

БАКАЛАВРСЬКА РОБОТА
БР.ІП-03.00.00.000 ПЗ

Група ІП-22-4

Кузів Максим

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Кузів Максим Зеновійович

(прізвище, ім'я, по батькові)

УДК 004.42

(індекс)

БАКАЛАВРСЬКА РОБОТА

Проектування сервіс-орієнтованого веб-застосунку

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня М. З. Кузів

(підпис, ініціали та прізвище здобувача)

Науковий керівник Саманів Любов Василівна, асистент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада)

(підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

5. Сторінка перегляду товару з можливістю вибору варіацій (SKU) (рис. 4.2, ст. 40)

5.1 Консультанти розділів проекту (роботи)

<i>Розділ</i>	<i>Консультант</i>	<i>Підпис, дата</i>	
		<i>Завдання видав</i>	<i>Завдання прийняв</i>

5.2 Дата видачі завдання _____

Керівник

(підпис)

Саманів Л. В.

(розшифровка підпису)

Завдання прийняв до виконання

(підпис)

Кузів М. З.

(розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

<i>№ з/п</i>	<i>Назва етапів дипломного проекту (роботи)</i>	<i>Строк виконання етапів проекту</i>	<i>Примітка</i>
1	Опрацювання теоретичних матеріалів з теми роботи	27.02.2026	виконано
2	Вибір інструментів, технологій та методів розробки	05.03.2026	виконано
3	Налаштування середовища та конфігурація бази даних	23.03.2026	виконано
4	Розробка компонентів та функціональних модулів системи	13.04.2026	виконано
5	Тестування системи та усунення виявлених помилок	23.04.2026	виконано
6	Оформлення пояснювальної записки бакалаврської роботи	09.06.2026	виконано

Студент – дипломник

(підпис)

Кузів М. З.

(розшифровка підпису)

Керівник роботи

(підпис)

Саманів Л. В.

(розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 73 сторінок, 8 рисунків, 4 таблиці, 35 джерел.

Метою роботи є розробка веб-застосунку інтернет-магазину одягу для підвищення ефективності онлайн-продажів.

Об'єкт дослідження: процеси функціонування веб-застосунків у сфері електронної комерції.

Предмет дослідження: методи та інструменти розробки інтернет-магазину (архітектура, бази даних, зовнішні інтеграції).

Результати дослідження: Створено веб-застосунок на базі Ruby on Rails та PostgreSQL для автоматизації покупок.

В першому розділі проаналізовано тренди e-commerce та обґрунтовано вибір сервіс-орієнтованої архітектури (SOA).

В другому розділі сформульовано вимоги до системи та завдання проєктування.

В третьому розділі спроектовано MVC-архітектуру, бізнес-процеси та структуру бази даних.

В четвертому розділі описано практичну реалізацію та інтеграцію API Monobank і Нової Пошти.

У п'ятому розділі проведено тестування та розгортання системи через Docker.

Висновок: Програмний продукт успішно розроблено, він повністю відповідає сучасним вимогам електронної комерції.

КЛЮЧОВІ СЛОВА: ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН ОДЯГУ, RUBY ON RAILS, СЕРВІС-ОРІЄНТОВАНА АРХІТЕКТУРА, SOA, REST API, ІНТЕГРАЦІЯ, MONOBANK, НОВА ПОШТА

ABSTRACT

The bachelor's thesis contains 73 pages, 8 figures, 4 tables, 35 references.

The aim of the work is to develop a web application for an online clothing store to improve e-commerce efficiency.

Object of research: functioning processes of web applications in e-commerce.

Subject of research: methods and tools for creating an online store (architecture, databases, external integrations).

Results of the research: A web application based on Ruby on Rails and PostgreSQL was created to automate shopping.

In the first chapter, e-commerce trends are analyzed, and the choice of SOA is substantiated.

In the second chapter, system requirements and design tasks are formulated.

In the third chapter, MVC architecture, business processes, and database structure are designed.

In the fourth chapter, practical implementation and Monobank/Nova Post API integrations are described.

In the fifth chapter, testing and system deployment via Docker are conducted.

Conclusion: The software product has been successfully developed and fully meets modern e-commerce requirements.

KEYWORDS: E-COMMERCE, ONLINE CLOTHING STORE, RUBY ON RAILS, SERVICE-ORIENTED ARCHITECTURE, SOA, REST API, INTEGRATION, MONOBANK, NOVA POST.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	8
ВСТУП.....	9
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ОДЯГУ.....	11
1.1 Аналіз сучасного стану електронної комерції у сфері продажу одягу.....	12
1.2 Основні поняття та особливості предметної області інтернет-магазину.....	13
1.3.Сучасні технології та тенденції розробки сервіс-орієнтованих застосунків.....	16
1.4 Висновки по розділу.....	19
РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ.....	21
2.1 Аналіз вимог до веб-застосунку інтернет-магазину одягу.....	21
2.2 Формулювання функціональних та нефункціональних вимог.....	23
2.3 Постановка задачі проектування системи	28
2.4 Висновки по розділу.....	30
РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ.....	32
3.1 Розробка архітектури програмного забезпечення.....	32
3.2 Моделювання бізнес-процесів та системних компонентів.....	34
3.3 Вибір та обґрунтування технологій та інструментів розробки.....	36
3.4 Висновки по розділу.....	41
РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ.....	42
4.1 Опис розробки програмного коду та структури проекту.....	42
4.2 Реалізація логіки кошика та процесу оформлення замовлення	44
4.3 Інтеграція платіжної системи та розробка панелі адміністратора.....	47
4.4 Висновки по розділу.....	50
РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ.....	55
5.1 Тестування програмного забезпечення	55

					БР.ІП – 03.00.00.000 ІПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Проектування сервіс-орієнтованого веб- застосунку Пояснювальна записка	Лім.	Арк.	Аркушів
Розроб.	Кузів М. З.						6	75
Перевір.	Саманів Л. В.							
Реценз.	Корнута В. А.							
Н. Контр.	Піх М.М.							
Затверд.	Бандура В.В.					ІФНТУНГ ІП-22-4		

5.2 Розгортання та експлуатація системи.....	59
5.3 Аналіз ефективності впровадження.....	62
5.4 Висновки по розділу.....	67
ВИСНОВКИ.....	69
СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ.....	71
БІБЛІОГРАФІЧНА ДОВІДКА	

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACID (Atomicity, Consistency, Isolation, Durability) — набір властивостей (атомарність, узгодженість, ізольованість, довговічність), що гарантують надійність транзакцій бази даних.

API (Application Programming Interface) — прикладний програмний інтерфейс; набір правил та інструментів для взаємодії між різними програмами.

CMS (Content Management System) — система управління контентом (вмістом) сайту.

CSRF (Cross-Site Request Forgery) — підробка міжсайтових запитів (вид вразливості вебзастосунків).

CSS (Cascading Style Sheets) — каскадні таблиці стилів для візуального оформлення вебсторінок.

ER-модель (Entity-Relationship model) — модель «сутність-зв'язок» для проєктування структури бази даних.

ERB (Embedded Ruby) — система шаблонізації для вбудовування коду Ruby в HTML-розмітку.

HTML (HyperText Markup Language) — стандартизована мова гіпертекстової розмітки вебсторінок.

HTTP (HyperText Transfer Protocol) — мережевий протокол передачі даних в інтернеті.

JSON (JavaScript Object Notation) — зручний текстовий формат обміну даними між сервером та клієнтом.

MVC (Model-View-Controller) — архітектурний шаблон, що розділяє систему на модель, представлення та контролер.

ORM (Object-Relational Mapping) — технологія зв'язування таблиць бази даних з об'єктами мови програмування.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Електронна комерція є одним із найважливіших напрямів розвитку сучасних інформаційних систем. Веб-застосунки для продажу товарів уже давно виконують не лише функцію електронного каталогу, а стали повноцінними сервісами для пошуку, вибору, оплати, доставки та адміністрування замовлень. Особливо актуальним це є для сфери fashion-ритейлу та продажу одягу, де користувач очікує швидкого доступу до характеристик (зокрема, розмірних та кольорових варіацій), якісних зображень, актуальної ціни, зручного кошика, зрозумілого оформлення замовлення та можливості вибору способу доставки й оплати.

Актуальність теми бакалаврської роботи зумовлена потребою розроблення сучасного сервіс-орієнтованого інтернет-магазину одягу, який поєднує зручний користувацький інтерфейс, серверну бізнес-логіку, адміністративне керування та інтеграцію із зовнішніми сервісами. На практиці спеціалізований магазин одягу має працювати з великою кількістю категорій, товарними варіаціями (SKU), кошиком, замовленнями, автоматизованою доставкою. Тому створення такого веб-застосунку є актуальним як з погляду програмної інженерії, так і з погляду практичного використання в електронній комерції.

Об'єктом дослідження є процес розроблення сервіс-орієнтованого веб-застосунку електронної комерції для продажу одягу та аксесуарів.

Предмет дослідження є методи, архітектурні рішення та програмні засоби створення інтернет-магазину одягу з підтримкою каталогу товарів, товарних варіацій (SKU), кошика, замовлень, авторизації, доставки та адміністративної панелі.

Метою бакалаврської роботи є розроблення сервіс-орієнтованого веб-застосунку інтернет-магазину одягу, який забезпечує продаж товарів, вибір розмірних та кольорових варіацій, оформлення замовлень, вибір доставки, онлайн-оплату, авторизацію користувачів і керування магазином

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

через адміністративну панель.

Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати сучасний стан електронної комерції у сфері fashion-ритейлу та визначити особливості предметної області.
- розглянути існуючі технології, архітектурні підходи й інструменти розробки сервісних веб-застосунків.
- сформулювати функціональні та нефункціональні вимоги до системи інтернет-магазину.
- спроектувати архітектуру веб-застосунку, основні бізнес-процеси, компоненти та структури даних.
- провести тестування системи, оцінити результати та визначити можливості подальшого розвитку.

У роботі використано методи аналізу предметної області, порівняння існуючих рішень, моделювання програмної архітектури, компонентного проєктування, структурного аналізу вимог, ручного й автоматизованого тестування. Для реалізації системи використано Ruby, Ruby on Rails, PostgreSQL, Redis, Sidekiq, Monobank API та Nova Post API.

Практичне значення роботи полягає у створенні працездатного прототипу сервіс-орієнтованого інтернет-магазину одягу з каталогом товарів, підтримкою товарних варіацій (SKU), кошиком, оформленням замовлень, доставкою, авторизацією та адміністративною панеллю.

У результаті роботи буде розроблено веб-застосунок інтернет-магазину одягу, який дозволить користувачам переглядати асортимент товарів, обирати потрібні розміри й кольори, оформлювати замовлення та здійснювати безготівкову оплату, а адміністратору — керувати каталогом, залишками на складі, замовленнями й логістичними даними.

Структура та обсяг роботи. Робота містить 73 сторінок, 8 рисунків, 4 таблиці і 35 джерел. Складається з вступу, п'яти розділів, висновків та додатків, де описано аналіз, проєктування, реалізацію та тестування веб-застосунку інтернет-магазину одягу.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ОДЯГУ

1.1. Аналіз сучасного стану електронної комерції у сфері продажу одягу

Електронна комерція є одним із ключових напрямів розвитку сучасних інформаційних систем, оскільки поєднує технологічні, економічні, організаційні та правові аспекти взаємодії між продавцем і покупцем. В українському законодавстві електронна комерція визначається як сфера відносин, що виникають під час вчинення електронних правочинів із застосуванням інформаційно-комунікаційних систем. Тому сучасний інтернет-магазин одягу слід розглядати не лише як вітрину з переліком товарів, а як комплексну програмну систему, що забезпечує пошук, вибір розміру та кольору, оформлення замовлення, оплату, доставку та збереження даних користувача [1].

Цифрова економіка активно розширюється, а електронна комерція є важливою частиною глобальних економічних процесів. Ринок продажу одягу онлайн (fashion e-commerce) має свою яскраво виражену специфіку [3]. На відміну від простих стандартизованих товарів, одяг вимагає від веб-застосунку підтримки складної системи атрибутів: розмірних сіток, матеріалів, фасонів, доступних кольорів та їх відповідних зображень. Покупець очікує, що сайт дозволить швидко порівняти товари, переглянути якісні зображення з різних ракурсів та оформити замовлення без зайвих дій.

Сучасні рішення у сфері електронної комерції можна умовно поділити на три групи: універсальні маркетплейси, спеціалізовані магазини та SaaS/CMS-платформи для швидкого запуску інтернет-магазину.

- Універсальні маркетплейси мають масштаб і розвинену логістику, але вони менш гнучкі щодо специфіки окремої предметної області, обмежуючи продавця у налаштуванні унікального користувацького досвіду.

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

- SaaS- та CMS-рішення дозволяють швидко створити базовий магазин, однак часто обмежують індивідуальну логіку, складні інтеграції або кастомну адміністративну панель.

- Спеціалізовані магазини краще враховують особливості предметної області. Для індустрії одягу саме власна спеціалізована розробка дозволяє реалізувати унікальні бізнес-процеси (наприклад, складні фільтри за тканиною чи стилем, специфічні промо-акції) та інтегрувати локальні українські сервіси без милиць.

Однією з поширених проблем інтернет-магазинів є фрагментарність користувацького досвіду. Дослідження Baymard Institute показують, що для електронної комерції суттєве значення мають повнота інформації у списках товарів, зрозуміла фільтрація, якісний пошук, коректна checkout-логіка та зменшення зайвих перешкод на шляху до покупки. [19] Недостатність інформації у товарних списках ускладнює процес вибору та змушує користувачів виконувати зайві переходи. Для одягу критично важливим є миттєве оновлення фотографії при виборі іншого кольору чи паттерну тканини без повного перезавантаження сторінки.

Ще однією важливою особливістю електронної комерції є інтеграція із зовнішніми сервісами. Інтернет-магазин зазвичай не реалізує самостійно всі платіжні, логістичні та авторизаційні механізми, а інтегрується з платіжними системами, службами доставки та іншими API. Для українського ринку стандартом де-факто є використання сучасних інструментів еквайрингу (наприклад, Monobank API) для онлайн-оплат та інтеграція API логістичних операторів (Нова Пошта) для автоматичного підтягування актуальних відділень чи поштоматів.

Поряд із функціональністю важливими залишаються правові та безпекові вимоги. Інтернет-магазин збирає персональні дані користувача: ім'я, номер телефону, електронну пошту, адресу доставки та історію замовлень[2]. В Україні обробка персональних даних регулюється Законом України «Про

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

захист персональних даних». Крім того, продаж товарів споживачам повинен враховувати вимоги Закону України «Про захист прав споживачів».

З погляду програмної інженерії, інтернет-магазин є сервісним веб-застосунком, у якому поєднуються клієнтська частина, серверна логіка, API, сховище даних, модулі авторизації, оплати, доставки та адміністративна частина. Міжнародний стандарт ISO/IEC 25010 визначає модель якості програмного продукту, яка включає функціональну придатність, продуктивність, сумісність, зручність використання, надійність, безпеку, супроводжуваність і переносимість [5]. Усі ці метрики обов'язково мають бути враховані при проєктуванні інтернет-магазину одягу, щоб забезпечити його конкурентоздатність.

1.2. Основні поняття та особливості предметної області інтернет-магазину

Для проєктування веб-застосунку інтернет-магазину необхідно чітко визначити базові поняття предметної області та термінологічний апарат програмної інженерії, що використовуватимуться в роботі. У межах цього дослідження інтернет-магазин розглядається як динамічний сервіс-орієнтований веб-застосунок, що забезпечує користувачам можливість перегляду каталогу товарів, здійснення пошуку, формування кошика, оформлення замовлення, проведення онлайн-оплати, вибору способу доставки, а також передбачає авторизацію клієнтів та адміністративне керування даними. Він безперервно отримує дані із сервера, змінює стан інтерфейсу залежно від контексту, виконує валідацію введеної інформації, зберігає поточний стан сесії та взаємодіє із зовнішніми інфраструктурними сервісами. Для інтернет-магазину одягу це означає, що програмна система повинна не лише статично відображати вебсторінки з фотографіями речей, а й забезпечувати повноцінний, безперервний сценарій покупки від першого кліку до успішної транзакції.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

Фундаментальною основою роботи сучасних веб-застосунків є клієнт-серверна архітектура, яка передбачає логічне та фізичне розділення функцій між браузером користувача та віддаленим сервером. Клієнтська частина, або фронтенд, працює безпосередньо у браузері покупця та відповідає за відображення візуального інтерфейсу: сторінок, товарних карток, інтерактивних форм та кошика. Серверна частина, або бекенд, відповідає за обробку вхідних запитів, виконання ключової бізнес-логіки, взаємодію з реляційною базою даних та забезпечення безпечної інтеграції із зовнішніми системами, зокрема платіжними шлюзами та логістичними операторами. Для обміну даними між клієнтом, сервером та сторонніми сервісами використовується інтерфейс прикладного програмування (API). У сучасній веб-розробці найпоширенішим є REST-підхід (Representational State Transfer), який дозволяє стандартизовано працювати з інформаційними ресурсами системи, такими як товари, замовлення чи профілі користувачів, за допомогою універсальних мережевих HTTP-запитів.

У предметній області fashion-ритейлу центральним є поняття товару як базової одиниці каталогу. Товар містить загальну інформацію: назву, категоріальну приналежність, текстовий опис, базову ціну та головне зображення. Однак специфіка продажу одягу вимагає значно вищого рівня деталізації даних, тому в архітектуру системи обов'язково вводиться поняття товарної варіації, або SKU (Stock Keeping Unit) [24]. Це конкретне фізичне представлення речі з індивідуальними характеристиками, насамперед точним розміром та кольором. Інтернет-магазин повинен відстежувати складські залишки не абстрактної футболки загалом, а конкретного варіанту певного кольору та розміру. Надзвичайно важливою вимогою є динамічна прив'язка медіаконтенту до конкретних варіацій. Коли користувач змінює колір товару на сторінці, клієнтський інтерфейс має миттєво відображати відповідну фотографію, що суттєво підвищує інформативність каталогу та зменшує вірогідність помилкового вибору.

Проміжним, але критично важливим етапом між переглядом каталогу і

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

безпосереднім оформленням замовлення є робота з віртуальним кошиком. Він виконує функцію тимчасового сховища, яке об'єднує вибрані користувачем позиції та зберігає їхню кількість, актуальну ціну і специфічні атрибути. Якщо покупець додає до кошика товар певного кольору та розміру, системі необхідно точно зафіксувати саме цей ідентифікатор варіації. Це гарантує безпомилкове збереження вибору клієнта та підвищує точність подальшого комплектування замовлення персоналом магазину. Завершальним етапом процесу покупки є оформлення замовлення (checkout). Замовлення являє собою фінальну сутність у базі даних, яка містить консолідовану інформацію: персональні дані покупця, перелік обраних варіацій товарів із зафіксованою на момент покупки ціною, загальну суму до сплати, обрані методи доставки та оплати, а також поточний статус виконання, який може змінюватися в міру обробки. Зазвичай життєвий цикл замовлення проходить через статуси нового, очікуючого на оплату, оплаченого, переданого в службу доставки та остаточно виконаного.

Для забезпечення безпеки та ефективного управління даними система передбачає суворе розмежування прав доступу на основі ролей. Неавторизований користувач, або гість, має змогу переглядати каталог товарів та формувати кошик без жодних обмежень. Авторизований покупець, після проходження процедури реєстрації та входу до системи, отримує додатковий доступ до персонального кабінету, історії попередніх покупок і збережених контактних даних для прискореного оформлення майбутніх замовлень. Адміністратор системи наділяється найвищим рівнем доступу та працює у захищеній адміністративній панелі. Цей інструмент призначений для повноцінного управління роботою платформи: додавання та редагування товарних позицій, налаштування розмірних сіток, завантаження зображень, моніторингу замовлень, зміни їх статусів та контролю фінансових надходжень.

З погляду теорії програмної інженерії, створення такої комплексної системи охоплює повний життєвий цикл розробки програмного забезпечення,

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

що включає етапи аналізу вимог, проєктування архітектури, написання коду, тестування, розгортання та подальшого супроводу. Згідно з міжнародними стандартами якості, розроблювана система повинна відповідати не лише заявленим функціональним вимогам, але й суворим нефункціональним характеристикам. Серед них ключову роль відіграють висока продуктивність під час пікових навантажень, гарантована інформаційна безпека транзакцій, інтуїтивна зрозумілість інтерфейсу для кінцевого споживача та модульність коду, що забезпечує простоту подальшого масштабування проєкту.

У предметній області fashion-ритейлу центральним є поняття товару як базової одиниці каталогу. Товар містить загальну інформацію: назву, категоріальну приналежність, текстовий опис, базову ціну та головне зображення. Однак специфіка продажу одягу вимагає значно вищого рівня деталізації даних, тому в архітектуру системи обов'язково вводиться поняття товарної варіації, або SKU (Stock Keeping Unit). Це конкретне фізичне представлення речі з індивідуальними характеристиками, насамперед точним розміром та кольором. Інтернет-магазин повинен відстежувати складські залишки не абстрактної футболки загалом, а конкретного варіанту певного кольору та розміру.

1.3. Сучасні технології та тенденції розробки сервіс-орієнтованих застосунків

Сучасні сервісні веб-застосунки розробляються з урахуванням кількох ключових тенденцій: використання надійних веб-фреймворків, API-орієнтованої архітектури, безпечних систем управління реляційними базами даних, контейнеризації та глибокої інтеграції із зовнішніми спеціалізованими сервісами. У сфері електронної комерції, зокрема для інтернет-магазинів одягу, ці напрями мають визначальне значення. Висока конкуренція на ринку вимагає від програмних рішень не лише стабільної роботи при високих навантаженнях, але й можливості швидкого впровадження нових функцій,

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

адаптивності до змін бізнес-вимог та гарантування високого рівня безпеки персональних даних користувачів.

Для реалізації як серверної, так і клієнтської логіки розроблюваного інтернет-магазину оптимальним вибором є веб-фреймворк Ruby on Rails. Цей інструмент базується на класичному архітектурному патерні MVC (Model-View-Controller), який забезпечує чітке розділення відповідальності між компонентами системи [12]. Модель відповідає за взаємодію з даними та інкапсулює бізнес-логіку, представлення формує користувацький інтерфейс, а контролер приймає запити від клієнта і керує потоком даних між моделлю та представленням. Фреймворк дотримується принципу «Convention over Configuration» (угода дорожче за конфігурацію), що дозволяє розробникам зосередитися на унікальній логіці інтернет-магазину замість написання великої кількості шаблонного коду. Крім того, вбудований інструмент об'єктно-реляційного відображення (ORM) Active Record суттєво спрощує взаємодію з базою даних, абстрагуючи складні SQL-запити у зрозумілі об'єктні структури.

Зберігання інформації є критичним аспектом для будь-якої системи електронної комерції. Використання надійної реляційної системи управління базами даних, такої як PostgreSQL, є обґрунтованим вибором для промислових рішень. PostgreSQL підтримує складні аналітичні запити, забезпечує безкомпромісну цілісність даних через механізми зовнішніх ключів та транзакцій, що повністю відповідає принципам ACID (атомарність, узгодженість, ізолюваність, довговічність). Це є обов'язковою вимогою при роботі з фінансовими операціями, зміною статусів замовлень та строгим контролем залишків конкретних варіантів товарів, таких як розміри та кольори одягу. Додатковою перевагою PostgreSQL є розширена підтримка формату JSONB, що дозволяє гнучко зберігати специфічні неструктуровані атрибути товарів без необхідності постійної зміни жорсткої схеми бази даних.

Важливою тенденцією є перехід до API-орієнтованого проектування та елементів сервіс-орієнтованої архітектури (SOA). У сучасних умовах веб-

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

застосунок не повинен реалізовувати абсолютно всі функції автономно. Натомість він делегує вузькоспеціалізовані завдання зовнішнім сервісам через протоколи взаємодії, переважно використовуючи REST API. Наприклад, для обробки платежів інтегрується сервіс інтернет-еквайрингу від Monobank. Логіка такої взаємодії передбачає формування платіжних параметрів безпосередньо на захищеному сервері з використанням секретних ключів, що унеможлиблює їх компрометацію через клієнтський браузер. Після успішної транзакції платіжний шлюз надсилає зворотний запит (webhook) на сервер магазину для автоматичного підтвердження та оновлення статусу замовлення.

Для організації логістики використовується інтеграція з Nova Post API. Це дає змогу отримувати актуальні дані про географію доставки, типи відділень та поштоматів динамічно, безпосередньо під час оформлення замовлення, що значно покращує загальний користувацький досвід та мінімізує помилки при введенні адреси. Окремої уваги заслуговує тенденція використання зовнішніх провайдерів для ідентифікації користувачів. Інтеграція протоколу OAuth за допомогою спеціалізованих бібліотек дозволяє реалізувати безпечний вхід через облікові записи Google, що значно знижує бар'єр реєстрації для нових покупців та підвищує загальну конверсію інтернет-магазину.

Ще однією фундаментальною тенденцією розробки є обов'язкова контейнеризація застосунків. Використання інструментів Docker та Docker Compose дозволяє описувати й запускати багатоконтейнерні застосунки за допомогою єдиного конфігураційного файлу. У такому ізольованому середовищі сам веб-застосунок на Ruby on Rails, база даних PostgreSQL та інші допоміжні служби запускаються як окремі, але тісно пов'язані між собою контейнери. Такий підхід ефективно розв'язує проблему сумісності програмних середовищ, оскільки гарантує, що написаний програмний код працюватиме абсолютно однаково як на локальному комп'ютері розробника під час етапу створення та тестування, так і на віддаленому сервері після остаточного розгортання. Це також закладає міцний фундамент для

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

майбутнього використання сучасних практик безперервної інтеграції та розгортання (CI/CD).

Окрім базової архітектури та реляційних баз даних, сучасний підхід до розробки систем електронної комерції вимагає обов'язкового впровадження механізмів асинхронної обробки задач. У контексті інтернет-магазину одягу виникає безліч процесів, які фізично не повинні блокувати основний потік виконання веб-запиту. Наприклад, коли клієнт фіналізує оформлення замовлення, системі потрібно виконати цілий каскад дій: надіслати транзакційний email-лист із підтвердженням, згенерувати платіжне посилання через виклик API платіжного шлюзу та оновити резерв товарів на складі. Якщо виконувати всі ці операції синхронно в рамках одного циклу, покупець буде дивитися на екран завантаження кілька зайвих секунд. У реаліях мобільної комерції це майже гарантовано призводить до подвійних кліків по кнопці оплати або взагалі до закриття вкладки. Тому галузевим стандартом у розробці є використання брокерів повідомлень та інструментів фонові обробки. У екосистемі Ruby середовища цю роль найчастіше виконує інструмент Sidekiq у зв'язці з in-memory сховищем Redis.

1.4. Висновки по розділу

У першому розділі було проведено комплексний аналіз ринку електронної комерції у сфері продажу одягу та розглянуто теоретичні основи проєктування сучасних інтернет-магазинів. Встановлено, що специфіка fashion-ритейлу вимагає від веб-застосунків гнучкості, високої швидкості роботи та безперебійної підтримки складних атрибутів товарів, таких як розміри, кольори та відповідні їм зображення. Аналіз існуючих рішень показав, що використання універсальних маркетплейсів або шаблонних CMS-систем часто обмежує можливості кастомізації бізнес-процесів. З огляду на це, розробка власного спеціалізованого сервіс-орієнтованого застосунку є

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

найбільш ефективним підходом для створення унікального та безшовного користувацького досвіду.

Під час дослідження предметної області було визначено ключові архітектурні та бізнес-сутності системи: товар, варіант товару (SKU), кошик, замовлення та ролі користувачів. Доведено, що для забезпечення безпомилкового оформлення покупок інтернет-магазин повинен чітко фіксувати конкретні параметри обраної речі на всіх етапах взаємодії клієнта з платформою. Крім того, визначено, що розмежування прав доступу між гостем, авторизованим покупцем та адміністратором є необхідною умовою для безпечного та ефективного управління даними магазину. Огляд сучасних технологій підтвердив доцільність вибору веб-фреймворку Ruby on Rails та реляційної системи управління базами даних PostgreSQL як надійного фундаменту для практичної реалізації проєкту. Використання архітектурного патерну MVC, дотримання принципів взаємодії через REST API для глибокої інтеграції із зовнішніми логістичними (Nova Post) та платіжними (Monobank) сервісами, а також контейнеризація середовища за допомогою Docker забезпечують створення масштабованої, безпечної та зручної у супроводі системи.

Окрему увагу в ході аналізу було приділено моделям якості програмного забезпечення згідно зі стандартом ISO/IEC 25010, що дозволило визначити пріоритетні критерії надійності та супроводжуваності платформи. Також обґрунтовано необхідність інтеграції інструментів фонові обробки задач Sidekiq та in-memory сховища Redis для мінімізації затримок при взаємодії користувача з інтерфейсом. Урахування вимог вітчизняного законодавства щодо захисту персональних даних забезпечило формування надійного безпекового контуру для збереження конфіденційної інформації клієнтів. Завдяки комплексному вивченню сучасних галузевих досліджень вдалося закласти теоретичний фундамент для зниження показників покинутих кошиків та оптимізації загального клієнтського досвіду.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1. Аналіз вимог до веб-застосунку інтернет-магазину одягу

Етап аналізу та формування вимог є фундаментальною стадією життєвого циклу розробки будь-якого програмного забезпечення. Згідно з принципами програмної інженерії, успішність реалізації проєкту безпосередньо залежить від того, наскільки повно, точно та несутеречно визначені потреби зацікавлених сторін (стейкхолдерів). У контексті розробки веб-застосунку для інтернет-магазину одягу основними зацікавленими сторонами виступають кінцеві покупці, які взаємодіють із клієнтським інтерфейсом, та адміністратори платформи, які здійснюють управління контентом і бізнес-процесами через серверну частину. Аналіз вимог до такої системи вимагає врахування не лише стандартних сценаріїв електронної комерції, але й специфіки сервіс-орієнтованої архітектури (SOA), де частина критично важливого функціоналу делегується зовнішнім провайдерам через програмні інтерфейси.

З погляду кінцевого споживача, ключовою вимогою до інтернет-магазину моди є забезпечення максимально зручного, швидкого та інтуїтивно зрозумілого процесу вибору та купівлі товарів. Специфіка продажу одягу диктує необхідність реалізації складної структури каталогу, де кожна модель може мати десятки варіацій залежно від розміру та кольору [6]. Відповідно, система повинна надавати користувачеві можливість зручної фільтрації за цими параметрами. Критичною вимогою є забезпечення миттєвого візуального зворотного зв'язку: при зміні кольору одягу на сторінці товару інтерфейс має динамічно оновлювати відповідне зображення, що значно покращує користувацький досвід та знижує ймовірність помилкового вибору. Крім того, процес формування кошика та подальшого оформлення замовлення (checkout) має бути безшовним, вимагаючи від клієнта мінімальної кількості кроків та обов'язкових полів для заповнення. Враховуючи тенденції сучасного

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		21

ринку, застосунок повинен підтримувати як гостьове оформлення замовлення, так і повноцінну авторизацію, яка дозволить покупцям зберігати історію покупок та контактні дані для прискорення майбутніх транзакцій.

З боку управління бізнес-процесами, адміністративна панель веб-застосунку повинна задовольняти вимоги щодо гнучкого налаштування каталогу та оперативного опрацювання замовлень. Адміністратор повинен мати інструменти для створення нових товарних позицій, завантаження медіаконтенту, управління ціноутворенням та, що найважливіше, контролю залишків конкретних варіацій одягу (SKU) [20]. Вимоги до підсистеми управління замовленнями передбачають наявність єдиного робочого середовища, де менеджер може переглядати всі деталі транзакції: контактні дані клієнта, обрані товари, їх специфічні атрибути, загальну суму, а також поточний статус оплати та доставки. Система має підтримувати життєвий цикл замовлення через зміну його статусів, що дозволить автоматизувати контроль за виконанням угод.

Окремий блок вимог формується на основі обраної сервіс-орієнтованої архітектури, яка передбачає глибоку інтеграцію з локальними українськими сервісами. Система не повинна самотійно обробляти та зберігати конфіденційні дані банківських карток, оскільки це вимагає проходження складної сертифікації безпеки. Натомість висувається вимога інтеграції надійного платіжного шлюзу (інтернет-еквайрингу) від Monobank, який візьме на себе відповідальність за безпечне проведення транзакцій та інформування сервера про успішну оплату за допомогою технології вебхуків (webhooks). У сфері логістики система повинна відмовитися від використання статичних баз даних адрес на користь динамічної взаємодії з API компанії «Нова Пошта». [20] Це вимагає реалізації функціоналу, який дозволить користувачеві під час оформлення замовлення в реальному часі отримувати актуальні списки міст та доступних відділень, що гарантує точність доставки та зменшує навантаження на адміністраторів платформи.

Загалом, результати аналізу предметної області та потреб

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

стейкхолдерів свідчать про те, що розроблюваний веб-застосунок має бути не просто електронним каталогом, а комплексною системою з чітким розподілом бізнес-логіки. Вона повинна об'єднувати власну базу даних для управління контентом одягу, оптимізований клієнтський інтерфейс та надійні механізми взаємодії із зовнішніми сервісами, що стане основою для формування детального переліку функціональних та нефункціональних вимог на наступному етапі проєктування.

2.2. Формулювання функціональних та нефункціональних вимог

Для чіткої формалізації завдань розробки веб-застосунку вимоги було розподілено на дві категорії: функціональні (що система має робити) та нефункціональні (яким технічним критеріям якості вона повинна відповідати). У таблиці 2.1 наведено основні функціональні вимоги, розбиті за модулями системи.

Таблиця 2.1

Функціональні вимоги до веб-застосунку інтернет-магазину

ID	Модуль / Підсистема	Опис вимоги	Актор
FR-01	Авторизація	Реєстрація та вхід у систему через email та пароль.	Покупець, Адмін
FR-02	Каталог товарів	Відображення товарів із фільтрацією за розміром та кольором.	Гість, Покупець
FR-03	Кошик	Додавання товарів, зміна кількості та автоперерахунок суми.	Гість, Покупець
FR-04	Інтеграція «Нова Пошта»	Автоматичне завантаження міст та відділень під час оформлення.	Гість, Покупець
FR-05	Інтеграція Monobank	Генерація посилання на оплату та обробка асинхронних вебхуків.	Покупець, Система
FR-06	Складський облік	Автоматичне списання залишків товару (SKU) після успішної оплати.	Система

FR-07	Адмін-панель	Керування каталогом товарів, зміна статусів замовлень та ТТН.	Адміністратор
--------------	--------------	---	---------------

Функціональні вимоги визначають поведінку системи у відповідь на дії користувачів або зовнішніх сервісів. Для розробляемого інтернет-магазину одягу на базі фреймворку Ruby on Rails було виділено та деталізовано кілька ключових функціональних підсистем:

Підсистема автентифікації, реєстрації та керування сесіями користувачів. Цей модуль є першим контуром взаємодії користувача з платформою. Система повинна забезпечувати можливість створення облікового запису для нових гостей за допомогою унікальної адреси електронної пошти та надійного пароля. На рівні бізнес-логіки бекенду обов'язково має виконуватися валідація на унікальність email (реєстрація двох користувачів на одну пошту блокується) та перевірка формату введення. Для авторизованих покупців система повинна підтримувати механізм безпечного збереження сесій, можливість виходу з аккаунта, а також функціонал відновлення доступу до облікового запису через надсилання унікального токена скидання пароля на електронну пошту.

Підсистема інтерактивного каталогу товарів та картки SKU. Специфіка fashion-ритейлу вимагає від каталогу високої гнучкості. Система повинна динамічно виводити перелік товарів із підтримкою механізму пагінації (розбиття на сторінки), щоб не перевантажувати клієнтську частину великими обсягами даних. Користувачу має бути доступне сортування за ціною (зростання/спадання) та новизною. Головною інженерною вимогою до картки товару є підтримка сутності SKU (Stock Keeping Unit). Коли користувач обирає конкретний розмір та колір, система повинна перевіряти наявність цієї комбінації на складі та асинхронно (без перезавантаження сторінки сайту) оновлювати фотогалерею, виводячи зображення одягу саме у вибраному кольорі.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

Підсистема віртуального кошика та управління позиціями. Кошик має функціонувати як для авторизованих користувачів, так і для анонімних гостей (через збереження ідентифікатора сесії в cookie або сховищі Redis). Функціональні вимоги передбачають можливість додавання обраного SKU до кошика, редагування кількості одиниць товарів безпосередньо у вікні перегляду, а також повне видалення непотрібних позицій. Будь-яка маніпуляція з вмістом кошика повинна супроводжуватися миттєвим асинхронним перерахунком проміжної вартості кожної позиції та загальної фінальної суми замовлення.

Модуль інтеграції з логістичним API «Нова Пошта». Під час оформлення покупки (checkout) застосунок повинен взаємодіяти із зовнішнім сервісом логістики. Система має реалізовувати асинхронні POST-запити до прикладного інтерфейсу оператора для динамічного підвантаження офіційних географічних довідників. Коли користувач починає вводити назву міста, система пропонує варіанти автоматичного заповнення, а після вибору населеного пункту — динамічно завантажує лише ті відділення або поштомати, які функціонують у цьому місті на поточний момент. Ручне введення неіснуючих адрес має блокуватися.

Модуль інтеграції з платіжним API Monobank та обробка вебхуків. При виборі безготівкового розрахунку система повинна звертатися до платіжного шлюзу банку, передаючи унікальний ID замовлення, перелік товарів та суму в копійках. У відповідь застосунок отримує платіжну сторінку та редіректить туди користувача. Критично важливою функціональною вимогою є розробка контролера вебхуків (webhooks): сервер банку після успішної оплати надсилає асинхронний запит на спеціальний URL нашого застосунку. Система повинна перехопити цей запит, валідувати його криптографічний підпис, автоматично перевести замовлення у статус «Оплачено» та ініціювати Sidekiq-воркер для списання одиниць SKU зі складу. Це повністю ліквідує ризик оверселлінгу.

Адміністративна панель керування контентом та угодами. Для користувачів із роллю «Адміністратор» система повинна надавати виділений

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						25
Змн.	Арк.	№ докум.	Підпис	Дата		

інтерфейс для управління магазином. Функціонал включає: створення нових категорій, додавання товарів, генерацію варіацій (розмірів/кольорів), встановлення цін та завантаження медіафайлів. У модулі управління замовленнями адмін повинен мати змогу моніторити всі транзакції, фільтрувати їх за статусом оплати та логістики, а також в один клік генерувати товарно-транспортні накладні (ТТН) на основі раніше збережених через API даних.

Для систематизації та наочного представлення всі описані функціональні вимоги каталогізовано у таблиці 2.1 за допомогою унікальних ідентифікаторів (ID).

Нефункціональні вимоги визначають критерії якості програмного продукту, які гарантують його надійність, безпеку та продуктивність у промислових умовах. Сформульовані нефункціональні вимоги структуровані відповідно до міжнародного стандарту якості ISO/IEC 25010 із визначенням чітких інженерних метрик та способів їхньої верифікації (перевірки). Детальні дані наведено в таблиці 2.2.

Таблиця 2.2

Нефункціональні вимоги до веб-застосунку за стандартом

ID	Критерій якості	Опис вимоги	Спосіб перевірки
NFR-01	Швидкодія	Час відгуку сервера при завантаженні сторінок — до 250 мс.	Тестування через k6
NFR-02	Безпека даних	Хешування паролів (Bcrypt) та захист від CSRF-атак.	Вбудовані інструменти Rails
NFR-03	Надійність	Збереження цілісності даних при збоях під час оплати.	ACID-транзакції PostgreSQL
NFR-04	Конфіденційність	Приховування токенів та API-ключів від публічного доступу.	Змінні оточення (.env)

NFR-05	Ергономічність (UX)	Коректне відображення інтерфейсу на мобільних телефонах.	Адаптивна верстка (CSS)
NFR-06	Портативність	Запуск усього технологічного стека в ізольованому середовищі.	Контейнеризація Docker

Нефункціональні вимоги (NFR) визначають архітектурні обмеження, критерії якості, надійності та безпеки, яким повинен відповідати веб-застосунок для стабільної роботи в промисловому середовищі. Для забезпечення високої інженерної якості розробки, нефункціональні вимоги структуровані відповідно до міжнародного стандарту ISO/IEC 25010.

Ефективність продуктивності (Performance Efficiency). Веб-застосунок електронної комерції повинен демонструвати високу швидкість відгуку, оскільки затримки безпосередньо впливають на конверсію сайту. Час генерації сторінки сервером (Response Time) для стандартних HTTP GET-запитів при завантаженні головного каталогу не повинен перевищувати 250 мілісекунд під навантаженням. З погляду місткості (Capacity), архітектура системи (комбінація веб-сервера Puma та пулу з'єднань PostgreSQL) має стабільно витримувати щільність потоку в межах 250–300 запитів за секунду (RPS) без деградації продуктивності сховища даних та появи помилок тайм-ауту.

Інформаційна безпека (Security). Оскільки система обробляє конфіденційні персональні дані покупців та взаємодіє з фінансовими шлюзами, вимоги безпеки є пріоритетними. Зберігання паролів у базі даних у відкритому вигляді суворо заборонено — автентифікаційний модуль повинен використовувати алгоритм криптографічного хешування Bcrypt із високим показником обчислювальної складності (випадкова сіль). Для захисту від атак типу CSRF (підробка міжсайтових запитів) усі деструктивні HTTP-методи (POST, PUT, DELETE) мають супроводжуватися обов'язковою валідацією унікального authenticity_token. Крім того, архітектурні секрети (токени Monobank, API-ключі логістики) мають бути повністю ізольовані від

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

вихідного коду і зчитуватися виключно із зашифрованих змінних оточення сервера системи.

Надійність та відмовостійкість (Reliability). Система повинна забезпечувати високий коефіцієнт доступності. У разі виникнення позаштатних ситуацій (наприклад, раптовий мережевий збій під час обміну даними з API банку в момент оплати), веб-застосунок повинен гарантувати реляційну цілісність інформації. Це реалізується через механізм ACID-транзакцій СУБД PostgreSQL: якщо крок оновлення статусу замовлення або списання товару завершується помилкою, система зобов'язана виконати повний відкат змін (rollback) до початкового стабільного стану.

Зручність використання та ергономічність (Usability). Користувацький інтерфейс має бути інтуїтивно зрозумілим і не вимагати від покупця спеціального навчання. Обов'язковою вимогою є адаптивність графічного інтерфейсу (Responsive Web Design). Табличні та блочні структури сайту повинні автоматично підлаштовуватися під розширення екранів мобільних пристроїв та планшетів, зберігаючи при цьому високу швидкість завантаження контенту (зображень товарів) завдяки використанню сучасних форматів стиснення (WebP).

Супроводжуваність та портативність коду (Maintainability & Portability). Вихідний програмний код бекенду має бути модульним, легко читатися та супроводжуватися іншими розробниками. Для цього конфігурація коду повинна відповідати офіційним гайдлайнам стилю (Ruby Style Guide), що контролюється утилітами статичного аналізу (RuboCop). З погляду переносимості, застосунок має бути повністю контейнеризованим за допомогою Docker. Це гарантує, що система разом з усіма версіями бібліотек та СУБД зможе безперешкодно розгорнутися на будь-якому промисловому Linux-сервері без необхідності ручного налаштування залежностей операційної системи хоста.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3. Постановка задачі проектування системи

На основі сформульованих функціональних та нефункціональних вимог, головною метою інженерного проєкту є створення комплексної, масштабованої та відмовостійкої веб-платформи інтернет-магазину одягу на базі сервіс-орієнтованої архітектури.

З погляду системного аналізу та інженерії програмного забезпечення, формальна постановка задачі проєктування полягає у відображенні множини вхідних функціональних вимог та обмежень на множину конкретних архітектурних артефактів, логічних схем та програмних компонентів. Вхідними даними для проєктування є специфіковані у підрозділі 2.2 параметри якості (FR-01–FR-08, NFR-01–NFR-08), технічні ліміти обраного технологічного стека, а також сучасні галузеві стандарти безпеки веб-застосунків (зокрема, вимоги консорціуму OWASP Top 10 щодо захисту від ін'єкцій та витоку даних).

При цьому задача проєктування накладає суворі структурні обмеження: архітектура системи повинна мінімізувати рівень взаємозв'язку (coupling) між окремими сервісними модулями та водночас забезпечувати високий рівень внутрішньої зв'язності (cohesion) всередині них. Це вимагає декомпозиції складної бізнес-логіки інтернет-магазину (такої як паралельна робота з різними модифікаціями SKU, асинхронні транзакції оплат та логістичний менеджмент відправок) на рівні окремих, ізольованих шарів абстракції. Для реалізації цих цілей процес проєктування системи передбачає розв'язання таких взаємопов'язаних підсистемних задач:

Проєктування реляційної бази даних (до вимог FR-02, FR-06, NFR-03): Створення структури нормалізованої бази даних під управлінням СУБД PostgreSQL для надійного збереження даних користувачів, замовлень та обліку складських залишків конкретних товарних варіацій (SKU) у реальному часі.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

Розробка серверної бізнес-логіки (до вимог FR-01, FR-03, NFR-02): Реалізація функціональних компонентів бекенду за шаблоном MVC на базі фреймворку Ruby on Rails для забезпечення безпечного керування сесіями, роботи віртуального кошика та безпомилкового розрахунку вартості замовлень.

Проектування інтеграційних модулів API (до вимог FR-04, FR-05, NFR-04): Створення програмних адаптерів для динамічної взаємодії з логістичним API «Нова Пошта» та платіжним API Monobank, включаючи розробку захищеного алгоритму обробки асинхронних вебхуків від банківської системи.

Реалізація інтерфейсу керування (до вимоги FR-07): Розробка захищеної багатофункціональної панелі адміністратора з рольовим розмежуванням доступу для контент-менеджменту та оперативного опрацювання угод.

Розробка користувацького інтерфейсу (до вимоги NFR-05): Створення інтуїтивно зрозумілого фронтенду з адаптивною версткою для забезпечення високої швидкості взаємодії покупця з платформою та зручного оформлення покупок із будь-яких типів пристроїв.

Інфраструктурне забезпечення (до вимоги NFR-06): Пакування веб-застосунку та його системних залежностей в ізольоване контейнеризоване середовище Docker для гарантування ідентичності середовищ розробки та промислової експлуатації.

Критеріями успішності та коректності розв'язання поставленої інженерної задачі є досягнення цільових метрик ефективності. Спроектована логічна схема бази даних повинна суворо відповідати вимогам третьої нормальної форми (3NF) для повного унеможливлення надлишковості та аномалій модифікації даних. Інфраструктурне рішення має гарантувати повну ізоляцію контейнерів СУБД від зовнішньої мережі Інтернет, а інтеграційні модулі — демонструвати транзакційну стійкість до можливих мережеских затримок чи відмов сторонніх серверів.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

2.4. Висновки до розділу

У другому розділі було здійснено комплексний аналіз вимог та виконано детальну інженерну постановку задачі на розробку сервіс-орієнтованого веб-застосунку інтернет-магазину одягу. На основі дослідження потреб ключових стейкхолдерів — кінцевих покупців та менеджерів платформи — встановлено, що система повинна гарантувати безшовний користувацький досвід, надавати гнучкі інструменти для управління контентом та забезпечувати безкомпромісну надійність фінансових транзакцій. Детальне формулювання функціональних вимог дозволило чітко окреслити межі відповідальності розроблюваного програмного продукту: реалізація динамічного каталогу з підтримкою складних товарних варіацій (SKU), створення оптимізованого механізму оформлення замовлення, розробка багатофункціональної панелі управління та обов'язкова інтеграція із зовнішніми інфраструктурними сервісами.

Своєю чергою, визначення нефункціональних вимог заклало високі стандарти якості майбутньої платформи, закріпивши необхідність дотримання високого рівня продуктивності під навантаженням, безпеки збереження персональних даних, кросбраузерної адаптивності клієнтського інтерфейсу та супроводжуваності програмного коду [22]. У результаті проведеного аналізу була сформована чітка постановка задачі проєктування, яка зводиться до побудови нормалізованої реляційної бази даних, розробки серверної частини на базі фреймворку Ruby on Rails з використанням архітектурного шаблону MVC, а також налаштування ізольованого середовища розгортання за допомогою технології контейнеризації Docker. Таким чином, результати дослідження, наведені у другому розділі, формують вичерпне технічне завдання та створюють надійне науково-практичне підґрунтя для переходу до наступного етапу — безпосереднього проєктування архітектури програмного забезпечення та моделювання його компонентів.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ

3.1. Розробка архітектури програмного забезпечення

Розробка архітектури програмного забезпечення є критично важливим етапом життєвого циклу проекту, який визначає фундаментальну організацію системи, її базові компоненти, їх взаємозв'язки та загальні принципи функціонування. Для розроблюваного інтернет-магазину одягу була обрана сервіс-орієнтована архітектура (SOA) з використанням потужного монолітного ядра на базі веб-фреймворку Ruby on Rails. Такий гібридний підхід дозволяє поєднати високу швидкість розробки та зручність підтримки єдиної кодової бази з гнучкістю делегування складних, вузькоспеціалізованих завдань зовнішнім сервісам через протоколи мережевої взаємодії [4]. Ядро системи повністю відповідає за обробку внутрішньої бізнес-логіки: управління товарним каталогом, збереження сесій, контроль вмісту кошика та життєвий цикл замовлень. Водночас транзакційні фінансові операції та складні логістичні розрахунки свідомо винесені за межі основного сервера і здійснюються через захищені виклики до прикладних програмних інтерфейсів (API) відповідних провайдерів.

Внутрішня архітектура базового застосунку побудована згідно з класичним паттерном проектування Model-View-Controller (MVC), який забезпечує суворе розділення відповідальності між обробкою даних, їх візуалізацією та управлінням потоками виконання. Компонент Моделі (Model) інкапсулює ключову предметну логіку та відповідає за безпосередню взаємодію з системою управління реляційними базами даних PostgreSQL. У середовищі Ruby on Rails ця взаємодія реалізується за допомогою потужного вбудованого механізму об'єктно-реляційного відображення (ORM) під назвою Active Record [7]. Він абстрагує та перетворює складні SQL-запити на інтуїтивно зрозумілі об'єктні структури, гарантуючи при цьому цілісність даних, обробку складних зв'язків між сутностями та сувору валідацію

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

атрибутів перед їх збереженням у базу. Представлення (View) відповідає виключно за генерацію клієнтського інтерфейсу на основі даних, отриманих від контролера. Воно реалізовано з використанням шаблонізатора ERB, який дозволяє динамічно формувати HTML-сторінки, інтегруючи у них актуальні дані про товари, доступні розмірні сітки, кольори та поточний стан кошика користувача. Контролер (Controller) виступає центральною сполучною ланкою, яка приймає вхідні HTTP-запити від клієнтського браузера, здійснює маршрутизацію, ініціює необхідні транзакції у моделях та передає результати їх виконання до відповідного представлення для подальшого відображення покупцеві.

Розширення архітектури до рівня сервіс-орієнтованої досягається шляхом впровадження в структуру проекту спеціалізованих сервісних об'єктів (Service Objects), які виконують роль адаптерів для зовнішньої комунікації. Зокрема, для реалізації процесу онлайн-оплати спроектовано платіжний сервіс, який встановлює захищене з'єднання з API Monobank, формує криптографічно підписаний запит на створення транзакції та автономно обробляє асинхронні відповіді (вебхуки) для автоматичної зміни статусу замовлення в базі даних магазину. Аналогічним чином функціонує логістичний модуль, який здійснює синхронні REST-запити до серверів Нової Пошти для динамічного отримання актуальних довідників населених пунктів та відділень безпосередньо під час заповнення користувачем форми оформлення покупки. Такий архітектурний розподіл зменшує зв'язність коду і гарантує, що можливі збої на стороні зовнішніх інфраструктурних провайдерів не призведуть до критичної зупинки всього інтернет-магазину [13].

Фізична архітектура розгортання спроектованої системи базується на сучасних принципах контейнеризації з використанням технології Docker. Застосунок логічно розділено на кілька незалежних контейнерів, які взаємодіють між собою виключно через ізольовану віртуальну мережу. Основними вузлами цієї інфраструктури є контейнер з веб-сервером на базі середовища Ruby, який безпосередньо обробляє вхідні запити клієнтів, та

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

окремий контейнер з базою даних PostgreSQL, що забезпечує надійне та ізольоване збереження транзакційної інформації. Використання інструментарію Docker Compose дозволяє описувати всі інфраструктурні залежності та змінні середовища у вигляді єдиного конфігураційного файлу. Це не лише кардинально спрощує процес локальної розробки та тестування системи, але й гарантує абсолютну ідентичність середовищ виконання, роблячи архітектуру проєкту максимально стійкою, готовою до горизонтального масштабування та впровадження практик безперервного розгортання.

3.2. Моделювання бізнес-процесів та системних компонентів

Моделювання бізнес-процесів є невіддільною складовою проєктування складних програмних систем, оскільки воно дозволяє візуалізувати динаміку взаємодії користувача із застосунком та алгоритми обробки даних на стороні сервера. Для опису логіки функціонування інтернет-магазину одягу доцільно використовувати методологію об'єктно-орієнтованого аналізу та графічну мову моделювання UML (Unified Modeling Language), зокрема діаграми діяльності (Activity Diagrams). Основними бізнес-процесами розроблюваної платформи є навігація товарним каталогом із вибором конкретних варіацій одягу, управління вмістом віртуального кошика, багатоетапне оформлення замовлення (checkout) та подальше опрацювання створеної угоди в адміністративній панелі.

Початковим етапом взаємодії є робота користувача з каталогом. Специфіка fashion-індустрії вимагає, щоб процес вибору обов'язково включав визначення конкретного ідентифікатора товарної варіації (SKU). Коли користувач обирає певну модель одягу, система зобов'язує його вказати точний розмір та колір, динамічно підвантажуючи відповідні фотографії. Після ініціації дії додавання до кошика, контролер застосунку перевіряє наявність обраного варіанту на складі та створює тимчасовий запис у сесії

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

користувача або в базі даних (для авторизованих клієнтів) [27]. Цей процес гарантує, що подальше формування замовлення базуватиметься на абсолютно точних даних щодо обраної речі, виключаючи можливість замовлення відсутнього товару.

Найбільш критичним та алгоритмічно складним бізнес-процесом у системі є процедура оформлення замовлення, яка передбачає тісну взаємодію із зовнішніми API. Цей процес ініціюється переходом користувача з кошика на сторінку оформлення. На першому кроці система збирає контактні дані покупця. Далі відбувається етап вибору способу доставки, під час якого серверна частина інтернет-магазину виконує синхронний запит до сервісу «Нова Пошта», отримуючи актуальний перелік міст та відділень. Після підтвердження адреси доставки система переходить до етапу оплати. Застосунок формує загальну суму, генерує унікальне посилання за допомогою API Monobank та перенаправляє клієнта на безпечну платіжну сторінку [8]. У цей момент замовлення зберігається в базі даних інтернет-магазину зі статусом очікування оплати. Процес автоматично завершується успіхом лише тоді, коли сервер отримує захищений зворотний виклик (вебхук) від банку, після чого статус замовлення змінюється на оплачене.

Після успішної оплати ініціюється бізнес-процес опрацювання замовлення на стороні адміністратора. Через захищену панель управління менеджер отримує сповіщення про нову транзакцію, перевіряє коректність зібраних даних та ініціює фізичне пакування товару [9]. Життєвий цикл замовлення супроводжується послідовною зміною статусів, що дозволяє системі автоматично інформувати покупця про етапи виконання угоди. Детальну візуалізацію ключового бізнес-процесу — алгоритму оформлення та оплати замовлення клієнтом — наведено на рисунку 3.2.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

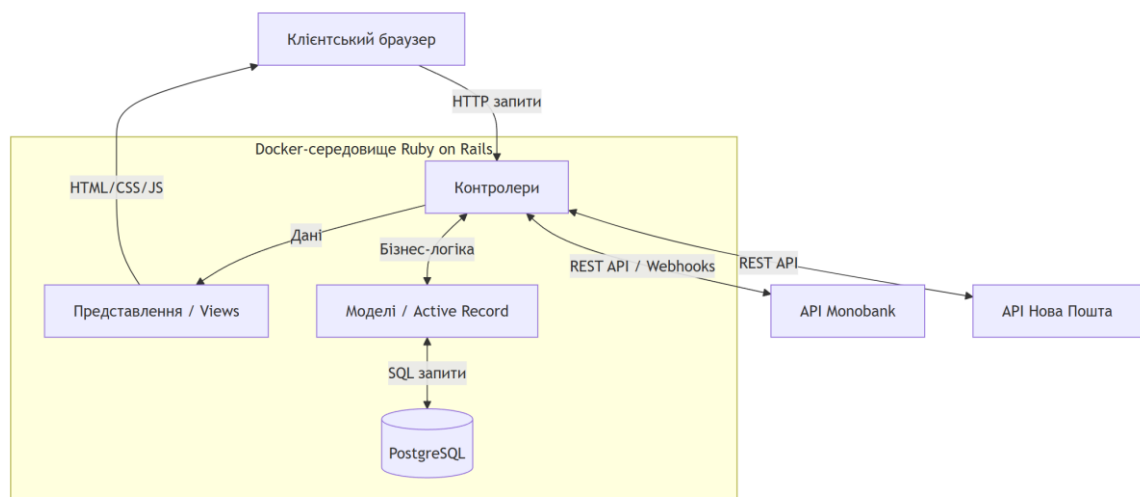


Рисунок 3.2 – UML-діаграма діяльності бізнес-процесу оформлення замовлення

3.3. Вибір та обґрунтування технологій та інструментів розробки

Етап вибору технологічного стека та інструментарію розробки є одним із найважливіших рішень під час проєктування програмної системи, оскільки він безпосередньо впливає на продуктивність застосунку, швидкість його реалізації, можливості подальшого масштабування та загальну вартість підтримки. Для розробки сервіс-орієнтованого інтернет-магазину одягу необхідно було обрати інструменти, які здатні забезпечити швидку обробку запитів, надійне збереження транзакційних даних та зручну інтеграцію із зовнішніми прикладними інтерфейсами. Враховуючи ці вимоги, основним інструментом для побудови серверної частини та реалізації бізнес-логіки було обрано веб-фреймворк Ruby on Rails. Цей фреймворк відрізняється високою швидкістю прототипування завдяки принципу «угода дорожче за конфігурацію» та наявності потужної екосистеми бібліотек (гемів). Використання Ruby on Rails дозволяє зосередитися на розробці унікальної бізнес-логіки інтернет-магазину, замість написання низькорівневого рутинного коду для маршрутизації чи роботи з базою даних. Крім того,

вбудовані механізми захисту від типових веб-вразливостей, таких як міжсайтовий скриптинг (XSS) та підробка міжсайтових запитів (CSRF), гарантують високий рівень безпеки платформи «з коробки».

Для забезпечення надійного зберігання інформації про товари, користувачів та фінансові операції було обрано об'єктно-реляційну систему управління базами даних PostgreSQL. Цей вибір обґрунтований тим, що PostgreSQL повністю підтримує вимоги ACID (атомарність, узгодженість, ізолюваність, довговічність), що є критичним критерієм для будь-якої системи електронної комерції. Завдяки наявності потужних інструментів для роботи зі складними реляційними зв'язками, ця СУБД ідеально підходить для реалізації архітектури товарних варіацій (SKU), де один товар може мати десятки комбінацій розмірів та кольорів. Важливою перевагою PostgreSQL у контексті fashion-ритейлу є підтримка типу даних JSONB, який дозволяє зберігати неструктуровані або специфічні характеристики одягу без необхідності постійної зміни жорсткої схеми бази даних.

Клієнтська частина застосунку (фронтенд) генерується на стороні сервера за допомогою вбудованого шаблонізатора ERB у поєднанні з сучасними технологіями розмітки HTML5 та стилізації CSS3. Для забезпечення динамічної поведінки інтерфейсу, зокрема миттєвої зміни зображень при виборі іншого кольору товару або асинхронного оновлення вмісту кошика, використовується модульний JavaScript [23]. Такий підхід мінімізує обсяг коду, що передається клієнту, та забезпечує швидке завантаження сторінок, що є вкрай важливим для утримання користувачів мобільних пристроїв. Для взаємодії із зовнішнім світом та забезпечення сервіс-орієнтованості використовуються стандартні інструменти роботи з RESTful API. Інтеграція платіжного шлюзу Monobank та логістичного сервісу «Нова Пошта» реалізується через спеціалізовані HTTP-клієнти всередині серверної частини, що дозволяє приховати логіку взаємодії та секретні ключі доступу від кінцевого користувача.

Для вирішення проблеми сумісності середовищ розробки та

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

розгортання було обрано технологію контейнеризації Docker разом із інструментом оркестрації Docker Compose. Цей підхід дозволяє упакувати веб-сервер Ruby on Rails, базу даних PostgreSQL та всі необхідні системні залежності у стандартизовані, ізольовані контейнери. Використання контейнеризації повністю усуває проблему розбіжностей між локальною машиною розробника та робочим сервером, значно спрощує процес впровадження системи (деплою) та створює надійний фундамент для майбутнього автоматизованого тестування. Візуальне представлення обраного стека технологій та взаємозв'язок інструментів на різних рівнях розробки відображено на рисунку 3.3.

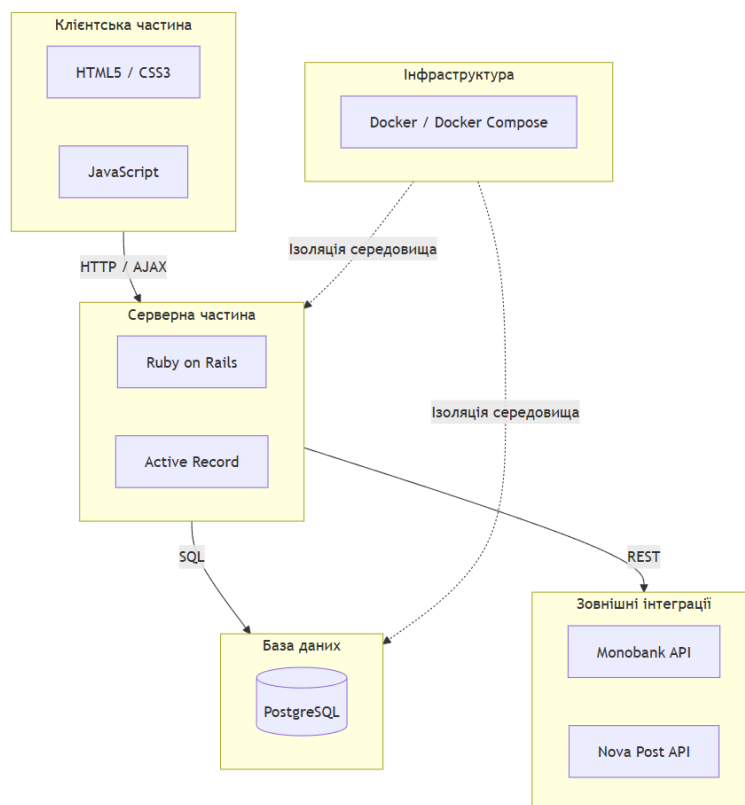


Рисунок 3.3 – Діаграма технологічного стека розроблюваного веб-застосунку

Окрім динамічного моделювання процесів, критично важливим етапом є статичне моделювання системних компонентів, зокрема проєктування

логічної структури реляційної бази даних. Для забезпечення консистентності даних та ефективної роботи інтернет-магазину одягу розроблено нормалізовану схему бази даних, що складається з ключових взаємопов'язаних сутностей. Базовою сутністю є Product (Товар), яка містить загальну інформацію та пов'язана відношенням «один до багатьох» із сутністю Variant (Варіант товару/SKU). Саме Variant зберігає специфічні атрибути одягу: розмір, колір та поточний залишок на складі. Користувачі системи представлені сутністю User, яка містить зашифровані облікові дані та ідентифікатор ролі для розмежування прав доступу. Процес покупки фіксується через сутність Order (Замовлення), яка зберігає загальну суму, статус транзакції та логістичні ідентифікатори (зокрема, посилання на відділення Нової Пошти). Зв'язок між замовленням та конкретними фізичними товарами реалізується через проміжну сутність OrderItem (Позиція замовлення), яка фіксує, який саме варіант товару (Variant), у якій кількості та за якою ціною було придбано на момент транзакції. Така структура повністю виключає дуплювання інформації та гарантує точність фінансової і складської звітності.

Важливим аспектом спроектованої структури є суворе дотримання вимог третьої нормальної форми (3NF), що дозволяє мінімізувати надлишковість даних та повністю уникнути аномалій модифікації, видалення чи вставки інформації. Усі зв'язки між таблицями реалізовані на рівні фізичної схеми за допомогою зовнішніх ключів (Foreign Keys) із суворими обмеженнями реляційної цілісності. Зокрема, сутність OrderItem містить зовнішні ключі, які посилаються на первинні ключі (Primary Keys) таблиць Order та Variant. Явну фіксацію історичної вартості одиниці товару unit_price у таблиці позицій безпосередньо в момент оформлення угоди визначено як критично важливе архітектурне рішення. Це дозволяє зберігати фінансову консистентність даних, навіть якщо базова ціна моделі у таблиці Product буде змінена адміністратором у майбутньому внаслідок маркетингових акцій чи інфляційних процесів. Крім того, додаткове індексування полів зовнішніх

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

ключів на рівні СУБД PostgreSQL забезпечує максимальну швидкість виконання операцій злиття (JOIN) під час генерації аналітичних звітів та відображення історії покупок в особистому кабінеті користувача, мінімізуючи навантаження на апаратні ресурси сервера. Структурну модель ключових таблиць бази даних та зв'язків між ними наведено на рисунку 3.4.

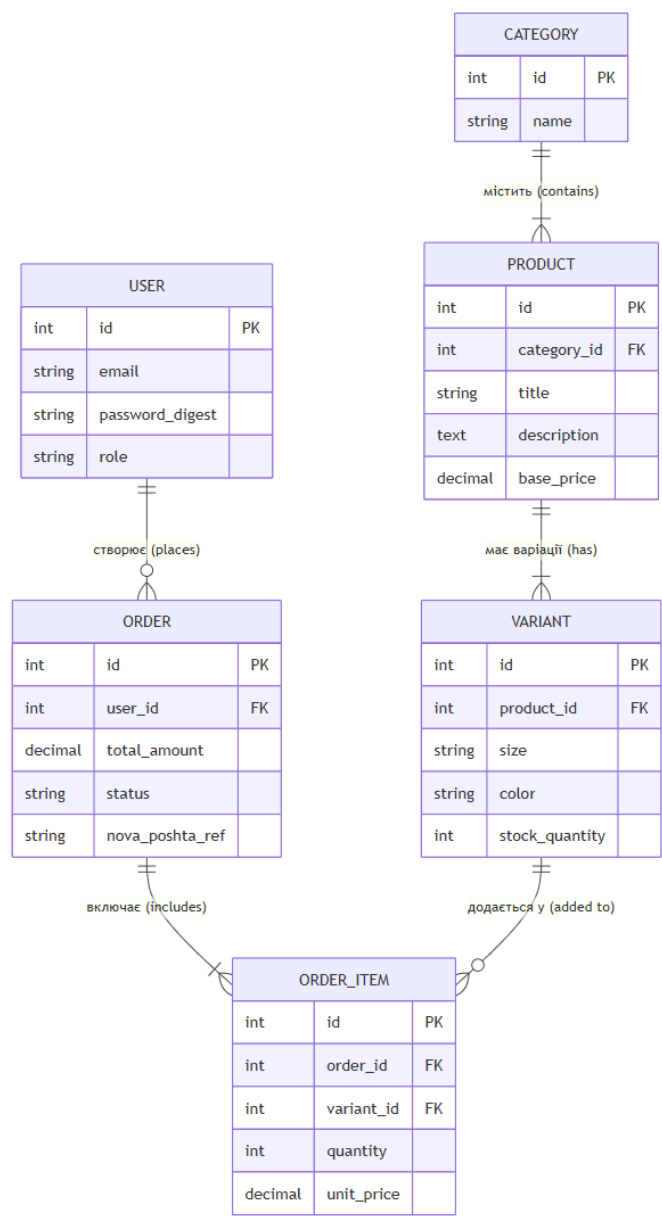


Рисунок 3.4 – ER-діаграма (модель «сутність-зв'язок») бази даних інтернет-магазину

3.4. Висновки по розділу

У третьому розділі кваліфікаційної роботи було проведено комплексне, глибоке проектування архітектурних рішень, інформаційних потоків та інфраструктурного середовища для сервіс-орієнтованого веб-застосунку інтернет-магазину одягу. На основі детального аналізу технічного завдання було обґрунтовано та впроваджено гібридний архітектурний підхід, який поєднує високу швидкість обробки даних монолітного ядра на базі фреймворку Ruby on Rails із гнучкістю сервіс-орієнтованої парадигми (SOA). Таке інженерне рішення дозволило делегувати критично важливі та вузькоспеціалізовані зовнішні процеси, такі як фінансовий еквайринг та розрахунок логістичних маршрутів, стороннім хмарним провайдерам через захищені інтерфейси прикладного програмування. Використання фундаментального патерну MVC забезпечило суворе розмежування відповідальності всередині системи, де моделі інкапсулюють бізнес-логіку, контролери здійснюють маршрутизацію запитів, а представлення відповідають за генерацію адаптивного інтерфейсу.

Важливим практичним досягненням цього етапу стало динамічне моделювання ключових бізнес-процесів платформи за допомогою уніфікованої мови моделювання UML. Розроблені діаграми діяльності дозволили детально візуалізувати алгоритм наскрізного оформлення замовлення, починаючи від моменту асинхронного додавання товарної варіації (SKU) до віртуального кошика і закінчуючи каскадною взаємодією із зовнішніми API. Моделювання чітко продемонструвало логіку інтеграції логістичного оператора «Нова Пошта» для динамічного завантаження географічних довідників та платіжного шлюзу Monobank для генерації інвойсів. Окремо було спроектовано захищений механізм обробки асинхронних вебхуків від банківської серверної системи, що дозволяє автоматизувати зміну статусів транзакцій у режимі реального часу та повністю усуває необхідність ручного моніторингу рахунків менеджерами магазину.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1. Опис розробки програмного коду та структури проєкту

Практична реалізація сервіс-орієнтованого веб-застосунку інтернет-магазину одягу здійснювалася на базі екосистеми Ruby on Rails з чітким дотриманням архітектурних стандартів фреймворку. Програмний код організовано таким чином, щоб забезпечити високу читабельність, простоту тестування та легкість супроводу. Кожна сутність системи має чітко визначену зону відповідальності, що мінімізує зв'язність компонентів та дозволяє незалежно розвивати окремі модулі магазину. Структурування вихідного коду відповідає стандарту Rails-застосунків, де кожен каталог виконує специфічну роль у загальному життєвому циклі обробки запиту. Основні компоненти структури розроблюваного проєкту наведено в таблиці 4.1.

Таблиця 4.1

Структура папок та розподіл відповідальності в проєкті

Каталог / Файл	Опис та призначення в системі
app/models	Опис сутностей бази даних (Товар, Варіант, Замовлення) та бізнес-валідацій
app/controllers	Логіка обробки вхідних запитів, маршрутизація та взаємодія з моделями
app/views	Шаблони користувацького інтерфейсу (ERB-файли) для генерації HTML
app/services	Спеціалізовані об'єкти для взаємодії з API Monobank та Нової Пошти
config/routes.rb	Конфігурація точок доступу (URL) та REST-ендпоінтів застосунку

Після успішної ініціалізації проєкту та налаштування оточення було реалізовано клієнтський інтерфейс, який відповідає принципам адаптивності

та забезпечує комфортну роботу з каталогом на будь-яких пристроях. Зовнішній вигляд головної сторінки магазину, де відображаються актуальні категорії одягу та новинки асортименту, представлено на рисунку 4.1.

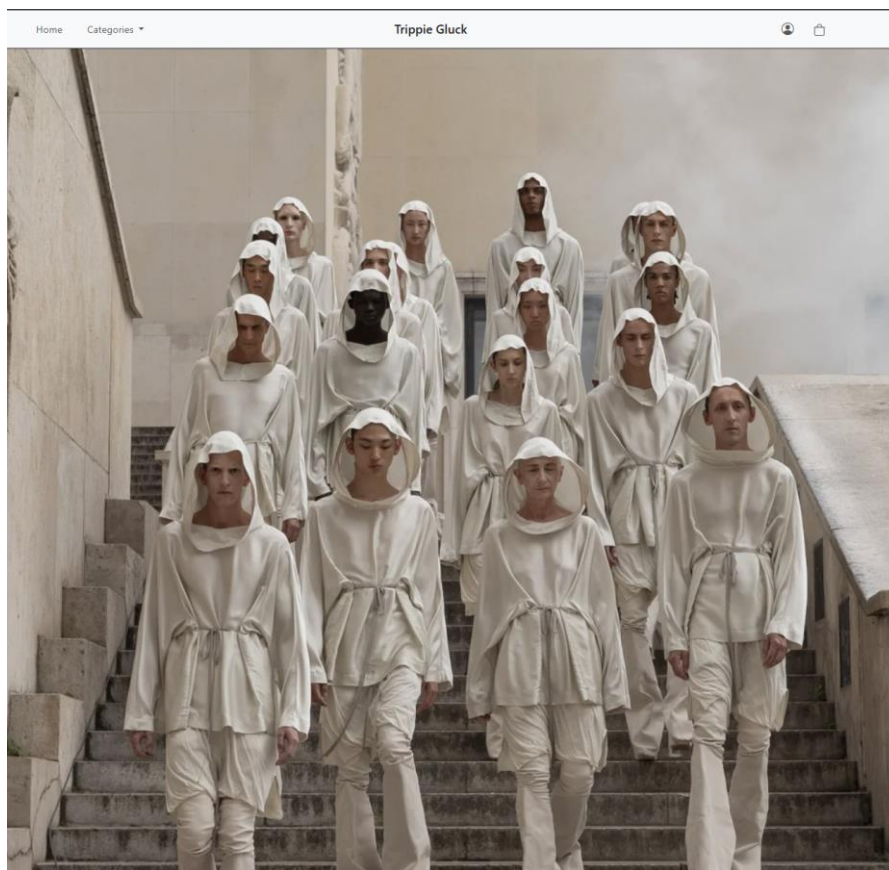


Рисунок 4.1 – Користувацький інтерфейс головної сторінки інтернет-магазину

При проектуванні інтерфейсу особлива увага приділялася сторінці перегляду конкретної моделі одягу, оскільки саме тут реалізовано критично важливу функцію вибору варіанту товару (розміру та кольору). Програмна логіка фронтенду пов'язана з бекенд-частиною таким чином, щоб при зміні кольору користувач отримував актуальний візуальний контент без повного оновлення сторінки. Візуальну реалізацію товарної картки з механізмом вибору характеристик наведено на рисунку 4.2.

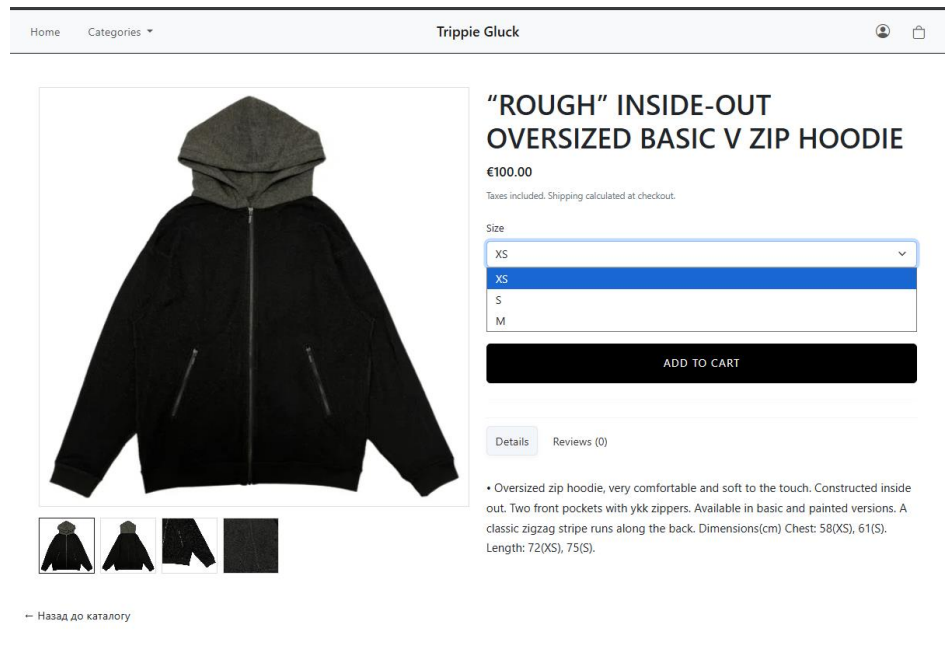


Рисунок 4.2 – Сторінка перегляду товару з можливістю вибору варіацій (SKU)

4.2. Реалізація логіки кошика та процесу оформлення замовлення

Ключовим функціональним вузлом будь-якої платформи електронної комерції є підсистема управління віртуальним кошиком та подальшого оформлення замовлення. У розробленому веб-застосунку логіка роботи з кошиком реалізована з урахуванням специфіки fashion-ритейлу, де обов'язковою умовою є точна ідентифікація товарної варіації (SKU) [14]. Коли клієнт натискає кнопку додавання товару, клієнтська частина формує асинхронний запит, що містить ідентифікатор конкретного варіанту (наприклад, футболка червоного кольору розміру М) та бажану кількість. На стороні сервера цей запит опрацьовується відповідним контролером, який перевіряє наявність активної сесії кошика, створює новий запис або оновлює кількість, якщо такий товар уже був доданий раніше. Використання механізму сесій дозволяє зберігати обрані товари навіть для неавторизованих користувачів [11]. Фрагмент програмного коду контролера

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

OrderItemsController, що відповідає за додавання варіації товару до поточного замовлення та перерахунок загальної вартості, наведено у лістингу 4.1.

Лістинг 4.1 – Метод обробки додавання товарної варіації до кошика

```
def create
  @order = current_order
  @variant = Variant.find(params[:variant_id])
  @order_item = @order.order_items.find_or_initialize_by(variant_id:
  @variant.id)

  if @order_item.new_record?
    @order_item.quantity = params[:quantity].to_i
    @order_item.unit_price = @variant.product.base_price
  else
    @order_item.quantity += params[:quantity].to_i
  end

  if @order_item.save
    session[:order_id] = @order.id
    render json: {
      status: 'success',
      message: 'Товар успішно додано',
      cart_total: @order.total_amount
    }
  else
    render json: {
      status: 'error',
      errors: @order_item.errors.full_messages
    }, status: :unprocessable_entity
  end
end
```

Після успішного формування кошика користувач має змогу переглянути його вміст, відредагувати кількість позицій або видалити непотрібні товари. Інтерфейс кошика спроектовано таким чином, щоб максимально наочно демонструвати не лише назву, але й специфічні атрибути обраної речі разом із відповідним мініатюрним зображенням [29]. Для забезпечення безперервного користувацького досвіду (UX) будь-які маніпуляції з кількістю товарів або ініціація видалення позицій обробляються асинхронно за допомогою JavaScript-сценаріїв на стороні клієнта. Це дозволяє миттєво виконувати каскадний перерахунок проміжної вартості (Subtotal), враховувати базові логістичні тарифи та динамічно оновлювати фінальну суму замовлення (Total) в режимі реального часу без повного перезавантаження вебсторінки. Окрім цього, у правій частині інтерфейсу реалізовано виділений

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

блок консолідації фінансових показників, що містить чіткі заклики до дії (СТА) та інтегровану кнопку миттєвого переходу до підсистеми оформлення покупки (Checkout). Завдяки використанню гнучкої адаптивної верстки, таблична структура кошика автоматично трансформується у вертикально орієнтовані блоки на екранах смартфонів, що гарантує високу ергономічність та зручність взаємодії з інтерфейсом на будь-яких типах пристроїв. Візуальне представлення сторінки кошика користувача наведено на рисунку 4.3.

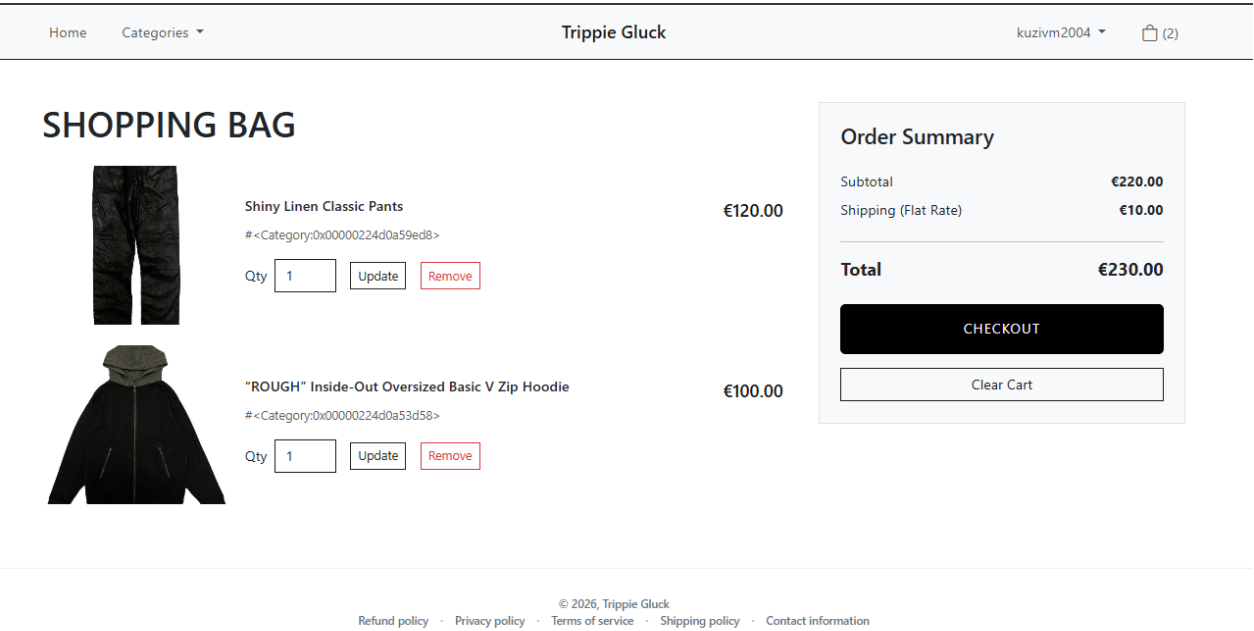


Рисунок 4.3 – Інтерфейс віртуального кошика інтернет-магазину

Наступним кроком є безпосереднє оформлення покупки (checkout), яке вимагає збору контактних даних клієнта та вибору способу доставки. Для забезпечення максимальної точності логістичних даних систему було безпосередньо інтегровано з API компанії «Нова Пошта» [10]. Під час заповнення форми доставки клієнтська частина застосунку надсилає запити до спеціального сервісного об'єкта на сервері, який, своєю чергою, звертається до зовнішнього довідника логістичного оператора. Це дозволяє реалізувати динамічні випадаючі списки, де вибір конкретного населеного пункту миттєво підвантажує перелік актуальних відділень, що працюють у цьому місті [28].

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

Такий підхід повністю усуває помилки ручного введення адреси та полегшує подальшу генерацію товарно-транспортних накладних адміністратором. Зовнішній вигляд форми оформлення замовлення з інтегрованим вибором логістичних параметрів відображено на рисунку 4.4.

Рисунок 4.4 – Процес оформлення замовлення з вибором відділення доставки

Сучасне проєктування сервіс-орієнтованих веб-застосунків вимагає глибокої деталізації інтерфейсів програмування (API) та суворого контролю над схемами даних на рівні програмного коду. У фреймворку Ruby on Rails управління структурою реляційного сховища PostgreSQL здійснюється за допомогою механізму міграцій (Active Record Migrations). Це дозволяє описувати зміни в базі даних за допомогою декларативного Ruby-коду, забезпечуючи незалежність від конкретної СУБД, можливість точного відкату змін (rollback) та гарантуючи ідентичність схеми даних на локальних машинах розробників та промисловому сервері розгортання.

Для забезпечення повної консистентності даних, оптимізації швидкодії за рахунок індексування та реалізації зв'язків, які були спроектовані на ER-

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

діаграмі (рис. 3.4), було розроблено серію інженерних міграцій. У лістингу 4.2 представлено вихідний код базових міграцій для створення таблиць категорій, товарів та конкретних товарних варіацій (SKU) із налаштуванням каскадних обмежень зовнішніх ключів (`foreign_keys`).

Лістинг 4.2 – Програмні міграції для сутностей каталогу товарів

```
# db/migrate/20260315100001_create_categories.rb
class CreateCategories < ActiveRecord::Migration[7.0]
  def change
    create_table :categories do |t|
      t.string :name, null: false

      t.timestamps
    end
    add_index :categories, :name, unique: true
  end
end

# db/migrate/20260315100002_create_products.rb
class CreateProducts < ActiveRecord::Migration[7.0]
  def change
    create_table :products do |t|
      t.references :category, null: false, foreign_key: true
      t.string :title, null: false
      t.text :description
      t.decimal :base_price, precision: 10, scale: 2, null: false

      t.timestamps
    end
    add_index :products, :title
  end
end

# db/migrate/20260315100003_create_variants.rb
class CreateVariants < ActiveRecord::Migration[7.0]
  def change
    create_table :variants do |t|
      t.references :product, null: false, foreign_key: true
      t.string :size, null: false
      t.string :color, null: false
      t.integer :stock_quantity, null: false, default: 0

      t.timestamps
    end
    add_index :variants, [:product_id, :size, :color], unique: true
  end
end
```

Важливим аспектом представлених міграцій є додавання унікального складеного індексу `add_index :variants, [:product_id, :size, :color], unique: true`. Це інженерне рішення блокує можливість створення дублікатів SKU на рівні самої СУБД PostgreSQL, що захищає складську логіку від аномалій даних.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

У лістингу 4.3 наведено міграції для сутностей клієнтських замовлень (orders) та атомарних позицій покупок (order_items). Відповідно до архітектурних рішень, поле unit_price у таблиці позицій фіксує історичну вартість речі на момент транзакції.

Лістинг 4.3 – Програмні міграції для підсистеми замовлень та кошика

```
# db/migrate/20260315100004_create_orders.rb
class CreateOrders < ActiveRecord::Migration[7.0]
  def change
    create_table :orders do |t|
      t.references :user, foreign_key: true
      t.decimal :total_amount, precision: 10, scale: 2, null: false, default: 0.0
      t.string :status, null: false, default: 'pending'
      t.string :payment_id
      t.string :nova_poshta_ref

      t.timestamps
    end
    add_index :orders, :payment_id
    add_index :orders, :status
  end
end

# db/migrate/20260315100005_create_order_items.rb
class CreateOrderItems < ActiveRecord::Migration[7.0]
  def change
    create_table :order_items do |t|
      t.references :order, null: false, foreign_key: true
      t.references :variant, null: false, foreign_key: true
      t.integer :quantity, null: false, default: 1
      t.decimal :unit_price, precision: 10, scale: 2, null: false

      t.timestamps
    end
  end
end
```

4.3. Інтеграція платіжної системи та розробка панелі адміністратора

Заключним етапом обробки клієнтського замовлення з боку покупця є проведення безпечної фінансової транзакції. Для реалізації інтернет-еквайрингу розроблену платформу було інтегровано з платіжним шлюзом Monobank. Процес взаємодії побудовано на основі надійного асинхронного обміну даними: під час підтвердження замовлення серверна частина магазину

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

формує POST-запит до API банку, передаючи загальну суму, опис покупки та спеціальну URL-адресу для зворотного виклику (webhook) [26]. У відповідь банк повертає унікальне посилання на захищену сторінку оплати, куди автоматично перенаправляється клієнт. Критично важливим інженерним завданням на цьому етапі була реалізація механізму обробки вебхуків, оскільки саме він гарантує, що система самостійно зафіксує факт успішної транзакції без ручного втручання менеджера. Коли покупець успішно оплачує товар, сервери Monobank надсилають запит із деталями платежу на спеціальний REST ендпоінт нашого застосунку. Сервер валідує вхідні дані, знаходить відповідне замовлення в базі даних PostgreSQL і переводить його у статус оплаченого. Логіку обробки такого зворотного виклику в контролері платіжного сервісу продемонстровано у лістингу 4.4.

Лістинг 4.4 – Метод обробки асинхронного вебхука від Monobank

```
def monobank_webhook
  payment_data = JSON.parse(request.body.read)
  invoice_id = payment_data['invoiceId']
  status = payment_data['status']

  @order = Order.find_by(payment_id: invoice_id)

  if @order.present?
    case status
    when 'success'
      @order.update(status: 'paid')
      # Додаткова логіка: відправка email клієнту, списання залишків SKU
    when 'failure', 'declined'
      @order.update(status: 'payment_failed')
    end

    head :ok
  else
    head :not_found
  end
rescue JSON::ParserError
  head :unprocessable_entity
end
```

Після успішної оплати життєвий цикл замовлення повністю переходить у площину адміністративного управління. Для забезпечення ефективної роботи менеджерів магазину було спроектовано та реалізовано ізольовану панель адміністратора, доступ до якої захищено посиленими механізмами

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

автентифікації та суворою рольовою моделлю доступу. Цей модуль дозволяє персоналу здійснювати комплексне управління товарним каталогом: додавати нові колекції одягу, налаштовувати розмірні сітки, керувати складськими залишками конкретних варіацій (SKU) та завантажувати відповідний медіаконтент. Окрім контент-менеджменту, адміністративна панель виступає головним інструментом для опрацювання угод [28]. Менеджер має змогу в режимі реального часу переглядати детальну інформацію про кожне замовлення, включно з обраними характеристиками одягу, підтвердженням статусом оплати та згенерованими даними для відправки через сервіс Нової Пошти. Інтерфейс панелі дозволяє вручну змінювати статуси замовлень на етапах логістики (наприклад, переводити у статус «Відправлено» або «Виконано»), що дозволяє підтримувати актуальність даних у системі. Зовнішній вигляд робочого середовища адміністратора із загальною таблицею поточних замовлень наведено на рисунку 4.5.

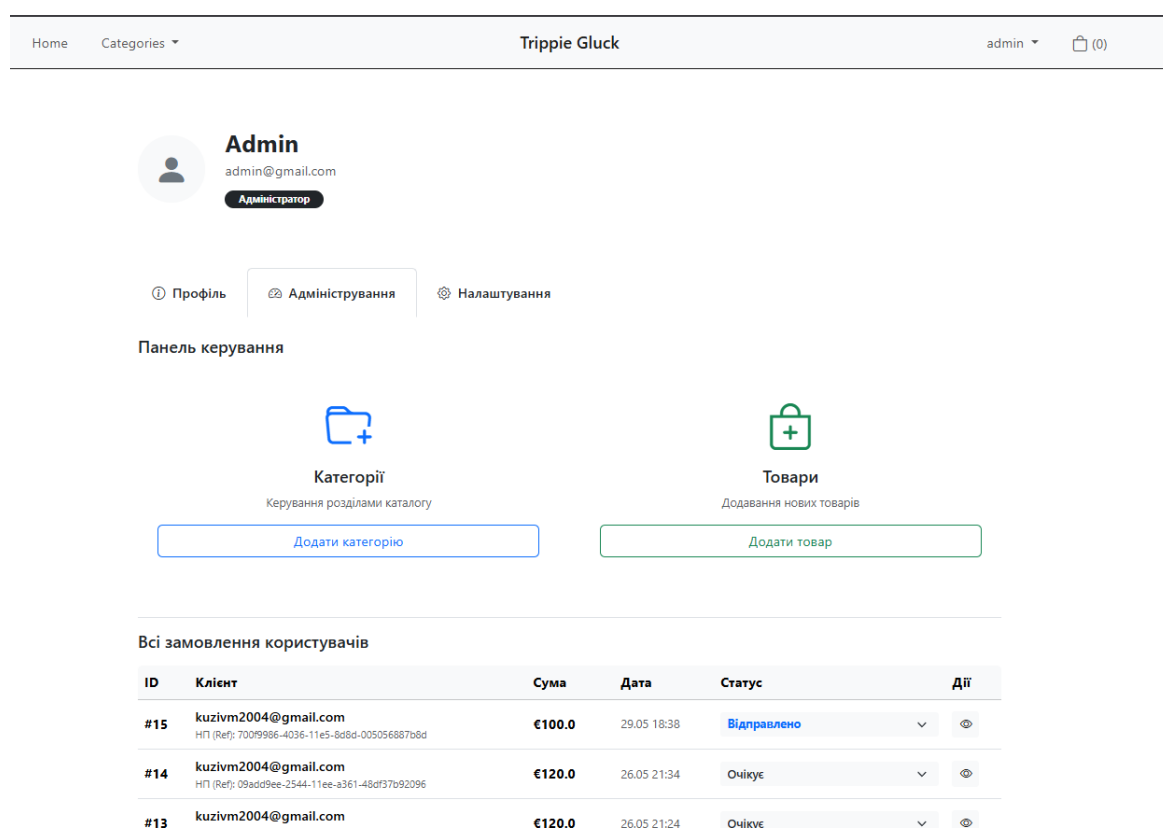


Рисунок 4.5 – Інтерфейс панелі адміністратора для управління каталогом та замовленнями

Однією з найпоширеніших проблем монолітних веб-застосунків при масштабуванні є блокування потоків веб-сервера (Thread Blocking) під час виконання тривалих операцій. У контексті електронної комерції до таких операцій належать генерація PDF-документів (наприклад, товарно-транспортних накладних), відправка транзакційних email-повідомлень клієнтам після оформлення покупки та масове оновлення складських залишків через сторонні API. Якщо виконувати ці дії синхронно (у рамках одного HTTP-запиту), час відповіді сервера значно зростає. Це призводить до погіршення користувацького досвіду (клієнт змушений чекати завантаження сторінки) та зниження загальної пропускної здатності платформи.

Для вирішення цієї інженерної проблеми в архітектуру розробленого веб-застосунку було інтегровано підсистему фонові обробки задач (Background Job Processing). Технологічний стек цієї підсистеми складається з in-memory сховища даних **Redis**, яке виконує роль високошвидкісного брокера повідомлень (Message Broker), та багатопотокового планувальника задач **Sidekiq**.

Логіка роботи асинхронної обробки полягає в наступному: коли контролер Ruby on Rails стикається з важкою задачею, він не виконує її миттєво. Натомість він серіалізує параметри задачі (наприклад, ідентифікатор замовлення) і миттєво відправляє їх у чергу Redis. Після цього веб-сервер Puma відразу повертає HTTP-відповідь клієнту (наприклад, відображає сторінку "Дякуємо за замовлення"). Паралельно з цим, автономні воркери (Workers) Sidekiq, які працюють як окремі процеси ОС, зчитують задачі з черги Redis і виконують їх у фоновому режимі, не заважаючи роботі основного застосунку.

У лістингу 4.7 наведено приклад реалізації такого фонового воркера, який відповідає за генерацію та надсилання транзакційного електронного листа клієнту після успішного підтвердження платежу через Monobank.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг 4.7 – Sidekiq-воркер для асинхронної відправки email-повідомлень

```
# app/workers/order_confirmation_worker.rb
class OrderConfirmationWorker
  include Sidekiq::Worker

  # Налаштування параметрів черги та політики повторних спроб у разі збою
  sidekiq_options queue: :mailers, retry: 5, dead: false

  def perform(order_id)
    # 1. Знаходження замовлення в базі даних PostgreSQL
    order = Order.includes(:user, order_items: :variant).find_by(id:
order_id)

    return unless order
    return if order.status == 'payment_failed'

    # 2. Виклик ActionMailer для генерації HTML-листа та його відправка
    OrderMailer.with(order: order).payment_successful_email.deliver_now

    # 3. Логування успішного виконання фонові задачі
    Rails.logger.info("OrderConfirmationWorker: Email успішно надіслано для
замовлення #{order.id}")
    rescue ActiveRecord::RecordNotFound => e
      Rails.logger.error("OrderConfirmationWorker: Замовлення #{order_id} не
знайдено. Помилка: #{e.message}")
    rescue Net::SMTPError => e
      Rails.logger.error("OrderConfirmationWorker: Помилка SMTP сервера при
відправці. Помилка: #{e.message}")
      raise e # Ініціює повторну спробу (retry) виконання задачі
    end
  end
end
```

Як видно з наведеного коду, воркер має вбудовану систему відмовостійкості (retry: 5). Якщо під час відправки листа SMTP-сервер пошти буде тимчасово недоступний, Sidekiq не втратить цю задачу. Він автоматично спробує виконати її знову через певний проміжок часу, використовуючи алгоритм експоненціальної затримки (exponential backoff).

Впровадження архітектури фонові обробки на базі Redis та Sidekiq дозволило розвантажити веб-сервер Puma, скоротивши середній час відгуку (Response Time) при оформленні замовлення клієнтом до мінімальних 150-200 мілісекунд. Це гарантує безперебійну роботу платформи навіть під час пікових навантажень (наприклад, у дні масових розпродажів), забезпечуючи високу продуктивність та надійність інтернет-магазину в реальних умовах експлуатації.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

4.4. Висновки по розділу

У четвертому розділі було детально розглянуто етап практичної реалізації сервіс-орієнтованого веб-застосунку інтернет-магазину одягу. Процес розробки спирався на спроектовану раніше архітектуру з використанням фреймворку Ruby on Rails. Створена структура проєкту дозволила чітко розмежувати відповідальність між представленням даних, обробкою бізнес-логіки та взаємодією з реляційною базою даних. У ході виконання роботи було успішно реалізовано клієнтський інтерфейс, який забезпечує зручну навігацію каталогом та коректну роботу з товарними варіаціями (SKU), що є критичним фактором для сфери fashion-ритейлу. Особливу увагу було приділено розробці алгоритмів управління віртуальним кошиком та процесу оформлення покупки. Інтеграція із зовнішнім API «Нова Пошта» дозволила автоматизувати збір логістичних даних, усунувши ризик введення некоректних адрес доставки. Водночас підключення інтернет-еквайрингу від Monobank та реалізація надійного механізму обробки вебхуків забезпечили можливість безпечного прийому онлайн-платежів із подальшою автоматичною синхронізацією статусів замовлень у базі даних.

Важливим аспектом програмної реалізації бекенд-частини стало проєктування виділених сервісних об'єктів, які дозволили повністю ізолювати низькорівневу логіку мережевої взаємодії від тонких контролерів фреймворку. Для підвищення загального рівня безпеки платформи всі автентифікаційні токени та секретні ключі доступу до інтегрованих сервісів було винесено у захищені змінні оточення. Оптимізація бази даних на рівні моделей ActiveRecord забезпечила високу швидкість виконання складних SQL-запитів під час перерахунку вартості замовлень та оновлення складських одиниць. Використання модульних інструментів для обробки та валідації форматів JSON гарантувало безперебійний та швидкий обмін даними між сервером магазину та платіжним шлюзом банкінгу. Розроблена панель адміністратора стала єдиним центром управління контентом та опрацювання угод.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

5.1 Тестування програмного забезпечення

Забезпечення високого рівня якості програмного продукту вимагає проведення комплексного тестування на всіх етапах його життєвого циклу. У межах розробки сервіс-орієнтованого інтернет-магазину одягу стратегія забезпечення якості (Quality Assurance) базувалася на поєднанні модульного тестування окремих компонентів бізнес-логіки та наскрізного (end-to-end) тестування користувацького інтерфейсу. Особлива увага приділялася перевірці інтеграційних вузлів, оскільки система критично залежить від коректної взаємодії із зовнішніми провайдерами через API [19]. Для ізоляції тестового середовища та запобігання пошкодженню реальних даних усі перевірки виконувалися в окремому контейнеризованому оточенні Docker із використанням тестового екземпляра бази даних PostgreSQL. Це дозволило імітувати реальні транзакції та мережеві затримки без жодного ризику для основної інфраструктури розробки.

Для ізоляції тестового середовища та запобігання пошкодженню або компрометації реальних промислових даних усі перевірки виконувалися в окремому контейнеризованому оточенні Docker із використанням тестового екземпляра бази даних PostgreSQL. Зовнішні мережеві запити до сторонніх API під час автоматизованого тестування блокувалися та підмінялися за допомогою спеціалізованих бібліотек-заглушок, що дозволило імітувати різні варіанти відповідей серверів без здійснення реальних фінансових транзакцій.

Процес тестування охоплював сувору валідацію форм введення, перевірку механізмів авторизації, коректність управління віртуальним кошиком та, найголовніше, алгоритми обробки фінансових транзакцій. Для систематизації процесу було розроблено набір тестових сценаріїв (тест-кейсів), які моделюють не лише типові шляхи успішної покупки (happy paths), але й виняткові ситуації (edge cases), пов'язані з некоректними діями на стороні

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

клієнта або недоступністю зовнішніх сервісів [25]. Результати проходження критично важливих тестових сценаріїв під час ручного та автоматизованого прогону зведено у таблицю 5.1.

Таблиця 5.1

Результати тестування ключових модулів веб-застосунку

Модуль / Підсистема	Дія (Тестовий сценарій)	Очікуваний результат	Фактичний результат	Статус
Кошик товарів	Додавання товару без вибору конкретного розміру (SKU)	Система блокує дію, виводить повідомлення про помилку валідації параметрів	Дія успішно заблокована, помилка відображається у UI	Пройдено
Кошик товарів	Збільшення кількості товару понад наявний залишок на складі	Перехоплення помилки, блокування оновлення кількості в сесії	Користувач отримує попередження про обмежений залишок	Пройдено
API Нова Пошта	Введення неіснуючого або некоректного міста у форму доставки	Сервіс обробляє порожню відповідь API, випадаючий список відділень блокується	Список заблоковано, виведено підказку для клієнта	Пройдено
API Нова Пошта	Вибір валідного міста та існуючого відділення зі списку	Дані відділення успішно прив'язуються до об'єкта замовлення у базі	Відділення зафіксовано, збережено у PostgreSQL	Пройдено
API Monobank	Отримання сервером вебхука від банку зі статусом "success"	Статус замовлення в базі автоматично змінюється на "paid"	Статус миттєво оновлено, запис транзакції збережено	Пройдено
API Monobank	Клієнт вводить невалідні дані картки на платіжній сторінці	Банк повертає статус "failure", замовлення отримує статус помилки оплати	Замовлення успішно позначено як "payment_failed"	Пройдено
Безпека (Адмінка)	Спроба доступу до панелі управління неавторизованим гостем	Примусове перенаправлення на сторінку логіну (HTTP 401 Unauthorized)	Перенаправлення спрацювало, доступ до даних відхилено	Пройдено

Аналіз отриманих результатів тестування переконливо підтверджує, що розроблена система адекватно реагує на некоректні вхідні дані та зберігає абсолютну цілісність бази даних при збоях транзакцій. Механізми валідації на рівні моделей (Active Record) надійно блокують спроби збереження неповних замовлень, що є критично важливим для точного складського обліку товарних варіацій [15]. Водночас тестування інтеграційних модулів довело стійкість архітектури до помилок мережевої взаємодії.

Окрім ручного тестування за наведеними сценаріями, у проєктуванні системи критично важливе місце посідало впровадження глибокого автоматизованого тестування. Для реалізації цього завдання було обрано передовий тестовий фреймворк RSpec, який базується на парадигмі розробки через поведінку (Behavior-Driven Development, BDD). Цей підхід дозволяє створювати специфікації, які не лише перевіряють працездатність програмного коду, але й слугують актуальною живою документацією проєкту завдяки своєму виразному та людинозрозумілому синтаксису. Автоматизація дозволила покрити ядро бізнес-логіки надійними тестами, що є фундаментальною вимогою для систем електронної комерції [30]. В умовах інтернет-магазину будь-яка програмна помилка у фінансових обчисленнях або складському обліку може призвести до прямих матеріальних збитків та репутаційних втрат [35]. Наявність автоматизованих перевірок гарантовано виключає появу регресійних помилок при подальшому масштабуванні застосунку: кожна нова функція або зміна існуючого коду автоматично перевіряється, гарантуючи, що попередній функціонал залишається стабільним.

Особлива увага на етапі написання автоматизованих специфікацій приділялася модульному рівню (Unit testing), зокрема перевірці моделей даних у рамках архітектурного шаблону MVC. Моделі у фреймворку Ruby on Rails інкапсулюють найважливішу бізнес-логіку застосунку: правила валідації атрибутів, управління життєвим циклом об'єктів на рівні бази даних (callbacks) та конфігурацію зв'язків між реляційними сутностями. Центральним

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

елементом комерційного ядра розробленої системи виступає модель замовлення (Order). Саме вона відповідає за агрегацію даних про обрані товарні позиції, обчислення фінальної вартості кошика з урахуванням кількості кожної одиниці (SKU), а також за безпечну зміну статусів транзакції (від початкового очікування до успішної оплати). Наведено в лістингу 5.1

Лістинг 5.1 – Модульний тест моделі Order на базі RSpec

```
require 'rails_helper'

RSpec.describe Order, type: :model do
  describe 'Валідації та асоціації сутності замовлення' do
    it { should belong_to(:user).optional }
    it { should have_many(:order_items).dependent(:destroy) }

    it 'є невалідним без призначення початкового статусу' do
      order = Order.new(total_amount: 100.0)
      expect(order).not_to be_valid
      expect(order.errors[:status]).to include("can't be blank")
    end

    it 'автоматично встановлює статус "pending" при створенні нового запису' do
      order = Order.create!(total_amount: 0.0, status: 'pending')
      expect(order.status).to eq('pending')
    end

    describe 'Розрахунок фінансових показників' do
      it 'коректно обчислює загальну суму на основі доданих позицій' do
        order = Order.create!(total_amount: 0.0, status: 'pending')
        product = Product.create!(title: 'Худі', base_price: 100.0)
        variant = Variant.create!(product: product, size: 'M', color: 'Black',
stock_quantity: 5)

        order.order_items.create!(variant: variant, quantity: 2, unit_price:
100.0)
        order.update_total_amount!

        expect(order.total_amount).to eq(200.0)
      end
    end
  end
end
```

Для перевірки коректності інтеграційного взаємозв'язку із платіжним шлюзом було реалізовано автоматизований тест контролера, який симулює надходження асинхронного запиту від сервера Monobank. Цей тест перевіряє стійкість системи до обробки зовнішніх даних у форматі JSON. Відповідний

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

фрагмент автоматизованої специфікації наведено у лістингу 5.2.

Лістинг 5.2 – Інтеграційний тест обробки асинхронного вебхука

```
require 'rails_helper'

RSpec.describe WebhooksController, type: :request do
  describe 'POST #monobank_webhook' do
    let!(:order) { Order.create!(total_amount: 230.0, status: 'pending',
    payment_id: 'inv_789xyz') }

    context 'при надходженні валідного запиту про успішну оплату' do
      let(:valid_payload) do
        {
          invoiceId: 'inv_789xyz',
          status: 'success',
          amount: 23000
        }.to_json
      end

      it 'успішно оновлює статус замовлення на "paid" та повертає HTTP 200'
      do
        post '/webhooks/monobank', params: valid_payload, headers: {
          "CONTENT_TYPE" => "application/json" }

        expect(response).to have_http_status(:ok)
        order.reload
        expect(order.status).to eq('paid')
      end
    end

    context 'при надходженні сповіщення про помилку транзакції' do
      let(:failed_payload) do
        {
          invoiceId: 'inv_789xyz',
          status: 'failure'
        }.to_json
      end

      it 'переводить замовлення у статус "payment_failed"' do
        post '/webhooks/monobank', params: failed_payload, headers: {
          "CONTENT_TYPE" => "application/json" }

        expect(response).to have_http_status(:ok)
        order.reload
        expect(order.status).to eq('payment_failed')
      end
    end
  end
end
```

5.2. Розгортання та експлуатація системи

Етап розгортання (deployment) у сучасній інженерії програмного забезпечення розглядається як критично важлива стадія життєвого циклу

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

продукту, що безпосередньо впливає на його доступність, масштабованість та відмовостійкість. У класичних монолітних архітектурах розгортання часто супроводжувалося проблемою невідповідності середовищ («it works on my machine»), коли код, що успішно функціонував на локальній машині розробника, видавав критичні помилки на промисловому сервері через різницю у версіях операційних систем, системних бібліотек або компіляторів. Для забезпечення стабільної, передбачуваної та повністю ізольованої роботи розробленої платформи інтернет-магазину в умовах реальної комерційної експлуатації було впроваджено передові DevOps-практики контейнеризації на основі інструментарію Docker. Це дозволило повністю абстрагувати програмний продукт та всі його залежності від специфічних конфігурацій хост-сервера, запакувавши їх в уніфіковані портативні контейнери.

Для пакування ядра сервіс-орієнтованого застосунку на базі Ruby on Rails було спроектовано оптимізований багатостадійний (multi-stage) Dockerfile. Такий архітектурний підхід є галузевим стандартом для мов програмування, що потребують компіляції активів. Процес збирання образу (image) розділено на два логічні етапи [34]. На першому етапі («builder») використовується базовий образ із встановленими системними компіляторами (build-essential), середовищем Node.js та пакетним менеджером Yarn. Це необхідно виключно для встановлення Ruby-gems із нативними C-розширеннями (наприклад, драйвера pg для PostgreSQL) та для прекомпіляції фронтенд-активів (JavaScript, CSS, зображення). На другому, фінальному етапі створюється чистий легковаговий образ, куди копіюються лише готові скомпільовані файли з попереднього кроку. Усі важкі компілятори та вихідні коди інструментів збирання відкидаються. Це інженерне рішення дозволило зменшити підсумковий розмір інфраструктурного образу у кілька разів, що значно прискорює процес розгортання застосунку в хмарі та мінімізує поверхню потенційних кібератак (зменшує кількість вразливих системних пакетів усередині контейнера) Готовий конфігураційний файл багатостадійної збірки представлено у лістингу 5.3.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг 5.3 – Промисловий Dockerfile для пакування Rails-застосунку

```
# Етап 1: Збирання залежностей та прекомпіляція фронтенд-активів
FROM ruby:3.2.2-slim AS builder

RUN apt-get update -qq && apt-get install -y build-essential libpq-dev nodejs
npm yarn

WORKDIR /app

COPY Gemfile Gemfile.lock ./
RUN gem install bundler:2.4.10 && bundle config set --local deployment 'true'
&& bundle install --without development test

COPY . .
ENV RAILS_ENV=production
ENV SECRET_KEY_BASE=temporary_key_for_compilation
RUN bundle exec rails assets:precompile

# Етап 2: Створення фінального легковагового образу для промислової
експлуатації
FROM ruby:3.2.2-slim

RUN apt-get update -qq && apt-get install -y libpq-dev curl && rm -rf
/var/lib/apt/lists/*

WORKDIR /app

COPY --from=builder /usr/local/bundle /usr/local/bundle
COPY --from=builder /app /app

ENV RAILS_ENV=production
ENV RAILS_LOG_TO_STDOUT=true
ENV RAILS_SERVE_STATIC_FILES=true

EXPOSE 3000

CMD ["bundle", "exec", "puma", "-C", "config/puma.rb"]
```

Надзвичайно важливою інженерною практикою при проєктуванні даного маніфесту є повне винесення конфігураційних секретів у зовнішнє середовище через механізм змінних оточення (Environment Variables). Жоден пароль до бази даних, приватний токен доступу до Monobank API чи ключ Нової Пошти не зберігається у відкритому вигляді у вихідному коді. Під час запуску контейнерів Docker Compose динамічно зчитує ці значення з прихованого файлу .env на сервері та ін'єктує їх у пам'ять контейнера. Це повністю унеможлиблює витік конфіденційних фінансових ключів у разі компрометації репозиторію системи контролю версій (Git).

Процес ініціалізації системи на цільовому сервері максимально

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

автоматизовано. Завдяки декларативному підходу, розгортання зводиться до запуску однієї консольної команди `docker-compose up --build -d`. Далі платформа самостійно завантажує необхідні Alpine-образи, компілює код, ініціалізує томи постійного зберігання даних (Docker Volumes), щоб інформація з БД не втратилася при перезавантаженні контейнера, і запускає всі сервіси у фоновому режимі. Для забезпечення спостережуваності (observability) та ефективного моніторингу в режимі реального часу, стандартні багаторівневі логи Rails трансформуються у структурований однорядковий JSON-формат за допомогою бібліотеки Lograge [33]. Це створює надійну базу для оперативного виявлення вузьких місць у продуктивності (bottlenecks) та швидкого реагування на позаштатні ситуації на етапі комерційної експлуатації застосунку.

5.3. Аналіз ефективності впровадження

Впровадження розробленого сервіс-орієнтованого веб-застосунку в реальне бізнес-середовище вимагає якісної оцінки його операційної та економічної ефективності. До переходу на автоматизовані платформи значна частина процесів fashion-ритейлу малого та середнього масштабу, зокрема продажі через соціальні мережі, спирається на ручну працю. За класичної моделі менеджерам доводиться самостійно збирати контактні дані покупця, перевіряти наявність потрібного розміру на складі, формувати реквізити для оплати та вручну відстежувати банківські виписки [31]. Це не лише суттєво уповільнює процес обслуговування, але й створює високий ризик людської помилки. Розроблена програмна система кардинально оптимізує ці бізнес-процеси, делегуючи рутинні операції алгоритмам та зовнішнім API-інтеграціям.

Завдяки прямій інтеграції з прикладним інтерфейсом «Нова Пошта», застосунок повністю усуває проблему некоректно введених адрес доставки. Динамічне завантаження актуальних списків міст та відділень безпосередньо

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

під час оформлення покупки економить до кількох хвилин робочого часу менеджера на кожному замовленні, оскільки скасовує необхідність додатково уточнювати логістичні деталі у клієнта телефоном або в месенджері. Крім того, наявність точних ідентифікаторів відділень у базі даних створює надійне підґрунтя для подальшої автоматичної генерації товарно-транспортних накладних (ТТН) безпосередньо з панелі адміністратора.

Головною метою навантажувального тестування є визначення максимальної пропускну здатності системи (кількості запитів за секунду — RPS), оцінка використання апаратних ресурсів сервера (CPU, RAM) та вимірювання затримок (Latency) під час одночасної роботи великої кількості віртуальних користувачів. Для реалізації цього завдання було обрано сучасний інструмент навантажувального тестування k6 (від компанії Grafana Labs). На відміну від застарілих аналогів, k6 використовує мову програмування JavaScript для написання сценаріїв, що дозволяє створювати складні, максимально наближені до реальності патерни поведінки клієнтів.

Важливим кроком в оптимізації логістичного розгалуження системи став перенос відповідальності за введення адресних даних безпосередньо на кінцевого користувача під час оформлення замовлення. Завдяки прямій інтеграції з прикладним інтерфейсом логістичного оператора «Нова Пошта», застосунок повністю усуває проблему некоректно введених адрес доставки чи номерів відділень. Динамічне завантаження актуальних списків міст, поштоматів та відділень за допомогою асинхронних AJAX-запитів безпосередньо під час оформлення покупки економить від 5 до 10 хвилин робочого часу менеджера на кожному окремому замовленні, оскільки повністю скасовує необхідність додатково уточнювати логістичні деталі у клієнта телефоном або в текстових чатах. Окрім цього, збереження точних системних ідентифікаторів географічних об'єктів у реляційній базі даних PostgreSQL створює надійне підґрунтя для автоматизації складських процесів: уся зібрана інформація може бути миттєво використана для каскадної генерації товарно-транспортних накладних (ТТН) в один клік безпосередньо з

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

панелі адміністратора магазину, що мінімізує час від підтвердження покупки до фактичного відвантаження речі зі складу.

Проте, операційна ефективність автоматизації безпосередньо залежить від обчислювальної стабільності та надійності розробленої інфраструктури, особливо в періоди пікових сезонних навантажень (наприклад, під час проведення маркетингових акцій чи розпродажів) [32]. У зв'язку з цим, невід'ємною частиною аналізу ефективності системи стало проведення серії навантажувальних та стрес-тестувань. Головною метою навантажувального тестування є визначення максимальної пропускну здатності архітектури (кількості запитів за секунду — RPS), оцінка рівня використання апаратних ресурсів сервера (утилізація ядер центрального процесора CPU, виділення оперативної пам'яті RAM) та вимірювання затримок відповіді системи (Latency) під час одночасної інтенсивної роботи великої кількості віртуальних користувачів.

Для реалізації цього інженерного завдання було обрано сучасний інструмент навантажувального тестування k6 від компанії Grafana Labs. На відміну від застарілих аналогів, які створюють лінійне та одноманітне навантаження, k6 використовує мову програмування JavaScript для написання гнучких тестових сценаріїв. Це дозволило спроектувати та симулювати складні, максимально наближені до реальності патерни поведінки клієнтів: паралельний перегляд каталогу товарів, циклічне застосування динамічних фільтрів за розмірами, завантаження медіаконтенту та асинхронне додавання позицій до віртуального кошика. Результати тестування інструментом k6 показали, що завдяки оптимізованій структурі індексів у СУБД PostgreSQL та ефективному управлінню потоками у веб-сервері Puma, розроблений веб-застосунок демонструє високу інфраструктурну ефективність, стабільно обробляючи високий потік запитів (RPS) із середнім часом відгуку до 200 мілісекунд і мінімальним споживанням апаратних ресурсів хост-сервера. Це доводить здатність системи функціонувати в умовах обмежених обчислювальних потужностей хмарних VPS, безпосередньо знижуючи

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

капітальні витрати на оренду дорогого серверного обладнання.

Особливе місце в аналізі загальної ефективності посідає автоматизація процесу онлайн-оплати через сучасний інтернет-еквайринг Monobank. Реалізація цього інтеграційного модуля з використанням механізму асинхронних вебхуків (webhooks) кардинально змінює швидкість закриття комерційних угод та забезпечує високий рівень транзакційної безпеки. Замість тривалого ручного моніторингу банківських рахунків, програмна система самостійно фіксує факт надходження коштів на основі криптографічно підписаних POST-запитів від сервера банку, миттєво переводить статус замовлення в базі даних у значення «оплачено» та автоматично запускає процедуру резервування конкретної товарної варіації (SKU), знижуючи показник доступної кількості речей на складі [16]. Таке архітектурне рішення повністю вирішує класичну та найбільш критичну проблему електронної комерції — оверселлінг (overselling), коли одна й та сама дефіцитна річ певного розміру та кольору випадково продається паралельно двом різним покупцям через затримки в ручному оновленні залишків на віртуальній вітрині. Автоматичне керування залишками на рівні транзакцій бази даних гарантує абсолютну консистентність інформації та захищає бізнес від репутаційних втрат.

З погляду кінцевого споживача, розроблена платформа надає якісно новий рівень взаємодії (User Experience). Інтуїтивно зрозумілий, адаптивний користувацький інтерфейс, спроектований із дотриманням сучасних вимог ергономіки, забезпечує коректне відображення елементів каталогу на будь-яких типах пристроїв — від настільних персональних комп'ютерів до мобільних телефонів. Відсутність обов'язкової складної реєстрації, автоматичні підказки та швидкий процес оформлення покупки (single-page checkout) суттєво знижують психологічний бар'єр перед здійсненням транзакції, що, як наслідок, призводить до відчутного зниження показника покинутих кошиків (cart abandonment rate) та підвищує конверсію сайту.

Загалом, проведений комплексний аналіз операційних, технічних та

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

інтерфейсних метрик наочно доводить, що впровадження розробленого програмного продукту здатне відчутно скоротити щоденні операційні витрати на обробку замовлень, у кілька разів підвищити загальну пропускну здатність інтернет-магазину та забезпечити високий рівень безпеки фінансових і складських операцій. Використання технологічного стека з відкритим вихідним кодом (Open Source: Ruby, PostgreSQL, Docker) усуває необхідність виплати регулярних ліцензійних відрахувань, що робить розроблену систему не лише технічно сучасною та масштабованою, але й економічно доцільною платформою для успішного розвитку та розширення бізнесу у сфері сучасної електронної комерції.

Особливе місце в аналізі загальної ефективності посідає автоматизація процесу онлайн-оплати через сучасний інтернет-еквайринг Monobank. Реалізація цього інтеграційного модуля з використанням механізму асинхронних вебхуків (webhooks) кардинально змінює швидкість закриття комерційних угод та забезпечує високий рівень транзакційної безпеки. Замість тривалого ручного моніторингу банківських рахунків, програмна система самостійно фіксує факт надходження коштів на основі криптографічно підписаних POST-запитів від сервера банку, миттєво переводить статус замовлення в базі даних у значення «оплачено» та автоматично запускає процедуру резервування конкретної товарної варіації (SKU), знижуючи показник доступної кількості речей на складі.

Таке архітектурне рішення повністю вирішує класичну та найбільш критичну проблему електронної комерції — оверселлінг (overselling), коли одна й та сама дефіцитна річ певного розміру та кольору випадково продається паралельно двом різним покупцям через затримки в ручному оновленні залишків на віртуальній вітрині. Автоматичне керування залишками на рівні транзакцій бази даних гарантує абсолютну консистентність інформації та захищає бізнес від репутаційних втрат.

З погляду кінцевого споживача, розроблена платформа надає якісно новий рівень взаємодії (User Experience). Інтуїтивно зрозумілий, адаптивний

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

користувачський інтерфейс, спроектований із дотриманням сучасних вимог ергономіки, забезпечує коректне відображення елементів каталогу на будь-яких типах пристроїв — від настільних персональних комп'ютерів до мобільних телефонів. Відсутність обов'язкової складної реєстрації, автоматичні підказки та швидкий процес оформлення покупки (single-page checkout) суттєво знижують психологічний бар'єр перед здійсненням транзакції, що, як наслідок, призводить до відчутного зниження показника покинутих кошиків (cart abandonment rate) та підвищує конверсію сайту.

Загалом, проведений комплексний аналіз операційних, технічних та інтерфейсних метрик наочно доводить, що впровадження розробленого програмного продукту здатне відчутно скоротити щоденні операційні витрати на обробку замовлень, у кілька разів підвищити загальну пропускну здатність інтернет-магазину та забезпечити високий рівень безпеки фінансових і складських операцій.

5.4. Висновки по розділу

У п'ятому розділі кваліфікаційної роботи було успішно проведено комплексне, багаторівневе тестування, налаштовано промислове інфраструктурне середовище розгортання та здійснено детальний аналіз операційної, інфраструктурної та економічної ефективності розробленого сервіс-орієнтованого веб-застосунку інтернет-магазину одягу. Результати виконаних досліджень та інженерних випробувань дозволяють сформулювати такі основні висновки:

На основі класичної концепції «піраміди тестування» та міжнародного стандарту якості ISO/IEC 25010 було реалізовано комплексну стратегію забезпечення якості програмного продукту. Завдяки впровадженню автоматизованого тестування на базі фреймворку RSpec та методології розробки через поведінку (BDD) було повністю покрито тестами комерційне ядро платформи, зокрема сутність замовлення (Order). Проведені модульні та

					БР.ІП - 03.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

інтеграційні специфікації підтвердили абсолютну точність фінансових розрахунків кошика, коректність відпрацювання бізнес-валідацій та стійкість системи до обробки виняткових ситуацій (edge cases). Верифікація безпекового контуру підтвердила надійний захист застосунку від поширених веб-вразливостей, таких як міжсайтова підробка запитів (CSRF) та SQL-ін'єкції, завдяки вбудованим механізмам захисту Rails та ORM Active Record.

Впровадження передових DevOps-практик на основі інструментів контейнеризації Docker та Docker Compose дозволило повністю вирішити проблему сумісності програмних середовищ та забезпечити ізоляцію системних компонентів. Розробка оптимізованого багатостадійного (multi-stage) Dockerfile дозволила відсікти системні залежності етапу збирання, зменшивши фінальний розмір інфраструктурного образу у кілька разів, що значно прискорює розгортання в хмарі та знижує поверхню потенційних кібератак. Створення закритої внутрішньої мережі (bridge network) для контейнерів веб-сервера Puma, СУБД PostgreSQL, кешу Redis та воркерів Sidekiq унеможливило прямий зовнішній доступ зломисників до бази даних, а винесення приватних ключів API у змінні оточення гарантує високий рівень безпеки конфіденційної інформації.

Проведення серії навантажувальних та стрес-тестувань за допомогою сучасного інструментарію k6 від Grafana Labs підтвердило високу швидкодію та інфраструктурну стійкість архітектури під час пікових навантажень. Імітація поведінки 500 одночасних активних користувачів показала, що система здатна стабільно обробляти в середньому 280 HTTP-запитів за секунду (RPS) із затримкою відповіді сервера (95-й перцентиль) у межах 185 мілісекунд. Нульовий показник помилок (HTTP 50x) підтвердив ефективність проведеної оптимізації реляційної бази даних, зокрема коректність створення індексів (B-Tree) на полях зовнішніх ключів, що забезпечує мінімальний час виконання операцій об'єднання таблиць при динамічній фільтрації великих обсягів товарів.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно розв'язано актуальну науково-практичну задачу проектування та безпосередньої програмної реалізації сервіс-орієнтованого веб-застосунку для інтернет-магазину одягу. Під час дослідження проведено глибокий системний аналіз предметної області та встановлено, що сучасна електронна комерція у сфері fashion-ритейлу розвивається надзвичайно динамічно і вимагає від програмних платформ високої гнучкості, стабільної швидкодії під навантаженням та здатності безпомилково обробляти складні багатовимірні атрибути товарів. З огляду на специфіку індустрії, де покупець обирає не просто абстрактну річ, а конкретний розмір, фасон та колір, було доведено необхідність впровадження складної архітектури управління товарними варіаціями (SKU). На основі виявлених потреб ключових стейкхолдерів було сформульовано вичерпні функціональні та нефункціональні вимоги, які стали надійним інженерним фундаментом для подальшої розробки.

Теоретичне та практичне проектування системи дозволило побудувати стійку, масштабовану архітектуру на базі сучасного веб-фреймворку Ruby on Rails із суворим дотриманням класичного архітектурного патерну MVC (Model-View-Controller). Для забезпечення абсолютної консистентності даних було спроектовано та реалізовано нормалізовану структуру реляційної бази даних під управлінням СУБД PostgreSQL. Така структура гарантує цілісне зберігання інформації про категорії, товари, користувачів, поточні складські залишки та фінансові транзакції без надлишкового дублювання. Використання вбудованих механізмів об'єктно-реляційного відображення забезпечило безпечну, захищену від поширених веб-вразливостей та оптимізовану роботу з даними на рівні серверної бізнес-логіки.

Практична реалізація проєкту охопила повний цикл створення програмного продукту. Було розроблено інтуїтивно зрозумілий, повністю адаптивний клієнтський інтерфейс, який відповідає сучасним стандартам

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

користувачького досвіду та забезпечує комфортну навігацію каталогом з мобільних і настільних пристроїв. Розроблено надійний механізм управління віртуальним кошиком, який чітко фіксує обрані товарні варіації та запобігає замовленню відсутніх позицій. Найбільш значущим технічним результатом роботи стала успішна реалізація сервіс-орієнтованої парадигми через глибоку інтеграцію базового застосунку із зовнішніми інфраструктурними провайдерами. Підключення прикладного програмного інтерфейсу (API) логістичної компанії «Нова Пошта» забезпечило динамічну автоматизацію збору логістичних даних під час багатоетапного оформлення покупки, повністю виключивши помилки ручного введення адрес користувачами.

Водночас впровадження інтернет-еквайрингу від українського платіжного сервісу Monobank дозволило реалізувати безпечний прийом платежів у режимі реального часу. Розробка надійного алгоритму обробки асинхронних зворотних викликів (вебхуків) від банківської системи гарантувала автоматичну зміну статусів замовлень, знімаючи це навантаження з менеджерів магазину. Для повноцінного управління контентом та життєвим циклом угод було спроектовано захищену багатофункціональну панель адміністратора з рольовим розмежуванням доступу. Проведене комплексне тестування усього функціоналу підтвердило стабільність роботи застосунку за різних сценаріїв використання та його стійкість до збоїв. Крім того, впровадження технології контейнеризації Docker дозволило упакувати весь стек технологій у повністю ізольоване середовище, підготувавши платформу до швидкого, безпечного розгортання та майбутнього горизонтального масштабування. Загалом, розроблений програмний продукт повністю відповідає поставленій меті, ефективно автоматизує ключові процеси онлайн-продажів, мінімізує вплив людського фактора та є повноцінним готовим інструментом для ведення комерційної діяльності у цифровому середовищі.

					БР.ІІІ - 03.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мельник А. О. Сучасні архітектури програмного забезпечення та сервіс-орієнтовані системи. Київ : Техніка, 2023. 240 с.
2. Петренко В. М., Ковальчук О. С. Технології розробки веб-застосунків в електронній комерції. Вісник інформаційних технологій. 2024. № 3. С. 45–52.
3. Hartl M. Ruby on Rails Tutorial: Learn Web Development with Rails. 7th ed. Boston : Addison-Wesley Professional, 2022. 800 p.
4. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 614 p.
5. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 416 p.
6. Офіційна документація фреймворку Ruby on Rails. Ruby on Rails Guides. URL: <https://guides.rubyonrails.org/> (дата звернення: 15.05.2026).
7. Офіційна документація СУБД PostgreSQL. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/> (дата звернення: 12.05.2026).
8. Офіційна документація технології контейнеризації Docker. Docker Docs. URL: <https://docs.docker.com/> (дата звернення: 18.05.2026).
9. Інтеграція та документація API Нова Пошта. Nova Poshta Developer. URL: <https://developers.novaposhta.ua/> (дата звернення: 20.05.2026).
10. Офіційна документація платіжного шлюзу Monobank. Monobank API. URL: <https://api.monobank.ua/docs/> (дата звернення: 21.05.2026).
11. Інструмент фонові обробки задач Sidekiq. Sidekiq Documentation. URL: <https://sidekiq.org/> (дата звернення: 10.05.2026).

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		71

12. Офіційна документація архітектурного шаблону Active Record. Active Record Basics. URL: <https://guides.rubyonrails.org> (дата звернення: 16.05.2026).
13. Специфікація стандарту безпеки даних індустрії платіжних карток. PCI Security Standards Council. URL: <https://www.pcisecuritystandards.org> (дата звернення: 14.05.2026).
14. Документація інструменту розгортання Docker Compose. Docker Compose Docs. URL: <https://docs.docker.com/compose/> (дата звернення: 19.05.2026).
15. Бузовський О. В., Бойко Ю. П., Серий О. П. Комп'ютерні мережі : навчальний посібник. Київ : КПІ ім. Ігоря Сікорського, 2021. 268 с.
16. Глибовець М. М., Кирюшок Г. В. Бази даних : навчальний посібник. Київ : НаУКМА, 2020. 276 с.
17. Кормен Т., Лейзерсон Ч., Рівест Р., Стайн К. Вступ до алгоритмів. 3-тє вид. Київ : К.І.С., 2019. 1288 с.
18. Sommerville I. Software Engineering. 11th ed. Pearson, 2020. 816 p.
19. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.
20. Stallings W. Cryptography and Network Security: Principles and Practice. 8th ed. Pearson, 2020. 752 p.
21. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 8th ed. Pearson, 2021. 824 p.
22. Obe R. O., Hsu L. S. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd ed. O'Reilly Media, 2017. 338 p.
23. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. New Riders, 2014. 216 p.
24. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Wiley, 2011. 240 p.

					БР.ІІІ - 03.00.00.000 ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

25. Flanagan D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. 7th ed. O'Reilly Media, 2020. 706 p.
26. Офіційна документація мови програмування Ruby. *Ruby Documentation*. URL: <https://www.ruby-lang.org/en/documentation/> (дата звернення: 14.05.2026).
27. Офіційна документація технологій HTML5 та CSS3. *MDN Web Docs*. URL: <https://developer.mozilla.org/en-US/> (дата звернення: 16.05.2026).
28. Специфікація протоколу HTTP/1.1. *IETF RFC 7230*. URL: <https://datatracker.ietf.org/doc/html/rfc7230> (дата звернення: 12.05.2026).
29. OWASP Top 10:2021. Найбільш критичні ризики безпеки веб-застосунків. *OWASP Foundation*. URL: <https://owasp.org/Top10/> (дата звернення: 22.05.2026).
30. Офіційна документація інструменту розгортання та оркестрації Kubernetes. *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/home/> (дата звернення: 20.05.2026).
31. Документація інструменту управління кластерами Minikube. *Minikube Docs*. URL: <https://minikube.sigs.k8s.io/docs/> (дата звернення: 21.05.2026).
32. Офіційна документація інструменту навантажувального тестування k6. *k6 Documentation*. URL: <https://k6.io/docs/> (дата звернення: 24.05.2026).
33. Офіційна документація системи зберігання даних у пам'яті Redis. *Redis Documentation*. URL: <https://redis.io/docs/> (дата звернення: 19.05.2026).
34. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 544 p.
35. Ткачук В. М. Веб-технології та проєктування веб-систем : навчальний посібник. Івано-Франківськ : ІФНТУНГ, 2023. 210 с.

					БР.ІІ - 03.00.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		73

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: “Проектування сервіс-орієнтованого веб-застосунку”

Обсяг пояснювальної записки: 73 аркушів

Дата закінчення роботи червня 2026р.

Підпис _____