**## 1. Introduction**

**### 1.1 Purpose**

This Requirements Design Specification (RDS) document specifies the system and software requirements for the SASUCare E-Commerce Platform. It provides a detailed description of the system's functional and non-functional requirements, architecture, and design constraints.

The document is intended for use by developers, testers, project managers, and stakeholders involved in the development and implementation of the SASUCare platform.

**### 1.2 Scope**

A diagram of a business

AI-generated content may be incorrect.

The SASUCare E-Commerce Platform is a comprehensive online marketplace that enables:

- Customers to browse products, manage shopping carts, and place orders

- Sellers to list products, manage inventory, and process orders

- Administrators to oversee the platform, manage users, and maintain the system

The system includes a web-based frontend for user interaction and a robust backend for data management and business logic processing. Key capabilities include:

- User registration and authentication with role-based access

- Product catalog browsing and searching

- Shopping cart management and checkout process

- Order processing and tracking

- Seller dashboard for product and order management

- Administrative controls for platform management

- Discount and promotion management

- File upload for product images

- Product categorization and filtering

- User profile and address management

- Communication between customers and sellers

**### 1.3 Definitions and Acronyms**

| Term/Acronym | Definition |

|---|---|

| API | Application Programming Interface |

| CRUD | Create, Read, Update, Delete |

| DTO | Data Transfer Object |

| JPA | Java Persistence API |

| JWT | JSON Web Token |

| MVC | Model-View-Controller |

| RBAC | Role-Based Access Control |

| REST | Representational State Transfer |

| SQL | Structured Query Language |

| UI | User Interface |

| UX | User Experience |

**### 1.4 Actor**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Actor** | **Description** | **Key Input to the System** | **Key Output from the System** | **Primary Responsibilities** |
| 1 | **Customer** | End user of the marketplace who browses products and places orders. | - Search terms, filter/sort criteria<br/>- Registration & login credentials<br/>- Product selections & quantities<br/>- Shipping & payment details | - Product listings & details<br/>- Shopping cart contents<br/>- Order confirmation & status updates | - Register / authenticate<br/>- Browse/catalog navigation<br/>- Add/remove items from cart<br/>- Checkout & pay<br/>- View order history & status |
| 2 | **Seller** | Vendor who lists and manages their own products, processes orders, and tracks shipments. | - Product data (name, description, price, inventory level, images)<br/>- Order acceptance/rejection<br/>- Shipment info (tracking ID, carrier) | - Live product listing pages<br/>- Sales & inventory reports<br/>- Order notifications & shipment confirmations | - Create/update/delete product listings<br/>- Manage stock levels<br/>- Accept/process incoming orders<br/>- Provide shipment details |
| 3 | **Admin** | Platform administrator who oversees users, content, and overall system health. | - User management actions (create/disable accounts, assign roles)<br/>- Category/taxonomy changes<br/>- Platform settings adjustments<br/>- Report queries | - System health dashboards<br/>- User & seller activity reports<br/>- Configuration confirmation messages<br/>- Audit logs | - Approve/reject seller registrations<br/>- Manage user roles & permissions<br/>- Configure global settings (tax, categories)<br/>- Monitor performance & security |
| 4 | **Gmail Service** | External SMTP/email provider used to deliver transactional and notification emails. | - Email requests from app (recipient address, subject, template data: order ID, reset token, etc.) | - Delivery status callbacks/logs (sent, bounced, delivered)<br/>- Error responses for failed sends | - Send out registration confirmations, order & shipment notifications, password‑reset emails<br/>- Handle bounces and retries |

**## 2. System Description**

**### 2.1 Product Perspective**

The SASUCare E-Commerce Platform is a standalone web application that provides a full-featured marketplace for buyers and sellers. It integrates the following key components:

- Frontend web interface built with Thymeleaf templates

- Planned TypeScript frontend for enhanced user experience

- RESTful API backend built with Spring Boot

- Database storage using Microsoft SQL Server

- Authentication services using both session-based and JWT mechanisms

- File storage system for product images and other uploads

The platform is designed to operate as an independent system but can potentially integrate with external services such as payment processors, shipping providers, and email notification systems.

**### 2.2 System Environment**

**### 2.3 Design and Implementation Constraints**

- The system must follow the MVC architectural pattern

- Backend implementation must use Spring Boot framework (version 3.1.5)

- Data access layer must use Spring Data JPA

- Frontend views must use Thymeleaf templating engine (with planned TypeScript integration)

- Security implementation must use Spring Security with role-based access control

- All passwords must be stored using BCrypt encryption

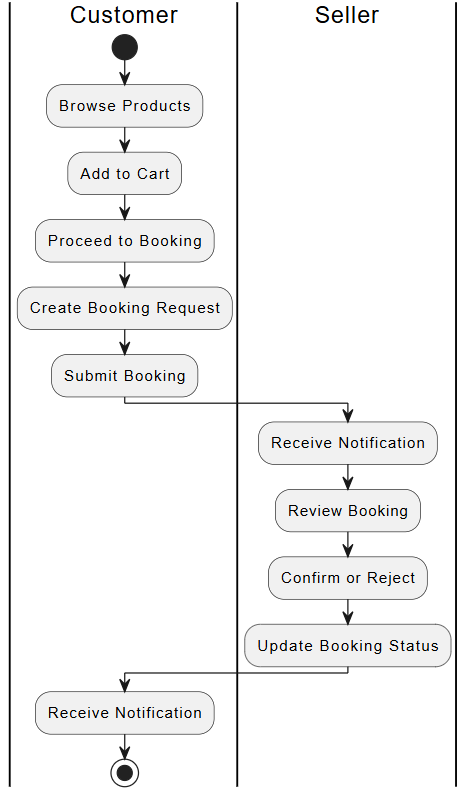
- The system must support JWT-based authentication for API access

- The system must implement proper input validation and sanitization

- The application must be containerizable for deployment flexibility

- The database schema should be manageable through JPA/Hibernate mappings

- All API endpoints must follow RESTful design principles

(SWINLANE DIAGRAM FOR CUSTOMER BOOKING)  


**### 2.4 User Documentation**

The following user documentation will be developed:

- User manuals for customers, sellers, and administrators

- Technical documentation for system administrators and developers

- Online help system integrated into the application

- API documentation for third-party integrations

- Frequently Asked Questions (FAQ) section

- Video tutorials for common operations

- Contextual help tooltips throughout the interface

**### 2.5 Assumptions and Dependencies**

### 3.1 Architectural Pattern  
  
SASUCare follows the Model-View-Controller (MVC) architectural pattern, which separates the application into three interconnected components:  
  
- \*\*Model\*\*: Represents the data and business logic  
 - Entity classes (User, Product, Order, etc.)  
 - Repository interfaces for data access  
 - Service classes for business logic  
  
- \*\*View\*\*: Handles the presentation layer  
 - Thymeleaf templates for dynamic HTML generation  
 - CSS and JavaScript for styling and client-side interactions  
  
- \*\*Controller\*\*: Manages user requests and responses  
 - Spring MVC controllers that handle HTTP requests  
 - REST controllers for API endpoints  
 - Form handling and data validation  
  
### 3.2 Component Architecture

```  
┌─────────────────────────────────────────────────────────────┐  
│ Presentation Layer │  
│ ┌─────────────┐ ┌─────────────┐ ┌─────────────────────┐ │  
│ │ Thymeleaf │ │ Static │ │ Error Handling │ │  
│ │ Templates │ │ Resources │ │ & Validation │ │  
│ └─────────────┘ └─────────────┘ └─────────────────────┘ │  
└───────────────────────────┬─────────────────────────────────┘  
 │  
┌───────────────────────────▼─────────────────────────────────┐  
│ Web Layer │  
│ ┌─────────────┐ ┌─────────────┐ ┌─────────────────────┐ │  
│ │ MVC │ │ REST API │ │ Security Filters │ │  
│ │ Controllers │ │ Controllers │ │ & Interceptors │ │  
│ └─────────────┘ └─────────────┘ └─────────────────────┘ │  
└───────────────────────────┬─────────────────────────────────┘  
 │  
┌───────────────────────────▼─────────────────────────────────┐  
│ Service Layer │  
│ ┌─────────────┐ ┌─────────────┐ ┌─────────────────────┐ │  
│ │ Business │ │ Transaction │ │ Security │ │  
│ │ Logic │ │ Management │ │ Services │ │  
│ └─────────────┘ └─────────────┘ └─────────────────────┘ │  
└───────────────────────────┬─────────────────────────────────┘  
 │  
┌───────────────────────────▼─────────────────────────────────┐  
│ Persistence Layer │  
│ ┌─────────────┐ ┌─────────────┐ ┌─────────────────────┐ │  
│ │ JPA │ │ Repository │ │ Database │ │  
│ │ Entities │ │ Interfaces │ │ Connections │ │  
│ └─────────────┘ └─────────────┘ └─────────────────────┘ │  
└───────────────────────────┬─────────────────────────────────┘  
 │  
┌───────────────────────────▼─────────────────────────────────┐  
│ Database │  
│ Microsoft SQL Server │  
└─────────────────────────────────────────────────────────────┘  
```  
  
### 3.3 Package Structure  
# Functional Requirements Document: SASUCare

## 1. Introduction

SASUCare is a web application that appears to function as an e-commerce and service booking platform. It allows users to register, browse products/services, manage carts, place orders, and make bookings. The system supports different user roles, including Customers, Vendors/Sellers who manage their own shops and offerings, and Administrators who oversee the platform. Key technologies include Spring Boot, Spring Security, JPA (Hibernate), Thymeleaf, and a SQL database.

## 2. Actors

Based on the codebase analysis (User, Role, Product models, and various controllers), the primary actors are:

1.  \*\*Anonymous User / Guest:\*\*

    \*   Any user accessing the site without logging in.

2.  \*\*Customer (Authenticated User):\*\*

    \*   A registered and logged-in user.

    \*   Likely associated with a "ROLE\_USER" or similar.

3.  \*\*Vendor / Seller (Authenticated User):\*\*

    \*   A registered and logged-in user who owns/manages a shop.

    \*   Indicated by the `shopName` attribute in the `User` model and the `seller` association in `Product` and `Booking` models.

    \*   Likely associated with a "ROLE\_VENDOR" or similar.

4.  \*\*Administrator (Authenticated User):\*\*

    \*   A privileged user responsible for system management.

    \*   Likely associated with a "ROLE\_ADMIN" or similar.

## 3. Database Schema Overview

The following is a high-level overview of the main database entities identified from the `com.sasucare.model` package.

### User Management & Authentication:

\*   \*\*`users` Table (from `User.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `email` (VARCHAR, Unique, Not Null)

    \*   `password` (VARCHAR, Not Null) - Expected to be hashed

    \*   `first\_name` (VARCHAR)

    \*   `last\_name` (VARCHAR)

    \*   `shop\_name` (VARCHAR) - For Vendor/Seller users

    \*   `is\_active` (BOOLEAN, Not Null, Default: true)

    \*   `is\_verified` (BOOLEAN, Not Null, Default: false)

    \*   `verification\_token` (VARCHAR)

    \*   `img` (VARCHAR) - Profile image URL

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`roles` Table (from `Role.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `name` (VARCHAR, Unique, Not Null) - e.g., "ROLE\_USER", "ROLE\_ADMIN", "ROLE\_VENDOR"

    \*   `description` (VARCHAR)

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`user\_roles` Table (Join Table for User-Role Many-to-Many)\*\*

    \*   `user\_id` (FK to `users.id`)

    \*   `role\_id` (FK to `roles.id`)

\*   \*\*`features` Table (from `Feature.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `name` (VARCHAR, Unique, Not Null) - User-friendly name of the feature

    \*   `description` (VARCHAR, Not Null)

    \*   `feature\_key` (VARCHAR, Unique, Not Null) - Programmatic key (e.g., "MANAGE\_USERS", "CREATE\_PRODUCT")

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`role\_features` Table (Join Table for Role-Feature Many-to-Many)\*\*

    \*   `role\_id` (FK to `roles.id`)

    \*   `feature\_id` (FK to `features.id`)

### Product & Catalog Management:

\*   \*\*`products` Table (from `Product.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `name` (VARCHAR, Not Null)

    \*   `description` (TEXT)

    \*   `price` (DECIMAL(10,2), Not Null)

    \*   `stock\_quantity` (INT, Not Null)

    \*   `sku` (VARCHAR)

    \*   `status` (VARCHAR, Not Null) - e.g., "ACTIVE", "INACTIVE"

    \*   `primary\_image\_url` (VARCHAR)

    \*   `version` (BIGINT) - For optimistic locking

    \*   `seller\_id` (FK to `users.id`, Not Null)

    \*   `category\_id` (FK to `categories.id`)

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`categories` Table (from `Category.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `name` (VARCHAR, Unique, Not Null)

    \*   `description` (VARCHAR)

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`product\_images` Table (from `ProductImage.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `product\_id` (FK to `products.id`, Not Null)

    \*   `image\_url` (VARCHAR)

    \*   `is\_primary` (BOOLEAN, Not Null, Default: false)

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

### Order Management:

\*   \*\*`orders` Table (from `Order.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `customer\_id` (FK to `users.id`, Not Null)

    \*   `shipping\_address\_id` (FK to `addresses.id`, Not Null)

    \*   `total\_amount` (DECIMAL(10,2), Not Null)

    \*   `shipping\_cost` (DECIMAL(10,2), Not Null)

    \*   `order\_status` (VARCHAR, Not Null) - e.g., "PENDING", "PAID", "SHIPPED", "DELIVERED", "CANCELLED"

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`order\_items` Table (from `OrderItem.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `order\_id` (FK to `orders.id`, Not Null)

    \*   `product\_id` (FK to `products.id`, Not Null)

    \*   `seller\_id` (FK to `users.id`, Not Null)

    \*   `quantity` (INT, Not Null)

    \*   `price\_at\_purchase` (DECIMAL(10,2), Not Null)

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`addresses` Table (from `Address.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `user\_id` (FK to `users.id`, Not Null)

    \*   `street` (VARCHAR, Not Null)

    \*   `street\_address` (VARCHAR, Not Null)

    \*   `city` (VARCHAR)

    \*   `state` (VARCHAR)

    \*   `country` (VARCHAR)

    \*   `postal\_code` (VARCHAR, Not Null)

    \*   `is\_default` (BOOLEAN, Not Null, Default: false)

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`salecode` Table (from `SaleCode.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `code` (VARCHAR, Unique, Not Null)

    \*   `discount\_percent` (DECIMAL(5,2), Not Null)

    \*   `start\_date` (TIMESTAMP, Not Null)

    \*   `end\_date` (TIMESTAMP, Not Null)

    \*   `quantity` (INT, Not Null) - Number of available uses

    \*   `user\_id` (FK to `users.id`, Not Null) - The user (admin/vendor) who created the code

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

\*   \*\*`order\_salecode` Table (Join Table for Order-SaleCode Many-to-Many)\*\*

    \*   `order\_id` (FK to `orders.id`)

    \*   `salecode\_id` (FK to `salecode.id`)

### Booking Management:

\*   \*\*`bookings` Table (from `Booking.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `booking\_number` (VARCHAR, Unique, Not Null)

    \*   `customer\_id` (FK to `users.id`, Not Null)

    \*   `seller\_id` (FK to `users.id`, Not Null) - The vendor providing the service/booking

    \*   `shipping\_address\_id` (FK to `addresses.id`, Not Null) - Could be service location

    \*   `total\_amount` (DECIMAL(10,2), Not Null)

    \*   `shipping\_cost` (DECIMAL(10,2), Not Null) - Could be service fees

    \*   `booking\_status` (VARCHAR, Not Null) - e.g., "PENDING", "CONFIRMED", "COMPLETED", "CANCELLED", "REJECTED"

    \*   `rejection\_reason` (VARCHAR)

    \*   `special\_instructions` (VARCHAR(500))

    \*   `created\_at` (TIMESTAMP, Not Null)

    \*   `updated\_at` (TIMESTAMP, Not Null)

    \*   `confirmed\_at` (TIMESTAMP)

    \*   `completed\_at` (TIMESTAMP)

\*   \*\*`booking\_items` Table (from `BookingItem.java`)\*\*

    \*   `id` (PK, BIGINT, Auto-increment)

    \*   `booking\_id` (FK to `bookings.id`, Not Null)

    \*   `product\_id` (FK to `products.id`, Not Null) - The product/service being booked

    \*   `quantity` (INT, Not Null)

    \*   `price` (DECIMAL(10,2), Not Null) - Price at time of booking

    \*   `product\_name` (VARCHAR, Not Null) - Snapshot of product name

    \*   `product\_details` (VARCHAR(1000)) - Snapshot of product details

\*\*Note:\*\* `ShoppingCart.java` and `CartItem.java` are session-scoped beans and are not directly persisted as database entities. Cart data is transient until an order is created.

## 4. Business Flows & Use Case Descriptions

This section outlines the key business flows and use cases for each actor. Controllers like `HomeController`, `AuthController`, `RegistrationController`, `ProductController`, `CartController`, `AccountController`, `CustomerBookingController`, `SellerController`, `SellerOrderController`, `SellerBookingController`, `AdminController`, `AdminOrderController`, and `SearchController` define these interactions.

### 4.1. Common / Public Flows (Anonymous & Authenticated Users)

\*   \*\*UC-001: View Homepage\*\*

    \*   \*\*Actor:\*\* Anonymous User, Customer, Vendor, Admin

    \*   \*\*Description:\*\* User navigates to the homepage. The system displays featured products, categories, promotions, etc.

    \*   \*\*Controller(s):\*\* `HomeController`

\*   \*\*UC-002: Browse Products/Services\*\*

    \*   \*\*Actor:\*\* Anonymous User, Customer, Vendor, Admin

    \*   \*\*Description:\*\* User browses products/services by category, searches, or views lists.

    \*   \*\*Controller(s):\*\* `ProductController`, `SearchController`

\*   \*\*UC-003: View Product/Service Details\*\*

    \*   \*\*Actor:\*\* Anonymous User, Customer, Vendor, Admin

    \*   \*\*Description:\*\* User selects a product/service to view its details (description, price, images, seller info, stock/availability).

    \*   \*\*Controller(s):\*\* `ProductController`

\*   \*\*UC-004: Search Products/Services\*\*

    \*   \*\*Actor:\*\* Anonymous User, Customer, Vendor, Admin

    \*   \*\*Description:\*\* User enters keywords to search for products/services. System displays matching results.

    \*   \*\*Controller(s):\*\* `SearchController`

### 4.2. Authentication & Registration Flows

\*   \*\*UC-101: User Registration\*\*

    \*   \*\*Actor:\*\* Anonymous User

    \*   \*\*Description:\*\* User provides necessary details (email, password, name, potentially shop name if registering as vendor) to create an account. System validates input, creates a user record (potentially inactive/unverified), and may send a verification email.

    \*   \*\*Controller(s):\*\* `RegistrationController`

    \*   \*\*Postcondition:\*\* User account created. User might need to verify email.

\*   \*\*UC-102: User Login\*\*

    \*   \*\*Actor:\*\* Anonymous User (intending to become Customer, Vendor, or Admin)

    \*   \*\*Description:\*\* User provides email and password. System validates credentials and establishes a session.

    \*   \*\*Controller(s):\*\* `AuthController` (likely handles POST for login), Spring Security

    \*   \*\*Postcondition:\*\* User is logged in and redirected to their dashboard or homepage.

\*   \*\*UC-103: User Logout\*\*

    \*   \*\*Actor:\*\* Customer, Vendor, Admin

    \*   \*\*Description:\*\* User chooses to log out. System invalidates the session.

    \*   \*\*Controller(s):\*\* `AuthController`, Spring Security

    \*   \*\*Postcondition:\*\* User is logged out and redirected to the homepage.

\*   \*\*UC-104: Email Verification\*\*

    \*   \*\*Actor:\*\* Newly Registered User

    \*   \*\*Description:\*\* User clicks a verification link sent to their email. System verifies the token and activates the account.

    \*   \*\*Controller(s):\*\* `RegistrationController` (or a dedicated verification controller)

\*   \*\*UC-105: Forgot/Reset Password (Assumed)\*\*

    \*   \*\*Actor:\*\* User (Customer, Vendor, Admin)

    \*   \*\*Description:\*\* User requests a password reset. System sends a reset link/code. User follows instructions to set a new password.

    \*   \*\*Controller(s):\*\* Likely a dedicated controller or part of `AccountController`/`AuthController`. (This is an assumption as no specific controller was immediately obvious for this).

### 4.3. Customer Flows

\*   \*\*UC-201: Manage Account Profile\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer views and updates their profile information (name, contact, password, profile image).

    \*   \*\*Controller(s):\*\* `AccountController`

\*   \*\*UC-202: Manage Addresses\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer adds, views, edits, and deletes their shipping addresses. Sets a default address.

    \*   \*\*Controller(s):\*\* `AccountController` (or a dedicated `AddressController`)

\*   \*\*UC-203: Add Product to Cart\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer selects a product and quantity and adds it to their shopping cart. System updates the session-based cart.

    \*   \*\*Controller(s):\*\* `CartController`

\*   \*\*UC-204: View Shopping Cart\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer views the items in their cart, quantities, and total price.

    \*   \*\*Controller(s):\*\* `CartController`

\*   \*\*UC-205: Update Cart Item Quantity\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer changes the quantity of an item in the cart. System updates the cart.

    \*   \*\*Controller(s):\*\* `CartController`

\*   \*\*UC-206: Remove Product from Cart\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer removes an item from their cart. System updates the cart.

    \*   \*\*Controller(s):\*\* `CartController`

\*   \*\*UC-207: Checkout and Place Order\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer proceeds to checkout from the cart. Selects/confirms shipping address, applies sale codes (if any), reviews order, and confirms. System creates an `Order` and `OrderItem` records, potentially processes payment (integration not detailed), and updates stock.

    \*   \*\*Controller(s):\*\* `CartController`, potentially an `OrderController` (logic might be split or within `CartController` for checkout)

    \*   \*\*Postcondition:\*\* Order is placed with "PENDING" or "PAID" status. Cart is cleared.

\*   \*\*UC-208: View Order History\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer views a list of their past and current orders and their statuses.

    \*   \*\*Controller(s):\*\* `AccountController` (or a dedicated `OrderController` for customers)

\*   \*\*UC-209: View Order Details\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer views the details of a specific order (items, prices, shipping, status).

    \*   \*\*Controller(s):\*\* `AccountController` (or dedicated `OrderController`)

\*   \*\*UC-210: Request Booking for a Service/Product\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer selects a bookable product/service, chooses date/time (if applicable), provides details, and submits a booking request.

    \*   \*\*Controller(s):\*\* `CustomerBookingController`

    \*   \*\*Postcondition:\*\* `Booking` record created with "PENDING" status.

\*   \*\*UC-211: View Booking History & Status\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer views their past and current bookings and their statuses (Pending, Confirmed, Cancelled, Completed).

    \*   \*\*Controller(s):\*\* `CustomerBookingController`

\*   \*\*UC-212: Cancel Booking (if allowed by status)\*\*

    \*   \*\*Actor:\*\* Customer

    \*   \*\*Description:\*\* Customer cancels a pending or confirmed booking. System updates booking status.

    \*   \*\*Controller(s):\*\* `CustomerBookingController`

### 4.4. Vendor / Seller Flows

\*   \*\*UC-301: Manage Seller Profile / Shop Information\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor updates their shop name, description, contact details, and other shop-specific settings.

    \*   \*\*Controller(s):\*\* `SellerController`, `AccountController`

\*   \*\*UC-302: Manage Products (CRUD)\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor adds new products/services, edits existing ones (details, price, stock, images, category), and deletes products.

    \*   \*\*Controller(s):\*\* `SellerController` (likely handles product CRUD for vendors)

\*   \*\*UC-303: View Own Product Listings\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor views a list of all products/services they offer.

    \*   \*\*Controller(s):\*\* `SellerController`

\*   \*\*UC-304: Manage Product Images\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor uploads, updates, and deletes images for their products. Sets a primary image.

    \*   \*\*Controller(s):\*\* `SellerController`, `FileController`

\*   \*\*UC-305: Manage Orders Received\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor views orders containing their products. Updates order item statuses (e.g., "PROCESSING", "SHIPPED").

    \*   \*\*Controller(s):\*\* `SellerOrderController`

\*   \*\*UC-306: View Order Item Details (for their products)\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor views details of specific order items related to their products.

    \*   \*\*Controller(s):\*\* `SellerOrderController`

\*   \*\*UC-307: Manage Bookings Received\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor views incoming booking requests for their services/products.

    \*   \*\*Controller(s):\*\* `SellerBookingController`

\*   \*\*UC-308: Confirm/Reject Booking Request\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor reviews a booking request and confirms or rejects it (providing a reason for rejection). System updates booking status and notifies customer.

    \*   \*\*Controller(s):\*\* `SellerBookingController`

\*   \*\*UC-309: Mark Booking as Completed\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor marks a confirmed booking as completed after service delivery.

    \*   \*\*Controller(s):\*\* `SellerBookingController`

\*   \*\*UC-310: View Seller Dashboard/Reports (Assumed)\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor views sales summaries, popular products, booking statistics, etc.

    \*   \*\*Controller(s):\*\* `SellerController`

\*   \*\*UC-311: Manage Sale Codes (for their products/shop - if applicable)\*\*

    \*   \*\*Actor:\*\* Vendor

    \*   \*\*Description:\*\* Vendor creates, views, updates, and deactivates sale codes applicable to their shop or products.

    \*   \*\*Controller(s):\*\* `SellerController` or a shared `SaleCodeController`. The `SaleCode` model has a `user\_id` which could be the vendor.

### 4.5. Administrator Flows

\*   \*\*UC-401: Manage Users (CRUD)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin views, creates, edits (details, roles, active status), and deletes user accounts.

    \*   \*\*Controller(s):\*\* `AdminController`

\*   \*\*UC-402: Manage Roles (CRUD)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin creates, views, edits, and deletes user roles. Assigns features to roles.

    \*   \*\*Controller(s):\*\* `AdminController` (or dedicated `RoleController`)

\*   \*\*UC-403: Manage Features (CRUD)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin defines and manages system features that can be assigned to roles for authorization.

    \*   \*\*Controller(s):\*\* `AdminController` (or dedicated `FeatureController`)

\*   \*\*UC-404: Manage Categories (CRUD)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin creates, views, edits, and deletes product/service categories.

    \*   \*\*Controller(s):\*\* `AdminController`

\*   \*\*UC-405: Manage All Products (View, Edit, Delete - Oversee)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin can view all products on the platform, edit details, and remove inappropriate or problematic listings.

    \*   \*\*Controller(s):\*\* `AdminController` (or a dedicated `AdminProductController`)

\*   \*\*UC-406: Manage All Orders (View, Update Status - Oversee)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin views all orders in the system, can update order statuses, and handle escalations or issues.

    \*   \*\*Controller(s):\*\* `AdminOrderController`

\*   \*\*UC-407: Manage All Bookings (View, Update Status - Oversee)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin views all bookings, can update statuses, and manage booking-related issues.

    \*   \*\*Controller(s):\*\* `AdminController` (or a dedicated `AdminBookingController`)

\*   \*\*UC-408: Manage Platform-Wide Sale Codes\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin creates, views, updates, and deactivates platform-wide sale codes.

    \*   \*\*Controller(s):\*\* `AdminController` or a shared `SaleCodeController`.

\*   \*\*UC-409: View System Reports/Analytics (Assumed)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin views overall platform statistics (sales, user registrations, popular items, etc.).

    \*   \*\*Controller(s):\*\* `AdminController`

\*   \*\*UC-410: Manage Site Settings (Assumed)\*\*

    \*   \*\*Actor:\*\* Admin

    \*   \*\*Description:\*\* Admin configures global settings for the application (e.g., payment gateways, shipping options, email templates).

    \*   \*\*Controller(s):\*\* `AdminController`

## 5. Screen Authorization (Role-Based Access Control using Features)

Access to screens and functionalities will be controlled by assigning `Feature` entities to `Role` entities. Users are assigned `Role`s. Spring Security, likely with method security annotations (`@PreAuthorize`) or controller-level annotations, will enforce these rules based on the `featureKey` associated with a user's roles.

\*\*Example Screen/Action to Role/Feature Mapping:\*\*

\*\*Authorization Implementation Notes:\*\*

\*   Thymeleaf Security Extras (`thymeleaf-extras-springsecurity6`) will be used for conditional rendering in templates based on roles/authorities/features.

\*   Spring Security annotations (`@PreAuthorize("hasAuthority('FEATURE\_KEY')")` or `@PreAuthorize("hasRole('ROLE\_ADMIN') and hasAuthority('FEATURE\_KEY')")`) will protect service methods and controller endpoints.

\*   The `Feature` entity's `featureKey` should be used as the authority string in Spring Security configurations.

\*   A custom `UserDetailsService` will load user details along with their roles and associated features (authorities).

This document provides a comprehensive starting point based on the codebase analysis. Further details for each use case (preconditions, postconditions, detailed steps, error handling) would typically be elaborated in a full Software Requirements Specification.