоректний перерахунок при подальшій зміні каталогу.

Між таблицями системи встановлено такі зв'язки: Products → ProductImages – один-до-багатьох, каскадне видалення; Products → ProductSizeStocks – один-до-багатьох, каскадне видалення; Orders → OrderItems – один-до-багатьох, каскадне видалення; nullable (замовлення може бути оформлене гостем без облікового запису). Такий дизайн забезпечує цілісність даних та підтримку повного циклу замовлення від наповнення кошика до підтвердження оплати та доставки [6].

Для автентифікації й авторизації використовується ASP.NET Core Identity. Тип користувача – IdentityUser. Контекст ApplicationDbContext

успадковує IdentityDbContext<IdentityUser> і працює з PostgreSQL через Npgsql.

При запуску в Program.cs виконується db.Database.Migrate(), створюється роль Admin і службовий обліковий запис адміністратора. Підтвердження email після реєстрації не є обов'язковим (RequireConfirmedAccount = false). Листи підтвердження пошти та відновлення пароля надсилає EmailSender через SMTP за наявності налаштувань.

Замовлення прив'язуються до клієнта за полем CustomerEmail. Розділ «Мої замовлення» відбирає записи за email авторизованого користувача. У формі checkout для авторизованих підставляється email з облікового запису; інші контактні дані вводяться під час оформлення.

Підхід Code First з міграціями Entity Framework Core дозволив поступово нарощувати схему бази даних паралельно з розвитком функціоналу. Кожна міграція фіксує конкретний крок еволюції схеми і може бути застосована або відкочена незалежно. Послідовність міграцій відображає ключові етапи розробки базова таблиця Products; таблиця ProductImages зі зв'язком та CASCADE; розширення Product полями Gender, Category та IsAvailable; таблиця ProductSizeStock для обліку залишків за розмірами; таблиці Order та OrderItem для повного циклу замовлення; інтеграція ASP.NET Core Identity з розширенням ApplicationUser; налаштування зовнішньої автентифікації через Google.

При кожному розгортанні застосунок автоматично застосовує всі незастосовані міграції через context.Database.MigrateAsync(), а також перевіряє та за потреби створює роль «Admin» і обліковий запис адміністратора. Це гарантує готовність системи до роботи на новому середовищі без ручного налаштування [21].

Для забезпечення швидкодії при роботі із взаємопов'язаними даними застосовано стратегію Eager Loading через EF Core. Методи Details() у HomeController та Edit() у ProductsController завантажують товар разом із

зображеннями та залишками за розмірами одним SQL-запитом із JOIN-операціями. Для операцій з каталогом (HomeController.Index()), де зображення галереї і залишки не потрібні, застосовується запит без Include(), що зменшує обсяг даних, які передаються з СУБД. Метод OrdersController.Details() завантажує замовлення разом із позиціями та даними клієнта одним запитом, що забезпечує швидке відображення деталей для адміністратора.

2.3. Алгоритми роботи системи та взаємодія модулів

Архітектура інтернет-магазину «ELARIUM» реалізована на базі патерну MVC платформи ASP.NET Core. Обробка кожного вхідного HTTP-запиту проходить через конвеєр проміжного програмного забезпечення (middleware pipeline), порядок якого визначено у файлі Program.cs: перенаправлення на HTTPS, обслуговування статичних файлів, маршрутизація, сесії, автентифікація, авторизація. Важливо, що UseSession() розташований перед UseAuthentication(), що дозволяє сесійному кошику функціонувати незалежно від стану автентифікації [18].

Представлення (View) отримують дані лише від контролерів і не звертаються до БД напряму. Залежності підключаються через DI у Program.cs – ApplicationDbContext, WayForPayService, IEmailSender<IdentityUser> / EmailSender, UserManager<IdentityUser>, SignInManager<IdentityUser>, RoleManager<IdentityRole> [17].

Маршрутизацію налаштовано так, щоб стартовою була сторінка Landing за замовчуванням {controller=Home}/{action=Landing}/{id?} – стартова сторінка HomeController.Landing(). Каталог – /Home/Index. Razor Pages (MapRazorPages()) обслуговують Identity – вхід, реєстрація, відновлення пароля, Google OAuth. Підтвердження оплати Stripe обробляє OrderController.Success() після redirect з Stripe Checkout.

У pipeline – UseForwardedHeaders, UseHttpsRedirection, UseStaticFiles, UseSession, UseAuthentication, UseAuthorization та HTTP-заголовки безпеки.

UseSession() розташований перед UseAuthentication(), тому кошик і Wishlist працюють і для гостей.

Контролер HomeController реалізує головну сторінку, каталог і детальні сторінки товарів. Метод Landing() завантажує до восьми останніх доступних товарів (IsAvailable == true) разом із зображеннями галереї для відображення на головній сторінці. Алгоритм методу Index() формує базовий LINQ-запит до Products із підключенням .Include(p => p.Images). За замовчуванням застосовується фільтр IsAvailable == true; за параметром stockFilter можливі режими перегляду всіх товарів або лише недоступних. Далі динамічно додаються умови: якщо передано category – фільтрація за категорією (для групових категорій «Денім» та «Верхній одяг» використовується перелік відповідних значень); якщо передано gender – WHERE p.Gender = @gender; якщо передано priceMin та/або priceMax – обмеження діапазону ціни; якщо передано size – перевірка наявності розміру в SizeStocks з Quantity > 0; якщо передано searchString – пошук через .Contains() у полях Name та Description (без урахування регістру). Результируючий запит сортується за параметром sort (за ціною або датою додавання) або, за замовчуванням, за спаданням ідентифікатора (OrderByDescending(p => p.Id)), що забезпечує відображення новіших товарів першими. Ідентифікатори товарів з сесійного Wishlist передаються у представлення через ViewBag.WishlistIds.

Метод Details() отримує товар разом із зображеннями галереї та залишками за розмірами через .Include(p => p.Images).Include(p => p.SizeStocks) одним запитом до БД. У представленні Details.cshtml розміри сортуються за фіксованим порядком (XS–XXL); для кожного розміру відображається наявність за даними ProductSizeStocks – розпродані позиції позначаються як SOLD OUT із блокуванням вибору. Реалізовано інтерактивну галерею зміни головного зображення та ефект масштабування 2.5× при наведенні курсора.

Взаємодію між клієнтом, контролером та базою даних при перегляді каталогу та деталей товару ілюструє діаграма послідовності (рис. 2.1). На ній

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						36

відображено ланцюжок запитів від ініціації браузером HTTP GET-запиту до HomeController.Index() або Details(), фільтрації та вибірки даних через LINQ-запит до PostgreSQL та формування HTML-відповіді засобами Razor-рушія. Окремо показано сценарій Details() із Eager Loading пов'язаних Images та SizeStocks одним SQL-запитом із JOIN.

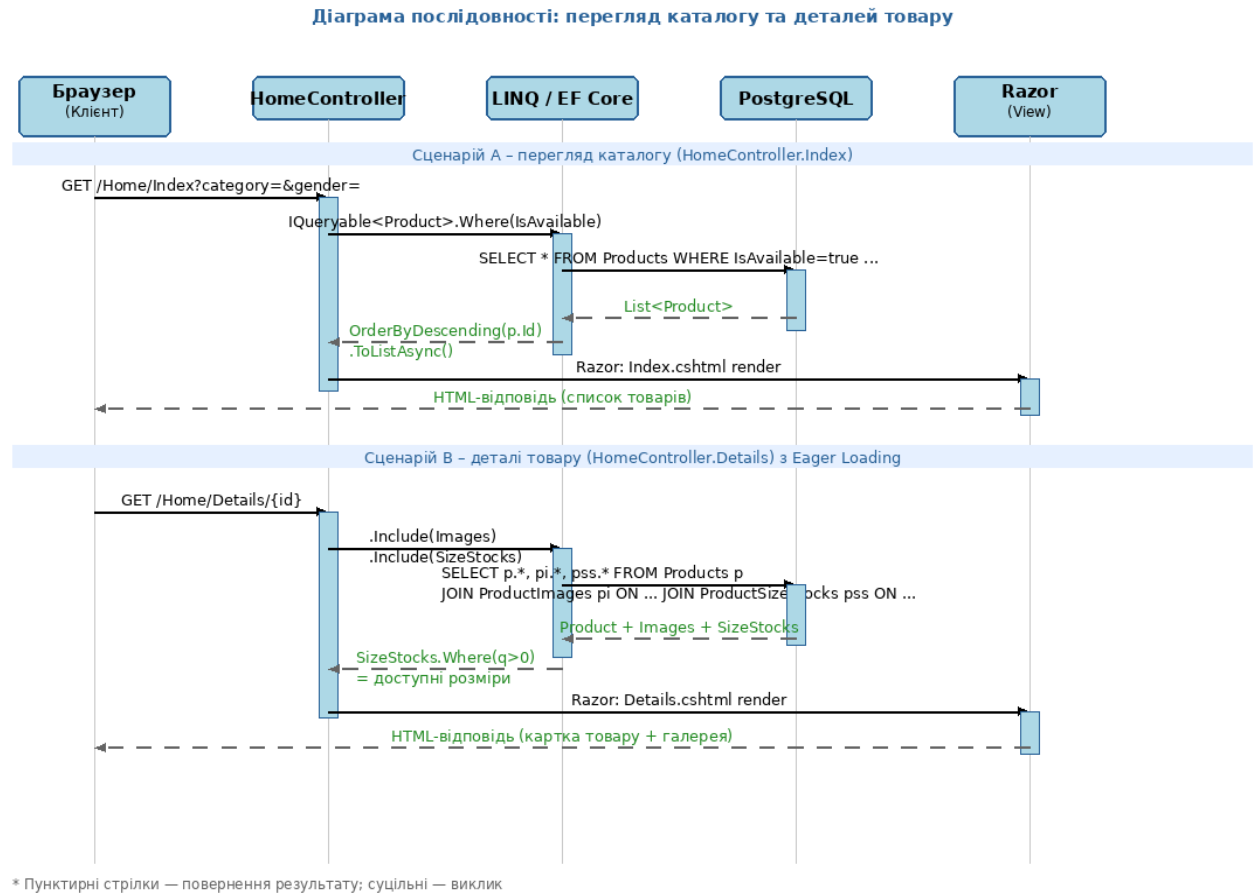


Рис. 2.1 – Діаграма послідовності перегляду каталогу та деталей товару

Кошик реалізований на серверних сесіях ASP.NET Core. Дані серіалізуються в JSON (ключ Cart) і зберігаються в Distributed Memory Cache з часом життя 2 години.

Метод AddToCart() отримує productId та обраний size з POST-запити; виконує пошук товару в БД з SizeStocks; перевіряє наявність відповідного розміру; враховує кількість, уже додану в кошик; при AJAX-запиті (X-Requested-With: XMLHttpRequest) повертає JSON з cartCount. Якщо позиція з

таким ProductId і Size вже є – збільшує Quantity на 1; інакше додає новий CartItem з денормалізованими полями (Name, Price, ImageUrl).

Метод RemoveItem() видаляє позицію за збігом ProductId та Size. Метод ClearCart() очищує сесію. Метод Count() повертає загальну кількість одиниць для бейджа в шапці. На сторінці /Cart користувач переглядає вміст кошика та може видалити позицію

Метод Checkout() [GET] відображає форму оформлення. Для авторизованого клієнта в поле email підставляється адреса з облікового запису (ClaimTypes.Email).

Метод Checkout() [POST] виконує серверну валідацію форми (CheckoutViewModel); розраховує TotalAmount за цінами з кошика; створює об'єкт Order зі статусом Pending та позиції OrderItem; зберігає запис у БД. Залишки на цьому етапі не зменшуються – кошик також ще не очищується.

Залежно від PaymentMethod Stripe (за замовчуванням) через SessionService.CreateAsync() створюється Checkout Session, StripeSessionId зберігається в замовленні, клієнт перенаправляється на session.Url.

WayForPayService.BuildFormFields() формує підписаний HMAC-MD5 набір полів; відображається view WayForPayForm з автопостом на <https://secure.wayforpay.com/pay>. Stripe: після редиректу на /Order/Success?session_id=...&order_id=... метод Success() отримує сесію через SessionService.GetAsync(), перевіряє PaymentStatus == "paid", встановлює Status = Paid, записує PaymentIntentId, викликає зменшення залишків, очищує кошик. Окремий endpoint StripeWebhook не реалізовано. WayForPayReturn() обробляє редирект після оплати; WayForPayCallback() – server-to-server callback. При transactionStatus == "Approved" викликається ідемпотентний MarkOrderPaidAsync() – якщо статус ще Pending, встановлюється Paid, оновлюються залишки; повторні callback не дублюють списання.

Після успішної оплати, якщо всі розміри товару мають Quantity == 0, для продукту встановлюється IsAvailable = false.

Автоматичне email-повідомлення про замовлення після оплати в поточній версії не надсилається. Сервіс EmailSender використовується лише для листів ASP.NET Core Identity (підтвердження пошти, скидання пароля). Інформація про успішне замовлення відображається на сторінці /Order/Success. Надсилення транзакційних листів після покупки заплановано як напрям подальшого розвитку системи.

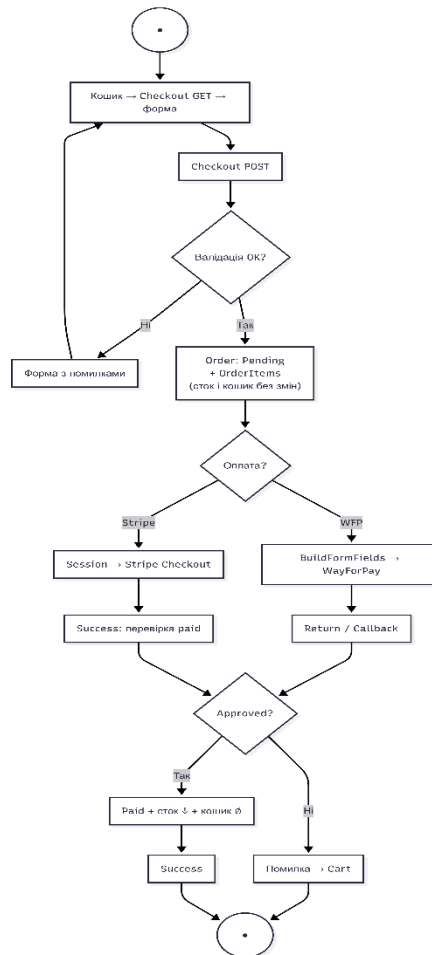


Рис. 2.2 – Діаграма активності оформлення та оплати замовлення

Замовлення зберігається зі статусом Pending; зменшення залишків і очищення кошика – лише після підтвердження оплати (WayForPay – Return/Callback; Stripe – Success).

Адміністративний контролер OrdersController надає повний набір операцій управління замовленнями. Метод Index() відображає повний список усіх замовлень, відсортований за датою створення (спадання), без пагінації та без фільтрації за статусом або діапазоном дат. Метод Details() завантажує замовлення з позиціями. Метод UpdateStatus() [POST] змінює значення

OrderStatus у записі замовлення. Автоматичне email-сповіщення клієнту про зміну статусу не реалізовано – клієнт бачить актуальний статус при перегляді «Мої замовлення» або через /Order/Track. Метод Delete() видаляє замовлення разом із позиціями.

При запуску застосунку в Program.cs виконується блок ініціалізації: перевіряється існування ролі «Admin» та облікового запису адміністратора; якщо відсутні – створюються через RoleManager та UserManager із захешованим паролем і призначенням ролі. Це забезпечує готовність системи без ручного налаштування при кожному розгортанні [12].

Зовнішня автентифікація через Google реалізована за протоколом OAuth 2.0 із використанням пакету Microsoft.AspNetCore.Authentication.Google. При першому вході через Google система автоматично створює запис уAspNetUsers та AspNetUserLogins, зберігаючи провайдер «Google» та унікальний ProviderKey. Подальші входи через Google не потребують введення пароля – система знаходить обліковий запис за провайдером і ключем.

2.4. Опис реалізації функціоналу системи

У цьому параграфі детально описується архітектурна організація програмного коду та реалізація ключових функціональних компонентів інтернет-магазину «ELARIUM».

Програмний комплекс «ELARIUM» організований відповідно до стандартних конвенцій ASP.NET Core MVC. Коренева директорія DesignerStore/ містить такі ключові компоненти: Areas/Identity/Pages/Account/ – Razor Pages для реєстрації, входу та відновлення паролю; Controllers/ – дев'ять контролерів: HomeController.cs, CartController.cs, WishlistController.cs, OrderController.cs, ProductsController.cs, OrdersController.cs, AdminController.cs, DefileController.cs, InfoController.cs; Models/ – доменні класи та моделі представлення; Data/ApplicationDbContext.cs – контекст бази даних; Services/ – WayForPayService.cs, EmailSender.cs; wwwroot/ – статичні

ресурси (CSS, JavaScript, зображення, відеобанер). Такий поділ відповідає принципу єдиної відповідальності: контролери обробляють HTTP-запити, моделі описують дані, служби інкапсують зовнішні інтеграції, а статичні файли обслуговуються безпосередньо веб-сервером.

Центральний клас Product містить усі специфічні для fashion-домену властивості. Поле Gender забезпечує фільтрацію за статтю, рядок Category – за категорією, булевий прапор IsAvailable керує відображенням товару у каталозі. Навігаційні властивості List<ProductImage> Images та List<ProductSizeStock> SizeStocks забезпечують завантаження пов'язаних зображень та залишків за розмірами через Eager Loading. Для захисту від XSS-атак на текстових полях застосовано атрибут [NoHtml]:

```
public class Product
{
    public int Id { get; set; }
    [Required, MaxLength(100), NoHtml] public string Name { get; set; } = string.Empty;
    [NoHtml] public string? Description { get; set; }
    [Required] public decimal Price { get; set; }
    public bool IsAvailable { get; set; } = true;
    [NoHtml] public string? Category { get; set; }
    [NoHtml] public string? Gender { get; set; }
    public List<ProductImage> Images { get; set; } = new();
    public List<ProductSizeStock> SizeStocks { get; set; } = new();
}
```

Клас Order моделює повний цикл замовлення. Перелічення OrderStatus визначає шість станів – від Pending (очікує оплати) до Cancelled (скасовано), що дозволяє адміністратору відстежувати кожен етап обробки. Окремого поля для ознаки оплати немає – факт оплати відображається значенням Status = Paid. Поля StripeSessionId та PaymentIntentId зберігають ідентифікатор сесії Stripe або код авторизації WayForPay для подальшої звірки:

```
public class Order
{
    public int Id { get; set; }
```

```

[MaxLength(100)] public string CustomerName { get; set; } = string.Empty;
[MaxLength(200)] public string CustomerEmail { get; set; } = string.Empty;
public OrderStatus Status { get; set; } = OrderStatus.Pending;
public decimal TotalAmount { get; set; }
public string? StripeSessionId { get; set; }
public string? PaymentIntentId { get; set; }
public List<OrderItem> Items { get; set; } = new();
}
public enum OrderStatus
{
    Pending = 0, Paid = 1, Processing = 2,
    Shipped = 3, Delivered = 4, Cancelled = 5
}

```

Клас `ApplicationDbContext` успадковує `IdentityDbContext<IdentityUser>`, що автоматично підключає системні таблиці ASP.NET Core Identity. Доменні `DbSet`-властивості реєструють усі сутності магазину:

```

public class ApplicationDbContext : IdentityDbContext<IdentityUser>
{
    public DbSet<Product> Products { get; set; }
    public DbSet<ProductImage> ProductImages { get; set; }
    public DbSet<ProductSizeStock> ProductSizeStocks { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<OrderItem> OrderItems { get; set; }
}

```

Зв'язки між сутностями Entity Framework Core визначає автоматично за конвенцією назв. Каскадне видалення для `Products` → `ProductImages` та `Products` → `ProductSizeStocks` налаштовано через Fluent API у методі `OnModelCreating()`, що гарантує цілісність даних при видаленні товару.

Клас `WayForPayService` інкапсулює логіку формування платіжних запитів і верифікації підписів. Метод `Sign()` обчислює HMAC-MD5 підпис від конкатенованого рядка параметрів із секретним ключем, отриманим з конфігурації. Суворий порядок полів у рядку підпису відповідає специфікації

WayForPay: merchantAccount; domain; orderRef; orderDate; amount; currency;
productNames...; productCounts...; productPrices...:

```
public string Sign(params string[] parts)
{
    var message = string.Join(";", parts);
    using var hmac = new HMACMD5(
        Encoding.UTF8.GetBytes(MerchantSecretKey));
    return Convert.ToHexString(
        hmac.ComputeHash(Encoding.UTF8.GetBytes(message))).ToLower();
}
```

Метод BuildFormFields() формує повний перелік полів для POST-форми переадресації на платіжну сторінку WayForPay. Товарні позиції передаються у вигляді масивів productName[], productPrice[], productCount[]. Метод VerifyReturn() виконує зворотну перевірку підпису при обробці відповіді від шлюзу, що унеможливорює підроблення статусу оплати:

```
public bool VerifyReturn(string merchantAccount, string orderReference,
    string amount, string currency, string authCode,
    string cardPan, string transactionStatus,
    string reasonCode, string signature)
{
    var expected = Sign(merchantAccount, orderReference, amount,
        currency, authCode, cardPan, transactionStatus, reasonCode);
    return string.Equals(expected, signature,
        StringComparison.OrdinalIgnoreCase);
}
```

Інтеграція Stripe виконана через офіційний NuGet-пакет Stripe.net. У методі Checkout() [POST] контролера OrderController спочатку виконується серверна валідація форми, після чого зберігається чернетка замовлення у БД. Потім формується об'єкт SessionCreateOptions із масивом LineItems, де ціна передається у копійках (множення на 100), а SuccessUrl містить ідентифікатор замовлення для зв'язки після повернення:

```
var sessionOptions = new SessionCreateOptions
{
```

```

        PaymentMethodTypes = new List<string> { "card" },
        LineItems = lineItems, Mode = "payment",
        SuccessUrl = $"{domain}/Order/Success?session_id={{CHECKOUT_SESSION_ID}}&order_id={order.Id}"
    },
    CancelUrl = $"{domain}/Cart"
};

var session = await new SessionService().CreateAsync(sessionOptions);
order.StripeSessionId = session.Id;

```

Після успішної оплати метод `Success()` отримує від Stripe підтвердження статусу `paid` та виконує зменшення залишків за конкретним розміром через таблицю `ProductSizeStocks`. Якщо запис для даного розміру відсутній, застосовується резервний механізм зменшення загального поля `Stock`. Коли всі розміри товару вичерпані, система встановлює `IsAvailable = false` і очищає сесійний кошик. Логіка зменшення залишків винесена в окремий метод `MarkOrderPaidAsync()`, який також використовується при оплаті через `WayForPay`.

Серверна валідація даних форм забезпечується стандартним механізмом ASP.NET Core `ModelState`. У методі `Checkout()` [POST] перевірка `ModelState.IsValid` виконується до будь-яких операцій із БД. Сума замовлення розраховується виключно з даних сесійного кошика – це унеможливорює маніпуляцію ціною на стороні клієнта:

```
if (!ModelState.IsValid) { ViewBag.Cart = cart; return View(model); }
```

```

var order = new Order
{
    CustomerName = model.Name, CustomerEmail = model.Email,
    TotalAmount = cart.Sum(i => i.Price * i.Quantity), // ціна з сервера
    Status = OrderStatus.Pending, ...
};

```

При завантаженні зображень галереї в `ProductsController` реалізовано дворівневу перевірку файлів. Перший рівень – перевірка розширення та

розміру (до 10 МБ). Другий рівень – перевірка магічних байтів заголовку файлу, що унеможлиблює завантаження шкідливого файлу під виглядом зображення:

```
if (!AllowedExtensions.Contains(ext)) return $"Недозволений формат «{ext}».";
if (file.Length > 10 * 1024 * 1024) return "Файл занадто великий (максимум 10 МБ).";
// перевірка magic bytes заголовка (JPEG, PNG, WebP)
if (!matches) return "Вміст файлу не відповідає розширенню.";
return null;
```

При видаленні зображення метод `DeleteImage()` спочатку видаляє фізичний файл з диску, а потім запис із БД. Відсутність файлу на диску не викидає виключення – запис із БД видаляється в будь-якому разі. Для платіжних операцій реалізовано захист від повторного опрацювання: перед зміною статусу замовлення система перевіряє значення `Status`, і якщо замовлення вже позначено як `Paid`, повторний виклик не зменшує залишки повторно.

Детальна сторінка товару (`Views/Home/Details.cshtml`) реалізує інтерактивну галерею засобами `Vanilla JavaScript` без зовнішніх бібліотек. Ліва вертикальна панель відображає мініатюри головного фото та додаткових зображень з колекції `Model.Images`; клік по мініатюрі викликає функцію `changeImage()`, яка оновлює `src` головного зображення. Контейнер головного фото має `overflow: hidden` і курсор `crosshair`; при русі курсора обчислюються відносні координати у відсотках, задається `transform-origin`, після чого застосовується `scale(2.5)`. Подія `mouseleave` скасовує масштабування. Повний `JavaScript`-код наведено в додатках до пояснювальної записки.

Підсистема `Wishlist` реалізована через `WishlistController`. Список ідентифікаторів товарів зберігається в серверній сесії у форматі `JSON`. Метод `Toggle()` [POST] за наявності `productId` у списку видаляє його, за відсутності – додає, після чого серіалізує оновлений список назад у сесію та повертає `JSON`-відповідь для оновлення інтерфейсу без перезавантаження сторінки:

```
[HttpPost]
public IActionResult Toggle(int productId)
```

```

{
    var ids = GetIds();
    bool added = ids.Contains(productId) ? (ids.Remove(productId), false).Item2
        : (ids.Add(productId), true).Item2;

    SaveIds(ids);
    return Json(new { added, count = ids.Count });
}

```

Рядок підключення до PostgreSQL зчитується з файлу конфігурації appsettings.json за ключем DefaultConnection та передається провайдеру Npgsql через метод UseNpgsql(). Сесії налаштовано в Program.cs із часом неактивності дві години, прапором HttpOnly та іменем .ELARIUM.Session. Налаштування куків авторизації визначають термін дії 14 днів із ковзним поновленням. Конфігурація ForwardedHeaders забезпечує коректне формування URL-адрес при роботі за зворотним проксі Railway:

```

builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromHours(2);
    options.Cookie.HttpOnly = true;
    options.Cookie.Name = ".ELARIUM.Session";
});

```

Google OAuth 2.0 підключається умовно – лише якщо в конфігурації присутні обидва ключі ClientId та ClientSecret. Це дозволяє запускати застосунок у локальному середовищі розробки без налаштованих OAuth-облікових даних. Dockerfile реалізує патерн багатоетапної збірки: перший етап на базі dotnet/sdk:8.0 виконує restore та publish; другий, фінальний, базується на dotnet/aspnet:8.0 і містить лише скомпільовані артефакти:

```

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
RUN dotnet publish "DesignerStore.csproj" -c Release -o /app/publish
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS final
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "DesignerStore.dll"]

```

При розгортанні на Railway змінні середовища (рядок підключення, ключі Stripe, WayForPay, облікові дані SMTP) задаються через панель

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						46

керування платформи та автоматично перекривають значення з appsettings.json. Автоміграція бази даних виконується під час запуску через db.Database.Migrate(), що унеможливило розбіжність між схемою БД та кодом моделей після кожного деплою (повний Program.cs наведено у Додатку А).

2.5. Опис компонентної будови програмного додатку

Компонентна будова застосунку «ELARIUM» відображає ієрархічну організацію структурних одиниць програмного коду та їх взаємодію. Система складається з трьох логічних рівнів – рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Такий поділ не передбачає фізичного розділення на окремі процеси чи служби: усі компоненти входять до єдиного монолітного застосунку ASP.NET Core, проте логічна сегментація спрощує супровід коду, локалізацію змін і подальше розширення функціоналу.

Рівень представлення (Presentation Layer) об'єднує MVC-контролери, Razor-шаблони та Razor Pages підсистеми Identity. Клієнтська частина включає HomeController із шаблонами Landing.cshtml (головна сторінка з відеобанером і добіркою колекції), Index.cshtml (каталог із багатокритеріальною фільтрацією та пошуком) та Details.cshtml (картка товару з інтерактивною галереєю, вибором розміру та кнопкою Wishlist); CartController із шаблоном Index.cshtml (перегляд позицій кошика, видалення товарів, перехід до оформлення); WishlistController із шаблоном Index.cshtml (перегляд обраних товарів із сесійного списку); OrderController із шаблонами Checkout.cshtml (форма доставки та вибір платіжного шлюзу), Success.cshtml (підтвердження успішного замовлення), MyOrders.cshtml (історія замовлень авторизованого користувача), Track.cshtml та TrackResult.cshtml (відстеження замовлення за номером і email); InfoController із шаблонами інформаційних сторінок (About, Contacts, FAQ, Delivery). Сторінки реєстрації, входу, відновлення пароля та зовнішньої автентифікації Google реалізовані як Razor Pages у каталозі Areas/Identity/Pages/Account/ (Login, Register, ForgotPassword, ResetPassword, ExternalLogin тощо).

Усі публічні сторінки використовують спільний макет Views/Shared/_Layout.cshtml, який формує навігаційний хедер, пошуковий рядок, іконки кошика та Wishlist з лічильниками, а також підключає каскадні таблиці стилів і скрипти валідації. Для адміністративних розділів застосовано атрибут [Authorize(Roles = "Admin")], що обмежує доступ до CRUD-операцій і аналітики лише користувачами з відповідною роллю. Передача даних від контролера до шаблону здійснюється через strongly-typed моделі (@model) або ViewBag/ViewData – наприклад, для збереження активних параметрів фільтрації каталогу чи ідентифікаторів товарів у Wishlist.

Адміністративна частина рівня представлення включає ProductsController (CRUD-інтерфейс управління каталогом, завантаження галереї зображень, керування залишками за розмірами), OrdersController (перегляд переліку замовлень та зміна їхніх статусів), AdminController (аналітична панель із показниками продажів), DefileController (управління відеоматеріалом та текстовим описом розділу дефіле).

Рівень бізнес-логіки (Business Logic Layer) представлений виділеними службами та методами контролерів. WayForPayService відповідає за формування підписаних платіжних запитів до шлюзу WayForPay (алгоритм HMAC-MD5) та обробку зворотних викликів. Ініціація платежу через Stripe виконується безпосередньо в OrderController за допомогою бібліотеки Stripe.net (SessionService). EmailSender реалізує інтерфейс IEmailSender<IdentityUser> і забезпечує надсилання листів відновлення пароля через SMTP; Логіка управління сесійним кошиком інкапсульована в допоміжних методах GetCart() та SaveCart() класу CartController; аналогічно список бажаного обробляється методами GetIds() та SaveIds() у WishlistController. Дані кошика та Wishlist зберігаються в серверній сесії у форматі JSON і не дублюються в окремих таблицях бази даних.

Суттєву частину бізнес-логіки зосереджено в OrderController: створення замовлення зі статусом Pending, ініціація Stripe Checkout або формування полів WayForPay, ідемпотентне оновлення статусу оплати в

MarkOrderPaidAsync(), зменшення залишків у ProductSizeStocks та очищення сесійного кошика після успішної транзакції. У HomeController реалізовано побудову динамічних LINQ-запитів для фільтрації каталогу; у ProductsController – двоетапне збереження товару з подальшим завантаженням файлів галереї; у AdminController – агрегацію показників продажів для передачі в DashboardViewModel. Моделі представлення (CheckoutViewModel, DashboardViewModel) відокремлюють структуру форм і аналітичних даних від доменних сутностей бази даних.

Рівень доступу до даних (Data Access Layer) базується на Entity Framework Core із класом ApplicationDbContext як єдиною точкою взаємодії з PostgreSQL. Контекст успадковує IdentityDbContext<IdentityUser> і містить набори сутностей Products, ProductImages, ProductSizeStocks, Orders, OrderItems. Усі запити до БД виконуються асинхронно через методи ToListAsync(), FirstOrDefaultAsync() та SaveChangesAsync(). Застосування Eager Loading через .Include() забезпечує завантаження пов'язаних даних одним SQL-запитом із JOIN, що унеможливлює проблему N+1 у критичних методах отримання деталей товару та замовлення.

Окрім доменних таблиць, у PostgreSQL автоматично створюються системні таблиці ASP.NET Core Identity (AspNetUsers, AspNetRoles, AspNetUserRoles тощо), оскільки контекст успадковує IdentityDbContext. Еволюція схеми бази даних керується міграціями EF Core; при запуску застосунку виконується Database.Migrate(), що гарантує узгодженість структури БД із програмними моделями на сервері розгортання. Для аналітичних запитів у AdminController використовується AsNoTracking(), що зменшує накладні витрати на відстеження змін сутностей під час читання великих наборів замовлень.

Взаємозв'язки між компонентами системи «ELARIUM» унаочнює діаграма компонентів (рис. 2.3). На діаграмі виділено три логічні рівні: представлення (Razor-шаблони, MVC-контролери та Razor Pages Identity), бізнес-логіку (служби WayForPayService, EmailSender, логіка OrderController,

Stripe.net, а також сесійне зберігання кошика й Wishlist) та доступ до даних (ApplicationDbContext, EF Core, PostgreSQL). Окремо показано доменні моделі, статичні ресурси wwwroot з AJAX-запитами до контролерів, підсистему ASP.NET Core Identity та зовнішні інтеграції: Stripe Checkout, WayForPay (HMAC), Google OAuth 2.0 і SMTP. Блок розгортання відображає контейнеризацію Docker і публікацію на Railway. Стрілки ілюструють напрям викликів і потік даних між компонентами.

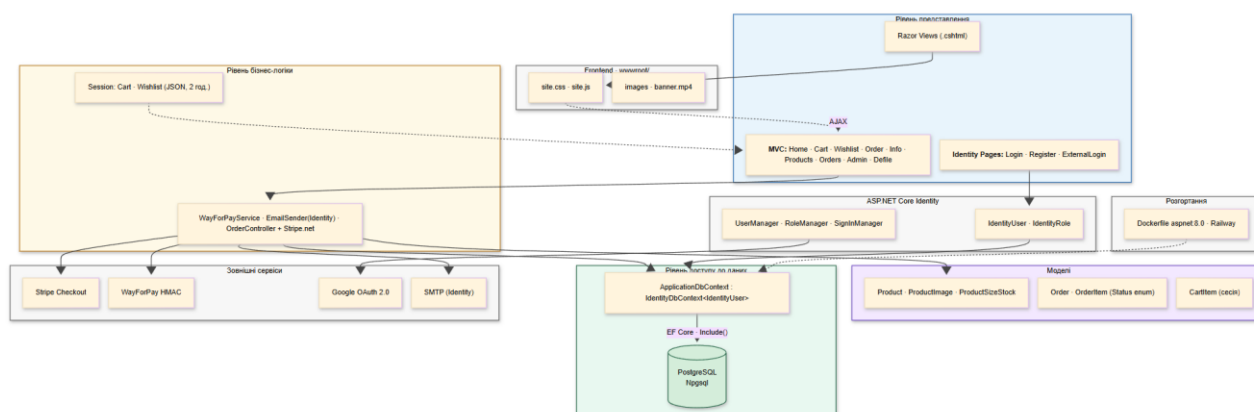


Рис. 2.3 – Діаграма компонентів системи ELARIUM

Статичні ресурси фронтенду (CSS, JavaScript, зображення, відеобанер) розміщено у директорії wwwroot/. Стили організовано у файл site.css, де визначено змінні CSS для кольорів бренду, адаптивну сітку Bootstrap, а також специфічні компоненти: анімована навігаційна панель, картки товарів із ефектом наведення, вертикальна панель мініатюр галереї. JavaScript організовано у файл site.js; критичні скрипти (ефект масштабування, AJAX-запити Wishlist та кошика) вбудовано безпосередньо у відповідні Razor-шаблони задля мінімізації кількості HTTP-запитів при завантаженні сторінки.

Висновки до другого розділу

У другому розділі здійснено повне проєктування системи інтернет-магазину «ELARIUM» відповідно до реалізованого програмного продукту.

Визначено функціональні вимоги для трьох ролей – Гостя, Зареєстрованого клієнта та Адміністратора – та сформульовано нефункціональні вимоги щодо продуктивності, безпеки та надійності системи. Специфіковано п'ять ключових варіантів використання – перегляд каталогу, оформлення та оплата замовлення, відстеження замовлення, управління каталогом та управління замовленнями.

Об'єктно-орієнтована структура включає шість реалізованих сутностей – Product (з підтримкою Gender, Category, IsAvailable та залишків за розмірами), ProductImage, ProductSizeStock (облік кількості за розмірами), Order і OrderItem (повний цикл замовлення) та CartItem (сесійна модель без таблиці в БД). Розроблена фізична схема PostgreSQL забезпечує цілісність даних через каскадні видалення та зовнішні ключі.

Описано алгоритми роботи дев'яти контролерів системи, включаючи складні сценарії двоетапного підтвердження оплати через Stripe (Webhook) та WayForPay (серверний зворотний виклик із HMAC-верифікацією). Для наочного представлення архітектурних рішень наведено три UML-діаграми: діаграма послідовності ілюструє взаємодію компонентів при перегляді каталогу; діаграма активності відображає повний алгоритм оформлення та оплати замовлення; діаграма компонентів показує структуру та залежності модулів системи. Описано реалізацію багаторівневої обробки помилок та валідації вхідних даних. Компонентна будова трирівневої архітектури забезпечує чіткий поділ відповідальності, що спрощує супровід і подальше розширення системи.

РОЗДІЛ 3. ВПРОВАДЖЕННЯ ТА ОПИС РОБОТИ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Порядок встановлення та налаштування параметрів системи

Третій розділ присвячено практичному аспекту впровадження розробленого програмного комплексу, порядку його встановлення та налаштування у виробничому середовищі, а також детальному опису інтерфейсу користувача із зазначенням функціонального призначення кожного екрана.

Розгортання інтернет-магазину «ELARIUM» організовано за принципом контейнеризації з використанням технології Docker та хмарної платформи Railway. Така модель забезпечує повну відтворюваність виробничого середовища, ізоляцію залежностей від операційної системи хоста та можливість миттєвого відкату до попередньої версії. Вибір Railway обумовлений нативною підтримкою GitHub-інтеграції з автоматичним запуском CI/CD-конвеєру після кожного коміту до основної гілки репозиторію, а також наявністю вбудованого провайдера PostgreSQL.

Точкою входу в процес збірки є файл Dockerfile у корені репозиторію. Його структура реалізує патерн багатоетапної збірки (multi-stage build), що мінімізує розмір кінцевого образу шляхом відокремлення середовища компіляції від середовища виконання.

Перший етап базується на образі .NET SDK 8.0 та виконує відновлення пакетів NuGet і публікацію застосунку у конфігурації Release. Другий, кінцевий, етап базується на легшому образі ASP.NET Runtime 8.0 та містить лише скомпільовані артефакти. Порт 8080 оголошується як внутрішній порт Kestrel-сервера. Завдяки кешуванню шарів Docker повторні збірки при незміненому файлі залежностей виконуються суттєво швидше.

Загальна архітектура розгортання системи «ELARIUM» відображає взаємодію між ключовими вузлами виробничого середовища та потік даних

від браузера користувача до постійного сховища. Діаграму розгортання у нотації UML 2.0 наведено на рис. 3.1.

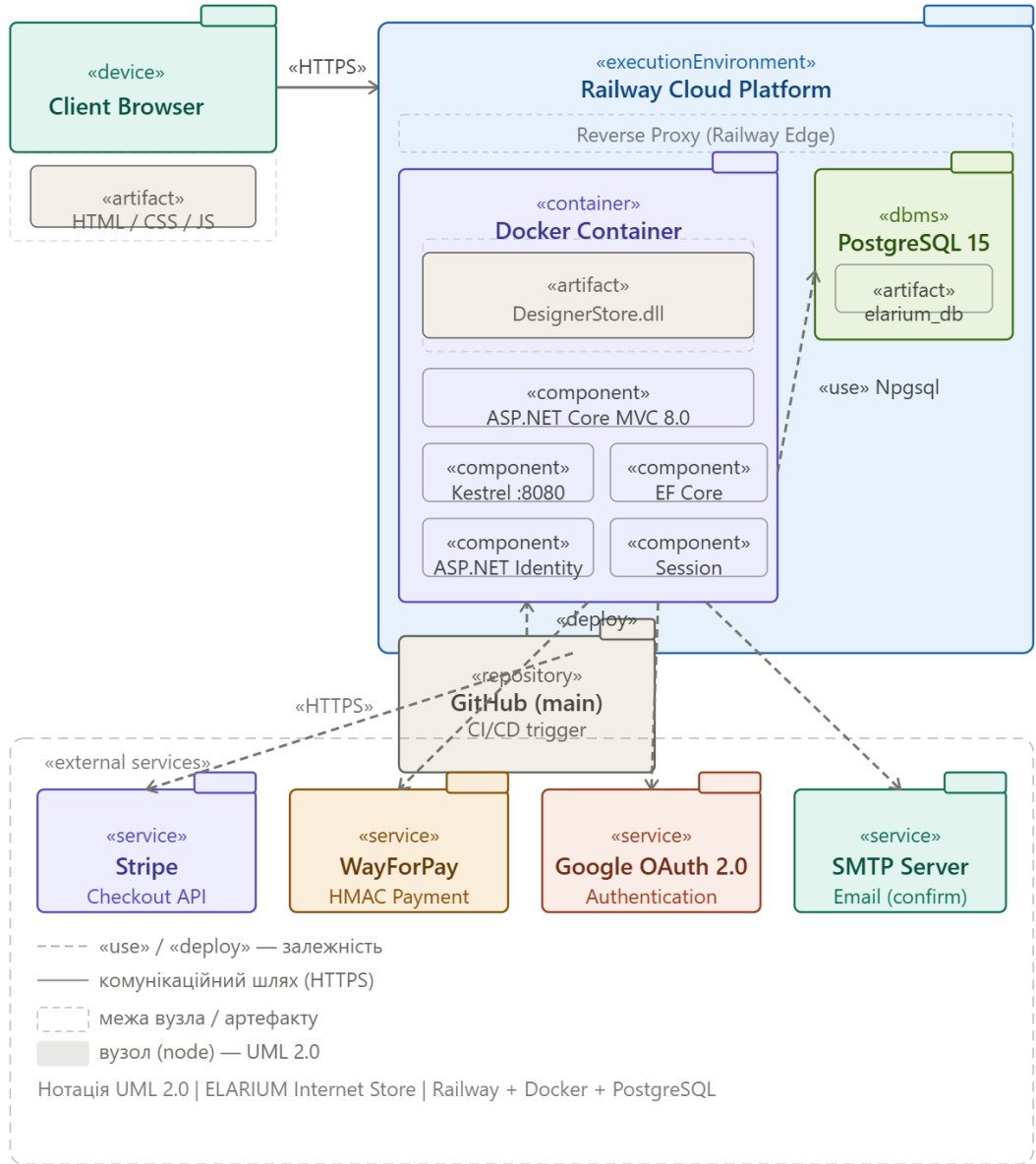


Рис. 3.1 – Діаграма розгортання системи ELARIUM

Архітектура включає чотири групи вузлів. Перший вузол – пристрій клієнта («device»), у якому виконується браузер, що відображає HTML/CSS/JS та надсилає HTTPS-запити до платформи. Другий вузол – «executionEnvironment» Railway Cloud Platform, у якому розміщено два сервіси: Docker-контейнер з виконуваним артефактом DesignerStore.dll та окремий сервіс СКБД PostgreSQL 15. Docker-контейнер позначено

стереотипом «container» і містить чотири компоненти – вебсервер Kestrel на порту 8080, рушій ASP.NET Core MVC 8.0, ORM-шар Entity Framework Core із постачальником Npgsql для з'єднання з PostgreSQL, а також підсистему ASP.NET Core Identity та менеджер сесій. Зворотний проксі-сервер Railway Edge приймає зовнішні HTTPS-запити і передає їх на внутрішній порт контейнера, забезпечуючи коректне відновлення схеми https:// через заголовки X-Forwarded-Proto.

Третій вузол – репозиторій GitHub («repository»), кожен коміт до гілки main якого ініціює CI/CD-конвеєр – Docker-образ збирається на основі Dockerfile, новий контейнер запускається поряд зі старим і після підтвердження готовності трафік перемикається без простою (zero-downtime). Четвертий вузол – зовнішні сервіси («external services»): платіжний шлюз Stripe (Checkout API), платіжний шлюз WayForPay (HMAC-захищений запит та серверний зворотний виклик), провайдер автентифікації Google OAuth 2.0 та SMTP-сервер для надсилання листів підтвердження електронної пошти й відновлення пароля.

Усі взаємодії між вузлами здійснюються виключно через захищений протокол HTTPS; внутрішнє з'єднання між контейнером ASP.NET Core та сервісом PostgreSQL на Railway відбувається через захищений внутрішній мережевий міст платформи з використанням рядка підключення, переданого через змінну середовища.

Конфігурація застосунку організована відповідно до методології «12-Factor App» – усі чутливі параметри задаються виключно через змінні середовища у панелі керування Railway, а не у файлах під контролем версій. Рядок підключення до PostgreSQL Railway встановлює автоматично. Ключі платіжних шлюзів Stripe та WayForPay, реквізити Google OAuth та параметри SMTP-сервера задаються вручну у розділі Variables проєкту. Завдяки ієрархічній системі конфігурації ASP.NET Core змінні середовища мають вищий пріоритет над значеннями з appsettings.json.

Після підключення репозиторію до Railway кожен коміт до гілки main автоматично ініціює новий цикл збірки без ручного втручання. Платформа виконує docker build на основі Dockerfile, запускає новий контейнер поряд зі старим, після підтвердження готовності перемикає трафік без простою (zero-downtime deployment) та завершує попередню версію.

При першому запуску нової версії в Program.cs автоматично виконуються міграції бази даних, перевіряється наявність ролі «Admin» та облікового запису адміністратора і за їх відсутності вони створюються. Це гарантує готовність системи на будь-якому новому середовищі без додаткових ручних дій.

Для локальної розробки або тестування необхідні – .NET SDK 8.0, PostgreSQL 15, Git та Visual Studio 2022 або VS Code. Послідовністю дій буде клонування репозиторію; задання рядка підключення до локальної БД у appsettings.Development.json; задання ключів зовнішніх сервісів через secrets.json або змінні середовища; виконання dotnet ef database update для застосування міграцій; запуск через dotnet run або кнопку запуску в IDE. Після успішного старту застосунок доступний за адресою <https://localhost:5001>.

3.2 Структура інтерфейсу інтернет-магазину «ELARIUM»

Підрозділ містить опис ключових екранів та елементів інтерфейсу клієнтської частини застосунку. У місцях, позначених як «Рис. 3.X», передбачено розміщення скріншотів відповідних сторінок.

Головна сторінка (маршрут /) є першою точкою контакту відвідувача з брендом. Верхня частина повністю відведена під повноекранний відеобанер (100vw × 100vh), що автоматично відтворюється без звуку в циклі. Поверх відео по центру розміщено логотип бренду «ELARIUM» у векторному форматі. При прокручуванні сторінки відеобанер плавно зменшує прозорість завдяки Parallax Fade ефекту, реалізованому засобами JavaScript, забезпечуючи плавний перехід до каталогу товарів (рис. 3.2).

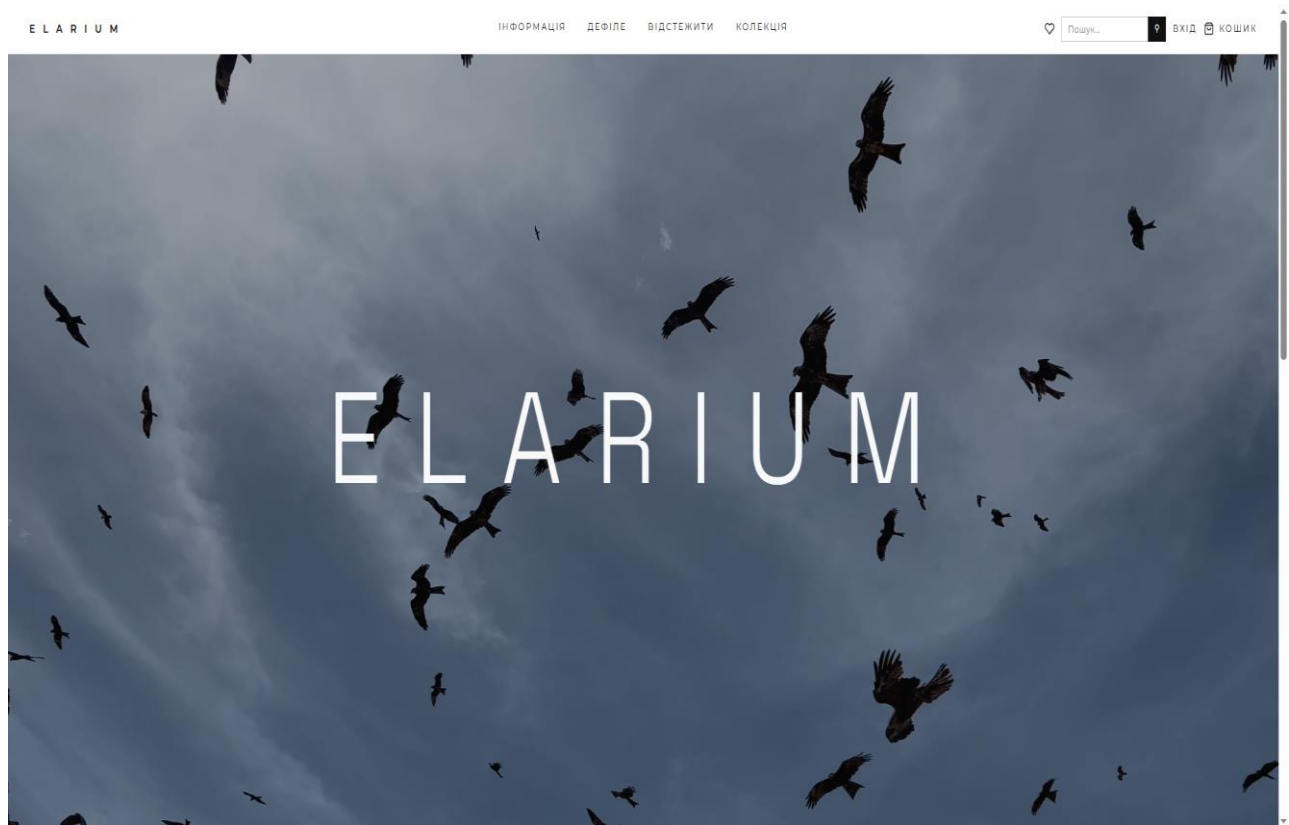


Рис. 3.2 – Головна сторінка з відеобанером та навігаційним хедером

Хедер (навігаційна панель) постійно присутній у верхній частині екрана при будь-якому прокручуванні. Ліворуч розміщено логотип-посилання на головну сторінку.

У правій частині – основне меню з такими елементами: посилання на каталог товарів; посилання «Дефіле» на розділ відеопоказів; випадаюче меню «Інформація»; іконка пошуку; іконка списку бажаного з лічильником; іконка кошика з кількістю позицій; кнопка «Увійти» або аватар профілю для автентифікованих користувачів. На мобільних пристроях хедер згортається у меню-«гамбургер» засобами Bootstrap Navbar.

Нижче відеобанера розміщено секцію «Нова колекція» – промо-блок з заголовком та кнопкою «Переглянути колекцію», що веде до каталогу. Ключовим елементом секції є горизонтальний роллер товарів поточної колекції, реалізований на CSS Flexbox зі скролінгом жестом на мобільних пристроях. На великих екранах одночасно видно чотири–шість карток товарів (рис. 3.3).

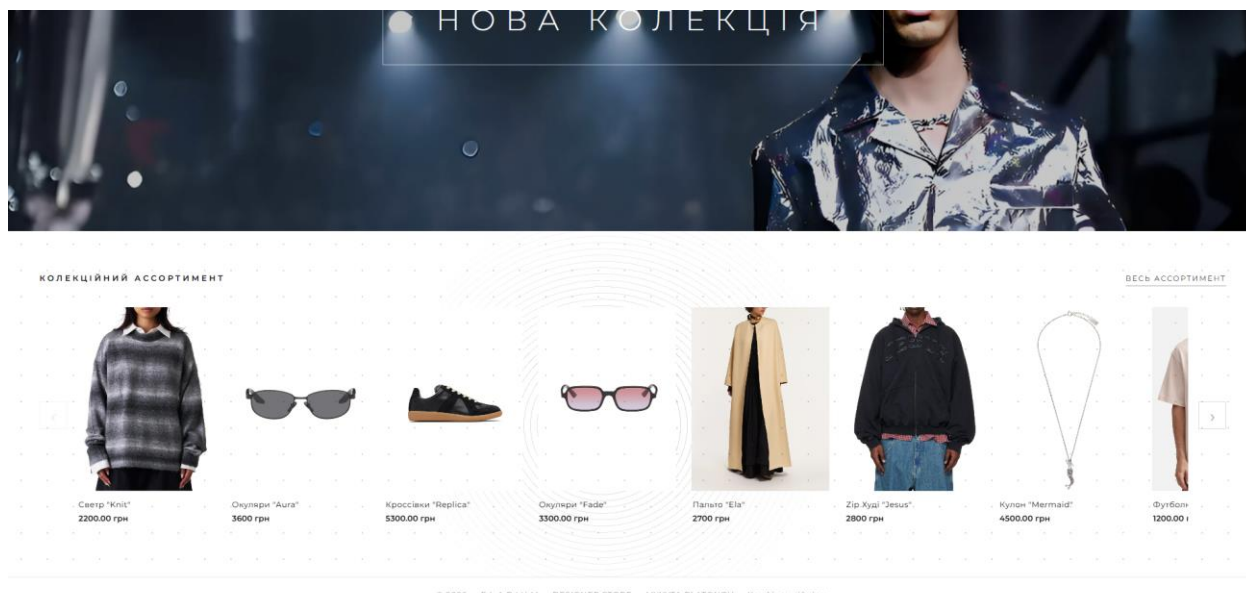


Рис. 3.3 – Секція нової колекції та горизонтальний роллер товарів

Кожна картка у роллері містить квадратне фото товару, скорочену назву та ціну і є посиланням на детальну сторінку. При наведенні курсора картка реагує анімацією підйому `translateY`, що підкреслює інтерактивність елемента.

Розділ «Дефіле» є унікальним елементом магазину, призначеним для демонстрації відеоматеріалів показів колекцій – фешн-шоу та лукбуків. Сторінка доступна за маршрутом `/Defile/Index` та обслуговується контролером `DefileController`, що зчитує перелік записів з бази даних: URL відеофайлу або посилання на зовнішній відеохостинг, назву та опис колекції (рис. 3.4).

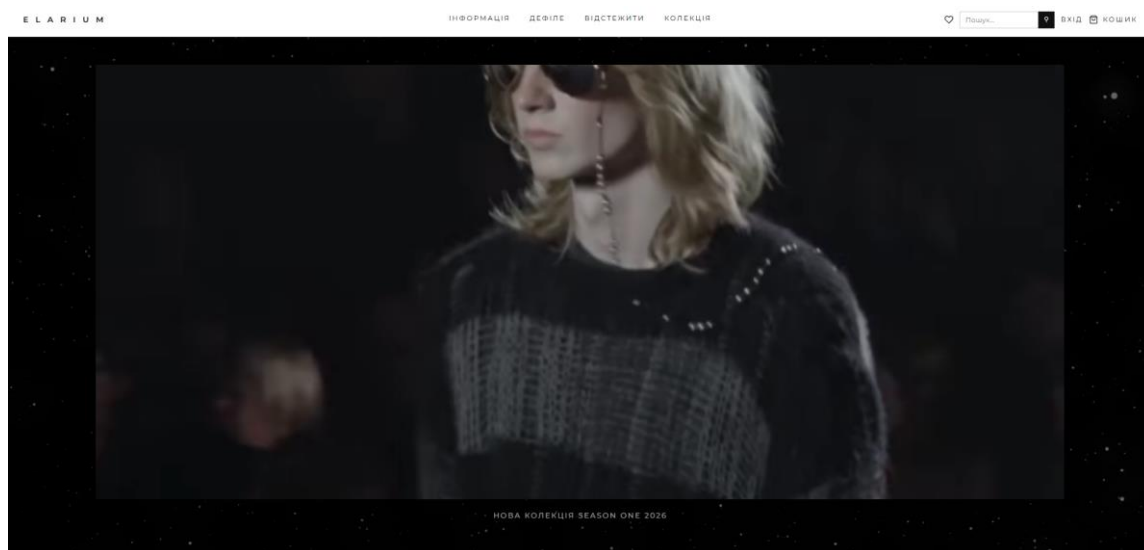


Рис. 3.4 – Розділ «Дефіле» із відеогалереєю колекцій

Сторінка організована у вигляді вертикального переліку колекцій. Кожна колекція представлена блоком із назвою, вбудованим відеоплеєром з атрибутами controls та poster, текстовим описом під плеєром та посиланням «Переглянути товари колекції». Відтворення не запускається автоматично, щоб уникнути одночасного навантаження на мережу від кількох відео. Адміністратор керує матеріалами розділу через адміністративну панель.

Елемент «Інформація» у хедері реалізований як Bootstrap Dropdown, що розкривається при наведенні на десктопі або натисканні на мобільному пристрої. Він забезпечує доступ до чотирьох статичних інформаційних сторінок без завантаження додаткових ресурсів (рис. 3.5).

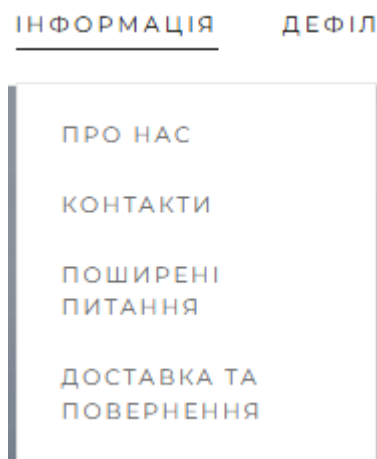


Рис. 3.5 – Випадаюче меню «Інформація»

Підпункти меню охоплюють: «Про нас» – концепція та історія ELARIUM; «Доставка та повернення» – доступні перевізники та строки доставки; «Поширенні питання» – основні питання які виникають у покупців; «Контакти» – контактні дані та форма зворотного зв'язку.

Сторінка входу доступна за маршрутом /Identity/Account/Login і реалізована засобами Razor Pages ASP.NET Core Identity. Вона відображається при явному переході користувача за посиланням «Увійти» у хедері або при автоматичному перенаправленні, коли неавторизований користувач намагається отримати доступ до захищеного ресурсу (рис. 3.6).

ВХІД

Електронна пошта

Пароль

☐ Запам'ятати мене

УВІЙТИ

[Забули пароль?](#)

Немає акаунта? [Зареєструватися](#)

_____ або _____

Увійти через Google

Рис. 3.6 – Сторінка входу /Identity/Account/Login

Сторінка містить форму з полями «Електронна пошта» та «Пароль», чекбокс «Запам'ятати мене», що регулює тривалість cookie між сесіями, та кнопку «Увійти». Додатково розміщено посилання «Забули пароль?» для переходу на сторінку відновлення доступу, а також кнопку «Увійти через Google» та посилання «Зареєструватися» для нових відвідувачів. Валідація форми здійснюється на двох рівнях: клієнтська перевірка формату email засобами JavaScript та серверна перевірка через ModelState.

Кнопка «Увійти через Google» присутня на сторінках входу та реєстрації. При натисканні браузер перенаправляється на захищений домен Google, де користувач обирає або підтверджує обліковий запис. Після успішної автентифікації Google повертає браузер на `redirect_uri` застосунку з кодом авторизації (рис. 3.7).

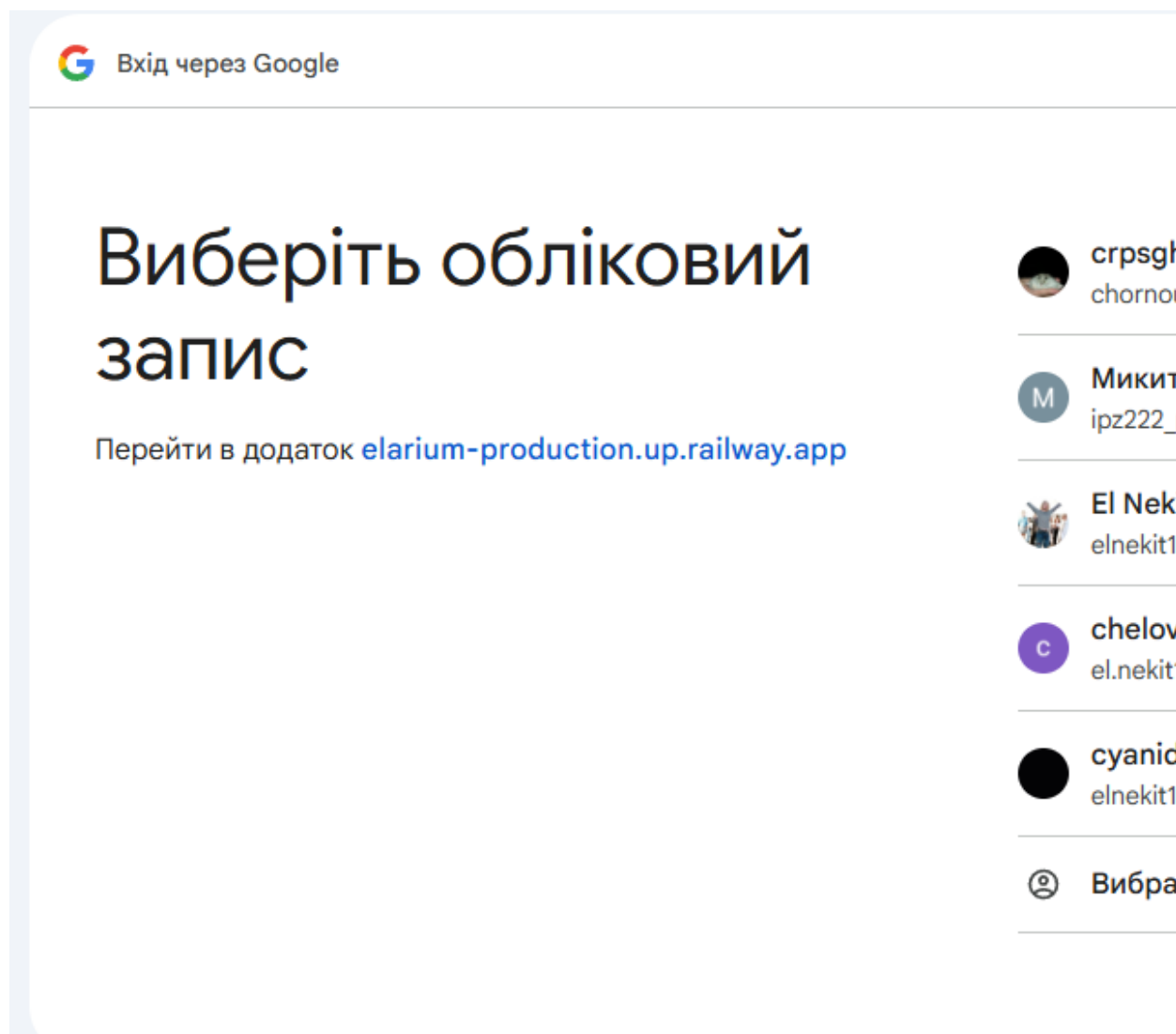


Рис. 3.7 – Вікно вибору облікового запису Google

Після вибору облікового запису система перевіряє, чи пов'язаний цей Google-акаунт із існуючим користувачем. Якщо ні – відображається проміжна сторінка з полем Email та кнопкою «Зареєструватись», де користувач підтверджує або змінює адресу електронної пошти для прив'язки акаунту (рис. 3.8). Після підтвердження обліковий запис створюється автоматично.

Рис. 3.8 – Реєстрація нового облікового запису Google

Сторінка реєстрації доступна за маршрутом `/Identity/Account/Register`. Візуально форма оформлена у вигляді центрованої картки з рамкою, що узгоджується зі стилем сторінки входу. Для кожного поля передбачено перевірку на стороні клієнта засобами jQuery Validation та на стороні сервера через механізм ModelState. Адреса електронної пошти має відповідати формату email, пароль – містити від шести до ста символів, а значення в полі підтвердження повинно збігатися з паролем.

Після натискання кнопки «Створити акаунт» система створює запис у таблицях ASP.NET Core Identity з хешованим паролем і автоматично виконує вхід користувача, оскільки підтвердження email не є обов'язковим (`RequireConfirmedAccount = false`). За наявності налаштованого SMTP-сервера може бути надіслано лист із посиланням для підтвердження пошти, проте доступ до розділу «Мої замовлення» відкривається одразу після успішної реєстрації. Параметр `returnUrl` дозволяє повернути користувача на ту сторінку

каталогу або кошика, з якої він перейшов до форми. Альтернативний спосіб створення облікового запису – вхід через Google на сторінці /Identity/Account/Login; у цьому разі окремої реєстрації не потрібно, а обліковий запис створюється під час першого успішного зовнішнього входу. (рис. 3.9).

The image shows a registration form titled "РЕЄСТРАЦІЯ" (Registration). It contains three input fields: "Електронна пошта" (Email), "Пароль" (Password), and "Підтвердіть пароль" (Confirm password). Below these fields is a dark button labeled "СТВОРИТИ АКАУНТ" (Create Account). At the bottom, there is a link that says "Вже є акаунт? Увійти" (Already have an account? Login).

Рис. 3.9 – Сторінка реєстрації /Identity/Account/Register

Після успішного відправлення форми система створює обліковий запис і виконує автоматичний вхід. Пароль зберігається у вигляді хешу та не може бути відновлений – лише скинутий через посилання у листі.

Функція «Обране» дозволяє зберігати товари для подальшого перегляду. Сторінка списку доступна за маршрутом /Wishlist/Index і відображається з лічильником збережених позицій у назві (наприклад, «ОБРАНЕ – 4

ТОВАРИ»), а також посиланням «← До колекції» у верхньому правому куті (рис. 3.10).

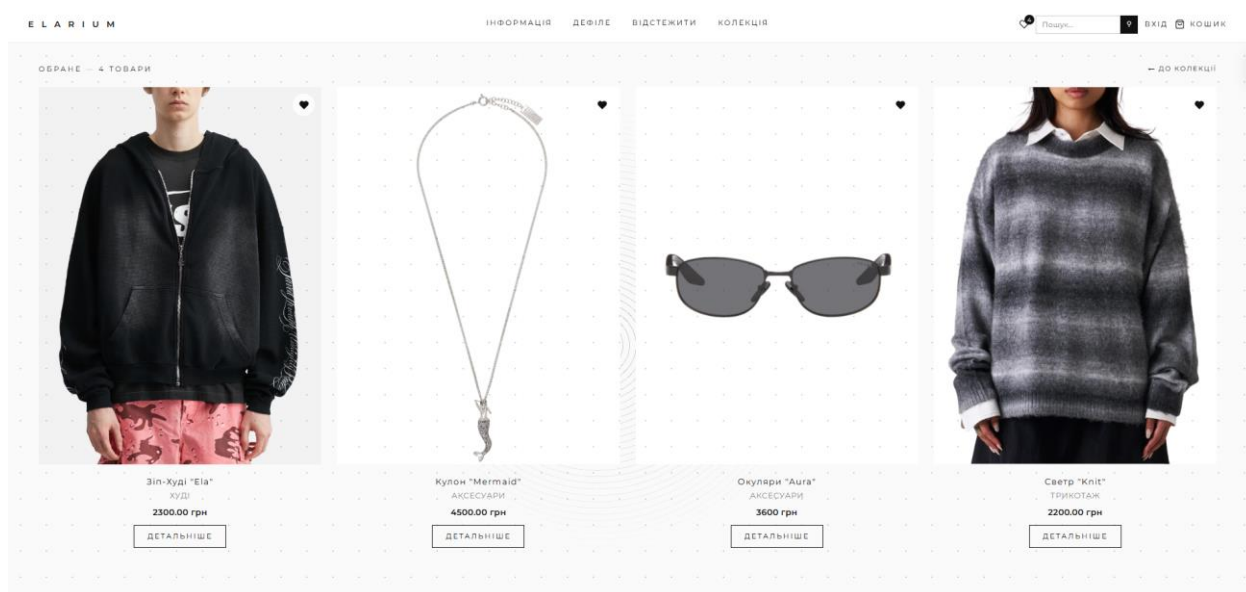


Рис. 3.10 – Сторінка «Список бажаного»

Кнопка додавання у вигляді іконки «серце» присутня на кожній картці каталогу та на детальній сторінці товару. Товари відображаються у вигляді сітки карток аналогічно до каталогу – фото, назва, категорія, ціна та кнопка «Детальніше». Іконка серця на кожній картці залишається активною (заповненою) і дозволяє видалити товар зі списку. У відповідь повертається кількість позицій у списку, яка одразу відображається у лічильнику біля іконки обраного в хедері. Користувач отримує коротке спливаюче повідомлення з назвою товару та посиланням «Переглянути», що веде на сторінку /Wishlist/Index. Стан Wishlist зберігається у серверній сесії і доступний як для гостей, так і для зареєстрованих користувачів. Додавання та видалення товару зі списку бажаного відбувається без перезавантаження сторінки.

Пошуковий рядок розміщено безпосередньо у хедері. Введений текст передається параметром GET, результати відображаються на сторінці каталогу /Home/Index з урахуванням активного пошукового запиту. Пошук виконується за назвою товару і може поєднуватись із фільтрами (рис. 3.11).

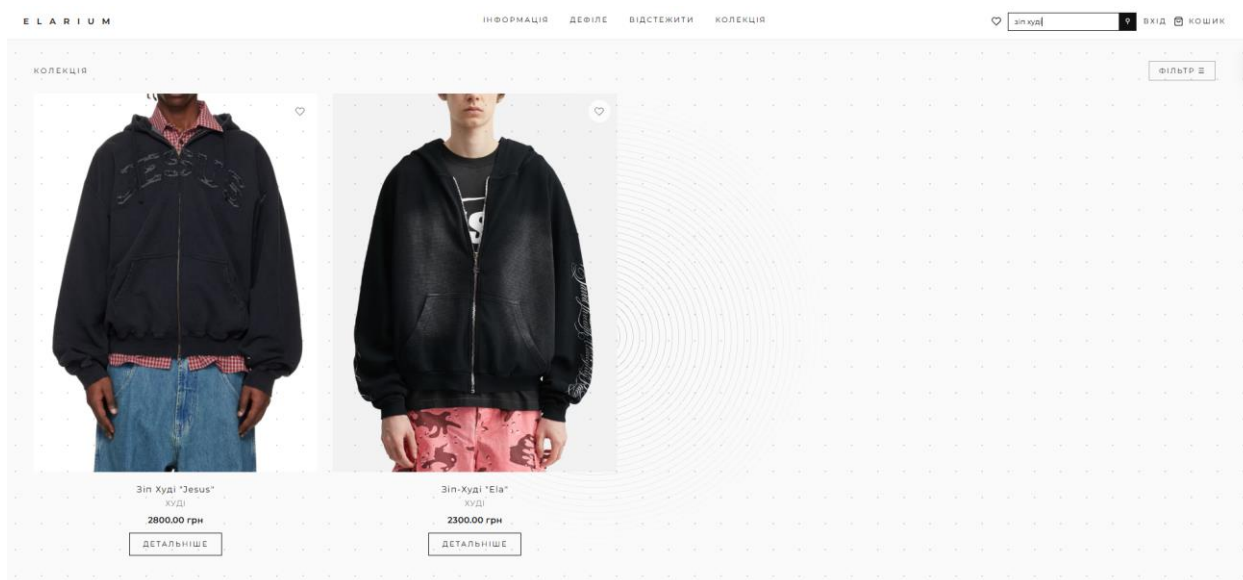


Рис. 3.11 – Результати пошукового запиту в каталозі

Якщо за запитом нічого не знайдено, сторінка відображає анімований напис «Товарів не знайдено» та кнопку «Скинути фільтри» (рис.3.12).



Рис. 3.12 – Результати невдалого пошуку товарів

Сторінка каталогу доступна за маршрутом /Home/Index та служить основним розділом для перегляду основного асортименту. Вгорі розміщено мітку «Колекція» та кнопку «Фільтр ≡» у правому верхньому куті сторінки. Товари відображаються у вигляді сітки карток з інформацією нижче (рис. 3.13).

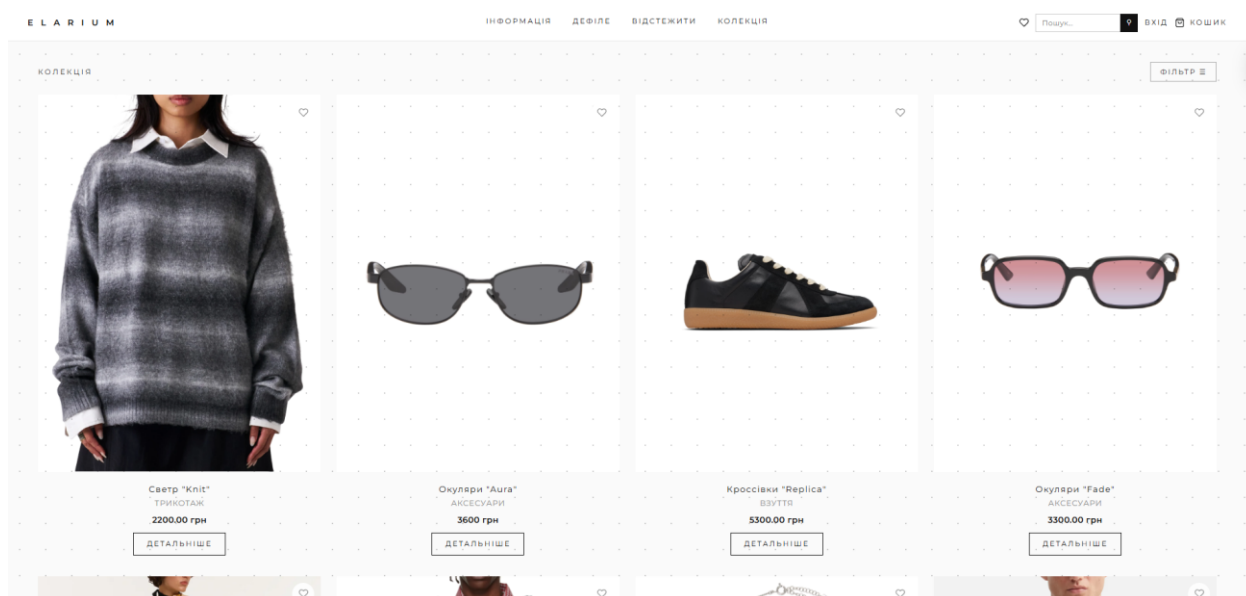


Рис. 3.13 – Сторінка каталогу товарів (/Home/Index)

Кожна картка товару містить фотографію з першого запису галереї, назву артикула, ціну та іконку-серце для додавання до списку бажаного. Заголовок блоку динамічно змінюється – за замовчуванням показується напис «Колекція», а при виборі категорії з навігації або фільтра – назва обраної категорії. Перехід на детальну сторінку товару відбувається при натисканні на будь-яку частину картки. Над сіткою товарів розміщено рядок пошуку та кнопку «Фільтри», що відкриває з правого боку панель уточнення результатів а також підсвічується, якщо застосовано хоча б один параметр уточнення. Також доступне сортування – за новизною або за ціною (від меншої до більшої та навпаки і т.д.). Якщо жодного товару не знайдено, відображається повідомлення з пропозицією скинути фільтри.

Кнопка «Фільтри» розміщена в правому верхньому куті і відкриває вспливаючу бокову панель або модальне вікно фільтрів, не перезавантажуючи сторінку. Бокова панель фільтрів виїжджає з правого краю екрана поверх затемненого напівпрозорого фону. Закрити її можна кнопкою «×», кліком по фону або клавішею Escape. Поточний пошуковий запит зберігається у прихованому полі форми, тому після застосування фільтрів текст пошуку не губиться (рис. 3.14).

«Від» та «До» для задання мінімальної та максимальної ціни. Наявність розміру – перелік стандартних розмірів, при виборі одного з яких відображаються лише ті товари, для яких цей розмір є у наявності зі ненульовим залишком. Група «Сортування» містить перемикачі: «Спочатку нові», «Спочатку старі», «Від дешевших», «Від дорожчих». Також доступні окремі розміри для взуття. Кнопка «Застосувати» передає обрані параметри у рядок URL як GET-параметри та оновлює сітку товарів. Кнопка «Скинути» очищає усі фільтри та повертає повний список асортименту. Параметри передаються у рядку URL, тому результати можна зберегти або поширити посиланням.

Детальна сторінка товару відкривається за маршрутом /Home/Details/{id} при натисканні на картку у каталозі або роллері. Вона надає вичерпну інформацію про конкретний виріб та забезпечує вибір розміру і додавання до кошика (рис. 3.15).

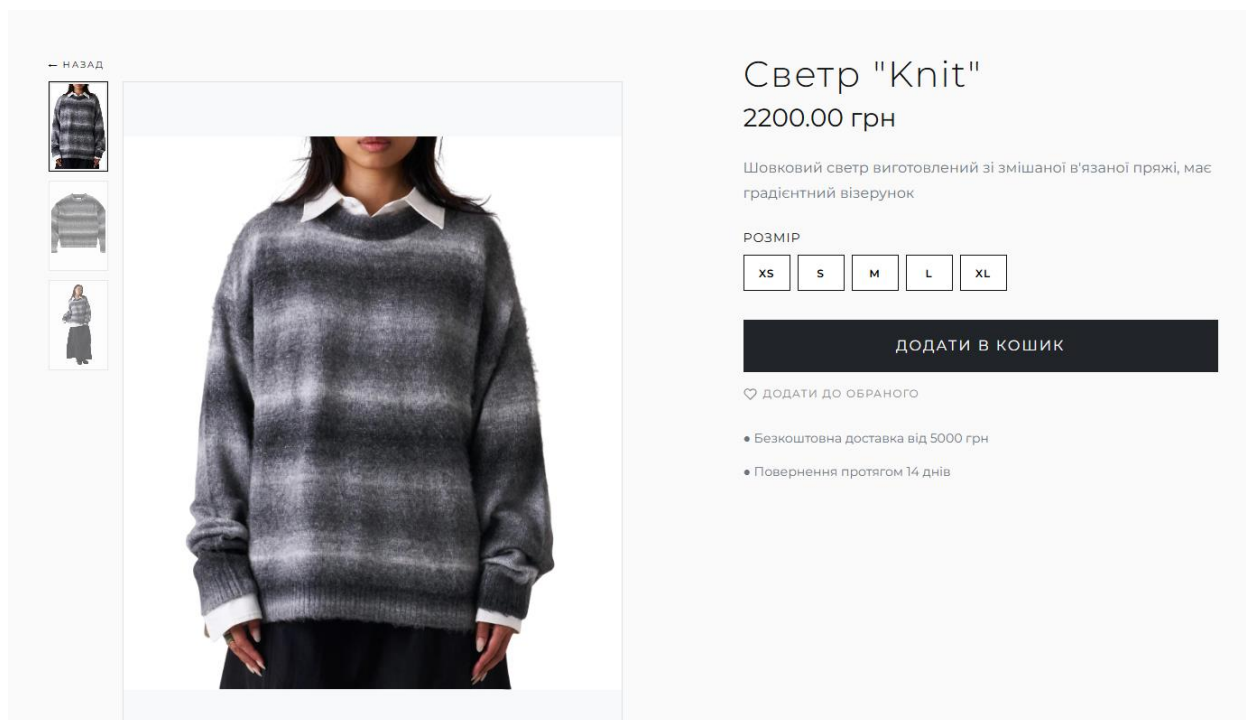


Рис. 3.15 – Детальна сторінка товару (/Home/Details)

Ліва частина сторінки відведена під галерею зображень товару. Головне фото відображається у великому форматі, нижче розміщено мініатюри решти знімків з галереї. Клік по мініатюрі замінює головне фото без

перезавантаження сторінки. При натисканні на головне фото відкривається модальне вікно з масштабованим зображенням, що дозволяє детально розглянути фактуру тканини або деталі пошиття.

Права частина сторінки містить назву товару, ціну та детальний текстовий опис. Нижче розміщено блок вибору розміру: доступні розміри відображаються як кнопки-перемикачі, а розміри з нульовим залишком позначаються закресленням або сірим кольором і недоступні для вибору (рис. 3.16). Вибір конкретного розміру є обов'язковим – без нього кнопка «Додати до кошика» залишається неактивною. Поруч із блоком розмірів розміщено іконку Wishlist. Після вибору розміру та натискання «Додати до кошика» товар додається до сесійного кошика, а користувач отримує підтвердження у вигляді спливаючого повідомлення (рис. 3.17).

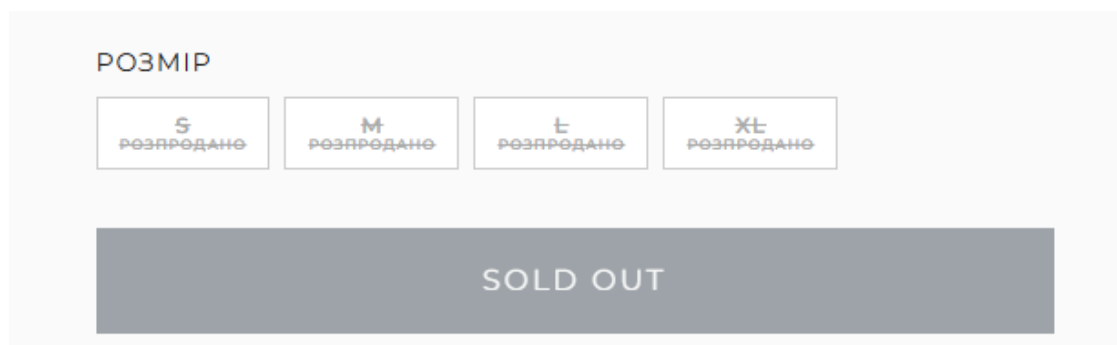


Рис. 3.16 – Розміри з нульовим залишком

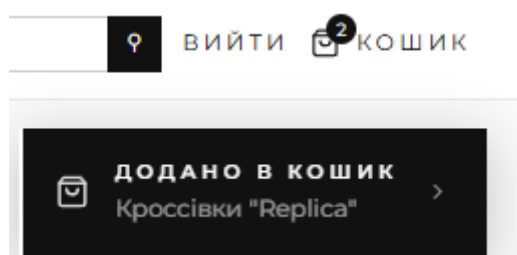


Рис. 3.17 – Розміри з нульовим залишком

Сторінка кошика доступна за маршрутом /Cart/Index і відображає всі товари, які користувач додав під час поточної сесії. Перехід до кошика можливий через іконку у хедері або через кнопку після додавання товару (рис. 3.18).

Ліва частина сторінки – форма «Дані покупця» з полями: ПІБ, Email, Телефон, Місто, Відділення Нової Пошти / Адреса. Права частина – блок «Ваше замовлення» з переліком позицій кошика (мініатюра, назва, розмір, кількість, ціна) та підсумковою сумою.

Під формою розміщено блок «Спосіб оплати» з двома варіантами на вибір: «КАРТКА (STRIPE) – Visa / Mastercard» та «WAYFORPAY – Українська платіжна система». Активний варіант виділяється рамкою. Кнопка «ПЕРЕЙТИ ДО ОПЛАТИ» запускає обраний платіжний шлюз. На прикладі Stripe – браузер перенаправляється на захищену сторінку Stripe Checkout, де покупець вводить реквізити картки. Після успішної транзакції Stripe надсилає підтвердження на сервер, замовлення переводиться у статус «Оплачено» і користувач перенаправляється на сторінку успішного оформлення. Посилання «← Повернутись до кошика» дозволяє скасувати оформлення.

Після вибору методу «КАРТКА (STRIPE)» та натискання кнопки «ПЕРЕЙТИ ДО ОПЛАТИ» браузер перенаправляється на захищену платіжну сторінку Stripe Checkout – зовнішній сервіс компанії Stripe, Inc. Реквізити картки вводяться безпосередньо на сторінці Stripe, а не на сервері магазину, що унеможливорює перехоплення платіжних даних (рис. 3.17).

Перед переходом до оплати форма проходить серверну валідацію, а запит захищено від CSRF-токеном. Контролер спочатку створює в базі даних запис замовлення зі статусом «Очікує оплати» і лише потім ініціює платіж: для Stripe – через редирект на Checkout Session, для WayForPay – через проміжну сторінку з автоматичною відправкою підписаної форми на платіжний шлюз. У разі Stripe після успішної оплати браузер повертається на /Order/Success, де сервер за ідентифікатором сесії перевіряє статус платежу через Stripe API, переводить замовлення в «Оплачено», зменшує залишки та очищає сесійний кошик. Для авторизованих користувачів поле Email заповнюється автоматично з облікового запису і не редагується, щоб замовлення коректно потрапляло в розділ «Мої замовлення».

Светр "Knit" / L

UAH 2,200.00

OR

Contact information

Email elnekit15@gmail.com

Payment method

Card

Card information

4242 4242 4242 4242

VISA

12 / 34

333

123

Cardholder name

PLATONOV MYKYTA

Billing address

Ukraine

Адреса

Address line 2

Київ

Київ — Kyiv city

11111

☐ Save my information for faster checkout

Pay securely on this site and everywhere Link is accepted.

Pay

Powered by stripe | Terms Privacy

Рис. 3.20 – Сторінка оформлення замовлення (/Order/Checkout)

Сторінка Stripe Checkout відображає назву магазину «ELARIUM», суму до сплати у відповідній валюті та форму введення даних картки з полями: номер картки, термін дії та CVC-код. Для тестування інтеграції Stripe надає набір тестових реквізитів: номер картки 4242 4242 4242 4242, будь-який майбутній термін дії (наприклад, 12/34) та довільний тризначний CVC. Після введення тестових даних та підтвердження Stripe імітує успішну транзакцію без реального списання коштів.

Після підтвердження оплати Stripe надсилає webhook-подію на серверний endpoint магазину. Сервер верифікує підпис події та оновлює статус замовлення на «Оплачено», після чого браузер перенаправляється на сторінку /Order/Success.

Після підтвердження оплати одним із шлюзів користувач перенаправляється на сторінку /Order/Success, що слугує підтвердженням завершення транзакції (рис. 3.21).

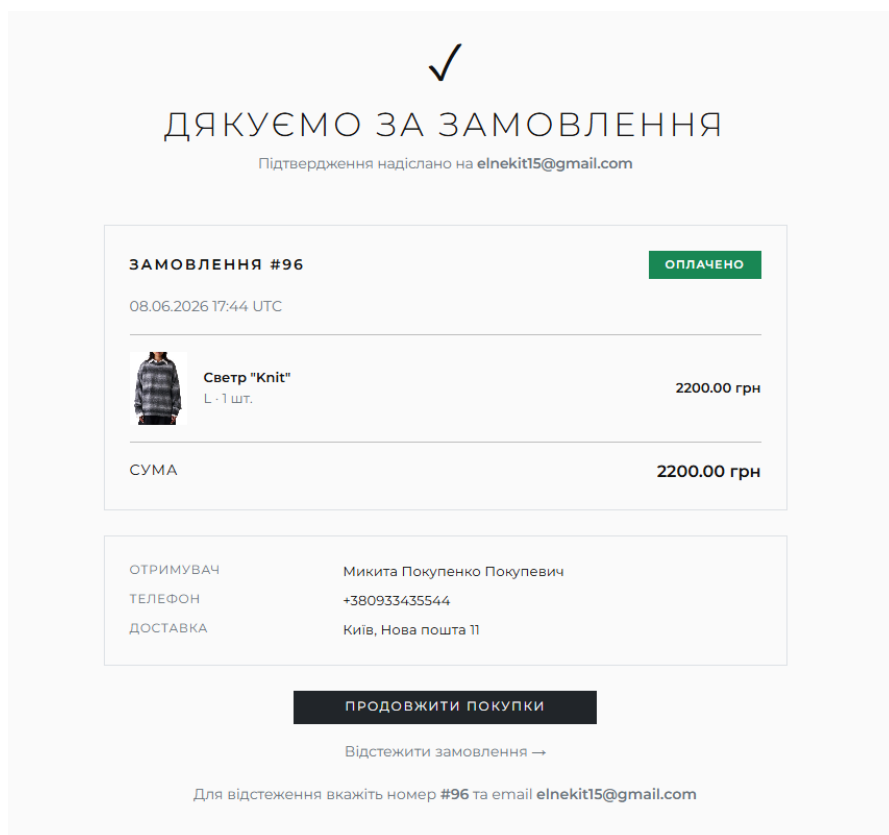


Рис. 3.21 – Сторінка підтвердження замовлення (/Order/Success)

Сторінка відображає повідомлення про успішне оформлення, унікальний номер замовлення та електронну адресу, на яку надіслано підтвердження. Для зареєстрованих користувачів замовлення автоматично відображається в розділі «Мої замовлення». Для гостей відображається посилання у вигляді кнопки «Відстежити замовлення» для відстеження замовлення за номером та email на окремій сторінці. Після відображення сторінки успіху кошик автоматично очищується, а залишки обраних розмірів у БД зменшено на відповідну кількість позицій. Якщо залишок за певним розміром стає нульовим, він позначається як недоступний у картці товару.

Розділ «Мої замовлення» доступний за маршрутом /Order/MyOrders для зареєстрованих та автентифікованих користувачів. Перехід можливий через меню профілю (рис. 3.22).

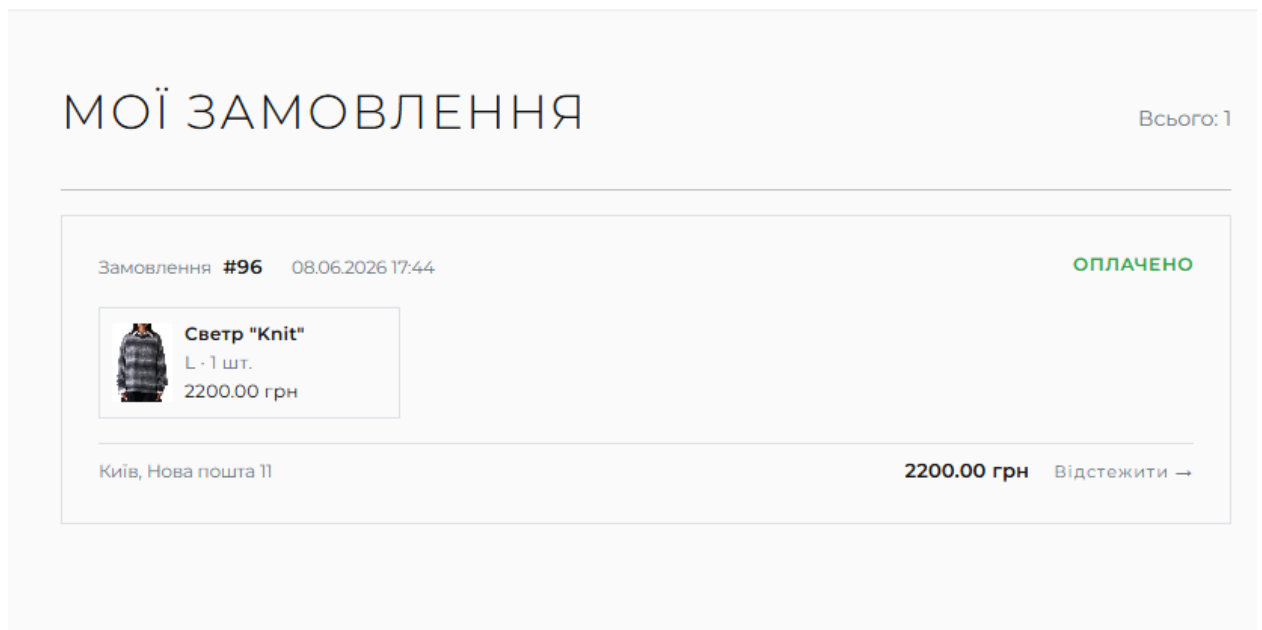


Рис. 3.22 – Сторінка «Мої замовлення» (/Order/MyOrders)

Сторінка відображає хронологічний перелік усіх замовлень поточного користувача. Сторінка містить: номер замовлення, дату оформлення, загальну суму, поточний статус (наприклад, «Оплачено», «В обробці», «Відправлено», «Доставлено») а також кнопку «Відстежити» для перегляду поточного статусу замовлення. Статус замовлення оновлюється адміністратором через адміністративну панель і відображається в реальному часі при кожному завантаженні сторінки. Замовлення зберігаються без обмежень за строком, що дозволяє користувачеві звертатися до повної історії покупок у будь-який момент.

Сторінка відстеження замовлення доступна за маршрутом /Order/Track і призначена передусім для гостей – неавторизованих покупців, які не мають особистого кабінету. Перехід на неї здійснюється через кнопку «Відстежити замовлення» на сторінці успішного оформлення /Order/Success, а також з розділу «Мої замовлення» для зареєстрованих користувачів (рис. 3.23).

Зареєстрованим клієнтам зазвичай достатньо розділу «Мої замовлення», де відображається повна історія покупок з актуальними статусами без повторного введення даних. Сторінка відстеження залишається універсальним інструментом для гостей, які оформили покупку без реєстрації, а також для

перевірки окремого замовлення за номером і email, якщо потрібно уточнити статус доставки після оформлення.

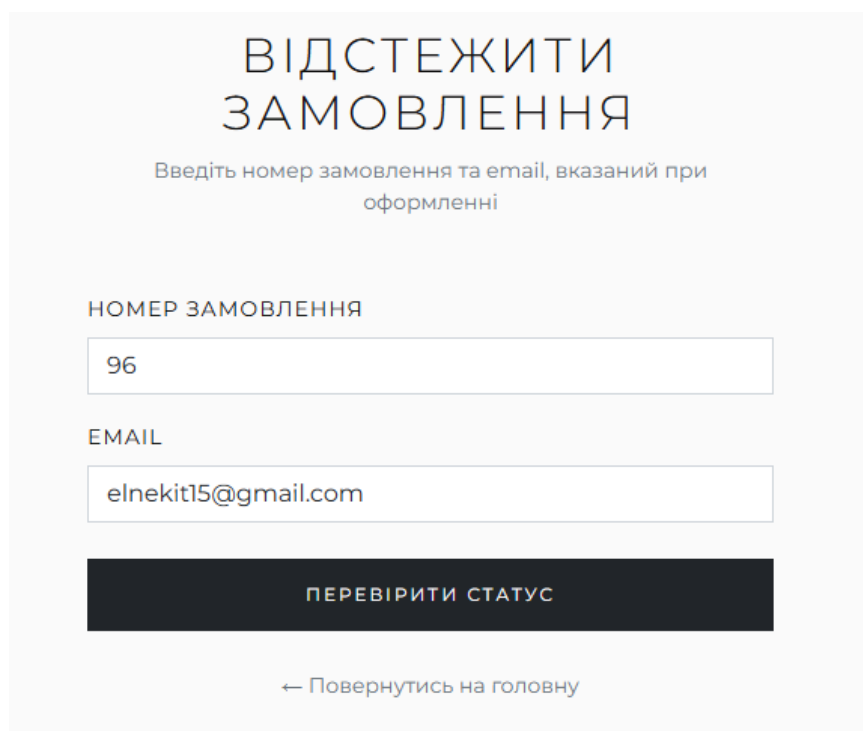


Рис. 3.23 – Сторінка відстеження замовлення (/Order/Track)

Сторінка містить форму з двома полями для введення: унікальний номер замовлення (видається системою на сторінці /Order/Success) та електронна адреса, вказана при оформленні. Обидва поля є обов'язковими та проходять серверну валідацію через ModelState. Після натискання кнопки «Перевірити» контролер OrderController.Track() [POST] виконує пошук замовлення в базі даних за збігом обох ідентифікаторів одночасно – номера замовлення та email. Така двофакторна перевірка унеможливорює несанкціонований доступ до чужого замовлення навіть при знанні його номера.

На сторінці кнопка підписана «Перевірити статус». Якщо пара «номер + email» не знайдена, система показує повідомлення про помилку без розкриття деталей замовлення. При успішній перевірці відкривається сторінка результату з горизонтальним таймлайном етапів обробки, переліком товарів, сумою та блоком даних доставки; для скасованих замовлень замість таймлайну відображається відповідне попередження (рис. 3.24).

ЗАМОВЛЕННЯ #96

ОПЛАЧЕНО

ОЧІКУЄ

ОПЛАЧЕНО

ОБРОБКА

ВІДПРАВЛЕНО

ДОСТАВЛЕНО

ТОВАРИ



Светр "Knit"

Розмір: L · 1 шт.

2200.00 грн

РАЗОМ

2200.00 грн

ДАНІ ДОСТАВКИ

ОТРИМУВАЧ

Микита Покупенко Покупевич

ТЕЛЕФОН

+380933435544

ДОСТАВКА

Київ, Нова пошта 11

ДАТА

08.06.2026 17:44

← Перевірити інше замовлення

До магазину

Рис. 3.24 – Результат перевірки статусу замовлення

Якщо замовлення знайдено та email збігається, сторінка відображає: номер замовлення, дані, дату та час оформлення, поточний статус із відповідним кольоровим маркером (наприклад, «В обробці», «Підтверджено», «Відправлено», «Доставлено»), загальну суму та перелік позицій замовлення з назвою товару, обраним розміром, кількістю та ціною. Якщо за введеними даними замовлення не знайдено, відображається повідомлення про помилку з пропозицією перевірити правильність введених даних або зв'язатися з підтримкою.

Адміністративна панель є захищеною частиною застосунку, доступною виключно для користувачів з роллю «Admin». Спроба звернутися до будь-якого маршруту адмін-панелі без відповідної ролі автоматично перенаправляє на сторінку входу. Навігація між розділами адміністративної панелі здійснюється через окреме адміністративне меню, що замінює стандартний

хедер після автентифікації адміністратора. Панель охоплює чотири основних розділи: управління товарами (/Products), управління замовленнями (/Orders), аналітичну інформаційну панель (/Admin/Dashboard) та управління дефіле (/Defile/Manage).

Сторінка переліку товарів доступна за маршрутом /Products/Index і є сторінкою розділу управління каталогом. Вона відображає повний список усіх товарів у базі даних незалежно від прапора IsAvailable, що дозволяє адміністратору бачити та редагувати як активні, так і приховані позиції (рис. 3.25).

ФОТО	НАЗВА	ЦІНА	ОПИС	ДІЇ
	Зіп-Худи "Ela"	2300.00 грн	Дизайнерська зіп худи преміум якості	Редагувати
	Штани "Ela"	1800.00 грн	Дизайнерські штани з цікавими деталями та авторським кроєм	Редагувати
	Лонгсліви "Ela"	1600.00 грн	Лонгсліви преміум якості з цікавим кроєм	Редагувати
	Футболка "Ela"	1200.00 грн	Футболка авторського крою	Редагувати
	Куртка "Ela Fly"	2300.00 грн	Куртка з авторським принтом та обробкою, у матеріалах преміум...	Редагувати
	Jeans "Ela"	2200.00 грн	Джинси з ефектом Washed та цікавим кольором	Редагувати
	Худи "Spray"	1800.00 грн	Худи з авторським кроєм преміум якості	Редагувати
	Футболка "Acne"	1200.00 грн	Мінімалістичний дизайн з якісним кроєм	Редагувати
	Купон "Mermaid"	4500.00 грн	Лінійований купон з високоякісних матеріалів	Редагувати

Рис. 3.25 – Сторінка переліку товарів (/Products/Index)

Перелік організований у вигляді таблиці з колонками: перше зображення товару у вигляді мініатюри, назва товару, ціна, опис та кнопка редагувати. Для кожного товару доступні дії: «Редагувати» – перехід на сторінку /Products/Edit/{id}; «Видалити» – видалення товару разом зі всіма пов'язаними зображеннями та записами залишків. У верхній частині сторінки розміщено кнопку «Додати товар», що веде на /Products/Create.

Сторінка створення товару відкривається за маршрутом /Products/Create із заголовком «Новий товар». Ліворуч розміщено блок «Головне фото» із полем завантаження головного зображення та попереднім переглядом. Праворуч – поля атрибутів товару: «Назва», «Категорія» (випадаючий список),

«Стать» (випадаючий список: Не вказано / Жіноче / Чоловіче / Унісекс), «Ціна (ГРН)» та чекбокс «В наявності» (рис. 3.26).

NOVYI TOVAR

ГОЛОВНЕ ФОТО

Не завантажено

Вибрати файл Файл не вибрано

НАЗВА

КАТЕГОРІЯ

Оберіть категорію...

СТАТЬ

Не вказано

ЦІНА (ГРН)

НАЯВНІСТЬ

☒ В наявності

РОЗМІРИ ТА ЗАЛИШОК

+ Додати розмір

РОЗМІР	К-СТЬ В НАЯВНОСТІ
M	0

0 = розпродано для цього розміру - якщо всі = 0, товар стає недоступним автоматично

ОПИС ТОВАРУ

ДОДАТКОВІ ФОТО (МОЖНА ВИБРАТИ ДЕКІЛЬКА)

Вибрати файли Файл не вибрано

ЗБЕРЕГТИ ТОВАР Назад до списку

Рис. 3.26 – Форма створення нового товару (/Products/Create)

Блок «Розміри та залишок» дозволяє динамічно додавати рядки розмірів кнопкою «+ Додати розмір». Кожен рядок містить випадаючий список з назвою розміру та числове поле кількості. Підказка під таблицею інформує: «0 = розпродано для цього розміру · якщо всі = 0, товар стає недоступним автоматично». Нижче розміщено поле «Опис товару» (textarea) та блок «Додаткові фото (Можна вибрати декілька)» з мультизавантаженням зображень галереї.

Сторінка редагування відкривається за маршрутом /Products/Edit/{id} із заголовком «Редагування товару». Структура форми ідентична формі

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						77

створення, але всі поля попередньо заповнені поточними значеннями з бази даних. Ліворуч відображається блок «Поточне фото» з наявним головним зображенням та кнопкою «Змінити фото» під ним (рис. 3.27).

РЕДАГУВАННЯ ТОВАРУ

ПОТОЧНЕ ФОТО



ЗМІНИТИ ФОТО

Вибрати файл

Файл не вибрано

НАЗВА

КАТЕГОРІЯ

Худі

СТАТЬ

Унісекс

ЦІНА (ГРН)

НАЯВНІСТЬ

☒ В наявності

РОЗМІРИ ТА ЗАЛИШОК

+ Додати розмір

РОЗМІР	К-СТЬ В НАЯВНОСТІ	
L	4	x
M	0	x
S	5	x
XL	5	x

0 = розпродано для цього розміру - якщо всі = 0, товар стає недоступним автоматично

ОПИС ТОВАРУ

Додаткові фото (Можна вибрати декілька)

Вибрати файли

Файл не вибрано





ЗБЕРЕГТИ ЗМІНИ

Назад до списку

Рис. 3.27 – Сторінка редагування товару (/Products/Edit)

Блок «Розміри та залишок» відображає наявні рядки з поточними значеннями кількостей. Кожен рядок можна видалити кнопкою «x» або додати новий рядок кнопкою «+ Додати розмір». Блок «Додаткові фото» показує поточну галерею – кожна мініатюра містить кнопку «x» для видалення конкретного знімку – при натисканні запис видаляється з БД разом із фізичним файлом (рис. 3.28). Кнопка «Зберегти зміни» зберігає всі зміни, а посилання «Назад до списку» повертає на /Products/Index без збереження. Нижня секція

«Небезпечна зона» містить кнопку «Видалити товар» для остаточного видалення товару разом із усіма пов'язаними записами.

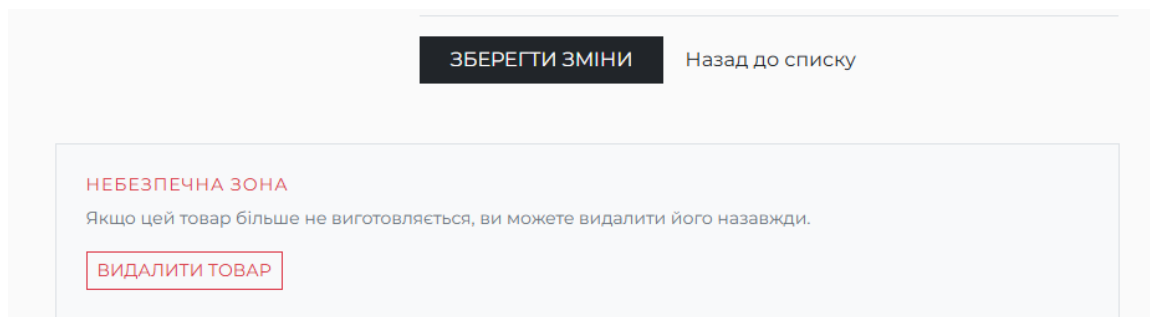


Рис. 3.28 – Блок галереї та небезпечна зона на сторінці редагування

Сторінка переліку замовлень доступна за маршрутом /Orders/Index. Заголовок «Замовлення» розміщено зліва, лічильник загальної кількості – праворуч. Замовлення відображаються таблицею у зворотному хронологічному порядку з колонками: номер (#), покупець (ім'я та email під ним), дата, кількість товарів, сума, статус із кольоровим маркером та кнопки дій «Деталі» і видалення (рис. 3.29).

ЗАМОВЛЕННЯ						Всього: 15
#	ПОКУПЕЦЬ	ДАТА	ТОВАРІВ	СУМА	СТАТУС	
96	Микита Покупенко Покупевич elnekit15@gmail.com	08.06.2026 17:44	1 шт.	2200.00 грн	ОПЛАЧЕНО	Деталі
95	Іванович Микола Грищенко mykola@mail.ua	07.06.2026 19:01	1 шт.	2200.00 грн	ВІДПРАВЛЕНО	Деталі
94	Михайло Іванович Даниленко myhallo@gmail.com	07.06.2026 11:44	1 шт.	1200.00 грн	ОПЛАЧЕНО	Деталі
61	Бондаренко Богдан Михайлович bohdan@gmail.com	04.06.2026 12:01	1 шт.	2800 грн	В ОБРОБЦІ	Деталі
54	Данило Даниленко Іванович danylo@mail.com	03.06.2026 14:01	1 шт.	2800 грн	ВІДПРАВЛЕНО	Деталі
50	Ely Vance gogle@gmail.com	02.06.2026 08:43	1 шт.	2300.00 грн	ВІДПРАВЛЕНО	Деталі
49	Зарембицький Єгор Юрійович indiraptor63983@gmail.com	02.06.2026 08:26	1 шт.	3600 грн	В ОБРОБЦІ	Деталі
12	Микита Іванов Іванович mail@xmpl.com	27.05.2026 12:47	1 шт.	2300.00 грн	ДОСТАВЛЕНО	Деталі
11	Микита Іванов Іванович mail@xmpl.com	27.05.2026 12:47	1 шт.	2300.00 грн	ДОСТАВЛЕНО	Деталі
10	Микита Іванов Іванович mail@xmpl.com	27.05.2026 12:47	1 шт.	2300.00 грн	ВІДПРАВЛЕНО	Деталі

Рис. 3.29 – Сторінка переліку замовлень (/Orders/Index)

Статуси відображаються кольорово: «Оплачено» – зеленим, «Відправлено» – фіолетовим, «В обробці» – синім, «Доставлено» – чорним. Натискання «Деталі» відкриває сторінку /Orders/Details/{id} з повною інформацією про конкретне замовлення.

Сторінка деталей відкривається за маршрутом /Orders/Details/{id} із заголовком «Замовлення #N» та поточним статусом у правому верхньому куті. Кнопка «← Назад» повертає до списку замовлень, можемо бачити деталі нашого замовлення (рис. 3.30).

← Назад

ЗАМОВЛЕННЯ #96

ОПЛАЧЕНО

ТОВАРИ



Светр "Knit"

Розмір: L

Кількість: 1 шт.

Ціна: 2200.00 грн/шт.

2200.00 грн

РАЗОМ

2200.00 грн

ПОКУПЕЦЬ

ІМ'Я

Микита Покупенко Покупевич

EMAIL

elnekit5@gmail.com

ТЕЛЕФОН

+380933435544

МІСТО

Київ

АДРЕСА

Нова пошта 11

ДАТА

08.06.2026 17:44

STRIPE SESSION

cs_test_a1M7L14e5gpTjLi8wxn1DFdv4iSiTvUQI9ahlGBKEcWQesXvz3IVTPYISk

PAYMENT INTENT

pi_3Tg7KeliHijbitJ520KqheDF

ЗМІНИТИ СТАТУС

Оплачено

▼

ОНОВИТИ СТАТУС

Рис. 3.30 – Сторінка деталей замовлення (/Orders/Details)

Ліва частина відображає блок «Товари» – мініатюра, назва, розмір, кількість та ціна кожної позиції, а також загальна сума «РАЗОМ». Права частина містить два блоки. Блок «Покупець» – ім'я, email, телефон, місто, адреса та дата оформлення. Блок технічних даних оплати – Stripe Session та Payment Intent ідентифікатори для зв'язки з платіжним шлюзом. Нижній блок «Змінити статус» містить випадаючий список із доступними статусами та кнопку «Оновити статус». Після збереження система автоматично показує відповідний статус у розділі «Мої замовлення» або «Відстежити».

Інформаційна панель доступна за маршрутом /Admin/Dashboard із заголовком «Аналітика – Дані Продажів». Верхній рядок містить шість блоків-лічильників: «Виручка» (сума оплачених замовлень), «Замовлення» (загальна

кількість із підрахунком оплачених), «Середній чек» (серед оплачених), «Клієнти» (унікальних email), «Товарів» (в каталозі) та «Продано» (одиниць товару) (рис. 3.31).

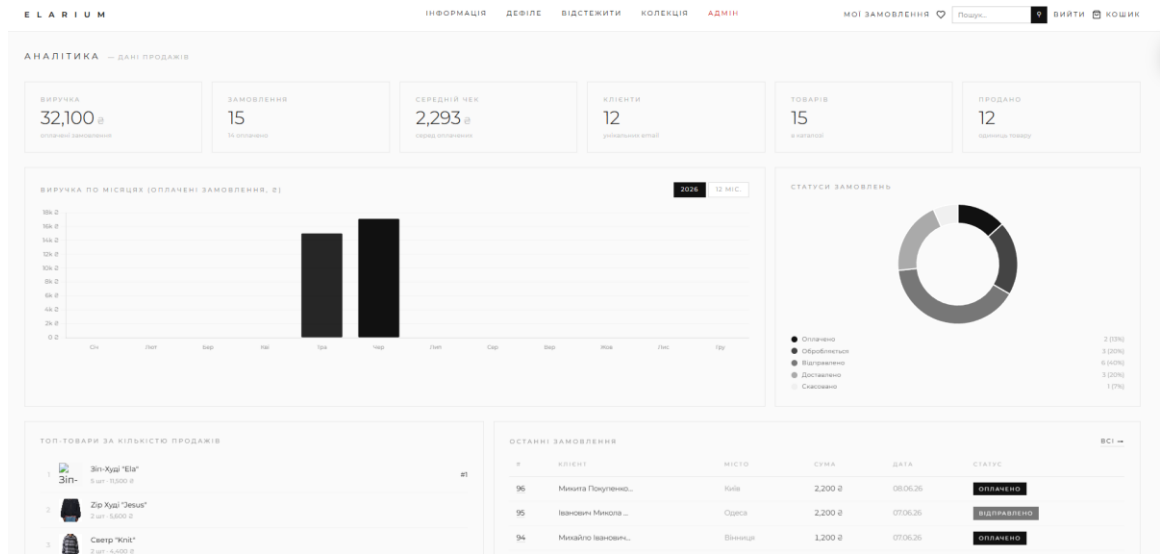


Рис. 3.31 – Аналітична інформаційна панель (/Admin/Dashboard)

Середній блок займає дві колонки. Ліворуч – стовпчаста діаграма «Виручка по місяцях» з перемикачем між поточним роком та останніми 12 місяцями. Праворуч – кругова діаграма «Статуси замовлень» із відсотковою розбивкою за кожним статусом – Оплачено, Обробляється, Відправлено, Доставлено, Очікує, Скасовано. Нижній блок також поділений на дві колонки: «Топ-товари за кількістю продажів» (рейтинг із мініатюрою, назвою, кількістю та виручкою) та «Останні замовлення» (таблиця з номером, клієнтом, містом, сумою, датою і статусом і посиланням «Всі→»).

Діаграми реалізовано засобами Chart.js на агрегованих даних з PostgreSQL; перемикач періоду оновлює графік виручки без перезавантаження сторінки. Над блоком діаграм розміщено ряд KPI-карток: загальна виручка, кількість замовлень, середній чек, унікальні клієнти, товарів у каталозі та продано одиниць.

Сторінка управління дефіле доступна за маршрутом /Defile/Manage із заголовком «Керування Дефіле». Вгорі розміщено блок «Поточне відео» з вбудованим відеоплеєром, що відтворює наявний файл колекції (рис. 3.32).

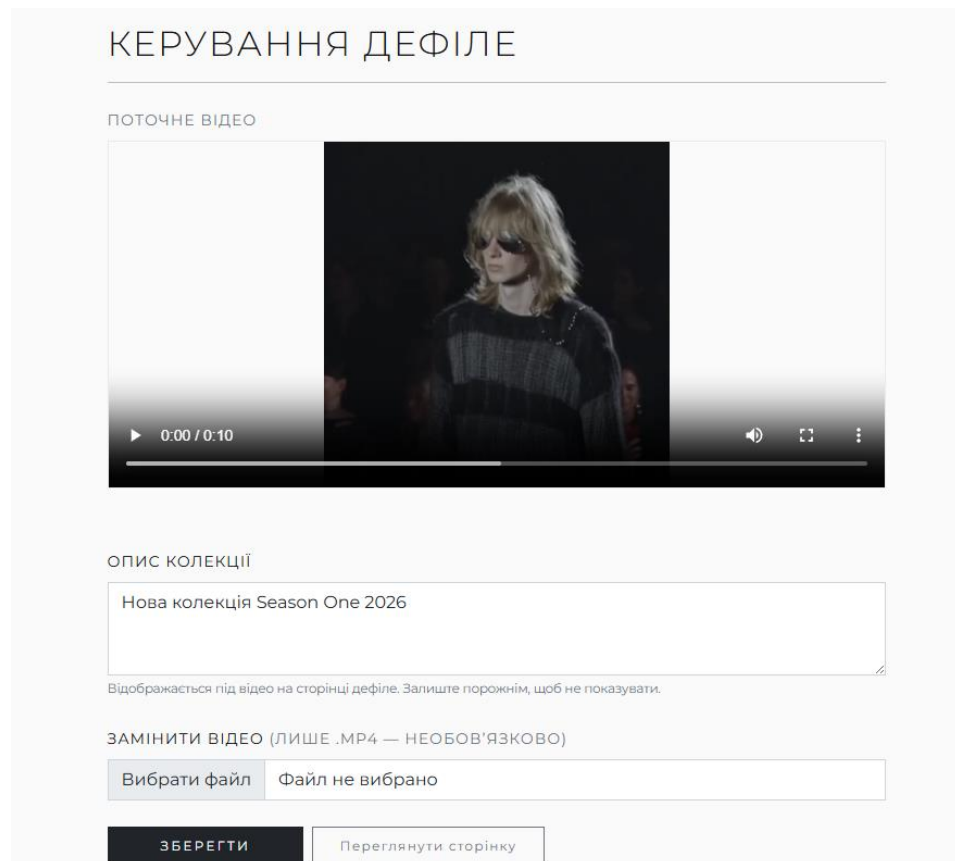


Рис. 3.32 – Сторінка управління розділом «Дефіле» (/Defile/Manage)

Нижче відеоплеєра розміщено поле «Опис колекції» (textarea) з підказкою «Відображається під відео на сторінці дефіле. Залиште порожнім, щоб не показувати.». Під ним – блок «Замінити відео»)» з кнопкою вибору файлу. Кнопка «Зберегти» оновлює запис у базі даних, а посилання «Переглянути сторінку» відкриває публічний розділ /Defile/Index для перевірки результату. Зміни набирають чинності негайно при наступному відвідуванні публічної сторінки.

3.3 Тестування роботи програмного комплексу

Тестування є обов'язковим етапом розробки програмного забезпечення, метою якого є верифікація коректності реалізованого функціоналу та виявлення потенційних вразливостей до введення системи в експлуатацію. Для програмного комплексу «ELARIUM» застосовано функціональне тестування методом «чорної скриньки» – перевірка сценаріїв здійснюється через реальний вебінтерфейс.

Тестування охоплює чотири групи сценаріїв: автентифікація та авторизація, захист адміністративних маршрутів, валідація та санітизація вхідних даних, граничні умови платіжного процесу.

Підсистема автентифікації є критичним компонентом системи, від коректності якого залежить безпека персональних даних клієнтів. Тестування охоплює сценарії входу з невірними даними, реєстрації через Google для вже існуючого акаунту та валідації полів форм реєстрації.

Таблиця 3.1

Результати тестування автентифікації та авторизації

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Вхід з невірним паролем	Email: існуючий; Пароль: «wrongpass»	Повідомлення про помилку, сесія не створена	Відображено повідомлення про невдалу спробу. Пароль очищено	Passed
2	Вхід з неіснуючим email	Email: «nouser@test.com»	Повідомлення, що не розкриває, який саме параметр хибний	Система повернула те саме повідомлення, що і для невірного пароля	Passed
3	Реєстрація через Google з email, що вже існує	Google-акаунт з адресою, вже зареєстрованою через форму	Пропозиція прив'язати Google до наявного акаунту без дублювання	Відображено проміжну сторінку підтвердження. Акаунт не дубльовано (рис. 3.31)	Passed
4	Реєстрація з паролем коротшим за мінімум	Пароль: «abc» (3 символи, вимога ≥ 6)	Повідомлення про мінімальну довжину, форма не відправлена	«Passwords must be at least 6 characters». Форма не відправлена	Passed
5	Реєстрація з паролями, що не збігаються	Пароль: «Secret123»; Підтвердження: «Secret321»	Повідомлення «Паролі не збігаються»	Серверна валідація ModelState. Форма повернута з повідмл.	Passed

продовж. Табл. 3.1

6	Повторна реєстрація з існуючим email	Email вже зареєстрованого користувача	Повідомлення «Email вже зайнятий»	Повернуто «Email is already taken». Запису не продубльовано	Passed
---	--------------------------------------	---------------------------------------	-----------------------------------	---	--------

Тест №3 потребує окремого коментаря. Відповідно до специфікації OAuth 2.0, система не повинна автоматично об'єднувати облікові записи без явного підтвердження користувача. Реалізований механізм відповідає цій вимозі – прив'язка відбувається лише після підтвердження на проміжній сторінці (рис. 3.31).

Рис. 3.31 – Проміжна сторінка прив'язки Google-провайдера до існуючого облікового запису

Несанкціонований доступ до адміністративної панелі може призвести до витоку даних клієнтів або фальсифікації замовлень. Перевірено три категорії: звернення неавторизованого гостя, звернення авторизованого клієнта без ролі Admin та підробка cookie.

При прямому переході гостя на /Admin/Dashboard браузер отримує HTTP 302 та перенаправляється на /Identity/Account/Login з параметром

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						84

ReturnUrl=/Admin/Dashboard (рис. 3.32). Після успішного входу адміністратора відбувається повернення на запитовану сторінку.

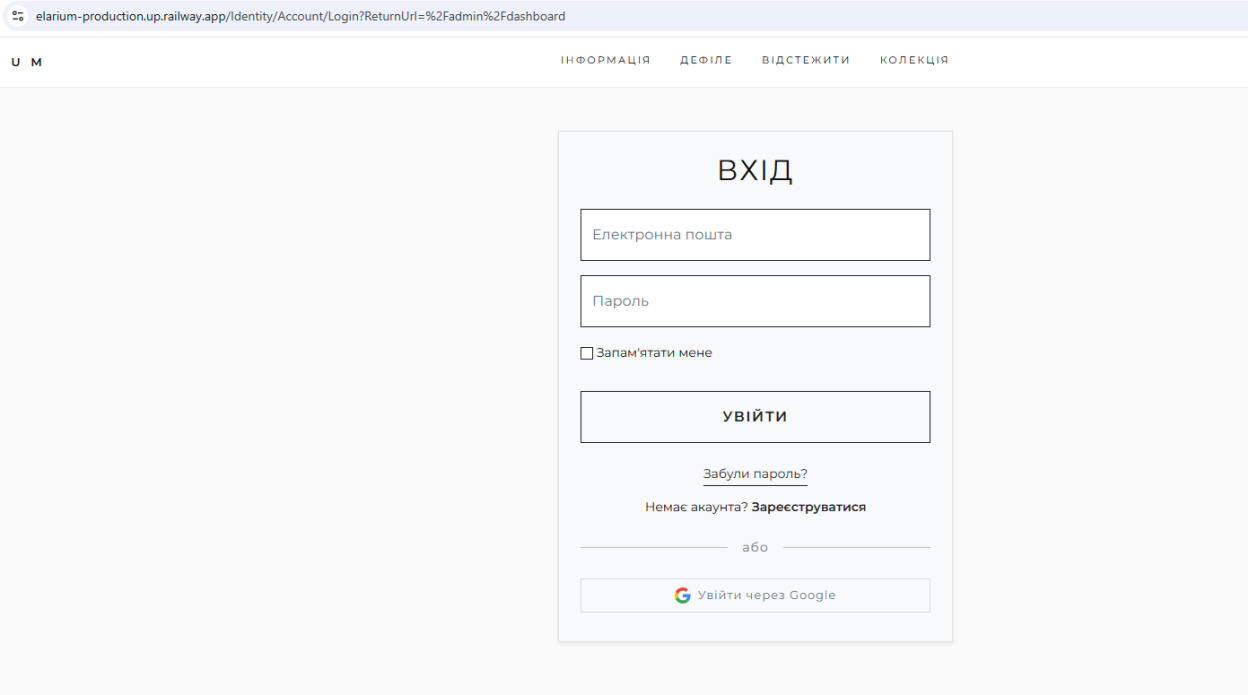


Рис. 3.32 – Перенаправлення на сторінку входу при спробі відкрити /Admin/Dashboard

Таблиця 3.2

Результати тестування захисту адміністративних маршрутів

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
7	Перехід на /Admin/Dashboard без авторизації	Гість вводить адресу у браузер	HTTP 302 → /Identity/Account/Login	Перенаправлено на Login з returnUrl. Після входу адміна – повернення на Dashboard	Passed
8	Перехід на /Products/Create без авторизації	Гість вводить адресу у браузер	Перенаправлення на сторінку входу	Поведінка ідентична тесту №7. Сторінка не відображена	Passed
9	Доступ до /Orders/Index авторизованим клієнтом (без ролі Admin)	Акаунт клієнта без ролі Admin	HTTP 403 або /Identity/Account/AccessDenied	Перенаправлено на /Identity/Account/AccessDenied	Passed

10	Пряме звернення до /Products/Delete/{id} без авторизації	URL з реальним Id товару без авторизації	Перенаправлення або відмова в доступі	Перенаправлення на Login. Товар не видалено	Passed
16	Підробка cookie для імітації ролі Admin	Підміна значення cookie у DevTools	Сервер відхиляє недійсний cookie	Cookie захищені підписом ASP.NET Core Data Protection. Підроблений cookie відхилено	Passed

Вхідні дані від користувача є потенційним вектором атак: впровадження HTML/JavaScript у поля форм (XSS), SQL-ін'єкції через параметри запиту, відправлення форм без обов'язкових полів. Нижче наведено результати перевірки стійкості системи до таких сценаріїв.

Таблиця 3.3

Результати тестування валідації та санітизації вхідних даних

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
12	Вставка HTML-тегів у поле адреси доставки	<script>alert('xss')</script> у полі «Відділення / Адреса»	Форма не відправляється, повідомлення про неприпустимий вміст	Клієнтська валідація: «Поле не може містити HTML-теги або скрипти». Форма заблокована (рис. 3.33)	Passed
13	SQL-конструкція у пошуковому рядку	«'; DROP TABLE Products; --»	Рядок обробляється як текст, таблиця не видаляється	EF Core використовує параметризовані запити. Товарів не знайдено, таблиця цілісна	Passed
14	Оформлення замовлення з порожнім полем ПІБ	Поле ПІБ залишено порожнім, решта полів заповнені	Повідомлення «Введіть ПІБ», форма не відправляється	Клієнтська валідація відображає підказку під полем. Замовлення не збережено	Passed

15	Підроблена сума у формі Checkout	Маніпуляція прихованим полем суми через DevTools браузера	Сервер перераховує суму з БД, клієнтське значення ігнорується	Checkout [POST] самостійно розраховує TotalAmount з цін БД. Підроблена сума не прийнята	Passed
16	Завантаження файлу .exe у галерею товару	Файл «test.exe» у полі завантаження фото товару	Сервер відхиляє файл з повідомленням про недозволений тип	Перевірка ContentType: дозволено лише image/jpeg та image/png. Файл відхилено	Passed

Особливу увагу привертає тест №12. На відміну від типового підходу, де HTML-теги лише екрануються при виводі рушієм Razor, у системі «ELARIUM» реалізовано проактивну клієнтську валідацію поля адреси: при введенні тексту, що містить символи «<» та «>» характерні для HTML-тегів або скриптів, форма миттєво відображає повідомлення «Поле не може містити HTML-теги або скрипти» та блокує відправлення (рис. 3.33). Це унеможливорює потрапляння потенційно шкідливого вмісту до бази даних ще на етапі введення.

Валідацію реалізовано власним атрибутом NoHtmlAttribute, який застосовано до текстових полів форми оформлення замовлення (ПІБ, місто, адреса) та до адміністративних полів товару. Перевірка виконується за регулярними виразами, блокуються HTML-подібні конструкції з символами < і >, а також небезпечні фрагменти на кшталт javascript: або обробників подій on*=. Якщо введення не проходить перевірку, ModelState повертає форму з повідомленням про помилку, не створюючи замовлення з некоректними даними. Це доповнює автоматичне HTML-кодування Razor при відображенні збережених значень і формує багаторівневий захист від XSS-ін'єкцій.

ОФОРМЛЕННЯ ЗАМОВЛЕННЯ

ДАНІ ПОКУПЦЯ

ПІБ

Введіть ПІБ

EMAIL ТЕЛЕФОН

МІСТО ВІДДІЛЕННЯ НОВОЇ ПОШТИ / АДРЕСА

Поле не може містити HTML-теги або скрипти

СПОСІБ ОПЛАТИ

КАРТКА (STRIPE)
Visa / Mastercard

WAYFORPAY
Українська платіжна система

ПЕРЕЙТИ ДО ОПЛАТИ

← Повернутись до кошика

Рис. 3.33 – Клієнтська валідація XSS-вмісту у полі адреси на сторінці
/Order/Checkout

Платіжний модуль є найбільш критичною частиною системи. Перевірено сценарії скасування платежу, дублювання webhook-сповіщення, переходу до Checkout з порожнім кошиком та некоректного пошуку замовлення.

Таблиця 3.4

Результати тестування платіжного процесу та граничних умов

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
17	Скасування платежу на сторінці Stripe	На сторінці Stripe натиснуто «← Back»	Повернення на Checkout. Замовлення зі статусом «Очікує оплати», IsPaid=false	Замовлення збережено зі статусом «Очікує оплати». Повторна оплата можлива	Passed

18	Повторний webhook від Stripe для вже оплаченого замовлення	Дублікат події checkout.session.completed для IsPaid=true	Ідемпотентна обробка: статус і залишки не змінюються вдруге	StripeWebhook перевіряє IsPaid перед оновленням. Повторна подія – HTTP 200 без змін у БД	Passed
19	Перехід на /Order/Checkout з порожнім кошиком	Прямий GET /Order/Checkout без товарів у сесії	Перенаправлення на /Cart/Index	Checkout() [GET] перевіряє кошик: якщо порожній – перенаправляє на Cart/Index	Passed
20	Некоректний номер замовлення на /Order/Track	OrderId: «99999» (не існує); Email: довільний	Повідомлення «Замовлення не знайдено» без деталей	Відображено повідомлення без розкриття інформації про інші замовлення	Passed

Тест При спробі перейти безпосередньо за адресою /Order/Checkout без попереднього додавання товарів до кошика система не відображає форму оформлення, а одразу перенаправляє користувача на сторінку /Cart/Index. Це унеможливорює створення порожнього замовлення та некоректний стан бази даних. Перевірка реалізована на рівні GET-методу контролера: якщо сесійний кошик не містить жодної позиції – виконується RedirectToAction на Cart/Index.

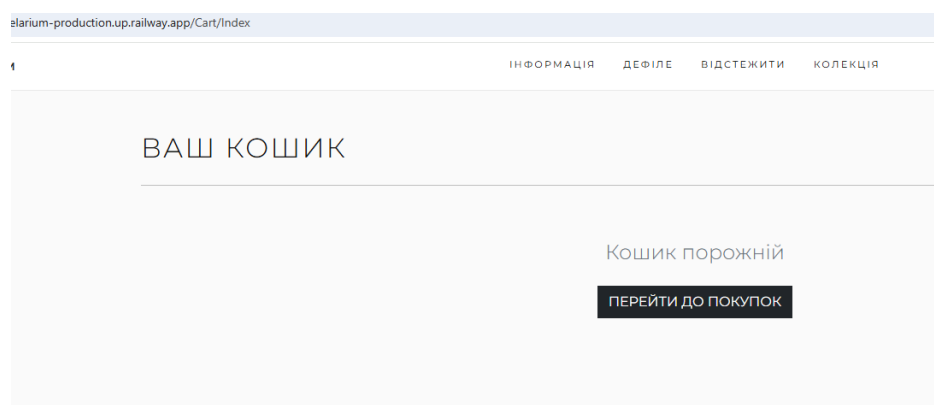


Рис. 3.34 – Спроба перейти на Order/Checkout з порожнім кошиком

Результати тестування підтверджують достатній рівень захисту системи від основних векторів атак – несанкціонованого доступу до адміністративних маршрутів, впровадження XSS-вмісту через поля форм, SQL-ін'єкцій через пошуковий рядок та підробки платіжних даних на стороні клієнта. Усі 20 тест-кейсів пройдено успішно, що підтверджує відповідність системи вимогам безпеки та функціональної коректності і готовність продукту до дослідної комерційної експлуатації.

Висновки до третього розділу

У третьому розділі розглянуто практичні аспекти впровадження, функціонування та тестування програмного комплексу інтернет-магазину «ELARIUM».

У підрозділі 3.1 описано порядок встановлення та налаштування системи. Розгортання реалізовано засобами Docker з використанням патерну багатоетапної збірки та хмарної платформи Railway, що забезпечує автоматичне CI/CD-розгортання при кожному коміті до репозиторію GitHub без простою сервісу. Конфігурація чутливих параметрів організована відповідно до методології «12-Factor App» через змінні середовища. Описано архітектуру розгортання з чотирма ключовими вузлами: браузером клієнта, Railway-контейнером з ASP.NET Core, базою даних PostgreSQL та зовнішніми сервісами Stripe, WayForPay і Google OAuth.

У підрозділі 3.2 наведено детальний опис інтерфейсу користувача. Клієнтська частина охоплює 19 ключових екранів: від головної сторінки з відеобанером та роллером колекції до розділу дефіле, сторінок автентифікації, каталогу з фільтрами, детальної картки товару, кошика, оформлення замовлення з двома платіжними шлюзами, сторінки успіху та відстеження замовлення. Адміністративна панель забезпечує управління товарами, замовленнями, дефіле та містить аналітичний дашборд.

У підрозділі 3.3 проведено функціональне тестування методом «чорної скриньки» за 20 тест-кейсом у чотирьох категоріях. Усі тест-кейси пройдено успішно – підтверджено захист від несанкціонованого доступу до адміністративних маршрутів, стійкість до XSS-атак завдяки клієнтській валідації полів форм, захист від SQL-ін'єкцій через параметризовані запити Entity Framework Core, а також коректну обробку граничних умов платіжного процесу. Результати тестування підтверджують готовність системи до дослідної комерційної експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра досягнуто поставленої мети – спроектовано та практично реалізовано повнофункціональний інтернет-магазин дизайнерського одягу «ELARIUM» на платформі ASP.NET Core MVC. Розроблений програмний комплекс замінює ручне ведення продажів через соціальні мережі та забезпечує централізоване управління каталогом товарів, автоматизований прийом онлайн-платежів і персоналізоване обслуговування клієнтів.

Відповідно до поставленого завдання реалізовано п'ять функціональних підсистем. Підсистема управління каталогом забезпечує повний набір CRUD-операцій над картками товарів, включаючи завантаження серії фотографій для галереї, управління залишками за розмірами та керування промо-контентом дефіле. Підсистема презентації товарів реалізує адаптивний каталог із багатокритеріальною фільтрацією за категорією, статтю, ціновим діапазоном і розміром, текстовим пошуком, а також детальну сторінку товару з інтерактивною галереєю та ефектом масштабування зображень і списком бажаного. Підсистема оформлення та оплати замовлень забезпечує наскрізний сценарій покупки через сесійний кошик із серверною валідацією залишків та інтеграцію двох платіжних шлюзів – міжнародного Stripe Checkout і локального WayForPay із захистом HMAC-підписом та ідемпотентною обробкою зворотних викликів. Підсистема супроводу замовлень надає відстеження статусу замовлення гостем за номером і електронною поштою без реєстрації, особистий кабінет із повною історією покупок для авторизованого клієнта та адміністративну панель керування замовленнями. Підсистема автентифікації та безпеки реалізує реєстрацію й вхід за обліковим записом Google (OAuth 2.0), рольове розмежування доступу, а також захист від типових вебзагроз на рівні HTTP-заголовків.

Усі визначені у завданні функціональні вимоги реалізовано в повному обсязі. Нефункціональні вимоги також виконано: застосунок розгорнуто на

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						92

хмарній платформі Railway з використанням Docker-контейнеризації та CI/CD-конвеєра з автоматичним zero-downtime deployment після кожного коміту до репозиторію GitHub; серверний рендеринг засобами рушія Razor забезпечує швидке первинне завантаження сторінок та SEO-оптимізацію; модульна архітектура із впровадженням залежностей спрощує подальше розширення функціоналу.

Практична цінність одержаних результатів полягає в тому, що розроблений програмний продукт повністю готовий до комерційного впровадження в діяльність бренду «ELARIUM» і задеплойований за адресою elarium-production.up.railway.app. Його використання дозволяє скоротити операційне навантаження на персонал бренду завдяки автоматизації прийому та обліку замовлень, забезпечити клієнтів зручним середовищем для детального перегляду колекцій та безпечного здійснення покупок, а також отримати аналітичні дані про продажі безпосередньо в адміністративній панелі. Відсутність залежності від CMS-платформ гарантує відсутність регулярних платежів за підписку та повну керованість системи.

Перспективними напрямками подальшого розвитку програмного комплексу є інтеграція з API служб доставки «Нова Пошта» та «Укрпошта» для автоматичного формування накладних і відстеження посилок безпосередньо з особистого кабінету; розширення аналітичного модуля адміністратора з побудовою графічних звітів динаміки продажів у розрізі категорій і часових проміжків; впровадження CDN для зберігання та роздачі медіаконтенту з метою подальшого зниження часу завантаження сторінок і зменшення навантаження на основний сервер; реалізація програми лояльності з накопичувальними бонусними балами для стимулювання повторних покупок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft Corporation ASP.NET Core MVC Overview [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview>
2. Microsoft Corporation. Developing ASP.NET Core MVC Apps [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/develop-asp-net-core-mvc-apps>
3. Microsoft Corporation. Introduction to Identity on ASP.NET Core [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity>
4. Microsoft Corporation. Entity Framework Core – Introduction to Performance [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/performance/>
5. Microsoft Corporation. ASP.NET Core Data Protection [Електронний ресурс]. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/security/data-protection/introduction>
6. PostgreSQL Global Development Group. PostgreSQL 15 Documentation [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.postgresql.org/docs/15/>
7. Npgsql Team. Npgsql – .NET Access to PostgreSQL: Documentation [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.npgsql.org/efcore/>
8. Stripe Inc. Stripe API Reference [Електронний ресурс]. – Режим доступу до ресурсу: <https://stripe.com/docs/api>
9. WayForPay. WayForPay: Документація платіжного сервісу [Електронний ресурс]. – Режим доступу до ресурсу: <https://wiki.wayforpay.com/>

10. Google LLC. Using OAuth 2.0 for Web Server Applications [Электронный ресурс]. – Режим доступа до ресурсу: <https://developers.google.com/identity/protocols/oauth2/web-server>
11. Docker Inc. Dockerfile Reference [Электронный ресурс]. – Режим доступа до ресурсу: <https://docs.docker.com/reference/dockerfile/>
12. Railway Technologies Inc. Railway Documentation [Электронный ресурс]. – Режим доступа до ресурсу: <https://docs.railway.app/>
13. Object Management Group. UML 2.5.1 Specification [Электронный ресурс]. – Режим доступа до ресурсу: <https://www.omg.org/spec/UML/2.5.1/>
14. Wiggins A. The Twelve-Factor App [Электронный ресурс]. – Режим доступа до ресурсу: <https://12factor.net/>
15. GitHub Inc. GitHub Actions Documentation [Электронный ресурс]. – Режим доступа до ресурсу: <https://docs.github.com/en/actions>
16. OWASP Foundation. OWASP Top Ten 2021 [Электронный ресурс]. – Режим доступа до ресурсу: <https://owasp.org/www-project-top-ten/>
17. Microsoft Corporation. Dependency Injection in ASP.NET Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection>
18. Microsoft Corporation. Session and State Management in ASP.NET Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/app-state>
19. Microsoft Corporation. Google External Login Setup in ASP.NET Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/social/google-logins>
20. Microsoft Corporation. Using External Login Providers with Identity in ASP.NET Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/social>

21. Microsoft Corporation. Migrations Overview – EF Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/>

22. Microsoft Corporation. Applying Migrations – EF Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/applying>

23. Microsoft Corporation. Overview of Caching in ASP.NET Core [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/performance/caching/overview>

24. Microsoft Corporation. ASP.NET Core Fundamentals Overview [Электронный ресурс]. – Режим доступа до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals>

25. Microsoft Corporation. NuGet Gallery: Microsoft.AspNetCore.Authentication.Google [Электронный ресурс]. – Режим доступа до ресурсу: <https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.Google>

26. Microsoft Corporation. NuGet Gallery: Npgsql.EntityFrameworkCore.PostgreSQL [Электронный ресурс]. – Режим доступа до ресурсу: <https://www.nuget.org/packages/Npgsql.EntityFrameworkCore.PostgreSQL>

27. Microsoft Corporation. NuGet Gallery: Microsoft.AspNetCore.Identity.EntityFrameworkCore [Электронный ресурс]. – Режим доступа до ресурсу: <https://www.nuget.org/packages/Microsoft.AspNetCore.Identity.EntityFrameworkCore>

28. Stripe Inc. How Checkout Works [Электронный ресурс]. – Режим доступа до ресурсу: <https://stripe.com/docs/payments/checkout/how-checkout-works>

29. Stripe Inc. Fulfill Orders with Checkout [Электронный ресурс]. – Режим доступа до ресурсу: <https://stripe.com/docs/payments/checkout/fulfill-orders>

30. Stripe Inc. Stripe Webhooks – Best Practices [Електронний ресурс]. – Режим доступу до ресурсу: <https://stripe.com/docs/webhooks/best-practices>

31. Stripe Inc. Set Up Your Development Environment (.NET) [Електронний ресурс]. – Режим доступу до ресурсу: <https://stripe.com/docs/development/quickstart?lang=dotnet>

32. Docker Inc. Multi-stage Builds [Електронний ресурс]. – Режим доступу до ресурсу: <https://docs.docker.com/build/building/multi-stage/>

33. OWASP Foundation. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet [Електронний ресурс]. – Режим доступу до ресурсу: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

34. Object Management Group. OMG Unified Modeling Language (UML) Specification, Version 2.5.1 [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.omg.org/spec/UML/2.5.1/>

35. Платонов М.С., Лобанчикова Н.М. Проектування інтернет-магазину дизайнерського одягу на базі ASP.NET Core MVC. Тези Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених «Тенденції розвитку технологій в автоматизації, електроенергетиці, приладобудуванні та робототехніці» (присвяченої Дню науки), м. Житомир, 13-15 травня 2026 р. – Житомир: Житомирська політехніка, 2026.

36. Платонов М.С., Лобанчикова Н.М. Застосування сучасних технологій електронної комерції у fashion-ритейлі. Тези Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених «Тенденції розвитку технологій в автоматизації, електроенергетиці, приладобудуванні та робототехніці» (присвяченої Дню науки), м. Житомир, 13-15 травня 2026 р. – Житомир: Житомирська політехніка, 2026.

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						98

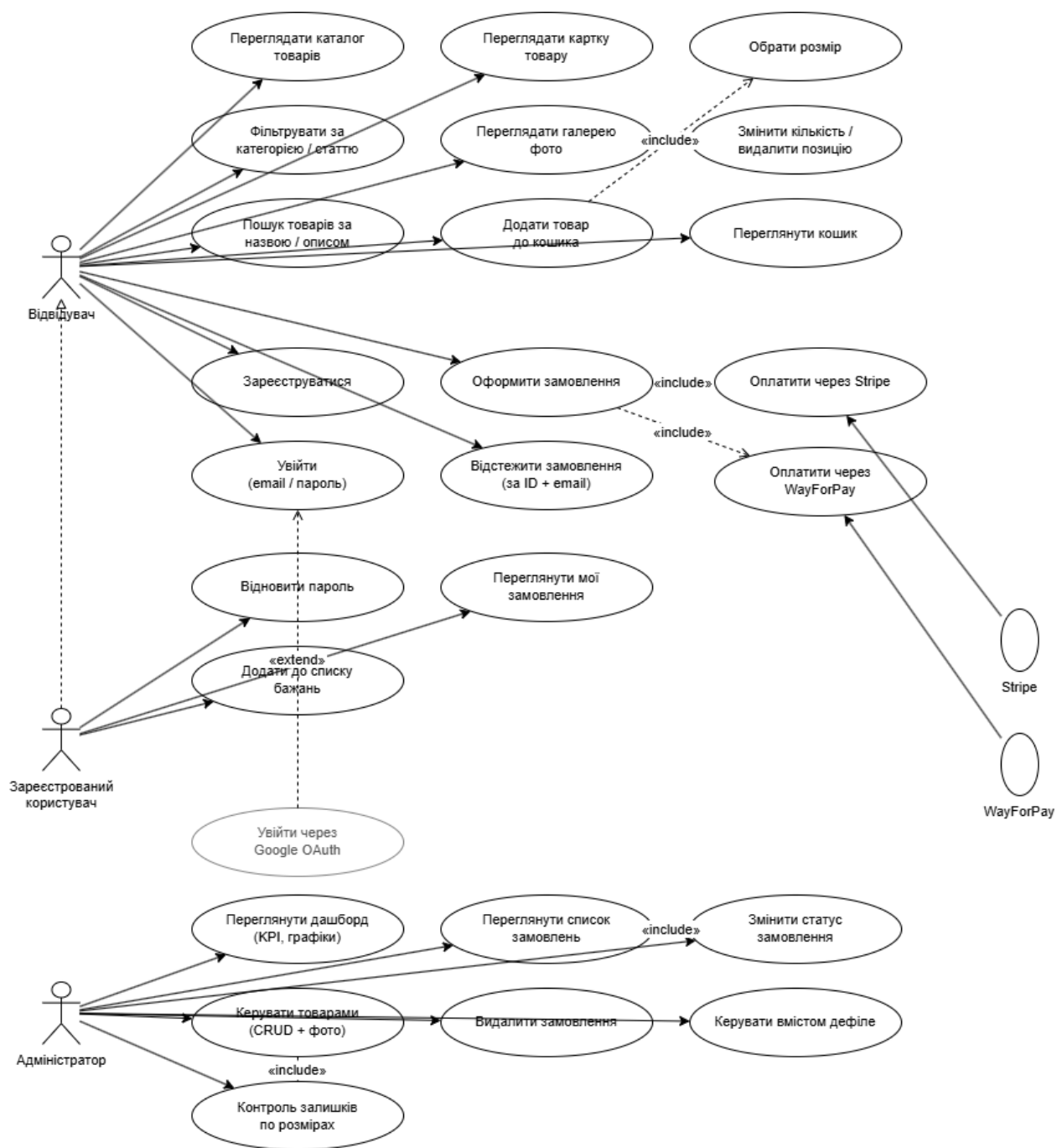


Рис. А.1 – Діаграма варіантів використання



Рис. Б.1 – Діаграма класів

Доменні моделі відображають структуру даних магазину: товар з підтримкою залишків по розмірах, замовлення з повним life-cycle статусів та прив'язкою до платіжних систем:

```
// Models/Product.cs
public class Product
{
    [Key]
    public int Id { get; set; }

    [Required(ErrorMessage = "Назва товару обов'язкова")]
    [MaxLength(100)]
    [NoHtml]
    public string Name { get; set; } = string.Empty;

    [NoHtml]
    public string? Description { get; set; }

    [Required]
    public decimal Price { get; set; }

    public string? ImageUrl { get; set; } = string.Empty;
    public List<ProductImage> Images { get; set; } = new();
    public bool IsAvailable { get; set; } = true;
    public string? Sizes { get; set; } = "S, M, L, XL";
    [NoHtml]
    public string? Category { get; set; }
    [NoHtml]
    public string? Gender { get; set; }
    public int Stock { get; set; } = 0; // загальний залишок (legacy)

    public List<ProductSizeStock> SizeStocks { get; set; } = new();
}
```

```

// Models/ProductSizeStock.cs
public class ProductSizeStock
{
    public int    Id      { get; set; }
    public int    ProductId { get; set; }
    public Product Product { get; set; } = null!;
    public string Size     { get; set; } = string.Empty;
    public int    Quantity { get; set; } = 0;
}

// Models/Order.cs
public class Order
{
    public int Id { get; set; }

    [MaxLength(100)] public string CustomerName { get; set; } = string.Empty;
    [MaxLength(200)] public string CustomerEmail { get; set; } = string.Empty;
    [MaxLength(30)]  public string CustomerPhone { get; set; } = string.Empty;
    [MaxLength(100)] public string City          { get; set; } = string.Empty;
    public string Address { get; set; } = string.Empty;

    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public OrderStatus Status   { get; set; } = OrderStatus.Pending;
    public decimal  TotalAmount { get; set; }

    public string? StripeSessionId { get; set; }
    public string? PaymentIntentId { get; set; }

    public List<OrderItem> Items { get; set; } = new();
}

public enum OrderStatus
{
    Pending = 0, Paid = 1, Processing = 2,
    Shipped = 3, Delivered = 4, Cancelled = 5
}

```

```
}
```

Реалізація кошика через HTTP-сесію з перевіркою залишків по розміру та підтримкою AJAX-запитів:

[HttpPost]

```
public async Task<IActionResult> AddToCart(int productId, string size)
{
    var isAjax = Request.Headers["X-Requested-With"] == "XMLHttpRequest";

    var product = await _context.Products
        .Include(p => p.SizeStocks)
        .FirstOrDefaultAsync(p => p.Id == productId);
    if (product == null)
        return isAjax ? Json(new { success = false, error = "Товар не знайдено" }) : NotFound();

    var sizeStock = product.SizeStocks.FirstOrDefault(s => s.Size == size);
    if (sizeStock != null)
    {
        var cart = GetCart();
        var inCart = cart.Where(c => c.ProductId == productId && c.Size == size).Sum(c =>
c.Quantity);

        if (sizeStock.Quantity == 0)
        {
            if (isAjax) return Json(new { success = false, error = $"Розмір {size} – розпродано" });
            TempData["CartError"] = $"Розмір {size} – SOLD OUT";
            return RedirectToAction("Details", "Home", new { id = productId });
        }
        if (inCart >= sizeStock.Quantity)
        {
            if (isAjax) return Json(new { success = false, error = $"Більше {sizeStock.Quantity} шт.
розміру {size} немає в наявності" });
            TempData["CartError"] = $"Більше {sizeStock.Quantity} шт. розміру {size} немає в
наявності";
            return RedirectToAction("Details", "Home", new { id = productId });
        }
    }
}
```

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						103

```

    }

    var existing = cart.FirstOrDefault(c => c.ProductId == productId && c.Size == size);
    if (existing != null) existing.Quantity++;
    else cart.Add(new CartItem { ProductId = product.Id, Name = product.Name,
        Price = product.Price, ImageUrl = product.ImageUrl, Size = size, Quantity = 1 });

    SaveCart(cart);
    if (isAjax) return Json(new { success = true, cartCount = cart.Sum(c => c.Quantity),
        productName = product.Name });
    return RedirectToAction("Index");
}
// ...
}

private List<CartItem> GetCart()
{
    var json = HttpContext.Session.GetString("Cart");
    return string.IsNullOrEmpty(json) ? new() :
        JsonSerializer.Deserialize<List<CartItem>>(json)!;
}

private void SaveCart(List<CartItem> cart) =>
    HttpContext.Session.SetString("Cart", JsonSerializer.Serialize(cart));

```

Оформлення замовлення з вибором між Stripe та WayForPay,
ідемпотентне підтвердження оплати та зниження залишків:

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Checkout(CheckoutViewModel model)
{
    // ... валідація ...

    var order = new Order
    {

```

					ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
						104

```

CustomerName = model.Name,
CustomerEmail = accountEmail,
CustomerPhone = model.Phone,
City        = model.City,
Address     = model.Address,
TotalAmount = cart.Sum(i => i.Price * i.Quantity),
Status      = OrderStatus.Pending,
Items       = cart.Select(c => new OrderItem
{
    ProductId = c.ProductId,
    ProductName = c.Name,
    ImageUrl  = c.ImageUrl,
    Size      = c.Size,
    Price     = c.Price,
    Quantity  = c.Quantity
}).ToList()
};

_context.Orders.Add(order);
await _context.SaveChangesAsync();

// — WayForPay —
if (model.PaymentMethod == "wayforpay")
{
    var fields = _wfp.BuildFormFields(order, domain, cart);
    return View("WayForPayForm", fields);
}

// — Stripe (default) —
var sessionOptions = new SessionCreateOptions
{
    PaymentMethodTypes = new List<string> { "card" },
    LineItems          = lineItems,
    Mode               = "payment",
    CustomerEmail      = model.Email,

```

```

        SuccessUrl =
        $"{domain}/Order/Success?session_id={{CHECKOUT_SESSION_ID}}&order_id={order.Id}"
    ,
        CancelUrl = $"{domain}/Cart",
        Metadata = new Dictionary<string, string> { { "order_id", order.Id.ToString()
    } }

    };

    var session = await new SessionService().CreateAsync(sessionOptions);
    order.StripeSessionId = session.Id;
    await _context.SaveChangesAsync();
    return Redirect(session.Url);
}

// Ідемпотентно: позначає замовлення оплаченим і зменшує сток лише раз
private async Task MarkOrderPaidAsync(Order order, string? authCode)
{
    if (order.Status != OrderStatus.Pending) return; // вже оброблено – не дублюємо

    order.Status = OrderStatus.Paid;
    order.PaymentIntentId = authCode;
    await ReduceStockAsync(order);

    var productIds = order.Items.Select(i => i.ProductId).Distinct().ToList();
    foreach (var pid in productIds)
    {
        var productStocks = await _context.ProductSizeStocks
            .Where(s => s.ProductId == pid).ToListAsync();
        if (productStocks.Any() && productStocks.All(s => s.Quantity == 0))
        {
            var product = await _context.Products.FindAsync(pid);
            if (product != null) product.IsAvailable = false;
        }
    }
    await _context.SaveChangesAsync();
}

```

Формування підпису HMAC-MD5 та побудова полів форми для
редиректу на платіжний шлюз WayForPay:

```
public class WayForPayService
{
    public const string PaymentUrl = "https://secure.wayforpay.com/pay";

    // HMAC-MD5 signature
    public string Sign(params string[] parts)
    {
        var message = string.Join(";", parts);
        using var hmac = new HMACMD5(Encoding.UTF8.GetBytes(MerchantSecretKey));
        return
Convert.ToHexString(hmac.ComputeHash(Encoding.UTF8.GetBytes(message))).ToLower();
    }

    public List<(string Name, string Value)> BuildFormFields(
        Order order, string domain, List<CartItem> items)
    {
        var orderRef = $"ELARIUM-{order.Id}";
        var orderDate = DateTimeOffset.UtcNow.ToUnixTimeSeconds().ToString();
        var amount = Fmt(order.TotalAmount);

        var productNames = items.Select(i => i.Name.Replace("\\", "").Replace(" ",
"").Replace(";", "")).ToList();
        var productPrices = items.Select(i => Fmt(i.Price)).ToList();
        var productCounts = items.Select(i => i.Quantity.ToString()).ToList();

        // Порядок підпису строго за документацією WayForPay
        var sigParts = new List<string>
        { MerchantAccount, MerchantDomainName, orderRef, orderDate, amount,
"UAH" };

        sigParts.AddRange(productNames);
        sigParts.AddRange(productCounts);
        sigParts.AddRange(productPrices);
        var signature = Sign(sigParts.ToArray());
    }
}
```

					ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
						107

```

var fields = new List<(string, string)>
{
    ("merchantAccount", MerchantAccount),
    ("merchantAuthType", "simpleSignature"),
    ("orderReference", orderRef),
    ("orderDate", orderDate),
    ("amount", amount),
    ("currency", "UAH"),
    ("returnUrl", $"{domain}/Order/WayForPayReturn"),
    ("serviceUrl", $"{domain}/Order/WayForPayCallback"),
    ("merchantSignature", signature),
};
// ...
return fields;
}
}

```

Реєстрація DI-сервісів, налаштування middleware-pipeline (сесії, автентифікація, security headers), seed адмін-ролі та автоматізація для деплою на Railway (Program.cs):

```

var builder = WebApplication.CreateBuilder(args);
StripeConfiguration.ApiKey = builder.Configuration["Stripe:SecretKey"];

builder.Services.AddScoped<WayForPayService>();
builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromHours(2);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
    options.Cookie.SecurePolicy = cookieSecure;
});

```

```

        options.Cookie.SameSite    = SameSiteMode.Lax;
        options.Cookie.Name       = ".ELARIUM.Session";
    });

    builder.Services.AddDefaultIdentity<IdentityUser>(o =>
o.SignIn.RequireConfirmedAccount = false)
        .AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>();

    // Google OAuth (підключається лише якщо ключі задано в конфігурації)
    if (!string.IsNullOrEmpty(googleId) && !string.IsNullOrEmpty(googleSecret))
    {
        builder.Services.AddAuthentication()
            .AddGoogle(options => { options.ClientId = googleId; options.ClientSecret =
googleSecret; });
    }

    // Security headers middleware
    app.Use(async (ctx, next) =>
    {
        var h = ctx.Response.Headers;
        h["X-Content-Type-Options"] = "nosniff";
        h["X-Frame-Options"]       = "SAMEORIGIN";
        h["Referrer-Policy"]       = "strict-origin-when-cross-origin";
        h["Permissions-Policy"]    = "camera=(), microphone=(), geolocation=()";
        await next();
    });

    // Seed ролей та admin-акаунту при запуску
    using (var scope = app.Services.CreateScope())
    {
        var db = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
        db.Database.Migrate(); // автоміграція для Railway
    }

```

```

var roleManager =
scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();

var userManager =
scope.ServiceProvider.GetRequiredService<UserManager<IdentityUser>>();

if (!await roleManager.RoleExistsAsync("Admin"))
    await roleManager.CreateAsync(new IdentityRole("Admin"));

if (await userManager.FindByEmailAsync("admin@store.com") == null)
{
    var admin = new IdentityUser { UserName = "admin@store.com", Email =
"admin@store.com", EmailConfirmed = true };
    await userManager.CreateAsync(admin, "Admin_12345!");
    await userManager.AddToRoleAsync(admin, "Admin");
}
}

var railwayPort = Environment.GetEnvironmentVariable("PORT");
if (!string.IsNullOrEmpty(railwayPort))
    appUrls.Add($"http://0.0.0.0:{railwayPort}");

app.Run();

```

Також варто винести окремо кастомний атрибут валідації NoHtmlAttribute (файл Models/NoHtmlAttribute.cs) – він захищає текстові поля від XSS-ін'єкцій на рівні моделі, до збереження в базу:

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public sealed class NoHtmlAttribute : ValidationAttribute
{
    private static readonly Regex TagPattern =
        new(@"<[a-zA-Z/!][^>]*>", RegexOptions.Compiled);

    private static readonly Regex DangerPattern =
        new(@"(javascript\s*:\s*|on\w+\s*=|data\s*:\s*)", RegexOptions.Compiled |
        RegexOptions.IgnoreCase);

    protected override ValidationResult? IsValid(object? value, ValidationContext ctx)
    {
        if (value is not string s || string.IsNullOrEmpty(s))
            return ValidationResult.Success;

        if (TagPattern.IsMatch(s) || DangerPattern.IsMatch(s))
            return new ValidationResult(ErrorMessage);

        return ValidationResult.Success;
    }
}
```