

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

**«Розробка мікросервісної архітектури для онлайн-
магазину електроніки на платформі .NET»**

Виконав студент групи ІПЗ-21-2
САВЧУК Ярослав Васильович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **САВЧУК Ярослав Васильович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«Розробка мікросервісної архітектури для онлайн-магазину електроніки на платформі .NET»**,

затверджена Наказом закладу вищої освіти від «14» лютого 2025 р., №61/с

Вихідні дані для роботи: **архітектура програмного забезпечення в контексті електронної комерції, що базується на мікросервісному підході, із використанням платформи .NET для забезпечення масштабованості та зручності супроводу онлайн-магазину електроніки.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М., Власенко О.В.	27.02.2025р.	27.02.2025р.
2	Єфремов Ю.М., Власенко О.В.	27.02.2025р.	27.02.2025р.
3	Єфремов Ю.М., Власенко О.В.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебзастосунку онлайн-магазину електроніки та пояснювальної записки. Пояснювальна записка до випускної роботи містить 73 сторінках, 36 ілюстрацій та 19 таблиць.

Метою роботи є розробка сучасної мікросервісної архітектури для онлайн-магазину електроніки із використанням платформи .NET для забезпечення масштабованості, надійності, високої продуктивності та зручності супроводу.

Архітектура онлайн-магазину базується на розподіленій системі мікросервісів, що комунікують між собою через повідомлення. Кожен мікросервіс відповідає за окрему бізнес-логіку: авторизацію, управління каталогом товарів, обробка кошика, надання знижок, оформлення замовлень.

Система зберігає дані в кількох базах даних, реалізує кешування найпопулярніших товарів, а також підтримує асинхронну обробку подій. Користувачі можуть переглядати каталог продукції, додавати товари до кошика, оформлювати замовлення та отримувати зворотний зв'язок в реальному часі через інтуїтивний інтерфейс.

У процесі реалізації роботи було застосовано сучасні методи аналізу вимог, об'єктно-орієнтованого проектування, побудови розподілених систем, використано шаблони архітектур, засоби контейнеризації, оркестрації та асинхронної взаємодії між сервісами. Застосовані підходи дозволяють не лише створити стабільну та ефективну систему, а й відкривають перспективи для подальшого масштабування й адаптації рішення під нові бізнес-завдання.

КЛЮЧОВІ СЛОВА: МІКРОСЕРВІСИ, .NET, RABBITMQ, POSTGRESQL, REDIS, ВЕБЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, КЕШУВАННЯ.

					ІПЗ.КР.Б – 121 – 25 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка мікросервісної архітектури для онлайн-магазину електроніки на платформі .NET. Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		Савчук Я.В.						
Керівник.		Лобанчикова Н.М.					3	73
Рецензент		Концидайло А.М.				Житомирська політехніка, група ІПЗ-21-2		
Зав. каф.		Вакалюк Т. А.						

ABSTRACT

The bachelor's thesis consists of a web application for an online electronics store and an explanatory note. The explanatory note includes 73 pages, 36 illustrations, and 19 tables.

The aim of this work is to develop a modern microservices architecture for an online electronics store using the .NET platform to ensure scalability, reliability, high performance, and ease of maintenance.

The architecture of the online store is based on a distributed system of microservices that communicate with each other. Each microservice is responsible for a specific business logic: product catalog management, shopping cart processing, discount provision, and order processing.

The system stores data in several databases, implements caching of the most popular products, and supports asynchronous event processing. Users can browse the product catalog, add items to the shopping cart, place orders, and receive real-time feedback through an intuitive interface.

During the implementation of the project, modern methods of requirements analysis, object-oriented design, distributed systems development, architectural patterns, containerization tools, orchestration, and asynchronous service interaction were applied. The approaches used made it possible not only to build a stable and efficient system but also opened up opportunities for further scaling and adaptation of the solution to new business challenges.

KEYWORDS: MICROSERVICES, .NET, RABBITMQ, POSTGRESQL, REDIS, WEB APPLICATION, E-COMMERCE, CACHING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ОНЛАЙН-ПЛАТФОРМИ.....	9
1.1 Постановка задачі	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Вибір архітектури онлайн-магазину електроніки	15
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	18
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ОНЛАЙН-МАГАЗИНУ ЕЛЕКТРОНІКИ.....	24
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	24
2.2 Опис структур баз даних.....	33
2.3 Опис алгоритмів роботи програми	39
2.4 Реалізація функціоналу системи онлайн-магазину	40
Висновки до другого розділу.....	45
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН- МАГАЗИНОМ ЕЛЕКТРОНІКИ	47
3.1 Порядок встановлення та налаштування параметрів платформи.....	47
3.2 Структура інтерфейсу онлайн-магазину електроніки	48
3.3 Тестування роботи програмного комплексу.....	56
Висновки до третього розділу	59
ВИСНОВКИ	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62
ДОДАТКИ.....	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

(Create, Read, Update, Delete)

API – прикладний програмний інтерфейс

CRUD – основні операції, що використовуються для управління даними

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						6

ВСТУП

У сучасному цифровому середовищі електронна комерція відіграє дедалі важливішу роль у повсякденному житті. Швидкість, зручність і доступність онлайн-покупок стали основними очікуваннями користувачів, особливо у сфері продажу електроніки. На цьому фоні стрімко зростають вимоги до якості програмного забезпечення: воно повинне бути швидким, гнучким, надійним та легко масштабованим. Із розвитком технологій класичні монолітні системи дедалі частіше поступаються місцем більш гнучким і динамічним рішенням, зокрема мікросервісній архітектурі, яка дає змогу будувати програмні комплекси з незалежних компонентів, що розвиваються та масштабуються окремо один від одного. Тому розробка мікросервісної архітектури для онлайн-магазину електроніки на платформі .NET є актуальною та своєчасною.

Метою кваліфікаційної роботи є розробка мікросервісної архітектури для онлайн-магазину електроніки на платформі .NET, яка забезпечує гнучкість, масштабованість, відмовостійкість та зручність подальшого розвитку.

Для досягнення поставленої мети необхідним є вирішення наступних фахових **задач**:

- провести аналіз виданого завдання, сформулювати задачі дипломного проєктування;
- провести аналіз аналогів програмних продуктів;
- провести вибір архітектури онлайн-магазину електроніки;
- провести проєктування мікросервісної архітектури онлайн- магазину електроніки: UML-діаграми, бази даних, алгоритми роботи;
- провести програмну реалізацію та тестування функціоналу мікросервісної архітектури для онлайн- магазину електроніки

Об’єктом дослідження є процес розробки та програмної реалізації інформаційних систем електронної комерції, орієнтованих на кінцевого користувача та адміністративне керування.

Предметом дослідження є моделі, методи та технології розробки мікросервісна архітектури для як онлайн-магазину електроніки на платформі .NET.

Практичне значення розробленого програмного комплексу полягає в тому, що його можна використовувати як основу для створення повноцінної комерційної платформи з можливістю подальшого розширення, адаптації під інші сфери діяльності або інтеграції з додатковими модулями. Архітектурні рішення, реалізовані в межах цієї роботи, можуть стати базою для розробки будь-яких сервісно орієнтованих систем із високими вимогами до продуктивності, стабільності та масштабованості.

Випускна кваліфікаційна робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел із 20 найменувань (2 стор.), 3-х додатків, 19-ти таблиць та 36 ілюстрацій по тексту. Повний обсяг роботи становить 73 сторінках.

РОЗДІЛ 1. АНАЛІЗ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ОНЛАЙН-ПЛАТФОРМИ

1.1 Постановка задачі

Сучасні тенденції в розвитку інформаційних технологій демонструють не лише зростання обсягів даних, а й збільшення кількості користувачів онлайн-сервісів, що закономірно тягне за собою зростання вимог до швидкодії, доступності та масштабованості програмних рішень. Однією з галузей, де це особливо помітно, є електронна комерція – динамічна і водночас складна сфера цифрової економіки. У межах цієї галузі інтернет-торгівля електронікою займає важливе місце, адже характеризується стабільно високим попитом на техніку, широким асортиментом товарів та постійною необхідністю оперативно оновлювати пропозиції для кінцевого споживача.

Зі зростанням трафіку, кількості одночасних покупок, необхідністю обробляти замовлення у реальному часі, а також ураховуючи потребу в інтеграції з платіжними сервісами, логістикою та зовнішніми маркетинговими платформами, класичні монолітні архітектури починають стрімко втрачати ефективність. Їх слабе місце – це складність масштабування окремих частин системи, високі ризики виникнення збоїв при оновленні компонентів, складна підтримка відмовостійкості та надмірна пов'язаність елементів, що гальмує розвиток проєкту.

Розв'язати ці проблеми дає змогу перехід до мікросервісної архітектури. Такий підхід базується на принципі поділу великої системи на окремі сервіси, кожен з яких виконує чітко визначену бізнес-функцію. У результаті з'являється можливість масштабувати не всю систему, а лише її окремі компоненти, швидко впроваджувати зміни без ризику для решти функціоналу, інтегрувати нові сервіси без необхідності кардинальної перебудови всього комплексу. Усе це значно підвищує стійкість і стабільність програмного продукту.

У випадку з онлайн-магазинами електроніки такі вимоги особливо актуальні. Тут потрібно враховувати не лише швидке оновлення каталогу, а й варіативність комплектацій товарів, підтримку гнучкої системи знижок, інтеграцію з платіжними шлюзами, формування замовлень і роботу з великою кількістю одночасних користувачів. Усе це вимагає архітектури, яка буде не лише технічно гнучкою, а й здатною до подальшого нарощування функціональності.

Основною задачею цієї роботи є саме побудова такої архітектури та реалізація її у вигляді прототипу програмного комплексу для онлайн-магазину електроніки. Передбачалося створити систему, яка дозволить вирішити вказані вище проблеми та буде відповідати вимогам до сучасних інформаційних платформ.

У межах дослідження було проаналізовано предметну область електронної комерції, вивчено типові бізнес-процеси –зокрема, авторизація та аутентифікація, управління каталогом, формування кошика, обробку замовлень, застосування знижок, а також виявлено вимоги до системи в умовах постійної динаміки даних і високого навантаження.

Після цього було здійснено огляд існуючих архітектурних підходів до розробки онлайн-магазинів, виявлено їх сильні та слабкі сторони з точки зору вимог до масштабованості, продуктивності та підтримки. Це дозволило обґрунтувати вибір саме мікросервісної архітектури як основи для майбутнього рішення.

У процесі проєктування було виділено окремі функціональні домени – такі як авторизація, каталог товарів, управління кошиком, оформлення замовлень, застосування знижок, а також обробка платежів. Було визначено спосіб взаємодії між сервісами через REST API та асинхронну подієво-орієнтовану комунікацію за допомогою брокера повідомлень. Важливу роль у побудові системи відіграє API-шлюз, через який маршрутизуються всі запити, та який виконує роль єдиної точки входу в систему.

Технологічний стек, обраний для реалізації рішення, включає C# та платформу .NET 9, реляційні бази даних PostgreSQL, SQL Server і SQLite, Redis для кешування, RabbitMQ для забезпечення асинхронної взаємодії, а також Docker для контейнеризації всіх компонентів.

Реалізація системи передбачала створення базового прототипу з найнеобхіднішими сервісами та ключовими сценаріями: авторизація, додавання товару до кошика, оформлення замовлення. Після цього було здійснено розгортання системи у контейнерах та перевірено її працездатність у локальному середовищі.

У процесі тестування було підтверджено стабільність роботи мікросервісів, їхню здатність обробляти запити незалежно один від одного та коректно взаємодіяти між собою. Також було сформовано базову технічну документацію, яка описує архітектуру, способи взаємодії сервісів та інтерфейси для інтеграції з іншими системами.

Окрім реалізації прототипу, окрему увагу приділено вимогам до системи: вона має бути масштабованою, відмовостійкою, простою в супроводі та здатною до оновлення окремих компонентів без впливу на роботу всієї платформи.

1.2 Аналіз аналогів програмного продукту

При розробці нового програмного комплексу для онлайн-магазину електроніки важливим етапом є аналіз існуючих аналогічних рішень, які вже функціонують на ринку. Такий аналіз дозволяє виявити найкращі практики побудови архітектури, функціональних можливостей та організації бізнес-процесів, а також визначити типові проблеми та недоліки, яких слід уникати під час реалізації власного продукту.

В якості об'єктів аналізу були обрані найбільш популярні українські та міжнародні онлайн-магазини електроніки, які займають лідируючі позиції за

обсягами продажів та відвідуваністю. Нижче наведено докладний огляд кожного з обраних аналогів із характеристикою їх сильних і слабких сторін.

Rozetka є безперечним лідером українського ринку електронної комерції, пропонуючи широкий асортимент товарів, включаючи електроніку, побутову техніку та аксесуари (рис. 1.1).

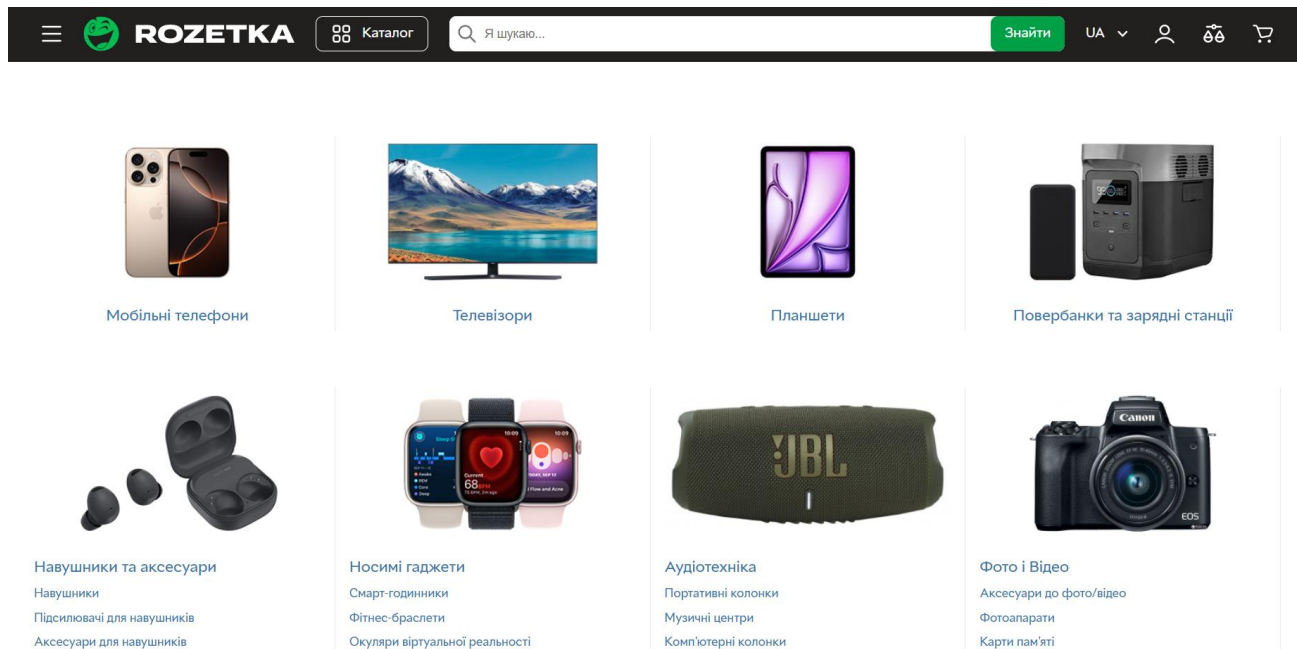


Рис. 1.1. Інтерфейс головної сторінки онлайн-магазину Rozetka [15]

Серед переваг слід відзначити високу продуктивність системи навіть при пікових навантаженнях, зручну навігацію та пошук товарів, підтримку численних платіжних методів і способів доставки, інтеграцію з програмами лояльності та бонусними системами, стабільну роботу мобільної версії та мобільного додатку [15].

До недоліків відносимо перевантаження інтерфейсу великою кількістю рекламних блоків, затримка при обробці складних замовлень, недостатньо адаптивна персоналізація пропозицій для користувачів [15].

Foxtrot спеціалізується на продажу побутової техніки та електроніки та є одним із найстаріших онлайн-ритейлерів в Україні. Інтерфейс головної сторінки представлено на рис.1.2 [7].

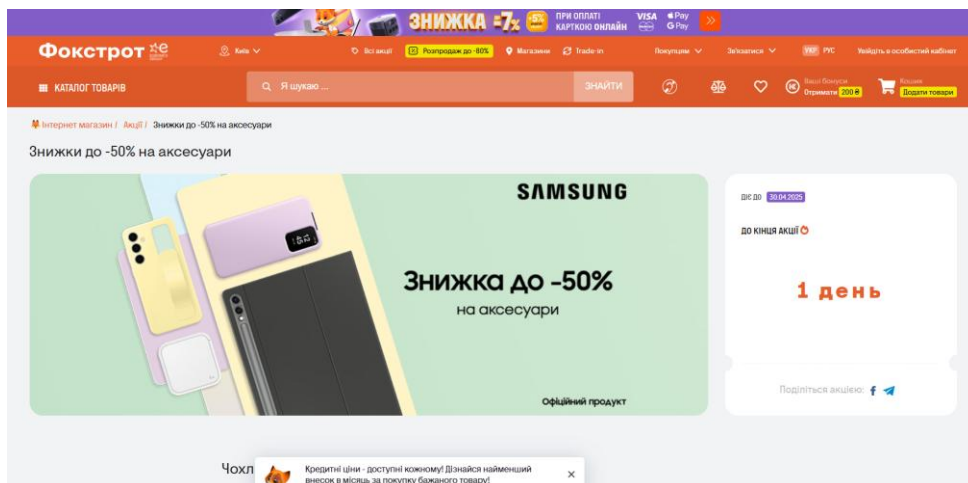


Рис. 1.2. Інтерфейс головної сторінки онлайн-магазину Foxtrot [7]

Аналіз функціоналу інтерфейсу онлайн-магазину Foxtrot дозволив визначити наступні переваги, зокрема слід відмітити чітку категоризацію товарів і структуровану навігацію; зручність оформлення замовлень, постійна система акцій і знижок, які відображаються на більшості вікон вебдодатку.

В результаті проведеного аналізу були відмічені наступні недоліки, зокрема: час завантаження сайту вищий у порівнянні з конкурентами; відмічена менш гнучка система пошуку (обмежені фільтри); невисока інтеграція додаткових сервісів (наприклад, онлайн-страхування товару) [7].

Comfy.ua позиціонується як магазин побутової електроніки нового покоління, акцентуючи увагу на сучасних технологіях та інноваційному підході до обслуговування клієнтів (рис. 1.3) [3].



Рис. 1.3. Інтерфейс головної сторінки онлайн-магазину Comfy.ua [3]

Провівши тестування та аналіз функціональних можливостей вебдодатку онлайн-магазину Comfy.ua були визначені наступні переваги, зокрема це швидка робота сайту і мобільної версії; інтуїтивно зрозумілий інтерфейс і швидкий доступ до основних функцій; високий рівень інтеграції клієнтської підтримки (онлайн-чат, чат-боти) [3].

Серед недоліків слід акцентувати увагу на тому, що іноді виникають збої при обробці складних замовлень з великою кількістю товарів та обмежені можливості персоналізації пропозицій.

На ступним проведемо аналіз функціоналу вебдодатку онлайн магазину «Eldorado», який є добре відомим брендом, що працює на українському ринку електроніки та побутової техніки (рис. 1.4) [6].

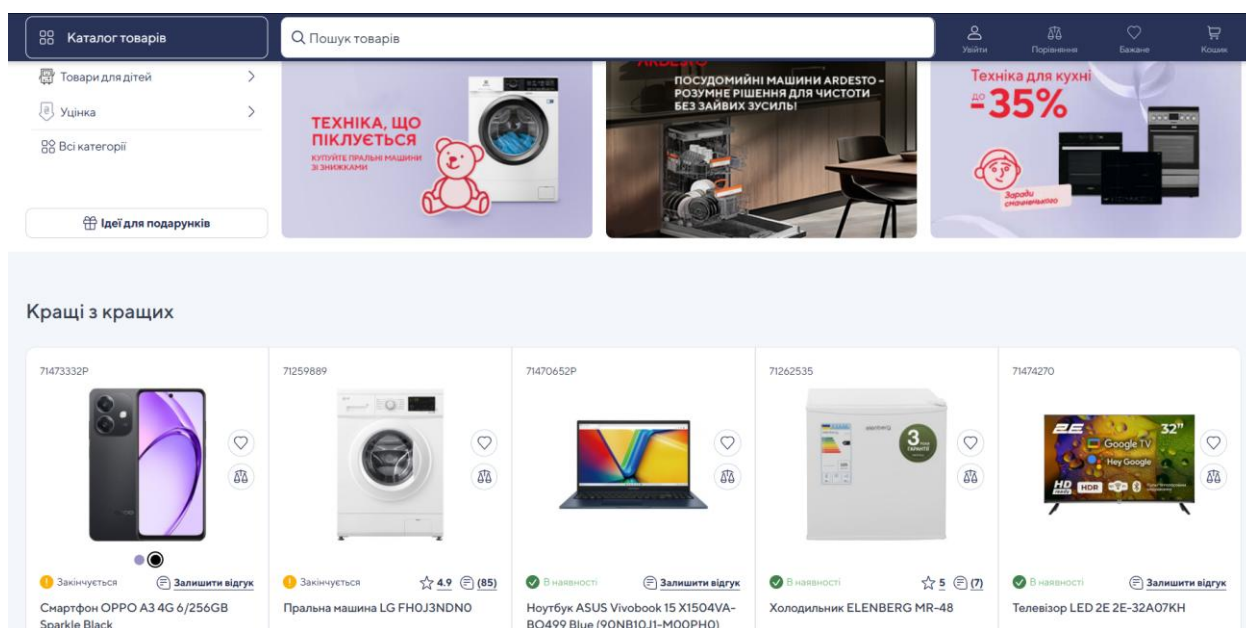


Рис. 1.4. Інтерфейс каталогу продукції онлайн-магазину Eldorado [6]

Серед переваг роботи вебдодатку слід відмітити просту реєстрацію та оформлення замовлень, розміщення рекламних блоків та опис товару. До недоліків відносимо нестабільність роботи сайту в години пікового навантаження та відсутність повноцінної підтримки асинхронної взаємодії між сервісами, що ускладнює масштабування.

Національний маркетплейс «Алло» спеціалізується на продажу смартфонів, гаджетів, аксесуарів та побутової техніки. Відомий своїм

акцентом на мобільних технологіях. Інтерфейс головної сторінки онлайн-магазину представлено на рис. 1.5 [1].

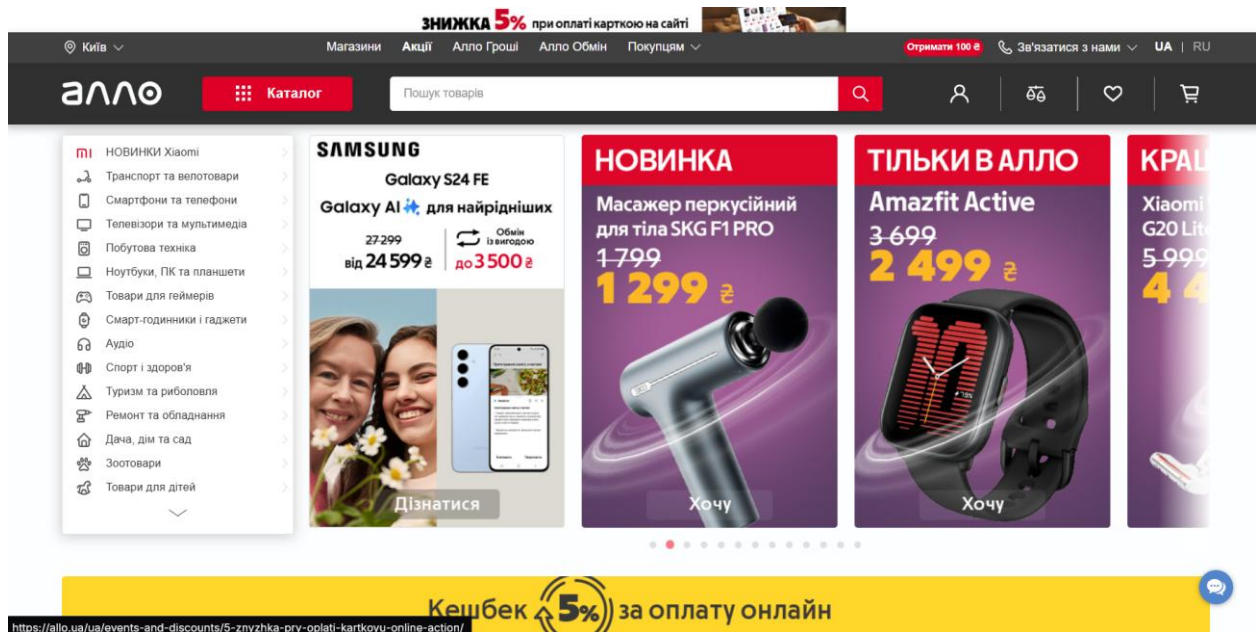


Рис. 1.5. Інтерфейс головної сторінки онлайн-магазину Allo.ua [1]

Проведений аналіз функціональності вебдодатку онлайн-магазину Allo.ua дозволив встановити високу швидкість роботи мобільного додатку, можливість інтеграції системи рекомендацій на основі штучного інтелекту, розвинена програма лояльності та персоналізовані пропозиції. Серед недоліків – наявність окремих проблем зі стабільністю процесингу платежів; перевантаженість деяких сторінок з товарами і рекламними банерами.

1.3 Вибір архітектури онлайн-магазину електроніки

Одним з найважливіших етапів у розробці розподілених інформаційних систем є вибір архітектурної моделі, яка здатна ефективно забезпечити всі функціональні та нефункціональні вимоги до програмного продукту. З огляду на специфіку електронної комерції – постійне навантаження, необхідність гнучкого масштабування, швидке оновлення даних, безперервний доступ і тісну взаємодію з зовнішніми сервісами – найоптимальнішим рішенням стало впровадження мікросервісної архітектури.

Побудова системи як сукупності незалежних сервісів дозволяє досягти автономності кожного функціонального блоку, що позитивно впливає як на продуктивність, так і на підтримку та розвиток проєкту. Кожен мікросервіс у системі має власну зону відповідальності, свою базу даних та ізольовану логіку, що дозволяє розгортати, оновлювати або масштабувати їх окремо один від одного без ризику для всієї системи.

Загальна архітектура системи включає шість основних мікросервісів: сервіс авторизації, каталог товарів, кошик, система знижок, замовлення та API-шлюз. Централізована маршрутизація запитів реалізована за допомогою YARP, який виступає єдиною точкою входу для всіх користувацьких звернень до системи.

Сервіс авторизації відповідає за реєстрацію, вхід, генерацію JWT-токенів, перевірку прав доступу та зберігання даних користувачів. Його архітектура побудована на основі вертикального зрізу з реалізацією CQRS. Для обробки запитів застосовується бібліотека MediatR, а маршрути реалізовані через Carter. База даних сервісу – PostgreSQL. Логіка розділена на запити та команди, що дозволяє ефективно обробляти як автентифікаційні дії, так і запити на отримання інформації про користувача або зміну паролю. Завдяки чіткій структурі коду сервіс легко масштабувати, підтримувати та розширювати новими функціями, такими як керування ролями або двофакторна автентифікація.

Каталог товарів реалізовано як окремий сервіс, який відповідає за зберігання, відображення, фільтрацію та управління товарами та категоріями. Архітектурно він також побудований на основі вертикального зрізу, CQRS, MediatR і Carter, що дозволяє логічно розділити обробку запитів та команд, зробити код підтримуваним і придатним до подальшого масштабування. Для зберігання даних використовується PostgreSQL, а інтерфейс API реалізовано з чітким розмежуванням прав доступу, що дозволяє адміністраторам створювати, редагувати й видаляти товари, а користувачам – лише переглядати інформацію.

Сервіс кошика виконує роль буферного сховища товарів, які користувач обирає перед оформленням замовлення. Він поєднує в собі роботу з PostgreSQL для зберігання постійних даних і Redis – для кешування, що забезпечує високу швидкодію операцій додавання, оновлення та видалення позицій у кошику. Архітектура сервісу базується на вертикальному зрізі та CQRS, із використанням MediatR і Carter, що дозволяє гнучко управляти запитами та зменшує зв'язаність між компонентами. Кожна зміна у складі кошика виконується через окрему команду, а перегляд – через окремі запити, що відповідає принципам поділу відповідальностей.

Система знижок функціонує як окремий сервіс, який визначає, чи застосовуються знижки до певного товару або всієї покупки, на основі заданих адміністратором знижок. Всі знижки зберігаються у базі даних SQLite, яка обрана за свою легкість, швидкодію і простоту інтеграції у сервіс з нечастими оновленнями даних. Комунікація з кошиком відбувається через gRPC – це дозволяє швидко, синхронно передавати запити на перевірку наявності знижки в момент додавання товару до кошика або перед оформленням замовлення. Крім gRPC, сервіс має власне API, реалізоване з використанням тієї ж архітектури, що й у сервісах авторизації та каталогу – вертикальний зріз, CQRS, MediatR, Carter. Це дозволяє адміністраторам створювати, редагувати та видаляти знижки через зручний інтерфейс.

Сервіс замовлень – найскладніший за логікою компонент системи. У ньому реалізовано повний життєвий цикл замовлення: перевірка валідності кошика, збереження замовлення в базу, обробка платіжної інформації, надсилання підтверджень. Цей сервіс побудований за принципами Clean Architecture і DDD (Domain-Driven Design), що передбачає чіткий поділ на доменну модель, застосункову логіку та інфраструктуру. База даних – SQL Server, обраний через свою надійність та інтеграцію з платіжними механізмами. Усі команди та запити реалізовано через CQRS, що дозволяє масштабувати частину, яка відповідає за читання, без впливу на логіку запису.

Замовлення можуть створюватися через API, а обробка їх завершення –через асинхронні події, які публікуються у чергу повідомлень.

Комунікація між сервісами забезпечується у два способи: через асинхронну взаємодію і синхронні виклики. Наприклад, при оформленні замовлення публікується подія в RabbitMQ, яка може бути підписана іншими сервісами –зокрема, складським обліком, звітністю або зовнішньою системою сповіщень. У той же час, gRPC використовується для швидкої синхронної перевірки знижок без необхідності чекати відповіді з черги.

Єдиною точкою входу для зовнішніх клієнтів слугує API-шлюз, побудований на базі YARP. Це рішення забезпечує маршрутизацію запитів, централізовану автентифікацію, агрегацію відповідей від кількох сервісів, облік звернень і базове логування. Завдяки YARP зовнішні користувачі не мають прямого доступу до мікросервісів, що підвищує безпеку та дозволяє централізовано контролювати всю взаємодію.

Сумарно архітектура системи забезпечує високу модульність, масштабованість і стійкість до відмов. Завдяки гнучкому розподілу відповідальностей між сервісами, ізольованому зберіганню даних і сучасним підходам до проєктування – система готова до подальшого розвитку, інтеграції з новими компонентами та адаптації до змін бізнес-вимог.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Успішна реалізація сучасного мікросервісного програмного комплексу потребує ретельно обґрунтованого вибору інструментів, технологій і середовища, які здатні забезпечити не лише високу продуктивність, а й масштабованість, гнучкість, стабільність та зручність подальшого супроводу. У процесі створення онлайн-магазину електроніки було прийнято низку ключових технологічних рішень, які охоплюють серверну логіку, бази даних,

засоби комунікації, інтерфейсну частину, а також інструменти контейнеризації, логування і моніторингу.

Основою серверної логіки стала мова програмування C# у поєднанні з фреймворком .NET 9, який забезпечує чудову продуктивність, сучасні можливості для розробки кросплатформених застосунків, вбудовану підтримку мікросервісної архітектури, HTTP/2, gRPC, а також відмінну інтеграцію з Docker. Розробник має змогу працювати в однорідному середовищі, використовуючи широку екосистему бібліотек і шаблонів, зокрема Carter, Entity Framework Core, MassTransit та інші. Потужна система типізації C# забезпечує контроль над помилками вже на етапі компіляції, що особливо важливо при роботі над великими багатокомпонентними проєктами.

Архітектура сервісів побудована на сучасних патернах. Для обробки запитів і команд використовується MediatR, який реалізує патерн CQRS і дозволяє логічно розділити обробку читання і запису. Це сприяє слабкому зв'язуванню між компонентами, підвищує тестованість та гнучкість.

Carter виступає легким фреймворком для побудови HTTP API з мінімальною надлишковістю, завдяки чому маршрутизація реалізується чисто, декларативно і без надлишкового коду.

Масштабування сервісної взаємодії забезпечується за допомогою бібліотеки MassTransit [11], яка працює поверх RabbitMQ [13] і значно спрощує реалізацію асинхронної комунікації: зокрема, обробку черг, повторні спроби доставки повідомлень, fault-толерантність, а також шаблони взаємодії на кшталт publish/subscribe.

Кожен мікросервіс має свою базу даних, відповідно до принципу "database per service" (табл. 1.1). Для кожного мікросервісу було обрано базу даних, яка найкраще відповідає характеру і частоті доступу до даних.

PostgreSQL використовується для каталогу товарів і авторизації – як сучасне реляційне сховище з потужною підтримкою складних запитів, JSON-структур та гнучкої індексації [12].

SQL Server було обрано для зберігання замовлень і пов'язаної фінансової інформації –це перевірене рішення, яке гарантує стабільність та інтеграцію з більшістю платіжних систем [18].

У системі знижок реалізовано легку базу даних SQLite, яка чудово підходить для сервісів із низьким навантаженням і потребою в простоті обслуговування [17].

Для швидкого доступу до даних у мікросервісі кошика додатково використовується Redis –високошвидкісне in-memory сховище, яке дозволяє значно зменшити затримку при типових операціях [14].

Таблиця 1.1

Обґрунтування вибору баз даних для кожного мікросервісу

Сервіс	База даних	Обґрунтування
Авторизація	PostgreSQL	Потужна реляційна БД з широкими можливостями для фільтрації та індексації
Каталог товарів	PostgreSQL	Потужна реляційна БД з широкими можливостями для фільтрації та індексації
Кошик	PostgreSQL + Redis	PostgreSQL для надійного збереження; Redis –для швидкодії при роботі з сесіями та кешем
Знижки	SQLite	Легка локальна БД для зберігання конфігурацій знижок, не вимагає серверної частини
Замовлення	SQL Server	Потужна корпоративна БД з підтримкою транзакцій і складних бізнес-процесів

Взаємодія між сервісами в системі реалізована як у синхронному, так і в асинхронному форматі. Асинхронна модель базується на зв'язці RabbitMQ та MassTransit – вона дозволяє сервісам обмінюватися повідомленнями без прямої залежності, що важливо для досягнення відмовостійкості та горизонтального масштабування. Події, наприклад, створення замовлення, надсилаються в чергу, де їх можуть перехопити інші сервіси – зокрема знижки, повідомлення чи логістика – без потреби у взаємному виклику.

Синхронна взаємодія реалізована через gRPC – вона застосовується у випадках, коли необхідна миттєва відповідь, наприклад, коли кошик звертається до сервісу знижок для актуалізації вартості замовлення.

Перевагами gRPC є бінарна передача, низька затримка, контрактна модель та висока ефективність при внутрішніх викликах [8].

Доступ до всіх функцій системи організовано через API-шлюз, побудований на базі YARP – сучасного зворотного проксі, який інтегрується безпосередньо в .NET-середовище. Його використання дозволяє централізовано контролювати авторизацію, маршрутизацію, обмеження доступу, логування та агрегацію відповідей із кількох мікросервісів. YARP виступає єдиною точкою входу в систему, спрощуючи керування і підвищуючи рівень безпеки [19].

Фронтенд-частина реалізована з використанням Bootstrap 5 – адаптивного CSS-фреймворку, який дозволяє швидко створювати інтерфейси, що коректно відображаються на різних пристроях [2].

JavaScript доповнює взаємодію з користувачем, забезпечуючи динамічну роботу з кошиком, інтерактивні компоненти і надсилання асинхронних запитів до API без перезавантаження сторінки [9].

Для забезпечення простоти розгортання та ізоляції середовищ було використано контейнеризацію на базі Docker. Кожен мікросервіс упаковано в окремий контейнер із власним середовищем, що дозволяє уникнути конфліктів залежностей, забезпечити швидкий запуск і уніфіковане оточення як для розробки, так і для тестування й продакшн-експлуатації. Під час локальної розробки застосовується Docker Compose – він дає змогу координувати запуск усіх компонентів системи одночасно, включаючи бази даних, брокер повідомлень і API-шлюз. Для промислового розгортання система легко адаптується до Kubernetes, що відкриває можливості автоматичного масштабування, самовідновлення сервісів, централізованого керування конфігураціями та моніторингу [4, 5, 10].

Для збору та аналізу логів використовується Serilog –потужний інструмент, який дозволяє зберігати структуровані журнали подій, будувати дашборди та виявляти аномалії в роботі сервісів [16].

Крім того, у всіх мікросервісах реалізовано Health Checks, що дозволяють перевіряти стан компонентів і повідомляти балансувальник у разі їхньої недоступності [20].

З погляду апаратного забезпечення, система не потребує надмірно потужних ресурсів для локального розгортання. Достатньо машини з чотириядерним процесором, 16 ГБ оперативної пам'яті, SSD-накопичувачем об'ємом від 512 ГБ та встановленим Docker Desktop і середовищем розробки на зразок Visual Studio або Rider. Для продуктивної експлуатації на сервері рекомендовано використовувати восьмиядерні процесори (на кшталт Xeon або EPYC), від 32 ГБ оперативної пам'яті, швидкі NVMe SSD обсягом від 1 ТБ і стабільне мережеве з'єднання з пропускною здатністю не нижче 1 Гбіт/с. Розгортання може здійснюватися як у хмарі, так і у власному серверному середовищі.

Для коректної роботи gRPC-з'єднань важливо забезпечити стабільне мережеве середовище з низькою затримкою – не більше 50 мс – та шифруванням трафіку за допомогою HTTPS і валідних SSL-сертифікатів.

Таким чином, обраний стек технологій дозволяє ефективно реалізувати всі поставлені функціональні завдання системи, забезпечити її стабільну роботу, високу доступність і можливість подальшого розвитку без масштабних перебудов.

Висновки до першого розділу

У результаті виконання першого розділу було проведено комплексний аналіз предметної області онлайн-торгівлі електронікою, що дозволило чітко визначити функціональні потреби, які має покривати програмний комплекс. Було сформульовано перелік задач, які підлягають автоматизації, зокрема: управління каталогом товарів, обробка кошика покупця, нарахування знижок, формування замовлень та їх обробка.

Також здійснено обґрунтований вибір мов програмування, бібліотек і фреймворків, а також визначено вимоги до серверного й мережевого обладнання, що необхідне для стабільної роботи системи в умовах реального використання.

Отже, на основі аналізу та прийнятих рішень сформовано концептуальну основу для подальшого проєктування, реалізації та впровадження програмного комплексу, що буде висвітлено в наступних розділах дипломної роботи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ОНЛАЙН-МАГАЗИНУ ЕЛЕКТРОНІКИ

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

Програмний комплекс, що розробляється побудований на основі мікросервісної архітектури, що забезпечує масштабованість, гнучкість та легкість супроводу. Основними користувачами системи є клієнти –відвідувачі онлайн-магазину, які можуть переглядати товари, додавати їх до кошика та здійснювати покупки, а також адміністратори, які мають доступ до керування товарами, знижками та обробки замовлень. Взаємодія користувача із системою реалізується через веб-інтерфейс.

Уся система складається з окремих мікросервісів, кожен з яких відповідає за окрему бізнес-область. Це дозволяє розділяти відповідальність між модулями, обмежувати області змін і забезпечувати надійну комунікацію. Система підтримує низку основних варіантів використання, які відображають типові сценарії взаємодії користувача із програмним комплексом. Ці варіанти реалізуються у вигляді окремих модулів або елементів інтерфейсу, які безпосередньо взаємодіють із відповідними мікросервісами через API Gateway. (рис. 2.1).

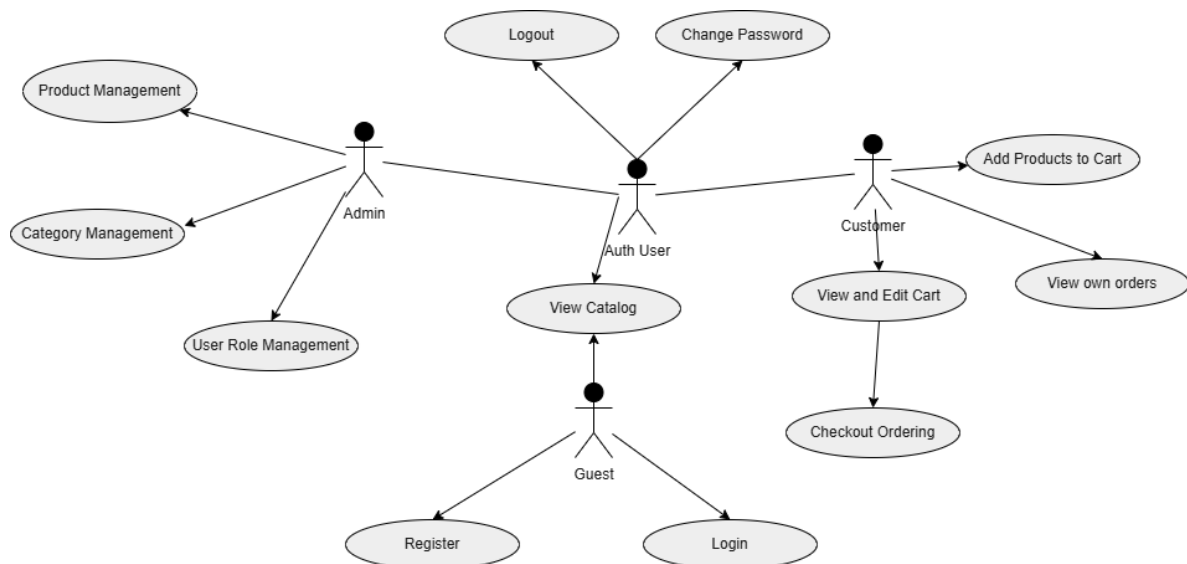


Рис. 2.1. Діаграма варіантів використання

Опис діаграми варіантів використання наведено в таблицях 2.1 – 2.9.

Таблиця 2.1

Опис прецеденту «Вхід/Реєстрація»

Назва варіанту використання	Вхід / Реєстрація
Основні актори	Клієнт, Адміністратор
Короткий опис	Користувач може зареєструвати новий обліковий запис або увійти до наявного, щоб отримати доступ до персоналізованих функцій.

Таблиця 2.2

Опис прецеденту «Перегляд каталогу товарів»

Назва варіанту використання	Перегляд каталогу товарів
Основні актори	Клієнт
Короткий опис	Користувач має змогу переглядати список доступних товарів, відфільтровувати за категоріями або ціною. Інформація надходить від мікросервісу каталогу.

Таблиця 2.3

Опис прецеденту «Додавання товару до кошика»

Назва варіанту використання	Додавання товару до кошика
Основні актори	Клієнт
Короткий опис	Користувач додає обрані товари до кошика. Запис про товар зберігається у Redis на стороні сервісу кошика.

Таблиця 2.4

Опис прецеденту «Перегляд та редагування кошика»

Назва варіанту використання	Перегляд та редагування кошика
Основні актори	Клієнт
Короткий опис	Користувач може переглянути поточний склад кошика, змінити кількість товарів або видалити позиції.

Таблиця 2.5

Опис прецеденту «Оформлення замовлення»

Назва варіанту використання	Оформлення замовлення
Основні актори	Клієнт
Короткий опис	Користувач підтверджує замовлення. Система перевіряє наявність товарів, застосовує знижки (через gRPC до сервісу знижок), формує замовлення та фіксує його в базі даних.

Таблиця 2.6

Опис прецеденту «Управління знижками»

Назва варіанту використання	Управління знижками
Основні актори	Адміністратор
Короткий опис	Адміністратор створює, змінює або видаляє знижки, які застосовуються до товарів.

Таблиця 2.7

Опис прецеденту «Керування каталогом»

Назва варіанту використання	Керування каталогом
Основні актори	Адміністратор
Короткий опис	Доступ до CRUD-операцій над товарами та категоріями у каталозі.

Таблиця 2.8

Опис прецеденту «Перегляд історії замовлень»

Назва варіанту використання	Перегляд історії замовлень
Основні актори	Клієнт
Короткий опис	Користувач може переглядати свої минулі замовлення та їх статуси.

Таблиця 2.9

Опис прецеденту «Управління ролями користувачів»

Назва варіанту використання	Перегляд історії замовлень
Основні актори	Адміністратор
Короткий опис	Можливість змінювати ролі користувачів

З точки зору користувача, будь-яка система повинна бути передусім зручною, швидкою та надійною. Онлайн-магазин електроніки не є винятком. Кінцевий користувач очікує, що інтерфейс буде інтуїтивно зрозумілим, а навігація – простою та логічною. Робота сайту повинна залишатися стабільною навіть за умов великої кількості товарів і запитів. Особливо важливою є швидкість реакції системи в момент оформлення замовлення або оплати, оскільки саме в ці моменти користувач очікує максимальної надійності. Не менш критичним є безпечне зберігання персональних даних – електронної пошти, адреси доставки, платіжної інформації. Користувачі також повинні

мати змогу переглядати історію своїх покупок, відстежувати статуси замовлень та керувати особистими налаштуваннями.

Для адміністратора, який відповідає за наповнення каталогу, облік знижок і обробку замовлень, головним є не лише повний контроль над даними, а й гарантія їхньої цілісності, актуальності та консистентного зберігання. Важливою є також наявність чіткої системи розмежування прав доступу, яка не дозволяє клієнтам втручатися у внутрішню бізнес-логіку або адміністративні функції.

Функціонально система повинна охоплювати повний цикл користувацької взаємодії – від реєстрації й авторизації до оформлення покупки та перегляду її статусу. Після створення облікового запису користувач має отримати доступ до каталогу товарів, який структурується за категоріями й завантажується з бази даних PostgreSQL. Обрані позиції можна додавати до кошика, який зберігається у Redis, що забезпечує високу швидкодію при повторному зверненні. При кожному додаванні товару система перевіряє наявність активних знижок – відповідна логіка реалізована як окремий мікросервіс, з яким взаємодія відбувається синхронно через gRPC.

Оформлення замовлення здійснюється після фінального підтвердження, а всі його параметри зберігаються у базі даних SQL Server. Одночасно з цим система надсилає подію через RabbitMQ, що дозволяє іншим сервісам реагувати на факт створення замовлення без прямого зв'язку. Користувач отримує підтвердження замовлення, а згодом має змогу переглянути його статус.

На додаток до функціональних, система повинна відповідати і низці нефункціональних вимог. Зокрема, продуктивність має залишатися стабільною навіть за умов одночасної роботи великої кількості користувачів. Час відповіді на більшість запитів не повинен перевищувати 500 мілісекунд, а мікросервісна архітектура повинна дозволяти масштабувати кожен компонент окремо – без потреби зупиняти або змінювати інші частини системи. Усі дані передаються захищеним каналом (HTTPS), а механізм авторизації

побудований на базі JWT-токенів, що гарантує контроль доступу без зберігання сесій на сервері.

Безпека персональних даних гарантується не лише шифруванням трафіку, а й правильним розмежуванням баз даних. Кожен мікросервіс має свою окрему модель даних і власне сховище, що знижує ризики витоку чи несанкціонованого доступу до інформації. Усе це разом утворює надійну, стійку до збоїв, масштабовану платформу для ведення електронної торгівлі.

Через специфіку архітектури, де кожен мікросервіс містить власну, незалежну доменну модель, побудова єдиної діаграми класів для всієї системи є недоцільною. Замість цього, для кращого розуміння структури даних рекомендовано створити окремі діаграми класів для ключових компонентів, таких як: каталог товарів, кошик, система знижок та обробка замовлень. Вони дозволять наочно представити, як саме влаштовано кожен із мікросервісів та які об'єкти в них використовуються.

Мікросервіс каталогу товарів відповідає за управління товарною базою, включаючи створення, оновлення, видалення та фільтрацію товарів за категоріями.

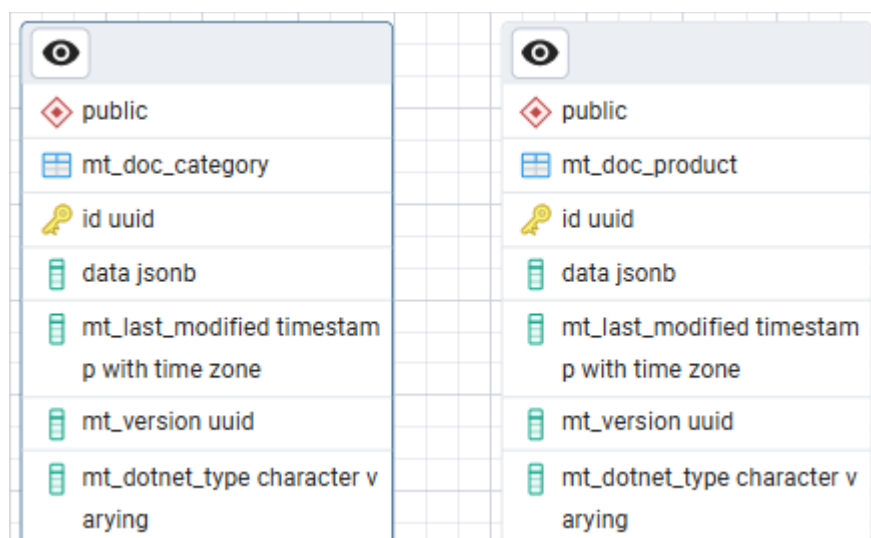


Рис. 2.2. Діаграма класів мікросервісу каталогу товарів

Опишемо діаграму класів (рис. 2.2).

Product – клас який відповідає за товар:

CreateProduct – відповідає за створення товару;

UpdateProduct – відповідає за оновлення товару;
DeleteProduct – відповідає за видалення товару;
GetProducts – відповідає за отримання всіх наявних товарів;
GetProductById – відповідає за отримання товару по ID;
GetProductByCategory – відповідає за отримання товарів в категорії;
Category – клас який відповідає за категорію:
CreateCategory – відповідає за створення категорії;
UpdateCategory – відповідає за оновлення категорії;
DeleteCategory – відповідає за видалення категорії;
GetCategories – відповідає за отримання всіх наявних категорій;
GetCategoryById – відповідає за отримання категорії по ID;

Мікросервіс кошик користувача відповідає за тимчасове зберігання вибраних користувачем товарів перед оформленням замовлення. Його структура зосереджена навколо класу ShoppingCart, який містить набір об'єктів типу ShoppingCartItem, що репрезентують окремі позиції в кошику.

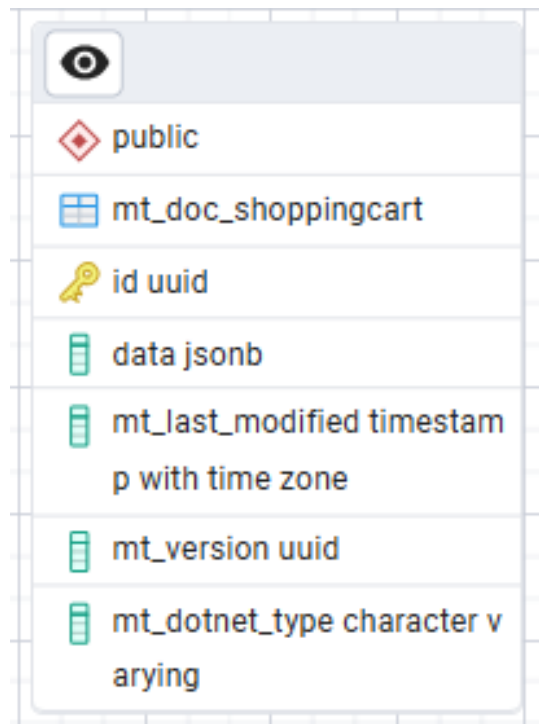


Рис. 2.3. Діаграма класів мікросервісу кошика користувача

Опишемо діаграму класів (рис. 2.3).

ShoppingCart – клас який відповідає за кошик:

GetBasket – відповідає за отримання кошику;

StoreBasket – відповідає за створення та зміну кошику;

DeleteBasket – відповідає за видалення кошику;

CheckoutBasket – відповідає за перевірку кошику перед замовленням.

ShoppingCartItem – клас який відповідає за товар в кошику.

Для реалізації системи знижок використовується окремий сервіс, який відповідає за знижки, у якому основним класом є Coupon. Він містить інформацію про знижкові коди, умови їх застосування та значення знижки.

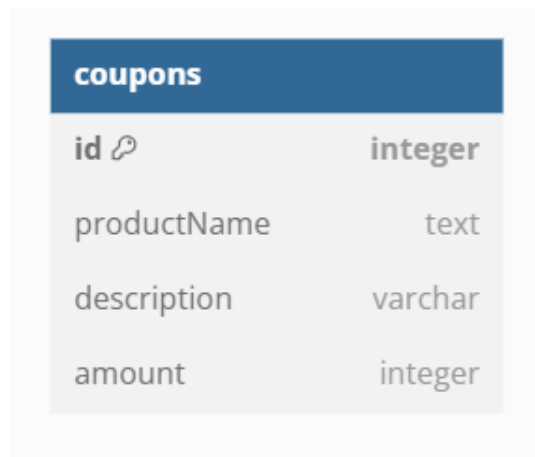


Рис. 2.4. Діаграма класів мікросервісу знижки на товар

Опишемо діаграму класів (рис. 2.4).

Coupon – клас який відповідає за знижки:

CreateDiscount – відповідає за створення знижки;

UpdateDiscount – відповідає за редагування знижки;

DeleteDiscount – відповідає за видалення знижки;

GetDiscounts – відповідає за отримання всіх наявних знижок;

GetDiscountsByName – відповідає за отримання знижки за іменем товару;

Найбільш комплексну структуру має мікросервіс, відповідальний за оформлення та обробку замовлень. Центральним у ньому є клас Order, який агрегує об'єкти типу OrderItem, Product та Customer.

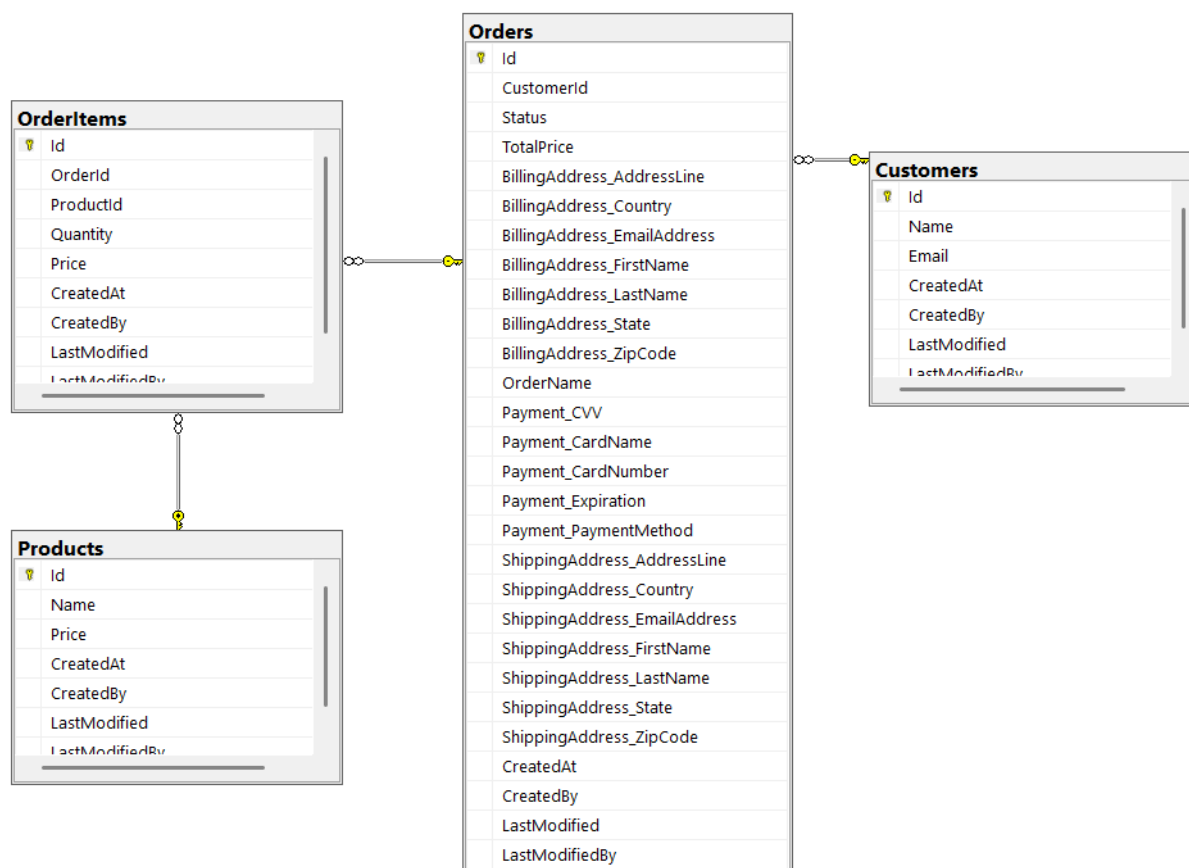


Рис. 2.5. Діаграма класів мікросервісу створення замовлення

Опишемо діаграму класів (рис. 2.5).

Order – клас який відповідає за замовлення:

CreateOrder – відповідає за створення замовлення;

UpdateOrder – відповідає за редагування замовлення;

DeleteOrder – відповідає за видалення замовлення;

GetOrders – відповідає за отримання всіх наявних замовлень;

GetOrdersByCustomer – відповідає за отримання замовлень користувача;

Product – клас який відповідає за одиницю товару в замовленні.

OrderItem – клас який відповідає за елемент в списку замовлення.

Customer – клас який відповідає за клієнта який створює замовлення.

Для реалізації системи авторизації використовується сервіс, який відповідає за авторизацію, у якому основним класом є User. Він містить інформацію про знижкові коди, умови їх застосування та значення знижки.

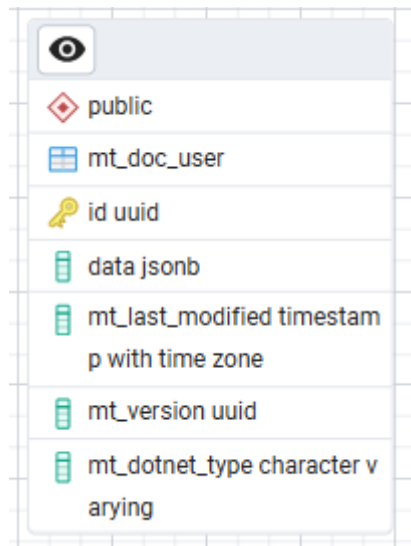


Рис. 2.6. Діаграма класів мікросервісу авторизації

Опишемо діаграму класів (рис. 2.6).

Order – клас який відповідає за замовлення:

Register – відповідає за реєстрацію користувача;

Login – відповідає за вхід користувача;

ChangePassword – відповідає за зміну пароля користувача;

ChangeUserRole – відповідає за зміну ролі користувача;

GetUsers – відповідає за отримання усіх користувачів;

GetUserByEmail – відповідає за отримання користувача за Email;

User – клас який відповідає за дані користувача.

2.2 Опис структур баз даних

У межах реалізації системи використовується мікросервісна архітектура, у якій кожен мікросервіс володіє власною базою даних, що відповідає принципу ізоляції доменів. Це дозволяє кожному сервісу зберігати дані у зручному для себе форматі, забезпечуючи високу модульність, масштабованість та незалежність між сервісами. Таким чином, замість централізованої БД використовується підхід "база даних на мікросервіс", що повністю відповідає сучасним підходам до побудови розподілених систем.

Кожен мікросервіс використовує базу даних, яка найкраще відповідає його завданням:

Мікросервіси авторизації та каталогу товарів працює з PostgreSQL, оскільки це надійна реляційна система з підтримкою складних запитів, транзакцій та розширеної типізації, що дозволяє ефективно оперувати структурами даних користувачів, товарів, категорій, складу.

Мікросервіс кошика використовує Redis як основне сховище через його високу швидкодію при роботі з тимчасовими даними та зручність реалізації кешу. Крім того, для фіксації стабільних даних використовується також PostgreSQL.

Мікросервіс знижок реалізований з використанням SQLite, оскільки обсяг даних невеликий, а сама БД є легкою, вбудованою, і не потребує окремого сервера для розгортання. Це дозволяє зменшити складність обслуговування мікросервісу.

Мікросервіс замовлень працює з Microsoft SQL Server, що забезпечує високу надійність, транзакційну цілісність, гнучкість і потужну підтримку складних зв'язків між таблицями –це критично важливо для обробки замовлень.

Зважаючи на це, структура кожної бази даних суттєво відрізняється і не перетинається з іншими мікросервісами.

База даних системи складається з дев'яти таблиць:

Мікросервіс Catalog: Products, Categories

Мікросервіс Basket: ShoppingCarts

Мікросервіс Discount: Coupons

Мікросервіс Ordering: Orders, OrderItems, Products, Customer

Мікросервіс Auth: Users

Опис кожної з таблиць бази даних наведений у таблицях 2.10 – 2.18.

Таблиця 2.10

Опис таблиці Products (мікросервіс Catalog)

Поле	Тип	Призначення
Id	guid	Ідентифікація
Name	string	Назва
Description	string	Опис
ImageFile	string	Назва файлу зображення
Price	decimal	Ціна
Category	List<guid>	Перелік категорій до яких відноситься товар

Таблиця 2.11

Опис таблиці Category

Поле	Тип	Призначення
Id	guid	Ідентифікація
Name	string	Назва
Description	string	Опис

Таблиця 2.12

Опис таблиці ShoppingCarts

Поле	Тип	Призначення
UserId	guid	Ідентифікація за користувачем
Items	List<ShoppingCartItem>	Перелік товарів в кошику

Таблиця 2.13

Опис таблиці Coupons

Поле	Тип	Призначення
Id	int	Ідентифікація
ProductName	string	Назва товару
Description	string	Опис
Amount	int	Сума знижки в грошових одиницях

Таблиця 2.14

Опис таблиці Orders

Поле	Тип	Призначення
Id	guid	Ідентифікація
CustomerId	guid	Ідентифікація користувача який створив замовлення
OrderName	string	Назва замовлення
ShippingAddress_FirstName	string	Адреса доставки: Ім'я
ShippingAddress_LastName	string	Адреса доставки: Прізвище
ShippingAddress_EmailAddress	string	Адреса доставки: Email
ShippingAddress_AddressLine	string	Адреса доставки: Адреса
ShippingAddress_Country	string	Адреса доставки: Країна
ShippingAddress_State	string	Адреса доставки: Область
ShippingAddress_ZipCode	string	Адреса доставки: Поштовий індекс
BillingAddress_FirstName	string	Платіжна адреса: Ім'я

Продовження таблиці 2.14

Поле	Тип	Призначення
BillingAddress_LastName	string	Платіжна адреса: Прізвище
BillingAddress_EmailAddress	string	Платіжна адреса: Email
BillingAddress_AddressLine	string	Платіжна адреса: Адреса
BillingAddress_Country	string	Платіжна адреса: Країна
BillingAddress_State	string	Платіжна адреса: Область
BillingAddress_ZipCode	string	Платіжна адреса: Поштовий індекс
Payment_CardName	string	Оплата: Ім'я на карті
Payment_CardNumber	string	Оплата: Номер карти
Payment_Expiration	string	Оплата: Термін дії
Payment_CVV	string	Оплата: CVV код
Payment_PaymentMethod	int	Оплата: Метод оплати
OrderStatus	int	Статус замовлення

Таблиця 2.15

Опис таблиці OrderItems

Поле	Тип	Призначення
Id	guid	Ідентифікація
OrderId	guid	Id замовлення
ProductId	guid	Id товару
Quantity	int	Кількість
Price	decimal	Ціна

Таблиця 2.16

Опис таблиці Products (мікросервіс Order)

Поле	Тип	Призначення
Id	guid	Ідентифікація
Name	string	Назва товару
Price	decimal	Ціна

Таблиця 2.17

Опис таблиці Customers

Поле	Тип	Призначення
Id	guid	Ідентифікація
Name	string	Ім'я
Email	string	Email

Таблиця 2.18

Опис таблиці User

Поле	Тип	Призначення
Id	guid	Ідентифікація
Email	string	Email
PasswordHash	string	Пароль
Role	string	Роль

Варто зазначити, що завдяки архітектурі CQRS в частині рідера можуть бути окремі read-моделі, що кешуються або оптимізовані під конкретні запити. Це може відображатися у вигляді додаткових таблиць для денормалізованих даних.

2.3 Опис алгоритмів роботи програми

Функціонування програмного комплексу базується на послідовності взаємопов'язаних дій, що реалізуються у вигляді алгоритмів, які відповідають очікуваній поведінці користувачів системи – як клієнтів, так і адміністраторів. В основі системи лежить мікросервісна архітектура, у якій кожен сервіс відповідає за окремий сегмент бізнес-логіки. Алгоритми реалізовані відповідно до принципів CQRS, DDD та Clean Architecture, що забезпечує гнучкість, модульність і простоту супроводу.

Алгоритми роботи користувачів умовно поділяються залежно від ролі, яку має особа, що входить до системи. Розглянемо типові сценарії для кожного з них. Алгоритм взаємодії клієнта з програмним комплексом. Користувач, який є клієнтом, проходить автентифікацію через мікросервіс авторизації. Після введення облікових даних, система перевіряє їх правильність і генерує JWT токен, який зберігається на клієнтській стороні та використовується для авторизації в подальших запитах. Дії, доступні для клієнта: перегляд списку товарів та детальної інформації про них; сортування за категоріями, додавання товарів до кошика; зміна кількості товарів у кошику або видалення товарів; перегляд загальної вартості товарів у кошику з урахуванням знижок; оформлення замовлення; введення платіжної інформації та здійснення оплати; перегляд історії замовлень із деталізацією кожного з них.

Алгоритм взаємодії адміністратора з програмним комплексом. Після входу в систему адміністратор також проходить автентифікацію. Його JWT токен містить іншу роль, яка дозволяє отримати доступ до розширеного функціоналу. Зокрема, адміністратор має змогу додавати, редагувати або видаляти товари з каталогу, переглядати замовлення клієнтів, вносити зміни в систему знижок.

Дії, доступні для адміністратора: вхід до адміністративної панелі; додавання, редагування, видалення нових товарів; додавання, редагування, видалення категорій, перегляд замовлень конкретних клієнтів; керування

системою знижок, зміна ролі клієнтів. Алгоритм роботи зображено на рисунку 2.7.

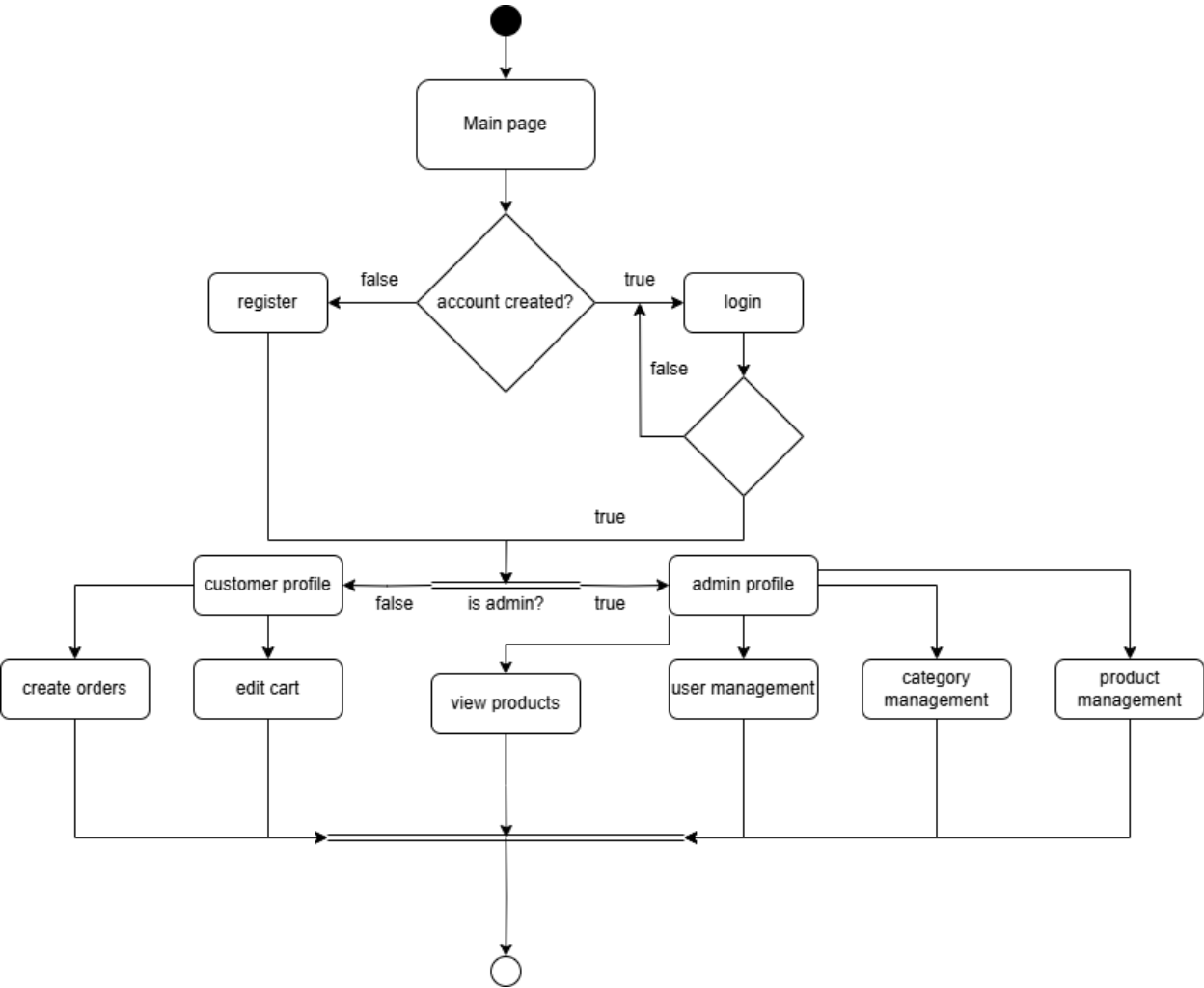


Рис. 2.7. Діаграма активності

Після проведення проєктних робіт переходимо безпосередньо до реалізації запропонованого функціоналу для онлайн-магазину електроніки на платформі .NET.

2.4 Реалізація функціоналу системи онлайн-магазину

Функціонал системи охоплює основні сценарії взаємодії клієнта та адміністратора. У цьому підрозділі наведено опис реалізації основних функціональних можливостей системи.

Авторизація в системі реалізована з використанням JWT-токенів. Після успішної реєстрації користувач отримує токен, який використовується для

подальших запитів до сервісів. Функція авторизації наведена в фрагменті коду:

```
app.MapPost("/users/login",
    async (LoginUserRequest request, IDocumentSession
session, IPasswordHasher hasher, IJwtTokenService jwt) =>
    {
        var user = await session.Query<User>()
            .FirstOrDefaultAsync(x => x.Email ==
request.Email);

        if (user is null ||
!hasher.Verify(user.PasswordHash, request.Password))
            return Results.Unauthorized();

        var token = jwt.GenerateToken(user);
        return Results.Ok(new
LoginUserResponse(token));
    })
```

Користувачі можуть переглядати список доступних товарів. Цей функціонал реалізовано в мікросервісі каталогу товарів. Дані зберігаються у базі PostgreSQL. Пошук, сортування та фільтрація реалізовані за допомогою CQRS з використанням MediatR. Функція отримання товарів наведена в фрагменті коду:

```
public async Task<GetProductsResult> Handle(GetProductsQuery
query, CancellationTokens cancellationTokens)
{
    var products = session.Query<Product>();
    return query is { PageNumber: not null, PageSize: not null
}
        ? new GetProductsResult(await
products.ToPagedListAsync(query.PageNumber ?? 1, query.PageSize ??
10, cancellationTokens))
        : new GetProductsResult(await
products.ToListAsync(cancellationTokens));
}
```

Коли користувач додає товар до кошика, запит обробляється мікросервісом кошика. Кошик реалізовано з використанням Redis для забезпечення швидкого доступу. Всі зміни зберігаються тимчасово до оформлення замовлення. Функція зміни вмісту кошика наведена в фрагменті коду:

```
public async Task<StoreBasketResult> Handle(StoreBasketCommand
command, CancellationTokens cancellationTokens)
```

```

    {
        await DeductDiscount(command.Cart, cancellationToken);

        await repository.StoreBasket(command.Cart,
cancellationToken);

        return new StoreBasketResult(command.Cart.UserId);
    }

```

Під час взаємодії з кошиком здійснюється перевірка наявності знижок. Кошик звертається до мікросервісу знижок через gRPC-запит, отримуючи відповідь із розрахованою знижкою для кожного товару. Знижки зберігаються в SQLite. Функція розрахунку знижки наведена в фрагменті коду:

```

private async Task DeductDiscount(ShoppingCart cart,
Cancellation token cancellationToken)
{
    foreach (var item in cart.Items)
    {
        var coupon = await discountProto.GetDiscountAsync(new
GetDiscountRequest { ProductName = item.ProductName },
cancellationToken: cancellationToken);
        item.Price -= coupon.Amount;
    }
}

```

Коли користувач оформлює замовлення, інформація передається до мікросервісу замовлень, який використовує SQL Server для зберігання даних. Логіка побудована на основі DDD, CQRS і Clean Architecture. У рамках замовлення також реалізовано оплату, яка перевіряється всередині цього ж сервісу. Функція оформлення замовлення наведена в фрагменті коду:

```

private Order CreateNewOrder(OrderDto orderDto)
{
    var shippingAddress =
Address.Of(orderDto.ShippingAddress.FirstName,
orderDto.ShippingAddress.LastName,
orderDto.ShippingAddress.EmailAddress,
orderDto.ShippingAddress.AddressLine,
orderDto.ShippingAddress.Country,
orderDto.ShippingAddress.State, orderDto.ShippingAddress.ZipCode);
}

```

```

        var billingAddress =
Address.Of(orderDto.BillingAddress.FirstName,
orderDto.BillingAddress.LastName,
        orderDto.BillingAddress.EmailAddress,
orderDto.BillingAddress.AddressLine,
        orderDto.BillingAddress.Country,
orderDto.BillingAddress.State, orderDto.BillingAddress.ZipCode);

        var payment = Payment.Of(orderDto.Payment.CardName,
orderDto.Payment.CardNumber, orderDto.Payment.Expiration,
        orderDto.Payment.Cvv, orderDto.Payment.PaymentMethod);

        var newOrder = Order.Create(
            id: OrderId.Of(Guid.NewGuid()),
            customerId: CustomerId.Of(orderDto.CustomerId),
            orderName: OrderName.Of(orderDto.OrderName),
            shippingAddress: shippingAddress,
            billingAddress: billingAddress,
            payment: payment);

        foreach (var orderItemDto in orderDto.OrderItems)
        {
            newOrder.Add(ProductId.Of(orderItemDto.ProductId),
orderItemDto.Quantity, orderItemDto.Price);
        }

        return newOrder;
    }

```

Користувач може переглядати історію своїх замовлень. Реалізовано окрему end-point логіку в мікросервісі замовлень, яка повертає список замовлень, що були створені певним користувачем. Функція отримання історії замовлень наведена в фрагменті коду:

```

public async Task<GetOrdersByCustomerResult>
Handle(GetOrdersByCustomerQuery query, CancellationToken
cancellation_token)
{
    var orders = await dbContext.Orders
        .Include(o => o.OrderItems)
        .AsNoTracking()
        .Where(o => o.CustomerId ==
CustomerId.Of(query.CustomerId))
        .OrderBy(o => o.OrderName.Value)
        .ToListAsync(cancellation_token);

    return new
GetOrdersByCustomerResult(orders.ToOrderDtoList());
}

```

Адміністратор має доступ до розширеного функціоналу –створення, редагування, видалення товарів, а також керування знижками. Цей функціонал реалізований через окремі ендпоінти з перевіркою ролі користувача на рівні авторизації. Функція додавання товару адміністратором наведена в фрагменті коду:

```
public async Task<CreateProductResult> Handle(CreateProductCommand
command, CancellationToken cancellationToken)
{
    var product = new Product
    {
        Name = command.Name,
        Categories = command.Categories,
        Description = command.Description,
        ImageFile = command.ImageFile,
        Price = command.Price,
    };

    session.Store(product);
    await session.SaveChangesAsync(cancellationToken);

    return new CreateProductResult(product.Id);
}
```

Підсумовуючи, було створено діаграму компонентів онлайн-магазину (рис. 2.8) і визначено основні компоненти мікросервісної архітектури. API Gateway (YARP) – центральна точка входу для клієнтських запитів, яка маршрутизує трафік до відповідних мікросервісів. Сервіс авторизації відповідає за реєстрацію, логін, генерацію та перевірку JWT-токенів, розмежування ролей (клієнт, адміністратор). Сервіс каталогу товарів забезпечує отримання, фільтрацію та управління даними про товари. Сервіс кошика зберігає тимчасову інформацію про товари, додані до кошика користувачем, використовує Redis. Сервіс знижок обробляє запити на розрахунок знижок через gRPC, база даних –SQLite.

Сервіс замовлень відповідає за створення та перегляд замовлень, зберігає інформацію у SQL Server, обробляє процес оплати. RabbitMQ це брокер повідомлень, який забезпечує асинхронну подієво-орієнтовану взаємодію між сервісами.

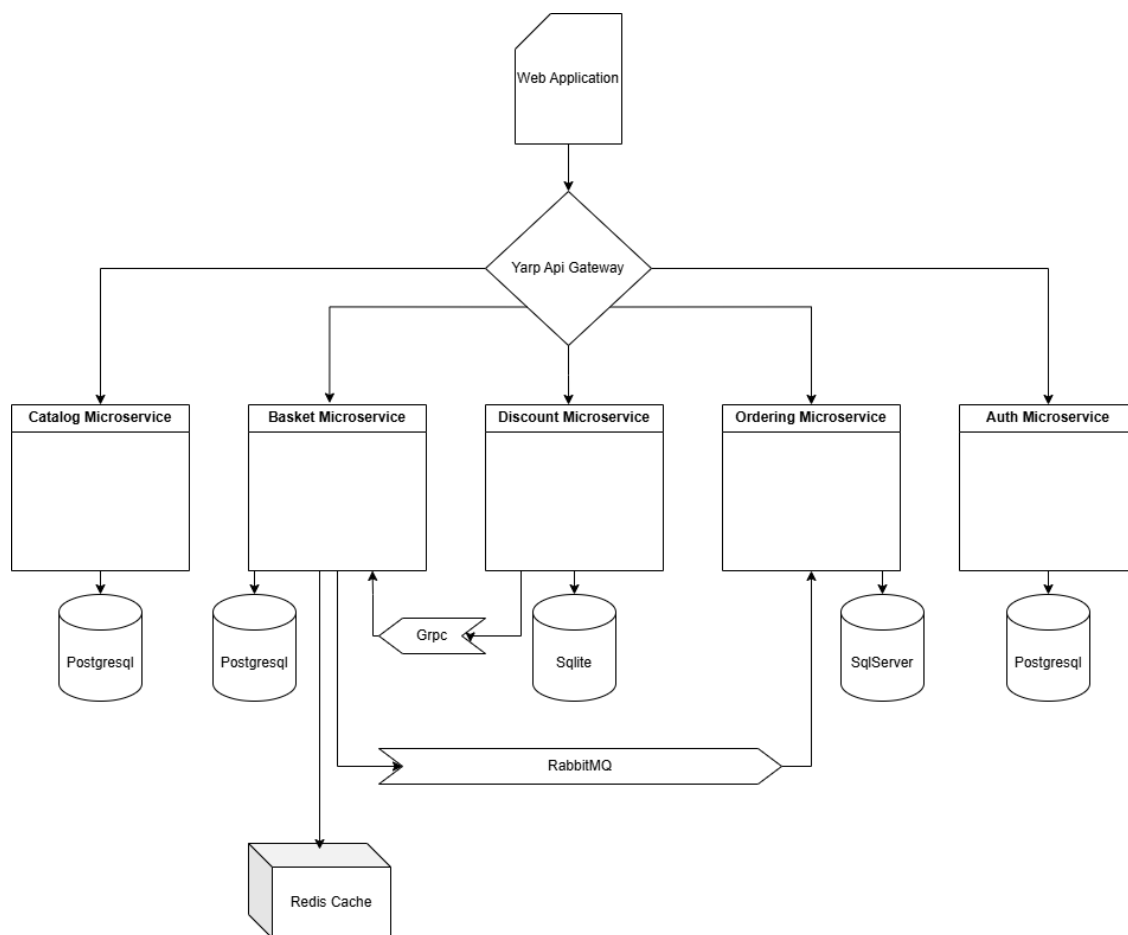


Рис. 2.8. Діаграма компонентів

Клієнтська частина взаємодіє з системою через API Gateway, реалізована на Bootstrap + JavaScript. Бази даних мікросервісів –кожен сервіс має власне ізольоване сховище.

Висновки до другого розділу

У другому розділі було здійснено проектування програмного забезпечення системи онлайн-магазину. Визначено основні варіанти використання, охарактеризовано дії користувачів і сформовано вимоги до системи. Проведено логічне структурування сервісів відповідно до функціональних обов’язків, описано сценарії взаємодії та алгоритми роботи залежно від ролі користувача.

Значну увагу приділено реалізації функціоналу, включно з методами, моделями даних і логікою обробки запитів у межах окремих модулів. Наведені та описані місця для відповідних діаграм дозволяють краще візуалізувати структуру і внутрішні процеси системи.

У підсумку, сформовано цілісне уявлення про проєктну структуру майбутнього програмного комплексу та його основні функціональні компоненти.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ОНЛАЙН-МАГАЗИНОМ ЕЛЕКТРОНІКИ

3.1 Порядок встановлення та налаштування параметрів платформи

Програмний комплекс онлайн-магазину реалізовано як набір мікросервісів, що розгортаються в ізольованих середовищах за допомогою Docker. Це дозволяє створити незалежну, масштабовану і легко переносиму систему, яка не залежить від конкретної операційної системи чи інфраструктури.

Перш ніж здійснювати розгортання системи, необхідно підготувати середовище. Потрібно встановити Docker і Docker Compose останньої стабільної версії, SQL Server, або запустити його як окремий контейнер у Docker, переконатися у наявності PostgreSQL, Redis, RabbitMQ, які також можуть бути підняті у контейнерах або доступні як зовнішні сервіси. Для локального тестування – Visual Studio / VS Code або інше середовище з підтримкою .NET SDK. Рекомендовано мати встановлений Docker Desktop для візуального контролю контейнерів.

Система побудована так, що всі зовнішні залежності – бази даних, брокери повідомлень, reverse проху – також включено до складу контейнерної інфраструктури. Таким чином, достатньо одного конфігураційного файлу docker-compose.yml, щоб запустити повноцінне середовище розробки, тестування або демонстрації. Для запуску всієї системи достатньо виконати команду: `docker-compose up --build -d`

Діаграму розгортання програмного комплексу зображено на рисунку 3.1. Також передбачено можливість зміни конфігурації портів, з'єднань до баз даних та рівня логування через файли конфігурації, що доступні для кожного окремого сервісу. Завдяки використанню Docker-інфраструктури, система легко переноситься, швидко розгортається і може бути адаптована до будь-якого середовища, що підтримує контейнеризацію.

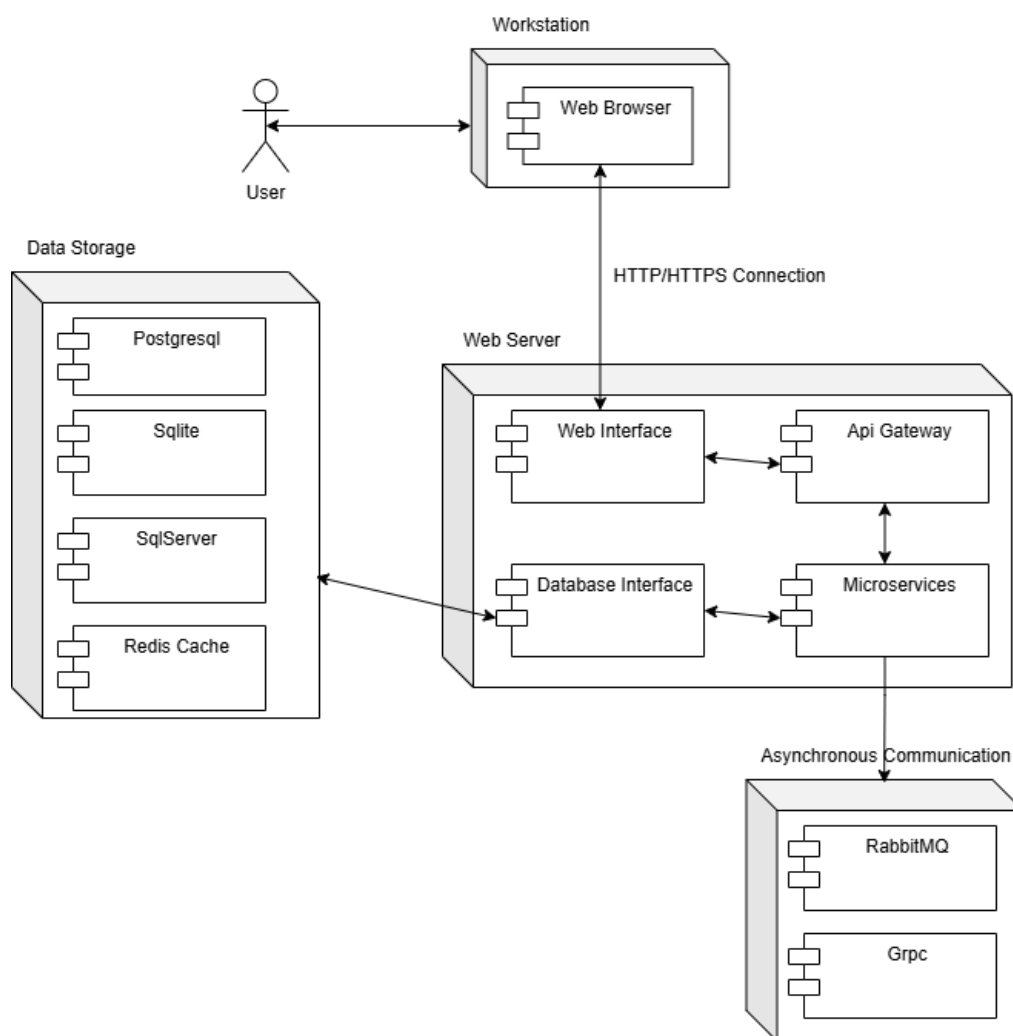


Рис. 3.1. Діаграма розгортання

Таким чином, для запуску системи не потрібна ручна інсталяція кожного модуля чи бази даних. Усі сервіси конфігуровано у Docker-оточенні, що дозволяє швидко розгорнути інфраструктуру у будь-якому середовищі з підтримкою контейнеризації. Такий підхід також дає змогу легко масштабувати будь-який сервіс, при потребі дублюючи його екземпляри або розділяючи навантаження.

3.2 Структура інтерфейсу онлайн-магазину електроніки

Інтерфейс системи онлайн-магазину побудований з урахуванням зручності, інтуїтивності та адаптивності. Він дає змогу як звичайному користувачеві, так і адміністратору ефективно взаємодіяти з системою, не

потребуючи додаткових інструкцій. Користувач має змогу бачити лише той функціонал, який відповідає його ролі та статусу авторизації.

Після переходу на головну сторінку користувач бачить промоційний слайдер, що містить рекламні банери або акційні пропозиції. Нижче відображаються чотири останні додані товари та чотири найпопулярніші (за кількістю покупок або оцінками), а також добірка топ-товару, який автоматично підтягується з сервера (рис. 3.2 – рис. 3.3).

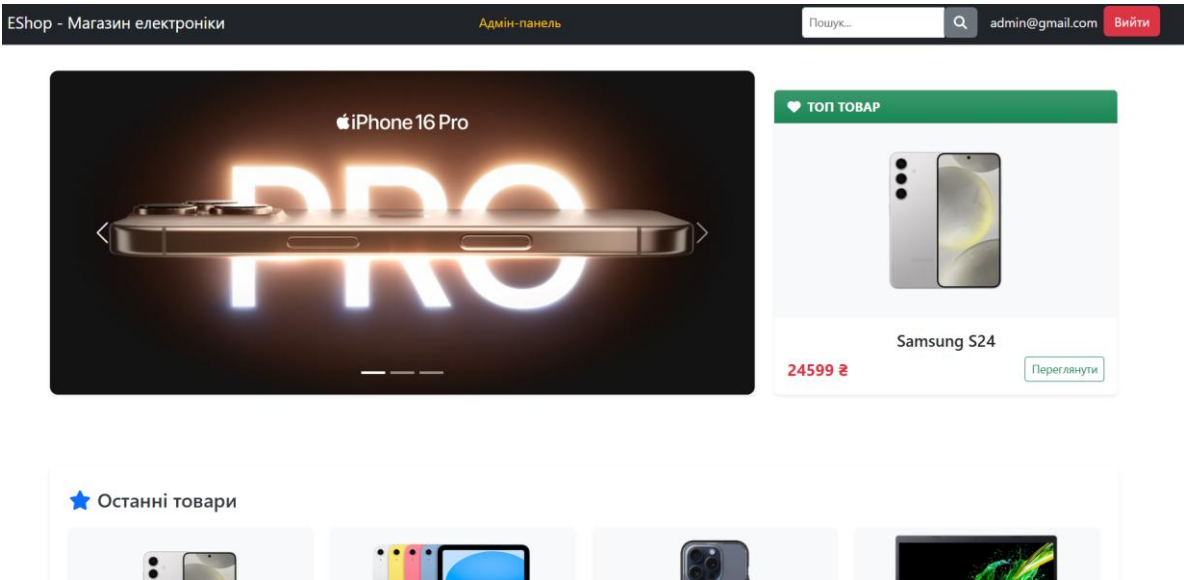


Рис. 3.2. Головна сторінка – верх

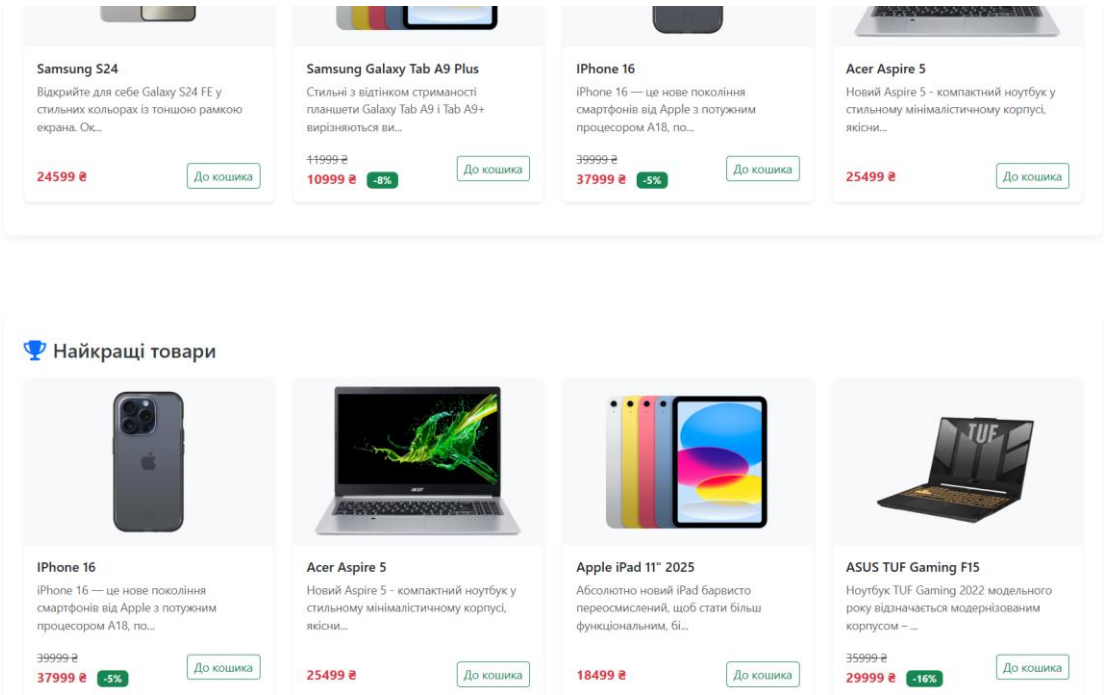


Рис. 3.3. Головна сторінка – низ

На всіх сторінках інтерфейс має фіксовану шапку, яка змінюється залежно від статусу користувача. Якщо користувач не авторизований, у шапці відображаються посилання «Головна», «Товари», «Контакти», поле пошуку, а також кнопки «Увійти» та «Реєстрація». Це забезпечує доступ до базових функцій сайту без входу в систему.

На сторінці товарів користувач бачить список наявних продуктів. Список реалізований з пагінацією – на кожній сторінці відображається до дев'яти товарів. Зліва розташований список категорій, який дає змогу відфільтрувати товари за категоріями. Також можна побачити позначку «Новий товар» під списком категорій.

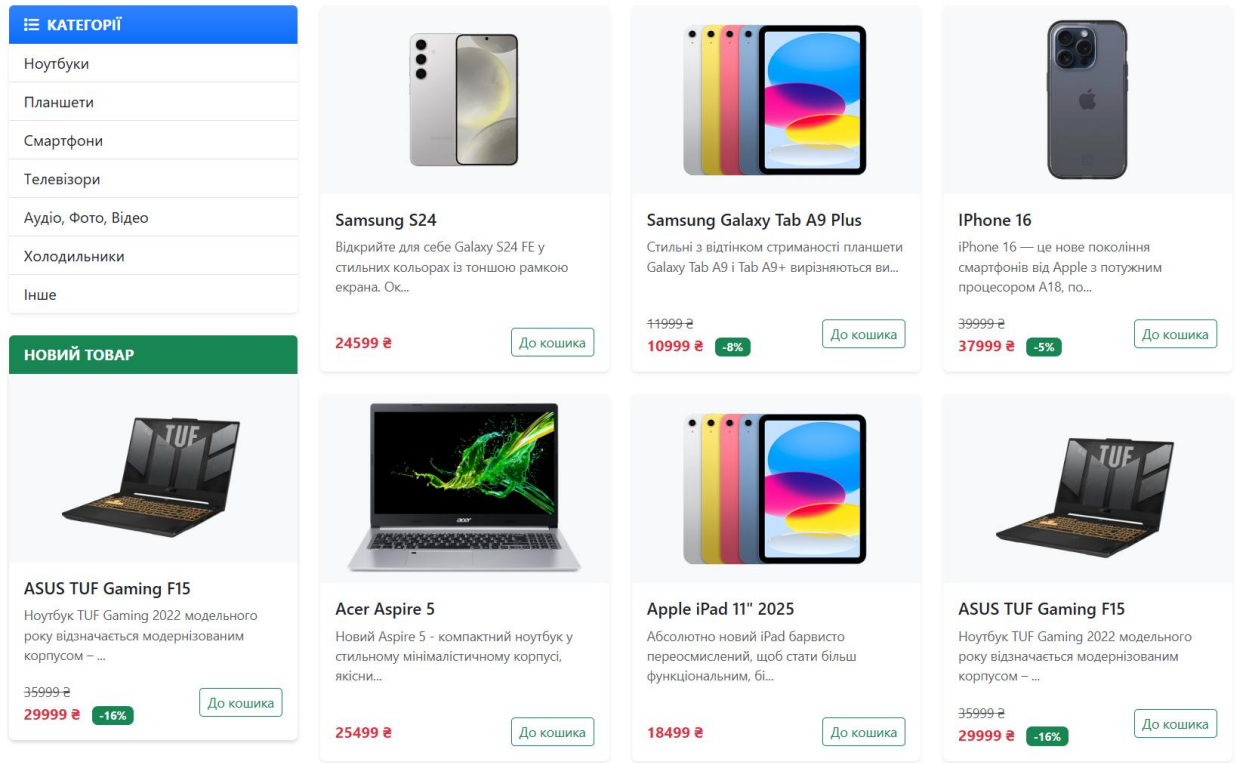


Рис. 3.4. Сторінка товарів

Переходячи до сторінки окремого товару, користувач отримує повну інформацію про продукт: назву, опис, ціну, доступні кольори, кількість, а також зображення. Є можливість обрати бажані параметри товару та додати його до кошика (рис. 3.5).



Samsung S24

24599 ₴

Колір

Чорний

Кількість

-

1

+

ДОДАТИ ДО КОШИКА

Оплата при отриманні

Онлайн оплата

+38 098 111 22 33

ОПИС ТОВАРУ

Відкрийте для себе Galaxy S24 FE у стильних кольорах із тоншою рамкою екрана. Округла форма забезпечує зручне використання, а плоский дизайн підвищує задоволення від перегляду.

Рис. 3.5. Сторінка товару

Якщо неавторизований користувач спробує натиснути кнопку «Додати до кошика», його буде автоматично перенаправлено на сторінку входу до системи (рис. 3.6).

Вхід

Email

Пароль

Увійти

Рис. 3.6. Форма логіну

Також користувач має змогу зареєструватись, натиснувши на відповідну кнопку (рис. 3.7).

Перехід у кошик дає змогу переглянути всі додані товари, змінити кількість, видалити непотрібні позиції (рис. 3.9).

[Головна](#) / Кошик

Зображення	Назва	Кількість	Ціна за одиницю	Сума	
	Samsung Galaxy Tab A9 Plus	3	8999 ₪	26997 ₪	
	iPhone 16	2	35999 ₪	107997 ₪	
Загальна сума:				134994 ₪	

[← Продовжити покупки](#)[ОФОРМИТИ ЗАМОВЛЕННЯ ✓](#)

Рис. 3.9. Кошик товарів

Під списком товарів розміщена кнопка «Оформити замовлення», натискання на яку відкриває форму з полями для введення контактних даних, адреси доставки та підтвердження покупки (рис. 3.10).

Ваш кошик

2

Samsung Galaxy Tab A9 Plus

8999 ₪

x 3

iPhone 16

35999 ₪

x 3

Загалом

134994 ₪

Деталі замовлення

Ім'я

Прізвище

Email (необов'язково)

you@example.com

Адреса

вул. Хрещатик, 1

Додаткова адреса (необов'язково)

Квартира або офіс

Країна

Оберіть...

Область

Оберіть область

Індекс

Рис. 3.10. Сторінка оформлення замовлення

При вході з роллю «Адміністратор» у шапці з'являється кнопка «Адмін-панель» (рис. 3.11).

EShop - Магазин електроніки

Адмін-панель



admin@gmail.com

Вийти

Рис. 3.11. Адмін-панель в шапці сторінки

Після переходу відкривається сторінка з трьома кнопками: «Керування товарами», «Керування категоріями», «Керування користувачами» (рис. 3.12).

Адмін-панель



Рис. 3.12. Адмін-панель

У розділі керування товарами адміністратор може шукати товари, створювати нові, редагувати існуючі, видаляти непотрібні. Також є можливість вказати або зняти знижку на конкретний товар (рис 3.13 – рис. 3.16).

Управління товарами

[+ Додати товар](#)

Пошук за назвою...

Пошук

Назва	Ціна	Категорія	Знижка	Дії
Acer Aspire 5	25499.00 ₴	Ноутбуки	—	  
Apple iPad 11" 2025	18499.00 ₴	Планшети	—	  
ASUS TUF Gaming F15	35999.00 ₴	Ноутбуки	-6000 ₴	  
iPhone 16	39999.00 ₴	Смартфони	-2000 ₴	  
Samsung Galaxy Tab A9 Plus	11999.00 ₴	Планшети	-1000 ₴	  
Samsung S24	24599.00 ₴	Смартфони	—	  

Рис. 3.13. Сторінка управління товарами

Додати товар

Назва товару

Введіть назву товару

Опис

Опишіть товар...

Ціна (₴)

0.00

Зображення

Виберите файл Файл не выбран

Обрати категорії

Створити Назад

Рис. 3.14. Додавання товару

Керування знижкою



Сума знижки

Скасувати

Зберегти

Рис. 3.15. Оновлення знижки на товар

Підтвердити видалення



Ви впевнені, що хочете видалити товар **ASUS TUF Gaming F15?**

Скасувати

Видалити

Рис. 3.16. Видалення товару

У розділі керування категоріями можна створювати, редагувати та видаляти категорії товарів.

У керуванні користувачами реалізовано пошук за email, перегляд замовлень конкретного користувача та можливість змінювати його роль (рис 3.17 – 3.18).

Управління користувачами

Пошук за поштою...			Пошук
Email	Роль	Дії	
admin@gmail.com	Admin		
customer@gmail.com	Customer		

Рис. 3.17. Сторінка управління користувачами

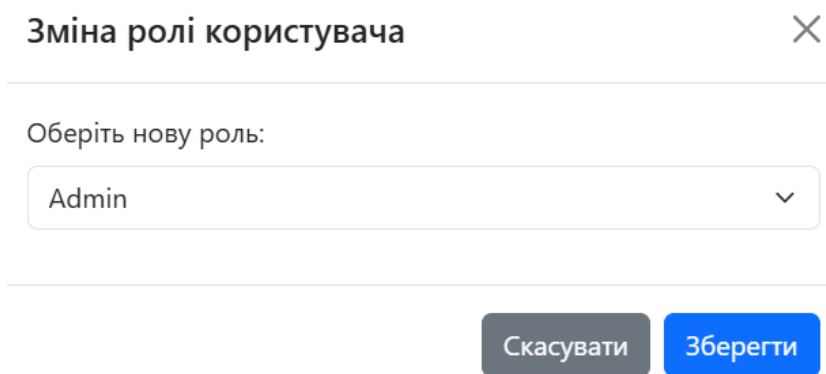


Рис. 3.18. Зміна ролі користувача

Інтерфейс системи динамічно реагує на зміну стану користувача, забезпечує швидкий доступ до ключових функцій та побудований відповідно до сучасних принципів UI/UX-дизайну, що сприяє комфортній роботі з платформою як для звичайних користувачів, так і для адміністраторів.

3.3 Тестування роботи програмного комплексу

Тестування системи проводилось з метою перевірки коректності реалізації основного функціоналу та відповідності поведінки програмного комплексу очікуваним результатам. Перевірка здійснювалася вручну, за допомогою інструменту Postman, а також у процесі взаємодії з інтерфейсом системи у веббраузері.

Оскільки система має мікросервісну структуру та чітко визначені HTTP API, було обрано ключові ендпоінти, які охоплюють основні бізнес-сценарії: реєстрація, авторизація, робота з товарами, кошиком і замовленнями.

Більшість запитів до захищених маршрутів вимагає передачі Bearer Token, який отримується після авторизації. Без токена або з недійсним токеном користувач отримає помилку 401 Unauthorized.

Ендпоінт реєстрації дозволяє створити нового користувача, передаючи email та пароль. Після успішної реєстрації користувач отримує свій Id та може авторизуватися (рис 3.19).

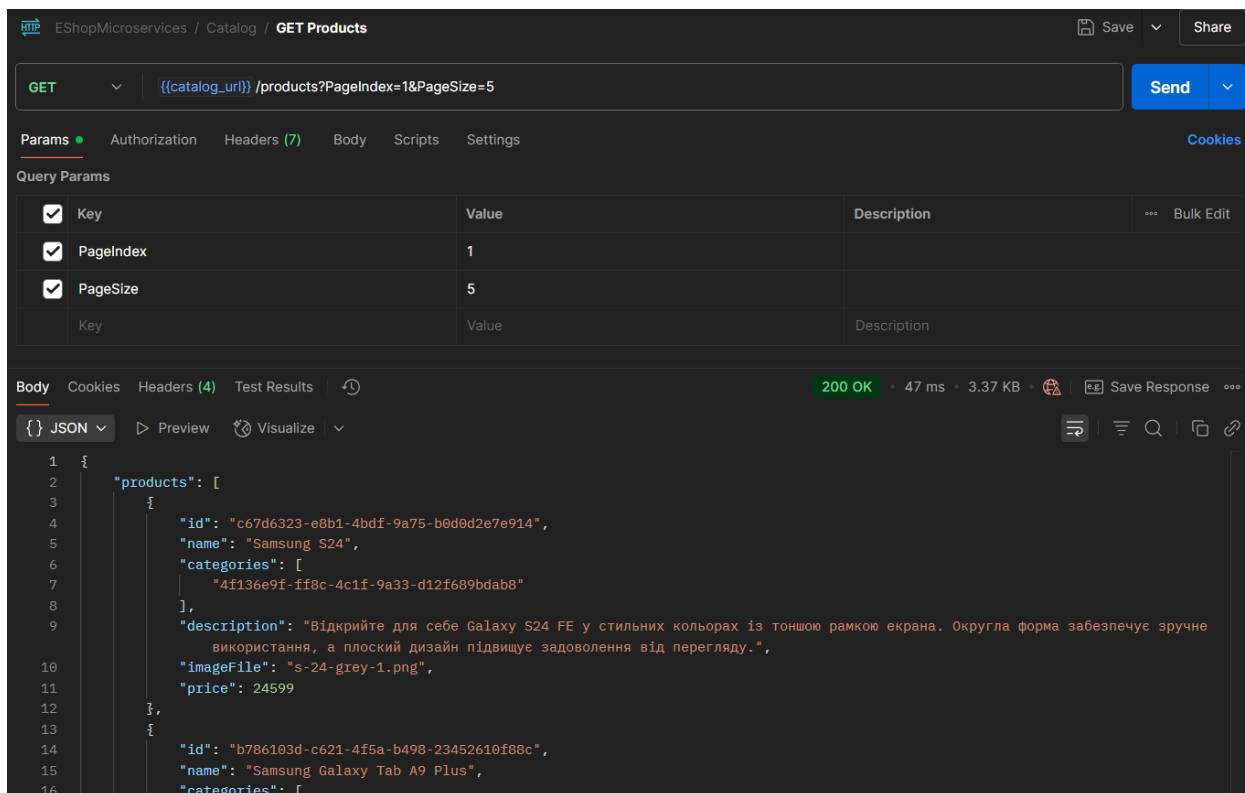


Рис. 3.21. Тестування ендпоінта отримання товарів з пагінацією

Ендпоінт створення нового товару доступний лише для користувачів із роллю адміністратора. Необхідно передати JWT у заголовок. Відповіддю є Id створеного товару (рис. 3.22).

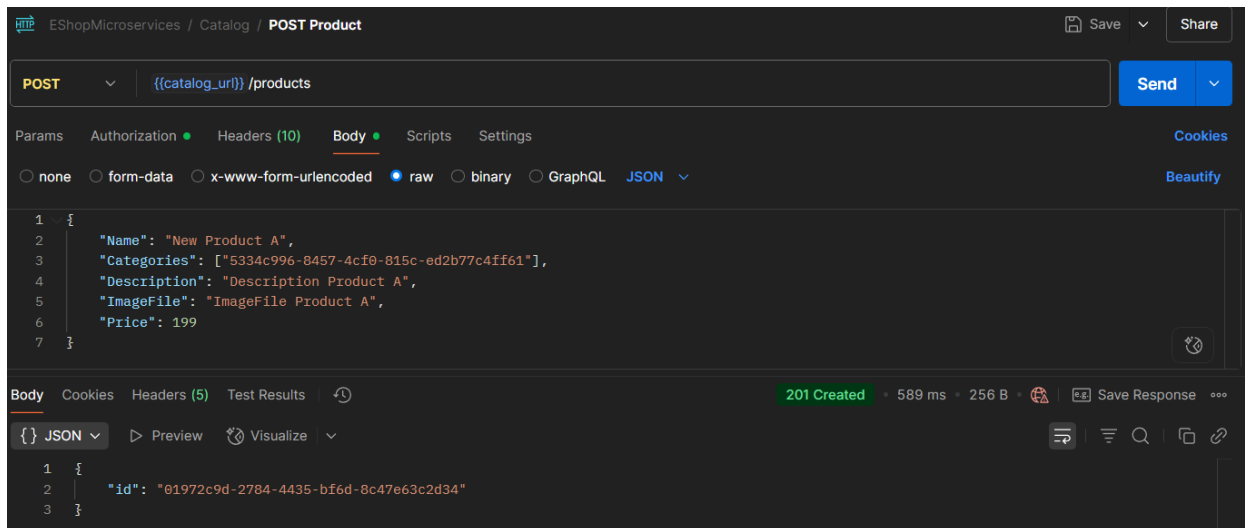


Рис. 3.22. Тестування ендпоінта створення нового товару

Ендпоінт збереження кошика дозволяє зберегти поточний стан кошика. Якщо товар уже в кошику – його кількість оновлюється. Якщо кошик існує –

він оновиться, якщо ні – буде створений. Відповіддю є Id користувача, кошик якого було збережено. Запит потребує авторизації (рис. 3.23).

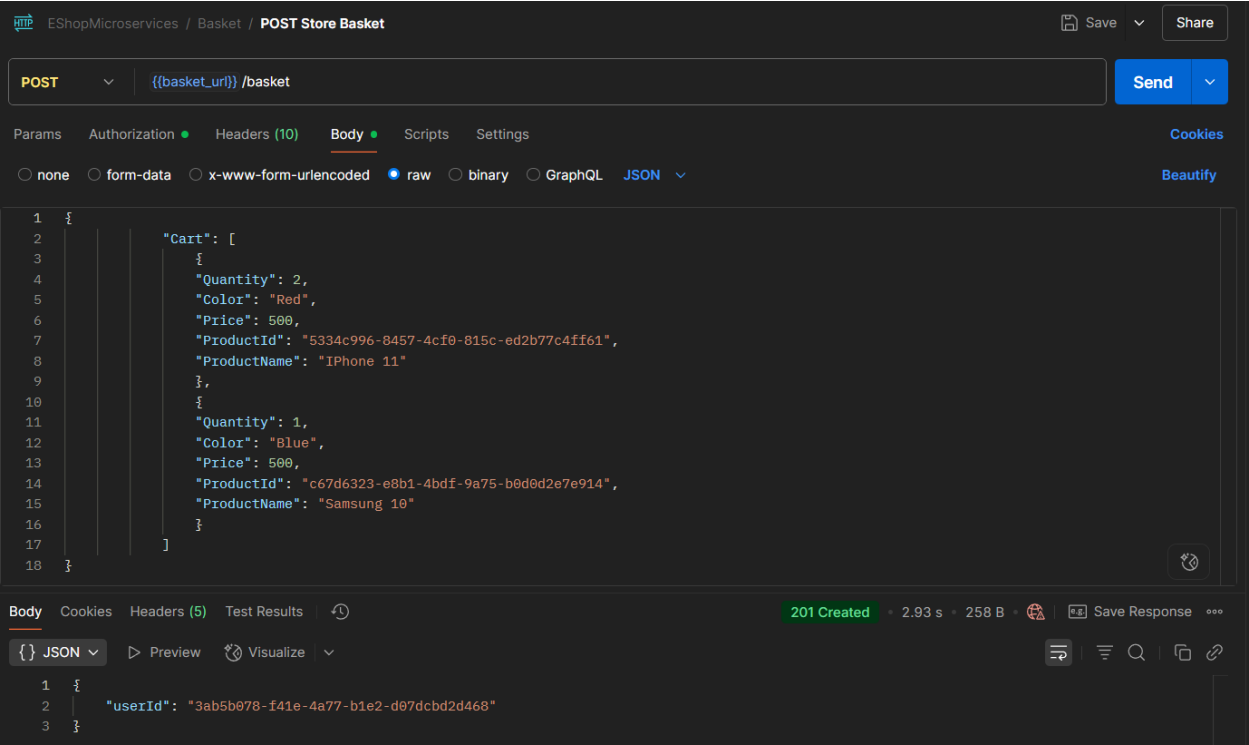


Рис. 3.23. Тестування ендпоінта збереження кошика

Усі протестовані ендпоінти функціонували відповідно до очікувань. Запити з правильними токенами успішно оброблювались, запити без авторизації або з недостатніми правами повертали статус 401 або 403, що підтверджує коректність реалізації механізму доступу.

Додатково було проведено ручне тестування взаємодії з інтерфейсом сайту у браузері, включно з перевіркою переходів, форм, повідомлень про помилки та поведінки при різних ролях користувача. Система коректно реагує на дії користувача, дотримується вимог валідації та обмежень доступу.

Висновки до третього розділу

У цьому розділі було розглянуто практичні аспекти розгортання, використання та тестування програмного комплексу онлайн-магазину. Запропоновано повноцінний порядок встановлення системи, що базується на сучасному підході до контейнеризації та автоматичного розгортання сервісів.

Це дозволяє швидко і зручно налаштовувати середовище як для розробки, так і для подальшої експлуатації.

Було проаналізовано структуру інтерфейсу користувача, що адаптується відповідно до ролі та стану авторизації. Описано, яким чином організовано навігацію між основними розділами сайту, як взаємодіє користувач із функціоналом. Це демонструє завершеність і продуманість побудови фронтенд-частини, а також ефективність інтеграції з мікросервісами бекенду.

У підрозділі тестування описано перевірку ключових ендпоінтів. Результати тестування підтвердили стабільність роботи системи, правильність обробки запитів та відповідність механізмів автентифікації й авторизації очікуваній поведінці.

Результати практичної реалізації підтверджують доцільність обраної архітектури, правильність розроблених алгоритмів і готовність системи до подальшого розгортання та використання.

ВИСНОВКИ

У межах дипломної роботи була досягнута основна мета –розроблено мікросервісну архітектуру для онлайн-магазину електроніки з використанням технологій .NET, яка відповідає сучасним вимогам до масштабованості, гнучкості та надійності.

1. У результаті проведеного аналізу предметної області та дослідження актуальних архітектурних рішень вдалося сформулювати оптимальний підхід до побудови системи, що поєднує в собі переваги розподілених сервісів, асинхронної взаємодії та незалежного масштабування кожного компонента.

2. У процесі реалізації було створено кілька незалежних мікросервісів, кожен з яких виконує чітко визначену бізнес-функції. Для зберігання даних застосовано відповідні СУБД залежно від особливостей конкретного сервісу. Особливу увагу приділено організації взаємодії між сервісами через асинхронну подієву комунікацію. Це дозволило досягти високого рівня стійкості та ізоляції компонентів.

3. Завдяки впровадженню шаблонів проєктування код кожного сервісу залишився чистим, структурованим і зручним для супроводу. Створений функціонал покриває базові сценарії взаємодії користувачів з системою.

4. Практичне значення розробленого рішення полягає в його готовності до подальшого впровадження в реальні бізнес-процеси. Система може бути основою для розгортання комерційної платформи з можливістю масштабування, інтеграції з іншими сервісами –як внутрішніми, так і сторонніми, –та поступового розширення функціоналу без порушення роботи всієї платформи. Це робить створене рішення актуальним не лише з навчальної, а й з прикладної точки зору.

Можливим напрямом подальших досліджень є вдосконалення механізмів аналітики та персоналізації пропозицій, розробка мобільної версії клієнта, впровадження системи сповіщень у реальному часі та використання хмарних рішень для автоматичного масштабування сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Allo.ua – інтернет-магазин техніки [Електронний ресурс]. – Режим доступу: <https://allo.ua>
2. Bootstrap. The most popular HTML, CSS, and JS library in the world [Електронний ресурс]. – Режим доступу: <https://getbootstrap.com>
3. Comfy.ua – інтернет-магазин побутової техніки [Електронний ресурс]. – Режим доступу: <https://comfy.ua>
4. Docker Documentation [Електронний ресурс]. – Режим доступу: <https://docs.docker.com>
5. Docker Compose Overview [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/compose>
6. Eldorado.ua – інтернет-магазин електроніки та техніки [Електронний ресурс]. – Режим доступу: <https://eldorado.ua>
7. Foxtrot.com.ua – інтернет-магазин електроніки [Електронний ресурс]. – Режим доступу: <https://foxtrot.com.ua>
8. gRPC: A high-performance, open-source universal RPC framework [Електронний ресурс]. – Режим доступу: <https://grpc.io>
9. JavaScript | MDN Web Docs [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
10. Kubernetes Documentation [Електронний ресурс]. – Режим доступу: <https://kubernetes.io/docs>
11. MassTransit – Distributed Application Framework for .NET [Електронний ресурс]. – Режим доступу: <https://masstransit.io>
12. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs>
13. RabbitMQ Documentation [Електронний ресурс]. – Режим доступу: <https://www.rabbitmq.com/documentation.html>
14. Redis Documentation [Електронний ресурс]. – Режим доступу: <https://redis.io/docs>

15. Rozetka.com.ua – інтернет-супермаркет електроніки та побутової техніки [Електронний ресурс]. – Режим доступу: <https://rozetka.com.ua>
16. Serilog – Simple .NET logging with fully-structured events [Електронний ресурс]. – Режим доступу: <https://serilog.net>
17. SQLite Documentation [Електронний ресурс]. – Режим доступу: <https://sqlite.org/docs.html>
18. SQL Server Documentation – Microsoft Learn [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/sql/sql-server>
19. YARP: Yet Another Reverse Proxy [Електронний ресурс]. – Режим доступу: <https://microsoft.github.io/reverse-proxy>
20. .NET Health Checks Documentation – Microsoft [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/health-checks>

ДОДАТКИ

Реалізація оберненого проксі-сервера:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ReverseProxy": {
    "Routes": {
      "auth-route": {
        "ClusterId": "auth-cluster",
        "Match": {
          "Path": "/auth-service/{**catch-all}"
        },
        "Transforms": [ { "PathPattern": "{**catch-all}" } ]
      },
      "catalog-route": {
        "ClusterId": "catalog-cluster",
        "Match": {
          "Path": "/catalog-service/{**catch-all}"
        },
        "Transforms": [ { "PathPattern": "{**catch-all}" } ]
      },
      "basket-route": {
        "ClusterId": "basket-cluster",
        "Match": {
          "Path": "/basket-service/{**catch-all}"
        },
        "Transforms": [ { "PathPattern": "{**catch-all}" } ]
      },
      "ordering-route": {
        "ClusterId": "ordering-cluster",
        "RateLimiterPolicy": "fixed",
        "Match": {
          "Path": "/ordering-service/{**catch-all}"
        },
        "Transforms": [ { "PathPattern": "{**catch-all}" } ]
      },
      "discount-route": {
        "ClusterId": "discount-cluster",
        "RateLimiterPolicy": "fixed",
        "Match": {
          "Path": "/discount-service/{**catch-all}"
        },
        "Transforms": [ { "PathPattern": "{**catch-all}" } ]
      }
    },
    "Clusters": {
```

```

    "auth-cluster": {
      "Destinations": {
        "destination1": {
          "Address": "http://auth.api:8080"
        }
      }
    },
    "catalog-cluster": {
      "Destinations": {
        "destination1": {
          "Address": "http://catalog.api:8080"
        }
      }
    },
    "basket-cluster": {
      "Destinations": {
        "destination1": {
          "Address": "http://basket.api:8080"
        }
      }
    },
    "ordering-cluster": {
      "Destinations": {
        "destination1": {
          "Address": "http://ordering.api:8080"
        }
      }
    },
    "discount-cluster": {
      "Destinations": {
        "destination1": {
          "Address": "http://discount.api:8080"
        }
      }
    }
  }
}

```

Реалізація файлу docker-compose.yml:

```
version: '3.4'
```

```
services:
```

```
  catalogdb:
```

```
    image: postgres
```

```
  basketdb:
```

```
    image: postgres
```

```
  authdb:
```

```
    image: postgres
```

```
  distributedcache:
```

```
    image: redis
```

```
  orderdb:
```

```
    image: mcr.microsoft.com/mssql/server
```

```
  messagebroker:
```

```
    image: rabbitmq:management
```

```
  catalog.api:
```

```
    image: ${DOCKER_REGISTRY-}catalogapi
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Services/Catalog/Catalog.API/Dockerfile
```

```
  basket.api:
```

```
    image: ${DOCKER_REGISTRY-}basketapi
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Services/Basket/Basket.API/Dockerfile
```

```
  auth.api:
```

```
    image: ${DOCKER_REGISTRY-}authapi
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Services/Auth/Auth.API/Dockerfile
```

```
  discount.grpc:
```

```
    image: ${DOCKER_REGISTRY-}discountgrpc
```

```
    build:
```

```
      context: .
```

```
      dockerfile: Services/Discount/Discount.Grpc/Dockerfile
```

```
  discount.api:
```

```
    image: ${DOCKER_REGISTRY-}discountapi
```

```
    build:
```

```
      context: .
```

					ІПЗ.КР.Б – 121 – 25 – ПЗ	Арк.
						67

```

    dockerfile: Services/Discount/Discount.API/Dockerfile

ordering.api:
  image: ${DOCKER_REGISTRY-}orderingapi
  build:
    context: .
    dockerfile: Services/Ordering/Ordering.API/Dockerfile

yarpapigateway:
  image: ${DOCKER_REGISTRY-}yarpapigateway
  build:
    context: .
    dockerfile: ApiGateways/YarpApiGateway/Dockerfile

shopping.web:
  image: ${DOCKER_REGISTRY-}shoppingweb
  build:
    context: .
    dockerfile: WebApps/Shopping.Web/Dockerfile

volumes:
  postgres_catalog:
  postgres_basket:
  postgres_auth:

```

Реалізація файлу docker-compose.override.yml:
version: '3.4'

```

services:
  catalogdb:
    container_name: catalogdb
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=CatalogDb
    restart: always
    ports:
      - "5432:5432"
    volumes:
      - postgres_catalog:/var/lib/postgresql/data/

  basketdb:
    container_name: basketdb
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=BasketDb
    restart: always
    ports:
      - "5433:5432"
    volumes:
      - postgres_basket:/var/lib/postgresql/data/

  authdb:
    container_name: authdb
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=AuthDb
    restart: always
    ports:
      - "5434:5432"
    volumes:
      - postgres_basket:/var/lib/postgresql/data/

  distributedcache:
    container_name: distributedcache
    restart: always
    ports:
      - "6379:6379"

  orderdb:
    container_name: orderdb
    environment:
      - ACCEPT_EULA=Y

```

```

    - SA_PASSWORD=SwN12345678
restart: always
ports:
  - "1433:1433"

messagebroker:
  container_name: messagebroker
  hostname: ecommerce-mq
  environment:
    - RABBITMQ_DEFAULT_USER=guest
    - RABBITMQ_DEFAULT_PASS=guest
  restart: always
  ports:
    - "5672:5672"
    - "15672:15672"

catalog.api:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    -
ConnectionString__Database=Server=catalogdb;Port=5432;Database=CatalogDb;User Id=postgres;Password=postgres;Include Error Detail=true
  depends_on:
    - catalogdb
  ports:
    - "6000:8080"
    - "6060:8081"
  volumes:
    -
    ${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:ro
    - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

basket.api:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    -
ConnectionString__Database=Server=basketdb;Port=5432;Database=BasketDb;User Id=postgres;Password=postgres;Include Error Detail=true
    - ConnectionStrings__Redis=distributedcache:6379
    - GrpcSettings__DiscountUrl=https://discount.grpc:8081
    - MessageBroker__Host=amqp://ecommerce-mq:5672
    - MessageBroker__UserName=guest
    - MessageBroker__Password=guest
  depends_on:
    - basketdb
    - distributedcache

```

```

    - discount.grpc
    - messagebroker
  ports:
    - "6001:8080"
    - "6061:8081"
  volumes:
    -
    ${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro
    - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

discount.grpc:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    - ConnectionStrings__Database=Data Source=discountdb
  ports:
    - "6002:8080"
    - "6062:8081"
  volumes:
    -
    ${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro
    - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

discount.api:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    - GrpcSettings__DiscountUrl=https://discount.grpc:8081
  depends_on:
    - discount.grpc
  ports:
    - "6007:8080"
    - "6067:8081"
  volumes:
    -
    ${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro
    - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

ordering.api:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    -
    ConnectionStrings__Database=Server=orderdb;Database=OrderDb;User
Id=sa;Password=SwN12345678;Encrypt=False;TrustServerCertificate=Tr
ue

```

```

    - MessageBroker__Host=amqp://ecommerce-mq:5672
    - MessageBroker__UserName=guest
    - MessageBroker__Password=guest
    - FeatureManagement__OrderFullfilment=false
depends_on:
  - orderdb
  - messagebroker
ports:
  - "6003:8080"
  - "6063:8081"
volumes:
  -
${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro
  - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

yarpapigateway:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
  depends_on:
    - catalog.api
    - basket.api
    - ordering.api
    - auth.api
  ports:
    - "6004:8080"
    - "6064:8081"
  volumes:
    -
${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro
  - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

auth.api:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    -
ConnectionString__Database=Server=authdb;Port=5432;Database=AuthD
b;User Id=postgres;Password=postgres;Include Error Detail=true
  depends_on:
    - authdb
  ports:
    - "6006:8080"
    - "6066:8081"
  volumes:
    -
${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro

```

```

- ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

shopping.web:
  environment:
    - ASPNETCORE_ENVIRONMENT=Development
    - ASPNETCORE_HTTP_PORTS=8080
    - ASPNETCORE_HTTPS_PORTS=8081
    - ApiSettings__GatewayAddress=http://yarpapigateway:8080
  depends_on:
    - yarpapigateway
  ports:
    - "6005:8080"
    - "6065:8081"
  volumes:
    -
    ${APPDATA}/Microsoft/UserSecrets:/home/app/.microsoft/usersecrets:
ro
    - ${APPDATA}/ASP.NET/Https:/home/app/.aspnet/https:ro

```