Table of Contents

# 1 SHOPSPHERE - E-COMMERCE APPLICATION

## 1.1 Project Documentation

# 2 VISION AND MISSION OF THE INSTITUTE

## 2.1 VISION

Carving the youth as dynamic, complement, valued and knowledgeable professionals who shall lead the nation to a better future.

## 2.2 MISSION

* Providing quality education, Student – Centered teaching – Learning process and state - of – art infrastructure for professional aspirants hailing from both rural and urban areas.
* Impacting technical and management education to encourage independent thinking, develop strong domain of knowledge, own contemporary skills and positive attitudes towards holistic growth of young minds.
* Evolving institution into a center of excellence and research.

## 2.3 QUALITY POLICIES

Sri Venkateswara College Of Engineering And Technology strides towards excellence by adopting a system of quality policies and processes with continued improvements to enhance student skills and talent for their exemplary contribution to the society, the nation and the world.

# 3 VISION AND MISSION OF THE DEPARTMENT

## 3.1 Under R20 Regulations

### 3.1.1 VISION:

* To achieve excellent standard of quality education by using latest tools in Artificial Intelligence and disseminating innovations to relevant areas.

### 3.1.2 MISSION

* To develop professionals who are skilled in Artificial Intelligence.
* Impact rigorous training to generate knowledge through the state -of -the – art concepts and technologies in Artificial Intelligence.
* Establish centers of excellence in leading areas of computing and artificial intelligence to inculcate strong commitment to work with a commitment to the progress of the nation.

### 3.1.3 PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

**PEO1:** To be able to solve wide range of computing related problems to cater to the needs of industry and society.

**PEO2:** Enable students to build intelligent machines and applications with a cutting – edge combinations of machine learning, analytics, and visualization.

**PEO3:** Produce graduates having professional competence through life-long learning such as advanced degrees, professional skills and other professional activities related globally to engineering & society.

### 3.1.4 PROGRAM SPECIFIC OUTCOMES (PSOs):

**PSO1:** Should have an ability to apply technical knowledge and usage of modern hardware and software tools related AI for solving real world problems.

**PSO2:** Should have the capability to develop many successful applications based on machine learning methods, AI methods in different fields, including neural networks, signal processing, and data mining.

# 4 CERTIFICATE

*This certifies that the project work titled “ShopSphere - E-Commerce Application with Spring Boot & MongoDB” has been successfully completed as part of the internship program*

# 5 ACKNOWLEDGEMENT

A grateful thanks to **Dr. Ravuri Venkata Swamy**, Chairman of Sri Venkateswara College of Engineering & Technology for providing education in their esteemed institution.

I wish to record my deep sense of gratitude and profound thanks to our beloved Vice Chairman, **Sri Ravuri Venkata Srinivas** for his valuable support throughout the course.

I express our sincere thanks to **Dr. M. MOHAN BABU**, our beloved principal for his encouragement and suggestion during the course of study.

With the deep sense of gratefulness, I acknowledge **Dr. M. LAVANYA**, Head of the Department, Computer Science Engineering (AI), for giving us inspiring guidance in undertaking this internship.

I express our sincere thanks to the **Dr. K. ANJANEYULU**, internship coordinator, for his keen interest, stimulating guidance, constant encouragement with our work during all stages, to bring this report into fruition.

Finally, I would like to express my sincere thanks to all teaching, non-teaching faculty members, our parents, friends and for all those who have supported us to complete the internship successfully.

**Name:** Ramisetty Divyasree  
**Roll No:** 22781A3137

# 6 ABSTRACT

Java is a versatile, object-oriented programming language widely used in developing platform-independent applications. Known for its “Write Once, Run Anywhere” capability, Java has become a cornerstone for building scalable, secure, and high-performance applications.

ShopSphere is a comprehensive E-Commerce application built using Spring Boot and MongoDB that demonstrates enterprise-level software development practices. This project implements a complete shopping platform with both CLI (Command Line Interface) and REST API capabilities, featuring product management, user authentication, shopping cart functionality, and order processing.

The application leverages Spring Boot’s powerful framework for rapid development, MongoDB for flexible NoSQL data storage, and Swagger for comprehensive API documentation. Through its implementation of CRUD operations, user management, cart handling, and order workflows, ShopSphere showcases modern software engineering principles including MVC architecture, RESTful API design, and database integration.

This project serves as a practical demonstration of building scalable, maintainable e-commerce systems using Java ecosystem technologies, preparing students for real-world enterprise application development challenges.

# 7 TABLE OF CONTENTS

## 7.1 CHAPTER 1: PROJECT OVERVIEW

* Introduction
* Project Features
* Technologies Used
* System Architecture
* Prerequisites

## 7.2 CHAPTER 2: PROJECT STRUCTURE AND DESIGN

* Project Structure
* Database Schema Design
* Model Classes
* Repository Layer
* Service Layer
* Controller Layer

## 7.3 CHAPTER 3: IMPLEMENTATION

* Application Setup
* Configuration Files
* Core Modules Implementation
* Product Management Module
* User Management Module
* Shopping Cart Module
* Order Management Module

## 7.4 CHAPTER 4: FUNCTIONALITY AND FEATURES

* CLI Interface
* REST API Endpoints
* Swagger Documentation
* Data Validation
* Error Handling

## 7.5 CHAPTER 5: TESTING AND DEPLOYMENT

* Running the Application
* Testing Procedures
* Sample Inputs and Outputs
* Troubleshooting

## 7.6 CHAPTER 6: CONCLUSION AND FUTURE SCOPE

* Conclusion
* Lessons Learned
* Future Enhancements
* References

# 8 WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES

## 8.1 WEEK - 1: Java Environment Setup and Core Fundamentals

| DAY | DATE | TOPIC/CONCEPT COVERED |
| --- | --- | --- |
| Saturday | 05/7/25 | Windows Setup and Introduction to Java |
| Monday | 07/7/25 | Linux Setup and Your First Java App |
| Tuesday | 08/7/25 | Variables and Data Types |
| Wednesday | 09/7/25 | Taking User Input and Operators |
| Thursday | 10/7/25 | Conditionals and Overview |
| Friday | 11/7/25 | Iterators and Arrays |

## 8.2 WEEK - 2: Intermediate Java and OOP Concepts

| DAY | DATE | TOPIC/CONCEPT COVERED |
| --- | --- | --- |
| Saturday | 12/7/25 | Strings |
| Monday | 14/7/25 | Methods |
| Tuesday | 15/7/25 | Exception Handling and Overview |
| Wednesday | 16/7/25 | Classes and Objects |
| Thursday | 17/7/25 | Constructors, Getter and Setter |
| Friday | 18/7/25 | Inheritance |

## 8.3 WEEK - 3: Advanced OOP, Collections, and File I/O

| DAY | DATE | TOPIC/CONCEPT COVERED |
| --- | --- | --- |
| Saturday | 19/7/25 | Polymorphism |
| Monday | 21/7/25 | Abstract keyword |
| Tuesday | 22/7/25 | Interface |
| Wednesday | 23/7/25 | Static Keyword |
| Thursday | 24/7/25 | Collections |
| Friday | 25/7/25 | File Handling |

## 8.4 WEEK - 4: Introduction to Spring Boot and MongoDB

| DAY | DATE | TOPIC/CONCEPT COVERED |
| --- | --- | --- |
| Saturday | 26/7/25 | Introduction to Spring Boot Framework |
| Monday | 28/7/25 | Spring Boot Project Structure |
| Tuesday | 29/7/25 | MongoDB Database Setup |
| Wednesday | 30/7/25 | Spring Data MongoDB Integration |
| Thursday | 31/7/25 | REST API Development |
| Friday | 01/8/25 | Swagger API Documentation |

# 9 MONTH 2 (02 Aug – 05 Sep 2025) – PROJECT PHASE

## 9.1 Project Title: ShopSphere - E-Commerce Application with Spring Boot & MongoDB

### 9.1.1 Activities:

* Requirement Analysis and System Design
* Database Schema Design (MongoDB Collections)
* Implementation of Product Management Module
* Implementation of User Management Module
* Implementation of Shopping Cart Module
* Implementation of Order Processing Module
* CRUD Operations Integration with REST APIs
* Swagger API Documentation Setup
* CLI Interface Development
* Testing and Debugging
* Report Preparation & Documentation

# 10 CHAPTER 1: PROJECT OVERVIEW

## 10.1 Introduction

ShopSphere is a modern, full-featured e-commerce application that provides both a Command Line Interface (CLI) and RESTful API capabilities. Built using Spring Boot 3.1.5 and MongoDB, this project demonstrates enterprise-level application development practices suitable for real-world deployment.

The application addresses the common requirements of online shopping platforms by implementing user authentication, product catalog management, shopping cart functionality, and complete order processing workflows. By leveraging the Spring Boot ecosystem and MongoDB’s flexible document-based storage, ShopSphere provides a scalable foundation for e-commerce operations.

This project showcases modern software development practices including:

* Model-View-Controller (MVC) architecture
* RESTful API design principles
* NoSQL database integration
* Automated API documentation with Swagger
* Input validation and error handling
* Separation of concerns through layered architecture

## 10.2 Project Features

### 10.2.1 1. Product Management

* Complete CRUD operations (Create, Read, Update, Delete)
* Product categorization (Electronics, Clothing, Books, Home Appliances, Sports & Fitness)
* Advanced search by name (case-insensitive)
* Category-based filtering
* Real-time stock management
* Price validation and constraints
* Product availability tracking

### 10.2.2 2. User Management

* User registration with comprehensive validation
* Email uniqueness constraint
* Secure login authentication
* User profile management
* Address tracking for delivery
* User role management

### 10.2.3 3. Shopping Cart

* Add products with quantity selection
* Remove items from cart
* Update item quantities
* Real-time price calculation
* Stock validation before adding items
* Persistent cart storage
* Clear cart functionality
* Cart summary with total amount

### 10.2.4 4. Order Management

* Place orders from shopping cart
* Automated stock reduction
* Order history tracking
* Itemized order details
* Payment status tracking (PAID, PENDING, FAILED)
* Simulated payment processing
* Order confirmation

### 10.2.5 5. REST API & Swagger Documentation

* Comprehensive RESTful API endpoints
* Interactive Swagger UI for API testing
* Automatic API documentation generation
* Request/Response examples
* HTTP status code handling

### 10.2.6 6. Command Line Interface

* Interactive menu-driven CLI
* User-friendly navigation
* Real-time data display
* Input validation
* Error handling and messages

## 10.3 Technologies Used

### 10.3.1 1. Backend Framework

**Spring Boot 3.1.5** \* Spring Boot Starter Web \* Spring Boot Starter Data MongoDB \* Spring Boot Auto-configuration

### 10.3.2 2. Database

**MongoDB (NoSQL Database)** \* Flexible document-based storage \* Embedded MongoDB support \* MongoDB Java Driver

### 10.3.3 3. API Documentation

**Springdoc OpenAPI 2.2.0** \* Swagger UI \* OpenAPI 3.0 Specification \* Interactive API Documentation

### 10.3.4 4. Build Tool

**Apache Maven** \* Dependency management \* Build automation \* Project lifecycle management

### 10.3.5 5. Java Version

**Java 17 (LTS)** \* Modern language features \* Enhanced performance \* Long-term support

### 10.3.6 6. Additional Libraries

**Lombok (Optional)** \* Reduces boilerplate code \* Auto-generates getters/setters \* Builder pattern support

## 10.4 System Architecture

ShopSphere follows a layered architecture pattern:

### 10.4.1 1. Presentation Layer

* CLI Interface (ECommerceCLI.java)
* REST Controllers (Controller package)
  + ProductController
  + UserController
  + CartController
  + OrderController

### 10.4.2 2. Service Layer

* Business logic implementation
* Service classes (Service package)
  + ProductService
  + UserService
  + CartService
  + OrderService

### 10.4.3 3. Repository Layer

* Data access layer
* MongoDB repositories (Repository package)
  + ProductRepository
  + UserRepository
  + CartRepository
  + OrderRepository

### 10.4.4 4. Model Layer

* Domain entities (Model package)
  + Product
  + User
  + Cart & CartItem
  + Order & OrderItem

### 10.4.5 5. Utility Layer

* Helper classes (Util package)
  + IdGenerator
  + InputValidator

### 10.4.6 6. Configuration Layer

* Application configuration
  + SwaggerConfig
  + application.properties

## 10.5 Prerequisites

### 10.5.1 Software Requirements:

* Java Development Kit (JDK) 17 or higher
* MongoDB Server 4.0 or higher
* Apache Maven 3.6 or higher
* Code Editor/IDE (IntelliJ IDEA, Eclipse, or VS Code)
* Windows 10/11, macOS, or Linux operating system
* Git (for version control)
* Postman (optional, for API testing)

### 10.5.2 Hardware Requirements:

* Processor: Intel i3 or higher / AMD equivalent
* RAM: 8 GB or more (recommended)
* Storage: 500 MB free disk space
* Internet connection for downloading dependencies

### 10.5.3 Knowledge Prerequisites:

* Core Java programming concepts
* Object-Oriented Programming (OOP)
* Basic understanding of databases
* Spring Boot framework basics
* RESTful API concepts
* Maven build tool
* Command line operations

# 11 CHAPTER 2: PROJECT STRUCTURE AND DESIGN

## 11.1 Project Structure

Shopsphere-ECommerce-App/  
│  
├── src/  
│ └── main/  
│ ├── java/  
│ │ └── com/  
│ │ └── ecommerce/  
│ │ ├── ECommerceApplication.java # Main application entry point  
│ │ ├── ECommerceCLI.java # CLI interface  
│ │ │  
│ │ ├── config/  
│ │ │ └── SwaggerConfig.java # Swagger configuration  
│ │ │  
│ │ ├── controller/ # REST API Controllers  
│ │ │ ├── CartController.java  
│ │ │ ├── OrderController.java  
│ │ │ ├── ProductController.java  
│ │ │ └── UserController.java  
│ │ │  
│ │ ├── model/ # Domain models  
│ │ │ ├── Cart.java  
│ │ │ ├── CartItem.java  
│ │ │ ├── Order.java  
│ │ │ ├── OrderItem.java  
│ │ │ ├── Product.java  
│ │ │ └── User.java  
│ │ │  
│ │ ├── repository/ # Data access layer  
│ │ │ ├── CartRepository.java  
│ │ │ ├── OrderRepository.java  
│ │ │ ├── ProductRepository.java  
│ │ │ └── UserRepository.java  
│ │ │  
│ │ ├── service/ # Business logic layer  
│ │ │ ├── CartService.java  
│ │ │ ├── OrderService.java  
│ │ │ ├── ProductService.java  
│ │ │ └── UserService.java  
│ │ │  
│ │ └── util/ # Utility classes  
│ │ ├── IdGenerator.java  
│ │ └── InputValidator.java  
│ │  
│ └── resources/  
│ └── application.properties # Application configuration  
│  
├── pom.xml # Maven configuration  
├── README.md # Project documentation  
├── MONGODB\_GUIDE.md # MongoDB setup guide  
├── mongodb-insert-data.js # Sample data script  
└── mongodb-quick-setup.bat # Quick setup script

## 11.2 Database Schema Design

ShopSphere uses MongoDB with four main collections:

### 11.2.1 1. Products Collection

{  
 "\_id": ObjectId,  
 "productId": "P001",  
 "name": "Laptop",  
 "description": "High-performance laptop",  
 "price": 899.99,  
 "category": "Electronics",  
 "stock": 50  
}

### 11.2.2 2. Users Collection

{  
 "\_id": ObjectId,  
 "userId": "U001",  
 "username": "john\_doe",  
 "email": "john@example.com",  
 "password": "hashed\_password",  
 "address": "123 Main St, City"  
}

### 11.2.3 3. Carts Collection

{  
 "\_id": ObjectId,  
 "cartId": "CART001",  
 "userId": "U001",  
 "items": [  
 {  
 "productId": "P001",  
 "productName": "Laptop",  
 "quantity": 1,  
 "price": 899.99,  
 "subtotal": 899.99  
 }  
 ],  
 "totalAmount": 899.99  
}

### 11.2.4 4. Orders Collection

{  
 "\_id": ObjectId,  
 "orderId": "ORD001",  
 "userId": "U001",  
 "items": [  
 {  
 "productId": "P001",  
 "productName": "Laptop",  
 "quantity": 1,  
 "price": 899.99,  
 "subtotal": 899.99  
 }  
 ],  
 "totalAmount": 899.99,  
 "orderDate": "2025-10-31T10:30:00",  
 "paymentStatus": "PAID"  
}

## 11.3 Model Classes

### 11.3.1 1. Product.java

* Represents product entities
* Fields: productId, name, description, price, category, stock
* Annotations: @Document, @Id, validation annotations

### 11.3.2 2. User.java

* Represents user entities
* Fields: userId, username, email, password, address
* Includes email validation

### 11.3.3 3. Cart.java & CartItem.java

* Cart represents shopping cart
* CartItem represents individual items in cart
* Automatic total calculation

### 11.3.4 4. Order.java & OrderItem.java

* Order represents customer orders
* OrderItem represents items in orders
* Includes payment status and order date

## 11.4 Repository Layer

Repositories extend MongoRepository interface:

### 11.4.1 1. ProductRepository

* Custom query methods for product search
* Category filtering
* Name-based search (case-insensitive)

### 11.4.2 2. UserRepository

* Find user by email
* Find user by username
* Email uniqueness validation

### 11.4.3 3. CartRepository

* Find cart by userId
* Cart persistence

### 11.4.4 4. OrderRepository

* Find orders by userId
* Order history retrieval

## 11.5 Service Layer

Service classes contain business logic:

### 11.5.1 1. ProductService

* Add/Update/Delete products
* Search and filter products
* Stock management
* Validation logic

### 11.5.2 2. UserService

* User registration
* Login authentication
* Profile management
* Email validation

### 11.5.3 3. CartService

* Add items to cart
* Update quantities
* Remove items
* Calculate totals
* Stock validation

### 11.5.4 4. OrderService

* Place orders
* Process payments
* Update stock
* Order history

## 11.6 Controller Layer

REST API endpoints:

### 11.6.1 1. ProductController

* GET /api/products - List all products
* GET /api/products/{id} - Get product by ID
* POST /api/products - Add new product
* PUT /api/products/{id} - Update product
* DELETE /api/products/{id} - Delete product
* GET /api/products/search - Search products

### 11.6.2 2. UserController

* POST /api/users/register - Register user
* POST /api/users/login - Login user
* GET /api/users/{id} - Get user profile

### 11.6.3 3. CartController

* GET /api/cart/{userId} - Get user’s cart
* POST /api/cart/add - Add item to cart
* PUT /api/cart/update - Update cart item
* DELETE /api/cart/remove - Remove item
* DELETE /api/cart/clear - Clear cart

### 11.6.4 4. OrderController

* POST /api/orders/place - Place order
* GET /api/orders/{userId} - Get order history
* GET /api/orders/details/{orderId} - Get order details

# 12 CHAPTER 3: IMPLEMENTATION

## 12.1 Application Setup

### 12.1.1 Step 1: MongoDB Installation and Setup

#### 12.1.1.1 1. Download MongoDB Community Server

* Visit: https://www.mongodb.com/try/download/community
* Select your operating system
* Download and install

#### 12.1.1.2 2. Start MongoDB Service

**Windows:** \* MongoDB should start automatically as a service \* Or use: net start MongoDB

**Linux/Mac:** \* sudo systemctl start mongod \* Or: brew services start mongodb-community

#### 12.1.1.3 3. Verify MongoDB is running:

* Open command prompt/terminal
* Type: mongo
* You should see MongoDB shell

#### 12.1.1.4 4. Create Database (Optional - Auto-created on first use):

* use shopsphere\_db

### 12.1.2 Step 2: Java Development Kit (JDK) Installation

#### 12.1.2.1 1. Download JDK 17

* Visit: https://www.oracle.com/java/technologies/downloads/
* Download JDK 17 for your OS
* Install following wizard instructions

#### 12.1.2.2 2. Set Environment Variables:

**Windows:** \* JAVA\_HOME = C:Files \* Add to PATH: %JAVA\_HOME%

**Linux/Mac:** \* Add to ~/.bashrc or ~/.zshrc: \* export JAVA\_HOME=/path/to/jdk-17 \* export PATH=$JAVA\_HOME/bin:$PATH

#### 12.1.2.3 3. Verify Installation:

* java -version
* Should show Java version 17

### 12.1.3 Step 3: Apache Maven Installation

#### 12.1.3.1 1. Download Maven

* Visit: https://maven.apache.org/download.cgi
* Download Binary zip archive

#### 12.1.3.2 2. Extract and Setup:

* Extract to C:Files(Windows)
* Or /usr/local/maven (Linux/Mac)

#### 12.1.3.3 3. Set Environment Variables:

* MAVEN\_HOME = path to maven directory
* Add to PATH: %MAVEN\_HOME%

#### 12.1.3.4 4. Verify:

* mvn -version

### 12.1.4 Step 4: Clone/Download Project

#### 12.1.4.1 1. Using Git:

git clone <repository-url>  
cd Shopsphere-ECommerce-App

#### 12.1.4.2 2. Or download ZIP and extract

### 12.1.5 Step 5: Build the Project

#### 12.1.5.1 1. Navigate to project directory:

cd Shopsphere-ECommerce-App

#### 12.1.5.2 2. Build using Maven:

mvn clean install

#### 12.1.5.3 3. Wait for dependencies to download

## 12.2 Configuration Files

### 12.2.1 1. application.properties

# MongoDB Configuration  
spring.data.mongodb.host=localhost  
spring.data.mongodb.port=27017  
spring.data.mongodb.database=shopsphere\_db  
  
# Application Configuration  
spring.application.name=ShopSphere  
  
# Server Configuration  
server.port=8080  
  
# Swagger/OpenAPI Configuration  
springdoc.api-docs.path=/api-docs  
springdoc.swagger-ui.path=/swagger-ui.html  
springdoc.swagger-ui.enabled=true  
  
# Logging Configuration  
logging.level.org.springframework.data.mongodb=INFO  
logging.level.com.ecommerce=DEBUG

### 12.2.2 2. pom.xml (Key Dependencies)

<dependencies>  
 <!-- Spring Boot Starter for MongoDB -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-mongodb</artifactId>  
 </dependency>  
  
 <!-- Spring Boot Web Starter for REST API -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- Springdoc OpenAPI (Swagger) -->  
 <dependency>  
 <groupId>org.springdoc</groupId>  
 <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
 <version>2.2.0</version>  
 </dependency>  
</dependencies>

## 12.3 Core Modules Implementation

### 12.3.1 1. Main Application Class

@SpringBootApplication  
public class ECommerceApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(ECommerceApplication.class, args);  
 }  
}

### 12.3.2 2. Swagger Configuration

@Configuration  
@OpenAPIDefinition(  
 info = @Info(  
 title = "ShopSphere E-Commerce API",  
 version = "1.0",  
 description = "REST API for E-Commerce Application"  
 )  
)  
public class SwaggerConfig {  
 // Swagger auto-configuration  
}

## 12.4 Product Management Module

### 12.4.1 1. Product Model

@Document(collection = "products")  
public class Product {  
 @Id  
 private String id;  
 private String productId;  
 private String name;  
 private String description;  
 private double price;  
 private String category;  
 private int stock;  
   
 // Constructors, getters, setters  
}

### 12.4.2 2. Product Repository

public interface ProductRepository extends MongoRepository<Product, String> {  
 Optional<Product> findByProductId(String productId);  
 List<Product> findByNameContainingIgnoreCase(String name);  
 List<Product> findByCategory(String category);  
}

### 12.4.3 3. Product Service

@Service  
public class ProductService {  
 @Autowired  
 private ProductRepository productRepository;  
   
 public Product addProduct(Product product) {  
 // Generate product ID  
 // Validate product data  
 // Save and return  
 }  
   
 public List<Product> getAllProducts() {  
 return productRepository.findAll();  
 }  
   
 public Optional<Product> getProductById(String id) {  
 return productRepository.findByProductId(id);  
 }  
   
 public Product updateProduct(String id, Product product) {  
 // Find existing product  
 // Update fields  
 // Save and return  
 }  
   
 public void deleteProduct(String id) {  
 productRepository.deleteById(id);  
 }  
   
 public List<Product> searchProducts(String keyword) {  
 return productRepository.findByNameContainingIgnoreCase(keyword);  
 }  
}

### 12.4.4 4. Product Controller

@RestController  
@RequestMapping("/api/products")  
public class ProductController {  
 @Autowired  
 private ProductService productService;  
   
 @GetMapping  
 public ResponseEntity<List<Product>> getAllProducts() {  
 return ResponseEntity.ok(productService.getAllProducts());  
 }  
   
 @PostMapping  
 public ResponseEntity<Product> addProduct(@RequestBody Product product) {  
 return ResponseEntity.ok(productService.addProduct(product));  
 }  
   
 @PutMapping("/{id}")  
 public ResponseEntity<Product> updateProduct(  
 @PathVariable String id,   
 @RequestBody Product product) {  
 return ResponseEntity.ok(productService.updateProduct(id, product));  
 }  
   
 @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteProduct(@PathVariable String id) {  
 productService.deleteProduct(id);  
 return ResponseEntity.noContent().build();  
 }  
}

## 12.5 User Management Module

Similar structure to Product Management:

* User model with validation
* UserRepository for data access
* UserService for business logic
* UserController for REST endpoints

**Key Features:**

* Email validation and uniqueness
* Password handling (can be enhanced with encryption)
* User registration and login
* Profile management

## 12.6 Shopping Cart Module

### 12.6.1 1. Cart & CartItem Models

@Document(collection = "carts")  
public class Cart {  
 @Id  
 private String id;  
 private String cartId;  
 private String userId;  
 private List<CartItem> items;  
 private double totalAmount;  
   
 // Auto-calculate total  
 public void calculateTotal() {  
 this.totalAmount = items.stream()  
 .mapToDouble(CartItem::getSubtotal)  
 .sum();  
 }  
}  
  
public class CartItem {  
 private String productId;  
 private String productName;  
 private int quantity;  
 private double price;  
 private double subtotal;  
}

### 12.6.2 2. Cart Service

@Service  
public class CartService {  
 public Cart addToCart(String userId, String productId, int quantity) {  
 // Find or create cart  
 // Check product stock  
 // Add item or update quantity  
 // Calculate total  
 // Save and return  
 }  
   
 public Cart removeFromCart(String userId, String productId) {  
 // Find cart  
 // Remove item  
 // Recalculate total  
 // Save and return  
 }  
   
 public Cart getCart(String userId) {  
 return cartRepository.findByUserId(userId)  
 .orElse(new Cart());  
 }  
   
 public void clearCart(String userId) {  
 // Find and delete cart  
 }  
}

## 12.7 Order Management Module

### 12.7.1 1. Order & OrderItem Models

@Document(collection = "orders")  
public class Order {  
 @Id  
 private String id;  
 private String orderId;  
 private String userId;  
 private List<OrderItem> items;  
 private double totalAmount;  
 private LocalDateTime orderDate;  
 private String paymentStatus; // PAID, PENDING, FAILED  
}

### 12.7.2 2. Order Service

@Service  
public class OrderService {  
 public Order placeOrder(String userId) {  
 // Get user's cart  
 // Validate cart not empty  
 // Check stock availability  
 // Create order from cart  
 // Simulate payment processing  
 // Update product stock  
 // Clear cart  
 // Save and return order  
 }  
   
 public List<Order> getOrderHistory(String userId) {  
 return orderRepository.findByUserId(userId);  
 }  
}

# 13 CHAPTER 4: FUNCTIONALITY AND FEATURES

## 13.1 CLI Interface

The CLI provides an interactive menu-driven interface:

### 13.1.1 Main Menu:

1. Product Management
2. User Management
3. Shopping Cart
4. Order Management
5. Exit

### 13.1.2 Product Management Submenu:

1. View All Products
2. Search Products
3. Filter by Category
4. Add Product (Admin)
5. Update Product (Admin)
6. Delete Product (Admin)
7. Back to Main Menu

### 13.1.3 User Management Submenu:

1. Register New User
2. Login
3. View Profile
4. Update Profile
5. Back to Main Menu

### 13.1.4 Shopping Cart Submenu:

1. View Cart
2. Add Product to Cart
3. Update Quantity
4. Remove Item
5. Clear Cart
6. Proceed to Checkout
7. Back to Main Menu

### 13.1.5 Order Management Submenu:

1. View Order History
2. View Order Details
3. Back to Main Menu

## 13.2 REST API Endpoints

### 13.2.1 Product APIs:

| Method | Endpoint | Description |
| --- | --- | --- |
| GET | /api/products | Get all products |
| GET | /api/products/{id} | Get product by ID |
| GET | /api/products/search?name= | Search products |
| GET | /api/products/category/{cat} | Filter by category |
| POST | /api/products | Add new product |
| PUT | /api/products/{id} | Update product |
| DELETE | /api/products/{id} | Delete product |

#### 13.2.1.1 Request Body Example (POST /api/products):

{  
 "name": "Wireless Mouse",  
 "description": "Ergonomic wireless mouse",  
 "price": 29.99,  
 "category": "Electronics",  
 "stock": 100  
}

#### 13.2.1.2 Response Example:

{  
 "productId": "P001",  
 "name": "Wireless Mouse",  
 "description": "Ergonomic wireless mouse",  
 "price": 29.99,  
 "category": "Electronics",  
 "stock": 100  
}

### 13.2.2 User APIs:

| Method | Endpoint | Description |
| --- | --- | --- |
| POST | /api/users/register | Register new user |
| POST | /api/users/login | User login |
| GET | /api/users/{id} | Get user profile |
| PUT | /api/users/{id} | Update profile |

#### 13.2.2.1 Request Body Example (POST /api/users/register):

{  
 "username": "john\_doe",  
 "email": "john@example.com",  
 "password": "password123",  
 "address": "123 Main St, City"  
}

### 13.2.3 Cart APIs:

| Method | Endpoint | Description |
| --- | --- | --- |
| GET | /api/cart/{userId} | Get user’s cart |
| POST | /api/cart/add | Add item to cart |
| PUT | /api/cart/update | Update item quantity |
| DELETE | /api/cart/remove | Remove item |
| DELETE | /api/cart/clear/{userId} | Clear cart |

#### 13.2.3.1 Request Body Example (POST /api/cart/add):

{  
 "userId": "U001",  
 "productId": "P001",  
 "quantity": 2  
}

### 13.2.4 Order APIs:

| Method | Endpoint | Description |
| --- | --- | --- |
| POST | /api/orders/place | Place order |
| GET | /api/orders/{userId} | Get order history |
| GET | /api/orders/details/{orderId} | Get order details |

#### 13.2.4.1 Request Body Example (POST /api/orders/place):

{  
 "userId": "U001"  
}

#### 13.2.4.2 Response Example:

{  
 "orderId": "ORD001",  
 "userId": "U001",  
 "items": [  
 {  
 "productId": "P001",  
 "productName": "Wireless Mouse",  
 "quantity": 2,  
 "price": 29.99,  
 "subtotal": 59.98  
 }  
 ],  
 "totalAmount": 59.98,  
 "orderDate": "2025-10-31T10:30:00",  
 "paymentStatus": "PAID"  
}

## 13.3 Swagger Documentation

**Access Swagger UI:** http://localhost:8080/swagger-ui.html

### 13.3.1 Features:

* Interactive API documentation
* Try out API endpoints directly
* View request/response schemas
* See example payloads
* HTTP status codes documentation
* Parameter descriptions

**OpenAPI JSON:** http://localhost:8080/api-docs

## 13.4 Data Validation

### 13.4.1 1. Product Validation:

* Name: Required, not empty
* Price: Must be positive
* Stock: Cannot be negative
* Category: Must be from predefined list

### 13.4.2 2. User Validation:

* Email: Valid format, unique
* Username: Required, not empty
* Password: Minimum length requirements

### 13.4.3 3. Cart Validation:

* Quantity: Must be positive
* Product availability check
* Stock validation

### 13.4.4 4. Order Validation:

* Cart must not be empty
* Sufficient stock for all items
* User must exist

## 13.5 Error Handling

The application implements comprehensive error handling:

### 13.5.1 1. HTTP Status Codes:

* **200 OK:** Successful request
* **201 Created:** Resource created
* **204 No Content:** Successful deletion
* **400 Bad Request:** Invalid input
* **404 Not Found:** Resource not found
* **500 Internal Server Error:** Server error

### 13.5.2 2. Custom Exception Handling:

* ProductNotFoundException
* UserNotFoundException
* InsufficientStockException
* InvalidInputException

### 13.5.3 3. Error Response Format:

{  
 "timestamp": "2025-10-31T10:30:00",  
 "status": 404,  
 "error": "Not Found",  
 "message": "Product with ID P999 not found",  
 "path": "/api/products/P999"  
}

# 14 CHAPTER 5: TESTING AND DEPLOYMENT

## 14.1 Running the Application

### 14.1.1 Method 1: Using Maven

1. Open terminal in project directory
2. Run command: mvn spring-boot:run
3. Application starts on http://localhost:8080

### 14.1.2 Method 2: Using JAR File

1. Build the JAR: mvn clean package
2. Run the JAR: java -jar target/ecommerce-mongodb-cli-1.0.0.jar

### 14.1.3 Method 3: Using IDE

1. Open project in IntelliJ IDEA/Eclipse
2. Locate ECommerceApplication.java
3. Right-click and select “Run”

### 14.1.4 Method 4: CLI Mode Only

To run only CLI interface:

1. Comment out @SpringBootApplication
2. Run ECommerceCLI.java directly

## 14.2 Testing Procedures

### 14.2.1 1. Manual Testing via CLI

**Step 1:** Start the application  
**Step 2:** Choose option from main menu  
**Step 3:** Follow prompts for data entry  
**Step 4:** Verify results displayed  
**Step 5:** Check MongoDB for data persistence

#### 14.2.1.1 Example Test Case - Add Product:

1. Select “1. Product Management”
2. Select “4. Add Product”
3. Enter product details:
   * Name: Laptop
   * Description: Gaming laptop
   * Price: 1299.99
   * Category: Electronics
   * Stock: 25
4. Verify success message
5. View all products to confirm addition

### 14.2.2 2. Testing via Swagger UI

**Step 1:** Open browser  
**Step 2:** Navigate to http://localhost:8080/swagger-ui.html  
**Step 3:** Expand API endpoint  
**Step 4:** Click “Try it out”  
**Step 5:** Enter request body  
**Step 6:** Click “Execute”  
**Step 7:** Verify response

#### 14.2.2.1 Example: Test POST /api/products

1. Expand ProductController
2. Click POST /api/products
3. Click “Try it out”
4. Enter JSON:

{  
 "name": "Smartphone",  
 "description": "Latest model",  
 "price": 699.99,  
 "category": "Electronics",  
 "stock": 50  
}

1. Click Execute
2. Check response code 200
3. Verify product created in response

### 14.2.3 3. Testing via Postman

**Step 1:** Install Postman  
**Step 2:** Create new request  
**Step 3:** Set method (GET/POST/PUT/DELETE)  
**Step 4:** Enter URL: http://localhost:8080/api/…  
**Step 5:** Add request body (for POST/PUT)  
**Step 6:** Send request  
**Step 7:** Verify response

#### 14.2.3.1 Example Collection:

* Register User
* Login User
* Add Product
* Search Products
* Add to Cart
* Place Order
* View Order History

### 14.2.4 4. Database Verification

**Step 1:** Open MongoDB Compass or Shell  
**Step 2:** Connect to localhost:27017  
**Step 3:** Select shopsphere\_db  
**Step 4:** View collections: \* products \* users \* carts \* orders

**Step 5:** Verify data matches application operations

## 14.3 Sample Inputs and Outputs

### 14.3.1 Scenario 1: Adding a Product

#### 14.3.1.1 CLI Input:

Enter Product Name: Wireless Headphones  
Enter Description: Bluetooth noise-cancelling headphones  
Enter Price: 149.99  
Enter Category: Electronics  
Enter Stock Quantity: 75

#### 14.3.1.2 CLI Output:

✓ Product added successfully!  
Product ID: P001  
Name: Wireless Headphones  
Price: $149.99  
Stock: 75 units

#### 14.3.1.3 MongoDB Document:

{  
 "\_id": ObjectId("..."),  
 "productId": "P001",  
 "name": "Wireless Headphones",  
 "description": "Bluetooth noise-cancelling headphones",  
 "price": 149.99,  
 "category": "Electronics",  
 "stock": 75  
}

### 14.3.2 Scenario 2: User Registration

#### 14.3.2.1 API Request: POST /api/users/register

{  
 "username": "alice\_smith",  
 "email": "alice@example.com",  
 "password": "securePass123",  
 "address": "456 Oak Avenue, Springfield"  
}

#### 14.3.2.2 API Response: 201 Created

{  
 "userId": "U001",  
 "username": "alice\_smith",  
 "email": "alice@example.com",  
 "address": "456 Oak Avenue, Springfield"  
}

### 14.3.3 Scenario 3: Adding to Cart

#### 14.3.3.1 API Request: POST /api/cart/add

{  
 "userId": "U001",  
 "productId": "P001",  
 "quantity": 2  
}

#### 14.3.3.2 API Response: 200 OK

{  
 "cartId": "CART001",  
 "userId": "U001",  
 "items": [  
 {  
 "productId": "P001",  
 "productName": "Wireless Headphones",  
 "quantity": 2,  
 "price": 149.99,  
 "subtotal": 299.98  
 }  
 ],  
 "totalAmount": 299.98  
}

### 14.3.4 Scenario 4: Placing an Order

#### 14.3.4.1 API Request: POST /api/orders/place

{  
 "userId": "U001"  
}

#### 14.3.4.2 API Response: 200 OK

{  
 "orderId": "ORD001",  
 "userId": "U001",  
 "items": [  
 {  
 "productId": "P001",  
 "productName": "Wireless Headphones",  
 "quantity": 2,  
 "price": 149.99,  
 "subtotal": 299.98  
 }  
 ],  
 "totalAmount": 299.98,  
 "orderDate": "2025-10-31T14:30:00",  
 "paymentStatus": "PAID"  
}

**Stock Update:**  
Product P001 stock: 75 → 73 (reduced by quantity ordered)

### 14.3.5 Scenario 5: Searching Products

#### 14.3.5.1 CLI Input:

Enter search keyword: wireless

#### 14.3.5.2 CLI Output:

Search Results (2 found):  
  
1. Product ID: P001  
 Name: Wireless Headphones  
 Price: $149.99  
 Stock: 73 units  
  
2. Product ID: P005  
 Name: Wireless Mouse  
 Price: $29.99  
 Stock: 150 units

### 14.3.6 Scenario 6: Viewing Order History

#### 14.3.6.1 API Request: GET /api/orders/U001

#### 14.3.6.2 API Response: 200 OK

[  
 {  
 "orderId": "ORD001",  
 "totalAmount": 299.98,  
 "orderDate": "2025-10-31T14:30:00",  
 "paymentStatus": "PAID",  
 "itemCount": 1  
 },  
 {  
 "orderId": "ORD002",  
 "totalAmount": 699.99,  
 "orderDate": "2025-10-31T15:45:00",  
 "paymentStatus": "PAID",  
 "itemCount": 2  
 }  
]

## 14.4 Troubleshooting

### 14.4.1 Problem 1: MongoDB Connection Failed

**Error:** “Unable to connect to MongoDB”

#### 14.4.1.1 Solutions:

1. Verify MongoDB is running:
   * Windows: Check Services for “MongoDB”
   * Linux/Mac: sudo systemctl status mongod
2. Check connection string in application.properties:
   * spring.data.mongodb.host=localhost
   * spring.data.mongodb.port=27017
3. Verify port 27017 is not blocked by firewall
4. Try connecting via MongoDB Compass to test connection

### 14.4.2 Problem 2: Port 8080 Already in Use

**Error:** “Port 8080 is already in use”

#### 14.4.2.1 Solutions:

1. Change port in application.properties:
   * server.port=8081
2. Or stop the application using port 8080:
   * Windows: netstat -ano | findstr :8080
   * Kill process: taskkill /PID <PID> /F
   * Linux/Mac: lsof -i :8080
   * Kill: kill -9 <PID>

### 14.4.3 Problem 3: Maven Dependencies Not Downloading

**Error:** “Cannot resolve dependencies”

#### 14.4.3.1 Solutions:

1. Check internet connection
2. Delete .m2/repository folder and rebuild
3. Use Maven clean install with -U flag: mvn clean install -U
4. Check Maven settings.xml for proxy settings

### 14.4.4 Problem 4: Swagger UI Not Loading

**Error:** “404 Not Found at /swagger-ui.html”

#### 14.4.4.1 Solutions:

1. Verify Springdoc dependency in pom.xml
2. Check application.properties for correct swagger paths
3. Ensure Spring Boot Web starter is included
4. Try accessing: http://localhost:8080/swagger-ui/index.html

### 14.4.5 Problem 5: Data Not Persisting

**Error:** “Data disappears after restart”

#### 14.4.5.1 Solutions:

1. Check MongoDB is running persistently
2. Verify database name in application.properties
3. Use MongoDB Compass to verify data is saved
4. Check for exceptions in application logs

### 14.4.6 Problem 6: Stock Not Updating

**Error:** “Product stock remains same after order”

#### 14.4.6.1 Solutions:

1. Check OrderService implementation
2. Verify transaction is committed
3. Review logs for exceptions
4. Test with MongoDB queries directly

# 15 CHAPTER 6: CONCLUSION AND FUTURE SCOPE

## 15.1 Conclusion

The ShopSphere E-Commerce Application successfully demonstrates the implementation of a full-featured online shopping platform using modern Java technologies. This project has achieved all its primary objectives:

### 15.1.1 1. Technical Achievements:

* Successfully integrated Spring Boot with MongoDB for a robust backend
* Implemented RESTful API following industry best practices
* Created comprehensive API documentation using Swagger
* Developed both CLI and API interfaces for versatile access
* Established layered architecture for maintainability and scalability

### 15.1.2 2. Functional Achievements:

* Complete product catalog management system
* User registration and authentication
* Shopping cart with real-time calculations
* Order processing with automated stock management
* Payment status tracking

### 15.1.3 3. Learning Outcomes:

* Gained practical experience with Spring Boot framework
* Understood NoSQL database design and implementation
* Learned RESTful API design principles
* Mastered dependency injection and inversion of control
* Developed skills in error handling and validation

### 15.1.4 4. Business Value:

* The application provides a solid foundation for e-commerce operations
* Scalable architecture allows for future growth
* API-first design enables integration with frontend frameworks
* Modular structure facilitates easy maintenance and updates

## 15.2 Lessons Learned

### 15.2.1 1. Technical Lessons:

* Spring Boot significantly reduces configuration overhead
* MongoDB’s flexibility suits e-commerce data models well
* Proper exception handling is crucial for user experience
* API documentation is essential for collaboration
* Testing should be integrated throughout development

### 15.2.2 2. Design Lessons:

* Separation of concerns improves code maintainability
* Validation should occur at multiple layers
* User experience matters even in CLI applications
* Database schema should be designed for common queries
* Error messages should be clear and actionable

### 15.2.3 3. Process Lessons:

* Incremental development helps manage complexity
* Regular testing catches issues early
* Documentation should be maintained alongside code
* Version control is essential for project management

## 15.3 Future Enhancements

### 15.3.1 1. Security Enhancements:

* Implement JWT-based authentication
* Add password encryption using BCrypt
* Implement role-based access control (Admin/User)
* Add HTTPS/SSL support
* Implement rate limiting for API endpoints
* Add CSRF protection

### 15.3.2 2. Functionality Enhancements:

* Product reviews and ratings system
* Wishlist functionality
* Product recommendations based on purchase history
* Advanced search with filters (price range, rating, etc.)
* Order tracking and status updates
* Email notifications for orders
* Multi-currency support
* Discount codes and promotions
* Inventory alerts for low stock

### 15.3.3 3. User Interface Enhancements:

* Develop React/Angular frontend
* Create mobile application (Android/iOS)
* Add product image upload and display
* Implement real-time chat support
* Add dashboard for analytics
* Create admin panel for management

### 15.3.4 4. Technical Enhancements:

* Implement caching with Redis
* Add pagination for large datasets
* Implement asynchronous processing for orders
* Add file upload for product images
* Implement full-text search with Elasticsearch
* Add API versioning
* Implement microservices architecture
* Add Docker containerization
* Implement CI/CD pipeline

### 15.3.5 5. Payment Enhancements:

* Integrate real payment gateways (Stripe, PayPal)
* Support multiple payment methods
* Add invoice generation
* Implement refund processing
* Add payment history

### 15.3.6 6. Analytics & Reporting:

* Sales reports and dashboards
* User behavior analytics
* Inventory reports
* Revenue tracking
* Popular products analysis
* Customer segmentation

### 15.3.7 7. Business Features:

* Multi-vendor support
* Shipping integration
* Tax calculation
* International shipping
* Return and refund management
* Customer support ticketing system

## 15.4 Project Impact

This project has demonstrated:

* Practical application of Java and Spring Boot knowledge
* Understanding of e-commerce business logic
* Ability to design and implement complex systems
* Skills in API development and documentation
* Database design and management capabilities
* Problem-solving and debugging proficiency

The knowledge and experience gained from this project are directly applicable to:

* Enterprise application development
* Microservices architecture
* E-commerce platform development
* API design and implementation
* Full-stack development projects

## 15.5 References

### 15.5.1 1. Documentation:

* **Spring Boot Official Documentation**  
  https://docs.spring.io/spring-boot/docs/current/reference/html/
* **Spring Data MongoDB Reference**  
  https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/
* **MongoDB Manual**  
  https://docs.mongodb.com/manual/
* **Springdoc OpenAPI Documentation**  
  https://springdoc.org/
* **Java 17 Documentation**  
  https://docs.oracle.com/en/java/javase/17/

### 15.5.2 2. Tutorials and Guides:

* **Baeldung - Spring Boot Tutorials**  
  https://www.baeldung.com/spring-boot
* **MongoDB University**  
  https://university.mongodb.com/
* **Spring Framework Guides**  
  https://spring.io/guides

### 15.5.3 3. Books:

* “Spring Boot in Action” by Craig Walls
* “MongoDB: The Definitive Guide” by Shannon Bradshaw
* “RESTful Web Services” by Leonard Richardson
* “Clean Code” by Robert C. Martin

### 15.5.4 4. Tools:

* IntelliJ IDEA - https://www.jetbrains.com/idea/
* MongoDB Compass - https://www.mongodb.com/products/compass
* Postman - https://www.postman.com/
* Maven - https://maven.apache.org/

# 16 LIST OF FIGURES

| S.No | Figure Description | Page Reference |
| --- | --- | --- |
| 01. | Project Architecture Diagram | Chapter 1 |
| 02. | Database Schema Design | Chapter 2 |
| 03. | Spring Boot Project Structure | Chapter 2 |
| 04. | MVC Architecture Flow | Chapter 2 |
| 05. | Product Management Module | Chapter 3 |
| 06. | User Management Module | Chapter 3 |
| 07. | Shopping Cart Module | Chapter 3 |
| 08. | Order Processing Module | Chapter 3 |
| 09. | CLI Main Menu | Chapter 4 |
| 10. | Swagger UI Interface | Chapter 4 |
| 11. | Product API Endpoints | Chapter 4 |
| 12. | Cart API Endpoints | Chapter 4 |
| 13. | MongoDB Collections Structure | Chapter 4 |
| 14. | Sample Product Output | Chapter 5 |
| 15. | Sample Order Output | Chapter 5 |
| 16. | Order History View | Chapter 5 |
| 17. | MongoDB Compass View | Chapter 5 |
| 18. | Postman Testing Example | Chapter 5 |
| 19. | Error Handling Example | Chapter 5 |
| 20. | Application Startup Logs | Chapter 5 |

# 17 APPENDIX A: COMPLETE API REFERENCE

Please refer to Swagger UI at http://localhost:8080/swagger-ui.html for:

* Complete API endpoint list
* Request/Response schemas
* Example payloads
* HTTP status codes
* Parameter descriptions

# 18 APPENDIX B: DATABASE QUERIES

Common MongoDB queries for ShopSphere:

### 18.0.1 1. Find all products:

db.products.find()

### 18.0.2 2. Find products by category:

db.products.find({category: "Electronics"})

### 18.0.3 3. Search products by name:

db.products.find({name: /wireless/i})

### 18.0.4 4. Find user by email:

db.users.findOne({email: "user@example.com"})

### 18.0.5 5. Find user’s cart:

db.carts.findOne({userId: "U001"})

### 18.0.6 6. Find user’s orders:

db.orders.find({userId: "U001"}).sort({orderDate: -1})

### 18.0.7 7. Get order count:

db.orders.countDocuments()

### 18.0.8 8. Update product stock:

db.products.updateOne(  
 {productId: "P001"},  
 {$inc: {stock: -5}}  
)

# 19 APPENDIX C: ENVIRONMENT SETUP CHECKLIST

* Java JDK 17 installed
* JAVA\_HOME environment variable set
* Maven installed and configured
* MongoDB installed and running
* MongoDB service started
* Project cloned/downloaded
* Dependencies downloaded (mvn clean install)
* application.properties configured
* MongoDB database created
* Port 8080 available
* IDE/Editor configured
* MongoDB Compass installed (optional)
* Postman installed (optional)

# 20 END OF DOCUMENT

**Project:** ShopSphere - E-Commerce Application  
**Technology Stack:** Spring Boot 3.1.5, MongoDB, Java 17  
**Author:** Ramisetty Divyasree  
**Roll No:** 22781A3137  
**Date:** October 31, 2025