Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Веселова Василя Олеговича*
(ПІБ)

академічної групи *122-20-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка backend застосунку для інтернет магазину*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*доц. Ширін А.Л.*			
розділів:				
спеціальний	*доц. Ширін А.Л.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро

2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-3* *Веселова Василя Олеговича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка backend застосунку для інтернет магазину*

затверджена наказом ректора НТУ «ДП» від №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2024 р.***

Завдання видав _____ *доц. Ширін А.Л.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Веселов В.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *24.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 75 с., 30 рис., 2 дод., 20 джерел.

Об'єкт розробки: процеси і технології, що стосуються створення серверної частини вебзастосунку та враховують специфіку онлайн шопінгу.

Предмет дослідження: серверна частина вебзастосунку для інтернет магазину, яка включає в себе аспекти її розробки та функціональності.

Мета кваліфікаційної роботи: створення та удосконалення серверної частини вебзастосунку для інтернет-магазину, з основним акцентом на поліпшенні функціоналу, продуктивності та забезпеченні зручного користувацького досвіду.

Методи дослідження: для досягнення поставлених цілей використовуються різноманітні підходи, включаючи аналіз літератури, проектування системної архітектури, програмування, тестування та експериментальне підтвердження отриманих результатів.

Наукова новизна полягає в впровадженні передових технологій, таких як асинхронне програмування, масштабованість та ефективна обробка даних, робить це дослідження винятковим та передовим у вирішенні завдань, пов'язаних із створенням та управлінням товарами та користувацьким кошиком.

Практичне значення роботи: створення результативної серверної частини вебзастосунку для онлайн курсів має практичне значення як для окремих користувачів, так і для великих компаній. Забезпечення стабільної функціональності системи, оптимізованої для великої кількості користувачів, покращує якість і зручність процесу онлайн шопінгу.

Ключові слова: REST API, ВЕБЗАСТОСУНОК, ФРЕЙМВОРК, СЕРВЕР, SPRING FRAMEWORK, ІНФОРМАЦІЙНА СИСТЕМА, БАЗА ДАНИХ, DOCKER, POSTGRESQL, SWAGGER.

ABSTRACT

Explanatory note: 75 pp., 30 pictures, 2 appendices, 20 sources.

Object of development: processes and technologies related to the creation of the server part of the web application, taking into account the specifics of online shopping.

The subject of research: the server part of a web application for an online store, which includes aspects of its development and functionality.

The purpose of the qualification work: Creation and improvement of the server part of the web application for the online store, with the main emphasis on improving the functionality, productivity and ensuring a convenient user experience.

Research methods. A variety of approaches are used to achieve the goals, including literature analysis, system architecture design, programming, testing, and experimental validation of the obtained results.

The scientific novelty lies in the implementation of advanced technologies such as asynchronous programming, scalability and efficient data processing, making this research exceptional and cutting-edge in solving tasks related to the creation and management of products and the custom basket.

Practical meaning of work. Creating an effective back-end web application for online courses is of practical importance for both individual users and large companies. Ensuring stable functionality of the system, optimized for a large number of users, improves the quality and convenience of the online shopping process.

Keywords: REST API, WEB APPLICATION, FRAMEWORK, SERVER, SPRING FRAMEWORK, INFORMATION SYSTEM, DATABASE, DOCKER, POSTGRESQL, SWAGGER.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRUD - Create, Read, Update, Delete – операції створення, зчитування, оновлення та видалення даних відповідно;

IoC - Inversion of Control - принцип проектування, згідно з яким написані на замовлення частини комп’ютерної програми отримують потік керування від фреймворку;

DI - Dependency Injection - техніка програмування, в якій об'єкт або функція отримує інші об'єкти або функції, які йому потрібні, на відміну від їх внутрішнього створення;

JDBC - Java DataBase Connectivity - це інтерфейс прикладного програмування (API) для мови програмування Java, який визначає, як клієнт може отримати доступ до бази даних;

REST - Representational State Transfer - це архітектурний стиль програмного забезпечення, створений для керівництва проектуванням і розробкою архітектури інтернет застосунків;

JDK - Java Development Kit - комплект розробника застосунків на мові Java, що включає компілятор Java (javac), стандартні бібліотеки класів Java, приклади, документацію, різноманітні утиліти;

JRE - Java Runtime Environment - мінімальна реалізація віртуальної машини, що необхідна для виконання Java-додатків, без компілятора й інших засобів розробки;

JWT - JSON web token - це стандартний токен доступу на основі JSON, стандартизований у RFC 7519. Як правило, використовується для передачі даних для автентифікації в клієнт-серверних програмах;

API - Application Programming Interface - це спосіб завдяки якому дві або більше комп’ютерні програми можуть спілкуватися між собою. Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

ЗМІСТ

ВСТУП

На сьогодні шоппінг є дуже великою частиною життя багатьох людей. Також, після епідемії і локдаунів все більшу популярність отримують інтернет магазини. На сьогодні інтернет магазинів дуже багато, але не кожен з них має зручний інтерфейс і швидкий час відгуку, а швидкість реакції застосунка відіграє значну роль в бажанні клієнта користуватись вашим застосунком. Якщо програма відповідає довго, то якою б красивою вона не була, клієнти навряд бажатимуть користуватись такою програмою.

Метою цієї роботи є розробка швидкої серверної частини для інтернет магазину, використовуючи Spring Boot та PostgreSQL. Поєднання цих технологій дає велику гнучкість в розробці і досить впевнену і стабільну швидкодію застосунка.

Цей проект присвячений архітектурі, розробці і реалізації вебзастосунку, який дозволить користувачам швидко і зручно продавати та купувати речі в інтернеті. Основна увага приділятиметься не тільки швидкості застосунку, а й безпеці даних користувачів.

Результати цієї роботи можуть зацікавити розробників веб-додатків, які прагнуть використовувати передові технології та інструменти для створення високоякісних засобів комунікації. Крім того, отримані знання та досвід можуть послужити цінним внеском у подальші дослідження в області веб-розробки.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Аналіз готових існуючих рішень. Для аналізу було обрано два популярні торгівельні сайти в Україні: ROZETKA та Comfy. ROZETKA відома своїм широким асортиментом товарів, включаючи електроніку, побутову техніку, одяг та багато іншого. Comfy спеціалізується на технічних та електронних пристроях, включаючи комп'ютери, смартфони, планшети та інші товари цього сегменту.

Обидва сайти мають значну аудиторію споживачів в Україні та надають зручні сервіси для покупців. ROZETKA визначається своєю репутацією як одного з найбільших та найпопулярніших онлайн-магазинів у країні. Comfy також має велику популярність, особливо серед тих, хто шукає високоякісну техніку та електроніку.

Обидва сайти забезпечують широкий вибір товарів, зручний пошук та фільтрацію, а також швидку та надійну доставку. Вони активно використовують різні маркетингові стратегії та акції, щоб привертати нових покупців та утримувати вже існуючих.

Обрані сайти є не лише місцем покупок, а й об'єктом дослідження для розуміння тенденцій українського ринку електроніки та товарів побутової техніки. Розглядання їх асортименту, цін, відгуків покупців та інших аспектів дозволить отримати повну картину про споживчий попит та конкуренцію у цьому сегменті ринку. Зображення головної сторінки цих сайтів зображено на рис. 1.1 – рис 1.2.

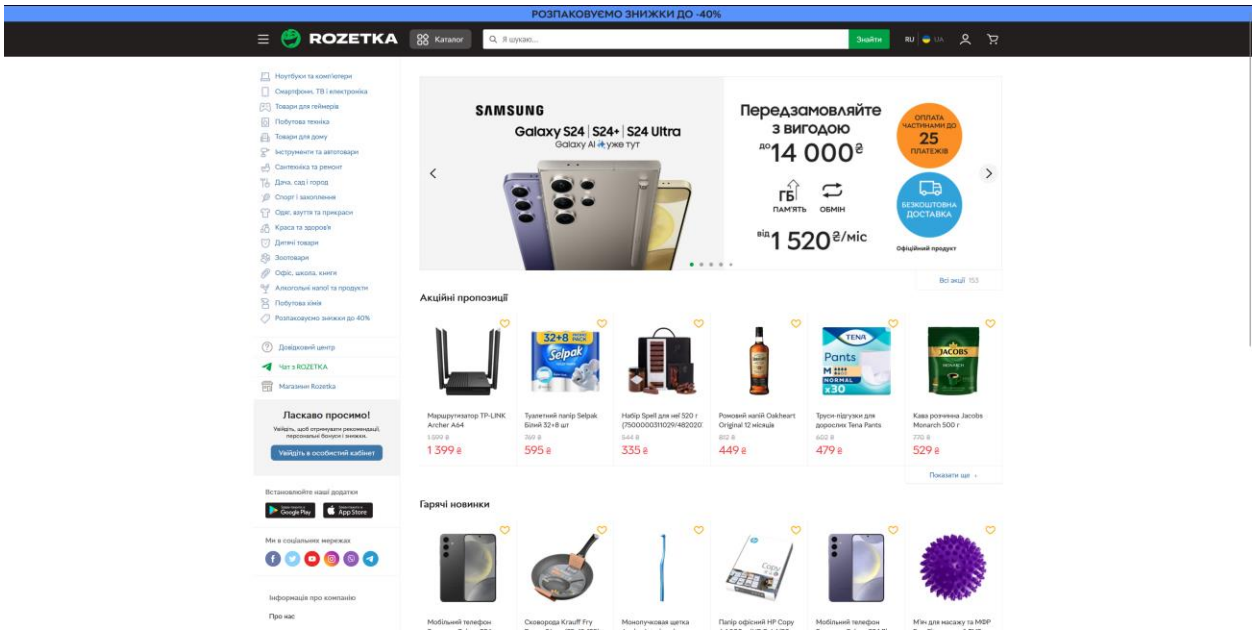

Рис.1.1. Головна сторінка магазину ROZETKA

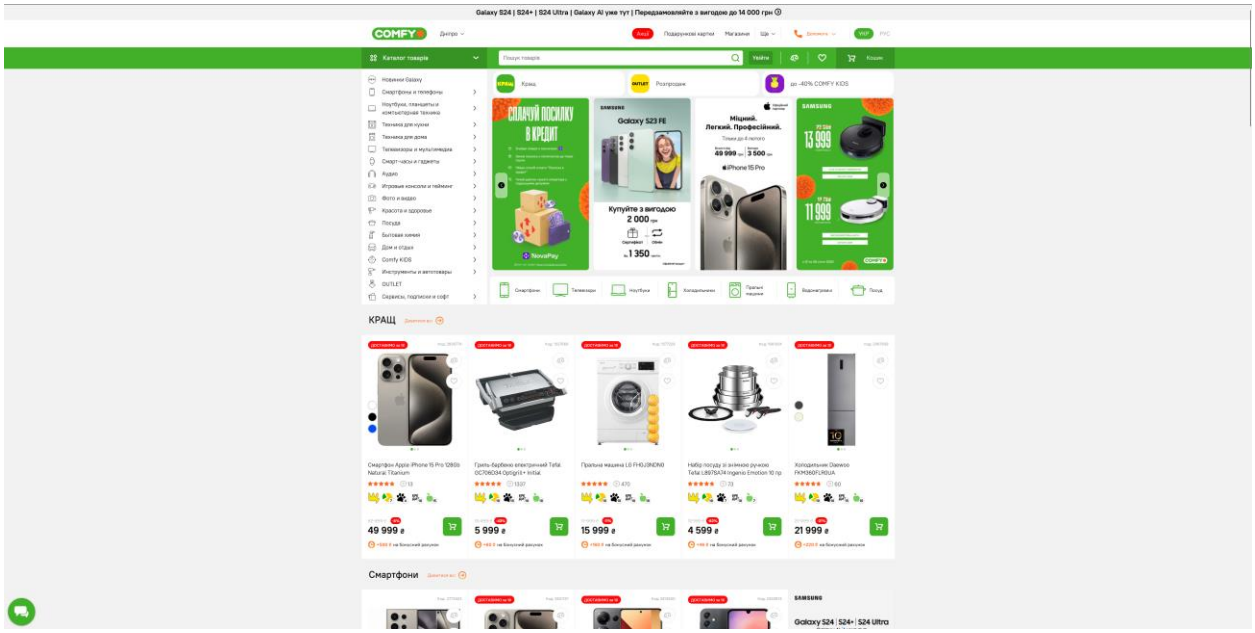

Рис.1.2. Головна сторінка магазину Comfy

Обидва сайти мають багатий функціонал та низьку затримку на переходах між сторінками. Завдяки цим якостям їм вдалося охопити дуже велику аудиторію і стати одними з найкращих і найпопулярніших інтернет магазинів України.

Кожен з сайтів на головній сторінці має такий функціонал:

- Зареєструватись/авторизуватись.
- Показати акційні/гарячі пропозиції.
- Зробити пошук по товарам.
- Продивитись товари за певним фільтром.
- Перейти до кошика.
- Якщо юзер зареєстрований – можливість перейти до особистого кабінету з особистою інформацією(ПІБ, місце проживання, адресу доставки і т.д), яку юзер може змінити.

1.2. Призначення розробки та галузь її застосування

Ця backend частина призначена для керування логікою і даними для інтернет магазину. До нього можна підключити один і навіть декілька різних фронтенд частин.

Створення цього програмного продукту є обумовленою потребою забезпечити користувачеві наступні переваги:

- Можливість продивлятись товари інших користувачів.
- Можливість викладати свої товари за своїми цінами.
- Підтвердження покупки через пошту.
- Відсутність реклами.
- Відсутнітсь будь-яких комісій.

1.3. Підстава для розробки

Відповідно до ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

– ОПП за спеціальністю 122 «Комп’ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровськаполітехніка» № 350-с від 16.05.2023 р;

– завдання на дипломний проект на тему «Розробка backend застосунку для інтернет магазину».

1.4. Постановка завдання

В даній кваліфікаційній роботі розглядатиметься створення backend застосунку для інтернет магазину.

Мета роботи: розробка backend частини інтернет магазину з використанням мови Java та фреймворка Spring.

Створення й розробка додатку повинна включати наступні етапи:

– аналіз вимог та обґрунтування необхідності: визначення функціональної та нефункціональної вимог до додатку, а також визначення його призначення та потенційної аудиторії;

– проектування: складання технічного завдання, створення концепції та дизайну інтерфейсу додатку, розробка архітектури та вибір технологій;

– розробка: програмування функціональності, розробка бази даних, тестування та відлагодження додатку;

– підтримка та розвиток: надання технічної підтримки, оновлення та покращення функціональності додатку згідно з вимогами та потребами користувачів.

Кожен з цих етапів має велике значення для успішної розробки додатку тайого подальшого ефективного функціонування.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставленої в роботі мети в інформаційній системі, що розробляється, повинні бути реалізовані:

– сучасні технології для розробки;
– можливість отримати онлайн послуги з закупівлі товарів;
– легкий пошук необхідних товарів;
– вхідні дані користувача валідуються;
– персональний кабінет користувачів;
– наявна система знижок;
– інструменти керування магазином адміністратором;

Користувач контактує з сайтом шляхом реєстрації персонального акаунту, перегляданння наявних товарів, замовлення товару. У відповідь отримую швидке задоволення свої вимог не виходячи з дому.

1.5.2. Вимоги до інформаційної безпеки

Під терміном "інформаційна безпека" розуміється стан, в якому системи обробки та зберігання даних забезпечують конфіденційність, доступність та цілісність інформації, а також використовуються та розвиваються в інтересах громадян, суспільства та держави. Це досягається шляхом впровадження комплексу заходів, спрямованих на захист інформації від несанкціонованого доступу, використання, розголошення, пошкодження, внесення змін, ознайомлення, перевірки запису або знищення. Реалізовані критерії безпеки системи:

– валідація вхідних даних;
– повідомлення про помилки;
– пароль користувача шифрується за допомогою кодувальнику

BCryptіз параметром «сили» - 10;

– користувачі поділяються на ролі, задля розподілення доступу;

– заради захищеного зберігання даних користувача та даних магазину,з серверу передається лише максимально необхідна інформація.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для коректного функціонування веборієнтованого додатку, повинні виконуватися певні вимоги до технічних засобів.

– Windows 7 (32/64 bit) або вище;

– оперативна пам'ять: мінімум 8 Гб;

– 10 GB вільного дискового простору;

– принаймні один інтернет браузер, наприклад Chrome, Firefox, Microsoft Edge, або застосунок для відправки запитів (Postman, або будь-який аналог);

– Java 17;

– мінімальна швидкість активного підключення до Інтернету 512 Кбіт/с і вище.

Для серверу бази даних:

– процесор: Intel Core або Xeon 3ГГц (або Dual Core 2ГГц) чи подібнийїм AMD процесор;

– оперативна пам'ять: 4 Гб.

Технічні характеристики, наведені вище, є рекомендованими. Це означає, що якщо в наявності є технічні засоби, які відповідають або перевищують зазначені характеристики, розроблений програмний продукт буде працювати з надійністю, швидкістю обробки даних і безпекою, що вимагається замовником.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для коректного функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде функціонувати веб орієнтована підсистема, відповідало наступним вимогам:

– операційна система Unix, Linux, Microsoft Windows 7/8/10/11;

– інтернет браузер (Microsoft Internet Explorer, Mozilla Firefox, Opera, Google Chrome).

Для створення системи на основі Spring конфігурація складає:

– Java 17;

– Spring Boot 3.2.2;

– підтримка збірки надається для Maven 3.2+ і Gradle 4.

Spring Boot підтримує такі сервлет-контейнери:

– Tomcat 8.5;

– Jetty 9.4;

– Undertow 1.4.

Вимоги до обладнання та програмного забезпечення для Java:

– оперативна пам'ять: мінімум 128 Мб;

– дисковий простір: 124 МБ для JRE, 2 МБ для оновлення Java;

– процесор: мінімум процесор Pentium 2 266 МГц.

Веборієнтований додаток має бути реалізовано на мові програмування Java для серверної частини, з використанням вебсерверу Apache Tomcat та СУБД PostgreSQL.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Розроблений веборієнтований додаток для інтернет магазину призначений для комерційної діяльності онлайн. Виходячи з цього функціональне призначення цієї інформаційної системи включає наступне:

– ведення бази даних магазину;

– відображення товарів в магазині інтернет магазину оптики;

– додавання товарів в кошик покупця та оформлення замовлення;

– додавання, перегляд або видалення обраних товарів з кошику;

– обробка замовлень та перегляд їх у персональному кабінеті користувача;

– авторизація та аутентифікація користувачів;

– підтримка різних методів оплати та доставки;

– система знижок;

– інструмент для зручного вибору окулярів онлайн за фото чи у режиміреального часу з відстежуванням обличчя;

Експлуатаційне призначення системи включає:

– зручний та простий інтерфейс для користувачів, що забезпечує швидкий та зручний пошук та замовлення товарів;

– використання Rest API, що збільшує адаптивність додатку під іншіплатформи;

– автоматизоване управління запасами та замовленнями, що зменшуєризик виникнення помилок та забезпечує ефективність управління;

– збільшення обсягів продажів через можливість розширення аудиторії, зокрема через залучення клієнтів з інших регіонів;

– можливість швидкої обробки замовлень, що забезпечує

задоволеність клієнтів та позитивний імідж бренду;

– наявність інструментів керування магазином у адміністраторській частині;

– зниження витрат на обслуговування клієнтів через автоматизаціюпроцесів замовлення товарів.

2.2. Опис застосованих математичних методів

Для розробки інформаційної системи у вебдодатку, що реалізує інтернет магазин, використовували стандартні математичні операції, бінарний пошук і підрахунок елементів.

2.3. Опис використаних технологій та мов програмування

Для реалізації веборієнтованого додатку було використано наступні мови програмування:

– Java;

– SQL.

Серед технологій було використано:

– фреймворк Spring (Spring Boot, Spring Data JPA, Spring Security);

– Docker, для розгортання бази;

– СКБД PostgreSQL.

Мова програмування Java

Java є потужною мовою програмування, яка має широке застосування в різних галузях. Вона використовується для розробки різноманітних додатків, включаючи настільні та мобільні додатки, обробку великих обсягів даних, вбудовані системи і багато іншого. За даними Oracle[64], компанії, що розробляє та підтримує Java, ця мова програмування працює на трьох мільярдах пристроїв у всьому світі, що свідчить про її велику популярність і

поширеність [3].

Завдячуючи своїй універсальності та безкоштовності, вона дозволяє створювати локалізоване та розподілене програмне забезпечення. Використання Java охоплює різноманітні сфери, включаючи:

– розробка ігор. Багато популярних мобільних, комп'ютерних та відеоігор побудовані з використанням Java. Вона використовується як для традиційних ігор, так і для ігор з використанням передових технологій, таких якмашинне навчання та віртуальна реальність;

– хмарні обчислення. Java часто називають "Write Once and Run Anywhere" (WORA)[6], оскільки вона може працювати на різних платформах. Це робить її ідеальним вибором для розробки децентралізованих хмарних додатків,які працюють на різних базових системах;

– великі дані. Java використовується для обробки великих обсягів даних в режимі реального часу. Вона надає потужні механізми для роботи зі складними наборами даних та виконання розрахунків над великими обсягами інформації;

– штучний інтелект. Java пропонує багато бібліотек для машинного навчання. Її стабільність та швидкодія роблять її ідеальним інструментом для розробки додатків штучного інтелекту, включаючи обробку природної мови та глибоке навчання.

Java є популярною мовою програмування, завдяки своїм зручним можливостям використання. Деякі з причин, чому розробники продовжують обирати Java більш, ніж інші мови програмування, включають.

– висока якість освітніх ресурсів. Java існує протягом тривалого часу, що забезпечує наявність багатьох різноманітних навчальних матеріалів для початківців. Доступна детальна документація, вичерпні книги та навчальні курси, які допомагають розробникам в процесі навчання. Початківці можуть почати з основ Java (Java Core) перед переходом до більш складних аспектів (Advanced Java);

– багатий набір вбудованих функцій та бібліотек. При використанні Java розробникам не потрібно писати кожну функцію з нуля. Java надає велику екосистему вбудованих функцій і бібліотек, які полегшують розробку різноманітних програм;

– активна підтримка спільноти. Java має велику активну спільноту користувачів, яка може надати підтримку розробникам при зустрічі з програмними проблемами. Платформа Java також підтримується і регулярно оновлюється;

– інструменти для розвитку якості. Java надає різноманітні інструменти для автоматизованого редагування, налагодження, тестування, розгортання та керування змінами. Ці інструменти роблять програмування на Java ефективним та економічним;

– багатоплатформність. Код, написаний на Java, може працювати на різних базових платформах, таких як Windows, Linux, iOS або Android, без необхідності в переписуванні. Це надає значну перевагу у сучасному світі, колинеобхідно запускати програми на різних пристроях;

– безпека. Java дозволяє завантажувати ненадійний код через мережу та запускати його в контрольованому безпечному середовищі, де він не може завдати шкоди хост-системі. Ненадійний код не може заражати хост-систему вірусами або читати/записувати файли на жорсткому диску. Рівні та обмеженнябезпеки в Java також можна легко налаштувати [4].

Декларативна мова програмування SQL

Структурована мова запитів (SQL) є стандартизованою мовою програмування, яка використовується для управління реляційними базами даних та виконання різноманітних операцій над даними в них. SQL була створена в 1970-х роках і зараз використовується не лише адміністраторами баз даних, але й розробниками, які пишуть сценарії інтеграції даних, та аналітиками даних, якіпроводять аналітичні запити.

SQL використовується для наступних цілей:

– зміна структури таблиць та індексів бази даних;

– додавання, оновлення та видалення рядків даних;

– отримання підмножини інформації з систем керування реляційними базами даних (РСУБД), яку можна використовувати для обробки транзакцій, аналітичних програм та інших програм, які взаємодіють з реляційною базою даних.

Запити SQL та інші операції складаються з команд, які записуються у вигляді операторів та об'єднуються в програми, що дозволяють користувачам додавати, змінювати або отримувати дані з таблиць бази даних.

Таблиця є основною одиницею бази даних і складається з рядків і стовпців даних. Кожна таблиця містить записи, які зберігаються в рядках таблиці. Таблиці є найпоширенішим типом об'єктів або структур бази даних, що містять або посилаються на дані в реляційній базі даних. Інші типи об'єктів бази даних включають представлення (логічне представлення даних з однієї або кількох таблиць), індекси (прискорюють функції пошуку) та звіти (дані, отримані з однієї або кількох таблиць, як правило, вибрані підмножини даних на основі критеріїв пошуку).

Кожен стовпець у таблиці відповідає категорії даних, наприклад, ім'я або адреса клієнта, і кожен рядок містить значення даних для відповідного стовпця. Реляційні бази даних названі так через те, що вони складаються з таблиць, які пов'язані між собою. Наприклад, SQL-база даних, що використовується для обслуговування клієнтів, може містити таблицю з іменами та адресами клієнтів, а також інші таблиці з інформацією про конкретні покупки, коди продуктів та контакти клієнтів. Таблиця, яка використовується для відстеження контактів клієнтів, часто містить унікальний ідентифікатор клієнта, відомий як ключ або первинний ключ, для посилання на запис клієнта в окремій таблиці, що містить дані про клієнта, такі як ім'я та контактна інформація.

SQL стала фактично стандартною мовою програмування для реляційних

баз даних після їхньої появи наприкінці 1970-х і на початку 1980-х років [7].

Spring Boot.

Основи конструювання ІС використовуючи Spring Framework.

До виникнення технології Enterprise Java Beans (EJB) розробники Java для створення вебдодатків, як правило, користувалися JavaBeans[28]. JavaBeans були ефективні для розробки компонентів користувацького інтерфейсу (UI), але вони не надавали управління транзакціями та безпекою, що стало критичним для створення надійних та стабільних корпоративних систем. З появою технології EJB сподівалися вирішити цю проблему та розширити можливості веб та корпоративних компонентів Java. Але використання EJB залишалося складким через обов'язкові завдання, такі як створення інтерфейсів Home і Remote, а також реалізацію методів зворотного виклику життєвого циклу.

Spring Framework виник як відповідь на ці проблеми в розробці корпоративних додатків. Завдяки інноваційним методам, таким як Аспектно-Орієнтоване Програмування (AOP), Plain Old Java Object (POJO) та Ін'єкція Залежностей (DI)[1], Spring спрощує процес розробки, усуваючи складності, пов'язані з використанням Enterprise Java Beans (EJB). Основна ідея Spring полягає в забезпеченні ефективного управління бізнес-об'єктами, що значно полегшує створення вебдодатків порівняно з традиційними Java фреймворками та інтерфейсами прикладного програмування (API), такими як Java Server Pages, Java Database Connectivity та Java Servlet.

Spring Framework можна розглядати як набір підфреймворків, також відомих як шари, до яких входять Spring Web MVC, Spring AOP, Spring Object-Relational Mapping (Spring ORM) та Spring Web Flow[1], серед інших. Під час розробки вебдодатку можна використовувати будь-який з цих модулів окремо або комбінувати їх для отримання кращої функціональності додатку. Особливості, такі як Інверсія Управління (IoC), Аспектно-Орієнтоване Програмування (AOP) та управління транзакціями, роблять Spring унікальним та потужним інструментом для розробки програмного забезпечення.

При розробці програмного забезпечення для зміни управління станом ресурсів можна скористатися рядом функціональних можливостей фреймворку Spring:

- Контейнер IoC. Spring надає два пакети - org.springframework.beans та org.springframework.context, які забезпечують функціональність IoC. Цей контейнер використовує патерн DI або IoC для неявного надання посилання на об'єкт в класі під час виконання.
- Фреймворк доступу до даних. Дозволяє розробникам використовувати API персистентності, такі як JDBC та Hibernate[10], для зберігання персистентних даних в базі даних. Це спрощує взаємодію з базою даних, керування підключенням, обробку винятків та управління транзакціями.
- Рівень абстракції JDBC. Допомагає в ефективній та простій обробці помилок.
- Фреймворк Spring MVC. Дозволяє створювати додатки на основі архітектури MVC, де всі запити від користувача спочатку обробляються контролером, а потім направляються на візуальну обробку, таку як JSP-сторінки або сервлети.
- Spring Web Service. Генерує кінцеві контексти та визначення вебсервісів на основі класів Java.
- Spring Security. Надає різноманітні функції безпеки, такі як автентифікація та авторизація, для створення захищених програм Java Enterprise.

Архітектура Spring Framework

Spring Framework, який є великим та розширюваним інструментом для розробки програмного забезпечення на платформі Java, включає в себе різноманітні модулі, що спрощують та оптимізують весь процес розробки. Ці модулі допомагають розробникам уникнути потенційних помилок, забезпечуючи високий рівень надійності та стабільності програм.

Застосування Spring Framework допомагає вирішувати важливі завдання розробки програм, зменшуючи ризик помилок та підвищуючи швидкість розробки завдяки високій рівню абстракції та гнучкості, яку він надає розробникам [1].

Основні модулі Spring Framework показані на рисунку 2.3.1.

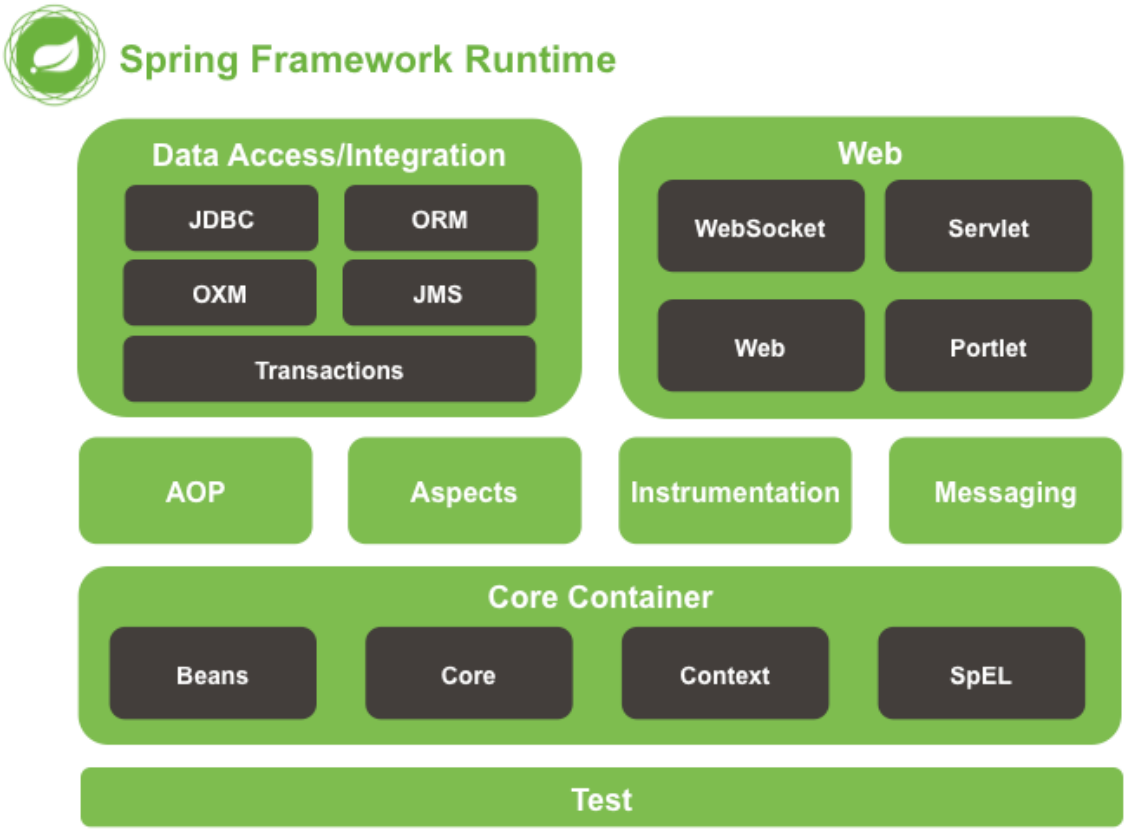

Рис.2.3.1. Модулі Spring Framework.

Це включає Spring Core, Spring AOP, Spring Web MVC, Spring DAO, Spring ORM, контекст Spring та Spring Web Flow [9, 19]. Кожен з цих модулів пропонує різноманітні інструменти для створення корпоративних додатків; наприклад, модуль Spring Web MVC розроблено для створення додатків з використанням паттерну Model–View–Controller.

Ключовим елементом у структурі фреймворку Spring є модуль Spring Core, що реалізує контейнер IoC [19]. У Spring існують два типи реалізацій контейнера: bean factory та application context. Перший визначається через

інтерфейс BeanFactory і служить як контейнер для бінів, дозволяючи відокремити конфігурацію та специфікацію залежностей від програмної логіки. Ця фабрика виступає в якості основного контейнера IoC та відповідає за створення екземплярів об'єктів, а також налаштовує та керує залежностями між цими об'єктами.

Подібно об'єктно-орієнтованому програмуванню, яке створює ієрархію об'єктів у додатках, AOP розкладає програми на аспекти або проблеми. Модуль Spring AOP дозволяє впроваджувати ці аспекти або проблеми в Spring-додаток. За допомогою анотації @Aspect [1, 3], звичайні класи або біни можуть стати аспектами у Spring AOP, що сприяє в моніторингу помилок у роботі програми, управлінні транзакціями і веденні журналів. Наприклад, важливим прикладом є управління транзакціями в банківських операціях, таких як переказ коштів з одного рахунку на інший.

Модуль Spring ORM [3, 4] використовується в програмі для взаємодії з інформацією в базі даних, пропонуючи можливості для роботи з Java Data Objects, Hibernate та iBatis [1, 5]. Spring ORM підтримує DAO, що дозволяє зручно створювати ORM-рішення на основі DAO. Це включає декларативне управління транзакціями, прозору обробку виключень, потокобезпечні легкі шаблонні класи та класи підтримки DAO.

Модуль вебархітектури (Web-MVC) фреймворку реалізує концепцію Model-View-Controller для розробки вебдодатків. Згідно з принципом роботи Spring MVC, при отриманні запиту від браузера він спочатку потрапляє до класу DispatcherServlet [1], який, використовуючи відображення-обробників, направляє запит до контролера. Контролер отримує та обробляє інформацію з запиту, повертаючи результат у вигляді об'єкту моделі до класу DispatcherServlet. Потім класи ViewResolver використовуються для направлення результатів до представлення.

Модуль Spring Web Flow є розширенням для Spring Web MVC. Там, де Spring Web MVC використовує контролери форм для реалізації

передвизначеного робочого процесу, такого як клас SimpleFormController, Spring Web Flow дозволяє створювати Java-клас або XML-файл для управління робочим процесом між різними сторінками вебдодатку.

Модуль Spring Web DAO [1, 5] включає підтримку структурного шаблону, який ізолює прикладний або бізнес-рівень від рівня персистентності (зазвичай, реляційна база даних) за допомогою абстрактного API. Цей модуль реалізує рівень абстракції JDBC та надає програмні та декларативні класи для управління транзакціями.

Модуль контексту в програмі Spring базується на основному модулі Core. ApplicationContext, який є контекстом додатку, виступає як інтерфейс BeanFactory. Таким чином, цей модуль використовує можливості пакету org.springframework.beans та підтримує різноманітні функціональні можливості, такі як інтернаціоналізація (I18N), валідація та завантаження ресурсів.

Структура діяльності автономного додатку

Spring Boot представляє собою спрощену та автоматизовану ітерацію Spring Framework[1, 6], яку можна легко розширити іншим функціоналом фреймворку за необхідності. Кожен проект, включаючи додаток для автоматизації зміни стану ресурсів тестування, розпочинає свій шлях розробки, будуючи базову структуру на цій компоненті фреймворку. Архітектура Spring Boot має багаторівневий підхід, де кожен рівень взаємодіє з іншими незалежно від їхнього розташування у ієрархії.

Основна мета Spring Boot полягає в усуненні непорозумінь та важких конфігурацій, що базуються на XML та окремих анотаціях у програмах. Одночасно він пропонує переваги, такі як гнучкість (можливості зміни конфігурації), автономність і можливість запуску додатка на ранніх етапах розробки.

Spring Boot складається з чотирьох компонентів:

1. Рівень Представлення У верхньому шарі архітектури Spring Boot розташований перший презентаційний рівень, що включає в себе

представлення (views), відповідальні за інтерфейсну частину програми. Цей рівень обробляє HTTP-запити та виконує аутентифікацію, відповідаючи за перетворення параметрів поля JSON в об'єкти Java та навпаки [5]. Після успішної аутентифікації запит передається на наступний, бізнес-рівень.

2. Бізнес-рівень Бізнес-рівень включає в себе всю бізнес-логіку та об'єднує класи сервісів, які відповідають за валідацію та авторизацію.

3. Рівень Збереження Рівень персистентності відповідає за збереження бази даних та виконує конвертацію між бізнес-об'єктами Java і рядками бази даних.

4. Рівень Бази Даних Останній рівень включає всі бази даних додатку, такі як MySql, MongoDB і інші. Зручність полягає в можливості оперувати декількома базами даних. Рівень бази даних відповідає за здійснення операцій CRUD.

Частіше за все, структура застосунків на Spring Boot виглядає як на рисунку 2.3.2.

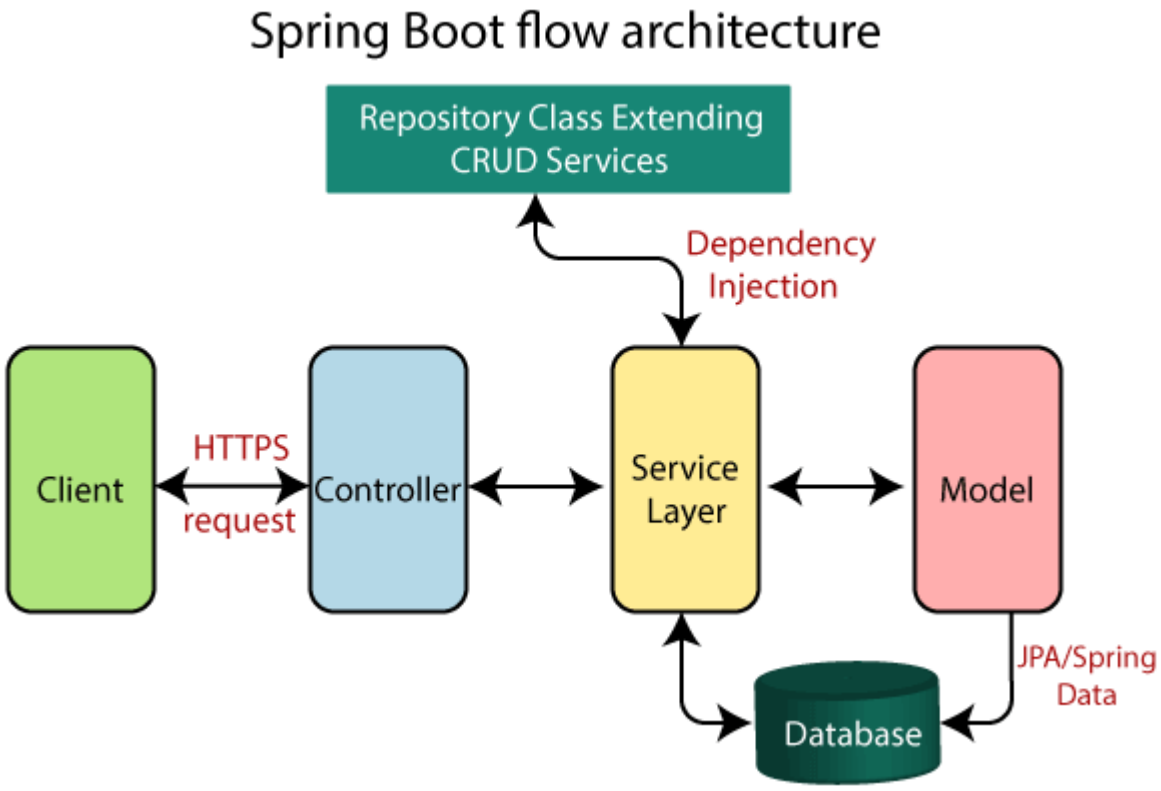

Рис.2.3.2. Флоу Spring Boot застосунка.

Розглянувши вказану структуру, наступний етап включає обробку HTTP-запиту контролером, де відбувається важливий взаємозв'язок з використанням різноманітних типів HTTP-запитів, таких як GET, PUT, POST і т.д. Контролер не лише відображає отримані дані, але і обробляє їхні дескриптори та запускає виконання логіки на сервері.

На рівні служб реалізується вся бізнес-логіка, включаючи класи сервісів, які відповідають за валідацію та авторизацію. Цей рівень визначається великим обсягом функцій, пов'язаних із бізнес-логікою, що гарантує правильне виконання операцій.

Далі рівень персистентності, відповідальний за зберігання даних у базі даних та конвертацію між бізнес-об'єктами Java і рядками бази даних. Використовуючи Java Persistence Library (JPA), дані з бази даних легко відображаються в класі моделі Spring Boot, що сприяє ефективній обробці та управлінню інформацією.

Останній рівень, що стосується бази даних, включає в себе різні типи баз даних, такі як MySql, MongoDB і інші. Ця гнучкість дозволяє оперувати декількома базами даних, а рівень бази даних взаємодіє з ними, забезпечуючи виконання операцій CRUD.

Ця архітектура взаємодіє із фреймворком Spring, використовуючи його ключові модулі, такі як Spring MVC та Spring Core. Такий підхід сприяє ефективній розробці, забезпечуючи гнучкість, автономність та зручність у керуванні конфігурацією.

Технології Spring для побудови додатка.

Платформа Spring Boot.

Для полегшення управління залежностями, Spring Boot автоматично включає необхідні сторонні бібліотеки для кожного типу програм, що базуються на Spring, та надає їх розробникам через starter-пакети (наприклад, spring-boot-starter-web, spring-boot-starter-data-jpa і інші)[43]. При створенні

додатка для автоматизації зміни стану тестових ресурсів, використовуються такі пакети:

- "spring-boot-starter-web" для розробки вебдодатків на базі Spring, який автоматично додає популярні бібліотеки, такі як spring-webmvc, jackson-json, validation-api та Tomcat;
- "spring-boot-starter-data-jpa" для надання доступу до бази даних без необхідності ручного пошуку та налаштування сумісних драйверів;
- "spring-boot-starter-security" для створення середовища, де здійснюється аутентифікація та авторизація користувачів.

Отже, при використанні Spring Boot файл конфігурації pom.xml[44], який містить інформацію про проект та деталі конфігурації, буде значно меншим, ніж у випадку традиційних Spring-додатків.

Однією з ключових можливостей Spring Boot є автоматична конфігурація програми, яку можна змінити за допомогою користувацьких налаштувань. Після вибору відповідного starter-пакету, Spring Boot автоматично старається налаштувати Spring-додаток, враховуючи додані залежності jar-файлів. Наприклад, для пакету spring-boot-starter-web, Spring Boot автоматично конфігурує біни, такі як DispatcherServlet та ResourceHandlers.

На початковому етапі розробки Spring Boot-додатку розглядаються різні моделі програм та відповідні DTO (об'єкти передачі даних), які розташовані у пакеті pojo. Використання DTO може бути обговорено з різних точок зору, і для вебдодатку, що орієнтований на бекенд сайту курсів, вирішено використовувати їх для полегшення взаємодії між модельним шаром програми та шаром інтерфейсу користувача. DTO дозволяють передавати лише ті дані, які необхідно відобразити в користувацькому інтерфейсі, уникнувши передачі всього об'єкта моделі, який може містити вкладені об'єкти та зберігається в базі даних.

Далі важливим елементом додатку є сервіси та об'єкти доступу до даних (DAO). Об'єкти доступу до даних розташовані у пакеті репозиторію services і є

розширеннями інтерфейсу JpaRepository [1, 5]. Методи цього інтерфейсу є базовими, як показано на рисунку 2.3.3, і допомагають сервісному рівню отримувати, зберігати та обробляти дані з бази даних. Сервісний рівень, розташований у пакеті сервісів services, завжди взаємодіє з моделями як вхідними або вихідними даними. Це визначено як одна з найкращих практик у розробці багаторівневої архітектури Spring-додатку для сайту онлайн курсів. Рівень контролера взаємодіє із сервісним рівнем, отримуючи запит від представлення (view), при цьому контролер не має доступу до об'єктів моделі та завжди взаємодіє за допомогою нейтральних DTO.

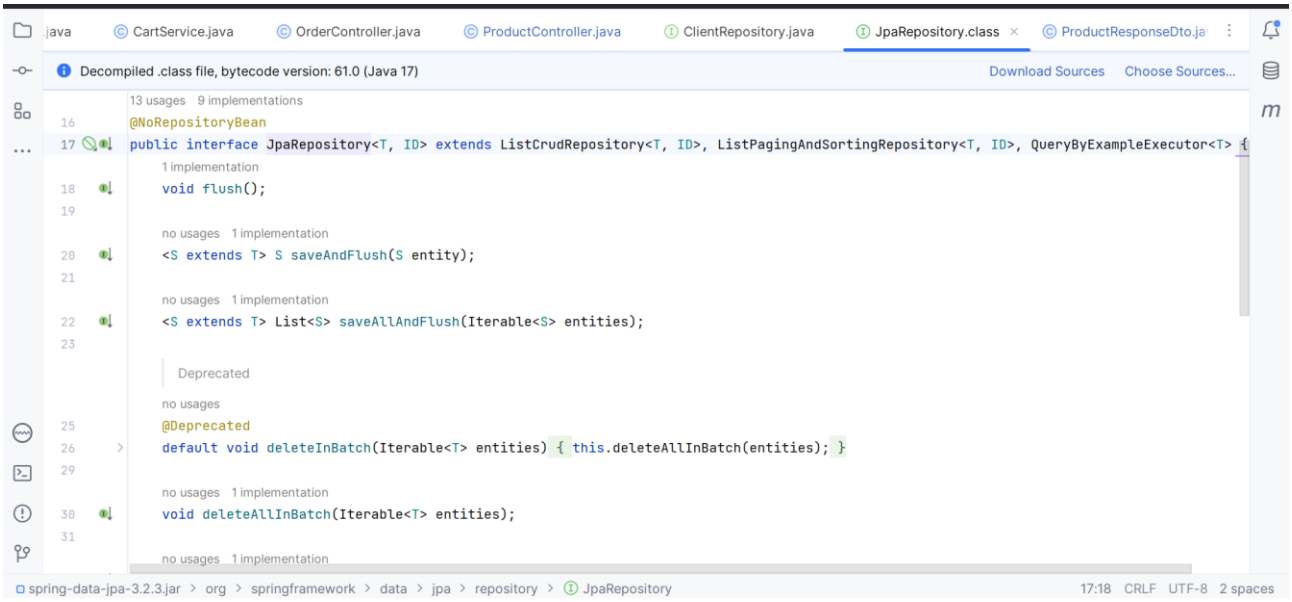

Рис.2.3.3. Вміст інтерфейсу JpaRepository.

Рівень контролера відповідає за координацію усього процесу, включаючи перехоплення запиту та завершення підготовки та відправлення відповіді. Цю функціональність реалізовано у пакеті controller.

Контролери, що використовуються у бекенд-додатку для сайту з онлайн курсами, використовують концепцію Spring WebMVC. Вони оброблятимуть вхідні вебзапити та взаємодіятимуть з об'єктами Spring ModelAndView, які містять дані для подальшого відображення на відповідних представленнях або формах.

Основний метод для запуску програми у Spring Boot додатку буде визначений у класі, обладнаному анотацією @SpringBootApplication[1,6]. Ця анотація охоплює автоматичну конфігурацію, сканування компонентів і налаштування завантаження Spring.

Вбудований сервер є ключовою рисою у Spring Boot. Отже, все, що потрібно для запуску програми, - це створити виконуваний jar-файл за допомогою Maven і викликати метод, зображений на рисунку 2.3.4.

Рис.2.3.4. Точка запуску(входу) в застосунок Spring Boot.

Захист додатку з Spring Security

У 2020 році впровадження глобальних локдаунів призвело до значного росту цифрової активності на світовому ринку. Збільшилася потреба в створенні вебсайтів та додатків для надання товарів та послуг клієнтам. Однак цей стрімкий розвиток цифрової трансформації також породив проблеми безпеки, оскільки багато організацій вперше стикнулися із необхідністю забезпечення безпеки в цьому новому контексті. Особливо важливою стала безпека корпоративних додатків, оскільки сучасні застосунки існують в різних мережах та взаємодіють з хмарними сервісами, що робить їх вразливими перед загрозами та можливими порушеннями безпеки.

Зростає необхідність забезпечення безпеки не тільки на рівні мережі, але й на рівні окремих додатків, оскільки атаки хакерів стають все поширенішими у напрямку цих додатків. Такий підхід природно надає значних переваг, і саме

цей принцип використовувався при розробці корпоративного додатку для автоматизації зміни стану ресурсів тестування.

Модуль безпеки у Spring є ефективним інструментарієм для реалізації функцій авторизації, аутентифікації та контролю доступу, надаючи при цьому розширений спектр доступних конфігурацій. Взагалі, це стандартний фреймворк, який використовується для захисту програмних додатків, побудованих на базі Spring.

У корпоративному додатку для бекенду сайту з онлайн курсами, Spring Security виконуватиме функції захисного шару, охоплюючи всі аспекти функціоналу програми, як це показано на діаграмі 2.3.5.

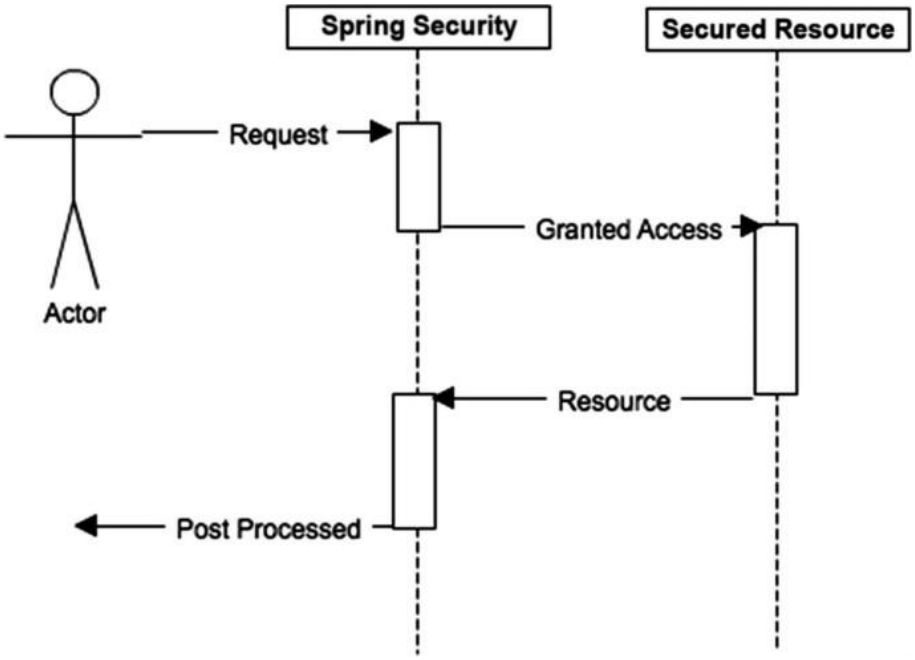

Рис. 2.3.5. Схема роботи Spring Security і його місце в загальному флоу.

Для впровадження Spring Security у програму, яка автоматизує зміну стану ресурсів тестування, необхідно виконати ряд етапів [4, 8].

На першому етапі, на початку проекту, додається стартер-пакет spring-boot-starter-security. Зазвичай додавання будь-якого стартера до POM-файлу не призводить до автоматичних змін у функціональності програми, і для їхньої реалізації зазвичай потрібно написати додатковий код. Проте у випадку Spring

Security ситуація відрізняється, оскільки він за замовчуванням надає функціонал для користувача. Його ім'я встановлено як "user", а пароль генерується автоматично при запуску програми.

Отже, при включенні лише стартер-пакету spring-boot-starter-security до POM-файлу відбувається:

- Spring Security автоматично генерує користувача з ім'ям "user" та створює пароль, який можна переглянути в консолі працюючого додатку;
- Створюється сторінка, яка містить форму для введення імені та пароля, реалізуючи функціонал аутентифікації на основі форми;
- Інформація для входу в програму автоматично перевіряється, включаючи ім'я та пароль;
- Всі URL залишаються недоступними до завершення успішного процесу входу під реєстрованим користувачем.

Далі проводиться налаштування InMemoryUserDetailsManager. Під час отримання параметрів користувача з запиту, що відбувається через Form-Based аутентифікацію, ім'я та пароль відправляються через форму та надходять на сервер як POST-параметри.

Для створення власного користувача в In-Memory автентифікації потрібно створити клас-конфігурацію SecurityConfig, який розширює клас WebSecurityConfigurerAdapter і позначити його анотацією @EnableWebSecurity для його біндування. AuthenticationManager відповідає за процес аутентифікації, і хоча він виконується автоматично, розробник може впливати на його конфігурацію, перевизначивши метод configure (AuthenticationManagerBuilder auth) класу WebSecurityConfigurerAdapter. Цей метод надає доступ до AuthenticationManagerBuilder, де можна визначити тип аутентифікації, тобто місце зберігання користувача. У випадку додатку з автоматизацією зміни стану ресурсів тестування, тип аутентифікації буде використовувати Jdbc.

Далі слід вказати конкретні налаштування для обраного AuthenticationManager, де визначається, як AuthenticationManager отримає

збереженого користувача та як потім порівняти його з введеними даними. Фактично, AuthenticationManager формує не лише ім'я та пароль, але й дозволи користувача.

Для впровадження повноцінної системи безпеки в додатку буде створено механізм отримання доступу до окремих частин функціоналу через систему ролей. Ключовим етапом є створення PasswordEncoder, який визначає, як саме буде шифруватися пароль. У стандартній реалізації, якщо Encoder не визначений, пароль залишається у своєму первісному вигляді. Проте в створюваному застосунку такий NoOpPasswordEncoder не призначений для використання, і замість цього пароль буде шифруватися з використанням BCryptPasswordEncoder з підвищеним рівнем захисту.

Один із способів авторизації у Spring Security використовує анотацію @PreAuthorize, яка дозволяє чітко визначити правила за допомогою виразів. Ці вирази визначають, чи дозволяється чи забороняється виконання конкретної операції. У додатку для автоматизації зміни стану ресурсів тестування кожен доступ до функціоналу контролюється анотацією @PreAuthorize, яка вимагає відповідних дозволів. Наприклад, для простого перегляду наявних ресурсів тестування використовується такий вираз: @PreAuthorize("hasAuthority('read')").

Останнім етапом є налаштування автентифікації та авторизації під час взаємодії з базою даних за допомогою DaoAuthenticationProvider[4,9]. Цей постачальник автентифікації, ймовірно, є одним із найбільш часто використовуваних у фреймворку. Схоже на більшість інших провайдерів автентифікації, DaoAuthenticationProvider використовує UserDetailsService для отримання імені користувача, пароля та повноважень. Однак, відмінно від інших провайдерів, які також використовують UserDetailsService, цей провайдер автентифікації очікує представлення пароля та перевіряє його валідність.

Аналіз безпечності та швидкості алгоритмів шифрування

Забезпечення безпеки даних у програмі невіддільно від використання

криптографії, що є ключовим аспектом цього процесу. Криптографія виступає основою для гарантування безпеки в сучасних інформаційних системах та застосовується у різних контекстах, таких як зберігання облікових даних, електронна пошта, мобільні платежі та цифрові транзакції, охоплюючи лише декілька сфер застосування.

Криптографія вважається складною та важливою галуззю в інформаційній безпеці. Спроби створити власні криптографічні бібліотеки можуть бути ризикованими, оскільки складно вручну перевірити новий алгоритм на всі можливі загрози та постійно відслідковувати оновлення ризиків безпеки даних. Багато сучасних мов програмування вже включають в себе реалізовані криптографічні бібліотеки та модулі. Таким чином, перед прийняттям рішення щодо найкращого методу захисту інформації, рекомендується ретельно проаналізувати та порівняти існуючі можливості.

Шифрування представляє собою процес перетворення даних у формат, який може бути розшифрований лише особами, які володіють відповідними ключами для відновлення початкового вигляду. Технології шифрування визнаються ключовим елементом будь-якого безпечного обчислювального середовища.

Безпека шифрування залежить від здатності алгоритму генерувати зашифрований текст, який важко або навіть неможливо відновити до початкового відкритого стану. Відповідно, для ефективного налаштування будь-якої основної схеми шифрування важливо правильно відповісти на наступні питання:

1) Оцінка безпекового рівня алгоритму відповідно до сучасних стандартів криптографії;

2) Розгляд характеристик продуктивності алгоритму, таких як підтримка паралельного шифрування та його ефективність;

3) Аналіз політичної придатності вирішення для використання конкретного алгоритму;

4) Використання правильно підібраних ключів і розмірів для забезпечення ефективності шифрування.

Важливою є ретельна настройка всіх параметрів, оскільки навіть маленький промах в безпеці конфігурації може зробити криптосистему вразливою до атак.

Під час дослідження, використовуючи бібліотеку javax.crypto, проводилося шифрування за допомогою найбільш розповсюджених алгоритмів JDK Sun: AES128, AES256, SHA256, SHA512, DES, Triple DES і Blowfish. Для кожної реалізації вимірювалася швидкість шифрування у мілісекундах (див. рис. 2.9), і були надані основні характеристики, які мають значення для забезпечення безпеки даних (див. рис. 2.3.6).

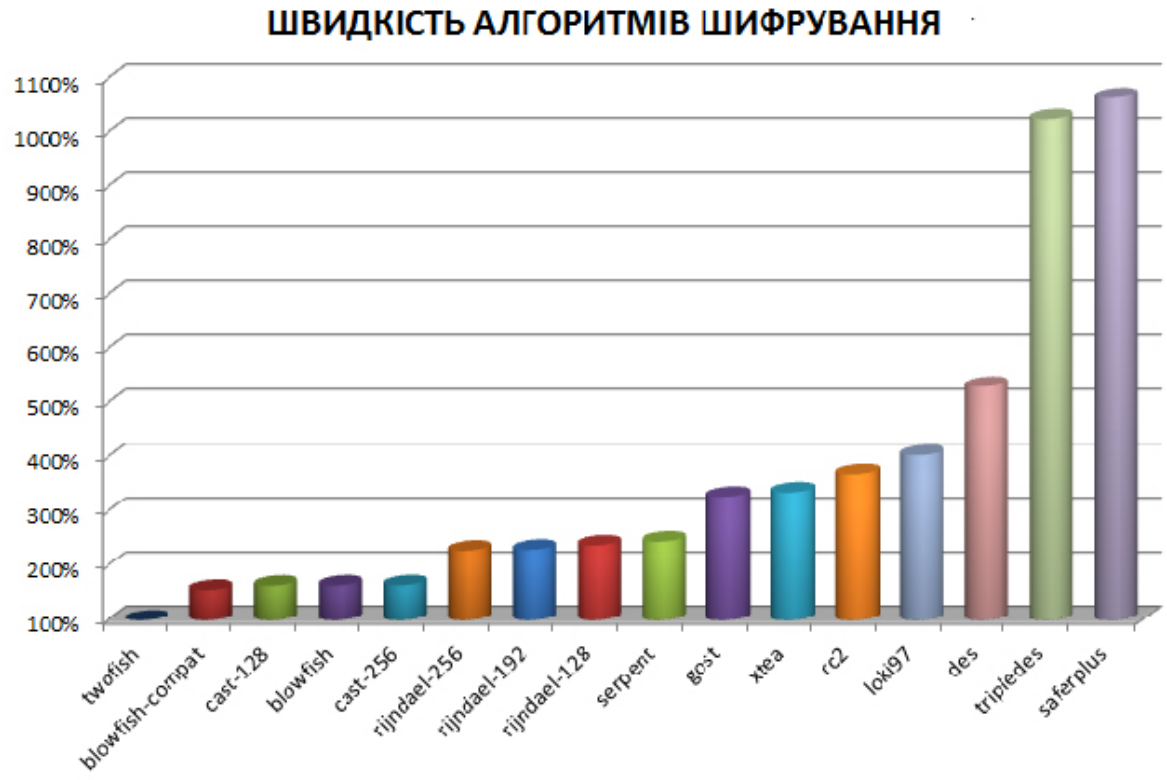

Рис. 2.3.6. Схема швидкостей алгоритмів шифрування.

Важливо зауважити, що AES є скороченням від "Advanced Encryption Standard" і є алгоритмом, який в кінцевому рахунку використовується більшістю людей, якщо у них немає важливих причин вибирати інший метод. Цей вибір є політично безпечним, оскільки стандарт шифрування,

рекомендований Національним інститутом стандартів і технологій США (NIST), передбачає використання AES з ключами 192 або 256 біт для зашифрування конфіденційної інформації.

Помімо вибору алгоритму, важливо визначити розмір ключа, що гарантує "міцність" шифрування, за умови, що сам алгоритм є безпечним в інших аспектах. Зазвичай розмір ключа взаємозв'язаний із кількістю припущень, які зловмиснику потрібно буде зробити для знаходження ключа методом "brute force", припускаючи, що всі можливі ключі мають однакову ймовірність.

Використання ключів сприяє підвищенню рівня безпеки методів захисту інформації. Алгоритми, які оперують з ключами, можна розділити на дві основні категорії:

- Симетричні алгоритми шифрування використовують один і той же ключ для обох операцій – шифрування та дешифрування. Ці алгоритми можуть функціонувати в потоковому (з бітами або байтами даних) або блочному режимі (з фіксованими блоками даних). Зазвичай їх використовують для шифрування даних, файлів та інформації у мережах зв’язку, таких як електронні листи, TLS, миттєві повідомлення і таке інше.
- Асиметричні алгоритми шифрування використовують два різних ключі – один для шифрування інший для дешифрування. Ці алгоритми використовуються для обчислення цифрових підписів і протоколів встановлення ключів.

Щодо визначення розміру ключа, зазвичай мінімально необхідний розмір визначається комбінацією максимального розміру ключа для конкретного алгоритму, екстрапольованого на кількість років, на які слід зберігати зашифровані конфіденційні дані. Наприклад, Національний інститут стандартів і технологій США (NIST) рекомендує мінімум "128 біт міцності" для забезпечення конфіденційності даних "після 2030 року". За найкращими практиками, запропонованими компанією Visa для передачі номерів кредитних карток, "ключі повинні мати потужність принаймні 112 еквівалентних бітів".

Вони фактично розглядають 128 біт (як у мінімальному розмірі ключа AES) як "достатньо міцні", можливо через обмежений термін служби кредитних карток.

PostgreSQL

PostgreSQL — це об'єктно-реляційна система керування базами даних (СКБД), яка виступає альтернативою як комерційним СКБД (таким як Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (наприклад, MySQL, Firebird, SQLite).

У порівнянні з іншими проєктами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL відрізняється тим, що він не контролюється жодною конкретною компанією. Його розробка стає можливою завдяки співпраці багатьох людей і компаній, які бажають використовувати цю СКБД та внести свій внесок у її розвиток.

Сервер PostgreSQL написаний на мові програмування C. Зазвичай він розповсюджується у вигляді набору текстових файлів із вихідним кодом. Щоб встановити його, необхідно скомпілювати файли на своєму комп'ютері і скопіювати їх у відповідний каталог. Весь процес інсталяції детально описаний у документації.

Структура та опис бази даних

Хоча у більшості систем управління базами даних існують стандартні функції для створення та отримання доступу до інформації у базі даних, методи виконання завдань можуть варіюватися.

Додаткові можливості, функції та підтримка є унікальними для кожної (СУБД). Для виконання цього завдання я обрав систему PostreSQL, як одну з найпопулярніших SQL СКБД опираючись на наступні фактори:

- В моєму проекті необхідно було зв’язати між собою таблиці за допомогою зовнішнього ключа. Стандартний інструмент для цього SQL база
- PostgreSQL легко підняти в Docker-контейнері[15], і для подальшого використання не потрібно нічого, окрім піднятого контейнера
- Через те, що PostgreSQL користується високою популярністю серед

розробників, для неї легко знайти гайди майже на всі випадки

Структура таблиць бази даних показана на рисунку 2.3.7.

```
                List of relations
 Schema |          Name           | Type  | Owner
--------+-------------------------+-------+-------
 public | auth_tokens             | table | psql
 public | cart                    | table | psql
 public | cart_products           | table | psql
 public | client                  | table | psql
 public | client_additional_info  | table | psql
 public | confirmation_codes      | table | psql
 public | flyway_schema_history   | table | psql
 public | orders                  | table | psql
 public | orders_products         | table | psql
 public | product                 | table | psql
(10 rows)
```

Рис.2.3.7. Список таблиць бази даних.

В самому застосунку взаємодія з базою даних відбувається через інструменти SpringData: Repository. SpringData надає out-of-box методи створення/оновлення/пошуку/видалення для об’єктів, до яких він підв’язаний. Для створення свого репозиторія потрібно створити інтерфейс та успадкувати SpringData інтерфейс JpaRepository<T, ID>. Базова реалізація цього інтерфейсу показана на рисунку 2.3.8.

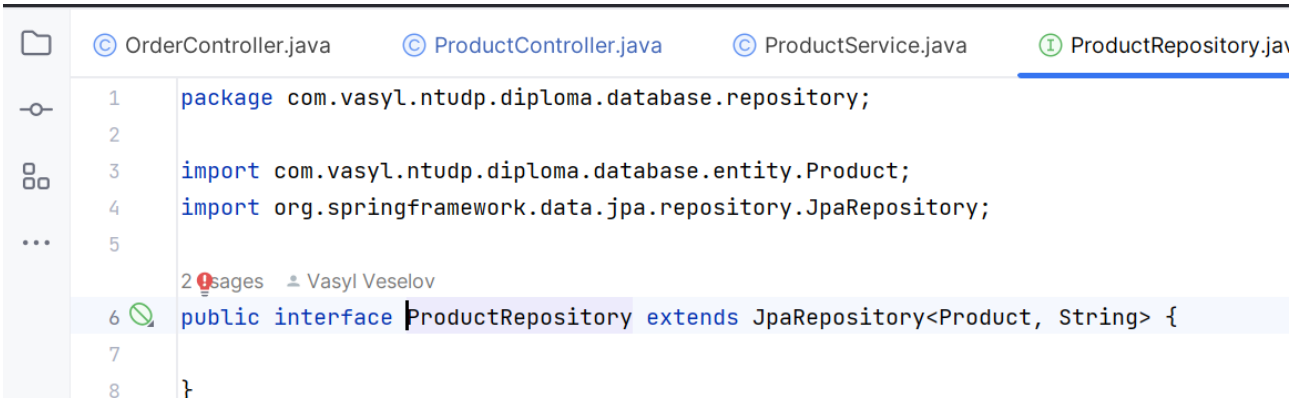

Рис.2.3.8. Базова реалізація SpringData репозиторія.

JWT Token Authorization

JSON Web Token (JWT)[12] - це стандарт для генерації токенів доступу на

основі JSON, що визначений в RFC 7519. Зазвичай його використовують для передачі даних для аутентифікації в програмах клієнт-сервер. Процес включає створення токену сервером, підпису його секретним ключем, та передачу клієнту. Клієнт використовує отриманий токен для підтвердження своєї ідентичності.

Для реалізації JWT використовується бібліотека io.jsonwebtoken.jjwt. Архітектура вебзастосунку буде структурована таким чином, що неавторизований користувач не матиме можливості здійснювати успішні виклики на будь-які з ендпоїнтів, за винятком службових (реєстрація, авторизація, хелсчек). Такий підхід дозволяє ефективно контролювати доступ до різних функцій системи та забезпечує безпеку обміну інформацією між клієнтом і сервером.

Внутрішня структура створюваної системи являє собою вебсайт, який складається з серверної частини (backend). Користувач системи взаємодіє безпосередньо через HTTP-протоколи, через функціонал якої має змогу:

- зареєструватись;
- увійти в систему;
- створити/оновити/видалити продукт
- продивитися список продуктів
- додати/прибрати продукт з кошика
- продивитися свій кошик
- оплатити продукти в кошику
- підтвердити 2ФА через електронну пошту
- продивитися історію замовлень

Усі запити, окрім «зареєструватись» і «увійти в систему» доступні тільки авторизованому користувачу. Авторизованим вважається користувач, в якого в хедері Authentication присутній валідний JWT Token.

Процес отримання токена виглядає наступним чином (рис. 2.3.9):

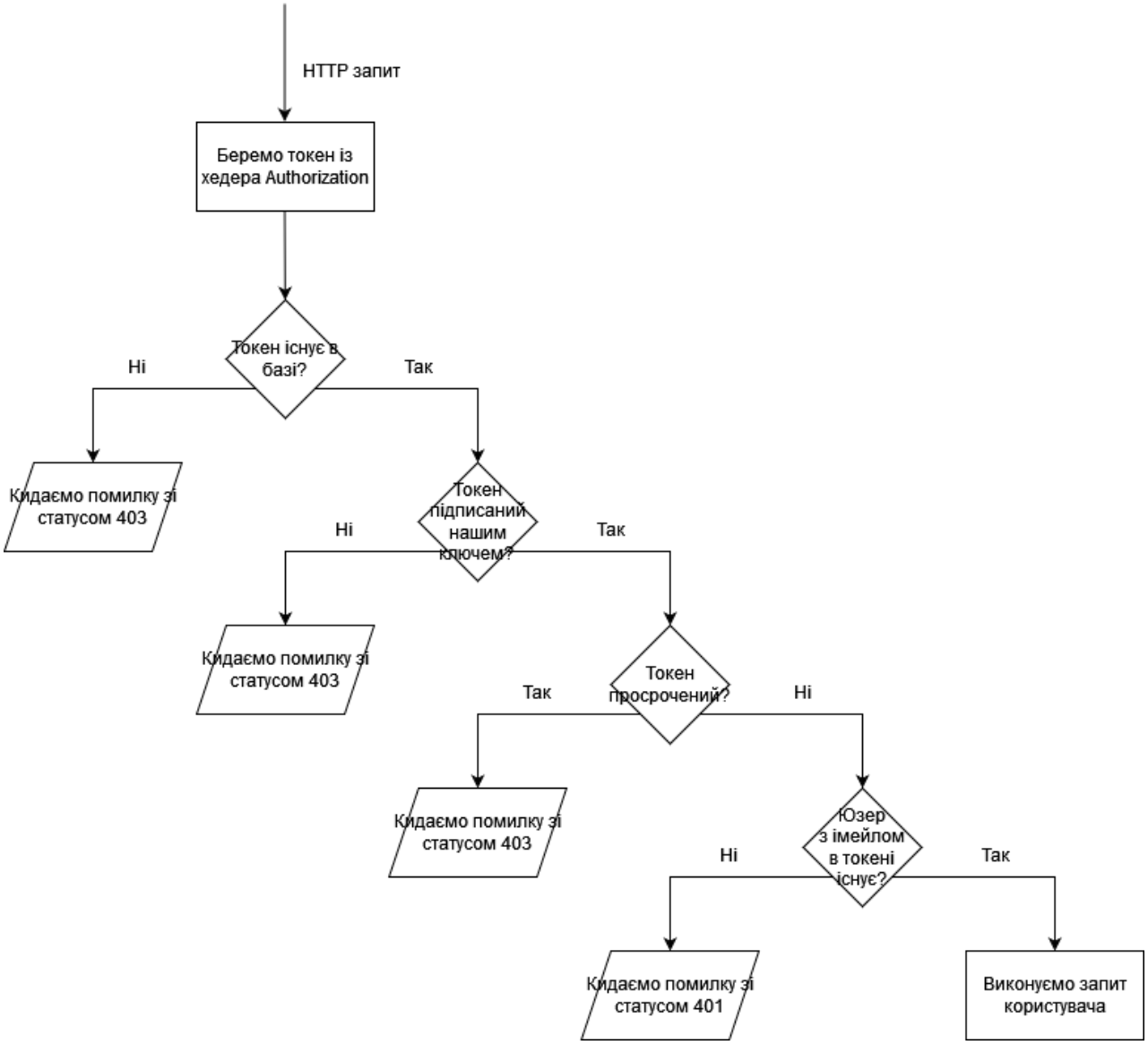

Рис.2.3.9. Процес перевірки токена на вірність.

Отримати токен може тільки зареєстрований користувач. Процес реєстрації показаний на рисунку 2.3.10.

Після реєстрації користувач матиме змогу зробити запит на логін і отримати свій токен авторизації. Процес отримання токена показаний на рисунку 2.3.11.

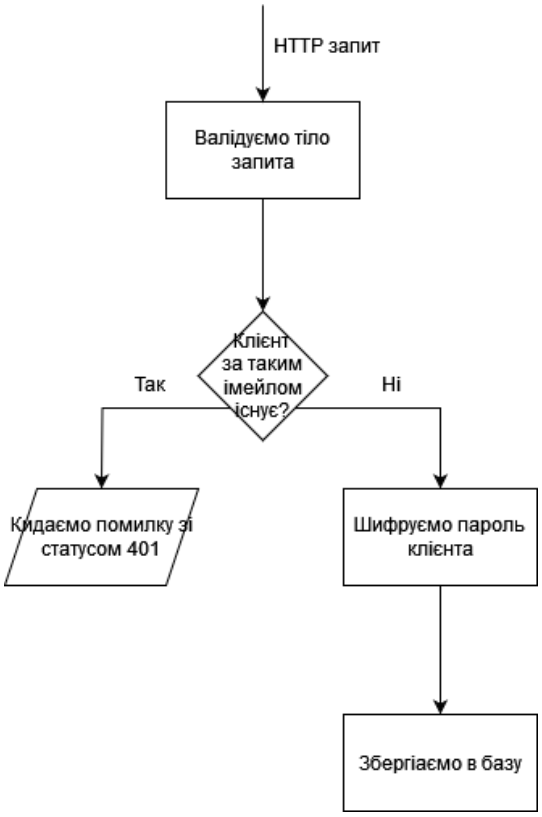

Рис. 2.3.10. Процес реєстрації користувача.

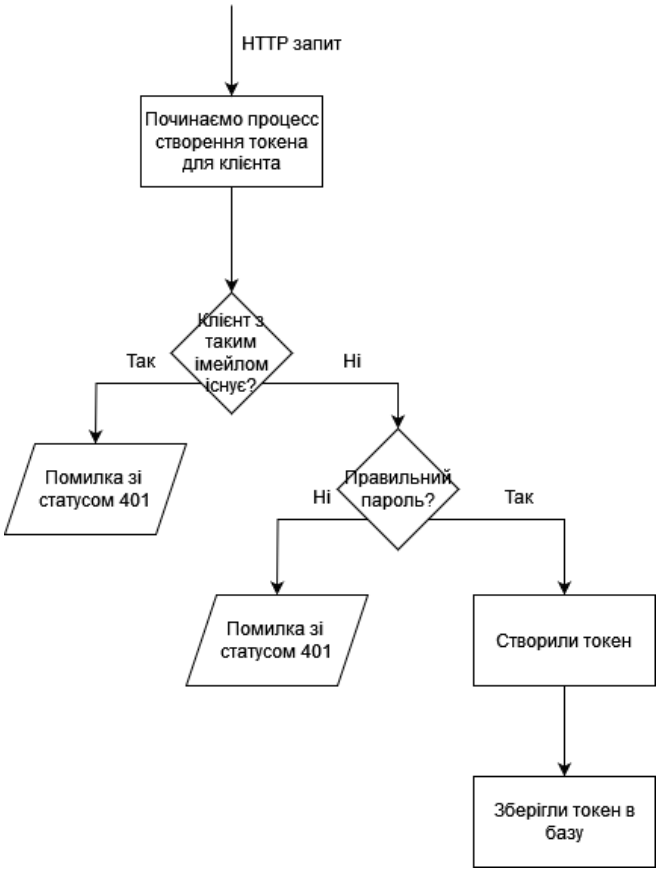

Рис. 2.3.11.Процес створення токена для користувача.

2.4. Опис структури системи та алгоритмів її функціонування

Обробка основних даних в системі відбувається на серверній стороні додатку. Вхідні дані для програмного комплексу надходять у вигляді текстових даних з форм, які користувач заповнює для керування курсами або заннятями. Результатом роботи вебдодатку є набір збережених у системі об’єктів, готових до використання та відображених на сторінках перегляду, згрупованих за їх типом.

Основна мета системи полягає в забезпеченні клієнту інтуїтивно зрозумілого та зручного інструменту для ефективного керування онлайн курсами, а також можливості створення сценаріїв для їхньої зміни у майбутньому.

Для розробки застосунку було застосовано наступні технології:

- Java 21 – створення серверної логіки;
- Spring Security – для аутентифікації та авторизації;
- Spring Data – API для роботи з базою даних;
- Spring Boot – створення бізнес логіки додатку;
- Project Lombok – скорочення шаблонного коду;

Розроблений застосунок для керування онлайн курсами має відповідати наступним вимогам:

- інтуїтивно-зрозумілий принцип для роботи з системою;
- наявна система автентифікації та авторизації;
- реєстрація користувачів;
- зручна навігація за допомогою API;
- можливість роботи в браузерах на ПК, ноутбуках та мобільних пристроях з різними характеристиками та конфігураціями;
- шифування особливо цінних даних користувачів;
- наявність модулів створення, оновлення та видалення курсів;
- гнучкість для модифікацій та масштабування.

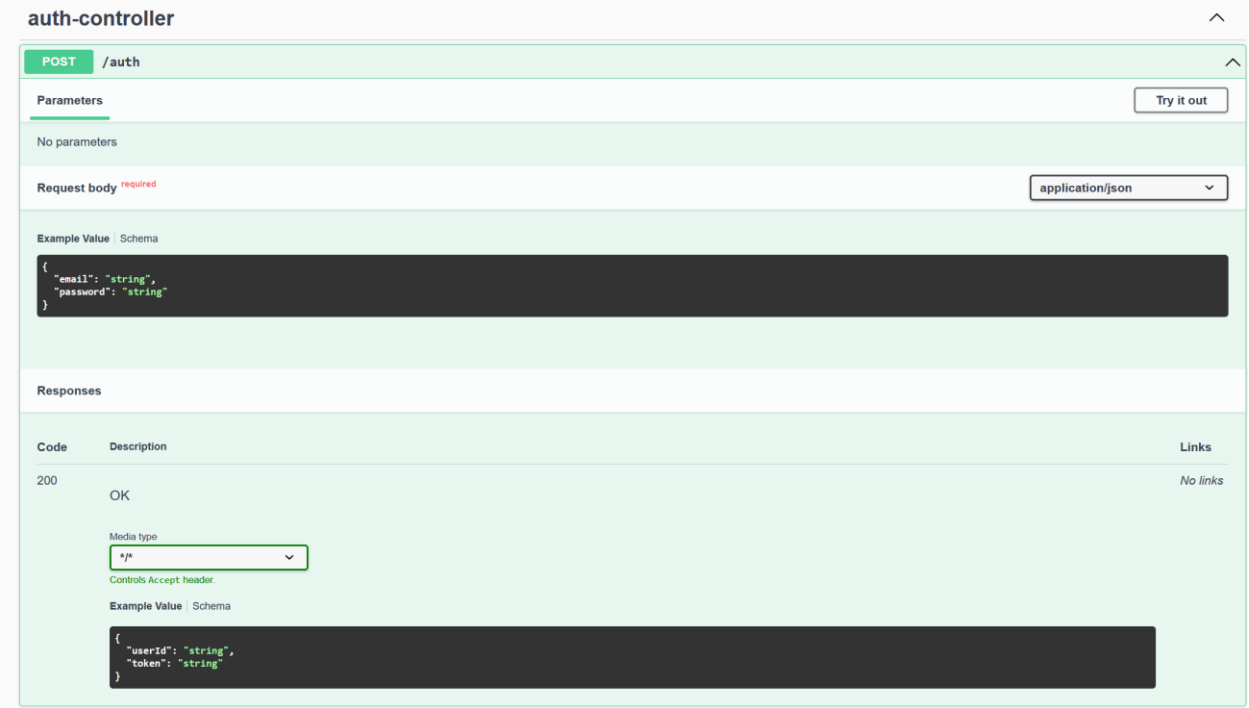

Рис. 2.4.1. Запит на авторизацію.

Зареєстровані користувачі можуть робити аутентифікацію, щоб отримати свій JWT Token. Щоб зареєструватись користувачу потрібно викликати API реєстрації(рис 2.4.2).

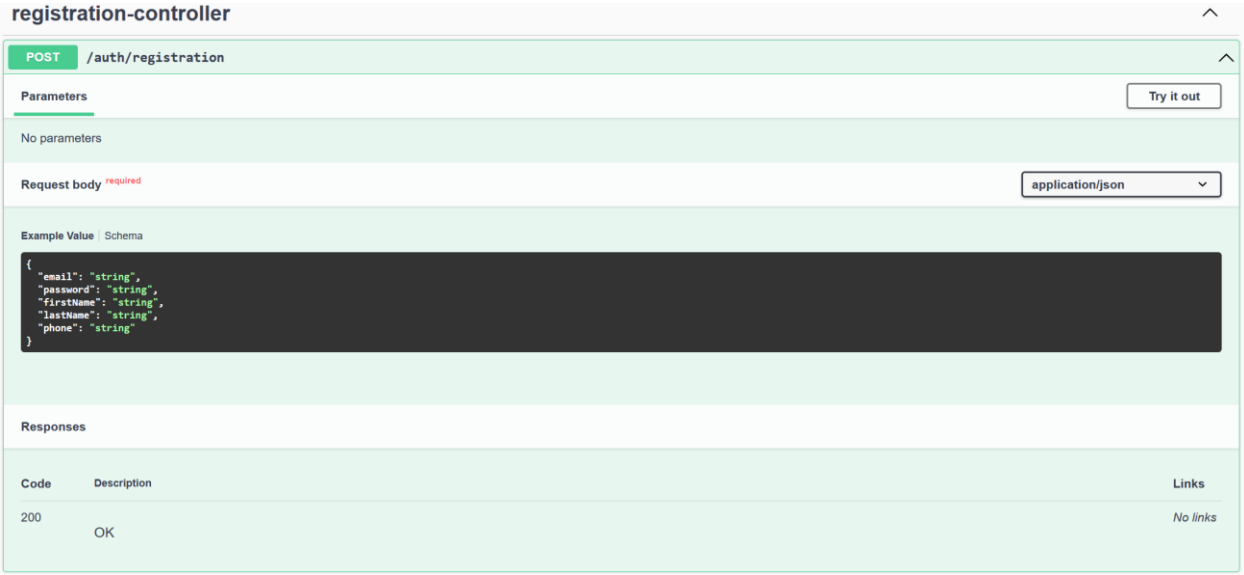

Рис. 2.4.2. Запит на реєстрацію.

Після реєстрації юзер може робити запит на аутентифікацію, отримати токен і з цим токеном викликати усі інші API застосунку.

Клієнт може отримати інформацію про себе, викликавши API деталей клієнта за імейлом(рис. 2.4.3).

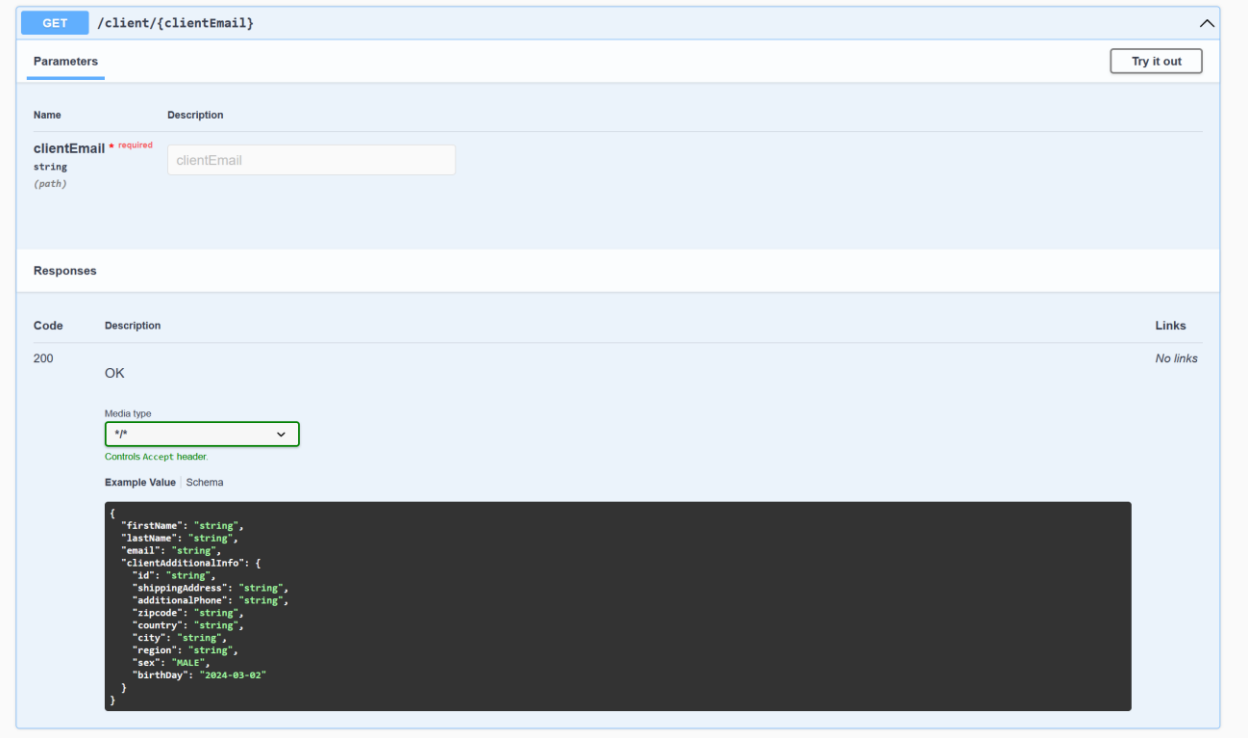

Рис. 2.4.3. API деталей клієнта за імейлом.

Також, якщо клієнт захоче змінити свою інформацію, він може викликати API зміни інформації(рис. 2.4.4).

Рис. 2.4.4. API зміни інформації клієнта.

Для керування продуктами юзер може використати API для продуктів. Наприклад, отримання продукта за його ідентифікатором(рис. 2.4.5).

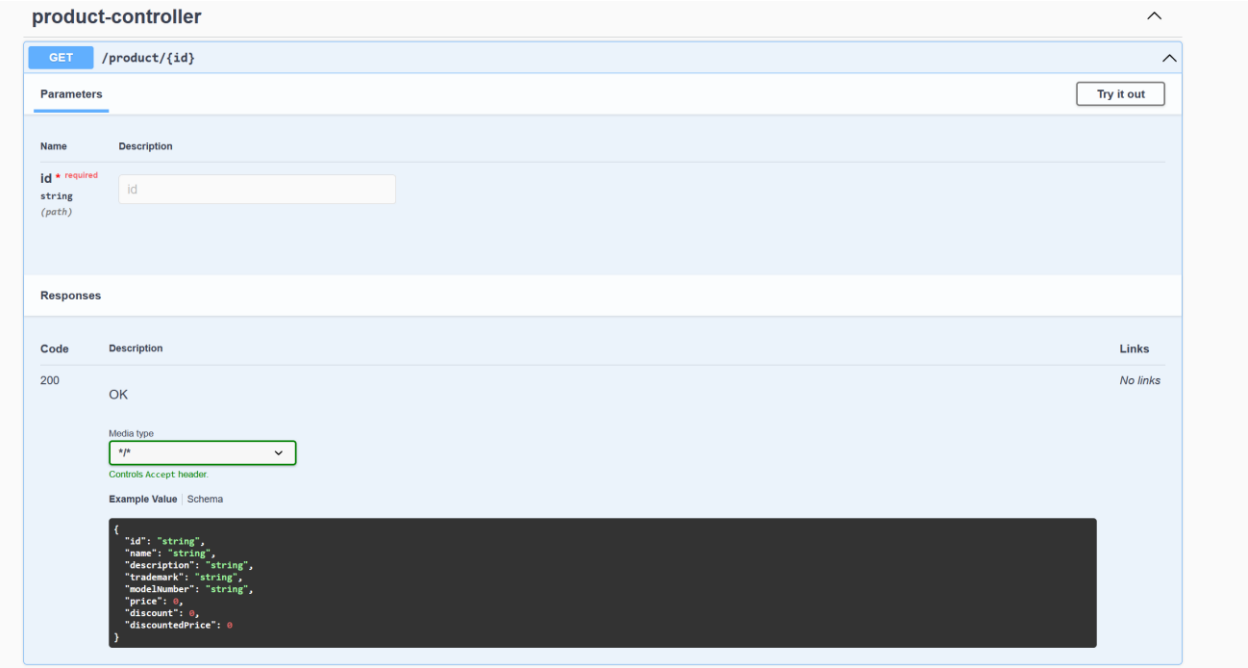

Рис. 2.4.5. API отримання продукту за ідентифікатором.

Якщо користувачу невідомий ідентифікатор продукта, він може отримати список усіх доступних продуктів, викликавши API на рис. 2.4.6.

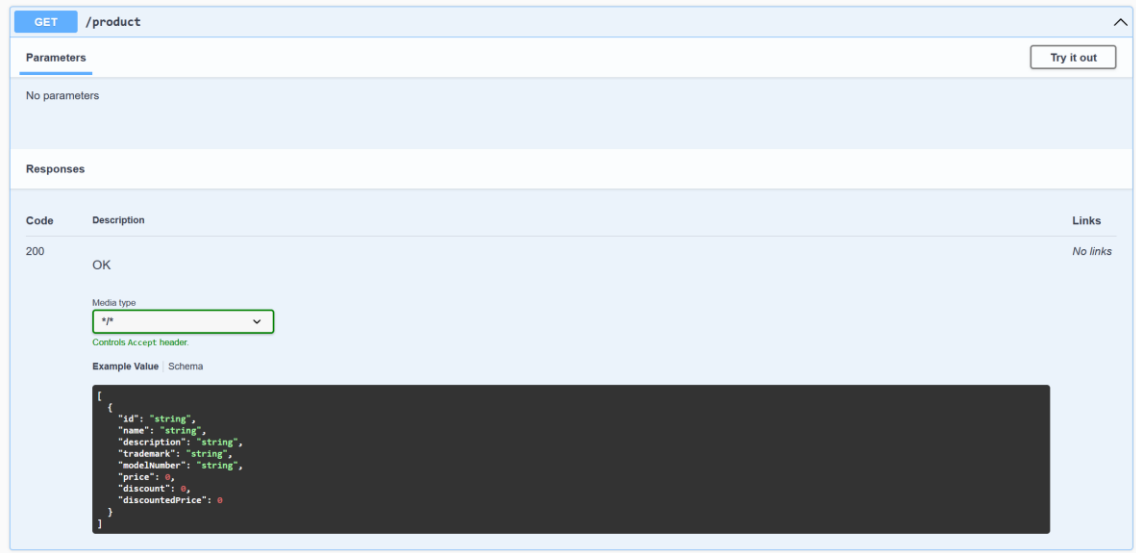

Рис. 2.4.6. API отримання списку усіх продуктів.

Щоб додати свій продукт, користувачу потрібно викликати API створення продукту(рис. 2.4.7) .

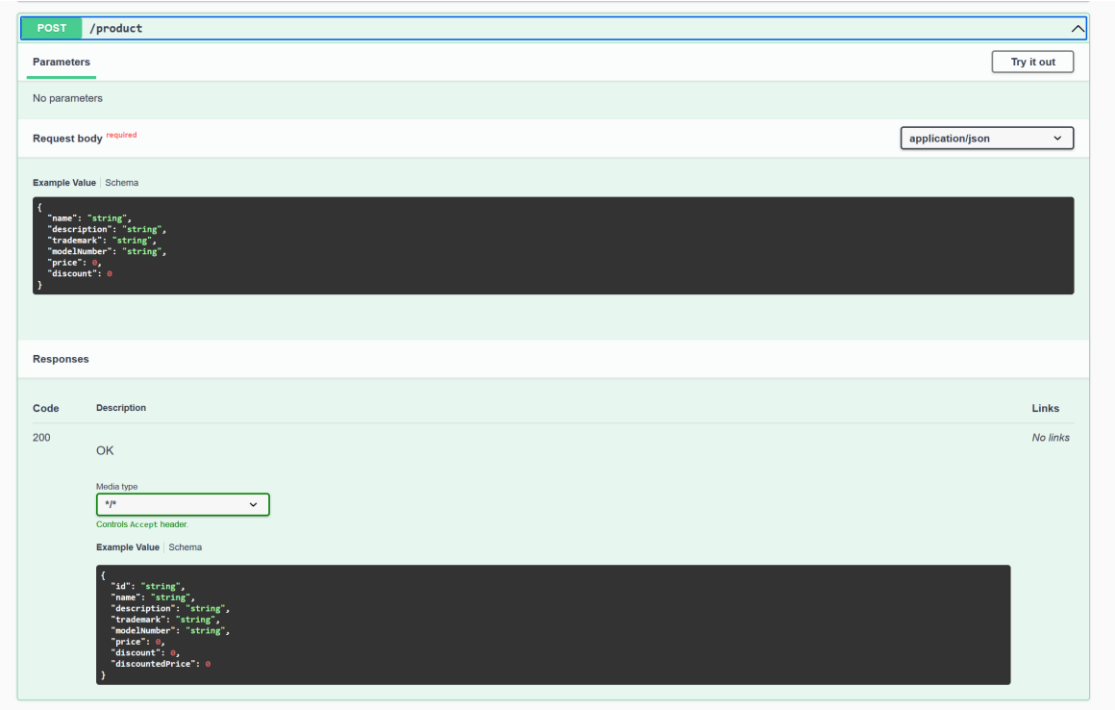

Рис. 2.4.7. API створення продукту.

Також, користувач може змінити, або видалити продукт, викликавши API на рисунку 2.4.8 та 2.4.9 відповідно.

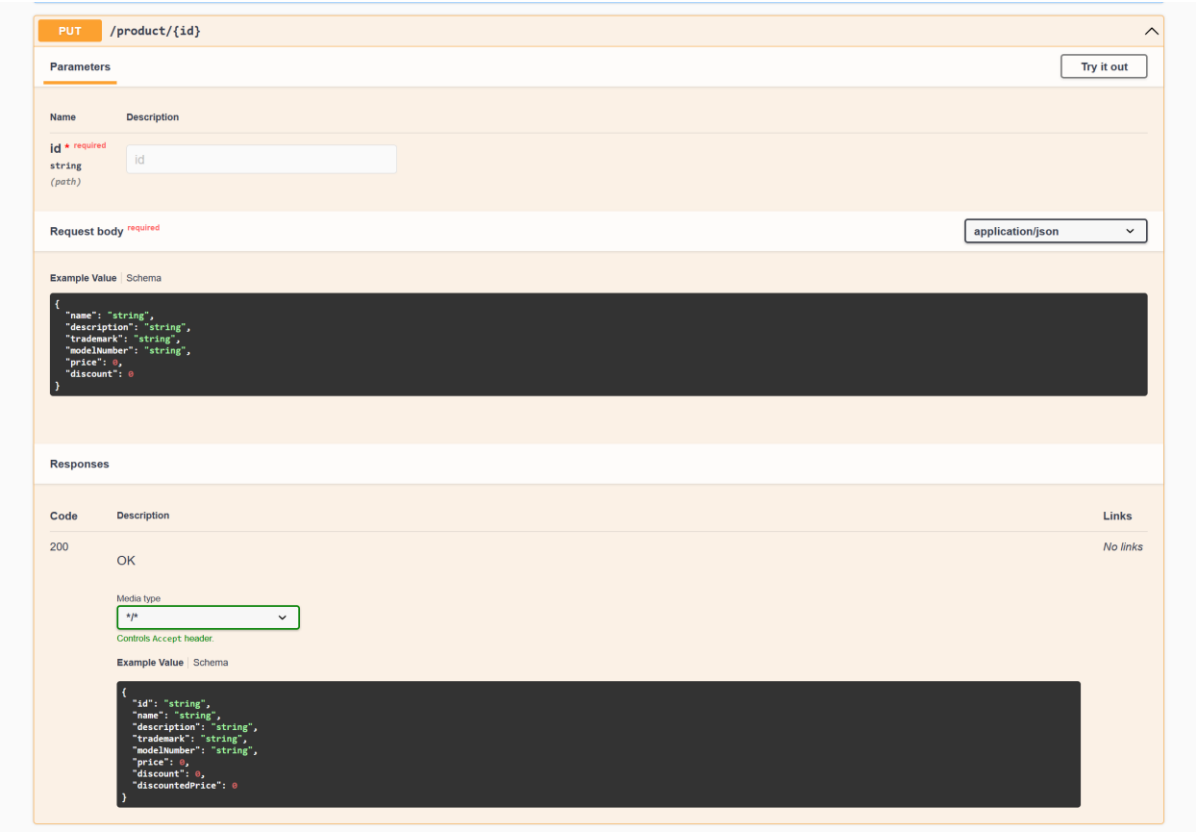

Рис. 2.4.8. API зміни продукта за ідентифікатором.

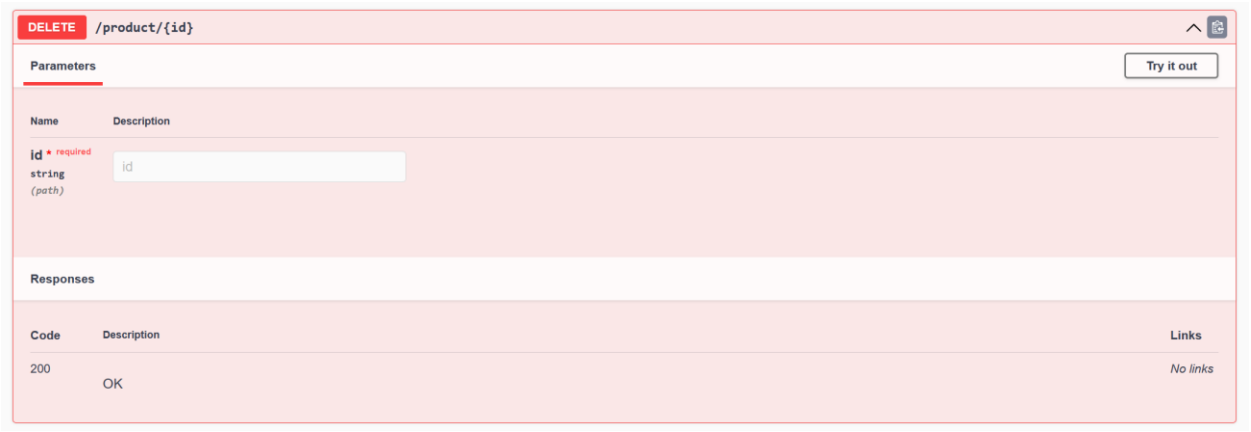

Рис. 2.4.9. API видалення продукта за ідентифікатором.

Маємо схожі API для взаємодії зі своїм кошиком (рис. 2.4.10-2.4.15).

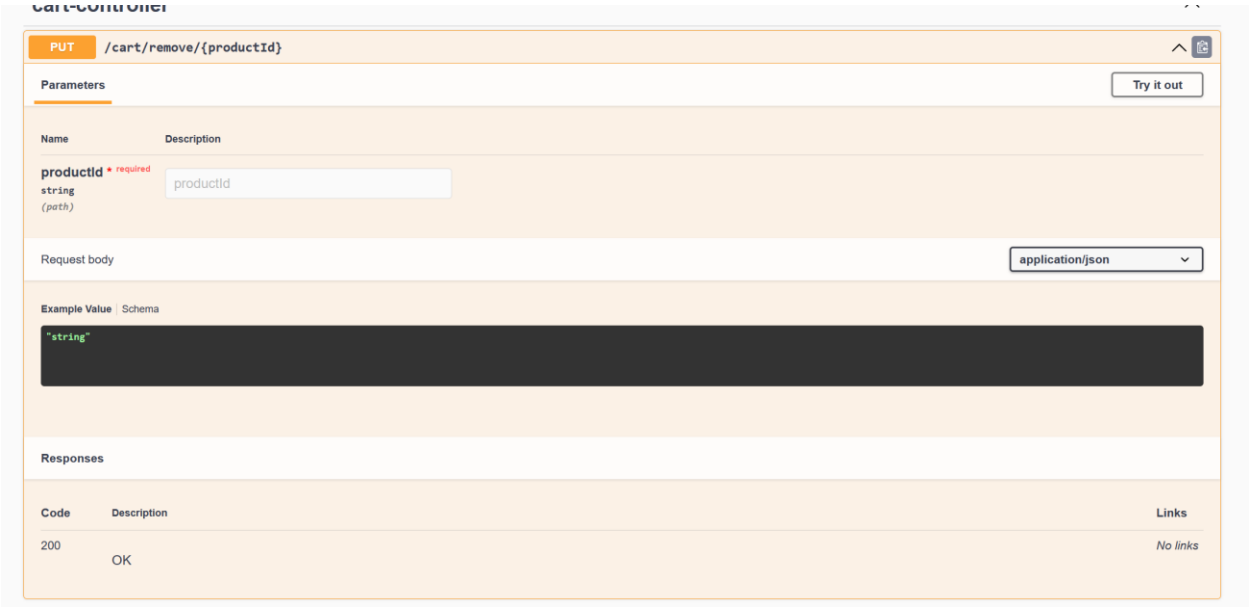

Рис. 2.4.10. API видалення продукту з кошика користувача. Видаляє тільки один інстанс продукту.

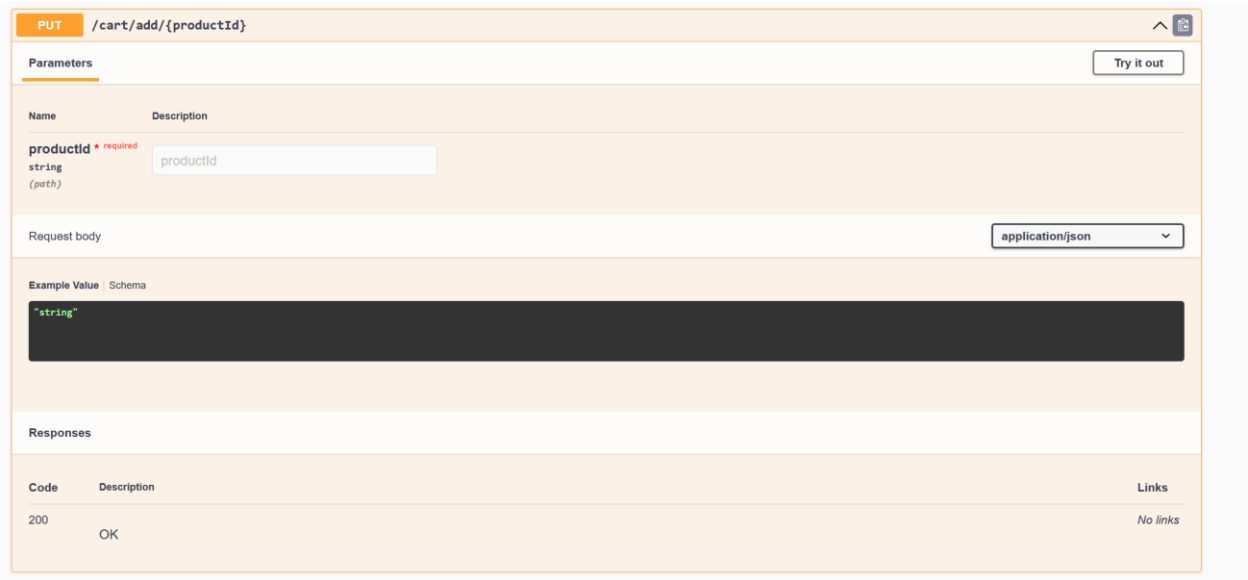

Рис. 2.4.11. API додавання продукта до кошика. Якщо продукт вже існує, збільшується його кількість в кошику.

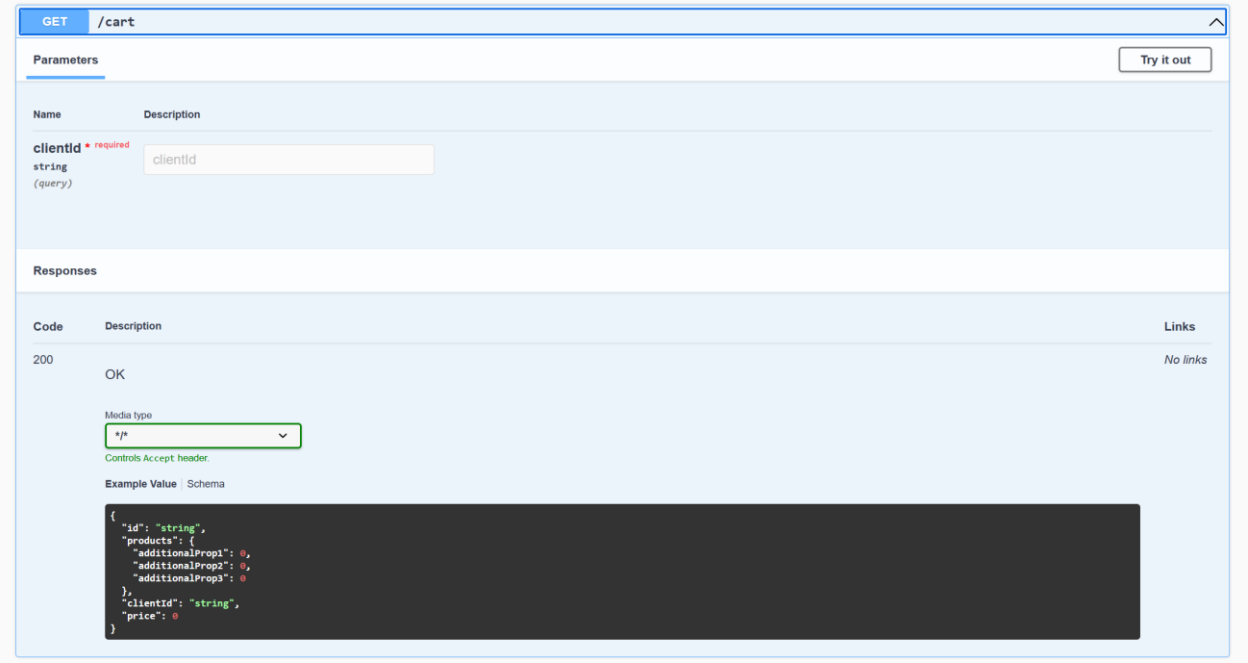

Рис. 2.4.12. API отримання всієї корзини користувача.

Логіка отримання корзини побудована наступним чином:

- Отримуємо список продуктів, які знаходяться в цьому кошику
- За допомогою StreamAPI рахуємо ціну
- За допомогою StreamAPI робимо зі списку продуктів список

об’єктів з (Назваою, брендом, кількістю) продукта

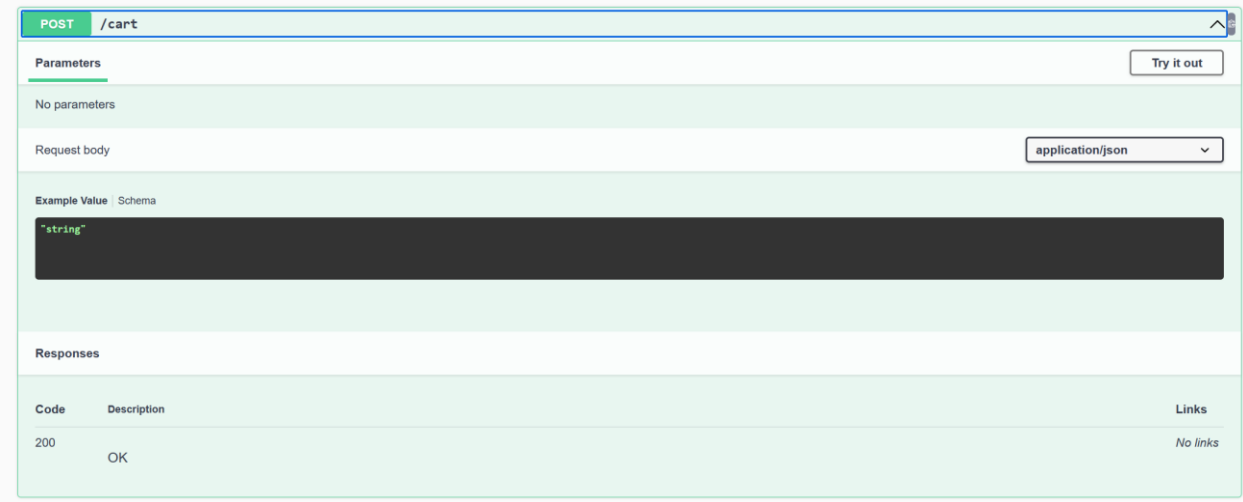

Рис. 2.4.13. API створення кошика користувача.

Даний ендпоїнт повинен викликатися фронтом одразу після реєстрації користувача. Інакше у юзера кошик буде відсутній.

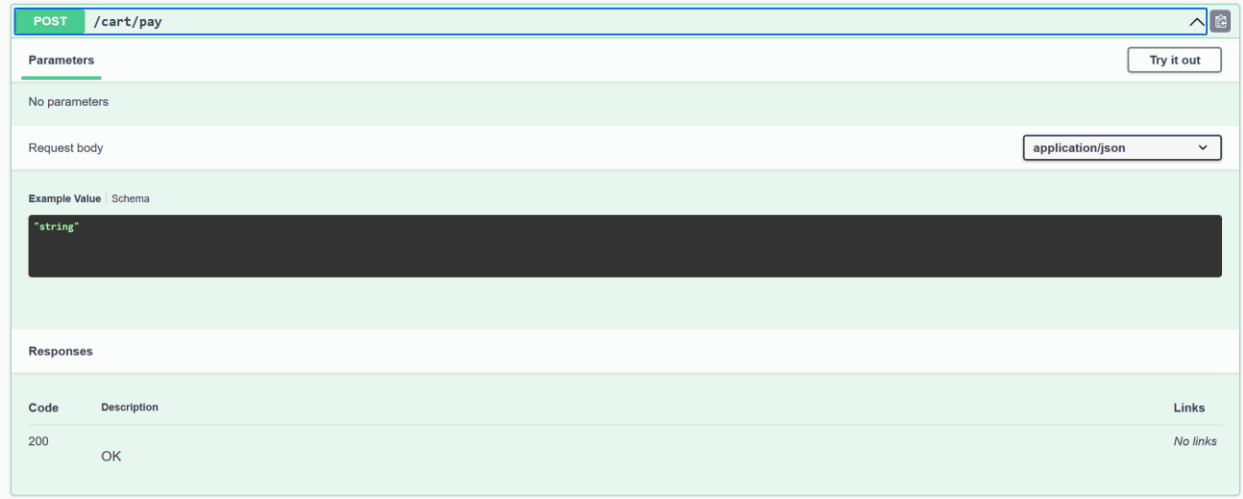

Рис. 2.4.14. API оплати кошика користувача.

При виклику API оплати кошика, юзеру на електронну пошту буде відправлено повідомлення з кодом підтвердження, який потрібно буде передати в API підтвердження оплати кошика(рис. 2.4.15)

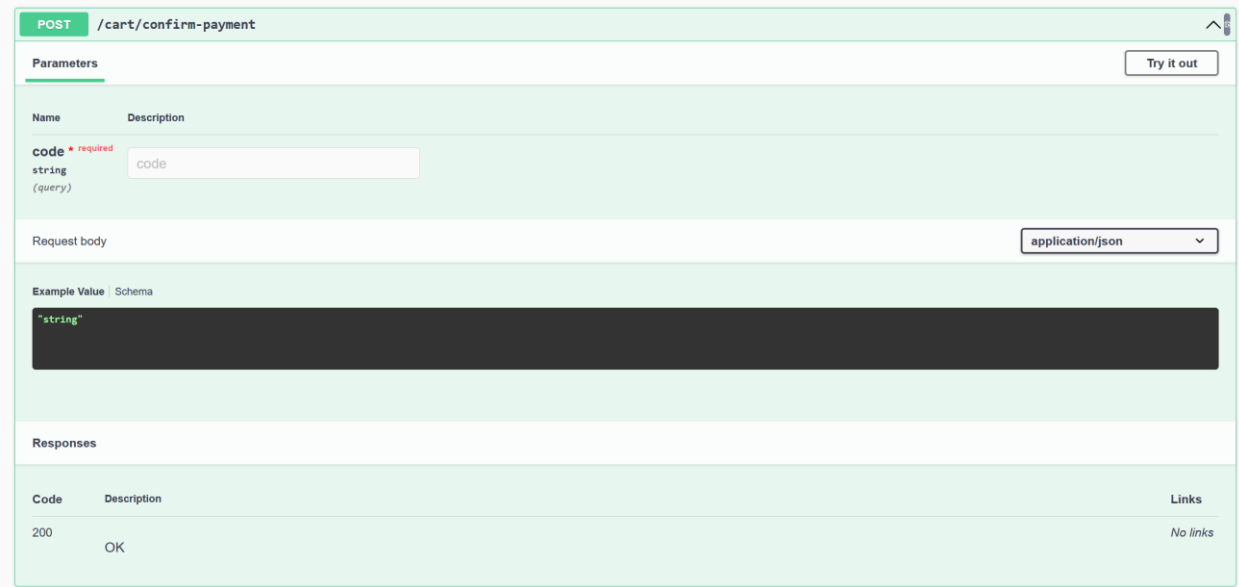

Рис. 2.4.15. API підтвердження оплати кошика.

Після підтвердження оплати, кошик користувача спорожнюється і усі товари, які були в ньому перед оплатою переносяться в об’єкт Замовлення.

API взаємодії з замовленнями вказано на рисунках 2.4.16-2.4.17.

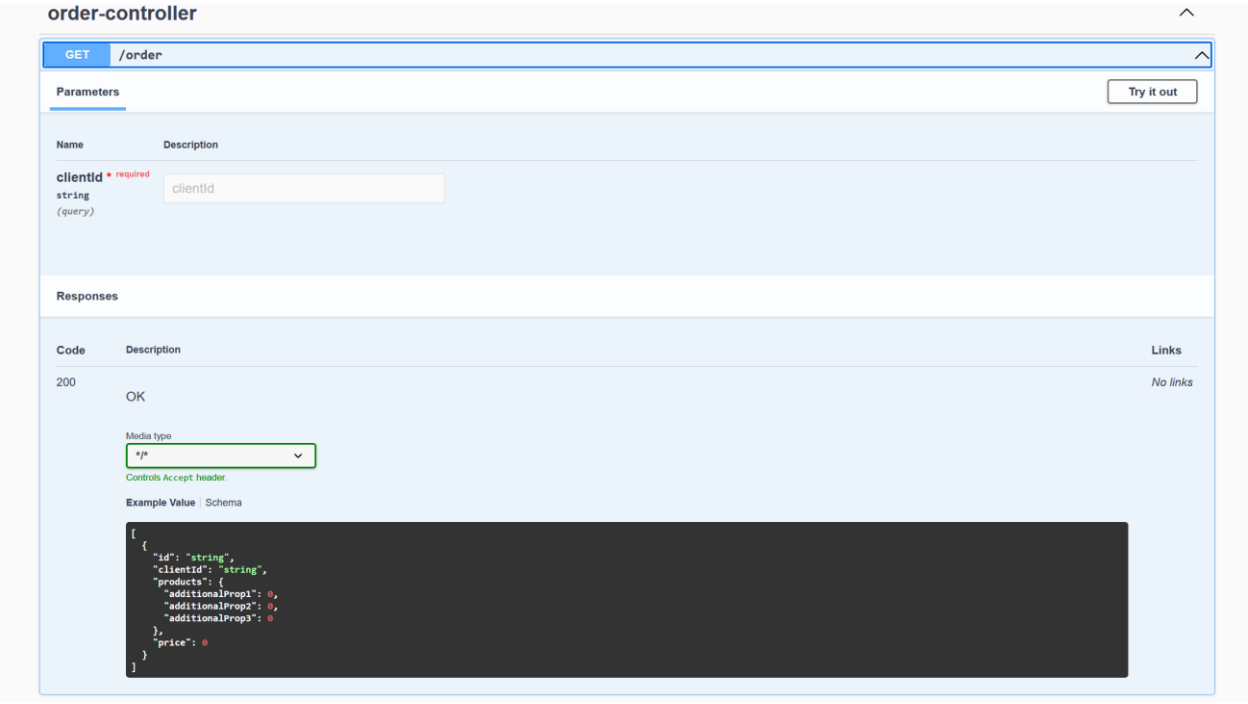

Рис. 2.4.16. API отримання всіх попередніх замовлень користувача.

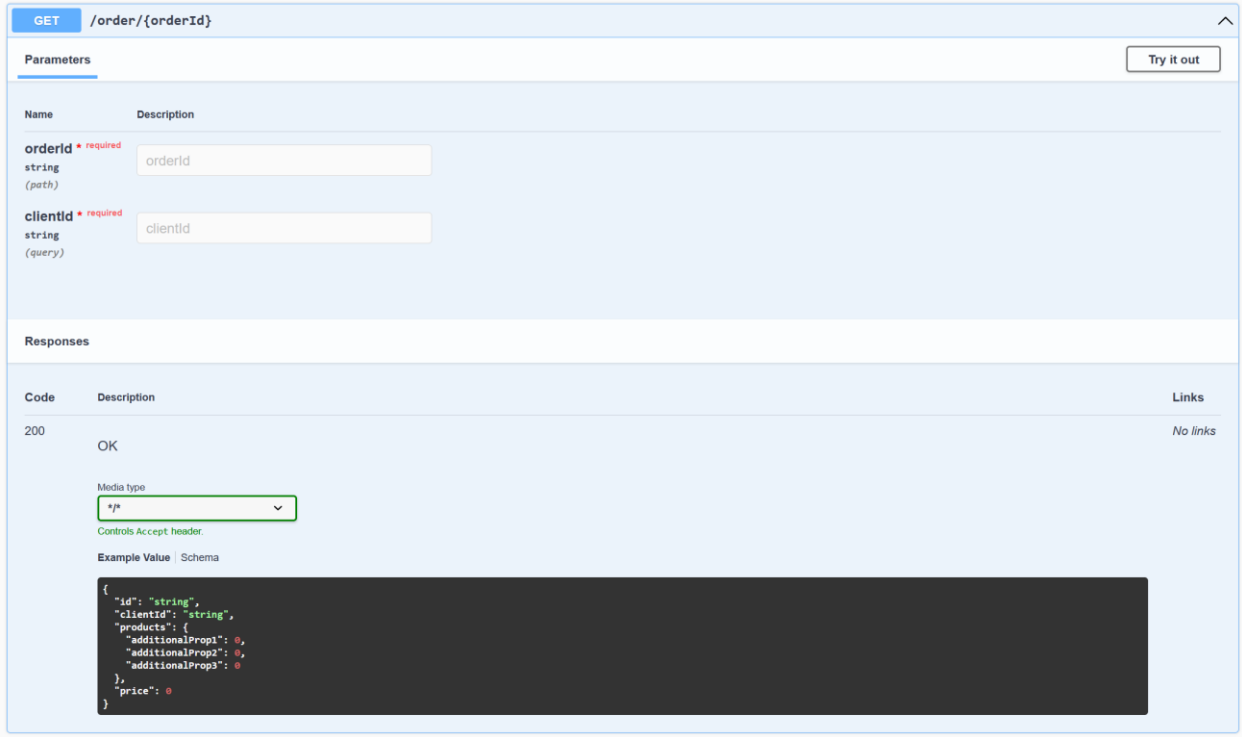

Рис. 2.4.17. API отримання замовлення користувача за його ідентифікатором.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідні дані:

Для обробки вхідних даних з HTTP-запитів у форматі JSON у Spring Boot додатку, використовуються анотації контролерів, такі як @RequestBody, які дозволяють автоматично прив'язувати JSON-об'єкти до моделей даних.

Валідація вхідних даних може бути забезпечена за допомогою анотацій, таких як @Valid, які використовуються разом із власними або вбудованими валідаторами.

Вихідні дані:

Spring Boot додатки можуть повертати відповіді у форматі JSON за допомогою об'єктів моделей, які автоматично конвертуються в JSON за допомогою вбудованих механізмів серіалізації.

Для керування форматуванням вихідних даних, таких як додавання або вилучення певних полів, можна використовувати анотації, такі як @JsonIgnore

або @JsonView.

Щоб забезпечити стабільність та сумісність вихідних даних, рекомендується використовувати версію API або контроль версій за допомогою URL-шляхів або заголовків запиту.

Організація коду:

Для зручності краще розділити логіку обробки вхідних та вихідних даних на контролери та сервіси. Контролери відповідають за обробку HTTP-запитів та передачу даних у сервіси, де відбувається бізнес-логіка.

Використання DTO (Data Transfer Object) для обміну даними між контролерами та сервісами допомагає визначити, які дані передаються через межі компонентів програми.

Використання аспектно-орієнтованого програмування (AOP) може спростити обробку вхідних та вихідних даних шляхом виділення загальної функціональності, такої як логування чи аутентифікація.

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Проект написаний на Java, а значить може бути запущений на будь-якому пристрої, що підтримує JVM машину.

В рамках тестування сервер та база даних були запущені на HP Probook 440 G9, який має такі характеристики:

1) CPU: 12 ядерний 64-бітний 12th Gen Intel(R) Core(TM) i7-1255U 1.70 GHz

2) GPU: інтегрований в процессор.

3) RAM: 32 ГБ LPDDR4.

Технічні засоби не мають обов’язково ідентичних характеристик для безперешкодної роботи системи.

2.6.2. Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

1) IntelliJ IDEA Ultimate Edition 2022.3.3;
2) Docker Desktop;
3) HeidiSQL, PostgreSQL;
4) Postman;
5) Mozilla Firefox.

2.6.3. Виклик та завантаження програми

Для початку роботи програми необхідно запустити контейнер за базою даних на порті 5432 і викликати метод com.vasyl.ntudp.diploma. DiplomaApplication::main. Усі необхідні таблиці будуть створені автоматично за допомогою Spring Data JPA.

2.6.4. Опис інтерфейсу користувача

Після завершення налаштування та запуску серверу й переходу за посиланням http://localhost:8080/swagger-ui/index.html ми зможемо побачити усі доступні для викликів ендпоїнти нашого за стосунку (рис 2.6.1). За допомогою Swagger UI юзер може побачити приклад запиту і відповіді, які надійдуть з цього ендпоїнту (рис .2.6.2).

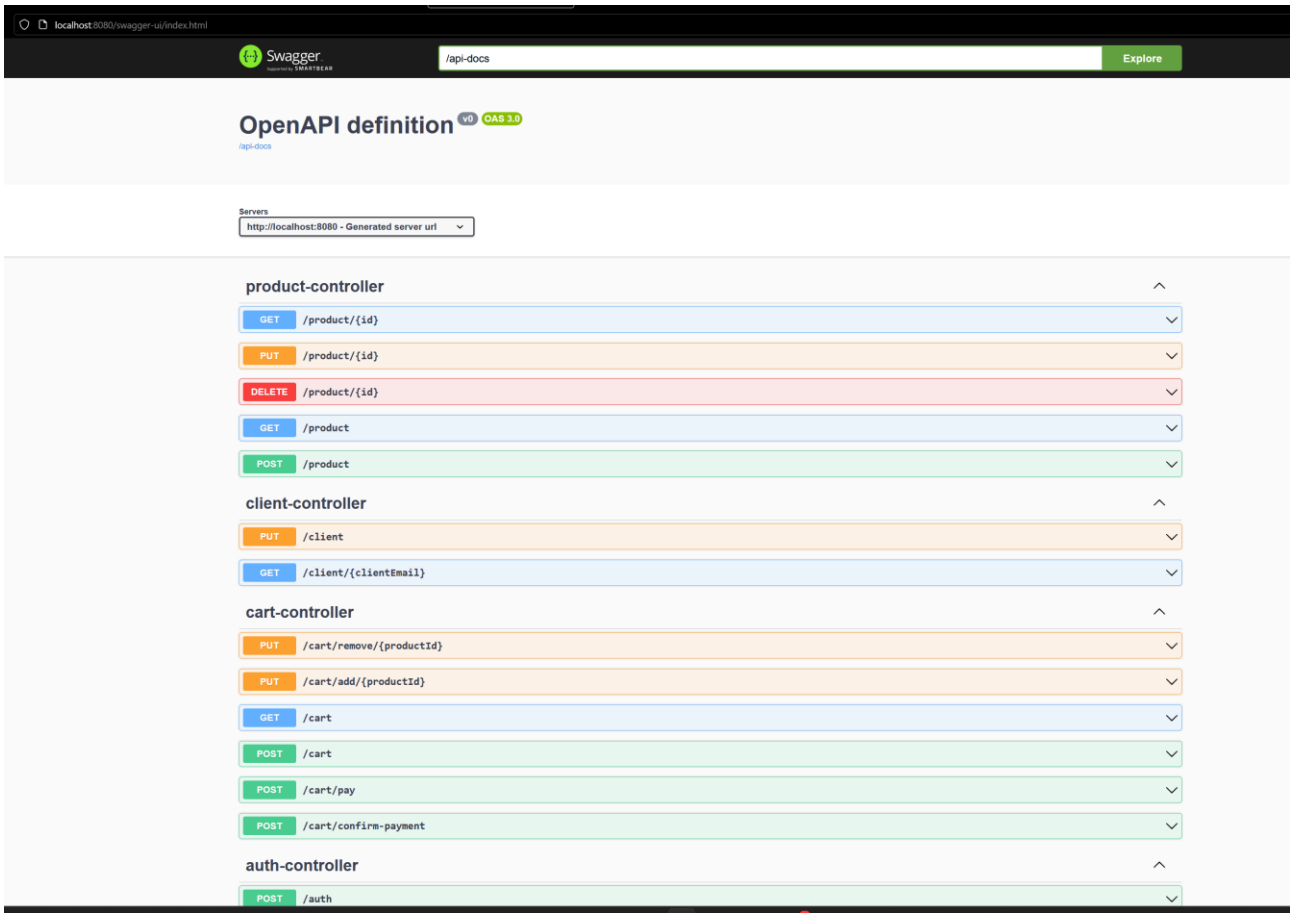

Рис. 2.6.1. Документація з усіма доступними для викликів ендпоїнтів сервера.

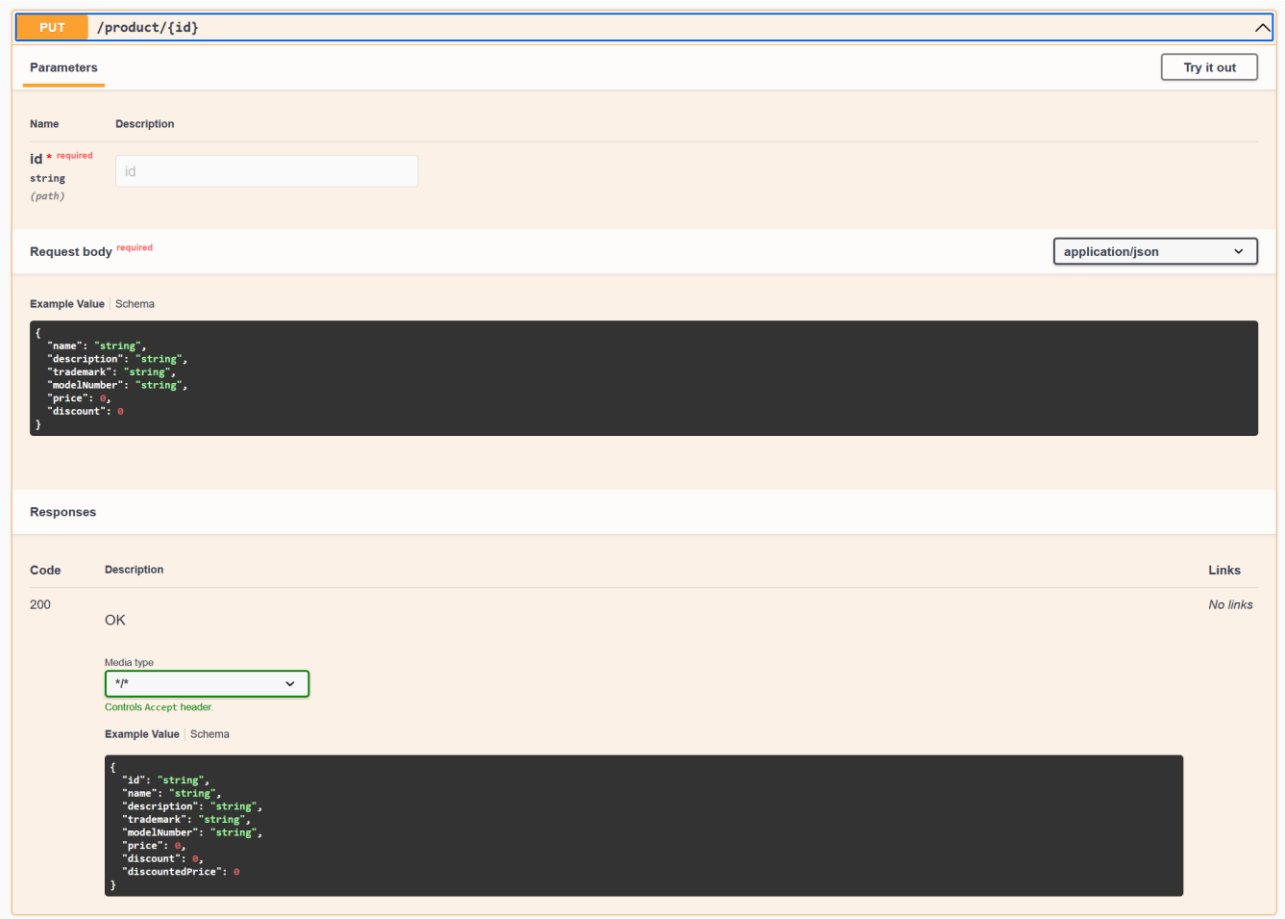

Рис. 2.6.2. Приклад запиту і відповіді сервера від одного з ендпоїнтів.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості інформаційної системи

Вихідні дані розробки програмного забезпечення:

1) передбачуване число операторів програми: 2342.

2) коефіцієнт складності програми: 1,2.

3) коефіцієнт корекції програми в ході її розробки: 0,1.

4) годинна заробітна плата програміста, грн/год: 496. Станом на початок 2024 року, заробітна плата розробника Java , варіюється від 500$ до 4000$. Таким чином, середня заробітна плата розробника складає 2200$. На середину квітня 2024 року курс валют складає 39,73 грн долар США. Отже, середня зарплата в гривнях за місяць дорівнює приблизно 87406 грн. При стандартному графіку (176 годин/місяць) заробітна плата за годину буде становити приблизно 496 грн.

5) коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,56.

6) коефіцієнт кваліфікації програміста, обумовлений від стажу роботи: 1,3.

7) вартість машино-години ЕОМ, грн/год: 1,2

Для роботи цього проекту потрібен сервер. Щоб піддримувати стабільну роботу домашнього серверу під навантаженням потрібно приблизно 451 Вт. Ціна за 1 кВт/год при споживані понад 250 кВт на місяць станом на 20 квітня 2024 становить 2.64 грн. Тобто вартість години роботи комп’ютера дорівнює 0,451 * 2.64 = 1.2 грн.

Розрахуємо умовне число операторів за формулою:

$$Q = q \cdot C \cdot (1 + p)$$

де: q – передбачуване число операторів;

С – коефіцієнт складності програми;

р – коефіцієнт кореляції програми в ході її розробки;

Результат: 2342 * 1,2 * (1,1) = 3091,44

Розрахуємо витрати праці на дослідження алгоритму рішення задачі за формулою:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot K}$$

де: Q - умовне число операторів програми;

K - коефіцієнт кваліфікації програміста;

B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

Результат: (3091,44 * 1,56) / (77 * 1,3) = 48,17 людино-годин.

Розрахуємо витрати праці на розробку блок-схеми алгоритму за формулою:

$$t_a = \frac{Q}{(20..25) \cdot K}$$

Результат: 3091,44 / (23 * 1,3) = 103,3 людино-годин.

Розрахуємо витрати праці на програмування по готовій блок-схемі за формулою:

$$t_n = \frac{Q}{(20..25) \cdot K}$$

Результат: 3091,44 / (22 * 1,3) = 108,09 людино-годин.

Розрахуємо витрати праці на налагодження програми на ЕОМ за формулою:

$$t_{\text{отл}} = \frac{Q}{(4..5) \cdot K}$$

Результат: 3091,44 / (5 * 1,3) = 475,6 людино-годин.

Розрахуємо трудомісткість підготовки матеріалів і рукопису за формулою:

$$t_{dp} = \frac{Q}{(15..20) \cdot K}$$

Результат: 3091,44 / (17 * 1,3) = 139,88 людино-годин.

Розрахуємо трудомісткість редагування, печатки й оформлення документації за формулою:

$$t_{do} = 0{,}75 \cdot t_{dp}$$

Результат: 0,75 * 139,88 = 104,91 людино-годин.

Розрахуємо витрати праці на підготовку документації за формулою:

$$t_d = t_{dp} + t_{do}$$

Результат: 139,88 + 104,91 = 244,79 людино-годин.

Розрахуємо трудомісткості розробки програмного забезпечення за формулою:

$$t = t_o + t_u + u_a + t_n + t_{отл} + t_d$$

де: t_o - витрати праці на підготовку й опис поставленої задачі (приймається як 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

t_d - витрати праці на підготовку документації.

Результат: 50 + 48,17 + 103,3 + 108,09 + 475,6 + 244,79 = 1029,95 людино-годин.

3.2. Розрахунок витрат на створення програми

Розрахуємо плату виконавців за формулою:

$$З_{зп} = t + С_{пр}$$

де: t - трудомісткості розробки програмного забезпечення;

$С_{пр}$ - середня годинна заробітна плата програміста, грн/година;

Результат: 1029,95 * 496= 510885 грн.

Розрахуємо вартість машинного часу, необхідного для налагодження програми на ЕОМ за формулою:

$$З_{мв} = t_{отл} + С_{мч}$$

де: $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ - вартість машино-години ЕОМ, грн/год;

Результат: 475,6 * 1,2 = 570,72 грн.

Розрахуємо витрати на створення ПЗ за формулою:

$$К_{по} = З_{зп} + З_{мв}$$

де: $З_{зп}$ - плата виконавців;

$З_{мв}$ - вартість машинного часу, необхідного для налагодження програми на ЕОМ.

Результат: 510885 + 570,72 = 511455,72 людино-годин.

Розрахуємо очікуваний період створення ПЗ за формулою:

$$Т = \frac{t}{B_k \cdot F_p}$$

де: t - трудомісткості розробки програмного забезпечення;

B_k- число виконавців;

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 176 годин).

Результат: 1029,95 / (1 * 176)) = 5,8 людино-годин.

Висновки: на розробку даного програмного забезпечення піде 1029,95 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 5,8 місяців при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 510885 грн.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є розроблена і повністю функціональна інформаційна система для керування інтернет-магазином.

У цьому проєкті була створена надійна архітектура та реалізована в сучасному виконанні, використовуючи відомий та ефективний фреймворк Spring. Функціонал клієнтської частини дозволяє виконувати всі операції з CRUD без необхідності взаємодії з кодом, забезпечуючи доступ до управління кошиком і продуктами відповідно до наданих дозволів ролі користувача. Адміністративна частина є інтуїтивно зрозумілою і не вимагає спеціальної підготовки для управління ролями та статусом користувачів у системі.

Створений додаток сприятиме підвищенню продуктивності, швидкості та складності сценаріїв для повноцінного керування інтернет-магазином. Основа додатку є багатофункціональною, що дозволяє використовувати отримані результати як у практичній розробці програмного забезпечення зі зміною бізнес-логіки, так і для поглиблення рівня самостійного освоєння галузі.

Для реалізації застосунку використовувалися сучасні технології програмування: Java 21, Spring Boot, Spring Data, MongoDB, Spring Security, Lombok. Розроблений проект інтернет-магазину є корисним для підтримки вебсайтів для більшості надавачів послуг. Таким чином, в ході виконання кваліфікаційної роботи були досягнуті всі поставлені цілі та виконані всі вимоги, що ставилися при проектуванні інформаційної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WALLS Craig. Spring in action. Simon and Schuster, 2022. p. 492

2. PONGE Julien. Vert. x in Action: Asynchronous and Reactive Java. Manning Publications, 2020. p. 330

3. STEFANOV Stoyan. React: Up & Running. " O'Reilly Media, Inc.", 2021. p. 230

4. MEYER Eric A. CSS: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.", 2006. p. 538

5. KUSUMAH Ariq Rafi; ANDARSYAH Roni. 5 Tahap Membuat Dashboard Admin Untuk Kemudahan Programmer Dengan ReactJS dan TailwindCSS (Studi Kasus: Data Koleksi Museum). Penerbit Buku Pedia, 2023. p. 96

6. BLOCH Joshua. Effective java (the java series). Prentice Hall PTR, 2008. p. 416

7. GOETZ Brian, et al. Java concurrency in practice. Pearson Education, 2006. p. 234

8. Java. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 09.02.2024).

9. Spring Framework Documentation. [Електронний ресурс] URL: https://docs.spring.io/spring-framework/reference/ (дата звернення: 09.02.2024).

10. Baeldung. [Електронний ресурс] URL: https://www.baeldung.com (дата звернення: 09.02.2024).

11. PostgreSQL Documentation. [Електронний ресурс] URL: https://www.postgresql.org/docs (дата звернення: 09.02.2024).

12. JWT. [Електронний ресурс] URL: https://jwt.io (дата звернення: 09.02.2024).

13. Swagger Documentation. [Електронний ресурс] URL: https://swagger.io/docs (дата звернення: 09.02.2024).

14. Java Documentation. [Електронний ресурс] URL: https://docs.oracle.com/en/java (дата звернення: 09.02.2024).

15. Docker Docs. [Електронний ресурс] URL: https://docs.docker.com (дата звернення: 09.02.2024).

16. Postman. [Електронний ресурс] URL: https://learning.postman.com/docs/introduction/overview (дата звернення: 09.02.2024).

17. DockerHub. [Електронний ресурс] URL: https://hub.docker.com (дата звернення: 09.02.2024).

18. MVN Repository. [Електронний ресурс] URL: https://mvnrepository.com (дата звернення: 09.02.2024).

19. Spring Initializr. [Електронний ресурс] URL: https://start.spring.io (дата звернення: 09.02.2024).

20. Geeks for Geeks. [Електронний ресурс] URL: https://www.geeksforgeeks.org/ (дата звернення: 09.02.2024).

ДОДАТОК А

КОД ПРОГРАМИ

ControllerInvocationLog.java

```
package com.vasyl.ntudp.diploma.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface ControllerInvocationLog {

}
```

InvocationLog.java

```
package com.vasyl.ntudp.diploma.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface InvocationLog {

}
```

UserId.java

```
package com.vasyl.ntudp.diploma.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.PARAMETER})
public @interface UserId {
boolean required() default true;
}
```

ControllerLogAspect.java

```
package com.vasyl.ntudp.diploma.aspect;

import com.vasyl.ntudp.diploma.annotation.ControllerInvocationLog;
import com.vasyl.ntudp.diploma.config.security.jwt.JwtUser;
import jakarta.servlet.http.HttpServletRequest;
import java.lang.annotation.Annotation;
import java.util.HashMap;
import java.util.Map;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
```

```
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.SoftException;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;

import static com.vasyl.ntudp.diploma.util.AspectUtilityClass.getAnnotations;


@Slf4j
@Aspect
@Component
@RequiredArgsConstructor
public class ControllerLogAspect {
private final HttpServletRequest request;
@SneakyThrows
@Around("execution(* *(..)) && @annotation(com.vasyl.ntudp.diploma.annotation.ControllerInvocationLog)")
public Object interceptMethodsWithAnnotatedParameters(ProceedingJoinPoint joinPoint) {
Annotation[][] annotations = getAnnotations(joinPoint);
logControllerMethodInvocation(getUserIdNewArg(joinPoint, annotations));
return joinPoint.proceed();
}

private void logControllerMethodInvocation(String data) {
Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
log.info("User with id {} made {} {} request and data: {}", getUserId(authentication), request.getMethod(),
request.getRequestURI(), data);
}

private String getUserIdNewArg(ProceedingJoinPoint joinPoint, Annotation[][] annotations) {
int i = 0;
StringBuilder result = new StringBuilder();
for (Object arg : joinPoint.getArgs()) {
for (Annotation annotation : annotations[i]) {
if (annotation.annotationType() == RequestParam.class) {
result.append("Request Parameter: ").append(arg).append(" ");
}
if (annotation.annotationType() == RequestBody.class) {
result.append("Request Body: ").append(arg).append(" ");
}
}
i++;
}
return result.toString();
}

private String getUserId(Authentication authentication) {
```

```
try{
JwtUser user = (JwtUser) authentication.getPrincipal();
return user.getId();
}catch (ClassCastException e){
return "anonymous user";
}
}
}
```

IdAspect.java

```
package com.vasyl.ntudp.diploma.aspect;

import com.vasyl.ntudp.diploma.annotation.UserId;
import com.vasyl.ntudp.diploma.config.security.jwt.JwtUser;
import com.vasyl.ntudp.diploma.exception.UnauthorizedException;
import jakarta.servlet.http.HttpServletRequest;
import java.lang.annotation.Annotation;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

import static com.vasyl.ntudp.diploma.util.AspectUtilityClass.getAnnotations;

@Slf4j
@Aspect
@Component
@RequiredArgsConstructor
public class IdAspect {

private final HttpServletRequest request;

@SneakyThrows
@Around("execution(public * *(.., @com.vasyl.ntudp.diploma.annotation.UserId (*), ..))")
public Object interceptMethodsWithAnnotatedParameters(ProceedingJoinPoint joinPoint) {
Annotation[][] annotations = getAnnotations(joinPoint);
Object[] newArgs = getUserIdNewArg(joinPoint, annotations);
return joinPoint.proceed(newArgs);
}
private Object[] getUserIdNewArg(ProceedingJoinPoint joinPoint, Annotation[][] annotations) {
Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
int i = 0;
Object[] newArgs = new Object[joinPoint.getArgs().length];
for (Object arg : joinPoint.getArgs()) {
for (Annotation annotation : annotations[i]) {
if (annotation.annotationType() == UserId.class) {
UserId userIdAnnotation = (UserId) annotation;
if (!userIdAnnotation.required() && authentication.getPrincipal().equals("anonymousUser")) {
newArgs[i] = null;
} else if (userIdAnnotation.required() && authentication.getPrincipal().equals("anonymousUser")) {
```

```
throw new UnauthorizedException("Cannot recognize user token");
} else {
newArgs[i] = getUserId(authentication);
}
} else {
newArgs[i] = arg;
}
}
i++;
}
return newArgs;
}

private String getUserId(Authentication authentication) {
JwtUser user = (JwtUser) authentication.getPrincipal();
return user.getId();
}
}
```

InvocationLogAspect.java

```
package com.vasyl.ntudp.diploma.aspect;

import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import lombok.extern.slf4j.Slf4j;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.reflect.MethodSignature;
import org.springframework.stereotype.Component;

@Slf4j
@Aspect
@Component
@RequiredArgsConstructor
public class InvocationLogAspect {

@SneakyThrows
@Around("execution(* *(..)) && @annotation(com.vasyl.ntudp.diploma.annotation.InvocationLog)")
public Object interceptMethodsWithAnnotatedParameters(ProceedingJoinPoint joinPoint) {
MethodSignature signature = (MethodSignature) joinPoint.getSignature();
logControllerMethodInvocation(joinPoint.getThis(), joinPoint.getSignature().getName(), joinPoint.getArgs());
return joinPoint.proceed();
}

private void logControllerMethodInvocation(Object joinPointThis, String methodName, Object[] args) {
log.info("{} {} method was called with arguments: {}", joinPointThis, methodName, args);
}

}
```

JwtAuthenticationException.java

```
package com.vasyl.ntudp.diploma.config.security.jwt;

import org.springframework.security.core.AuthenticationException;
```

```
public class JwtAuthenticationException extends AuthenticationException {

public JwtAuthenticationException(String msg, Throwable t) {
super(msg, t);
}

public JwtAuthenticationException(String msg) {
super(msg);
}
}
```

JwtConfigurer.java

```
package com.vasyl.ntudp.diploma.config.security.jwt;

import org.springframework.security.config.annotation.SecurityConfigurerAdapter;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.DefaultSecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;


public class JwtConfigurer extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

private JwtTokenProvider jwtTokenProvider;

public JwtConfigurer(JwtTokenProvider jwtTokenProvider) {
this.jwtTokenProvider = jwtTokenProvider;
}

@Override
public void configure(HttpSecurity httpSecurity) throws Exception {
JwtTokenFilter jwtTokenFilter = new JwtTokenFilter(jwtTokenProvider);
httpSecurity.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);
}
}
```

JwtTokenFilter.java

```
package com.vasyl.ntudp.diploma.config.security.jwt;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.GenericFilterBean;


@RequiredArgsConstructor
public class JwtTokenFilter extends GenericFilterBean {

private final JwtTokenProvider jwtTokenProvider;
```

```
@Override
public void doFilter(ServletRequest req, ServletResponse res, FilterChain filterChain) throws IOException,
ServletException {
HttpServletResponse response = (HttpServletResponse) res;
HttpServletRequest request = (HttpServletRequest) req;

response.setHeader("Access-Control-Allow-Origin", request.getHeader("Origin"));
response.setHeader("Access-Control-Allow-Credentials", "true");
response.setHeader("Access-Control-Allow-Methods", "POST, GET, PUT, PATCH, OPTIONS, DELETE");
response.setHeader("Access-Control-Max-Age", "3600");
response.setHeader("Access-Control-Allow-Headers", "Authorization, content-type, X-API-KEY, xrc_header");

String token = jwtTokenProvider.resolveToken(request);

if (token != null) {
jwtTokenProvider.validateToken(token);
try {
Authentication auth = jwtTokenProvider.getAuthentication(token);
SecurityContextHolder.getContext().setAuthentication(auth);
} catch (Exception ignored) {
}
}

if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
response.setStatus(HttpServletResponse.SC_OK);
} else {
filterChain.doFilter(req, res);
}
}

}
```

CartService.java

```
package com.vasyl.ntudp.diploma.service;

import com.vasyl.ntudp.diploma.annotation.InvocationLog;
import com.vasyl.ntudp.diploma.database.entity.Cart;
import com.vasyl.ntudp.diploma.database.entity.Client;
import com.vasyl.ntudp.diploma.database.entity.Product;
import com.vasyl.ntudp.diploma.database.repository.CartRepository;
import com.vasyl.ntudp.diploma.dto.cart.CartResponseDto;
import com.vasyl.ntudp.diploma.exception.CartNotFoundException;
import com.vasyl.ntudp.diploma.exception.InvalidPasswordException;
import com.vasyl.ntudp.diploma.mapper.cart.CartMapper;
import com.vasyl.ntudp.diploma.util.HashUtil;
import jakarta.transaction.Transactional;
import java.math.BigDecimal;
import java.util.ArrayList;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

@Service
```

```
@Slf4j
@RequiredArgsConstructor
public class CartService {

    private final CartRepository cartRepository;
    private final ClientService clientService;
    private final ProductService productService;
    private final EmailService emailService;
    private final ConfirmationCodeService confirmationCodeService;
    private final OrderService orderService;

    @InvocationLog
    public void createCartForClient(String clientId) {
        Client client = clientService.getById(clientId);

        Cart cart = new Cart();
        cart.setClient(client);
        cart.setPrice(BigDecimal.ZERO);

        cartRepository.save(cart);
    }

    @InvocationLog
    public void addProductToCart(String productId, String clientId) {
        Cart cart = getByClientId(clientId);
        Product product = productService.getById(productId);

        cart.addProduct(product);

        cartRepository.save(cart);
    }

    @InvocationLog
    public void removeFromCart(String productId, String clientId) {
        Cart cart = getByClientId(clientId);
        Product product = productService.getById(productId);

        cart.removeProduct(product);

        cartRepository.save(cart);
    }

    @InvocationLog
    public CartResponseDto getCartByUserId(String clientId) {
        Cart cart = getByClientId(clientId);
        return CartMapper.toCartResponseDto(cart);
    }

    @InvocationLog
    public Cart getByClientId(String clientId) {
        return cartRepository.getCartByClientId(clientId)
                .orElseThrow(() -> new CartNotFoundException("Cart not found for client " + clientId));
    }
```

```
    @InvocationLog
    public void payForCart(String clientId) {
        Cart cart = getByClientId(clientId);

        String confirmationCode = HashUtil.getSaltString();
        confirmationCodeService.saveConfirmationCode(cart.getId(), confirmationCode);

        emailService.sendEmail("Your code to confirm payment for cart: " + confirmationCode,
cart.getClient().getEmail(),
                "Payment confirmation code");
    }

    @InvocationLog
    @Transactional
    public void checkConfirmationCode(String clientId, String confirmationCode) {
        Cart cart = getByClientId(clientId);

        if (!confirmationCodeService.containsCode(cart.getId(), confirmationCode)) {
            throw new InvalidPasswordException("Code is wrong. Check your email for right code");
        }

        orderService.createOrder(cart);

        emailService.sendEmail("Confirmation successful. Thank you for your order!", cart.getClient().getEmail(),
                "Confirmation successful");
        confirmationCodeService.deleteCode(cart.getId(), confirmationCode);

        cart.setProducts(new ArrayList<>());
        cart.setPrice(BigDecimal.ZERO);
        cartRepository.save(cart);
    }
}
```

Решта коду додається окремо на CD-диску.

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра на тему:

"Розробка backend застосунку для інтернет магазину"

студента групи 122-20-3 Веселова Василя Олеговича

Керівник економічного розділу **Л.В. Касьяненко**
доц. каф. ПЕП та ПУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Веселов.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Веселов.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Веселов.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Веселов.ppt	Презентація кваліфікаційної роботи