

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему **РОЗПОДІЛЕНА СИСТЕМА ЕЛЕКТРОННОЇ КОМЕРЦІЇ З
ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ**

Виконав: студент 2 курсу, групи КНТ-622м
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Спеціалізовані комп'ютерні системи

ОПАРА О. С.

(ПРИЗВИЩЕ та ініціали)

Керівник КИРИЧЕК Г.Г.

(ПРИЗВИЩЕ та ініціали)

Рецензент САМОЙЛИК С.С.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти магістерський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)

Освітня програма (спеціалізація) «Спеціалізовані комп'ютерні системи»
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“12” грудня 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ОПАРИ Олександра Сергійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розподілена система електронної комерції з використанням хмарних технологій

керівник проєкту (роботи) к. т. н., доцент, КИРИЧЕК Галина Григорівна

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “24” жовтня 2023 року №400

2. Строк подання студентом проєкту (роботи) 10 грудня 2023 року

3. Вихідні дані до проєкту (роботи) наявні мови програмування, наявні провайдери хмарних технологій

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз технічного завдання;

2) Розробка структури та алгоритму роботи системи;

3) Реалізація та дослідження системи;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	КИРИЧЕК Г. Г., доцент		
нормоконтроль	ЩЕРБАК Н.В., ст. викл.		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз технічного завдання	10.10.2023 р.	
2	Реалізація Customer Service та Order Service	15.10.2023 р.	
3	Реалізація Authentication Service	20.10.2023 р.	
4	Реалізація Payments Service	25.10.2023 р.	
5	Порівняння та дослідження сервісів керування репозиторіями	10.11.2023 р.	
6	Дослідження сервісів Amazon для розгортання бази даних	20.11.2023 р.	
7	Дослідження сервісів Amazon для розгортання мікросервісів	22.11.2023 р.	
8	Оформлення отриманих результатів у ПЗ	25.11.2023 р.	

Студент  **Олександр ОПАРА**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи)  **Галина КИРИЧЕК**
(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 76 сторінок, 9 таблиць, 29 джерел, 9 рисунків, 3 додатки.

AMAZON WEB SERVICES, JAVA, SPRING FRAMEWORK,
РОЗПОДІЛЕНА СИСТЕМА, СЕРВІСИ ХМАРНИХ ОБЧИСЛЕНЬ,

Об'єкт дослідження – процес реалізації розподіленої системи електронної комерції з використанням хмарних технологій.

Мета роботи – аналіз технологій та проведення досліджень, пов'язаних з реалізацією розподіленої системи електронної комерції з використанням хмарних технологій.

У першому розділі проводиться аналіз предметної області. Розглядаються актуальні мови програмування, сучасні СУБД та сервіси хмарних обчислень. Проводиться вибір найбільш доцільних засобів, необхідних для виконання дипломної роботи.

У другому розділі описуються етапи реалізації системи електронної комерції.

У третьому розділі проводиться дослідження технологій та хмарних сервісів необхідних для впровадження системи електронної комерції.

ABSTRACT

II3: 76 pages, 9 tables, 29 sources, 9 pictures, 3 additions.

AMAZON WEB SERVICES, CLOUD SERVICES, DISTRIBUTED SYSTEM, JAVA, SPRING FRAMEWORK

Object of research – process of implementation of the distributed e-commerce system using cloud technologies.

Objective – analysis and research of technologies related to the implementation of the distributed e-commerce system using cloud technologies.

The first section analyzes the subject area. Current programming languages, modern DBMS and cloud computing services are considered. The selection of the most appropriate means necessary for the completion of the thesis is carried out.

The second chapter describes the stages of implementation of the distributed e-commerce system.

In the third section, a research of technologies and cloud services necessary for the implementation of an distributed e-commerce system is carried out.

ЗМІСТ

Вступ.....	8
1 Аналіз технічного завдання	8
1.1 Аналіз мов програмування та фреймворків для реалізації мікросервісів додатку.....	9
1.1.1 Аналіз мови програмування Java та Spring Framework	10
1.1.2 Аналіз мови програмування TypeScript та NestJS фреймворку	11
1.1.3 Аналіз мови програмування C# та ASP.Net Core фреймворку	12
1.1.4 Висновки аналізу мов програмування та фреймворків	14
1.2. Аналіз баз даних	14
1.2.1 Аналіз SQL та NO SQL баз даних	14
1.2.2 Аналіз PostgreSQL СУБД.....	16
1.2.3 Аналіз MongoDB СУБД	17
1.2.4 Висновки аналізу SQL та NoSQL баз даних	18
1.3. Аналіз сервісів хмарних обчислень.....	19
1.3.1 Аналіз платформи Amazon Web Services	19
1.3.2 Аналіз платформи Azure	20
1.3.3 Аналіз платформи Google Cloud.....	20
1.3.4 Висновки аналізу сервісів хмарних обчислень.....	21
2 Реалізація розподіленої системи з використанням хмарних технологій	22
2.1 Реалізація Customer Service та Order Service	22
2.1.1 Впровадження Customer Service	22
2.1.2 Реалізація Order Service.....	32
2.2 Реалізація Authentication Service.....	33
2.2.1 Реалізація доступу до даних за допомогою JWT	36
2.2.2 Реалізація класів сутностей для зберігання даних для логіну користувача	40
2.3 Реалізація Payments Service.....	46

2.3.1 Вибір платіжної системи	46
2.3.2 Створення функціоналу зі збереження методу оплати користувача	48
2.3.3 Створення функціоналу з проведення оплати користувача	49
2.3.4 Створення функціоналу з повернення коштів за виконану оплату	51
3 Дослідження технологій необхідних для реалізації та хмарних сервісів amazon	52
3.1 Порівняння та дослідження сервісів керування репозиторіями	53
3.1.1 Дослідження системи керування репозиторіями GitLab	54
3.1.2 Дослідження системи керування репозиторіями GitHub	55
3.1.3 Дослідження системи керування репозиторіями BitBucket	56
3.1.4 Порівняння систем керування репозиторіями	58
3.2 Дослідження сервісів Amazon для розгортання бази даних	61
3.2.1 Дослідження AWS RDS	61
3.2.2 Дослідження AWS AURORA	62
3.2.3 Дослідження AWS EC2	63
3.2.4 Висновок дослідження	64
3.3 Дослідження сервісів Amazon для розгортання мікросервісів	65
3.3.1 Дослідження AWS Lambda	65
3.3.2 Дослідження AWS Elastic Beanstalk	66
3.3.3 Порівняльна характеристика сервісів AWS для розгортання мікросервісів	66
Висновки	69
Перелік джерел посилання	71

ВСТУП

Сучасна епоха інформаційних технологій вимагає від бізнесів реагувати на зростаючий попит і виклики споживачів шляхом впровадження динамічних та масштабованих рішень для підтримки онлайн-торгівлі. В світлі стрімкого розвитку Інтернету, поширення мобільних технологій та змін споживацьких звичок, створення надійних і продуктивних систем електронної комерції стає завданням першочергового значення для бізнесу. У цьому контексті розподілені системи електронної комерції та використання хмарних технологій виявляються ключовими компонентами, що забезпечують стійку та ефективну інфраструктуру для бізнесу в епоху цифрової революції [1].

Розподілені системи електронної комерції поєднують в собі сучасні технології обробки та управління даними, забезпечуючи користувачів потужними інструментами для виконання та проведення електронних транзакцій.

Хмарні технології, в свою чергу, надають бізнесам можливість легко розгортати, масштабувати і керувати інфраструктурою без великих капіталовкладень у фізичне обладнання та забезпечують високу доступність та надійність. Це дозволяє електронним торговцям зосередитися на вдосконаленні свого продукту або послуг, що вони надають, не витрачаючи багато часу та ресурсів на управління інфраструктури.

1 АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Згідно технічного завдання метою дипломної роботи є аналіз технологій та проведення досліджень, пов'язаних з реалізацією розподіленої системи електронної комерції. Вона призначена для надання користувачу послуг з перегляду інформації про товар через графічний інтерфейс, замовлення товарів через додаток, можливості оплати товару.

Сервіс електронної комерції повинен:

- надавати функціонал для взаємодії з товарами – додавання, редагування;
- надавати зручний інтерфейс для перегляду, фільтрації та пошуку наявних товарів в додатку електронної комерції;
- вміти працювати з даними користувачів та зберігати інформацію про користувачів у захищеному вигляді;
- формувати замовлення на основі вибору користувачів;
- надавати користувачам можливість оплати товару.

У ході виконання дипломної роботи проаналізована та досліджена реалізація системи електронної комерції, що використовує хмарні технології для своєї роботи та функціонування, та технології, що використовувались для реалізації системи. Система дозволить користувачам використовувати додаток для перегляду наявних товарів, їх замовлення та оплати.

Порівняні та проаналізовані різні сервіси провайдерів хмарних технологій, що можна використати для реалізації додатка та налаштування інфраструктури сервісу електронної комерції. Проведене порівняння протоколів передачі даних між компонентами системи та швидкодія підходів REST та Event-Driven для взаємодії компонентів системи.

1.1 Аналіз мов програмування та фреймворків для реалізації мікросервісів додатку

Система, що потрібно впровадити, є розподіленою, тому потрібно обрати мову програмування, використовуючи яку, будуть реалізовані мікросервіси, що виконують усі необхідні функції. Але оскільки мова програмування частіше за все використовується з фреймворком, що надає необхідні інструменти для реалізації веб-додатків, то разом із мовою програмування буде виконано аналіз фреймворків.

1.1.1 Аналіз мови програмування Java та Spring Framework

Java - це об'єктно-орієнтована мова програмування, яка вирізняється своєю високою популярністю та використанням у різних галузях розробки програмного забезпечення. Вона розроблена компанією Sun Microsystems (пізніше придбаною Oracle) і випущена в 1995 році. Однією з основних переваг мови Java є її кросплатформеність, завдяки віртуальній машині Java (JVM), яка дозволяє виконувати код на різних операційних системах. Ця можливість робить Java ідеальною для розробки крос-платформеного програмного забезпечення.

Основні риси Java включають в себе об'єктно-орієнтований підхід, незалежність від платформи, безпеку, простоту та читабельність синтаксису, многопоточність, розширену екосистему бібліотек та фреймворків, можливості реалізації великих систем, підтримку мережевого програмування, а також активну спільноту розробників та регулярні оновлення мови.

Java також відома своєю безпекою завдяки вбудованим механізмам безпеки, обмеженням доступу до ресурсів та валідації коду під час виконання. Це робить її популярною для реалізації безпечних додатків, включаючи банківські та фінансові системи.

Узагальнюючи, Java - це потужна та універсальна мова програмування, яка знайшла широке використання в різних областях реалізації програмного

забезпечення і продовжує бути актуальною завдяки своїм перевагам і розвиваючійся спільноті розробників [2].

Разом із мовою програмування Java у сучасних веб додатків найчастіше використовують Spring Framework, що є одним із найбільш впливових та популярних фреймворків у світі розробки програмного забезпечення на мові Java. Цей фреймворк вирізняється численними перевагами, які роблять його зручним та потужним інструментом для розробників.

Spring Framework надає принцип інверсії керування (IoC), що дозволяє керувати залежностями між об'єктами, спрощуючи реалізацію та тестування. Це робить фреймворк особливо привабливим для створення додатків зі складними системними архітектурами.

Крім того, Spring надає інструменти для роботи з базами даних, включаючи JDBC та ORM (наприклад, Hibernate), що спрощує доступ до даних та їх управління. Фреймворк також допомагає в розробці веб-додатків через Spring MVC та надає можливість створення RESTful API.

Spring Framework підтримує аспектно-орієнтоване програмування (AOP), дозволяючи розділити крос-різноманітну функціональність, таку як журналювання та транзакції, на аспекти. Це полегшує управління аспектами додатка.

Фреймворк є високорозширюваним завдяки своїй модульній структурі. Розробники можуть вибирати лише ті модулі, які потрібні для свого проекту, що робить Spring вкрай гнучким.

Spring також має підтримку для асинхронного програмування, що стає все важливішим у сучасному програмуванні. Він також надає інструменти для взаємодії з мікросервісною архітектурою, яка вдосконалює створення та керування мікросервісами.

Загалом, Spring Framework є важливим інструментом для реалізації Java-додатків завдяки своїй потужності, розширюваності та зручності використання. Він надає набір інструментів та бібліотек, які роблять реалізацію програмного забезпечення більш продуктивною та ефективною [3].

1.1.2 Аналіз мови програмування TypeScript та NestJS фреймворку

TypeScript - це мова програмування, розроблена компанією Microsoft, яка базується на JavaScript і додає до неї статичну типізацію. Ця статична типізація дозволяє розробникам визначати типи змінних, функцій і об'єктів, що робить код більш надійним та піддається перевірці на ранніх стадіях розробки. TypeScript є надбудовою над JavaScript, що означає, що весь дійсний код JavaScript є дійсним кодом TypeScript. Це дозволяє розробникам з поступовим переходом до TypeScript без необхідності переписувати весь існуючий код.

Однією з ключових особливостей TypeScript є його об'єктно-орієнтована парадигма. Мова підтримує класи, спадкування, інтерфейси та інші концепції ООП, що полегшує структурування та організацію коду.

TypeScript також надає модульність для організації коду, підтримку нових стандартів ECMAScript і велику кількість інструментів розробки. Він інтегрується з популярними фреймворками та бібліотеками, що робить його привабливим для великої кількості розробників. TypeScript має активну спільноту та підтримку від Microsoft, що забезпечує стабільність та регулярні оновлення мови.

Завдяки можливостям TypeScript, розробники можуть створювати більш надійне та зрозуміле програмне забезпечення, знижуючи кількість помилок та полегшуючи обслуговування. Ця мова корисна як для клієнтської, так і для серверної розробки, і грає важливу роль у сучасній веб-розробці та інших галузях програмування [4].

Разом з цією мовою програмування можна використовувати багато фреймворків для створення веб додатків, але одним з найпопулярніших є NestJS Framework - це сучасний фреймворк для реалізації серверних додатків на мові TypeScript, побудований на основі Node.js. Він пропонує розробникам об'єктно-орієнтований підхід до побудови додатків і відзначається модульною структурою, яка полегшує організацію коду.

Основними перевагами NestJS є підтримка TypeScript, можливість створення RESTful API, вбудована підтримка WebSockets для реального часу, обробка HTTP-запитів через контролери, сервіси та провайдери, а також розвинута система middleware та фільтрів для обробки запитів та відповідей. Фреймворк також сприяє тестуванню додатків і інтегрується з різними іншими бібліотеками та фреймворками. З великою та активною спільнотою розробників, NestJS стає сильним інструментом для створення надійних і масштабованих серверних додатків [5].

1.1.3 Аналіз мови програмування C# та ASP.Net Core фреймворку

C# - це сучасна і високорівнева мова програмування, розроблена компанією Microsoft. Вона призначена для створення різноманітних програмних додатків, включаючи десктопні додатки, веб-додатки, мобільні додатки, ігри та багато інших. C# використовується в різних галузях завдяки своїй популярності та гнучкості.

Основні характеристики C# включають в себе:

- об'єктно-орієнтовану парадигму, яка дозволяє структурувати код навколо об'єктів і класів, полегшуючи модульність та повторне використання коду;
- статичну типізацію, яка дозволяє визначати типи даних на етапі компіляції, що сприяє виявленню та усуненню помилок на ранніх етапах розробки;
- підтримку багатьох платформ, включаючи Windows, Linux та macOS, завдяки використанню платформи .NET;
- інтеграцію з різними технологіями та бібліотеками, що робить C# універсальним інструментом для розробки.

C# також підтримує різні парадигми програмування, включаючи функціональне програмування та аспектно-орієнтоване програмування. Це робить мову гарним вибором для реалізації різноманітних додатків, від корпоративних програм до ігрових двигунів.

Завдяки постійному розвитку і оновленням, C# залишається сучасною і популярною мовою програмування, надаючи розробникам потужний інструмент для вирішення різноманітних завдань [6].

C# використовується у поєднанні з .NET платформою, що містить у собі багато інших фреймворків, необхідних для реалізації різних додатків – від мобільних до веб додатків. У ході виконання дипломної роботи проаналізовано ASP.Net Core - сучасний фреймворк для реалізації веб-додатків, який відрізняється крос-платформеністю, високою продуктивністю та гнучкістю. Він створений компанією Microsoft і вже завоював велику популярність серед розробників завдяки своїм ключовим особливостям.

Однією з найбільших переваг ASP.NET Core є його крос-платформеність, тобто здатність працювати на різних операційних системах, включаючи Windows, Linux і macOS. Це робить його ідеальним вибором для розробки веб-додатків незалежно від вибору серверної операційної системи.

Фреймворк також відзначається високою продуктивністю завдяки оптимізаціям та асинхронним операціям, що дозволяють обробляти великі навантаження. Вбудована підтримка Docker робить розгортання та масштабування додатків ще більш простими.

У фреймворку є вбудована підтримка SignalR для реалізації функціоналу, що залежить від реального часу, та багато інших інструментів та ресурсів для розробки. Активна спільнота розробників ASP.NET Core завжди готова допомогти та надати поради.

Отже, ASP.NET Core - це потужний фреймворк, який робить розробку веб-додатків ефективною та простою, незалежно від платформи та завдань проекту. Його крос-платформеність, продуктивність та гнучкість роблять його одним із найбільш привабливих інструментів у світі веб-розробки [7].

1.1.4 Висновки аналізу мов програмування та фреймворків

Проаналізувавши мови програмування та фреймворки, було вирішено обрати Java та Spring Framework. Оскільки проаналізовані фреймворки схожі

за можливостями, що вони пропонують, у Spring Framework є декілька переваг, що мають вплив на реалізацію додатка:

- Spring Framework надає широкий спектр можливостей для розробки додатків, включаючи підтримку різних рівнів архітектури (наприклад, MVC, REST, Data Access), інтеграцію з різними технологіями (наприклад, бази даних, меседж-брокери, робота з сек'юриті), а отже використання сторонніх бібліотек необов'язково;
- Spring Framework пропонує готові рішення та модулі з мінімальною конфігурацією, що можуть значно спростити розробку додатка;
- Spring надає Security модуль - один з найпопулярніших фреймворків для забезпечення безпеки в додатках Java, що має потужні можливості для автентифікації і авторизації, та допомагає захищати додаток від різних загроз;
- Java і Spring Framework є платформонезалежними, що означає, що додатки можуть працювати на різних операційних системах та середовищах виконання, що в свою чергу спростить процес розгортання інфраструктури додатка на хмарних сервісах.

1.2 Аналіз баз даних

1.2.1 Аналіз SQL та NO SQL баз даних

Для того, щоб реалізувати систему, що оперує даними, необхідно обрати сховище, для зберігання даних. У сучасних системах для цього використовують бази даних. Вони бувають реляційними (SQL) та не реляційними (No SQL).

SQL бази даних, або реляційні бази даних, є дуже поширеними і важливими інструментами для організації та збереження структурованих

даних. Вони використовують мову запитів SQL для взаємодії з даними та мають ряд характеристик, які роблять їх особливо корисними:

Дані в SQL базах даних організовані у вигляді двовимірних таблиць, де кожен рядок представляє окремий запис, а кожний стовпець визначає певну властивість даних. Це спрощує структурування та управління інформацією.

У реляційних базах даних використовується мова SQL (Structured Query Language) для виконання операцій над даними. Вона дозволяє створювати запити для вибірки, вставки, оновлення та видалення даних в базі даних.

Також SQL бази даних дотримуються принципів ACID (Atomicity, Consistency, Isolation, Durability), що гарантує атомарність операцій, цілісність даних та долучість навіть у випадку відмови.

SQL бази даних дозволяють створювати індекси для швидкого доступу до даних, що полегшує оптимізацію запитів, а також підтримують транзакції, які групують операції для забезпечення консистентності даних та атомарного виконання.

SQL бази даних знаходять застосування у багатьох галузях, включаючи бізнес, науку, веб-розробку та інші галузі завдяки їхнім можливостям збереження та керування даними.

У свою чергу NoSQL бази даних (Not Only SQL) представляють альтернативу SQL базам даних і відрізняються за багатьма характеристиками.

Однією з головних відмінностей NoSQL баз є відсутність реляційної структури. Дані зберігаються у різних форматах, таких як документи, ключ-значення, колонки або графи. Це дозволяє більшу гнучкість у представленні даних.

Деякі NoSQL бази даних, використовують модель документів. У цьому випадку дані зберігаються у документах, які можуть містити будь-яку структуру, включаючи вкладені дані, деякі бази, як Redis або Cassandra, використовують сховище ключ-значення - дані зберігаються під унікальним ключем, і вони можуть бути швидко витягнуті за цим ключем.

NoSQL бази даних добре підходять для горизонтального масштабування, що означає легке додавання нових серверів для розширення обсягів даних і обробки навантаження. Це дозволяє впоратися з великими обсягами даних та високою навантаженістю.

NoSQL бази даних, зокрема бази, що зберігають дані у вигляді ключ-значення, відомі своєю високою швидкістю доступу до даних і операцій з ними, що робить їх ідеальними для вимогливих за швидкістю додатків.

Такі бази даних допускають роботу без строго визначеної схеми, що дозволяє легко додавати та змінювати дані без перегляду всіх існуючих записів.

Загалом, NoSQL бази даних відкривають багато можливостей для збереження та опрацювання даних, особливо в областях, де потрібна гнучкість, розподілена архітектура і висока швидкість [8].

1.2.2 Аналіз PostgreSQL СУБД

Однією з найпоширеніших СУБД для реляційних баз даних є PostgreSQL - це потужна та високорозвинена система управління базами даних (СУБД) з відкритим вихідним кодом, яка надає реляційну модель для збереження та керування структурованими даними.

Однією з ключових особливостей PostgreSQL є відкритий вихідний код, що дозволяє використовувати, модифікувати та розповсюджувати СУБД безкоштовно. Вона підтримує реляційну модель для організації даних в таблицях з визначеними схемами та взаємозв'язками між таблицями. PostgreSQL підтримує мову запитів SQL, що дозволяє виконувати різноманітні операції з даними, включаючи вибірку, вставку, оновлення та видалення даних.

Важливою перевагою PostgreSQL є його розширюваність та можливість роботи в розподілених системах. Вона підтримує створення різних видів індексів для покращення продуктивності запитів та пошуку даних. Також вона дозволяє розширювати функціональність за допомогою розширень та користувацьких функцій.

PostgreSQL надає високу підтримку транзакцій згідно із принципами ACID, що гарантує атомарність, цілісність, ізоляцію та долучість операцій. Вона також має розширення для роботи з географічними та геопросторовими даними, що корисно для геолокаційних додатків та систем інформації про картографію [9].

1.2.3 Аналіз MongoDB СУБД

MongoDB - це NoSQL система управління базами даних, яка спеціалізується на зберіганні та опрацюванні документів у форматі BSON. Вона володіє кількома ключовими характеристиками, які роблять її популярною для різних застосувань.

MongoDB використовує модель документів для збереження даних. Кожен запис, відомий як "документ", представляє собою JSON-подібну структуру даних, яка може містити будь-які поля та піддокументи. Це дозволяє гнучко організовувати та зберігати дані без необхідності визначення жорстких схем.

Ця СУБД добре підходить для розподіленого масштабування. Можна легко додавати нові сервери для обробки великих обсягів даних та високих навантажень. Ця можливість дозволяє підтримувати високу доступність і високий рівень продуктивності.

MongoDB використовує запити на основі мови запитів, подібної до SQL, щоб вибирати, оновлювати та видаляти дані. Вона також підтримує індексацію, що полегшує швидкий доступ до даних.

СУБД дозволяє створювати репліки даних для забезпечення високої доступності і відмовостійкості. Реплікація даних відбувається автоматично, і в разі відмови одного сервера дані можуть бути взяті з іншого. Також MongoDB не вимагає фіксованих схем даних. Це дозволяє додавати, змінювати та видаляти поля в документах без перерви в роботі з існуючими записами.

Загалом, MongoDB - це потужна NoSQL СУБД, яка знаходить застосування в різних областях, включаючи веб-розробку, аналітику,

геолокаційні додатки та інші сфери, де потрібна гнучкість у збереженні та обробці даних [10].

1.2.4 Висновки аналізу SQL та NoSQL баз даних

Оскільки додаток, що впроваджується, є розподіленим, тобто буде складатися з групи мікросервісів, то для зберігання даних добре підійдуть як SQL так і NoSQL бази даних. Щоб підсумувати попередній аналіз PostgreSQL та MongoDB, було прийнято рішення провести невелике порівняння цих СУБД. Результат порівняння можна побачити у таблиці 1.1.

Таблиця 1.1 – Порівняння

-	PostgreSQL	MongoDB
Структура	Вимагає заздалегідь визначеної статичної структури даних	Структура у вигляді документа
Швидкість	Швидкість запиту залежить від коректності створеного SQL запиту. Велика кількість зв'язків між таблицями може сповільнити запити	Завдяки документо подібній структурі MongoDB надає велику продуктивність та швидкість виконання запитів
ACID	PostgreSQL повністю слідує принципам ACID та забезпечує атомарність та ізоляцію транзакцій	Дуже обмежена підтримка принципів ACID

Хоча MongoDB і значно простіша у плані взаємодії з даними за рахунок документоподібності даних, PostgreSQL надає більших контроль над цілісністю даних, а також гарантує цілісність та ізолюваність даних, що там зберігаються. І хоча PostgreSQL у деяких випадках не має такої ж швидкості виконання запитів, як у MongoDB, SQL запити завжди можна оптимізувати, щоб досягнути максимальної швидкості виконання запитів.

Отже, на основі аналізу та порівняння цих СУБД, було вирішено обрати PostgreSQL, оскільки SQL СУБД надає більше можливостей для проектування таблиць, необхідних для створення додатка.

1.3 Аналіз сервісів хмарних обчислень

Сервіси хмарних обчислень - це набір послуг і ресурсів, які надаються через інтернет із використанням обчислювальних серверів, розташованих у великих дата-центрах. Ці послуги можуть бути надані за допомогою хмарних платформ, таких як Amazon Web Services, Microsoft Azure, Google Cloud Platform та інших. Сервіси хмарних обчислень надають користувачам доступ до різних інфраструктурних, обчислювальних та програмних ресурсів без необхідності будувати та підтримувати власну інфраструктуру.

Сервіси хмарних обчислень використовуються для різних завдань, включаючи веб-розробку, аналітику даних, інфраструктурні проекти, розгортання додатків, зберігання даних та багато інших сфер. Вони надають користувачам можливість швидко масштабувати та забезпечувати доступність своїх додатків та послуг, не витрачаючи значних ресурсів на фізичну інфраструктуру.

1.3.1 Аналіз платформи Amazon Web Services

Amazon Web Services - це один з провідних провайдерів хмарних обчислювальних послуг у світі, який пропонує різноманітні послуги та інфраструктуру для бізнесів та розробників. AWS створений і управляється Amazon.com і був запущений у 2006 році. Він дозволяє користувачам здійснювати різні обчислювальні, зберігання, бази даних, мережеві та інші операції на хмарних серверах, замість власних фізичних серверів.

AWS надає можливість легко масштабувати інфраструктуру вгору або вниз в залежності від потреб бізнесу. Сервіс дозволяє збільшити або зменшити потужність обчислювальних ресурсів, якщо потрібно, що дозволяє ефективно використовувати ресурси і знижувати витрати.

Хмарні сервіси Amazon мають розгалужену мережу дата-центрів, розташованих у різних куточках світу, що дозволяє забезпечити високу доступність та надійність ваших додатків і послуг, а також пропонують

широкий вибір послуг і інфраструктури, включаючи обчислювальні ресурси, зберігання, бази даних, мережеві послуги, інструменти для розробників та багато інших.

AWS надає ряд інструментів і послуг для захисту даних і додатків, включаючи шифрування, ідентифікацію та керування доступом. Також Amazon постійно оновлює свої послуги та вводить нові функції, що дозволяє користувачам використовувати передові технології в своїх проектах [11].

1.3.2 Аналіз платформи Azure

Microsoft Azure - це хмарна платформа, розроблена компанією Microsoft, яка надає широкий спектр послуг та інфраструктури для бізнесів і розробників.

Azure дозволяє створювати, розгортати та керувати різноманітними застосунками та інфраструктурою у хмарному середовищі. Його основні переваги включають гнучкість і розширюваність, велику кількість доступних послуг, підтримку гібридних рішень, високий рівень безпеки та захисту даних, оплату за використанням, підтримку різних операційних систем і мов програмування, а також мережеві можливості.

Подібно до AWS, Azure пропонує модель оплати за використані ресурси, що дозволяє бізнесам платити лише за те, що вони використовують, зменшуючи витрати. Також Azure підтримує гібридні рішення, що означає, що ви можна поєднувати інфраструктуру в хмарі з власною фізичною інфраструктурою. Це особливо корисно для організацій, які мають вже існуючу інфраструктуру та потребують поетапного переходу до хмари [12].

1.3.3 Аналіз платформи Google Cloud

Google Cloud Platform (GCP) - це хмарна платформа, яку розробляє та надає компанія Google. Ця платформа пропонує широкий спектр хмарних послуг та інфраструктури для бізнесів і розробників. Нижче наведено детальний опис Google Cloud Platform та її переваг:

GCP надає доступ до великого пулу обчислювальних ресурсів, включаючи віртуальні машини, контейнери та сервери для обробки даних, які

можна легко масштабувати за потреби, також має широкий спектр послуг, включаючи зберігання даних, бази даних, машинне навчання та штучний інтелект, інтернет речей, аналітику даних та інші. Це дозволяє розробникам та компаніям будувати різноманітні застосунки та сервіси.

Платформа Google пропонує розширену мережеву інфраструктуру, включаючи віртуальні приватні мережи, балансування навантаження та CDN-сервіси. Це робить платформу ідеальною для будь-яких мережевих потреб.

GCP вкладає значні зусилля в забезпечення безпеки та захисту даних. Платформа має інструменти для шифрування, контролю доступу, виявлення загроз та захисту інфраструктури, також пропонує послуги для аналізу та обробки даних, включаючи BigQuery для великих обсягів даних та інструменти для реалізації потокового аналізу даних.

Загалом, Google Cloud Platform є потужною хмарною платформою з багатьма інструментами та послугами для розробників, які шукають хмарне середовище для своїх проєктів і інфраструктури [13].

1.3.4 Висновки аналізу сервісів хмарних обчислень

Серед проаналізованих хмарних платформ не можна виділити якийсь кращий чи гірший. Усі платформи надають велику кількість різноманітних сервісів, які можна використовувати для створення інфраструктури дипломної роботи. Тому, розглянувши хмарні платформи, було вирішено обрати Amazon Web Services.

AWS дуже легка у використанні, а щоб почати користуватись сервісами, необхідно просто створити обліковий запис. При цьому багато сервісів можна використовувати абсолютно безплатно. Також Amazon Web Services мають найбільшу кількість сервісів, що дозволяє гнучко налаштувати інфраструктуру додатка.

2 РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОЇ СИСТЕМИ ЕЛЕКТРОННОЇ КОМЕРЦІЇ З ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ

Система є розподіленою, тому її реалізація буде складатися з впровадження декількох мікросервісів. Мікросервіси повинні дозволяти користувачу надавати доступ до каталогу продуктів електронної комерції, можливість оформлювати замовлення, захист даних користувача, можливість виконувати оплату замовлень [14].

А тому, потрібно реалізувати наступні мікросервіси, що дозволять виконати описані вище вимоги:

- product service – виконання CRUD операцій з продуктами;
- authentication service – валідація запитів та виконання CRUD операцій з даними користувача;
- order service – створення та обробка замовлень користувача;
- payments service – виконання оплати та збереження.

2.1 Реалізація Customer Service та Order Service

2.1.1 Впровадження Customer Service

Першим мікросервісом буде реалізовано Product Service. Сервіс буде відповідальним за менеджмент продуктів у системі.

Сервіс буде працювати з даними, і тому ці дані потрібно зберігати. Перш ніж починати реалізацію сервісу, необхідно спроектувати сутності бази даних, що будуть відображати дані, з якими сервіс працює. За предмет продажу, було обрано велосипеди. Для реалізації сервісу були виділені наступні основні сутності, які будуть винесені в таблиці бази даних:

Таблиця 2.1 – Головні таблиці ProductService

Назва таблиці	Зміст таблиці
Family	Дані про групи велосипедів
Brand	Дані про наявні бренди велосипедів
Category	Дані про наявні категорії
BaseBike	Дані про основний продукт
ProductVariant	Дані про варіанти основного продукту
Currency	Дані про доступні валюти в цінах продуктів
Retailer	Дані про постачальників
ProductVariantSku	Дані складського обліку
ProductVariantStock	Дані про наявність варіантів
Material	Дані про матеріал, з якого зроблено велосипед
Media	Медіа дані
Colorway	Дані про забарвлення велосипеду
Color	Дані про наявні кольори
Feature	Дані про особливості велосипедів

Загалом уся структура бази даних можна побачити на рисунку 2.1.

Також деякі сутності будуть поєднані між собою за допомогою додаткових таблиць, що утворюють 1 : M – M : 1 зв'язки.

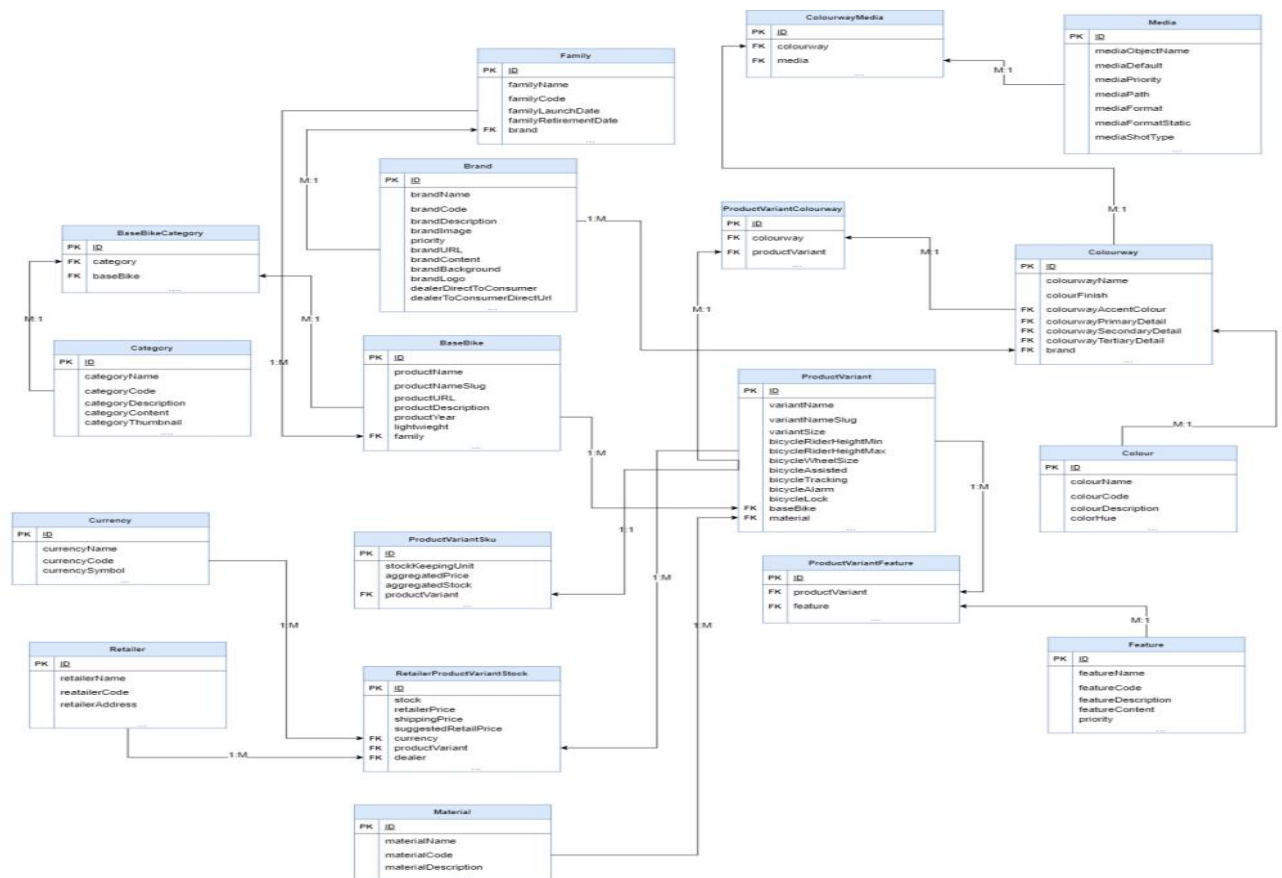
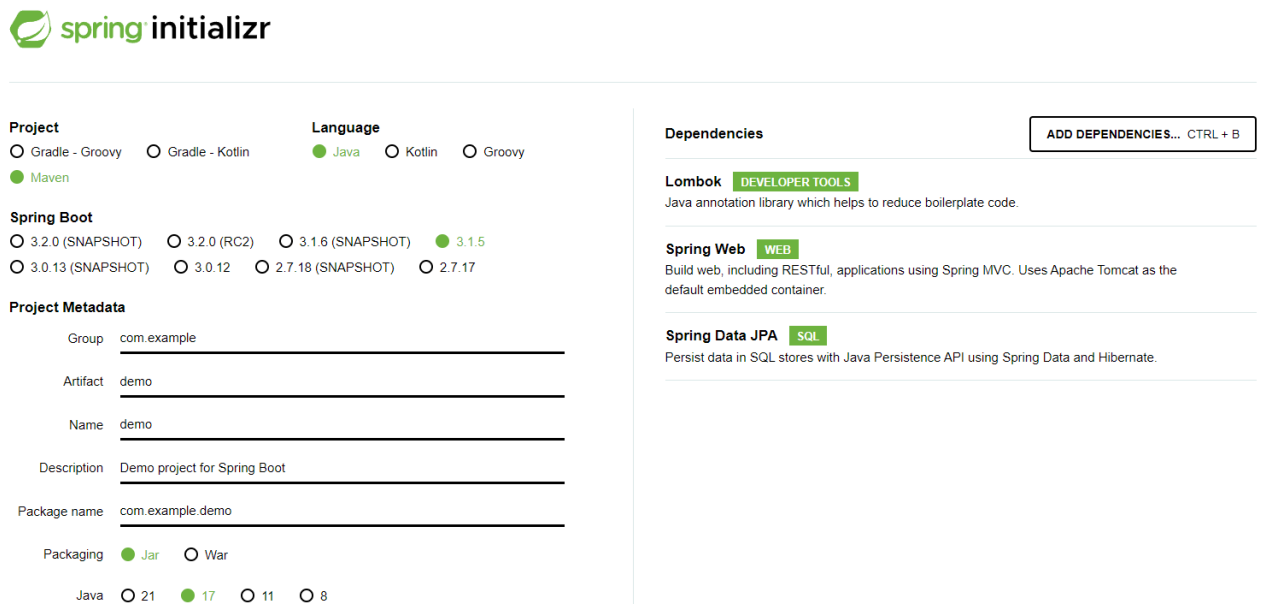


Рисунок 2.1 – Схема таблиць для Customer Service

Реалізація цих додаткових таблиць таким чином дозволяє зберегти зв'язок М : М між сутностями, але додати додаткові поля до таблиць, з якими можна взаємодіяти використовуючи класи Java.

Для створення Customer Service буде використовуватися Spring Boot фреймворк. Фреймворк надає автоматичну конфігурацію проекту та має вбудований веб-сервер Tomcat, що дозволяє запускати проект без додаткових налаштувань.

Для початку було створено проект. Для цього було використано веб ресурс <https://start.spring.io>. Ресурс дозволяє створити проект, додавши усі необхідні залежності та налаштування (рис. 2.2).



The screenshot shows the Spring Initializr web interface. It is divided into several sections:

- Project:** Includes radio buttons for **Gradle - Groovy**, **Gradle - Kotlin**, **Java** (selected), **Kotlin**, and **Groovy**. Below this are radio buttons for **Maven** (selected) and **Gradle**.
- Spring Boot:** Includes radio buttons for various versions: **3.2.0 (SNAPSHOT)**, **3.2.0 (RC2)**, **3.1.6 (SNAPSHOT)**, **3.1.5** (selected), **3.0.13 (SNAPSHOT)**, **3.0.12**, **2.7.18 (SNAPSHOT)**, and **2.7.17**.
- Project Metadata:** Includes input fields for **Group** (com.example), **Artifact** (demo), **Name** (demo), **Description** (Demo project for Spring Boot), and **Package name** (com.example.demo).
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Language:** Includes radio buttons for **Java** (selected), **17** (selected), **11**, and **8**.
- Dependencies:** Includes a button **ADD DEPENDENCIES... CTRL + B**. Below this are sections for **Lombok** (DEVELOPER TOOLS), **Spring Web** (WEB), and **Spring Data JPA** (SQL).

Рисунок 2.2 – інтерфейс веб ресурсу зі створення Spring Boot додатків.

Як можна побачити на рисунку 2.2, можна обрати мову програмування, її версію, засіб автоматизації проекту та різні залежності, що являють собою бібліотеки або інші модулі Spring Framework.

Для початку було обрано Maven, Java 17, та наступні залежності:

– **lombok** – дозволяє зменшити кількість одноманітного шаблонного коду за допомогою Java анотацій;

- spring web – модуль Spring Framework, що спрощує реалізацію та проектування веб додатків на Spring;
- spring data JPA – модуль Spring Framework, що дозволяє працювати з базами даних.

Для роботи з базою даних буде використано Hibernate, що є однією з реалізацій JPA. JPA є стандартом для роботи з об'єктно-реляційним відображенням даних в Java-додатках.

Його основна мета - забезпечити уніфікований спосіб управління об'єктами в базі даних. JPA визначає API та специфікації для взаємодії з об'єктами в Java-додатках та їх збереження в базі даних. Hibernate надає механізм об'єктно-реляційного відображення (ORM) та використовує анотації або файл мапування для визначення відповідності між класами Java та таблицями бази даних.

Після генерації проекту було імплементовано класи-сутності, через які додаток буде працювати з таблицями у базі даних. Кожен клас співпадає з таблицею у базі даних, та має такі ж самі поля [15].

Як приклад, сутність для таблиці бренду можна побачити у лістингу 2.1

Лістинг 2.1 – Приклад класу, що співствляється до таблиці Brand

```
@Entity
@Table(name = "brand")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Brand extends BaseEntity {

    private String brandName;
    private String brandCode;
    private String brandDescription;
    private String brandImage;
    private String brandUrl;
    private String brandContent;
    private String brandBackground;
    private String brandLogo;
    private boolean dealerDirectToConsumer;
    private String dealerToConsumerDirectUrl;
}
```

Таким самим чином було зроблено класи для всіх інших сутностей. Структура класів представлена на рисунку 2.3.

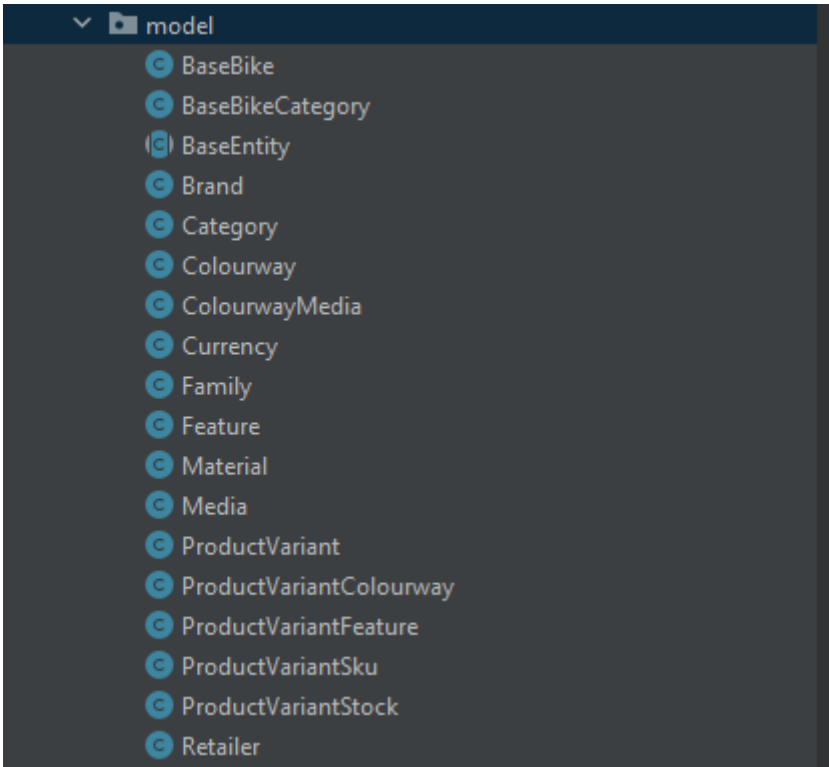


Рисунок 2.3 – Класи сутності для ProductService

Опис анотацій, що було використано для створення класів сутностей наведено в таблиці 2.2

Таблиця 2.2 – Анотації, використані для створення класу сутності.

Анотація	Призначення
@Table	Дозволяє сконфігурувати таблицю (назва і т.д.)
@Entity	Позначає клас як сутність
@AllArgsConstructor	Генерує конструктор класу з усіма компонентами
@NoArgsConstructor	Генерує конструктор за замовчанням
@Getter	Генерує методи доступу для змінних класу
@Setter	Генерує методи призначення для змінних класу

Після створення класів сутностей, було зроблено класи репозиторії, що дозволяють взаємодіяти с даними у базі даних. Це було досягнуто за допомогою Spting Data JPA, що є частиною Spring Framework. Код класу-репозиторію виглядає наступним чином (лістинг 2.2):

Лістинг 2.2 – Код інтерфейсу репозиторію

```
public interface BrandRepository extends JpaRepository<Brand,  
Long> {  
    Optional<Brand> findBrandByBrandName(String brandName);  
}
```

У цьому випадку не було використано жодних анотацій. Результат було досягнуто розширенням інтерфейсу `JpaRepository`.

Spring Framework надає підтримку для роботи з базою даних через різноманітні абстракції, і однією з них є інтерфейс `JpaRepository`. Цей інтерфейс є частиною підпроекту Spring Data JPA, який надає елегантний спосіб взаємодії з базами даних, зокрема з технологією Java Persistence API (JPA).

Коли створюється інтерфейс, який розширює `JpaRepository`, Spring Data JPA автоматично генерує реалізацію цього інтерфейсу під час виконання. Генерація реалізації включає в себе створення методів для основних операцій роботи з базою даних, таких як збереження, оновлення, видалення, отримання по ідентифікатору, отримання всіх об'єктів тощо.

Spring Data JPA визначить, які методи потрібно надати на виконання, з урахуванням підписів методів в `JpaRepository`. Наприклад, якщо в `JpaRepository` є метод з ім'ям `findByBrandName`, Spring Data JPA зрозуміє, що треба створити реалізацію для пошуку користувача за іменем.

Такий механізм генерації коду ґрунтується на конвенціях і може бути налаштований за допомогою власних методів у інтерфейсі-репозиторії. Якщо потрібно визначити власні методи для взаємодії з базою даних, їх можна додавати в інтерфейс-репозиторій, і Spring Data JPA намагатиметься надати відповідну реалізацію під час виконання.

Інтерфейс `JpaRepository` є generic-інтерфейсом, тобто приймає будь які два типа даних. Перший тип – тип класу сутності, другий – тип головного ключа у цьому класі сутності, у нашому випадку це `Long`.

Наступною частиною імплементації ProductService буде реалізація слою сервісів. У контексті веб-додатків термін "слой сервісів" зазвичай вказує на розділення функціональності додатку на різні рівні або компоненти. Це структурний підхід, який спрощує впровадження, тестування та управління веб-додатком, розбиваючи його на логічно зв'язані блоки.

Також у ході реалізації була додана нова залежність (лістинг 2.3).

Лістинг 2.3 – Підключення допоміжної бібліотеки, для спрощення реалізації.

```
<dependencies>
  <dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>${org.mapstruct.version}</version>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <source>17</source>
        <target>17</target>
        <annotationProcessorPaths>
          <path>
            <groupId>org.mapstruct</groupId>
            <artifactId>mapstruct-processor</artifactId>
            <version>${org.mapstruct.version}</version>
          </path>
        </annotationProcessorPaths>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Назва підключеної бібліотеки – Mapstruct. Її призначення – автоматизація процесу конвертації класу сутності у data-transfer об'єкт. Замість того, щоб робити конвертацію власними силами та створювати

шаблонний код для цього, бібліотека зможе автоматизувати цей процес, та сгенерувати необхідний функціонал самостійно.

DTO, або Data Transfer Object, є об'єктом, який використовується для передачі даних між об'єктами в програмі, зазвичай між різними частинами додатку чи між клієнтом та сервером. DTO служить як контейнер для даних, які ви хочете передати, і часто має тільки поля та методи доступу, але не містить бізнес-логіки.

Основні причини використання DTO включають:

- підвищення ефективності передачі даних, шляхом обирання лише необхідних даних з об'єкту класу сутності;
- зменшення зв'язку між компонентами, шляхом ізоляції логіки від структури даних.

Використати бібліотеку можна наступним чином (лістинг 2.4):

Лістинг 2.4 – Використання бібліотеки MapStruct

```
@Mapper(componentModel = "spring")
public interface BrandMapper {
    BrandDto brandToDto(Brand brand);
    Brand dtoToBrand(BrandDto dto);
}
```

Також створено сервіс, що виконує CRUD операції з сутністю бренду. CRUD операції є операціям створення (Create), читання (Read), оновлення (Update) та видалення (Delete). Приклад такого CRUD сервісу можна побачити у лістингу 2.5.

Лістинг 2.5 – Клас-сервіс для роботи з даними бренда.

```
@Service
@RequiredArgsConstructor
public class BrandServiceImpl implements BrandService {
    private BrandMapper mapper;
    private BrandRepository repository;
    @Override
    public BrandDto create(BrandDto dto) {
        repository.save(mapper.dtoToBrand(dto));
    }
}
```

```

        return dto;
    }
    @Override
    public void delete(Long id) {
        Brand brand = repository.findById(id)
            .orElseThrow(() -> new
ItemNotFoundException("Brand not found"));
        repository.delete(brand);
    }
    @Override
    public BrandDto update(BrandDto dto) {
        Brand brand = repository
            .findById(dto.getId()).orElseThrow(() -> new
ItemNotFoundException("Brand not found"));
        Brand updated = mapper.dtoToBrand(dto);
        updated.setId(brand.getId());
        repository.save(updated);
        return mapper.brandToDto(updated);
    }
    @Override
    public BrandDto getById(Long id) {
        Brand brand = repository.findById(id)
            .orElseThrow(() -> new
ItemNotFoundException("Brand not found"));
        return mapper.brandToDto(brand);
    }
    @Override
    public BrandDto getByBrandName(String brandName) {
        Brand brand = repository.findBrandByBrandName(brandName)
            .orElseThrow(() -> new
ItemNotFoundException("Brand not found"));
        return mapper.brandToDto(brand);
    }
    @Override
    public List<BrandDto> getAll() {
        return repository.findAll()
            .stream()
            .map(mapper::brandToDto)
            .collect(Collectors.toList());
    }
}

```

Схожим чином були реалізовані сервіси для інших сутностей, оскільки вони виконують більш менш одні й ті самі функції.

Фінальним етапом впровадження сервісу є створення контролерів, що будуть приймати запити від клієнтського додатку. У веб-розробці термін "контролер" використовується для позначення складової частини архітектури програмного забезпечення, яка відповідає за обробку запитів веб-додатка від користувача та взаємодію з іншими компонентами системи. Контролери

реалізують логіку обробки запитів та визначають, які відповіді повинні бути надіслані користувачеві.

За прикладом контролеру для сутності бренду (лістинг 2.6) було імплементовано контролери для інших сутностей.

Лістинг 2.6 – Контролер для проведення CRUD операцій для сутності бренду

```
@RestController
@RequestMapping("/brand")
@RequiredArgsConstructor
public class BrandController {
    private final BrandService brandService;
    @GetMapping("/{brandId}")
    public ResponseEntity<BrandDto> get(@PathVariable Long
brandId) {
        return
ResponseEntity.status(HttpStatus.OK).body (brandService.getById(b
randId));
    }
    @PostMapping()
    public ResponseEntity<BrandDto> create(@RequestBody
@Validated BrandDto dto){
        return
ResponseEntity.status(HttpStatus.CREATED).body (brandService.crea
te(dto));
    }
    @PostMapping("/{brandId}")
    public ResponseEntity<BrandDto> update(@RequestParam Long
brandId, @RequestBody @Validated BrandDto dto){
        return
ResponseEntity.status(HttpStatus.CREATED).body (brandService.upda
te(brandId, dto));
    }
    @PostMapping("/{brandId}")
    public ResponseEntity<BrandDto> delete(@RequestParam Long
brandId) {
        brandService.delete(brandId);
        return ResponseEntity.status(HttpStatus.OK).build();
    }
    @GetMapping
    public ResponseEntity<List<BrandDto>> getAll(){
        return
ResponseEntity.status(HttpStatus.OK).body (brandService.getAll())
;}}
}
```

На цьому імплементацию Product Service завершено. Як результат, створено сервіс, що дозволяє взаємодіяти з даними продуктів у базі даних

шляхом співставлення таблиць з класами та виконання операцій читання, створення, оновлення та видалення.

2.1.2 Реалізація Order Service

По-перше, слід зазначити, що реалізація Order Service ґрунтується на тих самих принципах, що й Customer Service. Order Service це окремий мікросервіс, побудований на базі технології Spring Boot. Це забезпечує консистентність у реалізації та управлінні обома сервісами.

Основний функціонал Order Service включає операції CRUD (створення, читання, оновлення та видалення), що дозволяє ефективно керувати процесом створення та управління замовленнями. Це означає, що користувачі можуть легко створювати нові замовлення, відстежувати їх статус, вносити зміни та видаляти непотрібні записи.

Однак головна причина винесення Order Service в окремий мікросервіс полягає в можливості подальших модифікацій та розширень. За допомогою мікросервісної архітектури, зміни в Order Service можуть здійснюватися незалежно від Customer Service. Це забезпечує гнучкість та легкість у впровадженні нового функціоналу, вдосконалення та змін.

Крім того, зазначено, що мікросервісна архітектура дозволяє додавати нові допоміжні сервіси, які можуть опрацьовувати логіку, не пов'язану із продуктами або замовленнями. Це сприяє розширенню функціональності системи загалом, надаючи можливість легко впроваджувати нові послуги та взаємодіяти з ними, не вносячи значних змін у основні сервіси.

Такий підхід до реалізації та управління сервісами підсилює модульність, гнучкість та масштабованість системи, що стає дуже важливим у сучасних умовах розвитку технологій та змін у бізнес-середовищі.

Серед змін у цьому сервісі дві нові таблиці: Order та OrderItem. Ці сутності зберігають інформацію про замовлення, якому користувачу воно належить, які продукти були замовлені та іншу інформацію.

Деталі цих сутностей можна побачити у таблицях 2.3 та 2.4

Таблиця 2.3 – зміст таблиці OrderItem.

Назва поля	Призначення
Id	Ключ
productVariantId	Унікальний ідентифікатор продукту
orderId	Унікальний ідентифікатор замовлення
price	Ціна продукту

Таблиця 2.4 – зміст таблиці Order.

Назва поля	Призначення
Id	Ключ
userId	Унікальний ідентифікатор користувача
totalPrice	Ціна замовлення

2.2 Реалізація Authentication Service

Наступним кроком у реалізації сервісу розподіленої інтернет комерції був крок з імплементацією сервісу аутентифікації.

Сервіс обробляє та валідує вхідні запити від користувачів та дозволяє йому виконати запит або ні. У рамках дипломної роботи була реалізована реєстрація та аутентифікація користувачів з використанням облікового запису в додатку, та з використанням протоколу OAuth2.

OAuth 2.0 (Open Authorization 2.0) - це протокол авторизації, який дозволяє користувачам надавати обмежений доступ своїм ресурсам іншим додаткам без передачі свого пароля. Цей протокол широко використовується в інтернет-додатках та API для забезпечення безпеки та контролю доступу.

Основні концепції в OAuth 2.0 включають Resource Owner (Власник ресурсів), Client (Клієнт), Authorization Server (Сервер авторизації), Resource Server (Сервер ресурсів), Authorization Grant (Дозвіл авторизації) та Access Token (Маркер доступу) (рисунок 2.4).

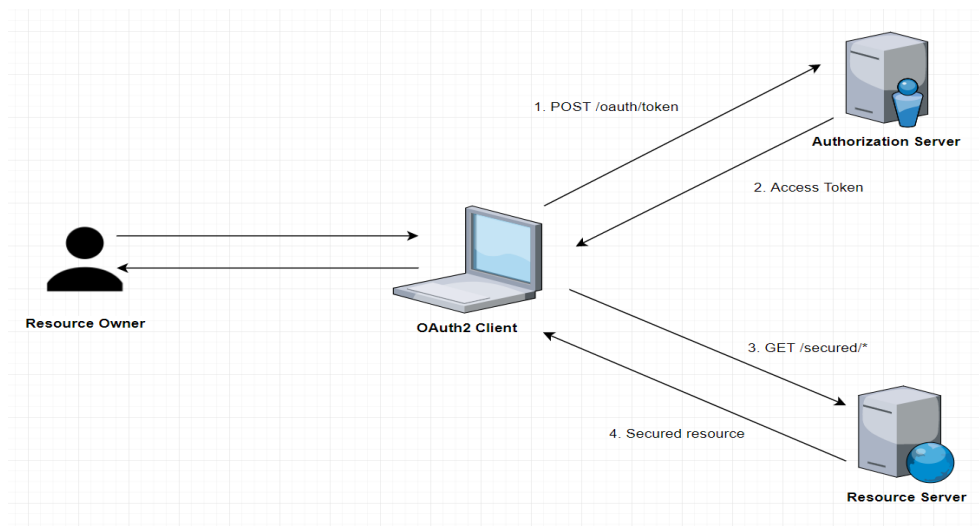


Рисунок 2.4 – процес доступу до ресурсу з використанням oAuth2.

Основний процес авторизації в OAuth 2.0 включає отримання авторизаційного запиту, направлення запиту до сервера авторизації, представлення запиту на дозвіл від власника ресурсів, обмін авторизаційного коду на маркер доступу та використання маркера доступу для отримання доступу до ресурсів. OAuth 2.0 використовується для забезпечення безпеки та контролю доступу до ресурсів в інтернет-сервісах і додатках.

Клас з налаштуваннями Spring Security можна побачити у лістингу А.1.

2.2.1 Реалізація доступу до даних за допомогою JWT

Доступ до ресурсів додатку здійснюється за допомогою Json Web Token (JWT) [17]. Цей тип захисту базується на тому, що у Authorization заголовку до запиту додається токен, що валідується сервером. У разі, коли токен валідний, запит виконується, інакше запит відхиляється.

Цей токен має невеликий час життя. Для того, щоб оновити його, використовується інший токен – refresh token. Він має значно більший термін дії, та використовується для того, щоб отримати новий JWT з серверу.

Для реалізації цього підходу необхідно реалізувати клас, що генерує токени (лістинг Б.1), та клас, що валідує токен (лістинг 2.7).

В реалізації валідатору токенів застосовано патерн програмування "Chain of Responsibility" (ланцюг відповідальності). Цей патерн


```

        if (authentication != null) {
SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    }
    } catch (JwtAuthenticationException e) {
        SecurityContextHolder.clearContext();
        throw new JwtAuthenticationException("Access
token is expired or invalid");
    }
}
filterChain.doFilter(servletRequest, servletResponse);
}
}
}

```

2.2.2 Реалізація класів сутностей для зберігання даних для логіну користувача

Окрім цього, за допомогою OAuth2 реалізовано логін за допомогою облікового запису Google [18].

Для відокремлення облікових записів додатку та Google створено роздільні таблиці з користувачами (таблиця 2.5).

Таблиця 2.5 – класи сутності з даними користувачів.

Назва таблиці	Призначення
BaseCredentialsEntity	Не є сутність, але є базовим класом зі спільними полями, що будуть успадковані класами-сутностями
ApplicationCredentialsEntity	Дані користувача у реалізованому додатку
GoogleCredentialsEntity	Дані облікового запису користувача Google

Клас BaseCredentialsEntity можна побачити на лістингу 2.8.

Лістинг 2.8 – Код класу BaseCredentialsEntity.

```

@MappedSuperclass
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public abstract class BaseCredentialsEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    protected Long id;
}

```

```

        protected String email;
        protected String password;
        @Enumerated(EnumType.STRING)
        protected UserRole role;
        protected LocalDateTime creationDate;
        @OneToOne
        @JoinColumn(name = "user_id")
        private UserEntity user;
        public BaseCredentialsEntity(String email, String password,
        UserRole role, LocalDateTime creationDate, UserEntity user) {
            this.email = email;
            this.password = password;
            this.role = role;
            this.creationDate = creationDate;
            this.user = user;
        }
    }
}

```

Код класів з даними користувача у додатку та даними облікового запису Google представлено у лістингах 2.9 та 2.10.

Лістинг 2.9 – Код класу ApplicationCredentialsEntity.

```

@Entity
@Table(name = "application_account_credentials")
@Getter
@Setter
@NoArgsConstructor
public class ApplicationCredentialsEntity extends
BaseCredentialsEntity {
    public ApplicationCredentialsEntity(String email, String
password, UserRole role, LocalDateTime creationDate, UserEntity
user){
        super(email, password, role, creationDate, user);
    }
}

```

Лістинг 2.10 – Код класу GoogleCredentialsEntity.

```

@Entity
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "google_account_credentials")
@Getter
@Setter
public class GoogleCredentialsEntity extends
BaseCredentialsEntity {
    private String clientId;
}

```

Паралельно з логікою, відповідальною за створення та валідацію токенів, реалізовано додатковий сервіс, який виконує роль посередника між контролером і токен-сервісом. У лістингу 2.13 можна побачити код цього сервісу. Його основна мета - забезпечити зручний інтерфейс для взаємодії з функціоналом токенів.

Сервіс виконує різноманітні завдання, такі як надання користувачу можливості отримати токен, оновити його або створити обліковий запис у додатку.

Коли запит користувача оброблено і токен валідовано, сервіс пересилає запит до ProductService або OrderService. Для відправки запитів з коду Java існує RestTemplate клас, що є частиною Spring Framework, який надає простий спосіб взаємодії з веб-сервісами за допомогою протоколу HTTP. Він дозволяє виконувати HTTP-запити до віддалених ресурсів і обробляти їхні відповіді.

Основні функції RestTemplate включають в себе відправлення HTTP-запитів (GET, POST, PUT, DELETE і т.д.), обробку відповідей в різних форматах (наприклад, JSON або XML), і взаємодію з різними типами ресурсів.

Отже, в ході реалізації сервісу аутентифікації реалізовано функціонал зі створення нових користувачів за допомогою даних користувача або його облікового запису. Також реалізовано захист з використанням JWT токена, функції надання токена користувачу та функції оновлення токена за допомогою використання refresh токена.

2.3 Реалізація Payments Service

Реалізований додаток, що являє собою сервіс електронної комерції, повинен включати в себе спеціальний сервіс для здійснення оплат. Цей сервіс

є ключовим компонентом для забезпечення успішної та ефективної електронної торгівлі.

У впровадженні платіжного сервісу важливо враховувати кілька ключових аспектів. По-перше, сервіс повинен бути інтегрований з різними платіжними системами та банками для підтримки різних методів оплати, таких як кредитні картки, електронні гроші та банківські перекази.

Однак не менш важливим є забезпечення високого рівня безпеки та конфіденційності під час обробки фінансових транзакцій. Використання шифрування та інших заходів захисту є важливим для запобігання можливих загроз безпеки.

Забезпечення відстеження та журналювання транзакцій дозволяє проводити подальший аналіз та звітність.

Такий платіжний сервіс стає необхідним елементом для оптимізації та покращення функціоналу електронної комерції, забезпечуючи користувачів безпечним та зручним механізмом проведення платежів.

2.3.1 Вибір платіжної системи

Перш за все було обрано платіжну систему. Оскільки не було сформовано чіткої вимоги з використання якоїсь певної платіжної системи, то обрано систему Aduen [19].

Aduen - це платіжна система та платіжний процесінговий провайдер, який надає комплексні рішення для обробки платежів. Aduen підтримує широкий спектр платіжних методів, включаючи кредитні та дебетові картки, електронні гроші, мобільні платежі та інші способи оплати у різних валютах та країнах.

Ця система обробляє як онлайн, так і офлайн платежі, забезпечуючи високий рівень безпеки та конфіденційності даних. Крім того, Aduen пропонує інструменти аналітики та звітності для відстеження та аналізу платежів.

Першим чином створено функціонал з додавання методу оплати користувача у систему додатка та у платіжну систему Aduen. Зазвичай, якщо платіжна система не надає для цього спеціального API, то це робиться шляхом

проведення оплати сумою в 0 одиниць валюти. У випадку з Adyen саме такий спосіб буде використовуватися для додання даних користувача у систему платіжного провайдера.

Для реалізації цього функціоналу та функціоналу з проведення оплати та повернення коштів використано паттерн програмування Strategy. У паттерні "Strategy" визначається сімейство алгоритмів, які інкапсулюються та роблять один одного взаємозамінними [20]. При цьому сам алгоритм може залежати від контексту, і програма може вибирати той чи інший алгоритм на льоту без перекомпіляції прямо у ході виконання програми.

Такий підхід дозволяє винести логіку для певного платіжного провайдера у сімейство окремих класів, що будуть викликатися коли використовується специфічний провайдер. Таким чином можна легко додати нову платіжну систему без зміни наявної логіки.

Найвищий метод в ієрархії викликів займає метод, що викликає метод стратегії для певної платіжної системи (лістинг 2.11).

Лістинг 2.11 – Початок обробки запиту на створення методу оплати

```
@Transactional
public                               ResponseEntity<PaymentMethodResponse>
createPaymentMethod(String          providerName, CreatePaymentMethodDto
request, Long userId)
throws ApiException
{
CreatePaymentMethodResponse          response          =
strategyRunner.createPaymentMethod
(aTenantId, aProviderRef, aRequest);
PaymentMethodWrapper                paymentMethod      =
CreatePaymentMethodRequestBuilder.build
(lStrategyResponse, aProviderRef, lRequestRef, aRequest);
validate(userId, paymentMethod);
paymentMethod = savePaymentMethod(paymentMethod, userId);
PaymentMethodResponse               lResponse          =          new
PaymentMethodResponse().paymentMethod(paymentMethod);
return new ResponseEntity<PaymentMethodResponse> (lResponse,
HttpStatus.CREATED);
}
```

2.3.2 Створення функціоналу зі збереження методу оплати користувача

Для створення оплати використовується спеціальний плагін для клієнтського додатку, що передає у зашифрованому вигляді дані картки користувача у систему Adyen. Результат цієї транзакції передається у PaymentsService, після цього у ході обробки викликається певна стратегія, що обробляє результат запиту. Код обробки можна побачити у листингу 2.12.

Лістинг 2.12 – Виконання методу створення методу оплати.

```
@Autowired
private Logger logger;

@Override
protected CreatePaymentMethodResponse
executeStrategy(CreatePaymentMethodStrategyRequest request, final
AdyenConfig config) throws BadInputException,
PaymentProviderException {

    AdyenProviderResponse response = parseProviderResponse(request,
    AdyenProviderResponse.class);
    if (StringUtils.isBlank(response.getPspReference())) {
        providerPaymentMethodRefMissingError();
    }
    CreatePaymentMethodStrategyResponse response =
    AdyenCreatePaymentMethodBuilder.build(request, response);
    validatePaymentMethod(response, request);
    return response;
}

@Override
protected String getProviderPaymentMethodRefParamName() {
    return "pspReference";
}

@Override
public Class getConfigurationClass() {
    return AdyenConfig.class;
}

private void validatePaymentMethod(CreatePaymentMethodResponse
response, CreatePaymentMethodRequest request) throws
BadInputException
{
    if (isSingleUse(response)) {
        logger.error("Unable to create one-off payment method {}", request.getProviderResponse());
        throw new BadInputException(
            ErrorCodes.PAYMENTGATEWAY_CANNOT_PARSE_PROVIDER_RESPONSE,
            "Invalid payment method pspReference/amount from provider response");
    }
    else if (response.isSingleUseMethod().equals(Boolean.FALSE) &&
        response.getPaymentMethodDetails() == null) {

```

```

logger.error          ("Unable          to          parse          {} ",
request.getProviderResponse());
throw new BadInputException (ErrorCodes.CANNOT_PARSE_RESPONSE,
"Cannot read payment method details from provider response"));}}

public                boolean                isSingleUse(final
CreatePaymentMethodStrategyResponse aResponse)
    {return  aResponse.isSingleUseMethod().equals(Boolean.TRUE) &&
(StringUtils.isEmpty(aResponse.getProviderReservedPaymentRef()) |
| aResponse.getAmount().equals(BigDecimal.ZERO));}

```

2.3.3 Створення функціоналу з проведення оплати користувача

Наступним кроком імплементовано функціонал з проведенням оплати. Взаємодіяти з платіжною системою можна шляхом сесій або ж через Adyen Арі. Приклад запитів та відповідей в обох випадках можна побачити на рисунках 2.5 та 2.6.

Як можна побачити, запити майже однакові, але ендпоінти для цих запитів різні. Також відрізняються відповіді серверу. У разі використання сесій Adyen повертає дані оплати у зашифрованому вигляді. Можливості їх розшифрувати немає, а тому з відповіді неможливо отримати додаткові деталі оплати, наприклад, унікальне посилання на транзакцію.

Тому, було обрано Арі Adyen з якого ці додаткові деталі отримати можливо. Загалом оплата відбувається у два етапи – витягування методу оплати для користувача з системи Adyen та виконання оплати через окремий ендпоінт. Оскільки функціонал зі збереження методу оплати додано раніше, необхідність роботи додатковий запит, щоб витягнути ідентифікатор методу оплати, відпадає. Залишається тільки створити тіло запиту та відправити POST запит до системи Adyen.

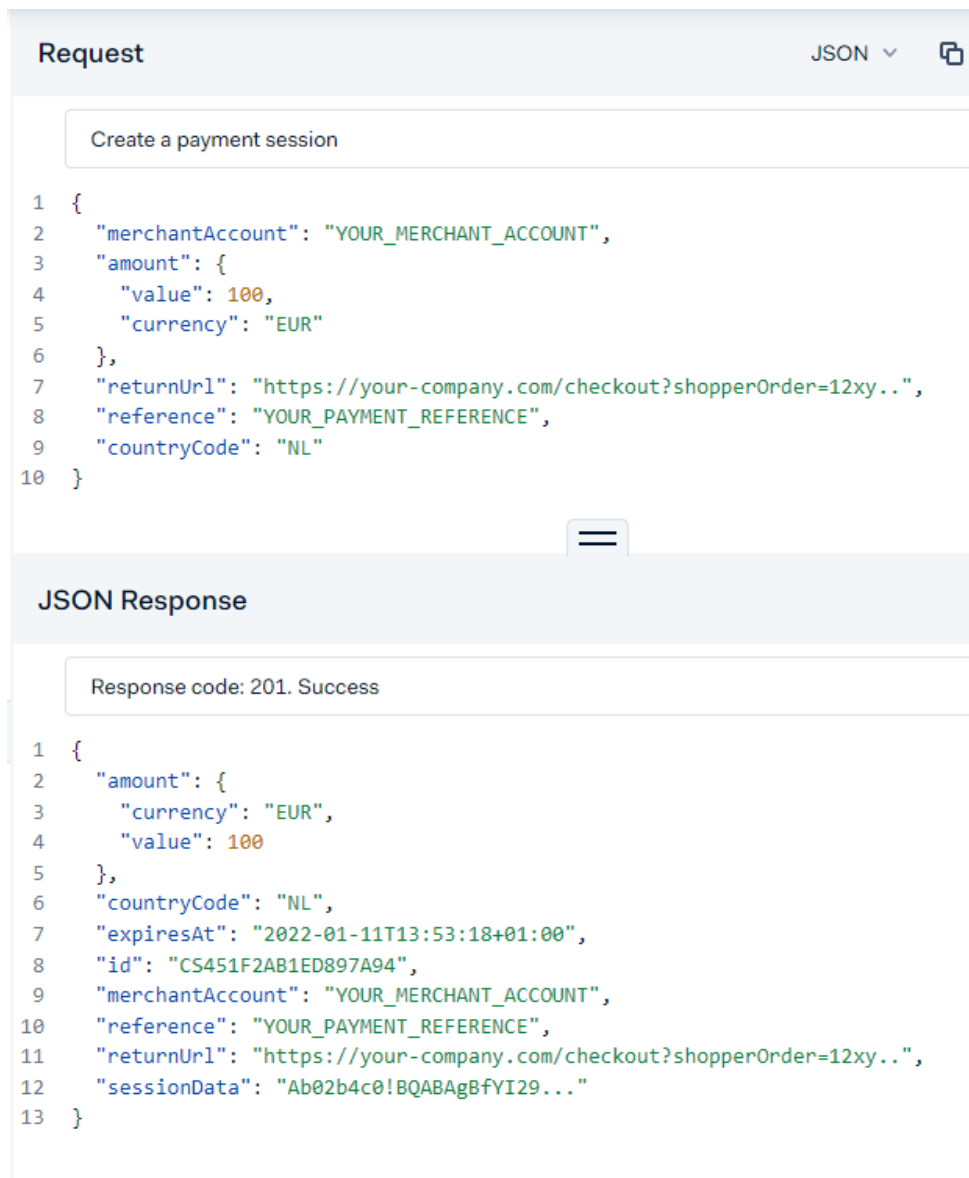


Рисунок 2.5 – приклад запиту та відповіді з використанням сесій Adyen.

2.3.4 Створення функціоналу з повернення коштів за виконану оплату

Фінальним етапом була імплементація методу повернення коштів. Для цього необхідно відправити POST запит у систему Adyen та додати до тіла запиту унікальний ідентифікатор транзакції. Також, результат повернення коштів завжди відправляються на вебхук (рис. 2.7).

Вебхуки Adyen - це механізм, який дозволяє серверу отримувати повідомлення про події, які сталися в системі Adyen. Webhooks

використовуються для інформування сервера про зміни стану платежів, повернень або інших подій, що стосуються обробки платежів.

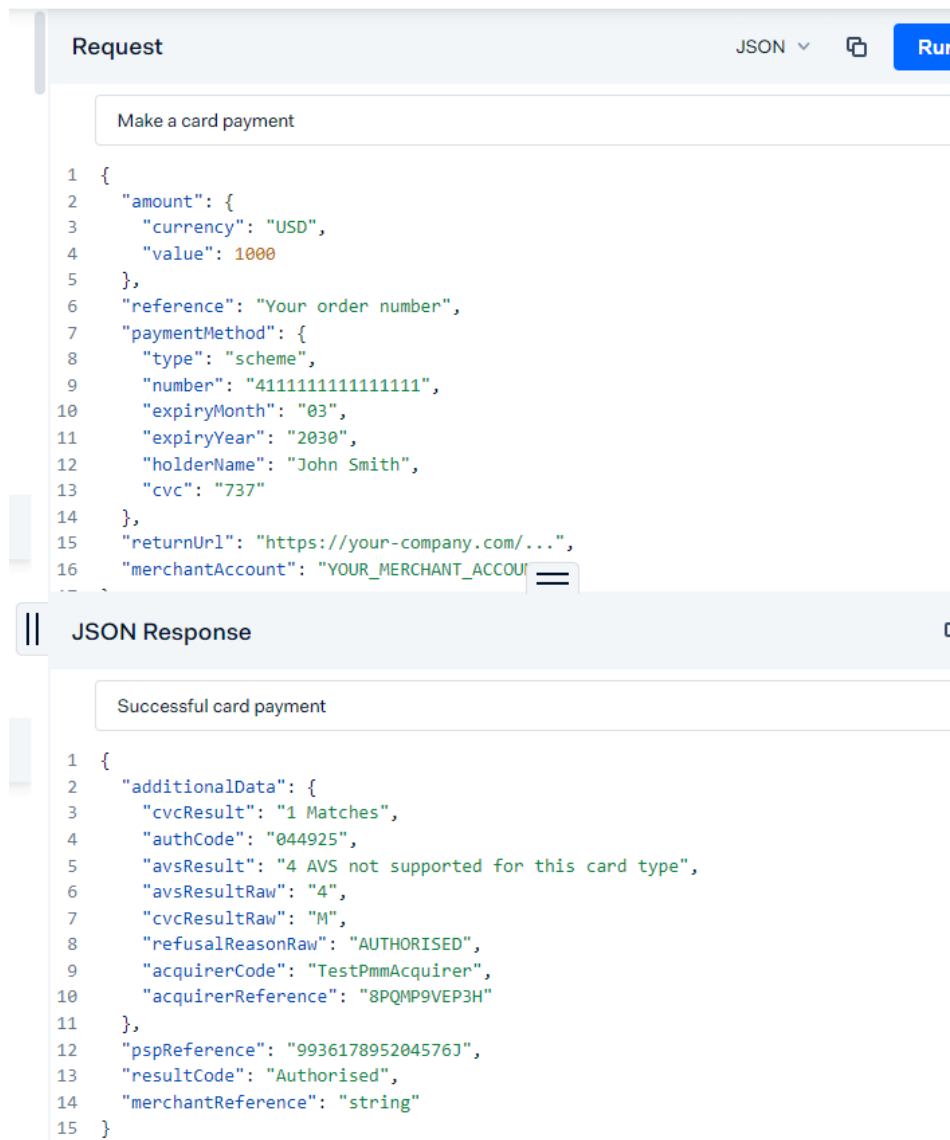


Рисунок 2.6 – приклад запиту та відповіді з використанням Арі Aduyen.

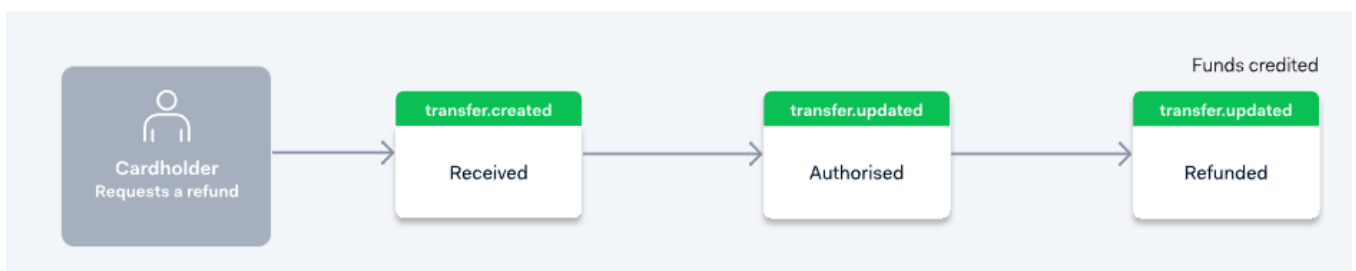


Рисунок 2.7 – Схема проведення повернення коштів у системі Aduyen з використанням вебхуків.

Код реалізації методу, що виконує запит до Adyen з повернення коштів можна побачити у лістингу 2.13.

Лістинг 2.13 – метод виконання повернення коштів.

```
@Override
protected RefundStrategyResponse
executeStrategy(RefundStrategyRequest request, AdyenConfig
config) throws PaymentProviderException {
    HttpHeaders headers = getHeaders(config);
    AdyenRefundPaymentRequest providerRequest =
    AdyenRefundBuilder.build(request, config);
    String url = config.getUrl() + PAYMENTS +
    request.getPayment().getProviderPaymentRef() + REFUNDS;
    ProviderResponse<AdyenRefundPaymentResponse,
    AdyenErrorResponse> providerResponse =
    getProviderClient().post(url, providerRequest, headers, AdyenRefund
    PaymentResponse.class, AdyenErrorResponse.class);
    int responseCode = providerResponse.getResponseCode();
    if (!responseCodes.contains(responseCode))
    {
        mLogger.info ("AdyenRefundOperation failed as
        response code from provider is {}", responseCode);
        AdyenErrorResponse refundResponse =
        providerResponse.getErrorResponseBody();
        throw new PaymentProviderException
        (refundResponse.getErrorType(),
        refundResponse.getMessage(),
        providerResponse.getResponseBodyAsString(),
        refundResponse.getPspReference());
    }
    RefundStrategyResponse response =
    AdyenRefundBuilder.build(providerResponse.getResponseBody(),
    providerResponse.getResponseBodyAsString());
    return response;}
```

3 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ НЕОБХІДНИХ ДЛЯ РЕАЛІЗАЦІЇ ТА ХМАРНИХ СЕРВІСІВ AMAZON

В еру стрімкого розвитку технологій та нестримної цифрової трансформації, важливість хмарних сервісів стає визначальною для ефективної реалізації програмного забезпечення. Як наступний етап після розділів, присвячених впровадженню сервісу електронної комерції, цей розділ містить інформацію про дослідження та аналіз технологій хмарних сервісів, необхідних для реалізації сервісу електронної комерції.

У попередніх розділах проведено аналіз хмарних технологій та впроваджено мікросервіси, що у сукупності є розподіленою системою електронної комерції. Для того, щоб ця система працювала і користувачі могли отримати доступ до її функціоналу, потрібно дослідити та обрати найбільш вдалі та доречні сервіси хмарних технологій. Саме тому цей розділ присвячено дослідженню та порівнянню хмарних технологій, що необхідні не лише для функціонування впроваджених мікросервісів, а й загалом для майже усіх веб та не тільки додатків.

Хмарні сервіси визначають новий стандарт у сфері інформаційних технологій, надаючи широкий спектр послуг, які спрощують реалізацію та підтримку програмного забезпечення. Важливість цих сервісів проявляється у здатності компаній адаптуватися до технологічного середовища, що швидко змінюється, та виходити за межі традиційних методів реалізації.

При розгляді хмарних сервісів, критерії порівняння стають ключовими в аналізі їхньої ефективності та придатності для конкретних завдань. Питання безпеки, масштабованості, доступності, вартості та інтегрованості стають домінуючими в контексті оцінки переваг та обмежень хмарних сервісів, особливо при їхньому використанні у впровадженні програмного забезпечення.

3.1 Порівняння та дослідження сервісів керування репозиторіями

Кожен додаток, що виходить за межі локального запуску та налаштування має знаходитись у спеціальному місці. Із цим допомагають сервіси керування репозиторіями - інструменти та платформи, які допомагають розробникам у зберіганні, відстеженні та управлінні версіями програмного коду. Репозиторій в цьому контексті вказує на місце, де зберігається весь код і пов'язана з ним інформація для певного проекту.

Основні функції сервісів керування репозиторіями включають:

- зберігання коду: сервіси репозиторіїв забезпечують централізоване зберігання вашого програмного коду. Розробники можуть завантажити свій код у репозиторій, де він зберігається та може бути використаний іншими членами команди;
- контроль версій: Системи керування репозиторіями дозволяють відстежувати зміни в коді та легко переключатися між різними версіями проекту;
- гілки (branches) та злиття (merge): розробники можуть створювати відокремлені гілки для роботи над конкретними функціями чи виправленнями помилок;
- відстеження змін: сервіси керування репозиторіями ведуть журнал змін, що дозволяє розробникам відстежувати, хто та коли вносив зміни до коду.

У ході дипломної роботи розглянуті, проаналізовані та порівняні між собою найбільш популярні системи керування репозиторіями, а саме: GitLab, GitHub, BitBucket [21].

3.1.1 Дослідження системи керування репозиторіями GitLab

Першою системою проаналізовано GitLab - це платформа для керування репозиторіями та співпраці розробників, яка включає в себе багато інших інструментів для автоматизації реалізації програмного забезпечення.

GitLab дозволяє створювати необмежену кількість репозиторіїв для проектів. Кожен репозиторій містить код та історію змін.

Система дозволяє розробникам створювати та управляти гілками для паралельної реалізації функціоналу. GitLab спрощує злиття гілок для об'єднання змін. Також GitLab має вбудований інструмент відстеження задач, що дозволяє створювати, призначати та відстежувати задачі. Це полегшує комунікацію та спільну роботу учасників проекту.

Окрім цього ця система керування репозиторіями надає можливості неперервної інтеграції (CI) та постійної поставки (CD). Це дозволяє автоматизувати тестування, збірку та розгортання коду.

За допомогою GitLab можна налаштовувати рівні доступу для користувачів і груп. Це дозволяє керувати тим, хто може робити зміни в репозиторії. Також надає засоби для виявлення та виправлення помилок безпеки в коді, забезпечує можливості автентифікації та авторизації для захисту ваших даних, та надає зручний графічний інтерфейс.

GitLab надає інструменти для моніторингу активності в репозиторіях та створення звітів. Це допомагає відстежувати продуктивність команди та розвиток проекту.

Узагальнюючи, GitLab - це потужний інструмент для керування репозиторіями, який надає широкі можливості для ефективного спільного розроблення програмного забезпечення. Його функціональність охоплює всі етапи реалізації, починаючи від відстеження задач і закінчуючи автоматизацією процесів CI/CD [22].

3.1.2 Дослідження системи керування репозиторіями GitHub

GitHub - це одна з найпопулярніших і найширше використовуваних систем керування репозиторіями, яка надає розширені засоби для спільної розробки програмного забезпечення.

Як і GitLab, дозволяє створювати та управляти репозиторіями для зберігання коду та історії змін, створювати, об'єднувати та видаляти гілки.

GitHub сприяє розповсюдженню відкритого коду, надаючи інфраструктуру для зручної співпраці розробників з усього світу, що робить цю систему гарним вибором для невеликих проєктів з відкритим вихідним кодом.

Має вбудований інструмент відстеження проблем, який дозволяє створювати, призначати та відстежувати проблеми та задачі.

Вбудована можливість неперервної інтеграції (CI) та неперервної поставки (CD) за допомогою сервісу GitHub Actions спрощує автоматизацію тестування, збірки та розгортання.

Дозволяє налаштовувати рівні доступу для користувачів та команд, що надає можливість гнучко керувати правами в репозиторіях.

Також ця система керування репозиторіями має маркетплейс, що надає широкий вибір додаткових інструментів та послуг, які можна легко інтегрувати з репозиторіями [23].

3.1.3 Дослідження системи керування репозиторіями BitBucket

Bitbucket - це система керування репозиторіями, яка надає інструменти для спільної реалізації програмного забезпечення. Власність компанії Atlassian, Bitbucket пропонує низку функціональних можливостей для зручного та ефективного управління кодом.

Система є не лише зручним інструментом для зберігання кодової бази, але й потужною системою керування версіями, яка вигідно виділяється своєю гнучкістю та багатофункціональністю. Подібно до попередніх розглянутих систем, Bitbucket надає можливість створення як публічних, так і приватних репозиторіїв, відкриваючи широкі можливості для зберігання коду та ведення історії змін.

Важливою особливістю Bitbucket є його підтримка гілок. Гілки дозволяють розробникам працювати над різними частинами проєкту паралельно, що полегшує спільну роботу та прискорює процес розробки. Кожна гілка може представляти окремий функціонал, виправлення помилок або нову функцію, а після завершення роботи гілки її зміни можуть бути

об'єднані з основною гілкою, щоб створити єдиний інтегрований код. Bitbucket надає можливість внесення внеску через pull-запити. Це спрощує код-рев'ю та обговорення змін в коді. Вбудований інструмент відстеження проблем дозволяє створювати та відстежувати завдання та різні проблеми в проекті. Bitbucket може інтегруватися з інструментами неперервної інтеграції (наприклад, Jenkins) для автоматизації тестування та розгортання коду.

Дозволяє налаштовувати рівні доступу для користувачів та груп, керуючи правами на читання, запис та адміністрування. Легко інтегрується з іншими продуктами компанії Atlassian, такими як Jira та Confluence, для покращення співпраці та обміну інформацією.

Система має Bitbucket Server для внутрішнього використання та Bitbucket Cloud (хмарний сервіс), який забезпечує масштабованість та доступність.

Також має інтуїтивний графічний інтерфейс для зручного користування, а також сервіс Bitbucket Pipelines для налаштування та автоматизації процесів CI/CD.

Система відзначається своєю гнучкістю, особливо в контексті інтеграції з іншими продуктами Atlassian. Надає розширені можливості для управління кодом та спільної реалізації, а також інтегрується з популярними інструментами для забезпечення повноцінного циклу впровадження програмного забезпечення [24].

3.1.4 Порівняння систем керування репозиторіями

Порівнюватися ці сервіси будуть за наступними критеріями:

- функціональність;
- безпека;
- наявність інструментів для розробників;
- цінова політика;
- інтеграції та розширення.

Порівняння систем можна побачити у таблиці 3.1.

Таблиця 3.1 – порівняння систем керування репозиторіями

Критерій	GitHub	GitLab	BitBucket
Функціональність	GitHub відомий своєю активною спільнотою та зручними Pull Requests для обговорення та об'єднання коду. Має вбудований інструмент для CI/CD та автоматизації робочих процесів. Можливість швидко розгортати статичні веб-сайти. Серед недоліків можна виділити обмежену кількість безплатних сервісів.	GitLab включає в себе вбудований інструмент Continuous Integration / Continuous Deployment (CI/CD). Надає можливість організації проектів в групи для кращого управління. Має зручний інструмент для ведення списку завдань та проблем. Серед недоліків можна виділити незручний інтерфейс, особливо для новачків.	Bitbucket часто використовується в середовищах, де вже використовуються інші продукти Atlassian, такі як Jira і Confluence. Безкоштовні приватні репозиторії для обмеженого числа користувачів. Вбудований інструмент для CI/CD. Серед недоліків можна виділити відсутність продвинутих функціональностей порівняно з GitHub і GitLab
Безпека	GitHub надає інструмент для пошуку та виправлення вразливостей у вихідному коді. Можливість захисту гілок для запобігання ненавмисним змінам. Серед недоліків є обмеження на обсяг файлів, яке може призвести до складнощів у роботі з великими даними.	Зручна система керування ролями та дозволами для керування доступом. Підтримка двофакторної аутентифікації. Як недолік, потребує налаштувань для повного використання функцій безпеки.	Можливість обмежити доступ до гілок для певних користувачів чи груп. Можливість обмежити доступ до репозиторію за IP-адресами. Як недолік, особливості безпеки менше розвинені порівняно з GitHub та GitLab.
Інструменти для розробників	GitHub надає низку інструментів для спільної роботи, включаючи Discussions та Project Boards. Можливість організації користувачів у команди.	Зручні інструменти для перегляду та обговорення коду в реальному часі. Можливість створювати власні сторінки для документації проекту.	Інтеграція з Jira для спільної роботи над задачами та проблемами. Підтримка різних моделей гілок для кращого управління робочим
Інтеграція та розширення	Велика кількість інтеграцій з іншими сервісами та інструментами, але деякі розширення та інтеграції платні.	Широкі можливості інтеграції, включаючи зовнішні CI/CD системи, проте деякі розширення можуть вимагати додаткової конфігурації.	Глибока інтеграція з іншими продуктами Atlassian, але обмежені можливості порівняно з іншими системами.

Продовження таблиці 3.1.

Критерій	GitHub	GitLab	BitBucket
Цінова політика	Безкоштовні публічні репозиторії, але приватні репозиторії потребують плати і вартість приватних репозиторіїв може бути високою для комерційних проектів.	Безкоштовні приватні репозиторії та різні варіанти ціноутворення, але напрямок на самостійне використання може бути складнішим для менших команд розробників.	Безкоштовні приватні репозиторії, можливість отримання безкоштовного доступу до інших продуктів Atlassian для невеликих команд. Але вартість може зростати зі збільшенням команди та використанням інших продуктів Atlassian.

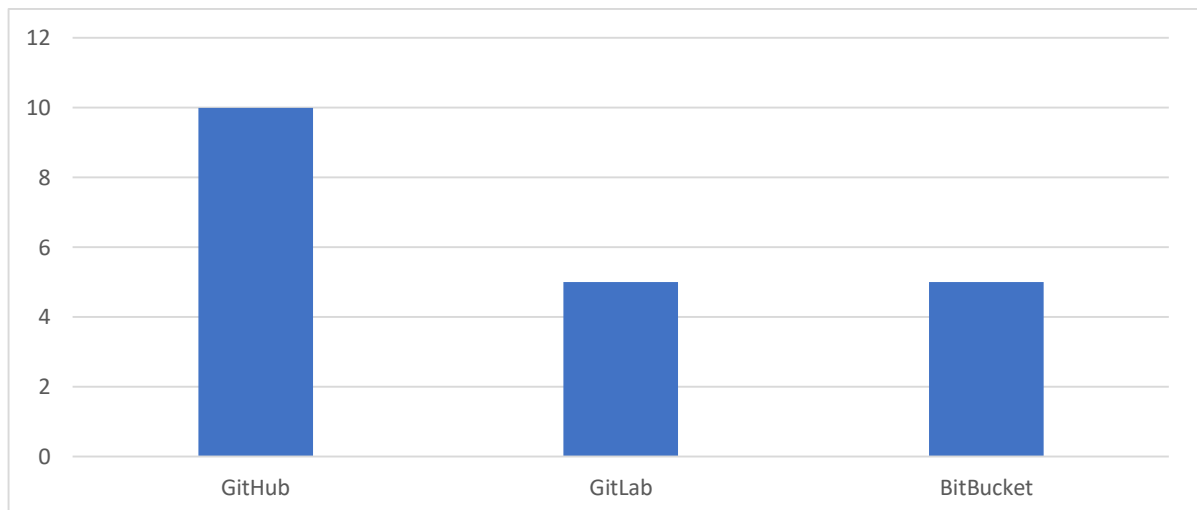


Рисунок 3.1 – Кількість безплатних користувачів на групу у системах керування репозиторіями GitHub, GitLab, BitBucket.

Отже, на основі порівняння цих систем керування репозиторіями можна зробити висновок, GitHub - найкращий вибір для великих відкритих проектів, де важлива екосистема. GitLab забезпечує повний набір інструментів для реалізації програмного забезпечення, ідеальний для команд, які цінують функціональність та гнучкість, а ідеальний для корпоративних команд, що вже використовують інші продукти Atlassian, та для невеликих команд, які шукають безкоштовні приватні репозиторії.

Для того, щоб розмістити код, що впроваджений у другому розділі ідеально може підійти GitLab, оскільки його можливості дозволяють створити надійних захист для вихідного коду, а інтеграція з CI/CD сервісами допоможе при розгортанні додатку на хмарних сервісах Amazon.

3.2 Дослідження сервісів Amazon для розгортання бази даних

Після аналізу та порівняння систем керування постає питання розгортання додатка та бази даних. Зробити це можна багатьма способами, але саме хмарні технології надають вже готове для цього рішення, яке можна конфігурувати та налаштовувати. У першому розділі був проведений аналіз технічного завдання, в ході якого обрано сервіси Amazon. Саме дослідження та порівняння цих сервісів проведене у цій частині.

Amazon пропонує на вибір багато сервіс для розгортання СУБД PostgreSQL, серед яких є AWS RDS, AWS AURORA та AWS EC2. Кожен з наведених сервісів має свої недоліки та переваги.

3.2.1 Дослідження AWS RDS

AWS RDS - керований сервіс баз даних, який пропонується Amazon Web Services (AWS). Цей сервіс призначений для спрощення створення, управління та масштабування реляційних баз даних в хмарному середовищі. Зокрема, AWS RDS підтримує різні типи баз даних, включаючи PostgreSQL, MySQL, MariaDB, Oracle та Microsoft SQL Server.

AWS RDS бере на себе багато рутинних завдань, пов'язаних з управлінням базами даних, таких як резервне копіювання, патчі, моніторинг та автоматичне резервне копіювання. Це дозволяє розробникам та адміністраторам баз даних фокусуватися на своїй роботі, не турбуючись про інфраструктурні завдання.

Сервіс дозволяє легко масштабувати базу даних. Можна змінювати типи інстансів, включаючи впровадження багатозоновості для високої доступності.

AWS RDS підтримує різні реляційні бази даних, що дозволяє вибрати той, який відповідає конкретним вимогам.

Сервіс надає інтегровані можливості для захисту вашої бази даних, такі як можливість налаштування груп безпеки, використання SSL для шифрування даних та можливість використання управління ключами AWS Key Management Service (KMS). Також є підтримка створення регулярних резервних копій баз даних.

Загалом, Amazon RDS є зручним інструментом для розгортання та управління реляційними базами даних в хмарному середовищі, роблячи багато рутинних завдань автоматизованими і спрощеними [25].

3.2.2 Дослідження AWS AURORA

Наступним проаналізовано AWS AURORA - керований сервіс баз даних у хмарному середовищі Amazon Web Services, який є альтернативою звичайним реляційним базам даних, таким як PostgreSQL. Amazon Aurora побудований на основі технології розподіленого зберігання даних та має за мету забезпечити високу доступність, надійність та продуктивність бази даних.

Amazon Aurora PostgreSQL сумісний з PostgreSQL, що робить його відмінним вибором для PostgreSQL, оскільки він підтримує ті самі команди, дані та синтаксис.

Сервіс забезпечує високу доступність та відновлення після збоїв за рахунок розподіленого зберігання даних і автоматичного резервного копіювання, окрім цього дозволяє легко масштабувати ресурси бази даних вгору або вниз в залежності від потреб.

Використання покращеного рушія зберігання даних надає високий рівень продуктивності порівняно зі стандартним PostgreSQL [26].

Може здатися, що сервіс дуже схожий на AWS RDS, але він має свої відмінності, які наведені у таблиці 3.2.

Таблиця 3.2 – Відмінності між AWS RDS та AWS AURORA PostgreSQL

Критерій	AWS RDS	AWS AURORA
Сумісність з PostgreSQL	Дозволяє розгорнути бази даних PostgreSQL безпосередньо на управлюваних серверах баз даних. Використання Amazon RDS, схоже на стандартний досвід роботи з PostgreSQL, можна встановлювати різноманітні розширення та інструменти PostgreSQL	Це високодоступний та масштабований сервіс баз даних, який сумісний з PostgreSQL. Однак, він не є чистим PostgreSQL, але має свою внутрішню архітектуру, яка робить його більш ефективним порівняно зі звичайним PostgreSQL
Продуктивність та масштабованість	Надає різні типи інстансів, але масштабування може бути менш гнучким порівняно з Aurora.	Зазвичай має кращу продуктивність та масштабованість, особливо при великому обсязі роботи з даними. Він використовує спеціалізований рушій зберігання даних, який дозволяє досягти високої швидкодії та ефективності.
Вартість	Зазвичай, вартість використання Amazon RDS є менше порівняно з Amazon Aurora, але це може залежати від конкретних потреб та обсягів роботи з даними.	Може бути дорожчим у порівнянні з Amazon RDS, але це компенсується покращеною продуктивністю та можливістю масштабування.

Згідно порівняння у таблиці 3.2 можна зробити висновок, що AWS RDS більш простіший у використанні сервіс, ніж AWS AURORA PostgreSQL.

3.2.3 Дослідження AWS EC2

Amazon Elastic Compute Cloud (Amazon EC2) - це сервіс в хмарному середовищі, що надається Amazon Web Services, який дозволяє користувачам орендувати віртуальні сервери, відомі як "екземпляри", для розгортання та виконання різних додатків і завдань. Amazon EC2 надає гнучкість та широкий спектр можливостей для обчислень в хмарному середовищі.

Екземпляри Amazon EC2 є віртуальними серверами, які можна легко розгорнути та керувати. Доступні різні типи екземплярів, що відрізняються за обчислювальною потужністю, пам'яттю та іншими характеристиками.

EC2 оперує спеціальними образами - віртуальними образами, які містять необхідні компоненти для створення екземпляра, такі як операційна система, програмне забезпечення та конфігураційні дані.

AWS поділений на регіони, які представляють собою фізично відокремлені області на географічному рівні. Кожен регіон має кілька доступних зон, які представляють собою ізольовані центри обчислень. Вони дозволяють масштабувати екземпляри вгору або вниз в залежності від потреб. Користувач може автоматизувати процеси масштабування за допомогою сервісів AWS Auto Scaling. AWS EC2 надає можливість використовувати Virtual Private Cloud (VPC) для ізоляції інфраструктури та налаштування мережевих параметрів, також він легко інтегрується з іншими сервісами AWS, такими як Amazon S3 для зберігання даних, Amazon RDS для управління базами даних, тощо.

Вартість використання Amazon EC2 залежить від кількості ресурсів, які використовуються, і часу їх використання.

Amazon EC2 надає користувачам гнучкість та контроль над обчислювальними ресурсами в хмарному середовищі, що робить його популярним серед розробників та підприємств для різноманітних завдань, від розгортання веб-сайтів до обробки великих обчислювальних завдань.

Інакше кажучи, цей сервіс надає у користування віртуальну машину з тою операційною системою, яку обирає користувач. Це дозволяє самостійно розгорнути сервер бази даних та налаштувати його так, як потрібно. Цей підхід більш ресурсозатратний та потребує значно більше часу. Також керування цією віртуальною машиною може бути не зручним, оскільки, наприклад віртуальна машина на Linux може не містити графічного інтерфейсу, а отже всі операції потрібно проводити через консоль. Крім цього, уся інфраструктура буде налаштовуватись вручну [27].

3.2.4 Висновок дослідження

Досліджені сервіси AWS для розгортання СУБД. Кожен сервіс має свої переваги та недоліки. У ході дослідження було встановлено, що сервіс AWS

RDS потребує мінімальних затрат ресурсів та часу для того, щоб розгорнути свою базу даних. EC2 потребує ручної конфігурації усіх елементів, а отже використовувати його для розгортання бази даних для сервісу електронної комерції є недоцільним. Також досліджено AWS AURORA, що також є чудовим сервісом для розгортання, але він має більші цінові параметри, через що AWS RDS є найкращим вибором для розгортання бази даних для сервісу електронної комерції.

3.3 Дослідження сервісів Amazon для розгортання мікросервісів

Для розгортання мікросервісів в інфраструктурі AWS можна використовувати багато сервісів, але у ході дипломної роботи були досліджені такі сервіси як AWS Elastic Beanstalk, AWS Lambda, AWS EC2.

3.3.1 Дослідження AWS Lambda

AWS Lambda - це сервіс в хмарному середовищі від Amazon Web Services (AWS), який дозволяє розробникам виконувати код безпосередньо в хмарному середовищі без необхідності вручну керувати інфраструктурою. AWS Lambda використовується для створення та виконання функцій, що дозволяє сконцентруватися на написанні коду, не втрачаючи час на налаштування та управління серверами.

AWS Lambda використовує концепцію "безсерверної" архітектури, де завантажуються лише код, а AWS автоматично керує усіма іншими аспектами, такими як масштабування та управління ресурсами (рисунок 3.2).

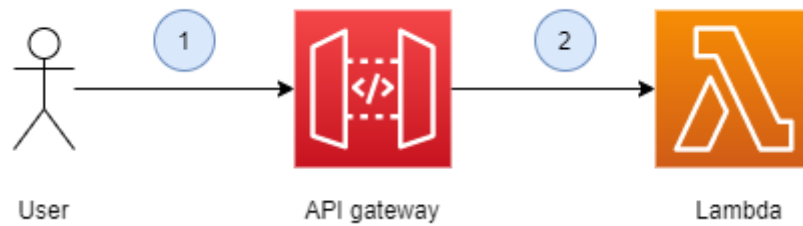


Рисунок 3.2 – Приклад безсерверної архітектури з використанням Арі Gateway.

Функції AWS Lambda можуть бути викликані відповідно до подій, таких як зміни в Amazon S3, оновлення в базі даних Amazon DynamoDB, або виникнення нового повідомлення в Amazon Simple Notification Service (SNS).

Сервіс підтримує автоматичне масштабування, що є його безперечною перевагою. Коли користувач викликає сервіс, що розгорнуто у Lambda, Lambda створює контейнер, та ініціалізує інстанс мікросервісу. В залежності від кількості запитів до Lambda, сервіс буде створювати додаткові контейнери автоматично, що зменшує необхідність самотійно займатись масштабуванням. При цьому плата списується тільки за час роботи функції.

AWS Lambda дозволяє розробникам створювати легкі, ефективні та автоматизовані рішення без необхідності керування інфраструктурою. Це особливо корисно для сценаріїв, де потрібна швидка реакція на події чи обробка навантаження в залежності від потреб [28].

3.3.2 Дослідження AWS Elastic Beanstalk

AWS Elastic beanstalk - це сервіс у хмарному середовищі від Amazon Web Services, який надає простий та автоматизований спосіб розгортання, масштабування та управління веб-застосунками.

AWS автоматично керує ресурсами, такими як віртуальні машини, мережа та бази даних, щоб спростити розгортання та управління застосунками. Це надає можливість легко масштабувати застосунок вгору або вниз в залежності від навантаження за допомогою інтерфейсу керування AWS або автоматичного масштабування.

Elastic Beanstalk інтегрується з іншими сервісами AWS, такими як Amazon RDS для баз даних, Amazon S3 для зберігання файлів, та багатьма іншими.

Сервіс надає засоби для оновлення застосунків та деплою нових версій, включаючи можливість розгортання застосунків з використанням Continuous Integration/Continuous Deployment (CI/CD) [29].

3.3.3 Порівняльна характеристика сервісів AWS для розгортання мікросервісів

Проведено порівняння сервісів AWS, яке можна побачити у таблиці 3.3.

На основі проведеного дослідження та порівняння різних сервісів для розгортання мікросервісної архітектури, виявлено, що AWS Elastic Beanstalk є ідеальним вибором для ефективного впровадження та управління мікросервісами, які були описані в другому розділі. Цей сервіс відзначається численними перевагами, роблячи його оптимальним рішенням для проектів, де ключовою є швидкість впровадження, автоматизація та надійність.

Однією з основних переваг AWS Elastic Beanstalk є його мінімальна потреба у великій кількості налаштувань. Це значно спрощує процес розгортання мікросервісів, забезпечуючи розробникам можливість швидко перевести свої рішення в обліковий запис AWS без великого обсягу попередніх підготовок.

Автоматичне масштабування - це ще одна ключова особливість Elastic Beanstalk, яка дозволяє автоматично адаптувати розмір та кількість ресурсів під навантаження системи. Це не тільки робить процес масштабування ефективним та еластичним, але й дозволяє економити ресурси та забезпечує безперервну доступність для користувачів.

Загалом, вибір AWS Elastic Beanstalk для розгортання мікросервісної архітектури є стратегічним рішенням, яке забезпечить ефективність, автоматизацію та високу доступність системи, що, в свою чергу, сприятиме успішному впровадженню та функціонуванню мікросервісів.

Таблиця 3.3 – порівняння сервісів AWS для розгортання мікросервісів

-	AWS Elastic Beanstalk	AWS Lambda	AWS EC2
Переваги	Elastic Beanstalk дозволяє розгорнути застосунки з найменшими зусиллями. Всі аспекти інфраструктури автоматизовані, що спрощує весь процес. AWS Elastic Beanstalk автоматично керує інфраструктурою, включаючи віртуальні машини, мережу та бази даних, забезпечуючи безпеку та масштабованість. Легко масштабувати застосунок вгору або вниз в залежності від навантаження, використовуючи автоматичне масштабування або ручне керування. Легка інтеграція з іншими сервісами AWS, такими як Amazon RDS, Amazon S3, Amazon SQS, AWS Identity and Access Management (IAM) та іншими. Зручні інструменти для деплою нових версій застосунків, включаючи можливість розгортання застосунків з використанням CI/CD.	Lambda працює у безсерверному режимі, що означає відсутність потреби в управлінні серверами та інфраструктурою. Платіжна модель Lambda базується на фактичному використанні ресурсів. Lambda автоматично масштабується вгору або вниз, в залежності від обсягу оброблюваних подій, що робить його ідеальним для реалізації масштабованих та високонавантажених застосунків. Підтримка різних мов програмування, включаючи Node.js, Python, Java, C#, Ruby та інші. Проста інтеграція з іншими сервісами AWS, такими як Amazon S3, Amazon DynamoDB, Amazon Kinesis та іншими. Реальний Час та Можливість виконувати код у реальному часі, що дозволяє створювати реактивні та спрощені застосунки. AWS пропонує безкоштовний рівень використання, який дозволяє виконувати певну кількість функцій без оплати.	EC2 дозволяє легко масштабувати кількість віртуальних машин вгору або вниз в залежності від обсягу роботи. Існує великий вибір типів інстансів, які можна налаштовувати під конкретні потреби щодо обчислювальної потужності, пам'яті, мережі та інших характеристик. EC2 надає повний контроль користувачеві над віртуальною машиною, що дозволяє використовувати власні образи, налаштувати мережу та здійснювати інші налаштування. EC2 легко інтегрується з іншими сервісами AWS, що дозволяє створювати комплексні рішення. Забезпечує високий рівень доступності через можливість розгортання в різних регіонах та доступ до аварійних зон. Користувач може вибирати операційну систему, яку буде використовувати віртуальна машина. Використовується для різноманітних завдань, від хостингу веб-застосунків до обчислювальної потужності високої продуктивності.

Продовження таблиці 3.3

-	AWS Elastic Beanstalk	AWS Lambda	AWS EC2
Недоліки	Можливості налаштування інфраструктури можуть бути обмеженими порівняно з іншими, менш абстрактними сервісами AWS, такими як Amazon EC2. Для деяких складних або нестандартних випадків інфраструктури розгортання Elastic Beanstalk може бути менш гнучким. Платіжна модель може бути менш передбачуваною, особливо при великому обсязі трафіку або інтенсивному використанні ресурсів. Elastic Beanstalk може обмежити можливість вибору операційної системи та стеку компонентів. Використання Elastic Beanstalk передбачає велику абстракцію, що може бути неперевагою для деяких проектів, де необхідна повна контроль над інфраструктурою.	Обмеження часу виконання функції (зазвичай 15 хвилин), що може бути проблематичним для завдань, які вимагають тривалої обробки. Локальна реалізація та тестування можуть бути складними через специфічний характер безсерверної архітектури. Lambda розглядається як "stateless" сервіс, тобто він не зберігає стан між викликами, що може вимагати інших AWS-сервісів для зберігання даних. Вартість використання Lambda також залежить від величини виділеної пам'яті та інших ресурсів, що може впливати на вартість застосунків. Обмежені можливості зберігання та обробки локальних даних без використання інших AWS-сервісів. При періодичних викликах може виникати затримка, відома як "холодний старт", коли новий екземпляр середовища Lambda активується.	В порівнянні з безсерверними рішеннями, вартість використання EC2 може бути менш передбачуваною. Користувач повинен самостійно взяти на себе відповідальність за адміністрування та управління інфраструктурою віртуальних машин. Час необхідний для розгортання нових інстанцій може бути довший порівняно з безсерверними рішеннями. Для масштабування потрібно вручну керувати обсягом ресурсів та взаємодіяти з різними інстанціями. Забезпечення безпеки складніше порівняно з деякими безсерверними архітектурами.

ВИСНОВКИ

У ході виконання дипломної роботи було визначено технічне завдання, реалізовано сервіс електронної комерції та проведено дослідження сервісів AWS, що потрібно використовувати для розгортання реалізованих мікросервісів.

У першій частині проаналізовані засоби, необхідні для впровадження дипломної роботи.

Проведено огляд програмних засобів, необхідних для реалізації системи.

Проаналізовані сучасні мови програмування та фреймворки, що використовуються для створення веб-додатків. У якості основної мови програмування обрано мову Java з фреймворком Spring.

Проведений огляд SQL та NoSQL баз даних, їх особливостей. У якості СУБД обрана PostgreSQL.

Також оглянуті сервіси хмарних обчислень. Для виконання дипломної роботи були обрані Amazon Web Services.

У другому розділі реалізовано мікросервіси електронної комерції, по ходу реалізації були використані бази даних, паттерни програмування. Також була використана платіжна система Adyen, для забезпечення проведення оплати.

Захист даних та доступ до них реалізовано за допомогою Spring Security та Json Web Token технології. Впроваджено аутентифікацію за допомогою сторонніх сервісів, а саме за допомогою облікового запису Google.

У третьому розділі проведено дослідження систем керування репозиторіями та сервісів Amazon.

У ході дослідження виявлено, що система керування GitLab найбільш доцільна для зберігання та версіонування вихідного коду впроваджених мікросервісів.

Для хостингу серверу бази даних у ході дослідження виявлено AWS RDS, через те, що сервіс не потребує багато часу та ресурсів на налаштування та надає зручний функціонал для менеджменту СУБД.

Для розгортання мікросервісів обрано AWS Elastic Beanstalk, адже сервіс надає спрощений функціонал з розгортання та підтримки мікросервісів, та надає автоматичне масштабування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Киричек Г.Г., Шавло Г.В. Система підтримки електронної комерції. Міжнародна науково-практична конференція «Наука, техніка і технології: глобальні тенденції, проблеми та рішення», Прага, Чеська Республіка, 25–26 вересня 2020, С.37-40.
2. Schildt, Herbert. Java: a beginner's guide. McGraw-Hill Education, 2022.
3. Bawiskar, Ankur, et al. Spring Framework: A Companion to JavaEE. IJCEM 1 (2012): 41-49.
4. Doglio, Fernando, and Fernando Doglio. "An Introduction to TypeScript." Introducing Deno: A First Look at the Newest JavaScript Runtime (2020): 27-62.
5. Sabo, Mario. NestJS. Diss. Josip Juraj Strossmayer University of Osijek. Department of Mathematics. Chair of Applied Mathematics. Computer Science Research Group, 2020.
6. Skeet, Jonathan. C# in Depth. Simon and Schuster, 2019.
7. Whitesell, Sean, et al. ASP. NET Core Overview. Pro Microservices in. NET 6: With Examples Using ASP. NET Core 6, MassTransit, and Kubernetes (2022): 29-49.
8. Sharma, Vatika, and Meenu Dave. Sql and nosql databases. International Journal of Advanced Research in Computer Science and Software Engineering 2.8 (2012).
9. Juba, Salahaldin, Achim Vannahme, and Andrey Volkov. Learning PostgreSQL. Packt Publishing Ltd, 2015.
10. Banker, Kyle, et al. MongoDB in action: covers MongoDB version 3.0. Simon and Schuster, 2016.
11. Wittig, Michael, and Andreas Wittig. Amazon web services in action. Simon and Schuster, 2018.

12. Collier, Michael, and Robin Shahan. Microsoft Azure Essentials-Fundamentals of Azure. Microsoft Press, 2015.
13. Geewax, John J. JJ. Google Cloud platform in action. Simon and Schuster, 2018.
14. Arthur, John, and Shiva Azadegan. "Spring Framework for rapid open source J2EE Web Application Development: A case study." Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Network. IEEE, 2005.
15. Pollack, Mark, et al. Spring Data: modern data access for enterprise Java. "O'Reilly Media, Inc.", 2012.
16. Spilca, Laurentiu. Spring security in action. Simon and Schuster, 2020.
17. Hong, Namsu, et al. "A study on a JWT-based user authentication and API assessment scheme using IMEI in a smart home environment." Sustainability 9.7 (2017): 1099.
18. Nguyen, Quy, and Oras F. Baker. "Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API." J. Softw. 14.6 (2019): 257-264.
19. Misback, Ann E. "Adyen, NV: Amsterdam, The Netherlands." Fed. Res. Bull. 107 (2021): 61.
20. Proulx, Viera K. "Programming patterns and design patterns in the introductory computer science course." ACM Sigcse Bulletin 32.1 (2000): 80-84.
21. Hethey, Jonathan M. GitLab repository management. Packt Publishing, 2013.
22. GitLab, B. V. "Gitlab." The One DevOps Platform (2022).
23. Cosentino, Valerio, Javier Luis, and Jordi Cabot. "Findings from GitHub: methods, datasets and limitations." Proceedings of the 13th International Conference on Mining Software Repositories. 2016.

24. Chakraborty, Sudip, and P. S. Aithal. "A Practical Approach to GIT Using Bitbucket, GitHub and SourceTree." *International Journal of Applied Engineering and Management Letters (IJAEML)* 6.2 (2022): 254-263.
25. Lakshmi, Narasimhan G. "Database Migration on Premises to AWS RDS." *EAI Endorsed Transactions on Cloud Systems* 3.11 (2018).
26. Verbitski, Alexandre, et al. "Amazon aurora: Design considerations for high throughput cloud-native relational databases." *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017.
27. Malta, Eva Maia, Sandra Avila, and Edson Borin. "Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications." *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 2019.
28. Edlund, Lucas, and Nils Stridbeck. "Exploring Cloud Architectures in AWS for Generating and Sending Real-time Data."
29. Vliet, Jurg, et al. *Elastic beanstalk*. " O'Reilly Media, Inc.", 2011.

ДОДАТОК А

Лістинг А.1 – Конфігурація модулю Spring Security.

```

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true,
securedEnabled = true, jsr250Enabled = true)
@RequiredArgsConstructor
@Profile("dev")
public class SecurityConfig extends WebSecurityConfigurerAdapter
implements WebMvcConfigurer {
    private final JwtFilter jwtFilter;
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/v2/api-docs",
            "/configuration/ui",
            "/swagger-resources/**",
            "/configuration/security",
            "/swagger-ui.html",
            "/webjars/**");
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.cors()
            .and()
            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy
            .STATELESS)
            .and()
            .exceptionHandling()
            .defaultAuthenticationEntryPointFor(
                authenticationEntryPoint(),
                new
                AntPathRequestMatcher("/login/oauth2/code/google"))
            .and()
            .authorizeRequests()
            .antMatchers("/auth/sign-up").permitAll()
            .antMatchers("/auth/sign-in").permitAll()
            .antMatchers("/auth/refresh").permitAll()
            .anyRequest()
            .authenticated()
            .and()
            .oauth2Login()
            .successHandler((HttpServletRequest,
                HttpServletResponse, authentication) -> {

                    OAuth2AuthenticationToken
                    oauth2Token = (OAuth2AuthenticationToken) authentication;

```

```

    httpRequest.setAttribute("oauthToken", oauth2Token);
    httpRequest.setAttribute("auth", authentication);
    httpRequest.getRequestDispatcher(String.format("/auth/login/oauth2/%s", oauth2Token.getAuthorizedClientRegistrationId())).
    forward(httpServletRequest, httpServletResponse);
    });
    http.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
}
@Override
public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**")
        .allowedOrigins("*")
        .allowedMethods("*");
}
@Bean
@Override
public AuthenticationManager authenticationManagerBean()
throws Exception {
    return super.authenticationManagerBean();
}
@Bean
public AuthenticationEntryPoint authenticationEntryPoint(){
    return new CustomAuthenticationEntryPoint();
}
@Bean
public PasswordEncoder encoder() {
    return new BCryptPasswordEncoder();
}
}

```

ДОДАТОК Б

Лістинг Б.1 – Клас генератор JWT та Refresh токенів.

```

@Component
@RequiredArgsConstructor
public class JwtProvider {

    private final UserDetailsService userDetailsService;
    @Value("${jwt.secret.access}")
    private String accessSecretKey;
    @Value("${jwt.secret.refresh}")
    private String refreshSecretKey;
    @Value("${jwt.expiration.access}")
    private int jwtValidity;
    @Value("${jwt.expiration.refresh}")
    private int refreshValidity;
    @Value("${jwt.header}")
    private String authorizationHeader;
    @PostConstruct
    protected void init(){
        accessSecretKey =
Base64.getEncoder().encodeToString(accessSecretKey.getBytes());
        refreshSecretKey =
Base64.getEncoder().encodeToString(refreshSecretKey.getBytes());
    }
    public String createToken(String email, String role){
        Claims claims = Jwts.claims().setSubject(email);
        claims.put("role", role);
        Date now = new Date();
        Date validity = addHoursToJavaUtilDate(now,
jwtValidity);

        return Jwts.builder()
            .setClaims(claims)
            .setIssuedAt(now)
            .setExpiration(validity)
            .setId("access")
            .signWith(SignatureAlgorithm.HS256,
accessSecretKey)
            .compact();
    }

    public String createRefreshToken(String email, String role){
        Claims claims = Jwts.claims().setSubject(email);
        claims.put("role", role);
        Date now = new Date();
        Date validity = addHoursToJavaUtilDate(now,
refreshValidity);

        return Jwts.builder()

```

```

        .setClaims(claims)
        .setIssuedAt(now)
        .setExpiration(validity)
        .setId("refresh")
        .signWith(SignatureAlgorithm.HS256,
refreshSecretKey)
        .compact();
    }

    public boolean validateToken(String token){
        try{
            Jws<Claims> claimsJws =
Jwt.parser().setSigningKey(accessSecretKey).parseClaimsJws(token);
            if (!claimsJws.getBody().getId().equals("access"))
                throw new RefreshTokenInvalidException("Current
token is not access token");
            return
!claimsJws.getBody().getExpiration().before(new Date());
        } catch (JwtException | IllegalArgumentException e){
            throw new JwtAuthenticationException("Jwt token is
expired or invalid");
        }
    }

    public boolean validateRefreshToken(String token){
        try{
            Jws<Claims> refreshToken =
Jwt.parser().setSigningKey(refreshSecretKey).parseClaimsJws(token);
            if
(!refreshToken.getBody().getId().equals("refresh"))
                throw new RefreshTokenInvalidException("Current
token is not refresh token");
            return
!refreshToken.getBody().getExpiration().before(new Date());
        } catch (JwtException | IllegalArgumentException e){
            throw new JwtAuthenticationException("Refresh token
is expired or invalid");
        }
    }

    public Authentication getAuthentication(String token){
        UserDetails userDetails =
this.userDetailsService.loadUserByUsername(getEmail(token));
        return new
UsernamePasswordAuthenticationToken(userDetails, "",
userDetails.getAuthorities());
    }

    public String getEmail(String token){
        try{
            return
Jwt.parser().setSigningKey(accessSecretKey).parseClaimsJws(token).getBody().getSubject();

```

```

        }catch (ExpiredJwtException e){
            return e.getClaims().getSubject();
        }
    }

    public Jws<Claims> getRefreshToken(String token){
        if(validateRefreshToken(token))
            return
Jwts.parser().setSigningKey(refreshSecretKey).parseClaimsJws(tok
en);
        else return null;
    }

    public String resolveToken(HttpServletRequest request){
        return request.getHeader(authorizationHeader);
    }

    public Date addHoursToJavaUtilDate(Date date, int minutes) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        calendar.add(Calendar.MINUTE, minutes);
        return calendar.getTime();
    }
}

```

ДОДАТОК В

Лістинг В.1 – Код логіки сервісу, що зв’язує контролер та токен-сервіс.

```
@Service
@RequiredArgsConstructor
public class AuthServiceImpl implements AuthService {
    private final AuthenticationManager authenticationManager;
    private final ApplicationCredentialsRepository
applicationCredentialsRepository;
    private final JwtProvider jwtProvider;
    private final GoogleCredentialsRepository
googleCredentialsRepository;
    private final FacebookCredentialsRepository
facebookCredentialsRepository;
    private final GithubCredentialsRepository
githubCredentialsRepository;
    private final UserService userService;

    public TokenDto signIn(LoginRequestDto loginRequestDto) {
        authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(loginRequestDto.getEmail(),
loginRequestDto.getPassword()));
        ApplicationCredentialsEntity credentials =
(ApplicationCredentialsEntity)
getCredentials(loginRequestDto.getEmail());
        return getAuthToken(credentials,
credentials.getUser().getAccountType(), null);
    }

    public TokenDto signInGoogle(HttpServletRequest
httpServletRequest) {
        OAuth2AuthenticationToken OAuth2token =
getOAuth2Token(httpServletRequest);
        GoogleCredentialsEntity credentials =
findGoogleCredentialsOrCreate(OAuth2token);
        return getAuthToken(credentials,
credentials.getUser().getAccountType(), OAuth2token);
    }

    public TokenDto signInFacebook(HttpServletRequest
httpServletRequest) {
        OAuth2AuthenticationToken OAuth2token =
getOAuth2Token(httpServletRequest);
        FacebookCredentialsEntity credentials =
findFacebookCredentialsOrCreate(OAuth2token);
        return getAuthToken(credentials,
credentials.getUser().getAccountType(), OAuth2token);
    }

    public TokenDto signInGithub(HttpServletRequest
httpServletRequest) {
```

```

        OAuth2AuthenticationToken OAuth2token =
getOAuth2Token(httpServletRequest);
        GithubCredentialsEntity credentials =
findGithubCredentialsOrCreate(OAuth2token);
        return getAuthToken(credentials,
credentials.getUser().getAccountType(), OAuth2token);
    }

    public TokenDto refreshToken(String refresh,
HttpServletRequest request) {
        String emailFromJwt =
jwtProvider.getEmail(jwtProvider.resolveToken(request));
        Jws<Claims> refreshToken =
jwtProvider.getRefreshToken(refresh);
        List<String> emailWithTypeAndClientIdFromJwt =
Arrays.asList(emailFromJwt.split("\\|"));
        List<String> emailWithTypeAndClientIdFromRefresh =
Arrays.asList(refreshToken.getBody().getSubject().split("\\|"));

        if(AccountType.fromString(emailWithTypeAndClientIdFromJwt.get(1)
) != AccountType.APPLICATION &&
!emailWithTypeAndClientIdFromJwt.get(2).equals(emailWithTypeAndC
lientIdFromRefresh.get(2))) throw new
JwtAuthenticationException("Invalid token");
        else if
(!emailWithTypeAndClientIdFromJwt.get(0).equals(emailWithTypeAnd
ClientIdFromRefresh.get(0))) throw new
JwtAuthenticationException("Invalid token");

        BaseCredentialsEntity credentials =
getCredentials(refreshToken.getBody().getSubject());
        String                                updatedToken                =
jwtProvider.createToken(credentials.getEmail(),
credentials.getRole().name());
        String                                updatedRefresh                =
jwtProvider.createRefreshToken(credentials.getEmail(),
credentials.getRole().name());

        return new TokenDto(credentials.getEmail(), updatedToken,
List.of(credentials.getRole()),                                updatedRefresh,
credentials.getUser().getAccountType());
    }

    private BaseCredentialsEntity getCredentials(String email){
        List<String>                                emailWithTypeAndClientId                =
Arrays.asList(email.split("\\|"));
        if(emailWithTypeAndClientId.size() > 1){
            switch
(AccountType.valueOf(emailWithTypeAndClientId.get(1))) {
                case APPLICATION:
                    return
getApplicationCredentials(emailWithTypeAndClientId.get(0));
                case GOOGLE:

```

```

        return
getGoogleCredentials(emailWithTypeAndClientId.get(2));
        case FACEBOOK:
            return
getFacebookCredentials(emailWithTypeAndClientId.get(2));
        default:
            return null;
    }
    } else return getApplicationCredentials(email);
}
private ApplicationCredentialsEntity
getApplicationCredentials(String email){
    return
applicationCredentialsRepository.findByEmail(email)
        .orElseThrow(() -> new
ItemNotFoundException("Credentials not found"));
}
private GoogleCredentialsEntity
findGoogleCredentialsOrCreate(OAuth2AuthenticationToken
oAuth2AuthenticationToken){
    Optional<GoogleCredentialsEntity> credentials =
googleCredentialsRepository.findByClientId(oAuth2AuthenticationT
oken.getName());
    if(credentials.isEmpty())
userService.createUserWithOAuth2(oAuth2AuthenticationToken);
    else return credentials.get();
    return
googleCredentialsRepository.findByClientId(oAuth2AuthenticationT
oken.getName())
        .orElseThrow(() -> new
ItemNotFoundException("Credentials not found"));
}
private OAuth2AuthenticationToken
getOAuth2Token(HttpServletRequest httpServletRequest){
    OAuth2AuthenticationToken OAuth2token =
(OAuth2AuthenticationToken)
httpServletRequest.getAttribute("oauthToken");
    if(OAuth2token == null) throw new
IllegalArgumentException();
    return OAuth2token;
}
private TokenDto getAuthToken(BaseCredentialsEntity
credentials, AccountType accountType, OAuth2AuthenticationToken
OAuth2token){
    String token = jwtProvider
        .createToken(String.format("%s|%s|%s",
credentials.getEmail(), accountType, OAuth2token != null ?
OAuth2token.getName() : null),
        credentials.getRole().name());
    String refreshToken = jwtProvider
        .createRefreshToken(String.format("%s|%s|%s",
credentials.getEmail(), accountType, OAuth2token != null ?
OAuth2token.getName() : null),
        credentials.getRole().name());

```

```
        return new TokenDto(credentials.getEmail(), token,  
List.of(credentials.getRole()), refreshToken,  
credentials.getUser().getAccountType());  
    }  
}
```