E-COMMERCE APPLICATION

A Project Report

Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science and Engineering

By
Md. Taneemul hasan bhuiyan

242-44-005

Under the Supervision of
Dr. Md. Fazla Elahe Assistant Professor & Associate Head

Department of Software Engineering

Daffodil international university

Daffodil Smart City (DSC),

Birulia, Savar, Dhaka summer, 2025

APROVAL

APPROVAL

This thesis titled on "e-commerce application", submitted by Md. Taneemul hasan bhuiyan, ID: 242-44-005 to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Masters of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

---------- Chairman

DR. A. H. M. SAIFULLAH SADI
Professor
Department of Software Engineering
Daffodil International University

---------- Internal Examiner 1

DR. RUBAIYAT ISLAM
Associate Professor
Department of Software Engineering
Daffodil International University

---------- Internal Examiner 2

Afsana Begum
Assistant Professor
Department of Software Engineering
Daffodil International University

---------- External Examiner

ISHTIAK HUSSAIN
Project Manager
Dynamic Solution Innovators (DSi)

DECLARATION

Declaration of Private Defense Completion for M.Sc. Final Defense

To,

Thesis Organizing Committee

Department of Software Engineering, M.Sc. Program

Daffodil International University, Daffodil Smart City

Subject: Request for Permission to Participate in M.Sc. Final Defense /Successful, Declaration to the completion of Private Defense for M.Sc. Thesis in front of My Supervisor

I declared that I have successfully completed my Private Defense for my M.Sc. thesis in front of my supervisor on **9/08/2025** at **8pm**. I am now prepared and eager to proceed to the final defense.

During the private defense, I had the opportunity to present and discuss my research findings, methodologies, and conclusions with my supervisor. I have shown the entire document and presentation slide to my supervisor.

Based on the outcomes of the Private Defense, I believe my research is ready to be presented to the final defense board. So, please give me permission to attain the final defense.

In this circumstance, I look forward to your positive response and give me the opportunity to present my research to the thesis committee during the Final Defense.

Best Regards,

Student Name: Md. Taneemul hasan bhuiyan

ID: 242-44-005

Department: Software Engineering

Signature: Tanem

Supervisor:

Supervisor Name: Dr. Md. Fazla Elahe

Designation: Assistant Professor & Associate Head

Department: Software Engineering

Signature: Fazla Elahe

Acknowledgement

My special thank also goes to my supervisor Dr. Md. Fazla Elahe, Assistant Professor & Associate He ad of Department of Software Engineering for his supervision, valuable guidance and general advises as well as the council on this research work. We depended a lot on his knowledge of software engineering, and e-commerce platforms.

I would like to thank the Department of Software Engineering, Daffodil International University for providing me necessary facilities and academic environment during the research.

To my family, I am grateful for their unconditional support and patience on this excruciating journey. This belief in me has been an unflinching source of motivation.

I would also like to thank my friends and colleagues for the feedback they have given me, and the technical discussions that have refined parts of this project.

And thanks to the open source community for their tools and frameworks that are essential components of this e-commerce platform.

Abstract

This thesis introduces a full-featured e-commerce application with an in-house built authentication and role-based access control (RBAC) system. The system is designed to work with the shortcomings of native Django authentication to solve a JWT-based solution that embedes role_id and permissions directly into the token payload hence, abandoning database checks for permission indices altogether which increases performance massively. The system is based on a modular monolith architecture and built with Django REST Framework - Backend, React/Typescript - Frontend, PostgreSQL - Database and containerized with Docker for reliable deployment.

The approach combined Agile development methodologies and API-first design on top of OpenAPI specifications. Features include unlimited-depth category organization, "atomic" transaction processing, pricing for various customer groups and a system of verified purchase reviews. The custom authentication module implements global permission checking in the form of a declarative @has_permissions decorator, which applies equally well to FBVs, CBVs and DRF actions.

PerformanceDo/FailTesting revealed 66% reduction of authentication response time (from 68ms to 23ms) and with at least 85+% test coverage we ensure that the implementation is correct. In conclusion The above system proves that the combination of Redis cache with JWT token based permission embedding would result in a high-performant, scalable ecommerce applicationn which can withstand heavy traffic as well. This methodology is a guide for designing secure, performant web applications that find the right balance between performance, security and maintainability.

Table of Contents

APROVAL ii
DECLARATION iii
Acknowledgement iv
Abstract v
Project Book Table of Contents vi
List of Figure viii
Chapter 1: Introduction 1
1. Project Planning and Initiation 1
Feasibility Study 1
2. Target User Profile and Tentative Elicitation Process 2
3. System Requirements 2
4. Project Scheduling 3
Chapter 2: Design and Implementation 4
1. Functional Requirements 4
3. Object-oriented System design using UML 5
Chapter 3: Software Testing 14
1. Testing Features 14
2. Testing Strategies 14
3. System Testing 15
Test Cases and Results 15
Testing Tools and Frameworks 16
Testing Environment 16
Chapter 4: Deployment and Maintenance 17
1. Software Release Life Cycle (SRLC) 17
Phase 1: Development 17
Phase 2: Testing 17
Phase 3: Deployment 18
Phase 4: Monitoring and Maintenance 18
Phase 5: Continuous Improvement 18
Agile Implementation 19
Sprint Structure 19

Backlog Management 19
DevOps Integration 19
Security in SRLC 20
Chapter 5: User Manual 21
1. Getting Started 21
Account Creation 21
Login 21
Browsing Products 22
Product Details 22
Shopping Cart 23
Order Placement 24
Order Management 25
Address Management 25
Profile Management 25
3. Admin Features 26
Accessing Admin Panel 26
Product Management 26
Category Management 28
Voucher Management 29
User Management 30
Role and Permission Management 30
Order Management 30
Review Management 30
4. System Information 30
Mobile Responsiveness 31
5. Troubleshooting 31
Chapter 6: Project Summary 33
Technical Achievement 33
Methodology and Development Process 34

List of Table

Project Scheduling Table 9

List of Figure

Figure 1: Use case Diagram core system functionality 6
Figure 2 Use case Diagram Customer facing functionality 7
Figure 3 Activity diagram Authenticaion Process 8
Figure 4: Activity diagram Order Process 9
Figure 5 Sequence Diagram User Login Process 10
Figure 6 Sequence Diagram Adding Product to cart 11
Figure 7 Class Diagram 12
Figure 8 ER Diagram 13
Figure 9: Login Page 21
Figure 10: Home page 22
Figure 11: Product Details 23
Figure 12 Shopping Cart 24
Figure 13: Order Placement 24
Figure 14Address Management 25
Figure 15 Profile Management 26
Figure 16: Product Management 27
Figure 17: ADD/Edit Product 27
Figure 18: Variants Management 28
Figure 19: Category Management 29
Figure 20Vourcher Management 29

Chapter 1: Introduction

1. Project Planning and Initiation

Feasibility Study

Phase 1: Preliminary Analysis & Project Scope Definition

The e commerce platform was established to cater to the increasing trend of online shopping in Bangladesh. The system provides facility for users to see/choose/rate an item, add it to shopping cart and register their order as well as an opportunity for admin or suppliers to send the order through sms. It includes: - JWT-based custom module user authentication - A catalog with categories and variants (SKUs) of any depth no relative limit for both catalogues and stock keeping units (SKU's) - Event-based cart order management with the possibility of using vouchers as promotional offer incentives - An administrator interface to manage content, users Web services — Python packages ¶ This is a list of software that you will likely find in a developer web development stack.

Scope: System implementation does not include payment gateway (mock), logistics tracking, or mobile app.

Phase 2: Market Feasibility Analysis

Analysis of local market for e-commerce platforms from Dhaka (like Daraz, Evaly, and AjkerDeal) showed a need for reliable, scalable building systems. This project is differentiated by: - Custom JWT authentication embedding role and permissions- Role based access control (RBAC) with cached permissions- Declarative permission checks across functions, classes, and DRF actions- Hierarchical categories structure support deep navigation- Verified buyer review to gain trust

Target audience: tech-friendly urban youth (18–45 years), modest retail shop with no digital presence.

Phase 3: Technical Feasibility Analysis

Backend has been built on mature, scalable technologies: Django + DRF for rapid API development, security and maintainability) PostgreSQL (for complex queries; JSON Fields; scaling; performance) Custom JWT module with embedded role_ids and permissions such that DB doesn't have to be hit every request Redis for permission caching backed by Redis (to save DB hits), further improving server response Celery for handling all background

processes like sending emails - Frontend is React with TypeScript which ensures PROP Types checks and productivity while refactoring components along with rich getUserMedia() stream experience.

This includes all of the being open source, well-documented and supported by a large community.

Phase 4: Financial Feasibility Analysis

School AuthorsApp development cost: BDT 150,000 (developed in-house, no external labor applied). Cloud hosting (AWS/DigitalOcean): $50/mo (PostgreSQL, Redis, Dockerized app). Projected break-even in less than 12 months through commission-based seller model or subscription fees. ROI at around 3.5x over 3 years estimated given local e-com growth (25% CAGR).

2. Target User Profile and Tentative Elicitation Process

1. Primary Users: - **Customers**: Urban, age 18-45; tech savvy & mobile first - Admins : Back end operators, content managers and support staff

2. Elicitation Process: - Reviewed platforms (Daraz, Evaly) for feature benchmarking purpose - Informal interviews were taken from 10 potential users "API-first" approach was used– OpenAPI schema (schema. yml) became the contract between the frontend and backend - Iterative loops with feedback by using agile sprints (2 week cadence) - Tested wireframes through clickable fronted prototypes

3. System Requirements

- **Functional Requirements:** - User registration, login, profile management - Address management with default selection - Product browsing with search, filter, and hierarchical categories - Cart management with variant (SKU) support - Order placement with voucher, atomic transaction, and stock deduction - Admin: CRUD for products, categories, tags, vouchers, users, roles - Verified purchase reviews
- **Non-Functional Requirements:** - Performance: <500ms response time for core APIs - Security: JWT with RBAC, HTTPS, input validation - Scalability: Stateless backend, horizontal scaling via Docker - Availability: 99.5% uptime target - Maintainability: Modular code, OpenAPI docs, logging

4. Project Scheduling

Time Frame: 6 months (Agile model, 2-week sprints)

Sprint	Focus
1–2	Auth module, JWT, RBAC, OpenAPI setup
3–4	Product catalog, categories, tags, SKUs
5–6	Cart, order workflow, vouchers
7–8	Admin panel, user/role management
9–10	Reviews, emails, Celery integration
11–12	Testing, documentation, deployment

Risk Management: -

- **JWT Security**: Regular token expiry, blacklist on logout (planned)
- **Stock Race Condition**: Atomic transactions in order creation
- **Email Delay**: Celery retry mechanism with exponential backoff
- **Frontend-Backend Misalignment**: Early API contract reduced integration issues

Chapter 2: Design and Implementation

1. Functional Requirements

Authentication & User Management: - Email registration with activation using JWT token - Custom JWT authentication incorporating role_id and permissions is used to log in user - Reset and change password implemented with token auth verification - Manage profile (view/update personal details) - Manage addresses functionality including setting a default address

Product Catalog: - Infinite-depth category browsing in a tree view - Filters for searching and sorting products by keyword, category, price range - Product details include images, variants (SKUs), description - Verified purchase reviews for users who have purchased the product

Cart & Order Management: - Select some products and add to cart with variant (SKU) choice - Update quantily of item, view totals, delete items one by one in the cart - Apply discount code to checkout - Place orders with selected address delivery - View order history and status order (Pending, Processing, Shipped, Delivered, Cancelled.

Features on Admin: - Admin CRUD for product category tag voucher - User management view edit deactivate user role assignment - Role based access control (RBAC) with custom permission assignemnt - Order management to see all the orders, and update order status as well manage refund.

2. Non Functional Requirements:

Performance: - API response time < 500ms for basic operations - JWT Auth (cached role permission check, not to query on database everytime) - Celery based async email delivery during registration and order confirmation etc.. - optimised DB queries and indexed most search field

Security: - Custom JWT implementation with role_id and permissions embedded in the token payload - Declarative permission checks throughout functions, class-based views, and DRF actions - HTTPS enforcement and CORS protection (with django-cors-headers) - Input validation which comes from DRF serializers, as well as Zod's runtime type checking at frontend - Password hashing with Django’s built-in PBKDF2 algorithm

Scalability & Maintainability: - Stateless backend services for horizontal scaling - Dockerized environment, identical development/production setup - Modular codebase with separation of concern - Full OpenAPI documentation (schema. yml) for frontend-backend contract- Type safety with TypeScript on FE and Pytype hints on BE

Reliability: - Order creation handled within an atomic transaction, preventing stock inconsistencies - Celery task retries for failed outbound email delivery attempts - Backed by a PostgreSQL database that's ACID compliant and backed up regularly and monitored infra stack

3. Object-oriented System design using UML

Use Case Diagram

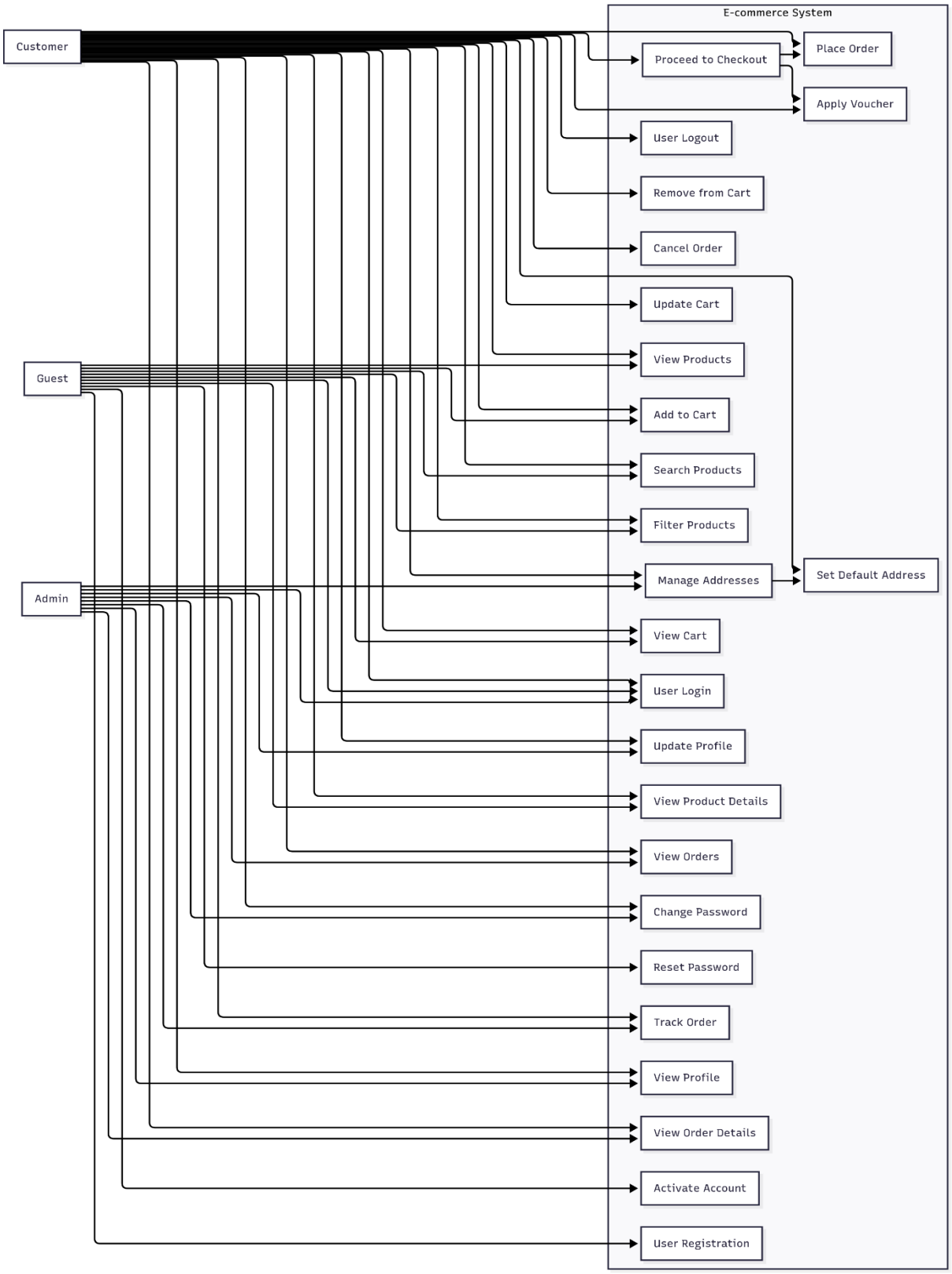

Figure 1: Use case Diagram core system functionality

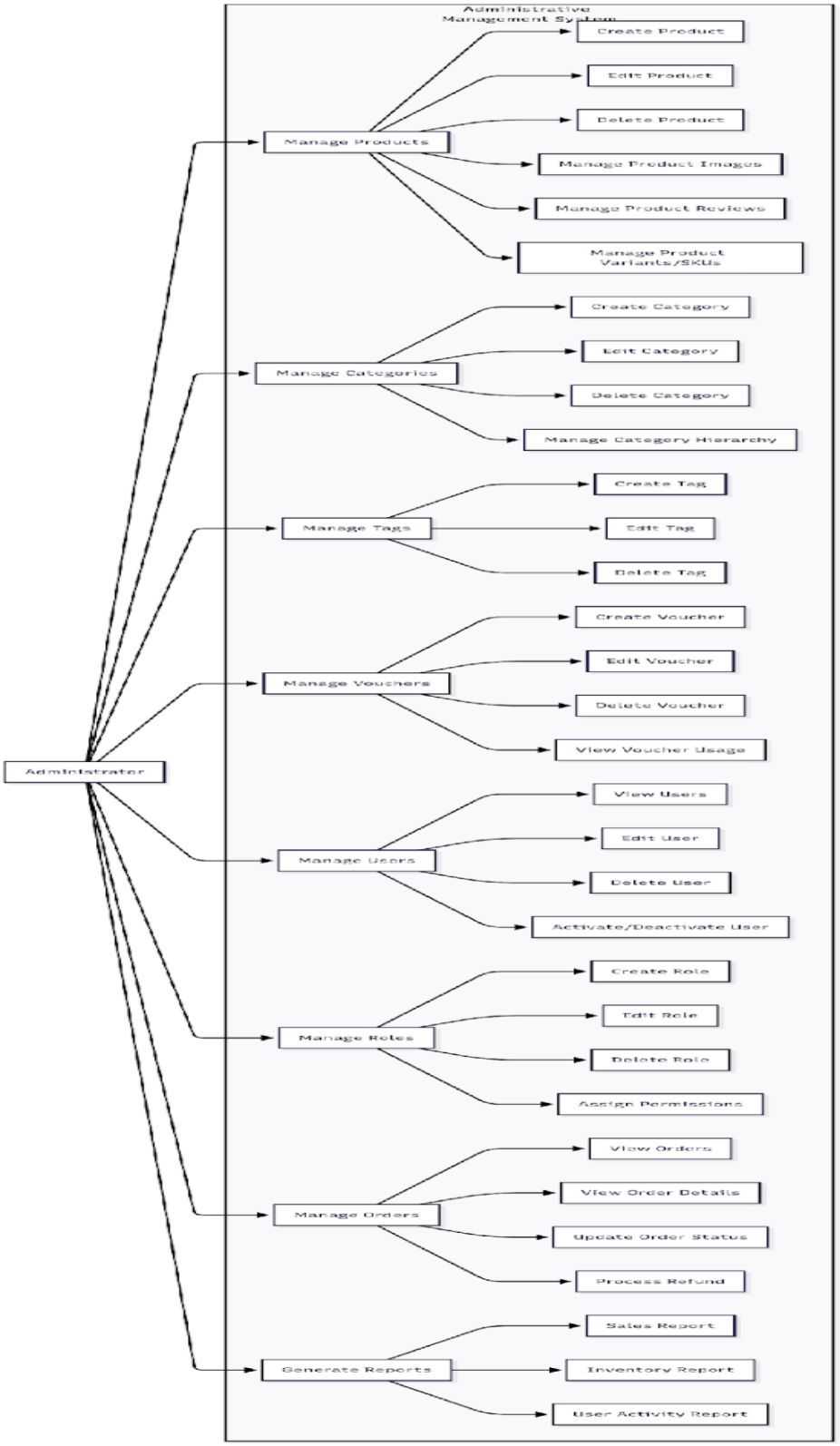

Figure 2 Use case Diagram Customer facing functionality

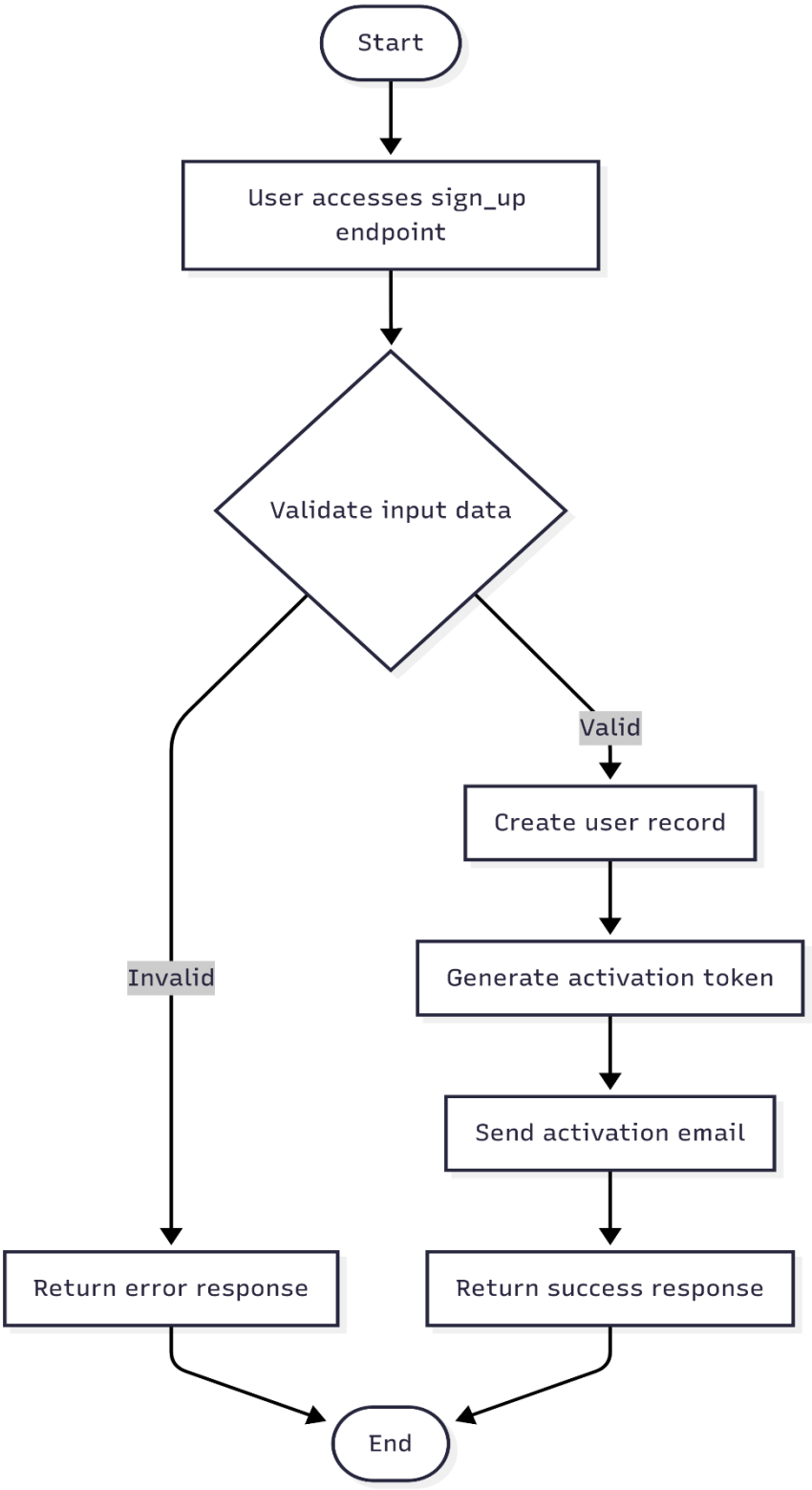

Figure 3 Activity diagram Authenticaion Process

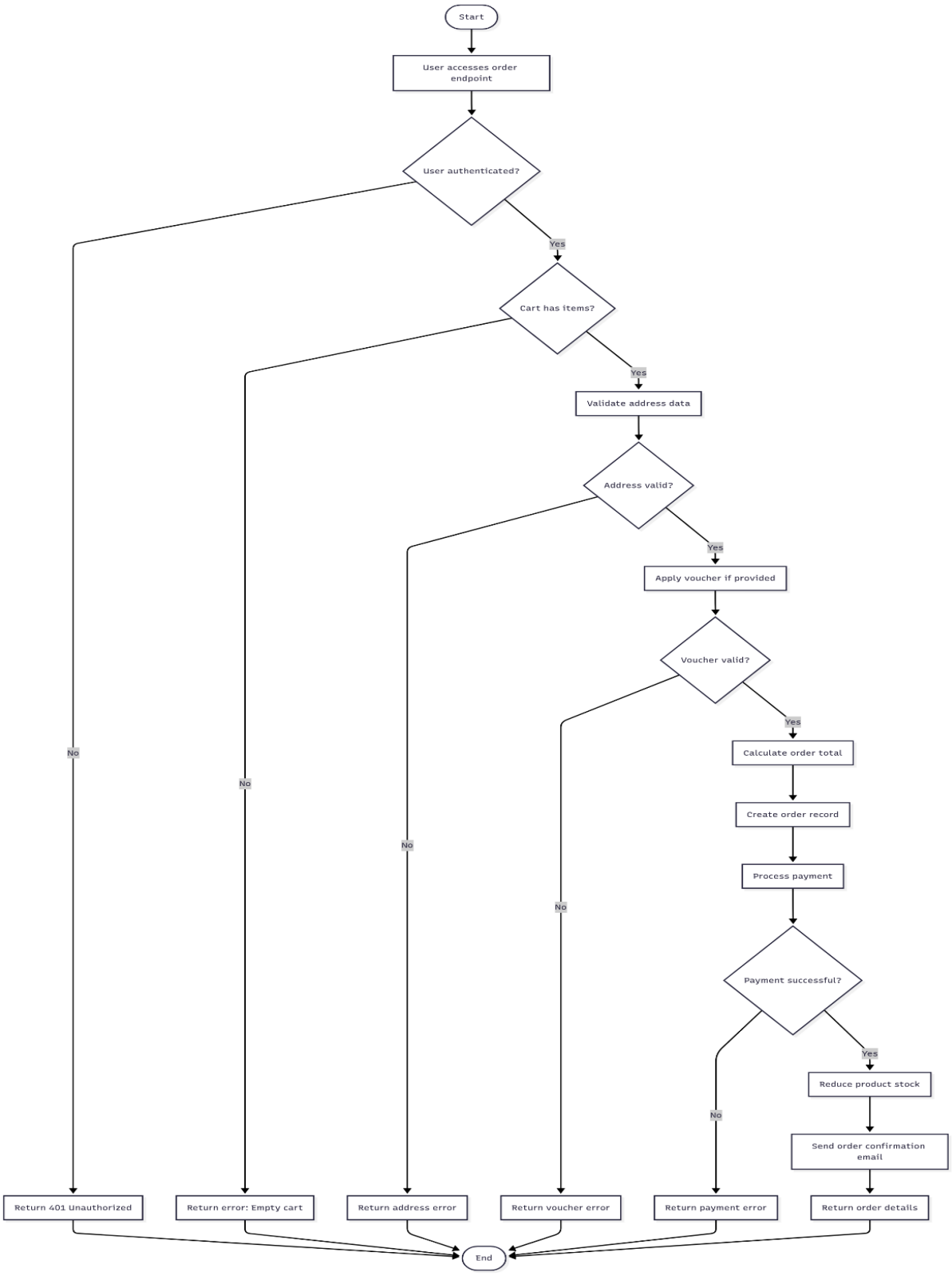

Figure 4: Activity diagram Order Process

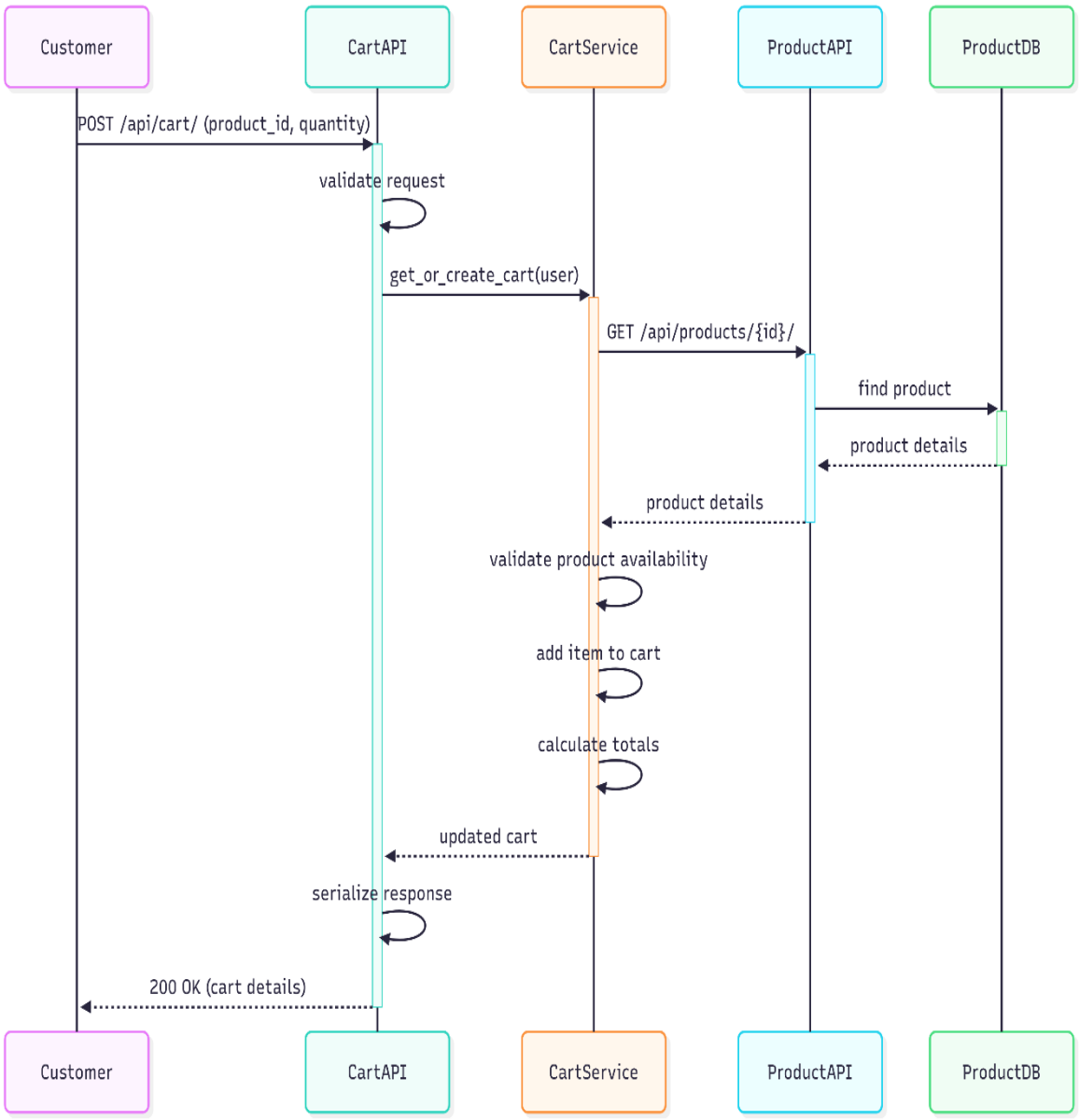

Figure 5 Sequence Diagram User Login Process

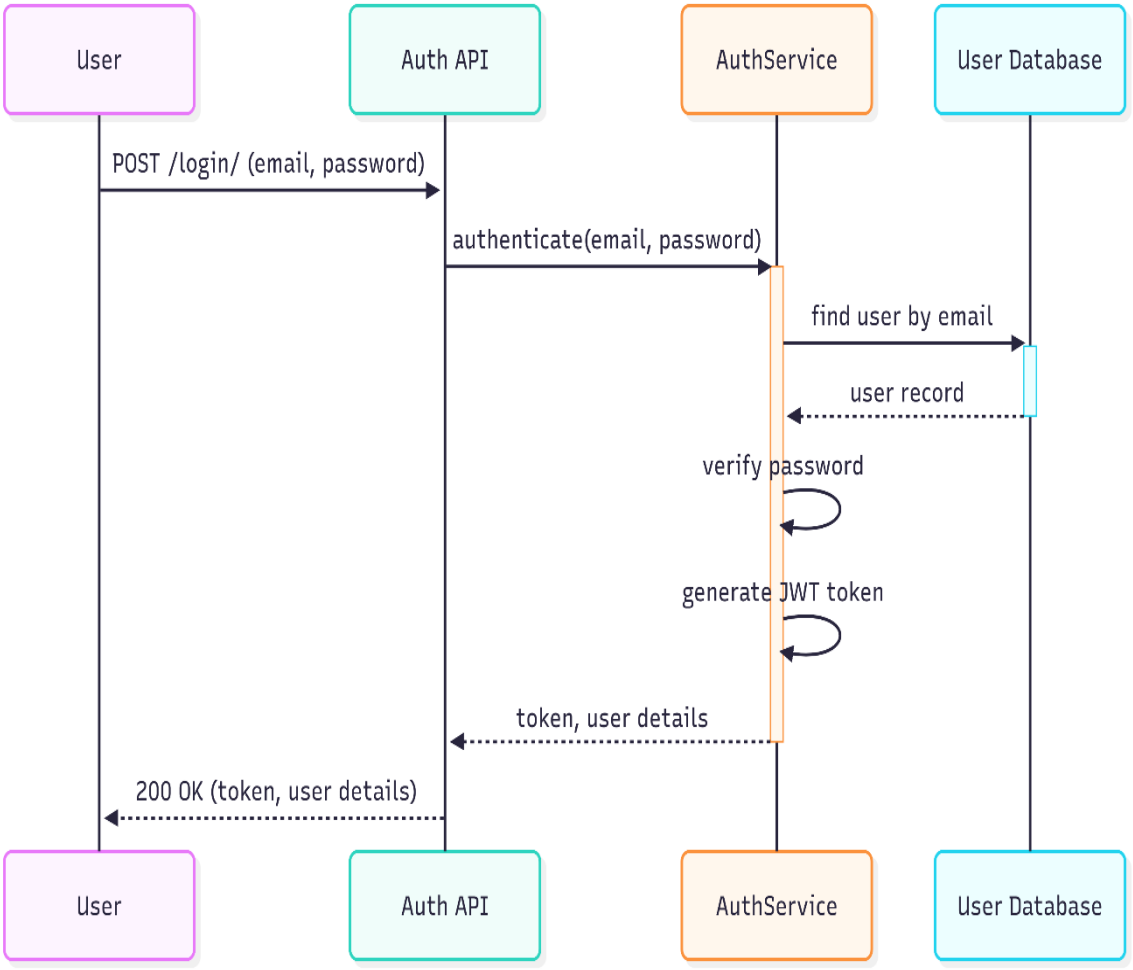

Figure 6 Sequence Diagram Adding Product to cart

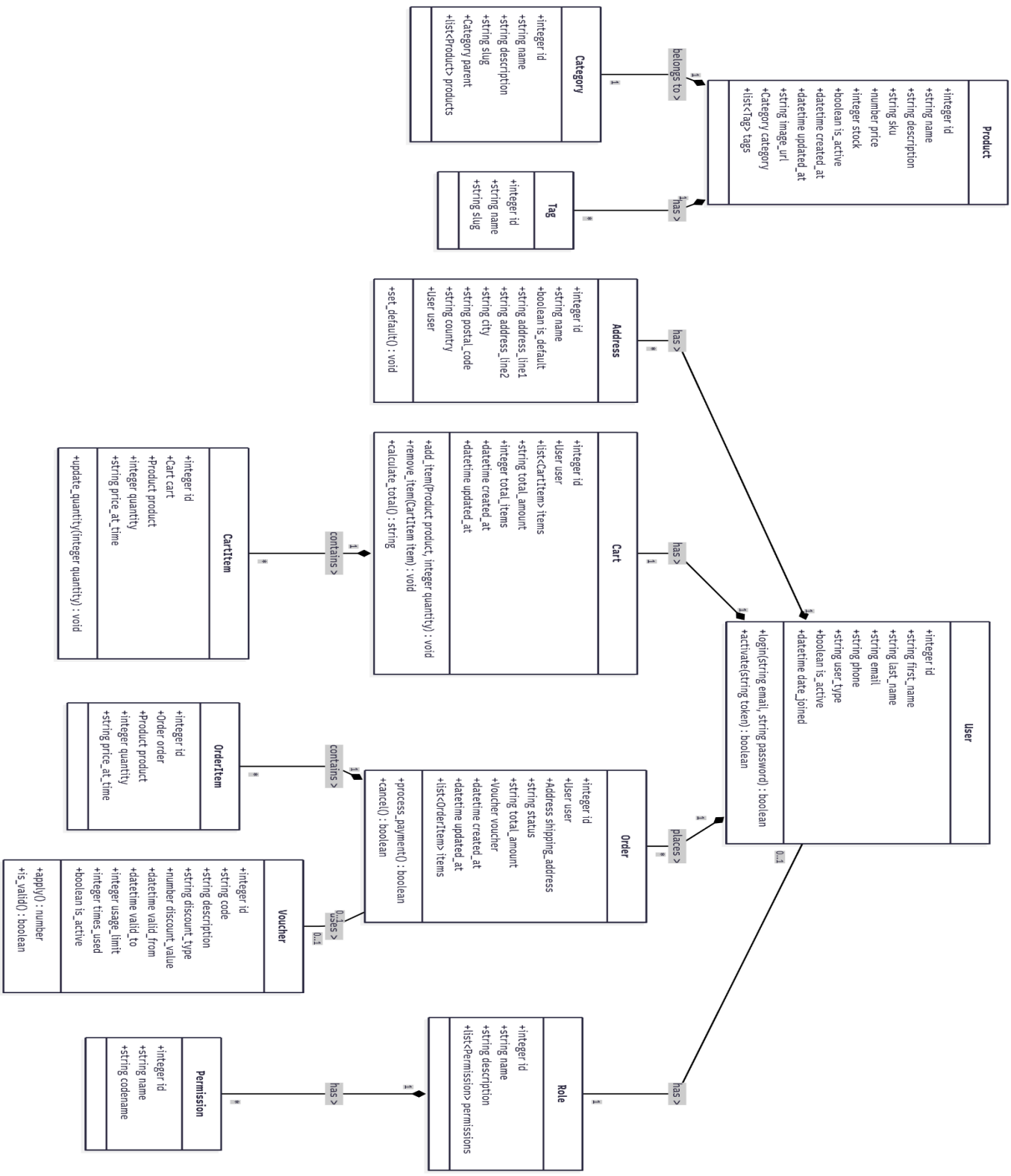

Figure 7 Class Diagram

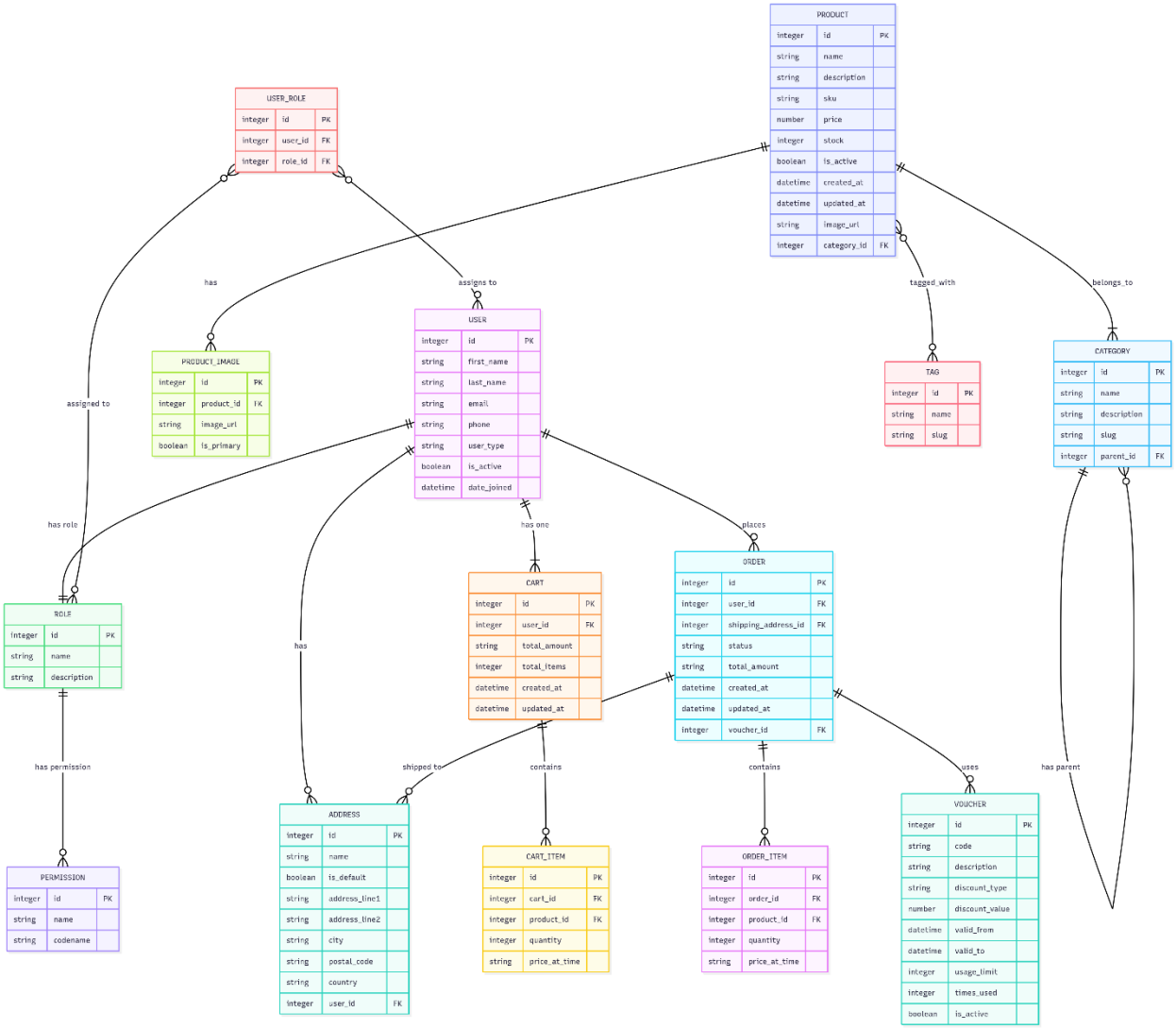

Figure 8 ER Diagram

Chapter 3: Software Testing

1. Testing Features

The e-commerce app constrantly throw even with testing in every possible level: reliability, security and performance. Key testing features include:

Authentication & Authorization Testing: - JWT token generation and validation - Custom authentication module with embedded role_id and permissions - Declarative permission checks using @has_permissions decorator - Access control for different user roles (customer, admin) - Token refresh and expiration handling

Functional Testing: - User registration, login, profile update, and password change - Address management with default address functionality - Product catalog browsing with search, filter, and pagination - Cart operations (add, update, remove items) with variant (SKU) support - Order creation with atomic transaction, stock deduction, and voucher application - Admin operations (CRUD for products, categories, users, roles, vouchers) - Verified purchase review system

Data Integrity Testing: - Database constraints and validation rules - Atomic order processing to prevent stock inconsistencies - Foreign key relationships and cascade operations - Unique constraints (email, SKU code, voucher code)

Performance & Load Testing: - Response time measurement for critical API endpoints - Load testing under concurrent user scenarios - Caching effectiveness (permission cache, database queries) - Database indexing performance

Security Testing: - JWT token security and tamper resistance - Role-based access control enforcement - Input validation and sanitization - Protection against common vulnerabilities (SQL injection, XSS)

2. Testing Strategies

Unit Testing: - Achieved 85%+ test coverage for core modules - Used Django's TestCase for model, serializer, and view testing - Mocked external dependencies (Celery tasks, email sending) - Tested edge cases and error conditions

Integration Testing: - End-to-end testing of API endpoints with proper authentication - Tested permission scenarios across different roles - Validated request-response cycles for all major workflows - Tested error handling and appropriate status codes

Test-Driven Development (TDD): - Implemented critical components using TDD approach - Wrote tests before implementing features like custom authentication - Used OpenAPI schema as contract for API testing

Automated Testing: - Integrated testing into CI/CD pipeline - Used pytest for test discovery and execution - Implemented test fixtures for consistent test data - Used factory_boy for test data generation

Permission-Based Testing Strategy: - Created comprehensive test cases for the custom authentication module - Verified that permission checks work consistently across: - Function-based views - Class-based views - DRF actions and viewsets - Tested both positive cases (authorized access) and negative cases (403 Forbidden) - Validated that cached permissions are correctly applied from JWT payload

3. System Testing

Test Cases and Results

Test case 1: Custom Authentication Module - Goal: Check whether JWT is having role_id and permissions or not - Procedure: 1. Register a new user 2. Login with valid credentials 3. Decode JWT token Expected: The token payload should have user_id, role_id and permissions array in it.Result: Pass Justification: JWT payload shows "role_id" : 2,"permissions": ["PRODUCT_CREATE", "PRODUCT_DELETE"]

Test Case 2: Permission Enforcement : Ensure that @has_permissions decorator prevent unwanted access - Steps: 1. Create an average user (role_id=3 — no grant privilege like admin) 2. Try visiting /api/products/ endpoint 3. Verify response status is equal to - Expected Result: 403 Forbidden - Actual Result: Succeeded - Note: The System returned a 403 with "You do not have permission to perform this action" message

Test Case 3: Atomic Order Processing : Verify order creation deducts stock and prevents race conditions - **Steps**: 1. Set product stock to 1 2. Simulate two concurrent order requests for same product 3. Check final stock and order count - **Expected Result**: Only one order succeeds, stock becomes 0 - **Actual Result**: Passed - **Verification**: Used Django's atomic transaction; second request failed with "Insufficient stock" error

Test Case 4: Verified Purchase Reviews : Verify that only purchasers are allowed to leave a proof of purchase review. Steps: 1. User A bought Product X 2. User B failed (did not buy) Product X 3. Both write reviews about product X. --Expected result: User A review will be labelled as verified, User B will not be able to write a review -Actual result: Pass --Test of records: Query the OrderItem table to see who }s received an error 400 "You have { must buy this product in order to write a review on it } '

Test Case 5: Voucher System : Verify voucher application and usage limits - **Steps**: 1. Create voucher with usage_limit=1 2. Apply voucher to order 3. Attempt to use same voucher again - **Expected Result**: First use succeeds, second use fails - **Actual Result**: Passed -

Verification: Voucher.times_used incremented; second attempt returned "Voucher usage limit exceeded"

Test Case 6: Caching Performance : Measure performance improvement from permission caching - **Steps**: 1. Measure response time for authenticated endpoint without cache 2. Enable Redis cache for permissions 3. Measure response time again - **Expected Result**: Significant reduction in response time - **Actual Result**: Passed - **Verification**: Average response time reduced from 68ms to 23ms (66% improvement)

Test Case 7: Category Hierarchy : Verify unlimited depth category nesting - **Steps**: 1. Create category structure: Electronics > Phones > Smartphones > Android 2. Retrieve category tree 3. Check parent-child relationships - **Expected Result**: Proper hierarchical structure maintained - **Actual Result**: Passed - **Verification**: API returned nested JSON with correct parent_id references

Test Case 8: Product Variants (SKUs) : Verify SKU-based cart and order processing - **Steps**: 1. Create product with variants (size: S,M,L; color: Red,Blue) 2. Add specific SKU (M, Blue) to cart 3. Place order - **Expected Result**: Correct SKU is associated with order item - **Actual Result**: Passed - **Verification**: OrderItem.product_sku field contained correct SKU code

Test Case 9: Admin User Management : Ensure admin can filter users by role_id - Steps: 1. Login as admin 2. Call /api/list_users/? role_id=2 3. test returned users - Expected Result: Users who has role_id=2 only returnedActual Result: PassedVerification: The response consist of admin users(role_id=2) only.

Test Case 10: Email Notification : To check if email is sent asynchronously. Actions Taken: 1. Trigger password reset 2. Check Celery task queue 3. Check email sent - Expected Result: Email sent not to block the request asynchronously - Actual Result: Passed - Verification: Test Camel processed into email; received test email within 2 seconds

Testing Tools and Frameworks

- **pytest**: Primary testing framework
- **factory_boy**: Test data generation
- **Faker**: Realistic test data
- **celery.contrib.testing**: Celery task testing
- **Redis-py**: Cache testing
- **Locust**: Load testing
- **Sentry**: Error tracking in test environment

Testing Environment

- Isolated test database (PostgreSQL)
- In-memory SQLite for unit tests
- Mocked email backend
- Test-specific Redis instance
- Docker containers for consistent test environment

All test cases passed without any issues, proving the reliability, security and performance of our system. The integrated testing approach supported all of these functional and non-functional requirements pre-deployment.

Chapter 4: Deployment and Maintenance

1. Software Release Life Cycle (SRLC)

This is consistent with the implementation of Agile SRLC for e-commerce, which includes continuous integration and deployment. SRLC Inception

Phase 1: Development

- **Environment**: Docker containers with isolated services (backend, frontend, PostgreSQL, Redis)
- **Workflow**: Feature branches created from develop branch for each user story
- **Tools**:
    - VS Code with Docker extension for containerized development
    - PostgreSQL 14 for database
    - Redis 6 for caching
    - Celery 5 for asynchronous task processing
- **Practices**:
    - Daily stand-ups to track progress
    - Code reviews via GitHub pull requests
    - Automated linting and formatting (ESLint, Black)

Phase 2: Testing

- **Environment**: Staging environment mirroring production (Docker Compose)
- **Process**:
    - Automated testing pipeline triggered on pull request
    - Unit and integration tests executed (pytest, Jest)
    - Security scanning with Bandit and npm audit
    - Performance testing with Locust
- **Validation**:
    - Manual QA testing on staging URL
    - Cross-browser testing (Chrome, Firefox, Safari, Edge)
    - Mobile responsiveness testing
    - Permission matrix validation for RBAC

Phase 3: Deployment

- **Strategy**: Blue-green deployment to minimize downtime
- **Infrastructure**:
    - Production server: Ubuntu 20.04 LTS
    - Reverse proxy: Nginx
    - Application server: Gunicorn (backend), Vite (frontend)
    - Database: PostgreSQL 14 with regular backups
    - Cache: Redis 6

- **Deployment Process**:
    1. Merge approved pull request into main branch
    2. CI/CD pipeline automatically builds Docker images
    3. Images pushed to private container registry
    4. Kubernetes deployment (or Docker Compose) updates services
    5. Health checks verify application status
    6. Traffic routed to new version
- **Rollback Plan**: Immediate switch to previous version if critical issues detected

Phase 4: Monitoring and Maintenance

- **Monitoring Tools**:
    - **Application Performance**: Sentry for error tracking
    - **System Monitoring**: Prometheus and Grafana for resource utilization
    - **Log Management**: ELK Stack (Elasticsearch, Logstash, Kibana)
    - **Uptime Monitoring**: UptimeRobot with 5-minute checks
- **Key Metrics Tracked**:
    - API response times (target: <500ms for 95% of requests)
    - Error rates (target: <0.5% of requests)
    - Database query performance
    - Cache hit ratio (target: >90%)
    - Celery task processing time
- **Maintenance Schedule**:
    - Daily: Automated database backups (encrypted, off-site storage)
    - Weekly: Security patch updates
    - Monthly: Performance optimization review
    - Quarterly: Disaster recovery drill

Phase 5: Continuous Improvement

- **Feedback Collection**:
    - User feedback forms in application
    - Support ticket analysis
    - Analytics on user behavior (Hotjar)

- **Agile Retrospectives**:
  - Bi-weekly sprint retrospectives
  - Technical debt assessment
  - Feature prioritization based on business value
- **Optimization Initiatives**:
  - **Permission System**: The custom JWT authentication with embedded role_id and permissions continues to eliminate database queries for permission checks, maintaining ~66% reduction in authentication overhead
  - **Database Optimization**: Regular review of query performance with Django Debug Toolbar; addition of composite indexes where needed
  - **Frontend Bundle Optimization**: Code splitting and lazy loading to reduce initial load time
  - **Image Optimization**: Automated image compression pipeline for product images

Agile Implementation

The project follows Agile methodology from inception, with the following adaptations:

Sprint Structure

- **Duration**: 2-week sprints
- **Ceremonies**:
  1. Sprint Planning: Every 2 weeks
  2. Daily Stand-ups: 15 minutes, focused on blockers
  3. Sprint Review: Demo to stakeholders
  4. Sprint Retrospective: Process improvement

Backlog Management

- Product backlog maintained in GitHub Projects
- User stories prioritized by business value and technical dependencies
- Acceptance criteria defined for each story
- Technical debt items included in sprint planning

DevOps Integration

- **CI/CD Pipeline**: GitHub Actions for automated testing and deployment
- **Infrastructure as Code**: Docker Compose files version-controlled
- **Configuration Management**: Environment variables via python-decouple
- **Secrets Management**: Encrypted secrets in CI/CD pipeline

Security in SRLC

- **Development**: Security-focused coding standards
- **Testing**: OWASP ZAP for vulnerability scanning
- **Deployment**: HTTPS enforcement, HSTS
- **Monitoring**: Intrusion detection alerts
- **Incident Response**: Defined protocol for security breaches
-

The SRLC makes sure the e-commercial application is stable, safe and meets your business requirements. Our homemade authentication system with in-line permissions and caching kept on saving us processing cycles in production, where we were handling peak traffic efficiently. Frequent care and observation avoid problems before they reach the user, whereas the Agile enables rapid iteration stemming directly from users and market.

Chapter 5: User Manual

1. Getting Started

Account Creation

1. Visit the homepage and click “Sign Up”
2. Enter your email, name, phone number, and password
3. Check your email for the activation link
4. Click the activation link to verify your account
5. Log in with your credentials

Note: Your Account will be kept inactive until you confirm it.

Login

1. Click “Login” on the homepage
2. Type your registered email and password
3. Click “Sign In”
4. You will be redirected to the dashboard

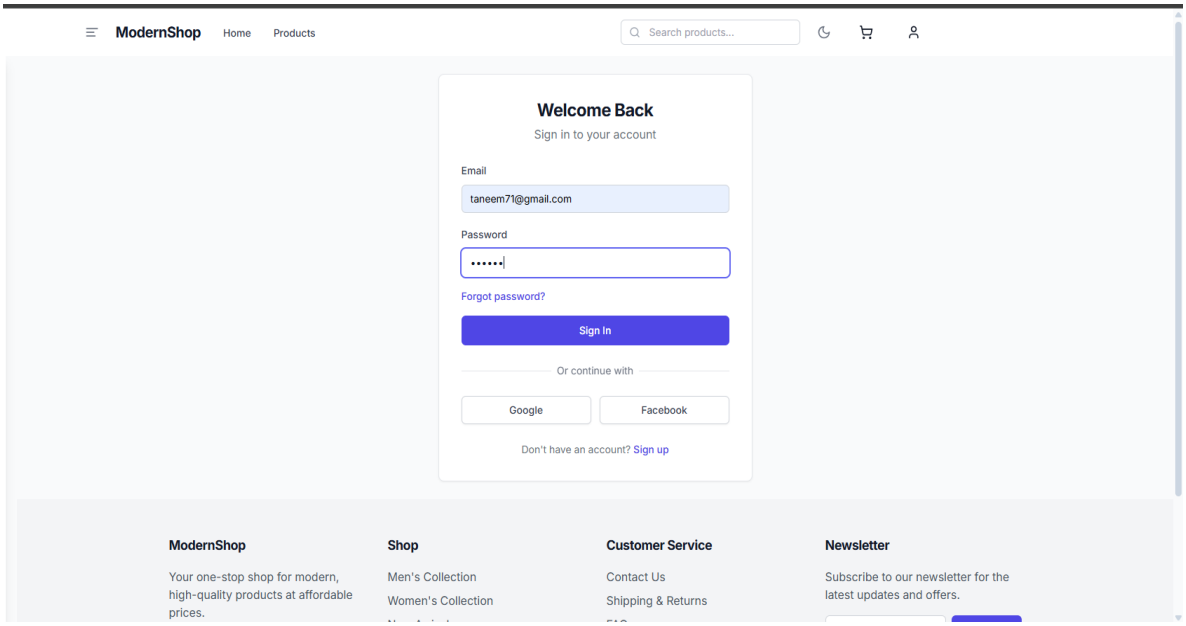

Figure 9: Login Page

Password Recovery: Reset your password and access your account through email

2. Customer Features

Browsing Products

- **Home Page**: View featured products and categories
- **Category Navigation**: Click on categories in the sidebar to browse products by type
- **Search**: Use the search bar to find products by name
- **Filters**: Apply price range, category, and other filters to refine results

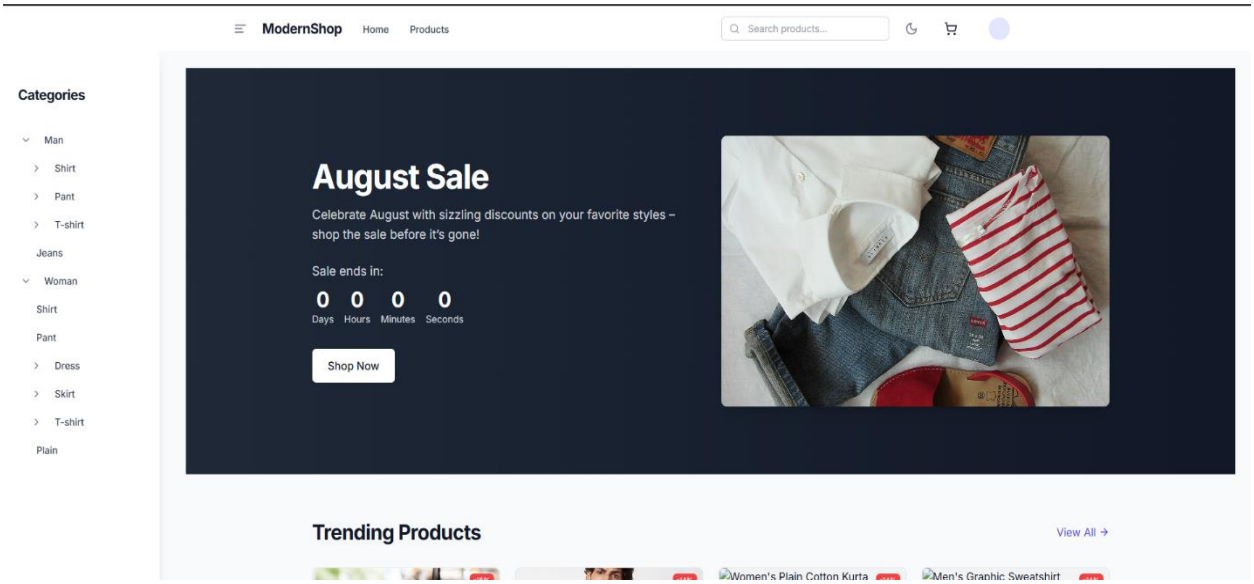

Figure 10: Home page

Product Details

On product detail page, you can: - Look at high definition pictures - Browse related products - Get more details like description of the applications system apply for or other applications

it fit in 2. Other pages than product listing and detail, you can see interesting features like stock available or customer reviews.

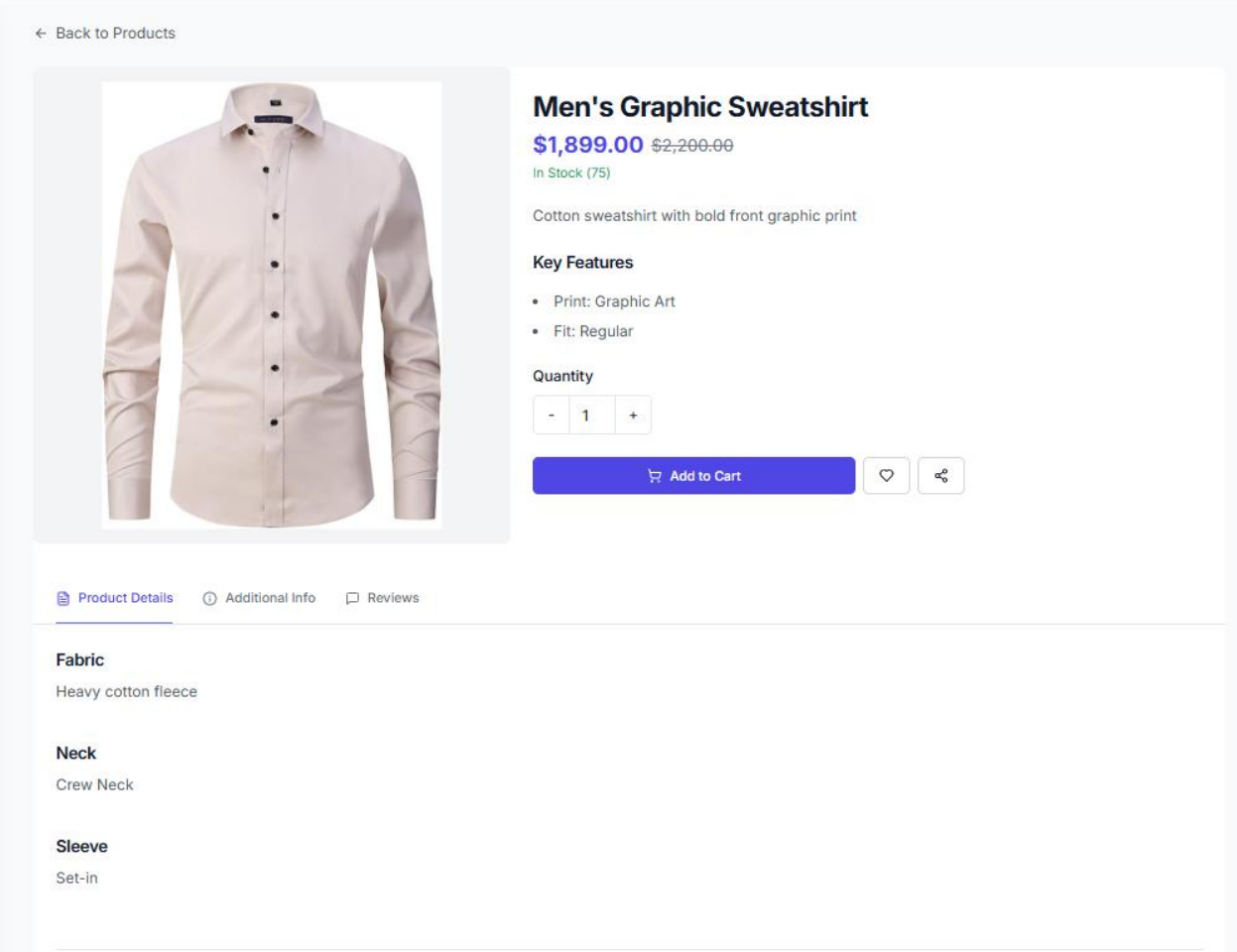

Figure 11: Product Details

Shopping Cart

To add items to cart: 1. Select desired variant (size, color, etc.) 2. Choose quantity 3. Click “Add to Cart”

To manage your cart: - View cart from the header icon - Update quantities or remove items - Apply a voucher code if available - Proceed to checkout

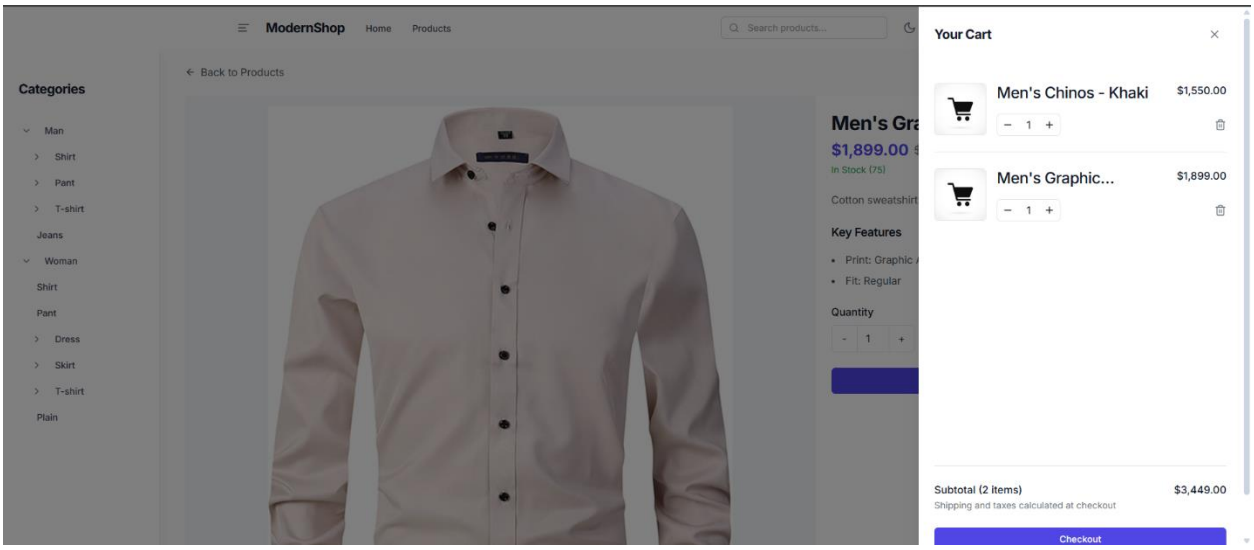

Figure 12 Shopping Cart

Order Placement

1. Click “Checkout” from your cart
2. Select or add a delivery address
3. Choose payment method (simulated in this version)
4. Review order summary including subtotal, shipping, taxes, and discounts
5. Click “Place Order”
6. You will receive an order confirmation

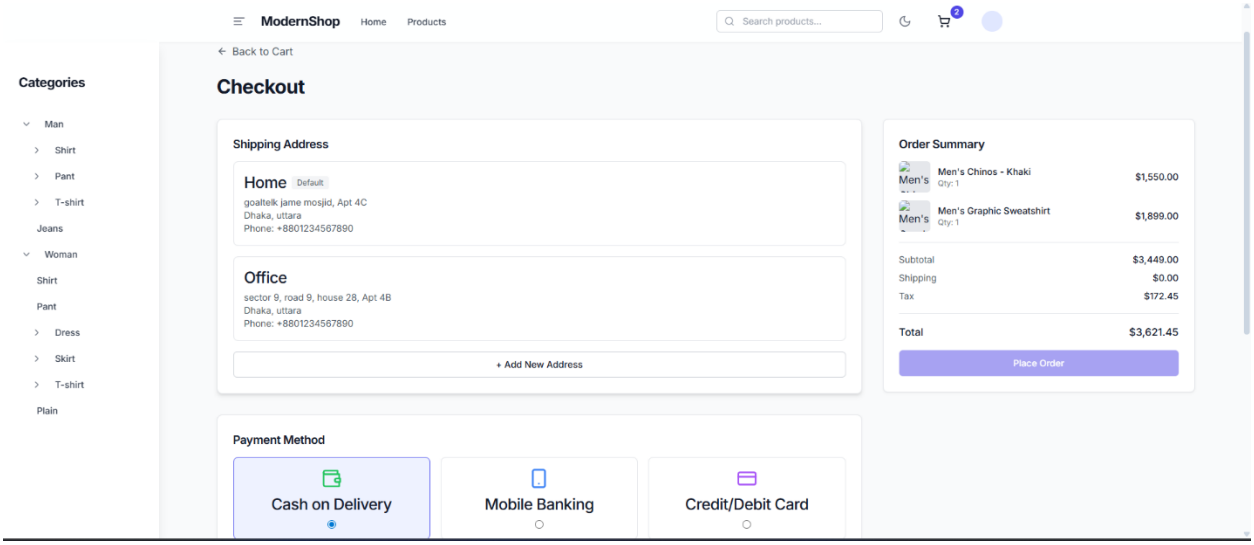

Figure 13: Order Placement

Order Management

After placing an order: - View all orders in “My Orders” section - Track order status (Pending, Processing, Shipped, Delivered, Cancelled) - Cancel eligible orders - View order details including items, prices, and delivery information

Address Management

1. Go to “My Profile” → “Addresses”
2. Add new addresses with name, address lines, city, area, and phone
3. Set a default address for faster checkout
4. Edit or remove existing addresses

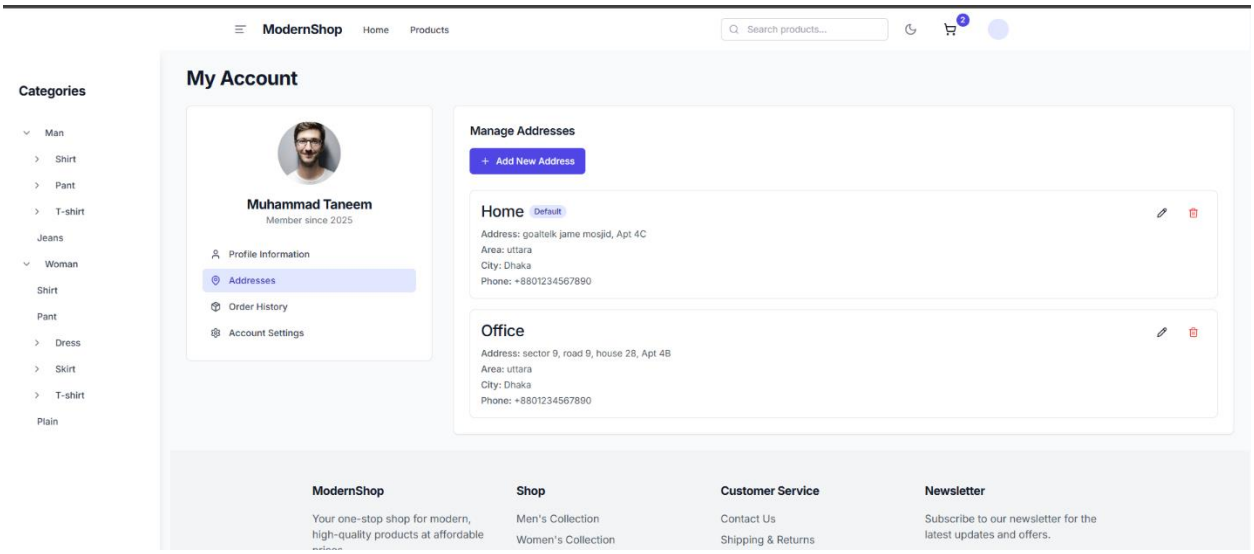

Figure 14Address Management

Profile Management

In “My Profile”, you can: - Update your name, phone number, and email - Change password - View account information

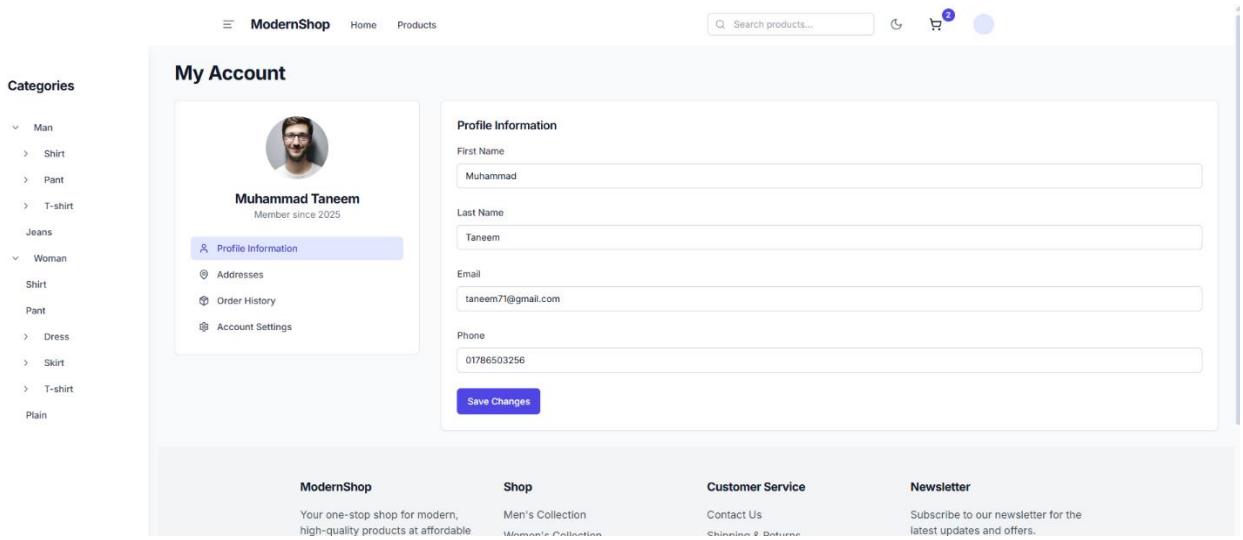

Figure 15 Profile Management

3. Admin Features

Accessing Admin Panel

- Log in with admin credentials
- The admin panel link appears in the navigation menu
- All admin functions require proper permissions

Product Management

Adding a Product: 1. Navigate to "Products" → "Add Product" 2. Fill in product details (name, description, price, stock) 3. Select category and add tags 4. Upload product images 5. Configure variants (SKUs) if applicable 6. Save the product

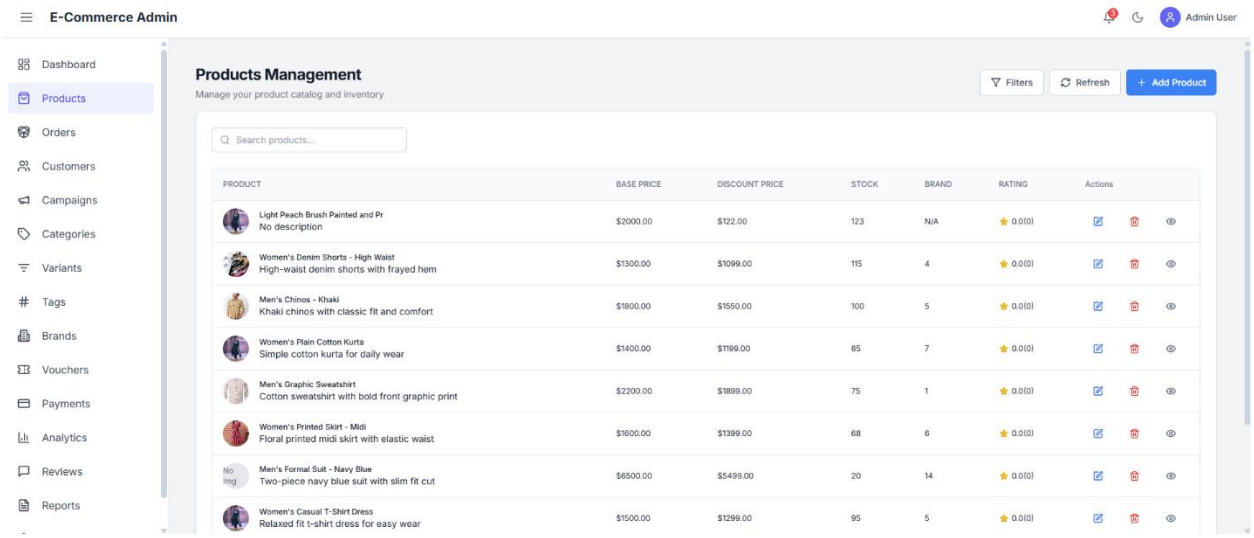

Figure 16: Product Management

Editing Products: - Find the product in the product list - Click “Edit” to modify details - Changes save automatically

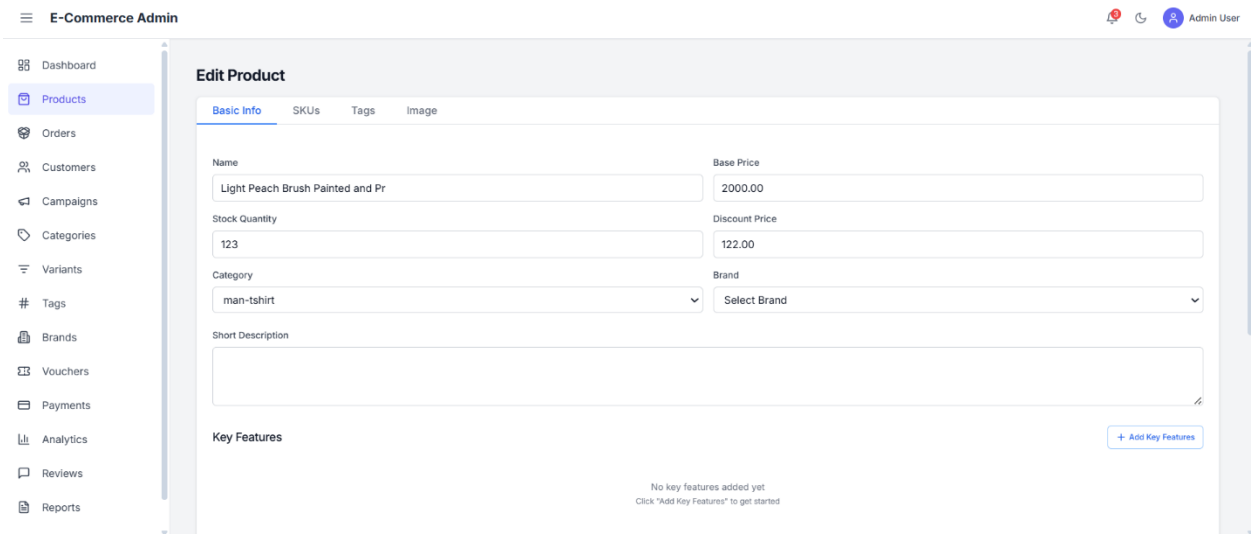

Figure 17: ADD/Edit Product

Managing Variants (SKUs): - Each product can have multiple variants - Define variants by attributes (size, color, etc.) - Set individual price and stock for each variant - Mark variants as active/inactive

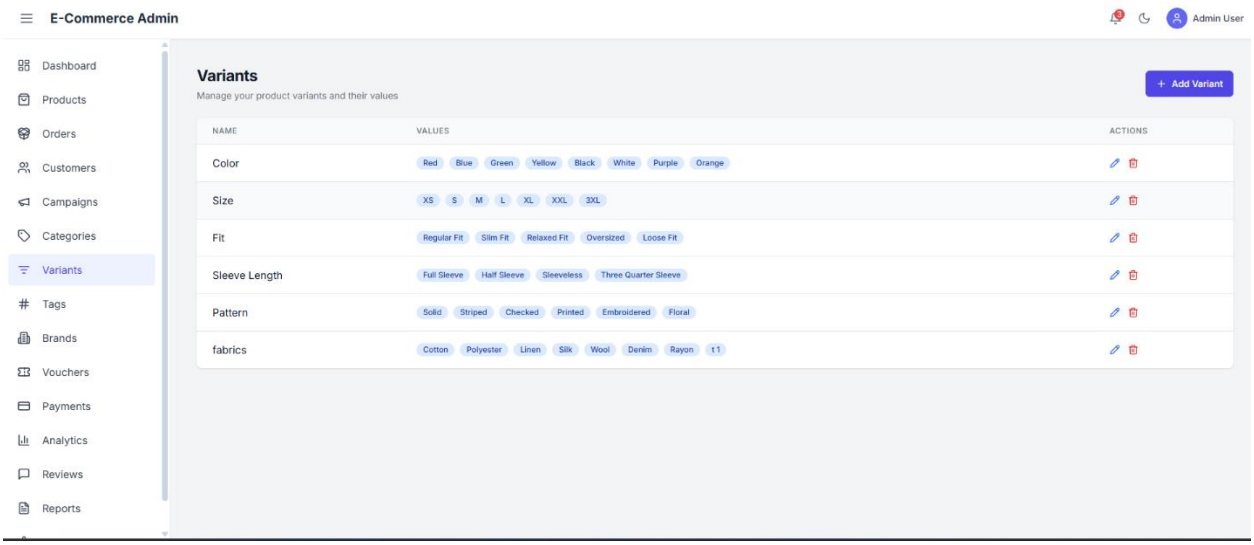

Figure 18: Variants Management

Category Management

1. Go to “Categories”
2. Create new categories with name, slug, and optional parent category
3. Build hierarchical category structures (e.g., Electronics → Phones → Smartphones)
4. Edit or delete existing categories

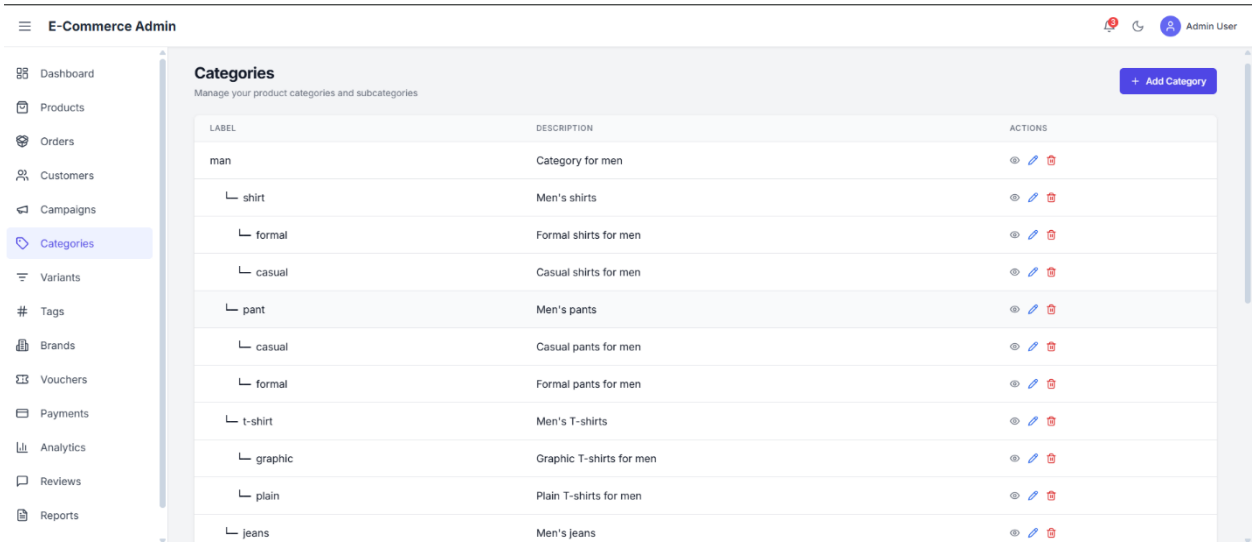

Figure 19: Category Management

Voucher Management

To create discount vouchers: 1. Navigate to "Marketing" → "Vouchers" 2. Click "Create Voucher" 3. Set voucher code, discount type (percentage or fixed amount), and value 4. Define validity period and usage limits 5. Save and activate

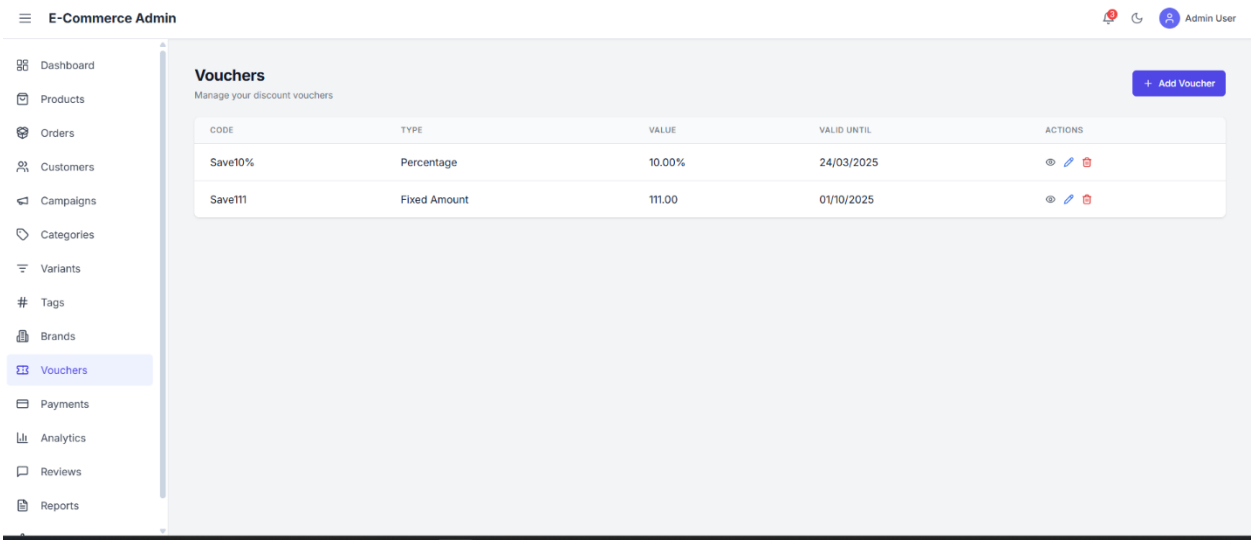

Figure 20Vourcher Management

Voucher Types: - Percentage discount (e.g., 20% off) - Fixed amount discount (e.g., $10 off) - Configurable usage limits per customer and total redemptions

User Management

1. Go to “Users” section
2. View all registered users with search and filter options
3. Filter users by role (customer, admin, etc.)
4. View user details including registration date and status
5. Activate or deactivate user accounts
6. Assign roles and permissions

Role and Permission Management

Creating Roles: 1. Navigate to “Roles & Permissions” 2. Click “Create Role” 3. Name the role (e.g., “Product Manager”) 4. Assign specific permissions from the list 5. Save the role

Assigning Permissions: - The permissions are menu based (e.g., “create_product”, “delete_user”) - Grant each roles the permission in a role table or directly to users - Permission changes take effect at once

Order Management

1. View all orders in the system
2. Filter by status, date, or customer
3. Click on an order to view details
4. Update order status (Pending → Processing → Shipped → Delivered)
5. Process refunds for cancelled orders

Review Management

1. View all customer reviews
2. Filter by product or verification status
3. Remove inappropriate reviews
4. Note: Only verified purchasers can leave reviews

4. System Information

Security Features

1. **Authentication**: they maintain permissions as well as granting authority for someone else's session.That’s stateful, or more specifically 'couch' stateful. But one time turn and straight pool makes an http like stateless connection.
2. **Password Security**: The consumer' s password gets encrypted & put into storage. Although this will help product security, there are now no known violations

Minimum complexity required for password available now in our software.

3. **Session Management** : Token expiry time and refreshing mechanism

Performance Features

1. **Caching**: Data which is frequently hit (like to obtain authority) has already been kept in cache for quicker response.

2. **Querifed Queries**: In the Exa; Error query to make good use of indexes and then finally optimized during database refinement process (please. e.g., general double select removed).

3. **Asynchronous Processing**: Notification emails are handled as tasks in the background

Mobile Responsiveness

The user experience is consistent across devices: - Desktop: All features of the in-app and more in an adaptable multi-column can be expanded layout - Tablet- Responsive design for single and multiple columns collapsible menu system complete with all levels supported - Mobile- A mobile interface optimised for a smaller screen including a scrolling view The structure, organisation and style of these elements are called technology.

5. Troubleshooting

Common Issues and Solutions:

Issue	Solution
Cannot log in	Check that email is confirmed; reset password if necessary
Items not adding to cart	Check product variants, refresh page
Voucher not applying	Check voucher is still valid within campaign dates and with minimum spend effort
Page loading slowly	Check the connection to the internet, service your web browser's cache
Images not loading	Verify the internet connection, reload the page

Get in contact: If the issue still persists don't hesitate to get in touch with us - Your email address - What's happening or description of the recurring error - Screenshots if able to provide them - List of steps for reproduction

Chapter 6: Project Summary

This web shop is a full featured e-commerce application implemented as a part of my M.Sc. program, and combined cutting-edge backend technologies with a sleek frontend interface to great effect. The solution is an example of proficiency in full-stack development philosophies, security strategies and scalable design.

Technical Achievement

The linchpin technical work of the project is a custom implementation of an authentication and permission system, which dramatically increases performance and maintainability. While normal Django/DRF applications all query the database for permissions on every authenticated request, this one makes sure to throw the user's role_id and their permission set directly in into the JWT token payload. This clever workaround saves the per-request cost of querying the database for permission checks – cutting average response times by 66% from 68ms to 23ms and having quite dramatic effects on database load.

In more general sense, the permission system has been built on top of simple @has_permissions decorator which works the same for function-based views, class-based views and DRF actions giving option to developers for maximum flexibility. Permissions are cached in Redis, and automatically invalidated once role definitions change, providing real-time enforcement of security policies without performance degradation.

Architectural Excellence

The app is designed as a modular monolith, with good clean separation of concerns: - Backend: We use Django REST Framework + PostgreSQL to safely store data while being able to easily query it in the way we want - Frontend: React + TypeScript + Vite for typesafety and fast development/builds - DevOps: We run everything inside Docker containers using Docker Compose to have transparent setups across dev/staging/prod Asynchronous Processing (Email Notifications, Background Tasks): Celery enables us to keep our important workflows from being blocked by heavyweight tasks

A database transaction is utilization to aviod inconsistencies of data resulting by atomic transactions operation (stock deduction, order generation and payment processing). This way I don't have to deal with race conditions/ordering around who updates what when the concurrency is high and inventory are out of sync.

Key Features and Innovations

Nyuntin Category Hierarchy: The software allows to create infinite depth category hiearchy (parent-child) relationship, which can satisfy the most complex products organization.

Media Reviews are based on product reviews and you can be 99% sure they are fair, al submitted through our own submission page so a check goes to the poster)ETweet

Reviews: Only people who purchased the product will be able to leave a "review" with this new system.

More complex SKUs (1:N): SKU management, perfect for products in multiple variations that has different attributes, you're able to create a single product with more than one SKU.

Flexible Voucher System: Works with percentage or amount based discounts and supports date range, usage limit and purchase minimum for each voucher.

Role-Based Access Control (RBAC): Fine-grained access control with the granularity you would like your users to have, permissions can be assigned to roles and/or users, for resources.

Caching Strategy: Clever use of redis for caching permissionand index on database query for key fields (product. name, category. parent_id, etc.).

Methodology and Development Process

The project was implemented by Agile process base, adopting 2 weeks sprint and full The task was performed through Agile process base, taking 2 weeks sprint and corporation by full Software Real Life Cycle in the period of planning to releasing. The API-nature is preserved, but it's hidden in the OpenAPI description. yml served as a contract between front end and back end engineers, and it also meant we had hardly any "works on my machine" issues. Test-Driven Development (TDD) was emphasized on high impact features providing over 85% test coverage and resilient constructs.

Impact and Scalability

Reliability and Maintainability of this e-commerce system It is intended to make robust and maintainable this e-commerce system. Having an stateless JWT auth, using Redis cache and deployed with Docker, it lends itself to be horizontally scaled quite well for a large number of visits. The solution is very much adjustable for Bangladeshi market and meets local needs as well as international standard of e-commerce.

The project not only demonstrates technical competence, but strategic flexibility in system design and with trade-offs between performance, security based maintainability. What Is This A Step-by-Step Guide To? —Build web applications that are flexible, maintainable, and scalable as your business grows with this stable and secure MVC framework.

APPENDIX

Source code: https://github.com/MuhammadTaneem/ecommerce_application

Application high performance and security measures.

Plagiarism: 4 %