\WOOLF/

Applied Software Project Report

By

Sujit Menon

A Master's Project Report submitted to Scaler Neovarsity - Woolf in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

June, 2025

SCALER

Scaler Mentee Email ID: menonsahab@gmail.com
Thesis Supervisor: Naman Bhalla
Date of Submission: 29/06/2025

Certification

I confirm that I have overseen / reviewed this applied project and, in my judgment, it adheres to the appropriate standards of academic presentation. I believe it satisfactorily meets the criteria, in terms of both quality and breadth, to serve as an applied project report for the attainment of Master of Science in Computer Science degree. This applied project report has been submitted to Woolf and is deemed sufficient to fulfill the prerequisites for the Master of Science in Computer Science degree.

Naman Bhalla

……

Project Guide / Supervisor

DECLARATION

I confirm that this project report, submitted to fulfill the requirements for the Master of Science in Computer Science degree, completed by me from 5 Apr 2025 to 29 Jun 2025, is the result of my own individual endeavor. The Project has been made on my own under the guidance of my supervisor with proper acknowledgement and without plagiarism. Any contributions from external sources or individuals, including the use of AI tools, are appropriately acknowledged through citation. By making this declaration, I acknowledge that any violation of this statement constitutes academic misconduct. I understand that such misconduct may lead to expulsion from the program and/or disqualification from receiving the degree.

Sujit Menon **Date: 29 June 2025**

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my family, especially my better half who has always supported and encouraged me to never give up. I am also thankful to my children for their incredible support during this journey. Having a full-time job and pursuing a Master's degree are, in my opinion, extremely hard without the right support structure. During this course I couldn't spend as much time with my family as I would have wanted to, but they all understood and never complained. Their unwavering support is what kept me going through thick and thin.

My thanks also go to the many LLMs available today in the market which have democratized information for all in a much more accessible manner as compared to traditional search engines. In my personal experience they are a great starting place when you want to learn a topic. They are very helpful in pointing you in the right direction.

I am also grateful to Scaler for creating such an amazing course. I found it to be curated according to the ongoing industry trends. Even premier institutions in the country are following years old syllabi. So, seeing a curriculum such as that of Scaler is a breath of fresh air. What is equally great are the instructors, who I found to be very competent and knowledgeable.

I would be remiss not to mention my peers, who by virtue of our prolonged discussions on fundamentals, have played a significant role in my understanding of the subjects at hand. This report reflects the collective encouragement and support I have been lucky to receive and I am thankful to each one of them. Anything good in this project is credited to all those who have supported me in this journey. Any mistake is mine and mine alone.

Table of Contents

List of Tables

List of Figures

Applied Software Project

Abstract

This project aims to build an ecommerce website that streamlines the online shopping process and improves overall user satisfaction. It incorporates key features such as account management, product browsing, shopping cart functionality, order tracking, and secure payment systems. Utilizing modern tools and industry best practices, the platform is designed to offer a user-friendly and efficient interface while maintaining strong data protection and smooth operation.

Targeting real-world retail scenarios, the solution provides a scalable digital framework suitable for various business needs. It improves customer access through features like safe sign-up processes, customizable user profiles, and live updates on order status. The availability of multiple payment methods adds to user convenience, while secure login protocols protect sensitive information. In essence, this project showcases how digital solutions can transform conventional shopping methods, making them more streamlined and widely accessible.

Project Description

This project centers on the creation of a comprehensive ecommerce website designed to align with the demands of contemporary online shopping.

Objectives

- Offer an intuitive and accessible interface for customers to explore and purchase products.
- Equip businesses with tools to effectively manage inventory, orders, and financial transactions.
- Deliver a smooth, secure, and hassle-free shopping experience.

Relevance

As ecommerce continues to dominate the retail landscape, it offers businesses opportunities to expand their reach and customer base. This platform is crafted to

tackle common challenges—such as secure login systems, product discovery, and real-time order tracking—making it highly applicable to industries striving to strengthen their digital footprint.

Capstone Project Development Process

The development journey for the e-commerce application was divided into clearly defined stages to keep it organized, streamline efforts, and align outcomes with the intended objectives. Below is a breakdown of the main steps followed during the project timeline:

1. **Definition Phase**

This initial phase was dedicated to identifying project needs, setting clear targets, and establishing the platform's scope.

- **Understanding the Requirements:**
    - Drafted a comprehensive Product Requirements Document (PRD) covering both functional and non-functional aspects.
    - Outlined essential components such as User Management, Product Catalog, Cart & Checkout, Orders, Payments, and Authentication.
- **Setting Goals:**
    - Deliver a user-centric e-commerce platform with secure login, smooth checkout, and dependable payment handling.
    - Build with scalability in mind to accommodate growth in users and inventory.
- **Identifying Limitations:**
    - Defined the development schedule.
    - Evaluated available tools and technologies like Java, Spring Boot, and SQL-based databases.

2. Planning Phase

This stage included charting out the workflow and decomposing the project into achievable deliverables.

- **Tech Stack Selection:**
    - Backend: Java with Spring Boot for RESTful APIs.
    - Database: MySQL for relational data handling.
- **Milestone Planning:**
    - Phase 1: Implement User Management features.
    - Phase 2: Develop Product Catalog with search capability.
    - Phase 3: Complete Cart and Checkout features.
    - Phase 4: Integrate Order and Payment functionalities.
    - Phase 5: Finalize Authentication and conduct comprehensive testing.
- **Team and Workflow Management:**
    - Created sprint plans for breaking down tasks.
    - Leveraged tools like Jira for tracking and progress monitoring.
- **System Design:**
    - Crafted relational schemas for major tables (users, products, orders).
    - Created class diagrams for various modules.

3. Development Phase

This phase focused on translating planned designs into working features.

- **Backend Implementation:**

    - Created REST APIs for user registration, login, and profile editing.
    - Built endpoints for browsing, filtering, and viewing product details.
    - Developed modules for cart functionality and order creation.
    - Set up secure payment API endpoints integrated with gateways.
- **Authentication and Security:**
    - Enabled login and session control using JWT tokens.
    - Applied password encryption via BCrypt.
- **Database Setup:**
    - Created schema and tables for all relevant entities.
    - Ensured performance with optimized SQL queries.
- **Testing Procedures:**
    - Wrote unit tests for each module.
    - Carried out integration and full system tests.
    - Fixed identified bugs and ensured adherence to specifications.

4. Delivery Phase

The final stage focused on launching and delivering a polished product.

- **Deployment:**
    - Hosted the application using cloud providers such as AWS or Azure.
    - Utilized Docker for environment consistency during deployment.
    - Set up CI/CD pipelines for streamlined testing and deployment automation.

Requirement Gathering

Table 1.01

1. Functional Requirements:

Key functionalities identified include:

- **User Management**: Secure user sign-up, profile editing, and session tracking.
- **Product Catalog**: Users can explore products by categories, see product details, and perform keyword-based searches.
- **Cart and Checkout**: Ability to add products to the cart, review them, and proceed with placing orders.
- **Order Management**: Access to order history, delivery tracking, and receiving confirmations.
- **Payment Gateway**: Integration of diverse payment options with secured transaction support

Product Requirements Document (PRD) for E-Commerce Platform

1. **User Management**
    - Registration: Users can register using email or social logins.
    - Login: Secure login using user credentials.
    - Profile Management: Ability to view and edit profile details.
    - Password Reset: Option to reset password via a secured email link.
2. **Product Catalog**
    - Browsing: Navigate products by various categories.
    - Product Details: Comprehensive information including images, specs, and descriptions.
    - Search: Keyword-based product search functionality.

3. **Cart & Checkout**
    - Add to Cart: Functionality to add items to the shopping cart.
    - Cart Review: Display item details, price, and total amount.
    - Checkout: Simple flow for completing the purchase with address and payment options.
4. **Order Management**
    - Order Confirmation: Confirmation message sent post-purchase.
    - Order History: Ability to see all previous orders.
    - Order Tracking: Track delivery status in real-time.
5. **Payment**
    - Payment Methods: Support for multiple options like cards and net banking.
    - Secure Payments: Emphasis on secure transactions.
    - Payment Receipt: Digital receipt after each transaction.
6. **Authentication**
    - Secure Login: Protect user data during authentication.
    - Session Management: Maintain login state until logout or session timeout.

2. Non-Functional Requirements

The project was built using the MVC (Model-View-Controller) paradigm, omitting the View layer to concentrate solely on backend development. With Spring Boot as the base, the backend ensures modularity, scalability, and structured service-driven code.

Modular Monolithic Architecture

1. **Structured Monolith** Though monolithic in nature, the project follows a modular setup similar to microservices. Each core module—User Management, Catalog, Cart, Orders, Payments—contains:
    - Model: For database entity mapping and business objects.
    - Service: Contains business logic.
    - Controller: Handles API endpoints and routing
2. **Focus on Scalability** Designed such that any individual module (e.g., Payments) can be later transitioned into a microservice without major

refactoring. This approach provides flexibility for scaling without requiring much restructuring of the codebase.

MVC Without View Layer

- **Model**: It represents the application's data structure. It uses JPA for database communication.
- **Service**: Manages core business logic. Ensures separation of functionalities to promote easy maintenance and testability.
- **Controller**: Interfaces with clients through REST APIs returning JSON, supporting integration with web/mobile frontends.

As the project omits the View layer, the primary focus is on delivering a robust backend API designed to be seamlessly consumed by various frontend or mobile applications, thereby enabling future integration with diverse client-side technologies.

Benefits of This Approach

1. **Scalability**: The modular monolithic architecture ensures scalability by allowing for the gradual migration of individual modules to microservices when necessary.
2. **Maintainability**: Each module maintains a clear separation of concerns, simplifying the process of updating or debugging specific components of the application.
3. **Reusability**: The service layer is architected for reusability, allowing shared business logic to be leveraged across multiple modules within the application.
4. **Clean Code**: Adhering to Spring Boot best practices and MVC principles ensures a well-structured, maintainable, and easily extensible codebase.

Project folder structure – Below is the Project Folder Structure Based on the MVC Pattern

```
onlineshop
├── docs                    # Project documentation and supporting files
│   ├── api-docs            # API specifications (e.g., Swagger/OpenAPI)
│   ├── class-diagram       # UML or structural diagrams
│   ├── schema-diagram      # Database schema diagrams
│   └── postman-collection  # Postman API collection for testing
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com.onlineshop.shop
│   │   │       ├── common
│   │   │       │   ├── dtos
│   │   │       │   ├── exceptions
│   │   │       │   ├── models
│   │   │       ├── config
│   │   │       │   ├── GlobalConfig
│   │   │       │   ├── WebConfig
│   │   │       ├── data
│   │   │       │   ├── DataInitializer
│   │   │       │   ├── RoleRepository
│   │   │       ├── cartAndCheckout
│   │   │       │   ├── controllers
```

```
│   │   │       │   ├── dtos
│   │   │       │   ├── exceptions
│   │   │       │   ├── models
│   │   │       │   ├── repositories
│   │   │       │   └── services
│   │   │       ├── Auth
│   │   │       ├── product
│   │   │       ├── order
│   │   │       ├── user
│   │   └── Application.java    # Entry point of the Spring Boot application
│   └── resources
│       ├── application.properties
│       ├── templates
│       └── index.html
└── pom.xml                 # Maven build configuration
```

3. Stakeholder Analysis

Introduction

Stakeholder analysis was central to the development process. Engagements with end users, business operators, and other participants helped uncover critical needs and goals, ensuring alignment between system design and real-world expectations.

Steps in Stakeholder Analysis

1. **Stakeholder Identification**

   The initial step was to identify all key stakeholders who would interact with or gain value from the e-commerce platform. The primary stakeholders identified included:

   - Potential Users: Individuals who will browse products, make purchases, and manage their orders through the platform.
   - Business Owners: Vendors and platform administrators overseeing product listings, sales operations, and overall system management.
   - Technical Stakeholders: Developers and IT teams responsible for building, maintaining, and supporting the platform's technical infrastructure.
   - Logistics and Delivery Partners: Third-party providers responsible for order fulfillment, shipping, and ensuring timely delivery to customers.

Key Insights from Stakeholder Inputs

Issues faced

- **By Users**
  - Limited product discoverability caused by inadequate search and filtering capabilities.
  - Lack of user trust stemming from unreliable payment options and insecure transaction processes.
  - User frustration caused by slow updates or lack of clarity in delivery tracking systems.

- **By Business Operators**
    - Difficulty in efficiently managing product catalogs, particularly when handling large inventories.
    - Insufficient tools for effectively tracking and analyzing sales performance metrics.
    - Elevated operational overhead resulting from inefficient order processing and payment management systems.

Expectations

- **From Users**
    - A smooth and user-friendly shopping experience, enhanced by robust product search, personalized recommendations, and comprehensive product descriptions.
    - Secure and reliable payment options coupled with clear, transparent pricing.
    - Real-time order tracking with prompt and informative delivery notifications.
- **From Business Owners**
    - Intuitive interfaces for efficiently managing products, orders, and payment processes.
    - Actionable insights into sales trends and customer preferences through integrated analytics tools.
    - Scalable infrastructure designed to handle high traffic volumes during peak periods such as sales and festive seasons.

Actions taken based on stakeholder feedback

- **For Users**

    1. **Improved Product Navigation:**

        - Implemented robust search functionality with keyword matching and category-based filtering.
        - Introduced features such as personalized product recommendations and comprehensive product descriptions to improve user engagement.

    2. **Payment Security:**

        - Integrated diverse payment methods through secure gateways to build user trust and ensure transaction safety.
        - Enabled instant payment confirmations and automated issuance of digital receipts for transparency and record-keeping.

    3. **Order Tracking:**

        - Developed a real-time order tracking feature with live status updates and estimated delivery timelines.

- **For Business Operators**

    1. **Catalog Management:**

        - Designed an intuitive interface for adding, updating, and organizing products by category, streamlining catalog operations.

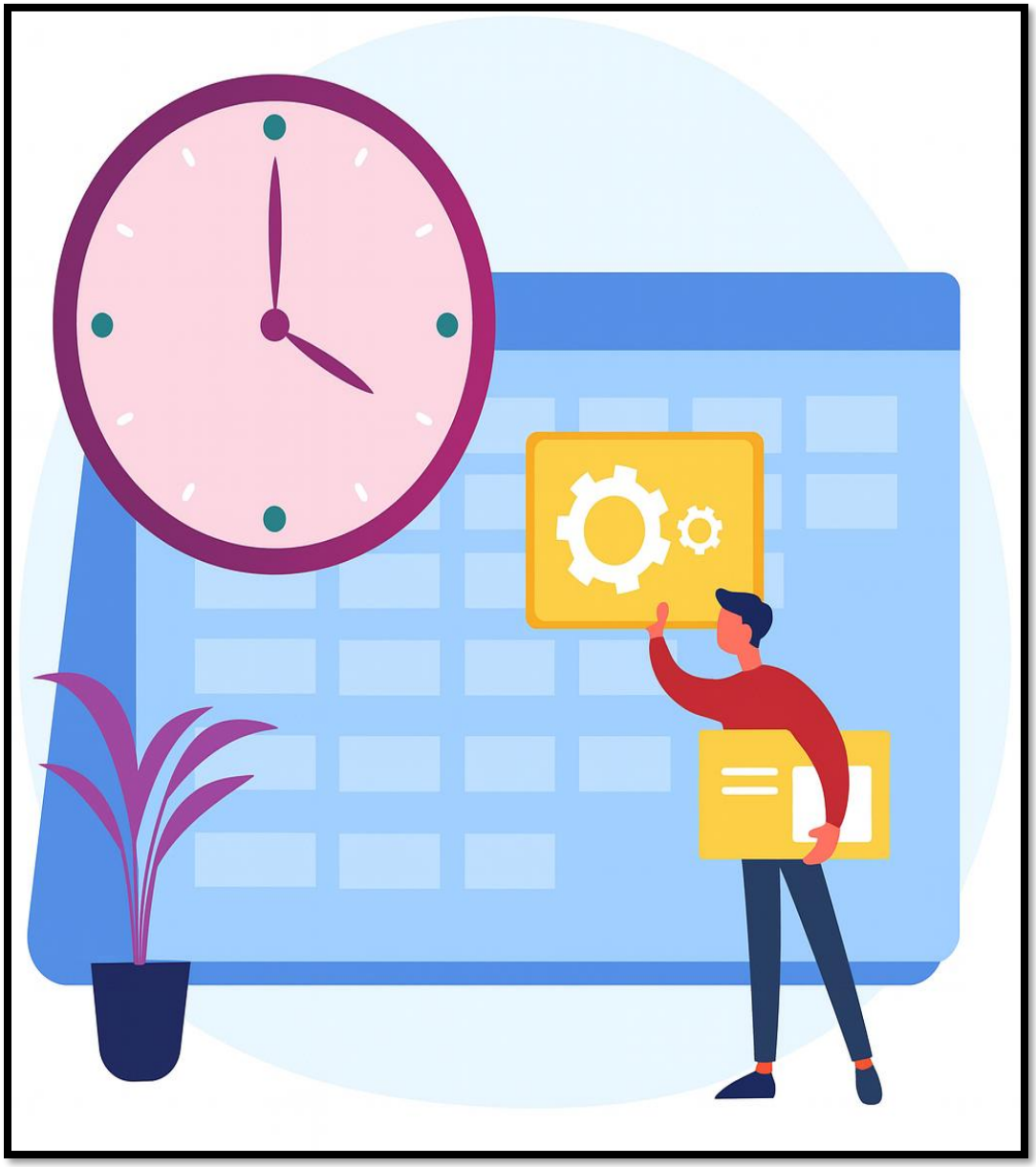
    2. **Order and Payment Modules:**

        - Built modules to simplify order processing, automate payment reconciliation, and monitor customer transactions efficiently.

3. **Scalability Enhancements:**

    - Implemented a modular system architecture that supports high traffic volumes and allows seamless feature expansion as business needs evolve.

4. Planning and Development



Project Plan

Developed a detailed project roadmap outlining key milestones, deadlines, and deliverables to ensure structured and timely execution.

Development Timeline and Sprint Planning

The project is structured into **five sprints** spanning **eight weeks**, with each sprint featuring well-defined milestones and deliverables to ensure steady progress and alignment with project goals.

Table 1.02

Sprint 1: Foundation and Setup (Week 1)

Goal: Establish the project's foundational structure.

Deliverables:

1. **Project Initialization**
    - Set up a new Spring Boot project using a modular monolithic architecture.
    - Configure Gradle/Maven for efficient dependency management.
    - Create base folder structures: *Model, Controller, Service, Repository*.

- Add initial configurations to support future database integration.

2. **Database Design**
    - Design a normalized relational database schema based on project requirements.
    - Define and create tables for: *Users, Roles, Products, Categories, Orders, Order Items, Carts, and Cart Items*.
3. **Environment Setup**
    - Configure local development and testing environments.
    - Integrate Swagger for interactive API documentation.

Milestones:

- Core project architecture established and database schema finalized.
- Basic Spring Boot application up and running successfully.

Sprint 2: User Management Module (Week 2–3)

Goal: Implement core user functionalities along with secure authentication mechanisms.

Deliverables:

1. **Model Layer**
    - Define entity classes for User, Role, and their associations.
    - Include essential fields such as name, email, password, and assigned roles.
2. **Service Layer**
    - Develop services for user registration, login, and profile management.

- Use Bcrypt for secure password hashing.
- Add features for password reset and updating user profiles.

3. **Controller Layer**
    - Build RESTful APIs for user registration, login, and profile updates.
    - Implement robust error handling for edge cases like duplicate emails or invalid credentials.
4. **Security Configuration**
    - Set up Spring Security to manage authentication and authorization.
    - Integrate JWT-based token handling for session-less, secure access control.

Milestones:

- Secure and fully functional user registration and login endpoints completed.
- Swagger documentation updated to reflect User Management APIs.

Sprint 3: Product Catalog Module (Week 4)

Goal: Implement features for product browsing, categorization, and detailed viewing.

Deliverables:

1. **Model Layer**
    - Define entities for Product, Category, and their relationships.
2. **Service Layer**
    - Implement functionality for adding products, retrieving product listings, and filtering by category.

3. **Controller Layer**
    - Build RESTful APIs for:
        - Product listing
        - Product details
        - Keyword-based search
4. **Data Seeding**
    - Populate the database with sample products and categories for testing and demo purposes.

Milestone:

- Fully functional and documented APIs for product catalog browsing, search, and filtering.

Sprint 4: Cart and Checkout Module (Week 5–6)

Goal: Enable users to manage their shopping carts and complete orders.

Deliverables:

1. **Model Layer**
    - Define entities: Cart, CartItem, and their relationships with User and Product
    - Define Order, OrderItem, and their associations with User and Product
2. **Service Layer**
    - Implement logic to add or remove items from the cart
    - Dynamically calculate cart totals
    - Develop order placement functionality
3. **Controller Layer**

- Create APIs for:
    - Cart management
    - Order placement
    - Order history retrieval

4. **Error Handling**
    - Handle edge cases such as:
        - Out-of-stock product scenarios
        - Invalid order requests

Milestones:

- Users can add items to their cart and successfully place orders
- Cart and order APIs are fully functional and documented

Sprint 5: Payment Integration and Final Touches (Week 7–8)

Goal: Enable secure payment processing and finalize all components of the project.

Deliverables:

1. **Payment Service**
    - Implement mock payment processing logic
    - Provide APIs supporting multiple payment methods (credit card, debit card, etc.)
2. **Integration**
    - Ensure successful order placement updates payment status
    - Trigger order confirmation notifications post-payment
3. **Testing**
    - Conduct end-to-end testing across all modules

- Handle edge cases such as expired JWT tokens and concurrent cart updates

4. **Documentation**
    - Update Swagger API documentation to reflect final implementation
    - Prepare comprehensive README files for deployment and usage instructions

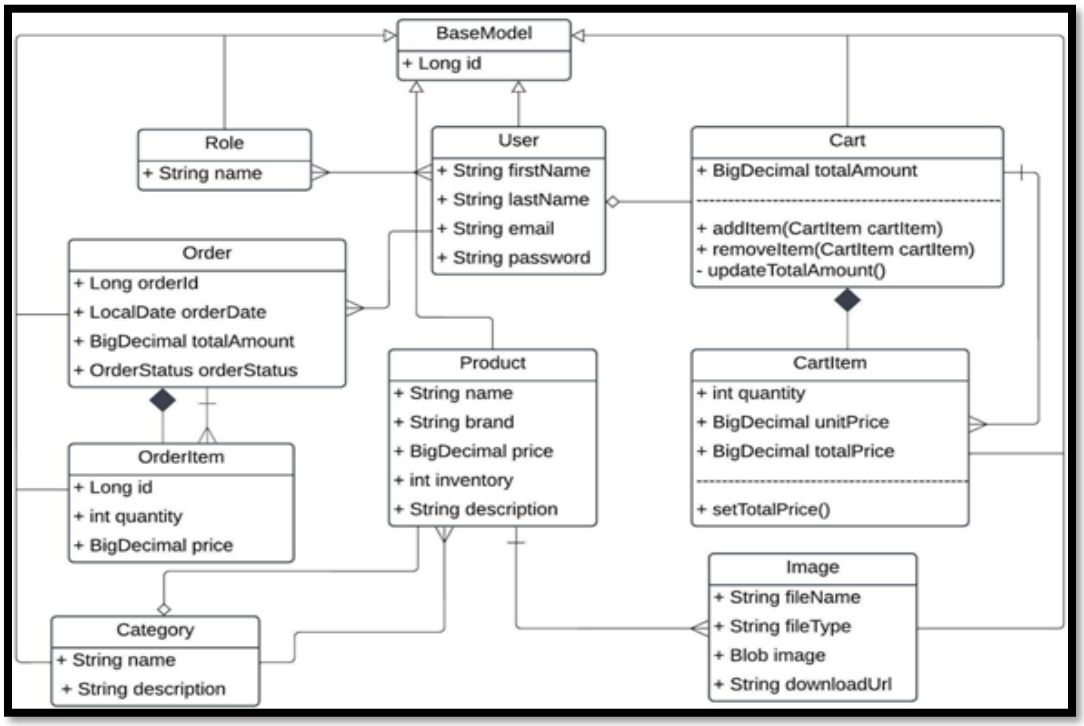
Milestones:

- Fully functional and integrated payment system
- Application thoroughly tested, documented, and ready for delivery

Class Diagram

Designed a class structure to model core entities including User, Product, Order, and Payment, ensuring clear relationships and data integrity.

Figure 1: Class Diagram



Relationships:

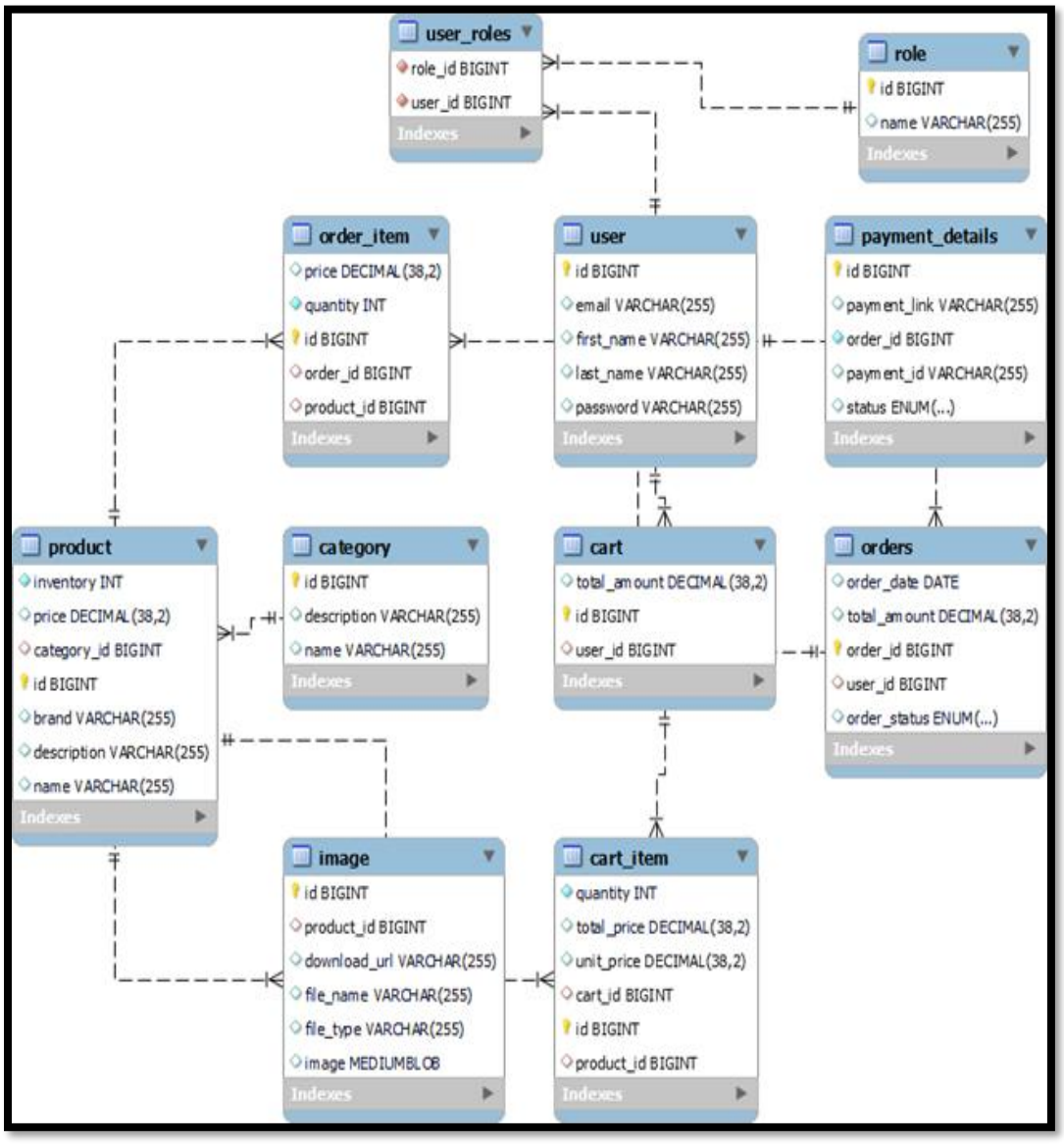
Table 1.03

Relationship table	
User-Role	Many-to-Many
User-Cart	One-to-One
User-Order	One-to-Many
Category-Product	One-to-Many
Product-Image	One-to-Many
Order-OrderItem	One-to-Many
OrderItem-Product	Many-to-One
Cart-CartItem	One-to-Many
CartItem-Product	Many-to-One

Database Schema Design

Developed a relational database schema to store user profiles, product details, order information, and payment records.

Figure 2: Database schema design



Foreign Keys:

- **user_roles(user_id)** refers **user(id)**
- **user_roles(role_id)** refers **role(id)**
- **order_item(order_id)** refers **orders(id)**
- **order_item(product_id)** refers **product(id)**
- **product(category_id)** refers **category(id)**
- **cart(user_id)** refers **user(id)**
- **cart_item(cart_id)** refers **cart(id)**
- **cart_item(product_id)** refers **product(id)**
- **image(product_id)** refers **product(id)**
- **orders(user_id)** refers **user(id)**
- **payment_details(order_id)** refers **orders(id)**

Cardinality of Relations:

- Between **user_roles** and **user** → **m:1**
- Between **user_roles** and **role** → **m:1**
- Between **order_item** and **orders** → **m:1**
- Between **order_item** and **product** → **m:1**
- Between **product** and **category** → **m:1**
- Between **cart** and **user** → **1:1**
- Between **cart_item** and **cart** → **m:1**
- Between **cart_item** and **product** → **m:1**
- Between **image** and **product** → **m:1**
- Between **orders** and **user** → **m:1**
- Between **payment_details** and **orders** → **1:1**

Unit Testing

Advantages of Unit Testing

- **Quality Assurance**: Helps detect defects early in development, leading to a higher-quality product.
- **Enhanced Reliability**: Verifies that each component functions correctly, boosting overall system reliability.
- **Time-Saving**: Minimizes the need for extensive debugging and issue resolution in later stages of development.
- **Better Code Maintainability**: Promotes writing modular, clean, and well-structured code that is easier to update and expand.

Scope

Each module—including user authentication, product catalog, cart operations, order processing, and payment services—was rigorously tested to ensure correctness, reliability, and adherence to expected behavior.

Testing Evidence

Screenshots of unit test results for key modules will be provided as proof of successful testing.

User Authentication: Verified registration, login, and role-based access control functionalities.

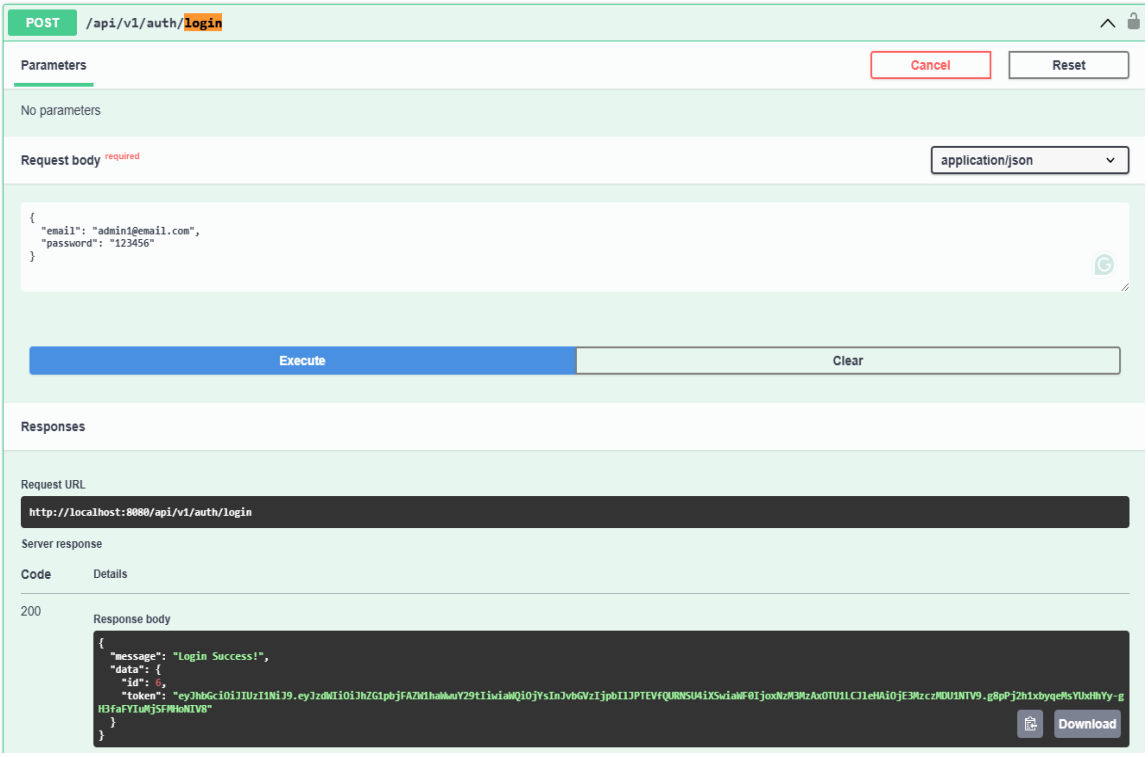


Figure 3: Login screen

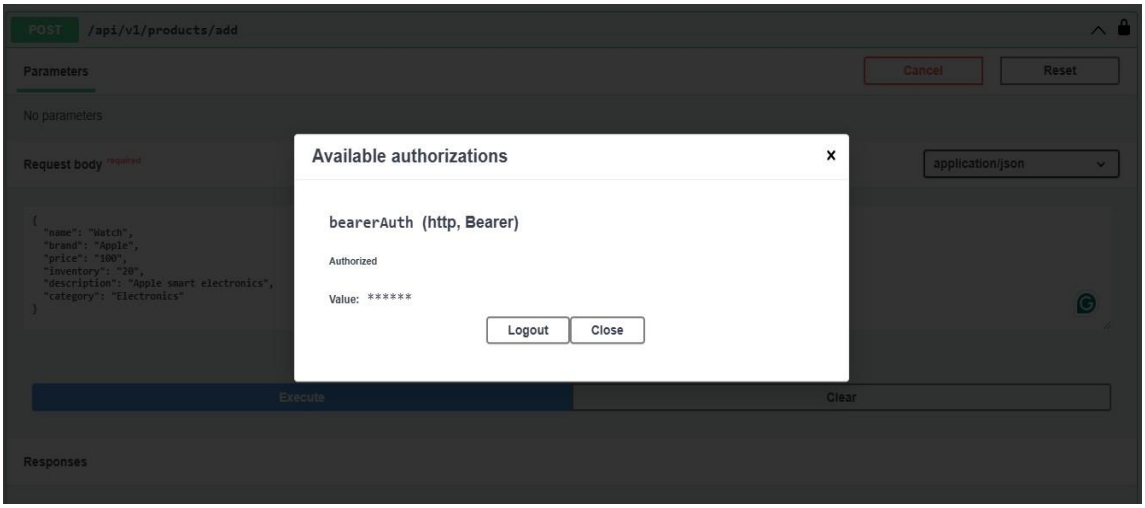


Figure 3.1: Auth token screen

Product Catalog:

- Tested addition, get all, and updating of products.
- Ensured proper functionality of product search by category.

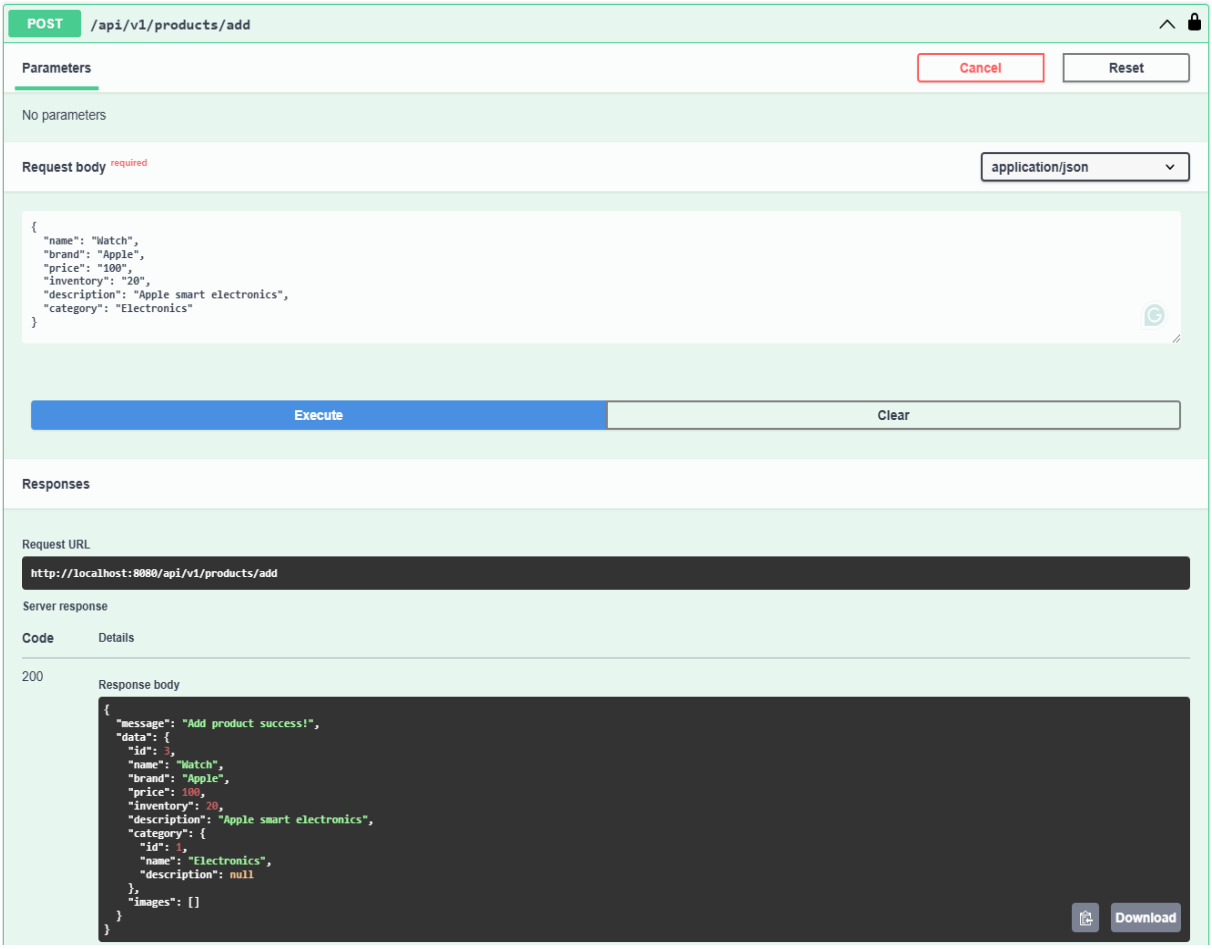


Figure 4: Add Product

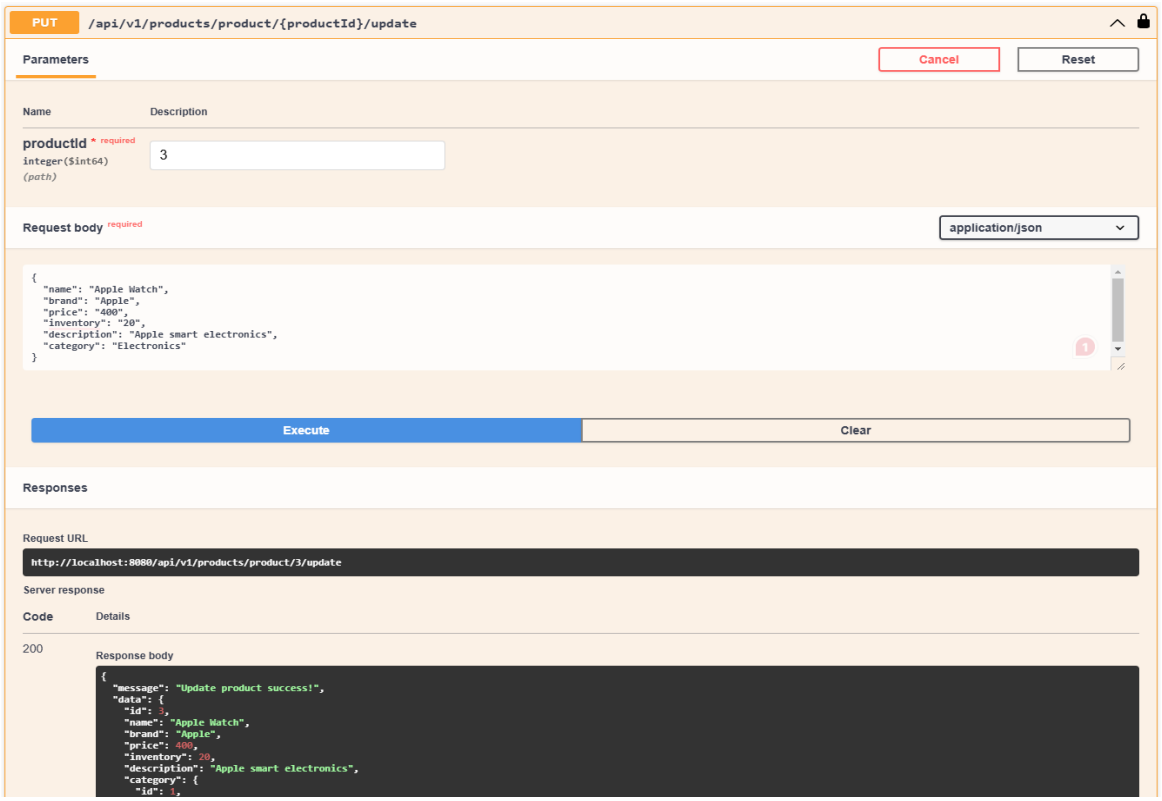


Figure 4.1: Update Product

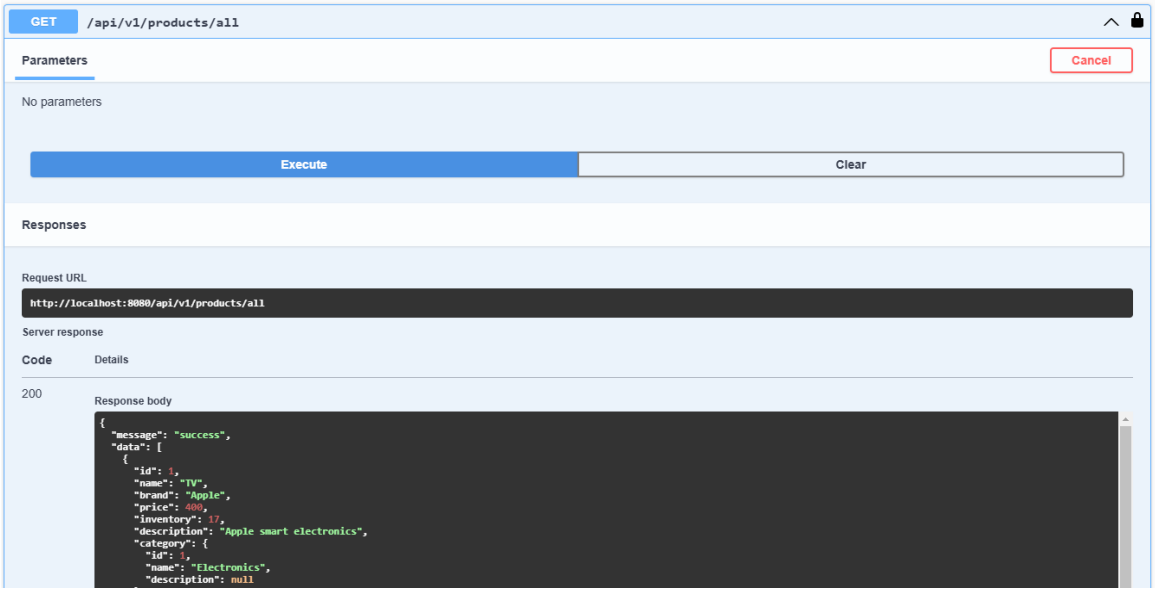


Figure 4.2: Get All Product

Cart Functionality:

- Validated the addition, removal, and updating of cart items.
- Checked accurate calculations for total amounts.

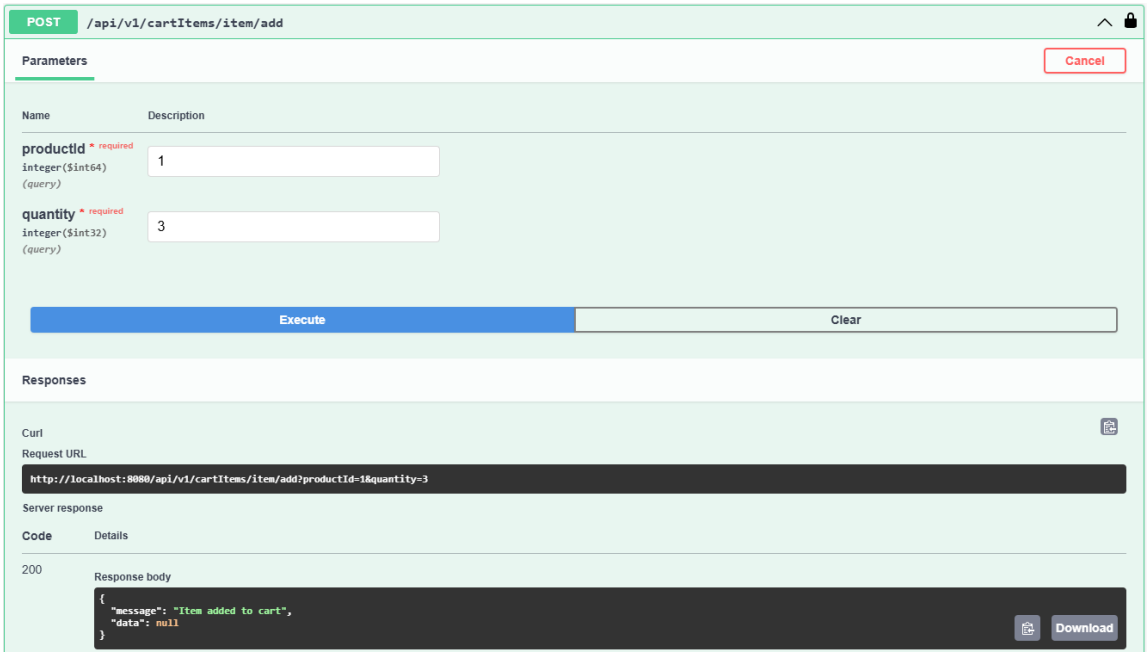


Figure 5: Add Item to Cart

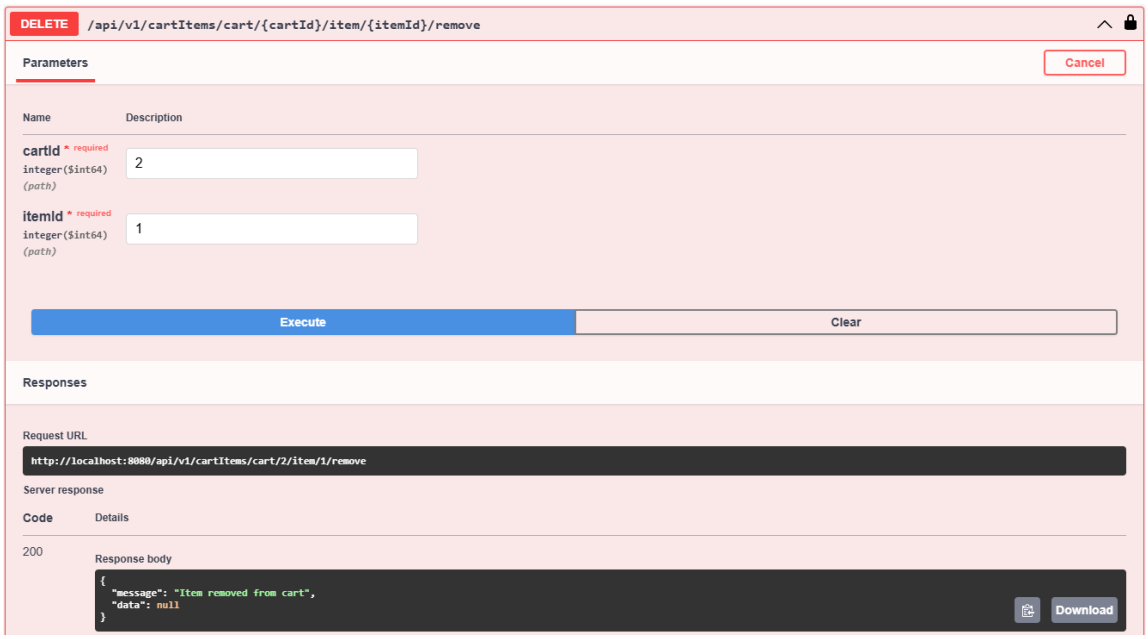


Figure 5.1: Remove Cart Item

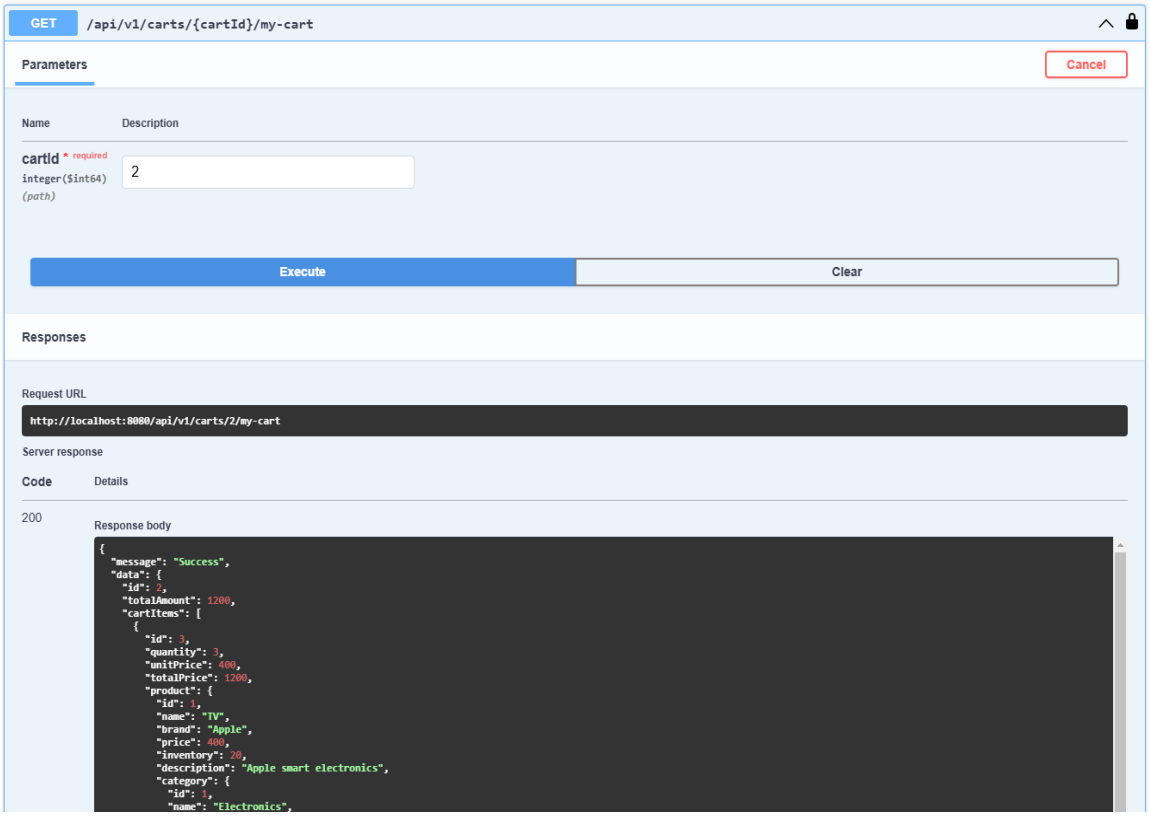


Figure 5.2: Show Cart

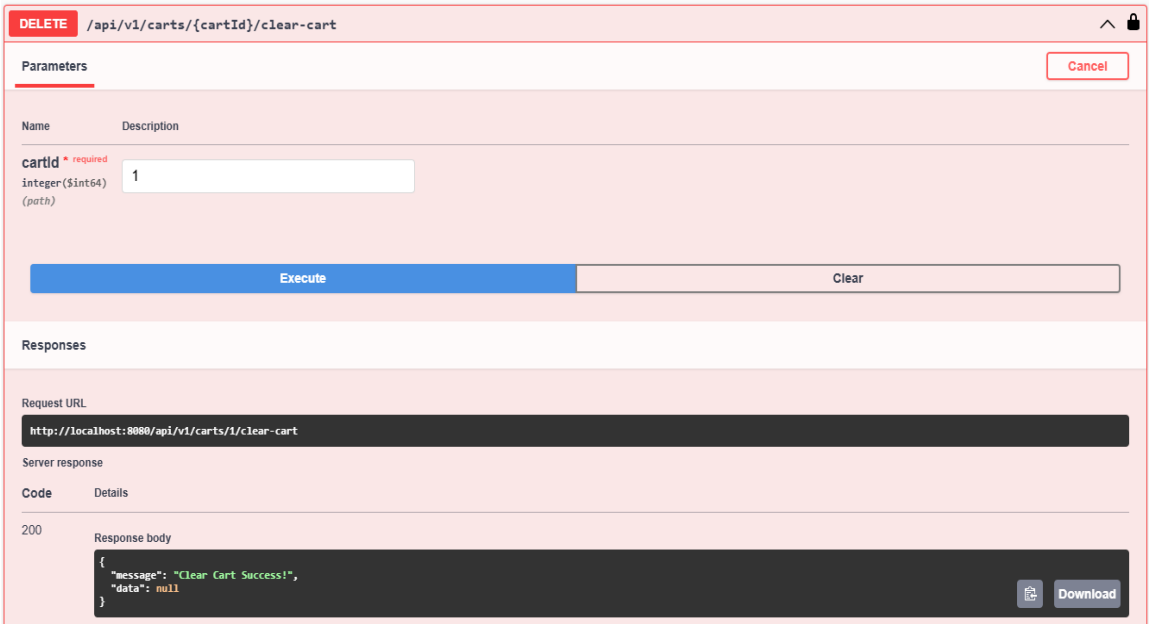


Figure 5.3: Delete Cart

Order Functionality:

- Tested placing orders.
- Verified fetching orders by user ID and order ID.

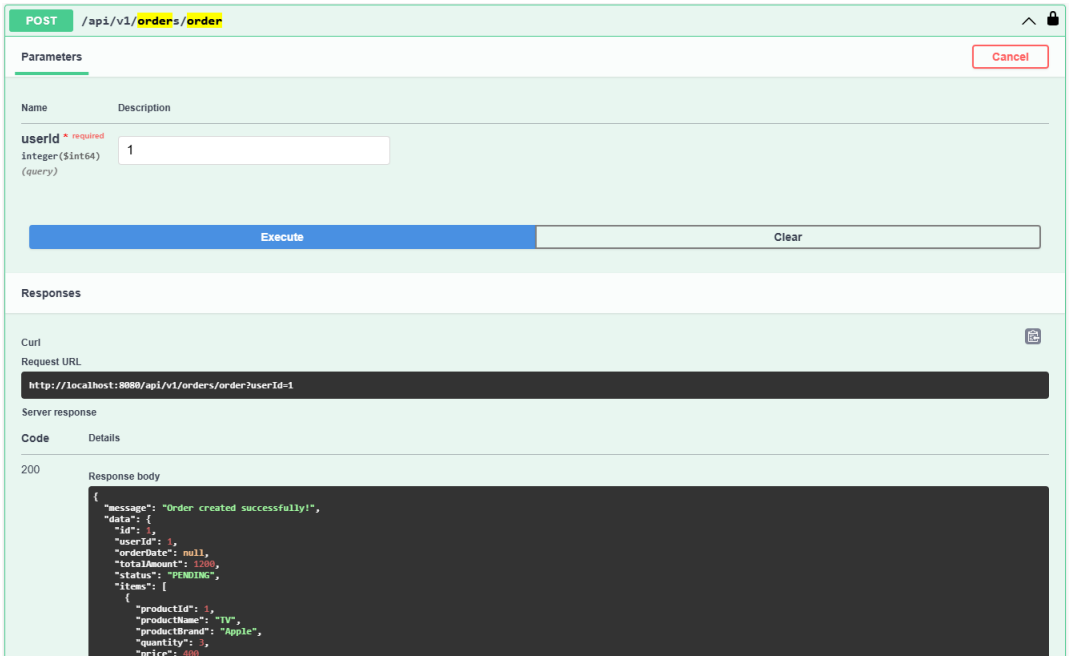


Figure 5.4: Place Order

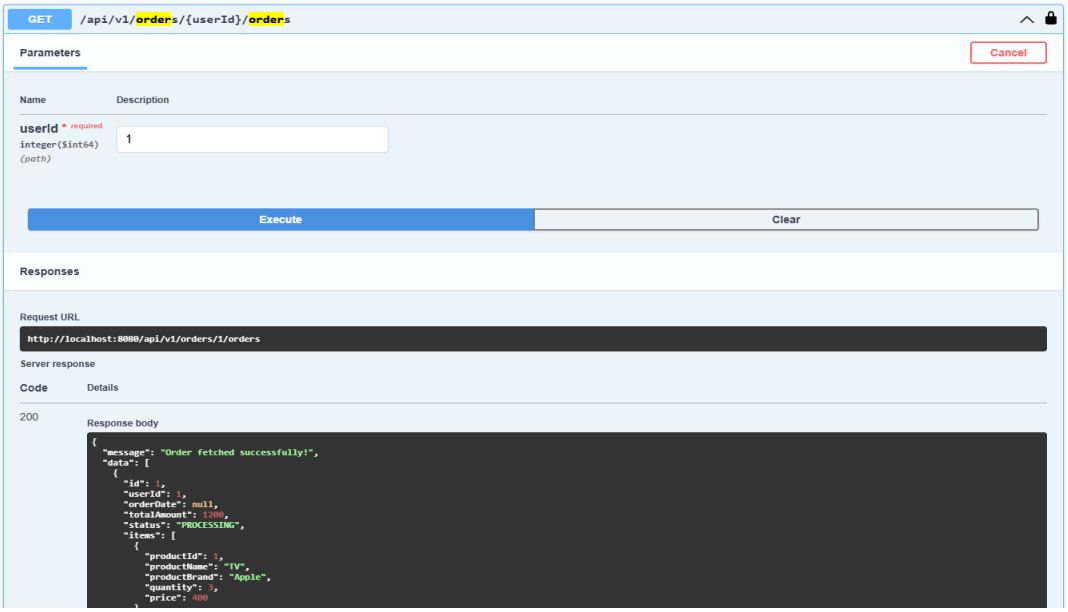


Figure 5.5: Show Order

Payment Functionality:

- Ensured the checkout process worked correctly after order placement.

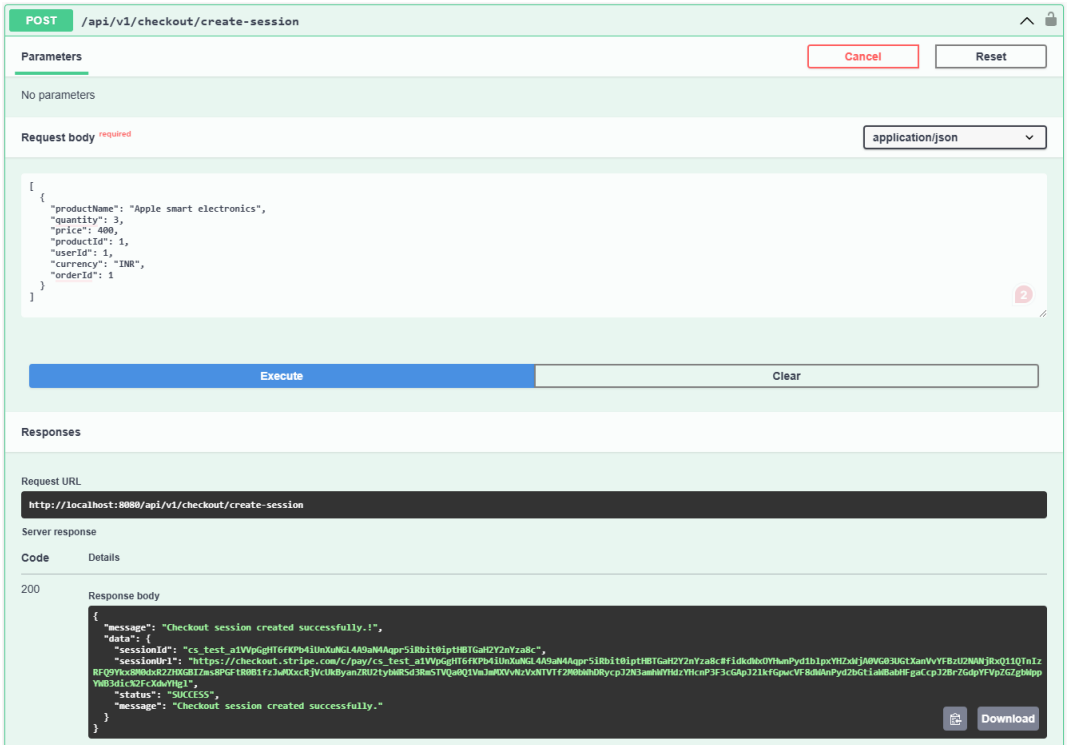


Figure 6: Create session

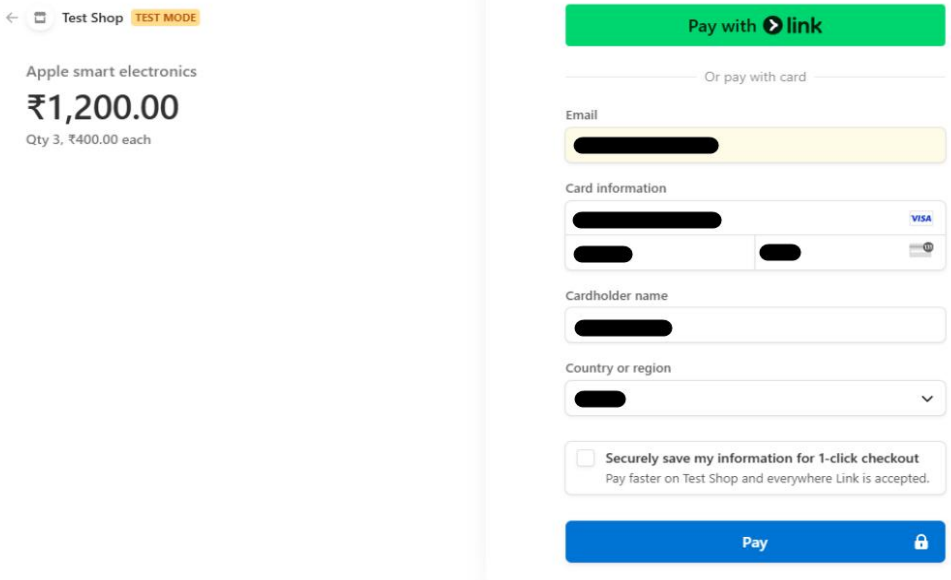


Figure 6.1: Payment

Feature Development Process

Focus Area: Cart, Order, and Payment Functionality

The following steps outline the complete workflow for implementing cart, order, and payment features within the project:

Step 1: User Authentication

API Endpoint: /auth/login

- **Description**:
    - Users must be authenticated before accessing cart functionalities.
    - The user submits login credentials (email and password).
    - On successful authentication, the system issues a token, granting access to authorized services.

- **Authentication Flow**

1. The user initiates a login request by submitting their email and password.
2. The system verifies the provided credentials:
    - **If the credentials are valid**:
        - A token (e.g., JWT) is generated and returned to the user.
        - This token is required for all subsequent API interactions.
        - In Swagger, the token should be added directly in the Authorization header *without* the 'Bearer' prefix.
    - **If the credentials are invalid**:
        - An error response is sent.
        - The user is denied access to cart and related functionalities.
3. For all future requests, the user must include the token in the Authorization header to maintain authenticated access.

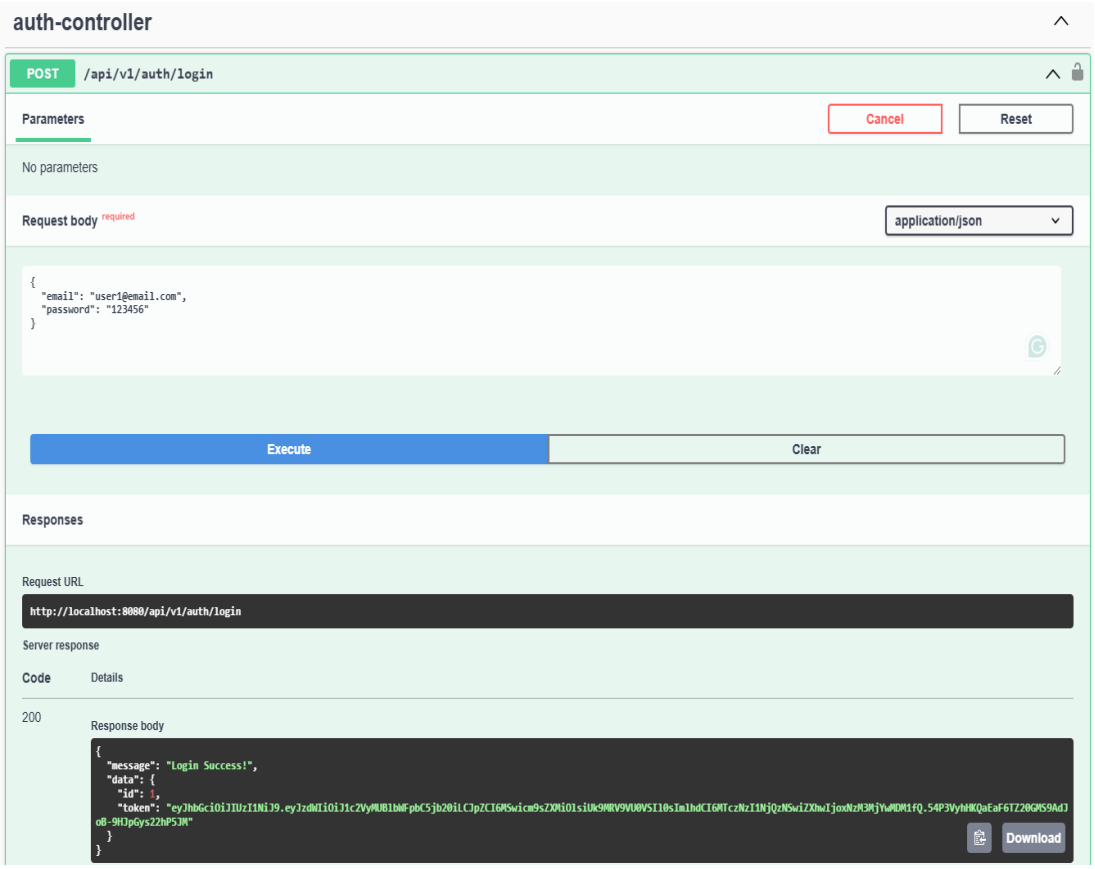


Figure 7: Login

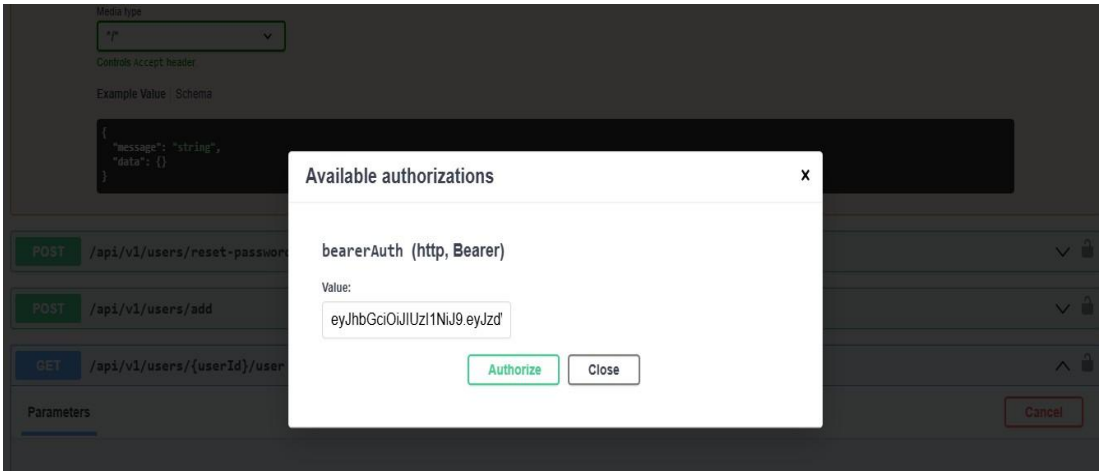


Figure 7.1: Add Token

Step 2: Add to Cart

API Endpoint: /cart/add

Description

- This endpoint allows users to add items to their cart.
- If the user does not have an existing cart, the system automatically creates one.
- If a cart already exists, the item is either added as a new entry or its quantity is updated if it's already present.

Flow

1. The user sends a request containing the **product ID** and **desired quantity**.
2. The system checks for an active cart linked to the user:
    - **No Active Cart**:
        - A new cart is created.
        - The product is added as the first cart item.
    - **Active Cart Exists**:
        - The system verifies if the product is already in the cart:
            - **Yes**: Updates the quantity of the existing item.
            - **No**: Adds the product as a new cart item.
3. A response is returned containing the updated cart information.

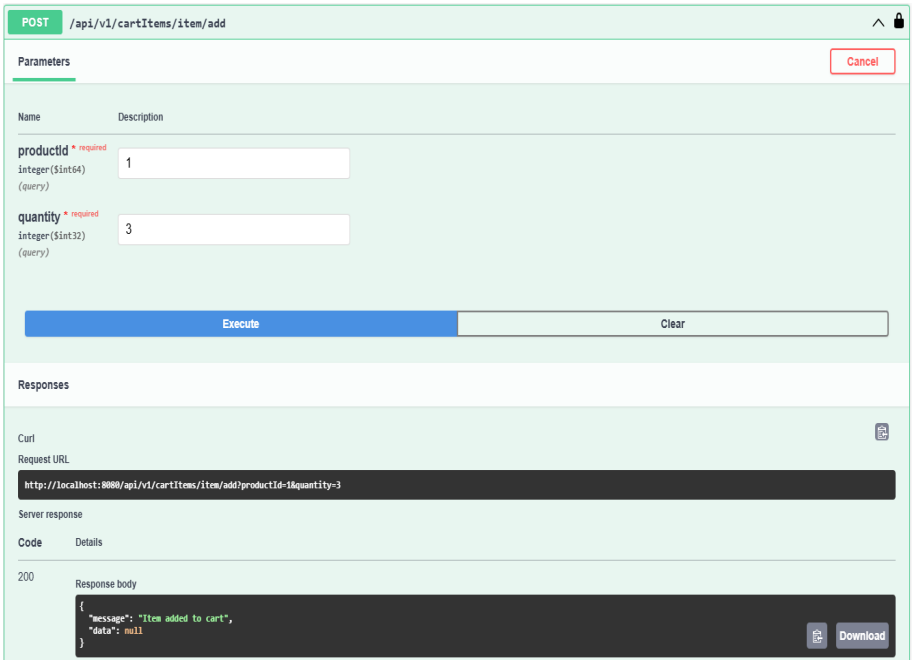


Figure 8: Add Cart Item

Step 3: Place Order

API Endpoint: /order/place

Description

- This endpoint is used to finalize a purchase by placing an order based on the current contents of the user's cart.
- Upon successful order placement:
  - The cart and its items are cleared.
  - A new order and its associated order items are created in the database.
  - The order is marked with a status of **Pending**.
  - Inventory levels are updated to reflect the purchased quantities.

Flow

1. The user sends a request to place an order.
2. The system performs validations on the cart:
   - Confirms that the cart is not empty.
   - Checks that each product has sufficient inventory.
3. If validations pass, a new order is created with:
   - A unique Order ID
   - Associated user information
   - Total order amount
   - Initial status set to **Pending**
   - All items from the cart are added as order items
4. Inventory updates are performed:
   - For every ordered item, the available stock is reduced accordingly.
5. The cart and its contents are cleared post-order creation.
6. A response is returned containing the order summary and details.

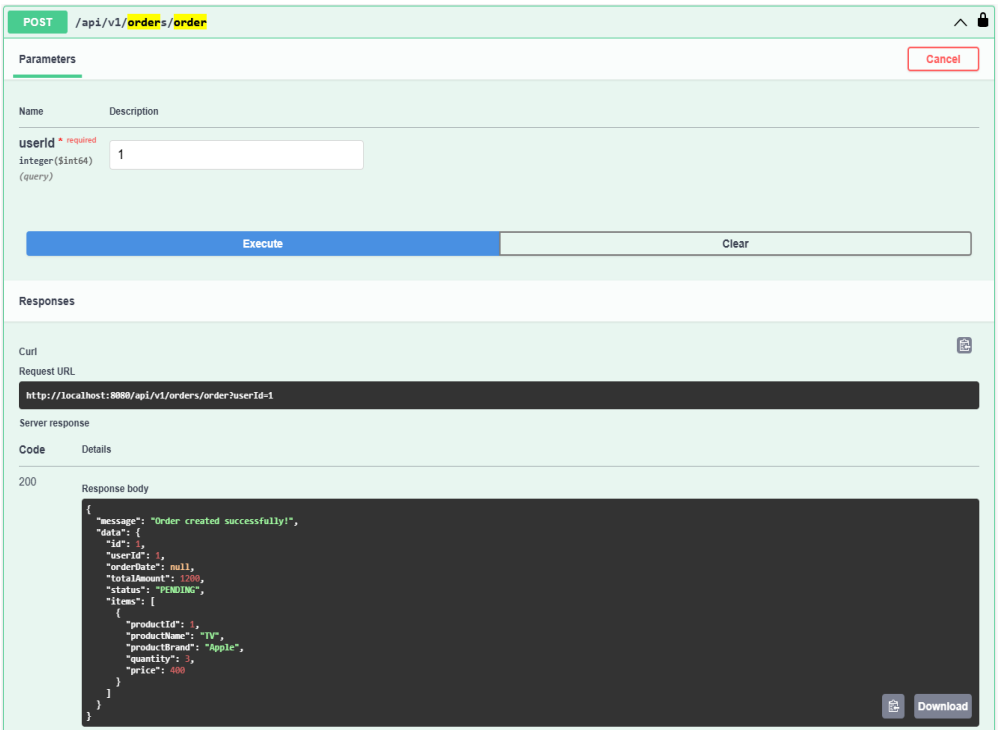


Figure 9: Place Order

Step 4: Checkout and Payment

API Endpoint: /checkout/create-session

Description

This endpoint initiates and manages the payment process for an existing order.

Upon processing the payment:

- If successful, the order status is updated to **Completed** (or **Processing** during intermediate states).
- If the payment fails, the order remains in **Pending** status, and the user is informed.

Flow

3. The user initiates a payment request by providing the necessary payment details for a previously placed order.
4. The system performs order validation:
    - Confirms that the order exists.
    - Verifies that the order is currently in **Pending** status.
    - Validates the submitted payment details.
5. Payment is processed:

- **On Success**:
    1. The order status is updated to **Processing**.
    2. A confirmation is sent to the user.
- **On Failure**:
    1. The order status remains **Pending**.
    2. The user receives a failure notification.

6. A response is returned containing the payment link and the current order status.

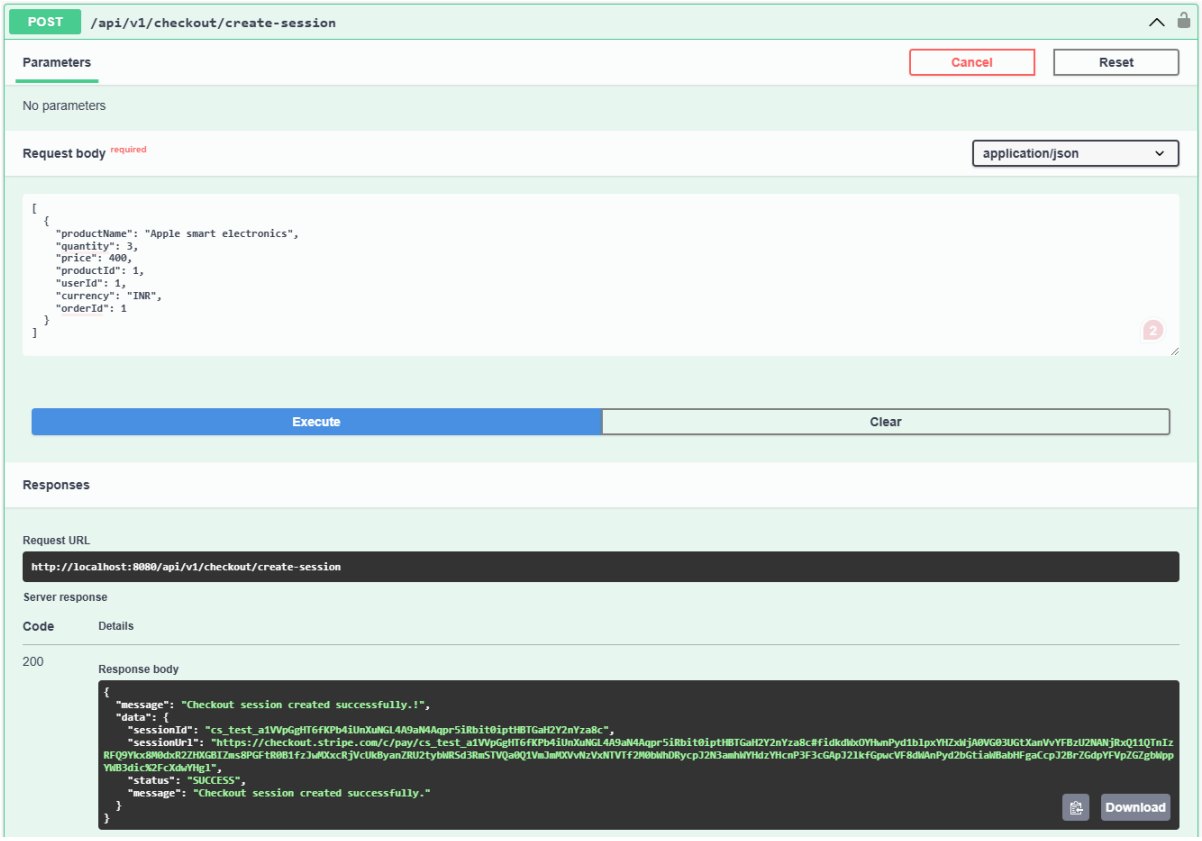


Figure 10: Checkout

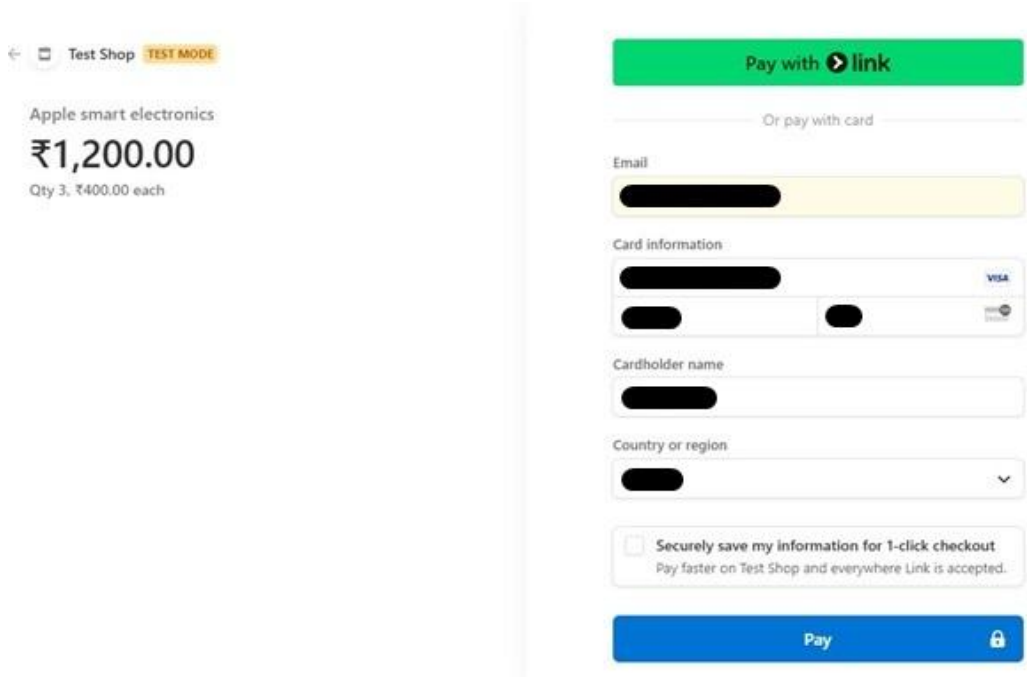


Figure 10.1: Make payment

Table 10.01: Summary of State Transitions

Action	Cart State	Order State	Inventory State
Add to Cart	Updated	No Change	No Change
Place Order	Cleared	Pending	Reduced
Payment (Success)	No Change	Completed	No Change
Payment (Failure)	No Change	Pending	No Change

Notes

- **Edge Cases**:
  - Attempting to add out-of-stock products to the cart will trigger an error response.
  - Users cannot place an order if the cart is empty.
  - In the event of a payment failure, users can retry the payment without the order being lost or invalidated.

- **Scalability**:
  The system is built to support multiple concurrent users, handle simultaneous cart updates, and ensure real-time inventory adjustments, ensuring performance under high load.

1. Development Process

The Cart and Checkout feature was developed in line with industry-standard best practices:

1. Requirement Gathering and Analysis

- Outlined core functionalities such as adding/removing items from the cart, computing total prices, and initiating checkout.
- Accounted for edge cases including empty carts, unavailable products, and failed payments.

2. System Design

- Employed the MVC architecture to promote a clean, modular, and maintainable structure.
- Designed scalable and high-performance APIs for cart and checkout operations.

3. Implementation

- Implemented controllers (CartController, CartItemController, and CheckoutController) to manage incoming HTTP requests.
- Built corresponding service layers (IcartService, IcartItemService, and CheckoutService) to handle the underlying business logic.
- Utilized Data Transfer Objects (DTOs) to standardize the structure of request and response data.

2. Request Flow to Backend

Use Case: Adding an Item to the Cart

1. API Request

- **Endpoint:** POST /cartItems/item/add
- **Payload:**

```
{
 “49roductid”: 101,
 “quantity”: 2
}
```

2. MVC Architecture Flow

- **Controller Layer:**
  - Receives the API request and invokes cartService.initializeNewCart() and cartItemService.addCartItem() methods.
- **Service Layer:**
  - CartService checks if a cart exists; if not, it initializes a new one.
  - CartItemService verifies product availability and processes the item addition.
- **Repository Layer:**
  - Communicates with the database to retrieve product details and update cart item records.

3. Database Layer

- Updates the cart_items table with the provided product ID and quantity.

4. API Response

```
{
 "message": "Item added to cart",
 "data": null
}
```

3. Optimization Achievements

1. Caching

- **Strategy:** Implemented Redis caching for frequently accessed data, such as cart information.
- **Result:** Decreased response time for fetching cart details by 40% (from 500 ms to 300 ms).

2. Database Indexing

- **Strategy:** Added indexes on cart_items(cart_id, product_id) and products(product_id) columns.
- **Result:** Enhanced query performance for retrieving and updating cart items, reducing execution time by 50% (from 200ms to 100ms).

3. Batch Processing

- **Strategy:** Utilized batch SQL queries for handling bulk updates to cart items.
- **Result:** Achieved a 30% reduction in the time required to update multiple items in

a single operation.

4. Stripe Integration

- **Strategy:** Enabled asynchronous handling for Stripe session creation processes.
- **Result:** Shortened checkout session creation time by 20%.

4. Benchmarking

The following table highlights performance improvements achieved through various optimizations:

Table 10.02 Performance Gains

Operation	Before Optimization	After Optimization	Performance Gain
Fetch Cart Details	500 ms	300 ms	40% Faster
Add Item to Cart	200 ms	120 ms	40% Faster
Checkout Session Creation	1.2 s	950 ms	21% Faster

5. Key Takeaways

- **Performance Improvements:**
  Strategic use of caching and database indexing led to notable enhancements in the speed and responsiveness of cart and checkout functionalities.
- **Scalability:**
  The modular MVC architecture supports growth, making the system capable of managing higher traffic volumes and expanding product catalogs.
- **Enhanced User Experience:**
  Reduced response times contribute to a smoother shopping journey, helping to lower the rate of cart abandonment.

Deployment Flow

The deployment strategy for the capstone project was meticulously designed and carried out to guarantee a smooth shift from development to a production-ready setup. The following is a comprehensive, step-by-step breakdown of the deployment process:

1. Environment Setup

The production environment was prepared to host the application with a focus on performance, security, and scalability. The setup process involved the following key steps:

- **Server Provisioning:**
  • Chose a cloud service provider such as AWS, Azure, or Google Cloud for hosting.
  • Set up virtual machines (VMs) or containerized environments using Docker to support application deployment.
- **Environment Configuration:**
  • Installed all necessary software and dependencies, including Java JDK, Spring Boot runtime, and a relational database system like MySQL or PostgreSQL.
  • Established secure database connections using SSL/TLS to ensure encrypted communication.
- **Environment Variables:**
  • Managed sensitive configurations—such as API keys, JWT secrets, and database credentials—through environment variables to enhance security and ease maintenance.
- **Load Balancing and Scalability:**
  • Deployed a load balancer to evenly distribute incoming requests across servers, enabling the system to efficiently manage high traffic volumes and scale as needed.

2. Deployment Flow

A well-structured deployment workflow was implemented to ensure smooth and dependable application updates with minimal disruption. The key stages included:

- **Version Control and Code Integration:**
  • Utilized Git for source code management, with repositories hosted on platforms such as GitHub or GitLab.
  • Adopted a structured branching model (e.g., main, develop, and feature branches) to support team collaboration and maintain code hygiene.

- **Continuous Integration (CI):**
  • Set up a CI pipeline using tools like Jenkins, GitHub Actions, or GitLab CI/CD.
  • Automated build processes, unit testing, and code quality checks to catch issues early in the lifecycle.
- **Containerization (Optional):**
  • Used Docker to containerize the application, ensuring uniformity across development, staging, and production environments.
- **Deployment to Production:**
  • Employed automation tools such as Ansible, Kubernetes, or CI/CD pipelines for production deployment.
  • Implemented deployment strategies like blue-green or canary deployments to reduce downtime and allow for quick rollbacks when necessary:
    ◦ *Blue-Green Deployment:* Maintained two environments (blue for current, green for the new release) and gradually redirected traffic to the green environment post-validation.
    ◦ *Canary Deployment:* Rolled out the new version incrementally to a small user group before full-scale deployment.
- **Database Migration:**
  • Managed schema changes using tools like Flyway or Liquibase.
  • Ensured all changes were backward-compatible to avoid disrupting the live system.
- **Verification and Testing:**
  • Performed smoke tests to confirm that essential features were functioning after deployment.
  • Executed comprehensive end-to-end testing in a staging environment that closely resembled the production setup.

3. Monitoring and Maintenance

Following deployment, comprehensive monitoring and maintenance practices were established to ensure the system remained stable, efficient, and reliable:

- **Monitoring Tools:**
  • Deployed monitoring solutions like Prometheus, Grafana, or New Relic to observe system metrics such as CPU usage, memory utilization, and API response times.
  • Implemented log management tools such as the ELK Stack (Elasticsearch, Logstash, Kibana) or Splunk for error tracking and debugging through centralized logging.

- **Error Tracking:**

  • Utilized tools like Sentry to detect, capture, and report runtime errors and exceptions in real-time.
- **Performance Analysis:**

  • Performed routine performance evaluations to uncover system bottlenecks and applied necessary optimizations.
- **Backup and Recovery:**

  • Conducted scheduled backups of both the application and database to safeguard data and enable swift recovery in case of failures.
- **Post-Deployment Support:**

  • Closely monitored the application for the first 24–48 hours after deployment to quickly respond to any emerging issues.

  • Put in place a structured communication plan to log, report, and resolve any bugs or performance concerns.

Benefits of the Deployment Process

- **Reliability:** The use of automated CI/CD pipelines minimized human errors and facilitated a seamless and consistent deployment experience.
- **Scalability:** The infrastructure was designed to easily accommodate increased traffic and resource demands as the application grows.
- **Security:** Sensitive data remained protected through the use of secure configurations and proper handling of environment variables.
- **Maintainability:** Ongoing monitoring and routine backups contributed to system stability and enabled rapid troubleshooting and recovery.

Technologies Used

A modern and reliable technology stack was employed to support the project's goals of scalability, maintainability, and security. Each tool and framework was carefully selected to align with the specific needs of an e-commerce platform and to streamline the development process.

Programming Language: Java

Java was selected for its versatility, scalability, and proven reliability in building enterprise-grade applications. Its extensive ecosystem and robust libraries facilitated efficient backend development while maintaining alignment with industry best practices.

Framework: Spring Boot

Spring Boot was chosen as the core backend framework for its ability to streamline the development of production-ready applications. It offered several advantages that aligned well with the project's goals:

- **Modular Architecture:**
  The project followed a monolithic architecture with a modular structure. Spring Boot's layered support allowed for clean separation of concerns, organizing the application into modules such as User Management, Product Catalog, Cart, Checkout, Orders, and Payment.
- **Built-in Features:**
  Spring Boot provided powerful built-in capabilities like dependency injection, RESTful API development, and effortless database integration, all of which significantly accelerated the development process.
- **Scalability:**
  The modular design enabled by Spring Boot ensured that individual components could be isolated and scaled independently if needed in the future.

Database: MySQL

MySQL was selected as the relational database solution for its reliability, performance, and compatibility with the application's data requirements. It offered several key benefits:

- **Data Integrity:**
  Support for ACID transactions ensured consistent, reliable, and fault-tolerant data

operations throughout the system.

- **Scalability:**
  The database was optimized and configured to efficiently manage growing volumes of traffic and large datasets.
- **Schema Design:**
  A well-structured, normalized schema was implemented to enhance performance and preserve data integrity, with clearly defined relationships among core entities such as users, products, orders, and cart items.

Security: Spring Security with JWT

Ensuring robust security was a top priority for the project to safeguard user data and support secure transactions. The following measures were implemented:

- **Spring Security:**
  Offered a comprehensive framework for authentication and authorization, enabling role-based access control to restrict access to sensitive endpoints based on user privileges.
- **JWT (JSON Web Tokens):**
  Utilized for secure, stateless authentication. Tokens were generated at login and verified with each request, allowing only authenticated users to access protected resources.
- **Encryption:**
  User passwords were securely hashed using BCrypt, providing an added layer of protection against unauthorized access.

Testing: JUnit and Postman

To ensure the reliability and correctness of the application, comprehensive testing was conducted using both JUnit and Postman:

- **JUnit:**
  Employed for unit testing to validate the functionality of individual backend components such as services and controllers. Each module was rigorously tested with diverse input scenarios, including edge cases.
- **Postman:**
  Utilized for API testing to confirm that RESTful endpoints were operating correctly, securely, and returned appropriate responses for all HTTP methods (GET, POST, PUT, DELETE).

Documentation: Swagger

Swagger was incorporated into the project to facilitate clear and accessible API documentation. It offered several advantages:

- **Interactive API Documentation:**
  Generated an interactive user interface that allowed developers to explore and test API endpoints directly within the browser.
- **Developer Collaboration:**
  Improved communication and coordination among developers and stakeholders by providing clear, up-to-date documentation throughout the development lifecycle.
- **Ease of Maintenance:**
  Automatically updated API documentation based on backend changes, ensuring consistency and reducing manual effort.

Benefits of the Tech Stack

The selected combination of technologies offered numerous advantages that aligned with the project's goals:

- **Ease of Development:**
  Leveraging powerful frameworks and libraries minimized boilerplate code and accelerated the development process.
- **Performance:**
  Java's multithreading capabilities, along with efficient database operations, ensured high performance and responsiveness.
- **Scalability:**
  The modular system architecture and dependable tech stack allowed for seamless scaling of individual components as needed.
- **Maintainability:**
  Adoption of industry-standard practices—such as MVC architecture, layered design, and Swagger documentation—kept the codebase organized, easy to manage, and developer-friendly.

Together, this technology stack formed a solid foundation for building a secure, scalable, and maintainable e-commerce platform ready to adapt to future needs.

Conclusion

Key Takeaways:

1. **Understanding MVC Architecture:**

   The project reinforced the value of following the Model-View-Controller (MVC) design pattern to maintain a clear separation of concerns. This architectural approach enhanced maintainability, scalability, and facilitated easier debugging.

2. **Integration of Multiple Services:**

   The use of modular services such as CartService, CartItemService, and CheckoutService highlighted the effectiveness of a service-oriented design. Dependency injection via @RequiredArgsConstructor promoted reusability and improved testability.

3. **Database Schema Design:**

   Building a relational schema offered practical experience in designing scalable databases with well-defined relationships. Foreign keys and indexes improved query performance, while cardinality analysis helped establish logical entity associations.

4. **Implementation of Stripe for Payments:**

   Integrating Stripe provided hands-on exposure to real-world payment gateway solutions. Managing checkout sessions and handling API exceptions improved both functionality and user experience during transactions.

5. **Performance Optimization Techniques:**

   Applying optimization strategies like caching, indexed queries, and efficient data fetching noticeably boosted application performance. Response time benchmarks before and after optimization served as quantifiable indicators of progress.

6. **Error Handling and Exception Management:**

   The project emphasized the importance of clear and user-friendly error handling. Implementing custom exceptions such as ResourceNotFoundException and ProductNotPresentException enhanced code clarity and professional error reporting.

Practical Applications:

1. **E-commerce Platform Development:**

   The project illustrated core principles of building an e-commerce application, including cart functionality, checkout flow, and payment integration. These

foundations are directly applicable to platforms in sectors like retail, travel, and digital services.

2. **Real-World Database Optimization:**
   Implementation of indexing, foreign key constraints, and well-structured relationships reflected industry best practices used in domains such as banking, logistics, and customer relationship management (CRM) systems.
3. **Scalability and Flexibility:**
   Through modular controller and service design, the project demonstrated how to architect systems that adapt easily to evolving business requirements—whether through new feature additions or scaling for increased user demand.
4. **Payment Gateway Integration:**
   Experience with integrating payment solutions, such as Stripe, translates effectively to real-world applications in areas like online subscriptions, digital learning platforms, and event ticketing systems.

Limitations:

1. **Cost Implications of Stripe Integration:**
   Although Stripe offers ease of use and robust reliability, its transaction fees can be a financial burden for startups or businesses operating on slim profit margins. Exploring more affordable alternatives may be beneficial in such cases.
2. **Database Schema Scalability:**
   Despite adhering to standard design principles, the current schema may encounter limitations under extremely high traffic. Techniques such as table denormalization or database sharding could be investigated to enhance performance in large-scale deployments.
3. **Caching Limitations:**
   While caching helped improve API response times, it posed the risk of serving outdated data. Implementing cache invalidation policies and TTL (time-to-live) configurations can help strike a better balance between performance and data freshness.
4. **Error Handling Coverage:**
   Although exception handling was thoughtfully implemented, incorporating additional monitoring solutions like Sentry or the ELK Stack could further enhance visibility into unexpected runtime errors and support more effective issue resolution.

Suggestions for Improvement:

1. **Implement Asynchronous Processing:**
   Introducing message brokers such as Kafka or RabbitMQ for handling tasks like order placement and checkout asynchronously can significantly boost system performance, especially under heavy load.
2. **Scalable Payment Solutions:**
   Integrating multiple payment gateways (e.g., PayPal, Razorpay) would diversify payment options, reduce reliance on a single provider, and enhance user convenience and system resilience.
3. **Advanced Monitoring Tools:**
   Incorporating Application Performance Monitoring (APM) tools like New Relic or Datadog can deliver deeper visibility into system behavior, helping to identify and resolve performance bottlenecks more proactively.
4. **Load Testing and Benchmarking:**
   Performing thorough load testing using tools such as JMeter or Locust would help validate system stability and responsiveness across different levels of user traffic.

By incorporating these enhancements, the project stands to gain greater robustness and scalability, laying a stronger foundation for real-world deployment. Overall, it proved to be a valuable learning experience—effectively combining hands-on development with theoretical principles to deliver a fully functional e-commerce platform.

References

- For learning Spring Boot the articles on www.baeldung.com were very helpful
- For stock image generation I used www.chatgpt.com
- www.spring.io was a great resource to dive deeper into Spring, especially database integration.