SOFTWARE REQUIREMENT SPECIFICATION (SRS)

HAN FOODS E-COMMERCE PLATFORM

Document: Software Requirement Specification
Project: Han Foods E-commerce Platform
Version: 1.0
Created Date: June 11, 2025
Created By: Phạm Nam Khánh
Status: Final

---

1. INTRODUCTION

1.1 Document Purpose

This document describes in detail the technical and functional requirements of the Han Foods E-commerce Platform system. This document serves as the foundation for system design, development, and testing.

1.2 Product Scope

Han Foods E-commerce Platform is a full-stack e-commerce system including: - **Frontend:** React.js application for customer and admin interfaces - **Backend:** Node.js/Express.js RESTful API - **Database:** MongoDB with Mongoose ODM - **Authentication:** JWT + Passport.js + Google OAuth2 - **Payment:** VietQR integration + Cash on Delivery

1.3 Definitions and Terminology

Term	Definition
JWT	JSON Web Token - Token authentication standard
OAuth2	Open Authorization framework
VietQR	Vietnam's QR code payment system
COD	Cash on Delivery - Payment upon receipt
CRUD	Create, Read, Update, Delete operations
SPA	Single Page Application
API	Application Programming Interface
ODM	Object Document Mapping

1.4 Reference Documents

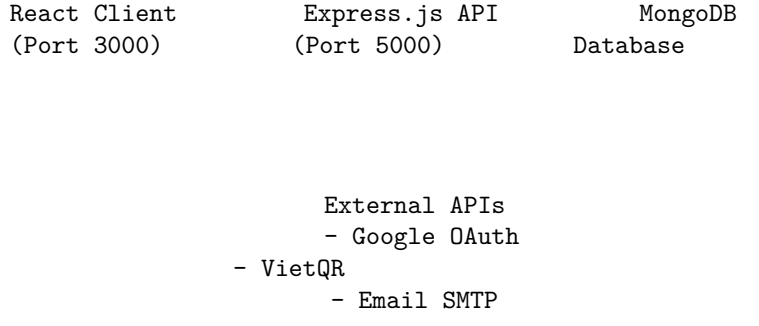
- Business Requirement Specification v1.0

- MongoDB Documentation
- React.js Official Guide
- Node.js Best Practices
- Material-UI Design System

---

2. SYSTEM OVERVIEW

2.1 System Architecture



2.2 Technology Stack

Frontend: - React.js 18 with Hooks and Context API - Material-UI (MUI) components - React Router v6 for navigation - Axios for API calls - Chart.js for data visualization

Backend: - Node.js with Express.js framework - Mongoose ODM for MongoDB - Passport.js for authentication - JWT for session management - Helmet for security headers - CORS middleware - Rate limiting protection

Database: - MongoDB with main collections: - Accounts (users) - Products - Categories - Orders - Carts - Wishlists - Addresses - Discounts - Reviews

2.3 Deployment Architecture

- **Development:** localhost:3000 (frontend), localhost:5000 (backend)
- **Production:** Server deployment with PM2, Nginx reverse proxy
- **Database:** MongoDB Atlas or self-hosted MongoDB
- **Static Files:** Local storage or CDN integration

---

3. DETAILED FUNCTIONAL REQUIREMENTS

3.1 Authentication & Authorization Module

3.1.1 User Registration Input Requirements:

```
{
  email: String (required, unique, email format),
  password: String (required, min 8 chars, alphanumeric),
  firstname: String (required),
  lastname: String (required),
  phone: String (optional, Vietnamese phone format),
  DOB: Date (optional),
  gender: Enum ['male', 'female', ''] (optional),
  address: {
    street: String,
    city: String,
    district: String,
    ward: String,
    postalCode: String,
    country: String (default: 'Vietnam')
  }
}
```

Process Flow: 1. Validate input data 2. Check email uniqueness 3. Hash password with bcrypt 4. Create account document 5. Generate JWT token 6. Return user data + token

API Endpoint:

```
POST /api/auth/register
Request Body: UserRegistrationData
Response: {
  success: Boolean,
  token: String,
  user: UserObject,
  message: String
}
```

3.1.2 User Login Authentication Methods: - Email/Password - Google OAuth2

Login Process: 1. Validate credentials 2. Authenticate using passport-local-mongoose 3. Generate JWT token (expires in 1 hour) 4. Setup refresh token mechanism 5. Return user data + tokens

API Endpoints:

```
POST /api/auth/login
POST /api/auth/google
```

```
GET /api/auth/google/callback
POST /api/auth/refresh-token
POST /api/auth/logout
```

3.1.3 Authorization Middleware Role-based Access Control:

```
// Middleware functions
isAuthenticated() // Verify JWT token
isAdmin() // Check admin role
isUserOrAdmin() // Allow user to access own data or admin
```

3.2 Product Management Module

3.2.1 Product Data Model

```
const ProductSchema = {
  productId: String (unique, auto-generated),
  name: String (required),
  description: String,
  price: Number (required, min: 0),
  salePrice: Number (default: 0),
  stock: Number (required, min: 0),
  categoryId: ObjectId (ref: Category),
  thumbnailImage: String (required),
  images: [String], // Array of additional images
  rating: Number (default: 4.5, range: 0-5),
  reviewCount: Number (default: 0),
  materials: String,
  dimensions: String,
  deleted: Boolean (default: false),
  createdBy: ObjectId (ref: Account),
  timestamps: true
}
```

3.2.2 Product CRUD Operations Create Product (Admin only):

```
POST /api/products
POST /api/products/with-images (with file upload)
```

Read Products:

```
GET /api/products?page=1&limit=10&category=&search=&sort=
GET /api/products/:id
GET /api/products/search?q=searchTerm
```

Update Product (Admin only):

```
PUT /api/products/:id
```

Delete Product (Admin only):

```
DELETE /api/products/:id
GET /api/products/:id/orders/check (check if product has orders)
```

3.2.3 Image Management

- Support multiple image formats (JPG, PNG, WebP)
- Thumbnail generation for listing views
- Image optimization for web delivery
- File upload with multer middleware

3.3 Shopping Cart Module

3.3.1 Cart Data Model

```
const CartSchema = {
  cartId: String (unique),
  userId: ObjectId (ref: Account, required),
  items: [{
    productId: ObjectId (ref: Product, required),
    quantity: Number (required, min: 1)
  }],
  deleted: Boolean (default: false),
  timestamps: true
}
```

3.3.2 Cart Operations API Endpoints:

```
GET /api/carts - Get user cart
POST /api/carts/add - Add item to cart
PUT /api/carts/update - Update item quantity
DELETE /api/carts/remove/:productId - Remove item
DELETE /api/carts/clear - Clear entire cart
```

Business Logic: - Validate product existence and stock - Merge duplicate products - Auto-remove items when stock = 0 - Persist cart across sessions

3.4 Order Management Module

3.4.1 Order Data Model

```
const OrderSchema = {
  orderId: String (unique, format: 'ORD-XXXXXXXX'),
  orderCode: String (unique, 8 chars),
  userId: ObjectId (ref: Account, required),
  email: String (required),
  totalAmount: Number (required, min: 0),
  status: Enum ['pending', 'processing', 'shipped', 'delivered', 'cancelled'],
  addressId: ObjectId (ref: Address, required),
```

```
  items: [{
    productId: ObjectId (ref: Product),
    name: String,
    price: Number,
    quantity: Number,
    subtotal: Number
  }],
  discountId: ObjectId (ref: Discount),
  discountAmount: Number (default: 0),
  paymentMethod: String,
  shippingFee: Number (default: 0),
  isPaid: Boolean (default: false),
  paidAt: Date,
  deleted: Boolean (default: false),
  timestamps: true
}
```

3.4.2 Order Processing Flow

1. **Validate Cart Items:**
   - Check product availability
   - Verify stock levels
   - Calculate pricing
2. **Address Handling:**
   - Use existing address or create new
   - Validate address format
3. **Apply Discounts:**
   - Validate discount code
   - Calculate discount amount
   - Update total amount
4. **Create Order:**
   - Generate unique order ID
   - Reserve inventory
   - Clear shopping cart
5. **Payment Processing:**
   - Handle COD orders
   - Process VietQR payments
   - Update payment status

3.4.3 Order API Endpoints

```
POST /api/orders - Create new order
GET /api/orders/user - Get user orders
GET /api/orders/:id - Get order details
PATCH /api/orders/:id/cancel - Cancel order
```

```
// Admin endpoints
GET /api/orders/admin - Get all orders
PATCH /api/orders/:id/status - Update order status
GET /api/orders/admin/stats - Order statistics
GET /api/orders/admin/export - Export orders
```

3.5 Payment Module

3.5.1 Payment Methods Cash on Delivery (COD): - No online payment required - Payment collected at delivery - Order can be modified before shipping

VietQR Payment: - Generate QR code for payment - Real-time payment verification - Automatic order confirmation

3.5.2 Payment API Endpoints

```
POST /api/payments/cash-on-delivery
POST /api/payments/vietqr
POST /api/payments/verify
GET /api/payments/order/:orderId
```

3.6 Admin Dashboard Module

3.6.1 Dashboard Statistics Key Metrics: - Total products, orders, users - Revenue analytics - Top selling products - Recent orders - Low stock alerts

API Endpoints:

```
GET /api/admin/dashboard/stats
GET /api/admin/revenue/analytics
GET /api/admin/dashboard/recent-orders
```

3.6.2 User Management Admin Capabilities: - View all users - Create new users - Update user status (active/inactive) - Update user roles - Soft delete users - Export user data

API Endpoints:

```
GET /api/admin/users
POST /api/admin/users
GET /api/admin/users/:id
PATCH /api/admin/users/:id/status
PATCH /api/admin/users/:id/role
DELETE /api/admin/users/:id
GET /api/admin/users/export
```

3.7 Discount Management Module

3.7.1 Discount Data Model

```
const DiscountSchema = {
  discountId: String (unique),
  code: String (unique, required),
  description: String,
  discountType: Enum ['percentage', 'fixed'],
  discountValue: Number (required, min: 0),
  minOrderValue: Number (default: 0),
  startDate: Date (required),
  endDate: Date (required),
  usageLimit: Number,
  usedCount: Number (default: 0),
  deleted: Boolean (default: false),
  timestamps: true
}
```

3.7.2 Discount Operations

```
GET /api/discounts - Get available discounts
POST /api/discounts/validate - Validate discount code
POST /api/discounts - Create discount (Admin)
PUT /api/discounts/:id - Update discount (Admin)
DELETE /api/discounts/:id - Delete discount (Admin)
```

---

4. USER INTERFACE REQUIREMENTS

4.1 Customer Interface Requirements

4.1.1 Responsive Design

- **Mobile-first approach:** Optimize for mobile devices first
- **Breakpoints:**
  - Mobile: < 768px
  - Tablet: 768px - 1024px
  - Desktop: > 1024px
- **Touch-friendly:** Buttons min 44px, adequate spacing

4.1.2 Navigation Structure

```
Header Navigation:
  Logo/Home
  Products (dropdown with categories)
  About Us
  Contact
  Cart (with item count badge)
  User Menu
      Login/Register (if not logged in)
```

```
      Profile/Orders/Logout (if logged in)

Footer:
  Company Info
  Customer Service
  Policies
  Social Media Links
```

4.1.3 Key Pages **Homepage:** - Hero section with call-to-action - Featured products grid - Product categories showcase - Customer testimonials - Company information

Product Listing: - Filter by category, price, rating - Sort by price, name, rating, date - Pagination - Product cards with image, name, price, rating

Product Detail: - Image gallery with zoom - Product information tabs - Add to cart/wishlist - Customer reviews - Related products

Shopping Cart: - Item list with quantities - Price calculations - Coupon code input - Checkout button

Checkout Process: - Step 1: Address selection/entry - Step 2: Payment method selection - Order confirmation

4.2 Admin Interface Requirements

4.2.1 Admin Layout

```
Admin Dashboard:
  Sidebar Navigation
      Dashboard
      Product Management
      Order Management
      Customer Management
      Discount Management
      Revenue Reports
      Settings
  Top Header
      Breadcrumb
      Search
      Notifications
      User Menu
  Main Content Area
```

4.2.2 Data Tables

- Sortable columns
- Filtering capabilities
- Bulk actions

- Pagination
- Export functionality
- Inline editing where appropriate

4.2.3 Forms

- Validation with real-time feedback
- File upload with preview
- Rich text editor for descriptions
- Date/time pickers
- Multi-select dropdowns

4.3 UI Component Library

Material-UI Components: - Buttons, Cards, Tables - Forms, Inputs, Selects - Modals, Snackbars, Progress indicators - Charts and data visualization

---

5. DATABASE REQUIREMENTS

5.1 Database Design

5.1.1 Collections and Relationships

```
Account          Address          Category


                                  Product
Cart


                                          Review

Wishlist          Order


                  Billing
```

5.1.2 Indexing Strategy **Primary Indexes:** - Account: email (unique), username (unique) - Product: productId (unique), name (text) - Order: orderId (unique), userId + createdAt - Category: categoryId (unique)

Compound Indexes: - Product: { categoryId: 1, price: 1 } - Order: { userId: 1, status: 1, createdAt: -1 } - Cart: { userId: 1, "items.productId": 1 }

5.1.3 Data Validation **MongoDB Schema Validation:**

```
// Example for Product collection
{
  $jsonSchema: {
    bsonType: "object",
    required: ["productId", "name", "price", "stock"],
    properties: {
      productId: { bsonType: "string" },
      name: { bsonType: "string", minLength: 1 },
      price: { bsonType: "number", minimum: 0 },
      stock: { bsonType: "int", minimum: 0 },
      email: {
        bsonType: "string",
        pattern: "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
      }
    }
  }
}
```

5.2 Data Security and Backup

5.2.1 Security Measures

- Password hashing with bcrypt (salt rounds: 12)
- Sensitive data encryption
- Input sanitization
- SQL injection prevention (MongoDB NoSQL injection)

5.2.2 Backup Strategy

- Daily automated backups
- Point-in-time recovery capability
- Backup verification and restore testing
- Offsite backup storage

---

6. API REQUIREMENTS

6.1 API Design Principles

- **RESTful architecture:** Standard HTTP methods
- **Consistent naming:** kebab-case URLs
- **Versioning:** /api/v1/ prefix for future versions
- **Status codes:** Proper HTTP status codes
- **Error handling:** Consistent error response format

6.2 API Response Format

```
// Success Response
{
  success: true,
  data: {},
  message: "Operation completed successfully",
  timestamp: "2025-06-11T10:30:00Z"
}

// Error Response
{
  success: false,
  error: {
    code: "VALIDATION_ERROR",
    message: "Invalid input data",
    details: {
      field: "email",
      issue: "Email format is invalid"
    }
  },
  timestamp: "2025-06-11T10:30:00Z"
}
```

6.3 Authentication Headers

```
Authorization: Bearer <jwt-token>
Content-Type: application/json
Accept: application/json
```

6.4 Rate Limiting

- **Public endpoints:** 100 requests/hour/IP
- **Authenticated endpoints:** 1000 requests/hour/user
- **Admin endpoints:** Unlimited (with monitoring)

6.5 API Documentation

- Swagger/OpenAPI specification
- Interactive API explorer
- Code examples for common operations
- Postman collection for testing

---

7. SECURITY REQUIREMENTS

7.1 Authentication Security

Token Management: - JWT access tokens: 1 hour expiration - Refresh tokens: 30 days expiration - Secure token storage (httpOnly cookies) - Token rotation on refresh

Password Security: - Minimum 8 characters - Must contain letters and numbers - Bcrypt hashing with salt rounds 12 - Password complexity validation

7.2 Authorization Controls

Role-based Access: - Customer: Limited to own data - Admin: Full system access - Middleware enforcement on all protected routes

7.3 Security Headers

```
// Helmet.js configuration
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      styleSrc: ["'self'", "'unsafe-inline'", "https:"],
      scriptSrc: ["'self'"],
      imgSrc: ["'self'", "data:", "https:"],
      connectSrc: ["'self'", "http://localhost:3000", "http://localhost:5000"]
    }
  },
  crossOriginEmbedderPolicy: false
}));
```

7.4 Input Validation and Sanitization

- Server-side validation for all inputs
- MongoDB injection prevention
- XSS protection
- File upload validation (type, size, content)

7.5 HTTPS and Transport Security

- Force HTTPS in production
- HSTS headers
- Secure cookie flags
- Certificate pinning (if applicable)

---

8. PERFORMANCE REQUIREMENTS

8.1 Response Time Requirements

Endpoint Type	Target Response Time
Static assets	$<$ 500ms
API calls	$<$ 1 second
Search queries	$<$ 2 seconds
Report generation	$<$ 5 seconds
File uploads	$<$ 10 seconds

8.2 Throughput Requirements

- **Concurrent users:** 500 simultaneous users
- **API requests:** 1000 requests/minute
- **Database queries:** $<$ 100ms average
- **Memory usage:** $<$ 512MB per instance

8.3 Optimization Strategies

Frontend Optimization: - Code splitting and lazy loading - Image optimization and compression - Browser caching headers - CDN integration for static assets - Minification of CSS/JS

Backend Optimization: - Database query optimization - Connection pooling - Redis caching for frequently accessed data - Compression middleware (gzip) - Pagination for large datasets

8.4 Monitoring and Analytics

- Application performance monitoring (APM)
- Database performance tracking
- Error rate monitoring
- User experience metrics
- Real-time alerts for performance degradation

---

9. DEPLOYMENT REQUIREMENTS

9.1 Development Environment

```
# Prerequisites
Node.js >= 14.x
MongoDB >= 4.4
npm >= 6.x
Git

# Environment Variables
NODE_ENV=development
PORT=5000
MONGO_URI=mongodb://localhost:27017/hanfoods
JWT_SECRET=your-jwt-secret
SESSION_SECRET=your-session-secret
GOOGLE_CLIENT_ID=your-google-client-id
GOOGLE_CLIENT_SECRET=your-google-client-secret
```

9.2 Production Deployment

Server Requirements: - **CPU:** 2+ cores - **RAM:** 4GB minimum, 8GB recommended - **Storage:** 50GB SSD - **OS:** Ubuntu 20.04 LTS or CentOS 8

Deployment Stack:

```
    Nginx Proxy     (Port 80/443)
    (Load Balancer)



        Node.js App     (Port 5000)
        (PM2 Cluster)



        MongoDB         (Port 27017)
        (Replica Set)
```

9.3 CI/CD Pipeline

```
# GitHub Actions workflow
stages:
  - Code Quality Check (ESLint, Prettier)
  - Unit Tests (Jest)
  - Integration Tests
```

```
  - Security Scan
  - Build Docker Image
  - Deploy to Staging
  - E2E Tests
  - Deploy to Production
```

9.4 Production Configuration

```
// Production environment variables
NODE_ENV=production
PORT=5000
MONGO_URI=mongodb://mongo-cluster/hanfoods?replicaSet=rs0
REDIS_URL=redis://redis-server:6379
JWT_SECRET=complex-production-secret
ENABLE_HTTPS=true
SSL_CERT_PATH=/etc/ssl/certs/cert.pem
SSL_KEY_PATH=/etc/ssl/private/key.pem
```

---

10. TESTING REQUIREMENTS

10.1 Unit Testing

Framework: Jest + Supertest

Coverage Requirements: - Controllers: 90% coverage - Services: 95% coverage - Utilities: 100% coverage - Models: 80% coverage

Test Categories:

```
// Example test structure
describe('ProductController', () => {
  describe('getAllProducts', () => {
    test('should return products with pagination', async () => {
      // Test implementation
    });

    test('should filter products by category', async () => {
      // Test implementation
    });

    test('should handle invalid pagination parameters', async () => {
      // Test implementation
    });
  });
});
```

10.2 Integration Testing

API Testing: - Test all API endpoints - Database integration tests - Authentication flow testing - Payment processing tests

Test Data: - Automated test data setup/teardown - Factory pattern for test data creation - Isolated test database

10.3 End-to-End Testing

Framework: Cypress or Playwright

Test Scenarios: - Complete user registration flow - Product browsing and searching - Shopping cart operations - Checkout process - Admin dashboard operations

10.4 Performance Testing

Tools: Artillery.js or K6

Test Scenarios: - Load testing with increasing user counts - Stress testing for peak loads - Endurance testing for extended periods - Spike testing for sudden traffic increases

10.5 Security Testing

- **OWASP ZAP:** Automated security scanning
- **Manual penetration testing:** Critical vulnerabilities
- **Dependency vulnerability scanning:** npm audit
- **Code security analysis:** SonarQube

---

11. MAINTENANCE AND SUPPORT REQUIREMENTS

11.1 Logging and Monitoring

Logging Strategy:

```
// Winston logger configuration
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
```

```
    new winston.transports.File({ filename: 'combined.log' }),
    new winston.transports.Console()
  ]
});
```

Monitoring Metrics: - Application uptime and health - Response times and error rates - Database performance - Memory and CPU usage - User activity patterns

11.2 Error Handling

Error Categories: - Validation errors (400) - Authentication errors (401) - Authorization errors (403) - Not found errors (404) - Server errors (500) - Database errors (503)

Error Response Format:

```
{
  success: false,
  error: {
    type: 'VALIDATION_ERROR',
    message: 'User-friendly error message',
    code: 'ERR_INVALID_EMAIL',
    timestamp: '2025-06-11T10:30:00Z',
    requestId: 'req_123456789'
  }
}
```

11.3 Backup and Recovery

Backup Schedule: - Database: Daily incremental, weekly full - Application files: Daily - Configuration files: On change - User uploads: Real-time sync

Recovery Procedures: - RTO (Recovery Time Objective): 4 hours - RPO (Recovery Point Objective): 1 hour - Automated failover procedures - Disaster recovery runbook

11.4 Documentation Requirements

Technical Documentation: - API documentation (OpenAPI/Swagger) - Database schema documentation - Deployment guides - Troubleshooting guides

User Documentation: - Admin user manual - Customer help center - API integration guides - FAQ and support articles

---

12. APPENDICES

12.1 Database Schema Examples

Account Collection

```
{
  _id: ObjectId("..."),
  email: "user@example.com",
  username: "user@example.com",
  role: "customer", // "customer" | "admin"
  firstname: "John",
  lastname: "Doe",
  DOB: ISODate("1990-01-01"),
  gender: "male", // "male" | "female" | ""
  phone: "+84901234567",
  avatar: "https://example.com/avatar.jpg",
  isActive: true,
  deleted: false,
  createdAt: ISODate("2025-06-11T10:30:00Z"),
  updatedAt: ISODate("2025-06-11T10:30:00Z")
}
```

Product Collection

```
{
  _id: ObjectId("..."),
  productId: "PROD-ABC12345",
  name: "Handmade Coconut Cup",
  description: "Cup made from natural coconut shell...",
  price: 150000,
  salePrice: 120000,
  stock: 50,
  categoryId: ObjectId("..."),
  thumbnailImage: "https://example.com/product-thumb.jpg",
  images: [
    "https://example.com/product-1.jpg",
    "https://example.com/product-2.jpg"
  ],
  rating: 4.5,
  reviewCount: 23,
  materials: "Natural coconut shell",
  dimensions: "Diameter: 8cm, Height: 10cm",
  deleted: false,
  createdBy: ObjectId("..."),
  createdAt: ISODate("2025-06-11T10:30:00Z"),
  updatedAt: ISODate("2025-06-11T10:30:00Z")
```

```
}
```

12.2 API Endpoint Examples

Product Management

```
// GET /api/products
{
  "success": true,
  "products": [...],
  "pagination": {
    "currentPage": 1,
    "totalPages": 5,
    "totalItems": 48,
    "hasNextPage": true,
    "hasPrevPage": false
  }
}

// POST /api/products (Admin only)
{
  "name": "New Product",
  "description": "Product description",
  "price": 100000,
  "salePrice": 80000,
  "stock": 20,
  "categoryId": "category_id",
  "thumbnailImage": "image_url",
  "images": ["image1_url", "image2_url"]
}
```

12.3 Error Code Reference

Error Code	HTTP Status	Description
AUTH_REQUIRED	401	Authentication required
INVALID_CREDENTIALS	401	Invalid login credentials
ACCESS_DENIED	403	Insufficient permissions
RESOURCE_NOT_FOUND	404	Requested resource not found
VALIDATION_ERROR	400	Input validation failed
DUPLICATE_ENTRY	409	Resource already exists
INSUFFICIENT_STOCK	400	Not enough product stock
INVALID_DISCOUNT_CODE	400	Discount code invalid or expired
PAYMENT_FAILED	402	Payment processing failed
SERVER_ERROR	500	Internal server error

Completion Date: June 11, 2025
Created By: Phạm Nam Khánh
Email: khanhpn31@gmail.com
Role: Lead Developer & Solution Architect
Status: Production Ready

Note: This document describes the technical details of the system that has been deployed and is operating stably. All changes need to be reviewed and approved before implementation.