Національний університет «Одеська політехніка»
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Об’єктно-орієнтоване програмування»

Тема «Розробка веб-додатка з продажу техніки»

Студента 2 курсу АІ-221 групи
Спеціальності 122 – «Комп’ютерні науки»
Попазова П.І.
Керівник д. Годовиченко М.А.

Національна шкала _____

Кількість балів: _____

Оцінка: ECTS _____

Члени комісії

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

м. Одеса – 2024 рік

ЗМІСТ

Національний університет «Одеська політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних систем

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ

студентці Попазову Петру Івановичу група АІ-221

1. Тема роботи «Розробка веб-додатка з продажу техніки»

2. Термін здачі студентом закінченої роботи 09.06.2024

3. Початкові дані до проекту (роботи). Користувачі мають змогу зарєструватися та у подальшому виконувати процес автентифікації (при реєстрації отримують посилання на електронну пошту для активації акаунту). Реалізація за допомогою JWT токенів. Користувачі можуть додати товари у кошик та у подальшому придбати, вказаши дані для відправки, та отримають лист-підтвердження з вмістом замовлення.

4. Зміст розрахунково-пояснювальної записки: Розділ 1 - Проектування і розробка веб-шару (REST-запити). Розділ 2 - Проектування і розробка сервісного шару. Розділ. 3 - Проектування і розробка шару сховища. Розділ 4 - Проектування і розробка журналювання Розділ 5 - Проектування і розробка безпеки. Розділ 6 - Розгортання додатка.

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

Завдання видано 11.04.2024 _____
(підпис викладача)

Завдання прийнято до виконання 11.04.2024 _____
(підпис студента)

АНОТАЦІЯ

Було розроблено веб-додаток з продажу техніки (електронної комерції). Користувачам пропонується зручний метод для реєстрації з покроковою валідацією даних. Запропоновано зрозумілий та user-friendly інтерфейс. Розроблено, на стороні клієнта, інтерфейс для додавання товарів у кошик, видалення, збільшення та зменшення їх кількості, очищення корзини. Розроблена форма для введення даних про відправку та у разі успішної валідації, користувач отримує лист-підтвердження з вмістом замовлення. На стороні сервера було запроваджено усі необхідні точки для правильної роботи веб-додатка, наприклад, реєстрація, автентифікація користувача, виведення усіх товарів та категорій, зроблення замовлення та надсилання даних для доставки (деякі кінцеві точки не потребують токен). Розроблено серівс для надсилання листів. Запрпоновано глобальний оброблювач помилок. Запроваджено логування для аудиту системи. Використані технології включають Spring Boot 3, Spring Security 6, JWT Token, PostgreSQL, Spring Data JPA, OpenAPI (Swagger), React.

ABSTRACT

An e-commerce web application was developed. Users are offered with a convenient method for registration with step-by-step data validation. A clear and user-friendly buying interface is offered. Functionality for adding products to the cart, removing, increasing and decreasing their quantity, cleaning the cart has been developed on the client side. A form has been implemented for entering shipping data, and in case of successful validation, the user receives a confirmation letter with the contents of the order. On the server side, all the necessary endpoints for the web application to work correctly have been implemented, such as registration, user authentication, listing all products and categories, placing an order, and sending data for delivery (some endpoints do not require tokens). A service for sending letters has been developed. A global error handler is provided. Logging for system audit has been introduced. Technologies such as Spring Boot 3, Spring Security 6, JWT Token, PostgreSQL, Spring Data JPA, OpenAPI (Swagger), React were used.

ВСТУП

Розробка веб-додатка для електронної комерції є релевантною з кількох причин, особливо в сучасну епоху цифрових технологій, коли онлайн-магазини стають все більш поширеними. По-перше, це – ширше охоплення та доступність: веб-додаток для електронної комерції дозволяє компаніям охопити глобальну аудиторію. На відміну від звичайних магазинів, онлайн-магазини доступні цілодобово і без вихідних, що дозволяє клієнтам з різних часових поясів і місць робити покупки, коли їм зручно. По-друге, – зручність для клієнтів: клієнти цінують зручність покупок онлайн. Вони можуть переглядати товари, порівнювати ціни, читати відгуки та робити покупки, не виходячи з дому. Ця зручність може призвести до більшої задоволеності та лояльності клієнтів. Рентабельність: Керування магазином електронної комерції часто є більш рентабельним, ніж утримання фізичної вітрини. Компанії можуть заощадити на оренді, комунальних послугах та інших накладних витратах, пов’язаних із підтримкою фізичного розташування. Нарешті, конкурентна перевага: у багатьох галузях наявність електронної комерції більше не є обов’язковою, а необхідною для збереження конкурентоспроможності. Компанії, які не адаптуються до цифрового ринку, ризикують втратити клієнтів через конкурентів, які пропонують варіанти онлайн-покупок.

Використання REST API для розробки веб-додатків електронної комерції є широко поширеним підходом завдяки його гнучкості, масштабованості та простоті інтеграції.

REST (Representational State Transfer) — це архітектурний стиль для проектування мережевих програм. Він покладається на зв’язок клієнт-сервер без збереження стану та використовує стандартні методи HTTP, такі як GET, POST, PUT, DELETE тощо для операцій CRUD.

Spring Boot дозволяє легко створювати автономні додатки на основі Spring продуктивного класу, які можна «просто запускати». Переваги використання Spring Boot [1]:

– створення автономних програми Spring;

– застосування Tomcat, Jetty або Undertow безпосередньо (не потрібно розгортати WAR-файли);

– надання «початкових» залежності, за для спрощення конфігураційної збірки;

– автоматичне налаштовування Spring і бібліотеки сторонніх розробників, коли це можливо;

– надання готових до виробництва функції, такі як показники, перевірки працездатності та зовнішня конфігурація;

– абсолютна відсутність генерації коду та жодних вимог щодо конфігурації XML.

1 ПРОЕКТУВАННЯ І РОЗРОБКА ВЕБ-ШАРУ (REST-ЗАПИТИ)

Для відображення усього фунцілналу системи було розроблено Use-Case діаграму (рис. 1.1).

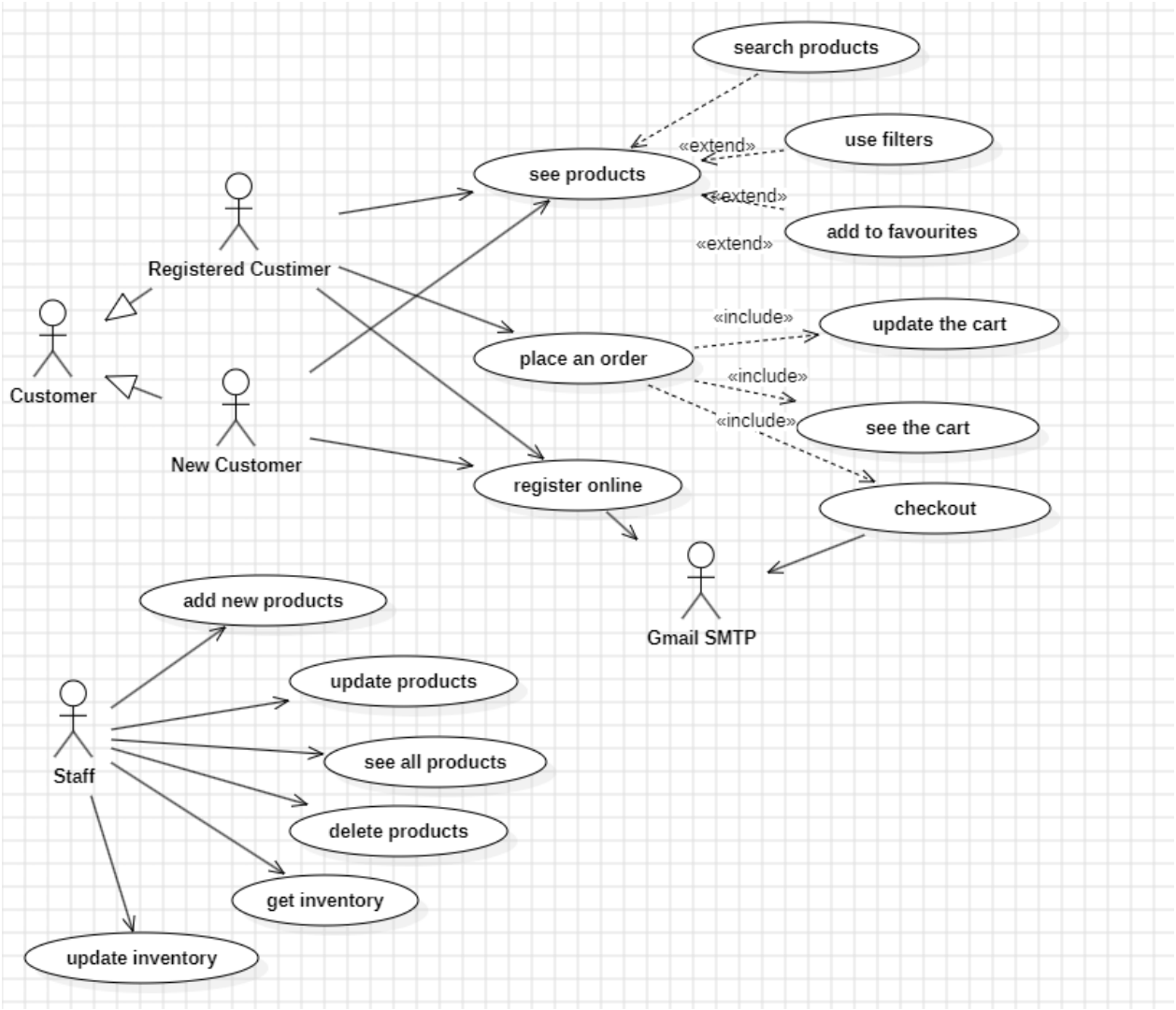

Рисунок 1.1 – Use-Case діаграма

У застосунку наявні наступні контролери з методами:

1. AuthController — це REST контролер, який відповідає за обробку операцій автентифікації та керування користувачами. Він забезпечує кінцеві точки для реєстрації користувача, входу в систему, активації облікового запису, отримання даних користувача та зміни паролів. Нижче наведено детальний опис кожної кінцевої точки в AuthController: Кінцеві точки:

URL: /register

Метод: POST

Опис: реєструє нового користувача з роллю «CUSTOMER». Кінцева точка приймає деталі реєстрації у формі об’єкта RegistrationBody, обробляє реєстрацію через AuthService та повертає RegisterResponse.

URL: / login

Метод: POST

Опис: автентифікує користувача за допомогою його облікових даних, наданих в об’єкті LoginBody. Кінцева точка повертає LoginResponse, що містить деталі автентифікації.

URL-адреса: /activate-account

Метод: GET

Опис: активує обліковий запис користувача за допомогою наданого токена активації. Кінцева точка підтверджує активацію та повертає повідомлення про успіх.

URL-адреса: /me

Метод: GET

Опис: отримує відомості про поточного автентифікованого користувача. Кінцева точка повертає інформацію про користувача.

URL-адреса: /change-password

Метод: PATCH

Опис: дозволяє поточному автентифікованому користувачеві змінити свій пароль. Кінцева точка приймає об’єкт ChangePasswordBody з деталями нового пароля та обробляє зміну через AuthService. Після успішної зміни пароля буде надіслано електронний лист із підтвердженням.

2. StaffController — це REST контролер, призначений для керування операціями, пов’язаними з персоналом, включаючи реєстрацію персоналу та операції CRUD для продуктів. Цей контролер гарантує, що лише користувачі з роллю STAFF можуть виконувати певні операції за допомогою анотацій

@PreAuthorize. Нижче наведено детальний опис кожної кінцевої точки в StaffController: Кінцеві точки:

URL: /staff/register

Метод: POST

Опис: реєструє нового співробітника. Кінцева точка приймає відомості про реєстрацію персоналу в об’єкті RegistrationBody, установлює роль «STAFF», обробляє реєстрацію через AuthService та повертає RegisterResponse.

URL: /staff/product

Метод: POST

Опис: додає новий продукт. Ця кінцева точка приймає відомості про продукт в об’єкті ProductBody та обробляє додавання через StaffService. Доступно лише користувачам із роллю STAFF.

URL-адреса: /staff/product/{id}

Метод: GET

Опис: Отримує продукт за його ідентифікатором. Ця кінцева точка повертає відомості про продукт і доступна лише користувачам із роллю ПЕРСОНАЛ.

URL-адреса: /staff/product/{id}

Метод: ПОСТАВИТИ

Опис: оновлює відомості про наявний продукт, ідентифікований за його ідентифікатором. Кінцева точка приймає оновлені відомості про продукт в об’єкті ProductBody та обробляє оновлення через StaffService. Доступно лише користувачам із роллю STAFF.

Метод: DELETE

URL: /staff/product/{id}

Опис: видаляє продукт, ідентифікований за його ідентифікатором. Кінцева точка обробляє видалення через StaffService і не повертає вміст після успішного видалення. Доступно лише користувачам із роллю ПЕРСОНАЛ.

3. ProductController — це REST контролер, призначений для керування операціями, пов’язаними з продуктом. Наразі він надає кінцеву точку для

отримання всіх продуктів, доступних у системі. Нижче наведено детальний опис кінцевої точки в ProductController. Кінцеві точки:

URL: /products

Метод: GET

Опис: отримує список усіх продуктів. Ця кінцева точка викликає ProductService, щоб отримати всі продукти та повертає список. Ця кінцева точка не має жодних обмежень доступу на основі ролей, і до неї може отримати доступ будь-який користувач.

4. CategoryController — це REST контролер, відповідальний за обробку операцій, пов’язаних із категоріями в системі. Кінцеві точки:

URL: /categories

Метод: GET

Опис: отримує список усіх категорій, доступних у системі.

5. InventoryController — це контролер REST, який керує інвентаризацією продуктів, дозволяючи виконувати операції CRUD користувачам із роллю «STAFF». Кінцеві точки:

URL: /inventory

Метод: POST

Опис: додає нові деталі асортименту для продукту.

URL: / inventory

Метод: GET

Опис: отримує список усіх предметів інвентарю.

URL: /inventory/{productId}

Метод: GET

Опис: отримує відомості про запаси для певного продукту на основі його ідентифікатора.

URL: / inventory

Метод: PUT

Опис: оновлює кількість кількох продуктів в інвентарі.

6. AddressController — це REST контролер, який обробляє операції, пов’язані з адресою користувача. Це дозволяє клієнтам і персоналу керувати адресами доставки, включаючи створення, отримання, оновлення та видалення адрес. Кінцеві точки:

URL: /delivery

Метод: GET

Опис: отримує список адрес, пов’язаних із автентифікованим користувачем. Доступний як клієнтам, так і персоналу.

URL: /delivery

Метод: POST

Опис: додає нову адресу доставки для автентифікованого користувача. Доступно лише клієнтам.

URL: /delivery/{addressId}

Метод: PUT

Опис: оновлює існуючу адресу, визначену ідентифікатором адреси. Доступно лише клієнтам.

URL: /delivery/{addressId}

Метод: DELETE

Опис: видаляє наявну адресу, визначену ідентифікатором адреси. Доступно лише клієнтам.

7. OrderController — це контролер REST, який обробляє операції, пов’язані із замовленнями клієнтів. Це дозволяє автентифікованим користувачам із ролями КЛІЄНТ і ПЕРСОНАЛ керувати замовленнями, зокрема переглядати, створювати та видаляти замовлення. Кінцеві точки:

URL: /order/all

Метод: GET

Опис: отримує список усіх замовлень, пов’язаних із автентифікованим користувачем.

URL: /order/{orderId}

Метод: GET

Опис: отримує деталі конкретного замовлення, визначеного ідентифікатором замовлення.

URL: /order/{addressId}

Метод: POST

Опис: створює нове замовлення для автентифікованого користувача з деталями замовлення та адресою доставки, визначеними за ідентифікатором адреси.

URL: /order/{orderId}

Метод: DELETE

Опис: видаляє існуюче замовлення, ідентифіковане ідентифікатором замовлення.

2 ПРОЕКТУВАННЯ І РОЗРОБКА СЕРВІСНОГО ШАРУ

Сервісний рівень – це важливий компонент в архітектурі REST API, особливо в багаторівневій або багаторівневій архітектурі, він є посередником між рівнем конролерів і рівнем доступу до даних (наприклад, сховищами або DAO). Основні цілі сервісного рівня:

– інкапсуляція бізнес-логіки: містить основну бізнес-логіку програми, рівень обробляє бізнес-правила та обчислення, необхідні для виконання вимог програми;

– сприяє багаторазовому використанню: завдяки централізації бізнес-логіки на рівні обслуговування цю логіку можна повторно використовувати в різних частинах програми, що дозволяє уникнути дублювання та спростити обслуговування;

– спрощує контролерів: сервісний рівень робить контролери лаконічними, вилучаючи з них бізнес-логіку. Контролери в першу чергу відповідають за обробку HTTP-запитів і відповідей, делегуючи бізнес-логіку на рівень обслуговування;

– управляє транзакціями: сервісний рівень часто керує транзакціями, гарантуючи, що бізнес-операції виконуються послідовним і надійним способом.

У застосунку розроблені наступні сервісні класи:

1. AuthService – сервіс автентифікації та керування користувачами в програмі, інкапсулює різні функції, пов’язані з реєстрацією користувачів, автентифікацією, активацією облікового запису, керуванням паролями та генерацією токенів:

– реєстрація користувача (метод реєстрації): обробляє реєстрацію користувача шляхом кодування пароля, призначення ролей і збереження інформації про користувача в базі даних. Він також надсилає електронний лист із підтвердженням зареєстрованому користувачеві для активації облікового запису;

– активація облікового запису (метод activateAccount): активує обліковий запис користувача шляхом перевірки токена активації. Якщо токен дійсний і термін його дії не минув, він оновлює статус користувача в базі даних на "увімкнено";

– керування паролями (метод changePassword): дозволяє користувачам безпечно змінювати свої паролі. Він перевіряє старий пароль, оновлює пароль новим і надсилає електронний лист для підтвердження;

– автентифікація (метод автентифікації): автентифікує користувачів шляхом перевірки їхніх облікових даних зі збереженими даними. Якщо автентифікація пройшла успішно, він генерує токен доступу за допомогою JWT (JSON Web Token) для подальшої авторизації;

– генерація токенів (метод generateToken): генерує токени JWT для автентифікації та авторизації користувачів. Ці токени містять заявки користувачів і використовуються для захисту кінцевих точок;

– надсилання листів (метод sendValidationEmail): надсилає електронні листи перевірки для запитів на активацію облікового запису та зміну пароля. Він використовує EmailService для асинхронного створення та надсилання електронних листів.

2. Клас StaffService надає функціональні можливості, спеціально розроблені для співробітників, що дозволяє їм керувати продуктами в системі:

– додавання продукту (метод addNewProduct): дозволяє співробітникам додавати нові продукти до системи. Він створює нову сутність продукту, використовуючи деталі, надані в об’єкті ProductBody. Якщо категорія продукту не існує, перед збереженням продукту створюється нова категорія;

– отримання продукту (метод getProductById): отримує продукт із бази даних на основі його унікального ідентифікатора (id). Якщо не знайдено жодного продукту з указаним ідентифікатором, створюється виняток під час виконання;

– оновлення продукту (метод updateProduct): оновлює деталі наявного продукту, ідентифікованого його ідентифікатором. Він отримує продукт із бази даних, оновлює його атрибути новими значеннями, наданими в ProductBody, і

зберігає зміни. Якщо вказаний ідентифікатор продукту не знайдено, створюється виняток під час виконання;

– видалення продукту (метод deleteProduct): видаляє продукт із системи на основі його ідентифікатора. Якщо продукт із вказаним ідентифікатором не знайдено, створюється виняток під час виконання.

3. Клас AddressService інкапсулює бізнес-логіку, пов’язану з керуванням адресами користувачів у програмі.:

– отримати адреси користувачів (метод getUserAddress): отримує список адрес, пов’язаних із певним користувачем. Він надсилає запит до AddressRepository, щоб знайти адреси, пов’язані з указаним AppUser;

– зберегти адресу (метод saveAddress): зберігає нову адресу для вказаного користувача. Він оновлює ім’я та прізвище користувача значеннями, наданими в AddressDTO, зберігає зміни в AppUserRepository, а потім створює нову сутність Address із деталями, наданими в AddressDTO. Нарешті, він зберігає нову адресу в базі даних через AddressRepository та повертає ідентифікатор новоствореної адреси;

– оновити адресу (метод updateAddress): оновлює наявну адресу за допомогою деталей, наданих у AddressDTO. Він отримує існуючу сутність адреси за її ідентифікатором за допомогою AddressRepository, оновлює її атрибути новими значеннями з AddressDTO та зберігає зміни в базі даних;

– видалити адресу (метод deleteAddress): видаляє адресу із системи на основі її ідентифікатора. Він безпосередньо викликає метод deleteById AddressRepository, щоб видалити адресу з бази даних;

4. Клас CategoryService забезпечує функції, пов’язані з керуванням категоріями в програмі. Отримати всі категорії (метод getAllCategories): отримує список усіх категорій, доступних у системі. Він делегує завдання CategoryRepository, який взаємодіє з основною базою даних, щоб отримати всі об’єкти категорії, що зберігаються в базі даних.

5. Клас ProductService інкапсулює функціональні можливості, пов’язані з керуванням продуктами в програмі. Отримати всі продукти (метод getAllProducts): отримує список усіх продуктів, доступних у системі. Він делегує завдання ProductRepository, який взаємодіє з основною базою даних, щоб отримати всі сутності продукту, що зберігаються в базі даних.

6. Клас InventoryService керує пов’язаними з інвентаризацією операціями в програмі:

– отримати інвентар для продукту (метод getInventoryForProduct): отримує інформацію про інвентар для певного продукту, ідентифікованого за його ідентифікатором. Він отримує продукт із ProductRepository, а потім отримує відповідний інвентар із InventoryRepository;

– отримати весь інвентар (метод getAllInventory): отримує список усіх інвентарних елементів, доступних у системі. Він отримує всі елементи інвентарю з InventoryRepository і зіставляє їх з об’єктами InventoryResponse, що містять ідентифікатори продуктів і кількість;

– додати інвентар (метод addInventory): додає новий інвентар для продукту. Спочатку він отримує продукт із ProductRepository на основі наданого ідентифікатора продукту. Потім він створює нову сутність запасів із зазначеною кількістю та пов’язує її з продуктом. Нарешті, він зберігає новий елемент інвентарю за допомогою InventoryRepository;

– оновити кількість продуктів (метод updateProductQuantities): оновлює кількість кількох продуктів в інвентарі. Він повторює список об’єктів InventoryResponse, що містять ідентифікатори продуктів і оновлені кількості. Для кожного продукту він отримує відповідну сутність продукту з ProductRepository та пов’язаний із ним інвентар із InventoryRepository. Потім він оновлює кількість елемента запасів і зберігає зміни.

7. Клас OrderService організовує операції, пов’язані з керуванням замовленнями в системі:

– отримати замовлення (метод getOrders): отримує список замовлень, пов’язаних із певним користувачем. Він отримує замовлення з WebOrderRepository на основі ідентифікатора користувачаж;

– отримати замовлення (метод getOrder): отримує конкретне замовлення, ідентифіковане його ідентифікатором. Він отримує замовлення з WebOrderRepository або створює виняток під час виконання, якщо замовлення не знайдено;

– додати замовлення (метод addOrder): додає нове замовлення до системи. Він створює нову сутність WebOrder із вказаним користувачем, адресою доставки та вмістом замовлення. Він переглядає список DTO вмісту замовлення, перевіряє наявність кожного продукту в інвентарі, вираховує замовлені кількості з інвентарю та створює сутності WebOrderContent, щоб пов’язати їх із замовленням. Крім того, він розраховує загальну вартість замовлення та надсилає користувачеві електронний лист із підтвердженням замовлення за допомогою EmailService;

– видалити замовлення (метод deleteOrder): видаляє певне замовлення, ідентифіковане його ідентифікатором. Він отримує замовлення з WebOrderRepository і видаляє його з бази даних.

3 ПРОЕКТУВАННЯ І РОЗРОБКА ШАРУ СХОВИЩА

У розробці веб-додатку було використано СУБД PostgreSQL. PostgreSQL — це потужна об’єктно-реляційна база даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші дані. Витоки PostgreSQL сягають 1986 року як частини проекту POSTGRES в Каліфорнійському університеті в Берклі та мають понад 35 років активного розвитку на основній платформі. PostgreSQL має багато функцій, які допомагають розробникам створювати додатки, адміністраторам — захищати цілісність даних і створювати відмовостійке середовище, а також допомагають вам керувати своїми даними, незалежно від того, наскільки великий чи малий набір даних. Крім того, що PostgreSQL є безкоштовним і відкритим вихідним кодом, він дуже розширюваний. Наприклад, ви можете визначати власні типи даних, створювати власні функції, навіть писати код з різних мов програмування без перекомпіляції бази даних [2]!

Клас AppUser представляє сутність бази даних, зіставлену з таблицею app_user. Поля:

– id: первинний ключ сутності. Створено автоматично за допомогою GenerationType.IDENTITY;

– username: унікальне ім'я користувача, не може бути дубльованим;

– password: хешований пароль користувача;

– firstName: ім'я користувача;

– lastName: прізвище користувача;

– email: адреса електронної пошти користувача, унікальний;

– enabled: позначка, яка вказує, чи ввімкнено обліковий запис користувача;

– accountLocked: позначка, що вказує, чи обліковий запис користувача заблоковано;

– address: зв’язок «один до багатьох» із сутністю «Адреса». Кожен користувач може мати декілька адрес;

– role: зв’язок «багато-до-багатьох» із сутністю «Role». Кожен користувач може мати декілька ролей$

– createdDate: позначка часу, що вказує на дату створення облікового запису користувача;

– lastModifiedDate: позначка часу, що вказує на дату останньої зміни облікового запису користувача.

Клас Address представляє сутність бази даних, відображену в таблиці адрес:

– id: первинний ключ сутності адреси, створено автоматично за допомогою GenerationType.IDENTITY;

– addressLine1: перший рядок адреси;

– addressLine2: другий рядок адреси;

– city: місто адреси;

– country: країна адреси;

– appUser: зв’язок «багато-до-одного» з сутністю AppUser. Кожна адреса належить одному користувачеві. Керується полем appUser в сутності AppUser.

Клас Product представляє сутність бази даних, зіставлену з таблицею product:

– id: первинний ключ сутності продукту, створено автоматично за допомогою GenerationType.IDENTITY.

– name: назва продукту;

– filePath: шлях до файлу зображення продукту;

– shortDescription: Короткий опис товару;

– longDescription: довгий опис продукту;

– brand: Марка продукту;

– price: Ціна товару;

– inventory: зв’язок «один-на-один» із сутністю «Inventory». Кожен продукт має відповідний інвентар. Керується полем продукту в сутності запасів;

– category: відношення «багато до одного» з сутністю «Category». Кожен продукт відноситься до однієї категорії.

Клас Category представляє сутність бази даних, зіставлену з таблицею категорій:

– id: первинний ключ сутності категорії, створено автоматично за допомогою GenerationType.IDENTITY$

– name: назва категорії, має бути унікальним.

Клас Inventory представляє сутність бази даних, зіставлену з таблицею інвентаризації:

– id: первинний ключ сутності інвентаризації, створено автоматично за допомогою GenerationType.IDENTITY;

– product: посилання на пов’язаний продукт. Це зв’язок «один-до-одного», коли кожен елемент інвентарю пов’язаний точно з одним продуктом.

– quantity: кількість товару, доступна в інвентарі.

Вищезазначені сутності представлені на ER-діаграмі сутностей (рис. 3.1).

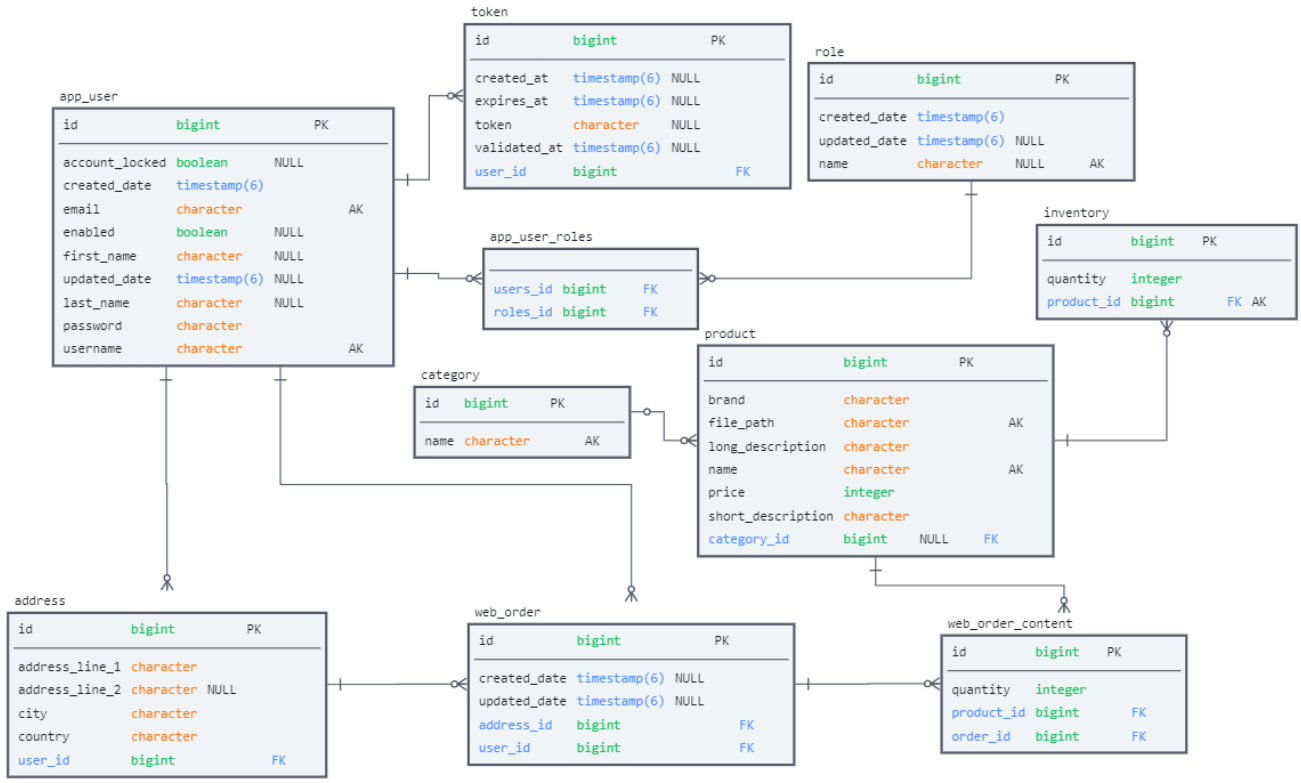

Рисунок 3.1 –ER-діаграма сутностей.

4 ПРОЕКТУВАННЯ І РОЗРОБКА ЖУРНАЛЮВАННЯ

The Simple Logging Facade for Java (SLF4J) служить простим фасадом або абстракцією для різних фреймворків журналювання, таких як java.util.logging, log4j 1.x, reload4j і logback. SLF4J дозволяє кінцевому користувачеві підключати потрібну структуру журналювання під час розгортання [3].

У застосунку журналювання налагоджується за допомогою аспектно-орієнтованого програмування та класом LoggingAspect у пакеті com.ecommerce.app.logging. Аспектно-орієнтоване програмування (AOP) — це парадигма програмування, яка покращує модульність, уможливлюючи відокремлення наскрізних проблем, таких як журналювання, безпека та керування транзакціями, від основної бізнес-логіки. AOP досягає цього поділу, визначаючи аспекти, які інкапсулюють ці проблеми, і вплітаючи їх у кодову базу в конкретних точках з’єднання, таких як виконання методів або обробка винятків. Використовуючи аспекти, розробники можуть застосовувати фрагменти коду, які можна багаторазово використовувати, у кількох частинах програми, не змінюючи безпосередньо вихідний код, що призводить до чистіших, модульніших і простіших у обслуговуванні систем. Завдяки таким концепціям, як точки з’єднання, поради, поінткути та плетіння, AOP надає потужний механізм для вирішення типових проблем у розробці програмного забезпечення, таких як виконання методів журналювання, дотримання політик безпеки та керування транзакціями, зрештою покращуючи якість коду та придатність до обслуговування.

Спосів роботи класу LoggingAspect:

1. Клас LoggingAspect використовує аспектно-орієнтоване програмування (AOP) для перехоплення викликів методів і застосування функцій журналювання.

2. Анотації, такі як @Around, використовуються для визначення pointcut, які вказують методи, які перехоплюються та реєструються. Ці анотації розміщуються в методах у класах, анотованих @LoggingController або @LoggingService.

3. Метод appLogger відповідає за журналювання викликів методів. Він отримує інформацію про клас, назву методу та аргументи методу за допомогою

відображення. Залежно від назви класу та методу, він вирішує, чи реєструвати виклик методу, чи продовжувати виконання початкового методу.

4. ObjectMapper використовується для серіалізації аргументів методу та результатів у рядки JSON для цілей журналювання.

5. Умовне журналювання застосовується на основі імені класу та імені методу. Наприклад, методи в класах AuthController і AuthService, а також метод registerStaff журналюються не вийнково для запобізання збереження особистих даних таких, як паролі та електронні адреси у журналі.

6. Для реєстрації використовується інтерфейс Logger від SLF4J. Різні рівні реєстрації (наприклад, info, warn, error) використовуються.

7. Винятки, створені під час виконання методу, не перехоплюються методом appLogger. Натомість їм дозволено поширюватися вгору по стеку викликів, де їх обробляє GlobalExceptionHandler.

8. Зареєстровані повідомлення записуються до файлів журналу, налаштованих у конфігурації журналу програми. Ці файли журналу фіксують записи про виклики методів і результати, допомагаючи в моніторингу програм, налагодженні та усуненні несправностей.

Журналювання застосовується за допомогою анотації @LoggingService, @LoggingController для усіх методів, окрім зазначених у виключенях.

5 ПРОЕКТУВАННЯ І РОЗРОБКА БЕЗПЕКИ

Клас AuthService відповідає за обробку операцій пов’язаних з автентифікацією користувачів:

1. Реєстрація користувача (метод реєстрації). Коли користувач реєструється, його пароль хешується за допомогою PasswordEncoder перед збереженням у базі даних. Токени підтвердження створюється та зберігається разом із даними користувача. Цей токен використовується для підтвердження електронної пошти для активації облікового запису користувача.

2. Підтвердження електронної пошти (метод sendValidationEmail). На адресу електронної пошти користувача надсилається електронний лист із посиланням для активації з токеном підтвердження. Посилання для активації містить унікальний токен, термін дії якого закінчується через 5 хвилин після чого, за необзідністю, надсилаються ще листи.

3. Аутентифікація (метод автентифікації). Автентифікація користувача виконується шляхом перевірки наданого імені користувача та пароля зі збереженими обліковими даними за допомогою AuthenticationManager. Після успішної автентифікації JWT (JSON Web Token) генерується за допомогою JWTService. Згенерований JWT повертається клієнту як частина відповіді на вхід.

4. Активація облікового запису (метод activateAccount). Коли користувач переходить по посиланню для активації в електронному листі, токен перевіряється, щоб переконатися, що термін його дії не минув. Якщо токен дійсний, обліковий запис користувача активується встановленням прапорця enabled на true.

5. Зміна пароля (метод changePassword). Коли користувач хоче змінити свій пароль, він повинен надати свій старий пароль для перевірки. Якщо старий пароль збігається з тим, що зберігається в базі даних, новий пароль хешується та оновлюється для користувача.

Клас WebSecurityConfig налаштовує параметри безпеки для веб-додатку за допомогою Spring Security:

1. SecurityFilterChain Bean – метод налаштовує ланцюжок фільтрів безпеки за допомогою HttpSecurity.

– .csrf(AbstractHttpConfigurer::disable) вимикає захист CSRF. Захист CSRF зазвичай непотрібний для API без збереження стану, захищених JWT;

– .authorizeHttpRequests(authorize -> { ... }) налаштовує правила авторизації на основі URL-адреси. URL-адреси, що відповідають певним шаблонам, дозволені без автентифікації (permitAll()), тоді як для всіх інших запитів потрібна автентифікація (authenticated());

– .sessionManagement(session -> session.sessionCreationPolicy (SessionCreationPolicy.STATELESS)) вказує, що сеанси не слід використовувати для зберігання стану автентифікації, оскільки натомість використовуються токени JWT;

– .authenticationProvider(authenticationProvider) встановлює спеціальний постачальник автентифікації (AuthenticationProvider) для обробки запитів автентифікації;

– .addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class) додає JWTRequestFilter перед UsernamePasswordAuthenticationFilter, дозволяючи виконувати автентифікацію на основі JWT перед іншими методами автентифікації.

3. Правила безпеки. Деякі кінцеві точки (/products/**, /register, /login тощо) дозволені без автентифікації (permitAll()), тоді як для інших потрібна автентифікація (authenticated()). Кінцеві точки, пов’язані з документацією API Swagger (/v3/api-docs, /swagger-ui/** тощо), також дозволені без автентифікації, щоб отримати доступ до документації API.

3. Автентифікація JWT. Аутентифікація на основі JWT реалізована за допомогою JWTRequestFilter, який перехоплює вхідні запити та витягує токени JWT для автентифікації.

Клас JWTRequestFilter відповідає за перехоплення вхідних HTTP-запитів і обробку токенів JWT для автентифікації:

2. Метод doFilterInternal - метод перевизначено з OncePerRequestFilter і в ньому реалізована основна логіка фільтра. Він перехоплює кожен вхідний HTTP-запит і обробляє токен JWT, що міститься в заголовку авторизації. Якщо шлях запиту містить /register, фільтр дозволяє проходити запит без автентифікації. Якщо запит не містить токена JWT або токен не починається з «Bearer», запит можна продовжити без автентифікації. Якщо дійсний токен JWT знайдено, фільтр отримує ім’я користувача з токена за допомогою JWTService та перевіряє, чи токен дійсний для користувача. Якщо токен дійсний, він створює об’єкт автентифікації (UsernamePasswordAuthenticationToken) і встановлює його в SecurityContextHolder, вказуючи, що користувача автентифіковано для поточного запиту. Нарешті, він викликає filterChain.doFilter(request, response), щоб дозволити запиту продовжити ланцюжок фільтрів для подальшої обробки.

3. Автентифікація без збереження стану. Фільтр умикає автентифікацію без стану, тобто він не покладається на серверне сховище сеансу для підтримки стану автентифікації.

4. Фільтр перехоплює кожен запит і витягує токен JWT із заголовка авторизації. Потім він перевіряє токен і встановлює контекст автентифікації, якщо токен дійсний. Це дозволяє наступним компонентам обробки запитів отримувати доступ до даних автентифікованого користувача через SecurityContextHolder.

Клас JWTService відповідає за створення та перевірку веб-токенів (JWT), які використовуються для автентифікації у вашій програмі:

1. Метод generateToken створює JWT для заданого об’єкта UserDetails із набором даних, створює токен за допомогою конструктора Jwts, встановлюючи претензії, тему (ім’я користувача), дату видачі, термін дії, повноваження та підписуючи токен секретним ключем.

2. Метод generateRefreshToken генерує токен оновлення з іншим часом закінчення. Цей токен можна використовувати для отримання нового токен доступу, не вимагаючи від користувача повторного входу.

3. Метод isTokenValid перевіряє, чи дійсний токен для наданого об’єкта UserDetails. Він перевіряє, чи ім’я користувача токена збігається з наданим ім’ям користувача та чи термін дії токена не минув.

4. Вилучення та декодування токенів. Метод getUsername витягує ім’я користувача з токена JWT. Метод getAllClaims аналізує та отримує всі претензії з токена JWT.

5. Метод getSignInKey декодує секретний ключ із Base64 і створює об’єкт Key для підпису та перевірки JWT.

Анотація @PreAuthorize є частиною функції безпеки Spring Security на рівні методів.У додатку @PreAuthorize використано для захисту контроллерів. Так, користувачі (звичайні відвідувачі веб-додатка) не зможуть додати нових товарів, видвлити їх або змінити кілкість на скаладі – це можуть робити користувачі з «STAFF» роллю.

6 РОЗГОРТАННЯ ДОДАТКА

Для розгортання клієнтської частини додатку використовується Vercel (рис. 6.1). Vercel — це платформа для розробників, яка надає інструменти, робочі процеси та інфраструктуру, необхідні для швидшого створення та розгортання веб-додатків без додаткової конфігурації. Vercel підтримує популярні фреймворки інтерфейсу, а його масштабована безпечна інфраструктура розподілена по всьому світу, щоб обслуговувати вміст із центрів обробки даних поблизу ваших користувачів із оптимальною швидкістю. Під час розробки Vercel надає інструменти для співпраці в реальному часі над вашими проектами, такі як автоматичний попередній перегляд і робочі середовища, а також коментарі до розгортання попереднього перегляду [4].

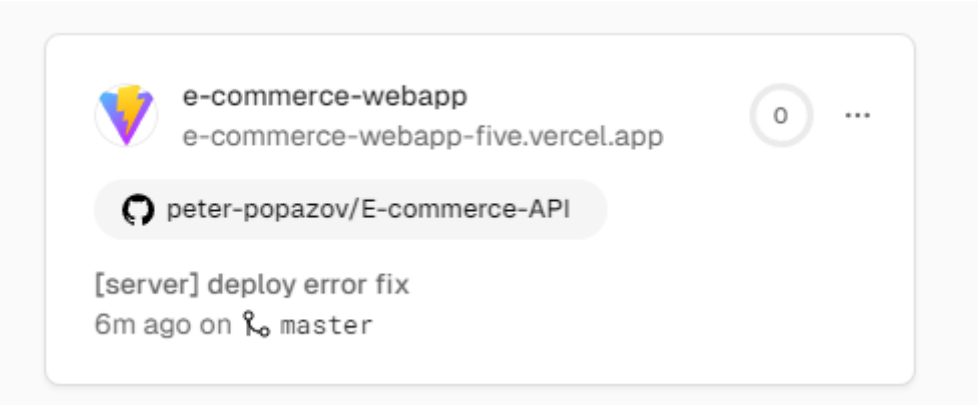

Рисунок 6.1 – Розгортання клієнтської частини додатку з Vercel

Серверна частина (з базою даних) додатку розгортається за допомогою Railway (рис. 6.1). Railway — це платформа розгортання, розроблена для оптимізації життєвого циклу розробки програмного забезпечення, починаючи з миттєвого розгортання та легкого масштабування, поширюючись на інтеграцію CI/CD і вбудовану можливість спостереження [5].

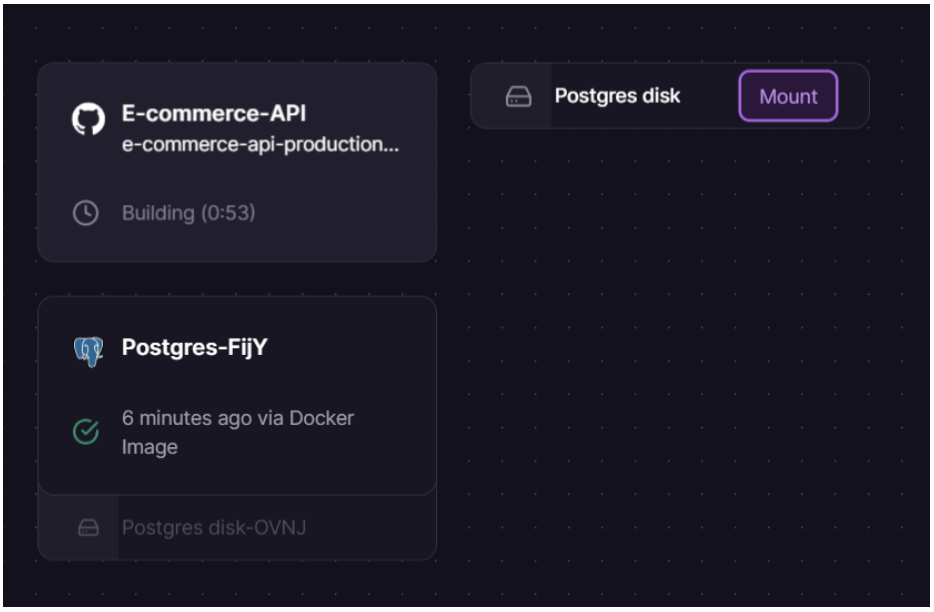

Рисунок 6.2 – Розгортання серверної частини додатку з Railway

Для налаштування роботи СУБД, сервісу з надсилання листів та встановлення даних для генерації токенів на сервері, використовуються змінні (рис. 6.3).

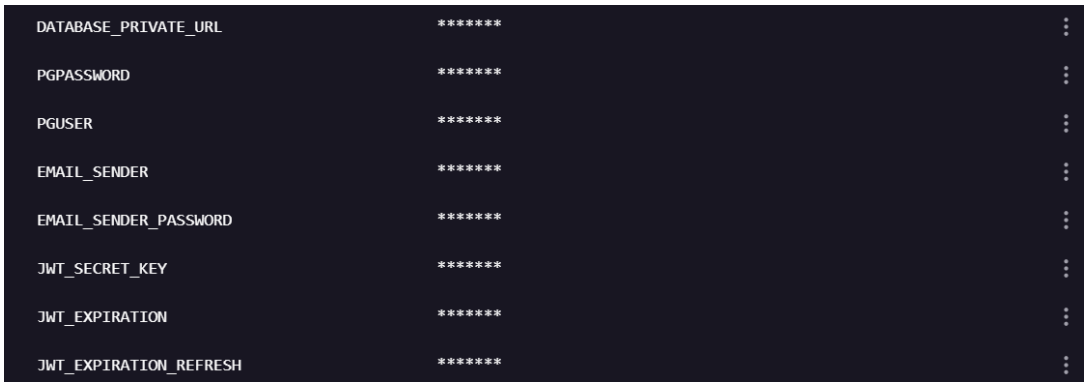

Рисунок 6.3 – Використання змінних у Railway

Посилання на працюючий додаток доступне у репозиторії.

Додана документація згенерована Swagger у репозиторій.

ВИСНОВКИ

Було успішно розроблено веб-додаток електронної комерції, який пропонує реєстрацію користувача з покроковою перевіркою вхідних даних, зрозумілим інтерфейсом для купівлі та комплексним керуванням кошиком для покупок. Користувачі можуть керувати своїми кошиками, вводити дані про доставку та отримувати електронні листи з підтвердженням замовлення. Сторона сервера підтримує основні кінцеві точки для реєстрації, автентифікації, виведення списку продуктів і категорій, розміщення замовлень і подання даних про доставку, при цьому деякі кінцеві точки доступні без токенів. Розроблено окремий контролер для робітників, який дозволяє їх додавати, змінювати продукти. Задіяні ролі користувачів та дозвіл ло окремих методів відповідним користувачам. Програма має надійну службу електронної пошти, глобальну обробку помилок і журнал аудиту системи. Використовуючи такі технології, як Spring Boot 3, Spring Security 6, JWT Token, PostgreSQL, Spring Data JPA, OpenAPI (Swagger) і React, програма забезпечує безпечні, зручні та ефективні покупки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація Spring Boot https://docs.spring.io/spring-framework/reference/index.html (дата звернення 07.06.2024).

2. Офіційна документація PostgreSQL https://www.postgresql.org/docs/ (дата звернення 07.06.2024).

3. Офіційна документація SLF4J https://www.slf4j.org/docs.html (дата звернення 07.06.2024).

4. Офіційна документація Vercel https://vercel.com/docs (дата звернення 07.06.2024).

5. Офіційна документація Railway https://docs.railway.app/quick-start (дата звернення 07.06.2024).