Project: Online Auction System

1. Introduction

This document outlines the **Low-Level Design (LLD)** for an **Online Auction System**, allowing users to list products for auction, place bids, and manage transactions in a secure and transparent manner.

This design supports both **Java (Spring Boot)** and **.NET (ASP.NET Core)** frameworks for backend development.

2. Module Overview

2.1 User Management

- Handles authentication and authorization.
- Supports different roles: **Buyer, Seller, and Admin**.

2.2 Auction & Product Management

- Sellers can create auctions for products.
- Buyers can browse and participate in live auctions.

2.3 Bidding System

- Buyers can place real-time bids.
- The system updates and notifies users about the highest bid.

2.4 Payment & Transactions

- Handles secure payments and bid settlements.
- Supports multiple payment methods.

2.5 Review & Rating

- Buyers can review sellers and products.
- Sellers can rate buyers based on transactions.

3. Architecture Overview

3.1 Architectural Style

- **Frontend**: Angular or React

- **Backend**: REST API-based architecture
- **Database**: Relational Database (MySQL/PostgreSQL/SQL Server)

3.2 Component Interaction

- The frontend interacts with the backend via REST APIs.
- The backend handles user authentication, business logic, and database operations.
- Users access the platform through a web-based interface.

4. Module-Wise Design

4.1 User Management Module

4.1.1 Features

- User registration and authentication (**Buyer, Seller, Admin**).
- Role-based access control for auction features.

4.1.2 Data Flow

1. Users register through the frontend.
2. The backend validates user data and assigns a role.
3. User details are stored in the database.
4. On login, an authentication token is generated.

4.1.3 Entities

- **User** (UserID, Name, Email, Password, Role, ContactNumber)

4.2 Auction & Product Management Module

4.2.1 Features

- Sellers can list products for auction with images, descriptions, and starting prices.
- Buyers can view auction listings and product details.

4.2.2 Data Flow

1. Sellers submit product details via the frontend.
2. The backend validates and stores auction data.
3. Buyers browse active auctions using filters (category, price, time left, etc.).

4. The system returns relevant product listings.

4.2.3 Entities

- **Product** (ProductID, SellerID, Title, Description, StartPrice, Category, Status)
- **Auction** (AuctionID, ProductID, StartDate, EndDate, CurrentBid, Status)
- **Images** (ImageID, ProductID, URL)

4.3 Bidding System Module

4.3.1 Features

- Buyers place real-time bids on auctions.
- Automatic bid updates and notifications for outbids.

4.3.2 Data Flow

1. A buyer places a bid via the frontend.
2. The backend validates the bid amount.
3. If the bid is highest, the auction's current bid updates.
4. Outbid users receive notifications.
5. When the auction ends, the highest bidder is declared the winner.

4.3.3 Entities

- **Bid** (BidID, AuctionID, BuyerID, Amount, BidTime)

4.4 Payment & Transactions Module

4.4.1 Features

- Secure payment processing for winning bids.
- Integration with payment gateways.

4.4.2 Data Flow

1. The winning buyer makes payment via the frontend.
2. The backend validates the transaction via a payment gateway.
3. If successful, the auction is marked as **Sold**, and payment details are stored.
4. The seller receives a notification to ship the item.

4.4.3 Entities

- **Transaction** (TransactionID, BuyerID, AuctionID, Amount, PaymentStatus, PaymentDate)

4.5 Review & Rating Module

4.5.1 Features

- Buyers can submit reviews and ratings for sellers.
- Sellers can rate buyers based on transactions.

4.5.2 Data Flow

1. A user submits a review via the frontend.
2. The backend stores the review and links it to the relevant user/product.
3. Reviews and ratings are displayed on user profiles.

4.5.3 Entities

- **Review** (ReviewID, UserID, TargetUserID, Rating, Comment, Date)

5. Deployment Strategy

5.1 Local Deployment

- **Frontend Deployment**: Runs on a local development server (ng serve for Angular, local React server).
- **Backend Deployment**: Spring Boot/ASP.NET Core application running on a local server.
- **Database**: MySQL/SQL Server set up for development.

6. Database Design

6.1 Tables and Relationships

- **User** (UserID, Name, Email, Password, Role, ContactNumber)
- **Product** (ProductID, SellerID, Title, Description, StartPrice, Category, Status)
- **Auction** (AuctionID, ProductID, StartDate, EndDate, CurrentBid, Status)
- **Bid** (BidID, AuctionID, BuyerID, Amount, BidTime)
- **Transaction** (TransactionID, BuyerID, AuctionID, Amount, PaymentStatus, PaymentDate)
- **Review** (ReviewID, UserID, TargetUserID, Rating, Comment, Date)

7. User Interface Design

7.1 Wireframes

- **Buyer Dashboard**: Browse auctions, place bids, manage payments.
- **Seller Dashboard**: List products, track auction progress, view transactions.
- **Admin Dashboard**: Manage users, auctions, and transactions.
- **Auction Page**: Displays product details, bidding history, countdown timer.

8. Non-Functional Requirements

8.1 Performance

- The system should handle up to **5000 concurrent users**.

8.2 Security

- **JWT-based authentication** and **role-based access control**.
- Secure bid placement with **real-time validation**.

8.3 Usability

- **Responsive UI** with real-time auction updates.

9. Assumptions and Constraints

9.1 Assumptions

- Users must create an account before participating in an auction.
- Auction winners must complete payments within **24 hours**.

9.2 Constraints

- Payments are processed via **third-party gateways**.
- **No mobile app** (only web-based operations).