# Software Requirements Specification (SRS)

**Project Title:** BizzBuy - E-Commerce and Auction Platform

**Group name:** Error 404

**Team Members:**

- Abhinav (IMT2024007)
- Raj Vardhan (IMT2024003)
- Rohith Sai (BT2024144)
- Pranay Sanjan (IMT2024024)
- Suhith Reddy (BT2024066)
- Rohith.V (IMT2024055)

# 1. Introduction

## 1.1 Purpose

The purpose of this project is to create an e-commerce and auction platform that enables users to buy and sell products through both traditional shopping and bidding.

## 1.2 Scope

The system will provide the following functionalities:

- User registration and authentication with role-based access (Buyer, Seller)
- Product listing and management with inventory tracking
- Shopping cart functionality with checkout and payment processing
- Auction system with bidding capabilities
- Order tracking and transaction history
- Product search and filtering capabilities

The backend will be built in Java + Spring Boot, and data will be stored using file handling (JSON format).

## 1.3 Intended Users

- Buyers - Customers who purchase products and participate in auctions
- Sellers - Vendors who list products and create auctions

## 2. System Overview

The system will take input using REST API endpoints, process it through Java classes (Model → Service → Controller), and store the data in JSON files.

The architecture follows a layered approach:

- Controller Layer - Handles HTTP requests and responses
- Service Layer - Contains core logic
- Model Layer - Defines structure of the objects
- Utility Layer - Provides JSON file management and ID generation

## 3. System Features

### 3.1 Feature 1: User Authentication & Profile Management

**Description:** Secure user registration and login system with role-based access control. Users can manage their profiles and digital wallets.
**Functionalities:**

- User registration with role selection (BUYER, SELLER)
- User login with username/password authentication
- Wallet balance management and fund addition
- Transaction history tracking

### 3.2 Feature 2: Product Management

**Description:** Comprehensive product listing system allowing sellers to create, update, and manage their inventory.
**Functionalities:**

- Create product listings with details (name, description, price, stock, images, tags)
- Update product stock quantities
- Browse all products and search products by keyword
- Filter products by price range, seller, and auction status
- View products by specific seller

### 3.3 Feature 3: Shopping Cart & Checkout

**Description:** Shopping cart system with checkout and payment processing (just a simulation).
**Functionalities:**

- Add products to cart with quantity selection
- Update cart item quantities and remove items from cart
- View cart with total calculation
- Checkout with wallet-based payment
- Automatic wallet transfers and inventory updates

### 3.4 Feature 4: Auction System

**Description:** Auction platform for bidding on products.
**Functionalities:**

- Create auctions with starting price and status
- View active auctions
- Track highest bidder in real time
- Auction closure and winner determination

### 3.5 Feature 5: Bidding Mechanism

**Description:** Bidding system with validation and tracking.
**Functionalities:**

- Place bids higher than current price
- View all bids for an auction
- View personal bidding history
- Automatic wallet deduction for winning bids

### 3.6 Feature 6: Wallet & Transactions

**Description:** Digital wallet system for financial transactions.
**Functionalities:**

- Initialize wallet with zero balance on registration
- Add funds to wallet
- Automatic deductions for purchases
- Automatic credits for sales
- Complete transaction history
- Balance verification before purchases

## 3.7 Feature 7: Order Management

**Description:** Order tracking and history for buyers.
**Functionalities:**

- Create orders from cart checkout
- Track order status (CREATED, PAID, SHIPPED, COMPLETED, CANCELLED)
- View order history and details

# 4. API Endpoints

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/auth/register | Register new user with role |
| POST | /api/auth/login | Login with username/password |
| GET | /api/items | Get all products |
| GET | /api/items/search?keyword={text} | Search products by keyword |
| GET | /api/items/filter?min={price} max={price} | Filter products by criteria |
| GET | /api/items/{id} | Get single product details |
| GET | /api/items/seller/{id} | Get products by seller |
| POST | /api/items/products | Create new product (Seller) |
| PUT | /api/items/{id}/stock?stock={qty} | Update product stock (Seller) |
| GET | /api/cart/my-cart | View current user's cart |
| POST | /api/cart/add | Add item to cart |
| PUT | /api/cart/update | Update cart item quantity |
| DELETE | /api/cart/remove/{productId} | Remove item from cart |
| POST | /api/cart/checkout | Checkout and create order |
| GET | /api/auctions/active | Get all active auctions |
| GET | /api/auctions/{id} | Get auction details |
| POST | /api/auctions/create | Create new auction (Seller) |
| POST | /api/auctions/{id}/close | Close auction (Seller) |
| POST | /api/bids/place | Place bid on auction |
| GET | /api/bids/auction/{id} | Get all bids for auction |
| GET | /api/bids/my-bids | Get current user's bids |
| GET | /api/users/me | Get current user profile |
| PUT | /api/users/me | Update profile information |
| GET | /api/users/me/wallet | View wallet balance |
| POST | /api/users/me/wallet/add?amount ={amt} | Add funds to wallet |
| GET | /api/users/me/orders | View order history |

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/users/me/transactions | View transaction history |

# 5. Data Storage Using Files

## 5.1 File Type: JSON

## 5.2 File Name

All data files are stored in src/main/resources/data/:

- users.json - User accounts and authentication
- products.json - Product listings
- auctions.json - Auction items
- bids.json - Bid history
- carts.json - Shopping carts
- wallets.json - User wallet balances
- orders.json - Order records
- transactions.json - Financial transactions

## 5.3 File Format Example

**users.json:**

```
[{"id": 1, "username": "alice_seller", "password": "pass123",
"email": "alice@bizzbuy.com", "fullName": "Alice Smith",
"role": "SELLER"}]
```

**products.json:**

```
[{"id": 1,"sellerId": 1,"name": "iPhone 15 Pro",
"description": "Latest iPhone","price": 999.99,
"stockQuantity": 15,"isAuction": false,
"images": ["url1", "url2"],"tags": ["electronics", "phone"]}]
```

**wallets.json:**

```
[{"userId": 1,"balance": 2500.00,"currency": "USD"}]
```

**auctions.json:**

```
[{"id": 1,"itemId": 2,"sellerId": 1,"currentPrice": 1800.00,
"startingPrice": 1500.00,"startTime": "2025-11-22T10:00:00",
"endTime": "2025-11-25T18:00:00","status": "LIVE","winnerId": 2}]
```

**transactions.json:**

```
[{"id": 1,"payerId": 2,"payeeId": 1,"amount": 999.99,"status": "SUCCESS",
"timestamp": "2025-11-21T17:30:00", "reference": "Order #1" }]
```

## 6. Data Model (Classes)

**Class Name: User**
**Attributes:**

- id : Long
- username : String
- password : String
- email : String
- fullName : String
- role : Role (enum: BUYER, SELLER)
- enabled : boolean

**Class Name: Product**
**Attributes:**

- id : Long
- sellerId : Long
- name : String
- description : String
- price : Double
- stockQuantity : Integer
- isAuction : Boolean
- images : `List<String>`
- tags : `List<String>`

**Class Name: Auction**
**Attributes:**

- id : Long
- itemId : Long
- sellerId : Long
- currentPrice : Double
- startingPrice : Double
- startTime : LocalDateTime
- endTime : LocalDateTime

- status : AuctionStatus (enum: SCHEDULED, LIVE, ENDED, CANCELLED)
- winnerId : Long

**Class Name: Bid**
**Attributes:**

- id : Long
- auctionId : Long
- bidderId : Long
- amount : Double
- timestamp : LocalDateTime

**Class Name: Cart**
**Attributes:**

- userId : Long
- items : `List<CartItem>`
- totalAmount : Double

**Class Name: CartItem**
**Attributes:**

- productId : Long
- quantity : Integer
- price : Double

**Class Name: Order**
**Attributes:**

- id : Long
- buyerId : Long
- items : `List<CartItem>`
- totalAmount : Double
- date : LocalDateTime
- status : OrderStatus (enum: CREATED, PAID, SHIPPED, COMPLETED, CANCELLED)

**Class Name: Wallet**
**Attributes:**

- userId : Long
- balance : Double

- currency : String

**Class Name: Transaction**
**Attributes:**

- id : Long
- payerId : Long
- payeeId : Long
- amount : Double
- status : TransactionStatus (enum: PENDING, SUCCESS, FAILED)
- timestamp : LocalDateTime

## 7. Technology Stack

- Programming Language: Java 21
- Framework: Spring Boot 3.5.8
- Build Tool: Maven
- Development Tool: IntelliJ IDEA
- Testing Tool: Postman
- Storage: File Handling (JSON)
- Libraries: Lombok, Spring Web

## 8. Sample Input/Output

**Sample Request (POST /api/auth/register):**

```
{
  "username": "bob_buyer",
  "password": "pass123",
  "email": " bob@bizzbuy.com ",
  "fullName": "Bob Johnson",
  "role": "BUYER"
}
```

**Sample Response:**

```
{
  "id": 2,
  "username": "bob_buyer",
  "password": "pass123",
  "email": " bob@bizzbuy.com ",
  "fullName": "Bob Johnson",
  "role": "BUYER",
}
```

**Sample Request (POST /api/cart/add):**

```
{
  "productId": 1,
  "quantity": 2
}
```

**Sample Response (Cart):**

```
{
  "userId": 2,
  "items": [
    {"productId": 1,"quantity": 2,"price": 999.99}
  ],
  "totalAmount": 1999.98
}
```

## 9. Assumptions

- File-based storage for the application scale
- Auction timing is managed by the service layer without scheduled tasks
- All users start with a $0 wallet balance
- Unique IDs are generated sequentially using IdGenerator utility

## 10. Limitations

- No database is used; only file handling (JSON)
- No real payment gateway integration - wallet-based mock payments
- No concurrent access control for file operations
- Passwords are stored in plain text for simplicity