INNOVIOR

Software Requirements Specification (SRS)

E-commerce + Integrated Stock Management System

Document Version: 1.0
Date: August 28, 2025
Prepared by: Thisara Dasun

INNOVIOR

Table of Contents

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements for the E-commerce Integrated Stock Management System. The system provides a public e-commerce storefront tightly integrated with internal stock management capabilities.

1.2 Scope

The system encompasses:

- **Public E-commerce Storefront**: Product catalog, shopping cart, checkout, and customer accounts
- **Internal Stock Management**: Inventory tracking, reservations, stock movements, and anti-oversell protection
- **Administrative Functions**: Order management, fulfillment, reporting, and system configuration
- **Payment Processing**: Stripe integration (test mode) and Cash on Delivery options

1.3 Definitions and Abbreviations

Term	Definition
SKU	Stock Keeping Unit - unique product variant identifier
COD	Cash on Delivery payment method
qty_on_hand	Physical quantity available for sale
qty_reserved	Quantity held for pending transactions
MVP	Minimum Viable Product
RBAC	Role-Based Access Control
API	Application Programming Interface
SPA	Single Page Application

1.4 References

- Laravel 11 Documentation
- Stripe API Documentation
- MongoDB Documentation
- Docker Documentation

---

2. Overall Description

2.1 Product Perspective

The system operates as a comprehensive e-commerce platform with integrated inventory management, serving both public customers and internal administrators through role-based interfaces.

2.2 Product Functions

- **Catalog Management**: Product browsing, variant selection, category navigation
- **Inventory Control**: Real-time stock tracking with reservation system
- **Order Processing**: Cart management, secure checkout, payment processing
- **Fulfillment Management**: Order status tracking, shipment management
- **Reporting & Analytics**: Sales reports, inventory status, performance metrics
- **User Management**: Customer accounts, administrative roles, permissions

2.3 User Classes and Characteristics

User Type	Access Level	Primary Functions
Customer	Public	Browse catalog, place orders, track shipments
Admin	Internal	System settings, user management, global oversight
Inventory Manager	Internal	Product management, stock adjustments, pricing
Fulfillment Staff	Internal	Order processing, shipment management
Support Staff	Internal	Customer assistance, order modifications

2.4 Operating Environment

- **Backend**: PHP 8.2+, Laravel 11, MongoDB, Redis
- **Frontend**: Blade templates with Inertia.js/Vue.js components
- **Infrastructure**: Docker containers, NGINX, SMTP mail service
- **Development**: Git version control, GitHub Actions CI/CD

2.5 Design and Implementation Constraints

- Must prevent overselling through database row locks and atomic operations
- Payment processing limited to Stripe test mode and COD
- No raw payment card data storage (PCI compliance)
- Inventory reservations must auto-expire to prevent deadlocks
- System must handle concurrent user sessions without data corruption

---

3. System Features

3.1 Product Catalog Management

3.1.1 Description

Comprehensive product catalog with categories, variants, pricing, and media management.

3.1.2 Functional Requirements

FR-CAT-01: Product Creation and Management

- System shall allow authorized users to create, edit, and deactivate products
- Each product must have unique SKU, name, description, and category assignment
- Products can have multiple variants (size, color) with individual pricing
- Product images must support multiple formats with alt-text for accessibility

FR-CAT-02: Category Management

- System shall support hierarchical category structure
- Categories must have unique slugs for SEO-friendly URLs
- Inactive categories shall be hidden from public storefront

FR-CAT-03: Product Search and Filtering

- Public catalog must support keyword search functionality
- Filtering options include category, price range, availability status
- Results shall be paginated with configurable items per page
- Sorting options: price (low to high, high to low), newest, popularity

3.2 Inventory Management and Reservation System

3.2.1 Description

Real-time inventory tracking with reservation mechanism to prevent overselling.

3.2.2 Functional Requirements

FR-INV-01: Stock Quantity Tracking

- System shall maintain two quantity fields: qty_on_hand and qty_reserved
- Inventory invariant: qty_on_hand >= 0 and qty_reserved >= 0 at all times
- Available stock calculation: qty_on_hand - qty_reserved

FR-INV-02: Stock Reservation System

- Adding items to cart shall create temporary stock reservation
- Reservations must expire automatically after configurable time (default: 30 minutes)
- Failed payments shall immediately release reserved quantities
- Successful payments shall deduct from qty_on_hand and reduce qty_reserved

FR-INV-03: Concurrency Protection

- Database row-level locking for all inventory updates
- Atomic operations for reservation/deduction/release transactions
- Concurrent checkout attempts cannot cause overselling

FR-INV-04: Stock Movement Tracking

- All inventory changes must be logged with timestamp, reason, and user
- Movement types: adjustment, sale, return, transfer, expiry
- Audit trail must be immutable and searchable

3.3 Shopping Cart and Checkout

3.3.1 Description

Secure shopping cart with real-time inventory validation and streamlined checkout process.

3.3.2 Functional Requirements

FR-CART-01: Cart Management

- Customers can add, update, and remove items from cart
- Cart shall validate item availability before quantity changes
- Cart totals include subtotal, tax, shipping, and discounts
- Guest checkout support with optional account creation

FR-CART-02: Checkout Process

- Address capture for billing and shipping information
- Shipping method selection with rate calculation
- Tax calculation based on shipping address
- Payment method selection (Stripe or COD)

FR-CART-03: Payment Processing

- Stripe Payment Intents integration for card payments
- Webhook handling for payment status verification
- Idempotent payment processing to prevent duplicate charges
- COD orders marked as pending payment

3.4 Order Management and Fulfillment

3.4.1 Description

Complete order lifecycle management from placement to delivery.

3.4.2 Functional Requirements

FR-ORD-01: Order Processing

- Order status workflow: PENDING → PAID → FULFILLED → DELIVERED
- Unique order number generation for tracking
- Order cancellation only allowed for unfulfilled orders

- Refund processing through Stripe API for paid orders

FR-ORD-02: Shipment Management

- Fulfillment staff can create shipments with carrier and tracking information
- Shipment status tracking: READY → SHIPPED → DELIVERED
- Email notifications for order confirmation and shipping updates
- Bulk shipment processing for efficiency

3.5 Promotions and Discounts

3.5.1 Description

Coupon code system for customer discounts and marketing campaigns.

3.5.2 Functional Requirements

FR-PROMO-01: Coupon Management

- Support for percentage and fixed amount discounts
- Date range validation for coupon validity
- Minimum order value requirements
- Usage limits per coupon and per customer
- Atomic coupon usage counting to prevent over-redemption

3.6 Reporting and Analytics

3.6.1 Description

Business intelligence dashboard with sales analytics and inventory reporting.

3.6.2 Functional Requirements

FR-REP-01: Sales Reporting

- Daily, weekly, and monthly sales summaries
- Top-selling products by revenue and quantity
- Customer purchase history and analytics
- Revenue trends and growth metrics

FR-REP-02: Inventory Reporting

- Current stock levels and reserved quantities
- Low stock alerts with configurable thresholds
- Stock movement history and trends
- Supplier performance metrics

FR-REP-03: Data Export

- CSV export functionality for all reports
- Scheduled report generation and email delivery
- Custom date range selection for historical analysis

---

4. External Interface Requirements

4.1 User Interfaces

4.1.1 Public Storefront Interface

- Responsive design compatible with desktop, tablet, and mobile devices
- Clean, modern aesthetic with intuitive navigation
- Product search and filtering controls
- Shopping cart with real-time updates
- Secure checkout flow with progress indicators

4.1.2 Administrative Interface

- Role-based dashboard with relevant widgets and metrics
- Data tables with sorting, filtering, and pagination
- Form validation with clear error messaging
- Modal dialogs for confirmation and quick actions
- Breadcrumb navigation for deep page hierarchies

4.2 Hardware Interfaces

- Standard web server hardware requirements
- Database server with sufficient storage and memory
- Redis server for caching and session management

4.3 Software Interfaces

4.3.1 Payment Gateway Integration

- **Stripe API v2**: Payment Intent creation, confirmation, and webhook processing
- **SSL/TLS**: Secure communication for payment data
- **Webhook Endpoints**: Real-time payment status updates

4.3.2 Email Service Integration

- **SMTP Configuration**: Order confirmations, shipping notifications
- **Template Engine**: Dynamic email content generation
- **Queue Processing**: Asynchronous email delivery

4.3.3 Database Integration

- **MongoDB**: Primary data storage with document-based structure
- **Redis**: Caching layer and session storage
- **Connection Pooling**: Efficient database connection management

---

5. Non-Functional Requirements

5.1 Performance Requirements

- **Page Load Times**: Product listing pages < 1.5 seconds
- **Checkout Performance**: Complete checkout process < 2 seconds
- **Concurrent Users**: Support minimum 100 concurrent active sessions
- **Database Queries**: Complex queries optimized to execute < 200ms

5.2 Security Requirements

- **Authentication**: Secure password hashing using bcrypt
- **Authorization**: Role-based access control (RBAC) implementation
- **Data Protection**: PCI compliance through Stripe tokenization
- **Input Validation**: All user inputs sanitized and validated server-side
- **Session Management**: Secure session handling with automatic expiration
- **HTTPS**: All communications encrypted using SSL/TLS

5.3 Reliability Requirements

- **Uptime**: 99.5% system availability during business hours
- **Error Handling**: Graceful degradation with user-friendly error messages
- **Data Backup**: Automated daily backups with point-in-time recovery
- **Transaction Integrity**: ACID compliance for all financial transactions

5.4 Scalability Requirements

- **Horizontal Scaling**: Architecture supports load balancing across multiple servers
- **Database Scaling**: MongoDB sharding capability for large datasets
- **Caching Strategy**: Redis implementation to reduce database load
- **Queue Management**: Background job processing for resource-intensive tasks

5.5 Usability Requirements

- **Intuitive Navigation**: Maximum 3 clicks to reach any product
- **Mobile Responsiveness**: Full functionality on devices 320px width and above
- **Accessibility**: WCAG 2.1 AA compliance for users with disabilities
- **Browser Support**: Chrome, Firefox, Safari, Edge (latest 2 versions)

---

6. System Architecture

6.1 Architectural Overview

The system follows a Model-View-Controller (MVC) pattern with service-oriented architecture (SOA) principles.

6.2 Technology Stack

Layer	Technology	Purpose
Frontend	Blade Templates, Vue.js/Inertia	User interface rendering
Backend	PHP 8.2, Laravel 11	Business logic and API
Database	MongoDB	Primary data storage
Cache	Redis	Session storage and caching
Queue	Redis/Laravel Queue	Background job processing
Email	SMTP/Laravel Mail	Notification delivery
Payments	Stripe API	Payment processing

6.3 System Components

6.3.1 Presentation Layer

- **Public Storefront**: Customer-facing product catalog and checkout
- **Administrative Panel**: Internal management interfaces
- **API Layer**: RESTful endpoints for frontend communication

6.3.2 Business Logic Layer

- **Services**: Core business logic encapsulation
- **Repositories**: Data access abstraction
- **Events/Listeners**: Decoupled system communication
- **Jobs/Queues**: Asynchronous processing

6.3.3 Data Layer

- **Models**: Eloquent ORM with MongoDB integration
- **Migrations**: Database schema versioning
- **Seeders**: Test and initial data population

---

7. Data Requirements

7.1 Database Schema

7.1.1 Core Collections

Products Collection

```
{
 _id: ObjectId,
 name: String,
 description: String,
 category_id: ObjectId,
 active: Boolean,
 created_at: DateTime,
 updated_at: DateTime
}
```

Variants Collection

```
{
 _id: ObjectId,
 product_id: ObjectId,
 sku: String, // unique
```

option_values: Object,
price: Decimal,
compare_at_price: Decimal,
active: Boolean,
created_at: DateTime,
updated_at: DateTime
}

Inventory Collection

{
_id: ObjectId,
variant_id: ObjectId,
qty_on_hand: Integer,
qty_reserved: Integer,
reorder_point: Integer,
created_at: DateTime,
updated_at: DateTime
}

Orders Collection

{
_id: ObjectId,
customer_id: ObjectId,
order_number: String, // unique
status: Enum,
subtotal: Decimal,
discount: Decimal,
tax: Decimal,
shipping: Decimal,
total: Decimal,
currency: String,
billing_address: Object,
shipping_address: Object,
created_at: DateTime,
updated_at: DateTime
}

7.2 Data Validation Rules

- All monetary values stored as decimal with 2 decimal places
- Email addresses validated using RFC 5322 standard
- SKU format: alphanumeric with hyphens, 3-20 characters
- Phone numbers stored in E.164 international format
- All timestamps stored in UTC timezone

7.3 Data Backup and Recovery

- **Backup Frequency**: Daily automated backups at 2 AM UTC
- **Retention Period**: 30 days for daily backups, 12 months for monthly
- **Recovery Time Objective**: 4 hours maximum downtime
- **Recovery Point Objective**: Maximum 1 hour data loss

---

8. Implementation Plan

8.1 Development Phases

Phase 1: Foundation Setup (Week 1, Days 1-7)

Objectives: Establish development environment and core infrastructure

Deliverables:

- Docker environment with MongoDB and Redis
- Laravel application with authentication
- Basic models and database structure
- Core API routing architecture

Key Files Created:

- Docker configuration files
- MongoDB migrations for core collections
- User authentication enhancements
- Basic model definitions (User, Product, Category, Inventory)

Phase 2: Core Business Logic (Week 2, Days 8-14)

Objectives: Implement primary business functionality

Deliverables:

- Complete product and category management
- Inventory tracking with reservation system
- Basic CRUD operations for all entities
- Stock movement logging

Key Files Created:

- Product/Category controllers and services
- Inventory management system
- Stock reservation logic
- Basic API endpoints for core functionality

Phase 3: Advanced Features & Frontend (Week 3, Days 15-21)

Objectives: Develop user interfaces and advanced features

Deliverables:

- Complete order management system
- Administrative dashboard interfaces
- Customer-facing storefront
- Reporting and analytics functionality

Key Files Created:

- Complete Blade template structure
- Vue.js components for interactive elements
- Order processing workflow
- Report generation services

Phase 4: Integration & Optimization (Week 4, Days 22-28)

Objectives: System integration, testing, and performance optimization

Deliverables:

- Stripe payment integration
- Email notification system
- Performance optimization and caching
- Comprehensive testing suite

Key Files Created:

- Payment processing services
- Email templates and notification system
- Redis caching implementation
- Test suites for all major functionality

8.2 Implementation Priority Matrix

Priority	Component	Business Impact	Technical Risk	Implementation Days
1	Foundation & Setup	High	High	7
2	Product Management	Critical	Medium	4
3	Inventory System	Critical	High	5
4	Order Processing	Critical	Medium	4
5	User Interfaces	High	Low	6
6	Payment Integration	High	Medium	2

8.3 Risk Mitigation Strategies

- **Concurrency Issues**: Implement comprehensive database locking and transaction management
- **Payment Security**: Use Stripe's secure tokenization, never store card data
- **Performance Bottlenecks**: Redis caching and database query optimization
- **Data Integrity**: Automated testing and validation at all data entry points

---

9. Quality Assurance

9.1 Testing Strategy

9.1.1 Unit Testing

- **Coverage Target**: Minimum 80% code coverage
- **Framework**: Pest/PHPUnit for Laravel
- **Focus Areas**: Business logic services, data validation, calculations

9.1.2 Integration Testing

- **API Testing**: All REST endpoints with various input scenarios
- **Database Testing**: Transaction integrity and concurrency handling
- **Payment Testing**: Stripe integration with test card numbers

9.1.3 End-to-End Testing

- **User Workflows**: Complete customer journey from browsing to purchase
- **Admin Workflows**: Order management and fulfillment processes
- **Error Scenarios**: Payment failures, stock shortages, system errors

9.2 Code Quality Standards

- **PHP Standards**: PSR-12 code style compliance
- **Static Analysis**: PHPStan level 6+ implementation
- **Code Formatting**: Laravel Pint automated formatting
- **Documentation**: Comprehensive inline comments and README files

9.3 Security Testing

- **Vulnerability Scanning**: Regular security audit with automated tools
- **Penetration Testing**: Manual testing of authentication and authorization
- **Input Validation**: Comprehensive testing of all form inputs
- **SQL Injection Prevention**: Prepared statements and ORM usage

---

10. Appendices

Appendix A: File Structure Overview

The complete application structure follows Laravel conventions with MongoDB integration:

```
├── app/
|   ├── Http/Controllers/ (Admin, API, Customer, Supplier)
|   ├── Models/ (Product, Order, Inventory, Customer, etc.)
|   ├── Services/ (Business logic encapsulation)
|   ├── Repositories/ (Data access layer)
|   └── Jobs/ (Background processing)
├── resources/views/ (Blade templates by role)
├── database/migrations/ (MongoDB collection definitions)
├── config/ (Application configuration files)
└── routes/ (Web, API, and role-specific routes)
```

Appendix B: API Endpoint Documentation

Product Management

- GET /api/products - Retrieve product list with filtering
- POST /api/products - Create new product (admin only)
- PUT /api/products/{id} - Update product information
- DELETE /api/products/{id} - Soft delete product

Inventory Management

- GET /api/inventory/{variant_id} - Get current stock levels
- POST /api/inventory/{variant_id}/adjust - Manual stock adjustment
- GET /api/inventory/movements - Stock movement history

Order Processing

- POST /api/orders - Create new order
- GET /api/orders/{id} - Retrieve order details

- PUT /api/orders/{id}/status - Update order status
- POST /api/orders/{id}/ship - Create shipment

Appendix C: Database Indexing Strategy

Performance Indexes

- Products: compound index on (category_id, active, created_at)
- Orders: compound index on (customer_id, status, created_at)
- Inventory: unique index on variant_id
- Variants: unique index on sku

Search Indexes

- Products: text index on (name, description)
- Categories: text index on name
- Customers: compound index on (email, active)

Appendix D: Deployment Configuration

Production Environment Requirements

- **Server**: Minimum 4 CPU cores, 8GB RAM
- **Storage**: SSD with minimum 100GB available space
- **Network**: 100 Mbps connection with static IP
- **SSL Certificate**: Valid SSL certificate for HTTPS

Docker Compose Configuration

```
version: '3.8'
services:
  app:
    build: .
    ports:
      - "80:80"
    environment:
      - APP_ENV=production
  mongodb:
    image: mongo:latest
    volumes:
      - mongodb_data:/data/db
```

```
  redis:
    image: redis:alpine
    volumes:
      - redis_data:/data
```

Document Control

- **Version**: 1.0
- **Last Updated**: August 28, 2025
- **Next Review**: September 28, 2025
- **Approved By**: Development Team Lead