**Bidding site - Software Design Document (SDD)**

**1. Introduction**

**1.1 Purpose**

The purpose of this Software Design Document (SDD) is to describe the design of the **Full-Stack Online Bidding System**. This application allows users to participate in an online auction, register accounts, list auction items, place bids, and manage their profiles.

**1.2 Scope**

The system will provide a web-based platform with two primary functionalities:

1. **Auction Management**: Users can create, view, update, and delete auction items.
2. **Bidding Functionality**: Users can place bids, view current bids, and receive notifications for outbid events.

**1.3 Intended Audience**

This document is intended for:

* **Developers** working on implementing the system
* **Testers** involved in system testing
* **Project managers** overseeing the development
* **Stakeholders** interested in understanding the system's design

**1.4 Definitions, Acronyms, and Abbreviations**

* **SDD**: Software Design Document
* **API**: Application Programming Interface
* **JWT**: JSON Web Tokens
* **CRUD**: Create, Read, Update, Delete

**2. System Overview**

The **Full-Stack Online Bidding System** is a web application that uses the following technology stack:

* **Frontend**: React.js
* **Backend**: Node.js with Express.js
* **Database**: MongoDB

The system is divided into two major parts:

1. **Frontend**: The user interface built with React for interacting with the system.
2. **Backend**: The server-side logic and database interaction powered by Node.js and MongoDB.

**3. System Architecture**

**3.1 Architecture Diagram**

A diagram of a computer flowchart

Description automatically generated

**3.2 Components Overview**

* **Frontend (React.js)**:
  + User interface for all actions including registration, login, creating/viewing auction items, and placing bids.
  + Communicates with the backend using RESTful API calls.
* **Backend (Node.js + Express)**:
  + Provides APIs for authentication, auction management, and bidding functionality.
  + Handles business logic and interactions with MongoDB.
* **Database (MongoDB)**:
  + Stores user data (credentials, profiles), auction items, and bids in collections.

**3.3 Technologies**

* **Frontend**: React.js (UI framework)
* **Backend**: Node.js, Express.js (Web framework for RESTful APIs)
* **Database**: MongoDB (NoSQL database)
* **Authentication**: JWT (JSON Web Token) for secure user authentication

**4. Functional Design**

**4.1 User Stories & Functionalities**

**4.1.1 User Authentication**

* **Registration**: User can create an account by providing a username, email, and password.
* **Login**: Users can log in with their credentials to access their profile and bidding system.
* **Logout**: Users can log out to end the session and invalidate the JWT token.

**4.1.2 Auction Management**

* **Create Auction**: Users can list new auction items with a title, description, starting bid, and end date.
* **View Auction Items**: Users can browse all available auction items.
* **Update Auction**: Users can modify the details of their listed auction items.
* **Delete Auction**: Users can delete auction items they no longer wish to sell.

**4.1.3 Bidding Functionality**

* **Place Bid**: Users can place a bid on an auction item.
* **View Bid Information**: Users can see the highest bid, bid history, and remaining time for an auction.
* **Receive Notifications**: Users are notified when they are outbid on an item.

**5. System Design**

**5.1 Frontend Design**

The frontend will be developed using **React.js**, which will interact with the backend through RESTful APIs.

**5.1.1 Pages & Components**

* **Home Page**: Displays the list of available auction items.
* **Login/Signup Page**: Allows users to log in or register.
* **Profile Page**: Users can view and edit their profiles.
* **Create Auction Page**: Form to create a new auction item.
* **Auction Detail Page**: Shows auction details, current bids, and bid history.
* **Bidding Page**: Allows users to place a bid.

**5.1.2 State Management**

* **React Context API** will be used to manage user authentication state and global auction data.

**5.1.3 UI/UX Design**

* **Bootstrap** will be used for responsive design and UI components.

**5.2 Backend Design**

The backend will be developed using **Node.js** and **Express.js**.

**5.2.1 API Endpoints**

**Authentication**

* **POST** /api/users/register: Registers a new user.
* **POST** /api/users/login: Logs in a user and provides a JWT token.

**Auction Management**

* **GET** /api/auctions: Retrieves all available auctions.
* **POST** /api/auctions: Creates a new auction item.

**Body:**

{

    "title": "Bolt Premium head phones",

    "description": "Headphones.",

    "imageName":"headphones.PNG",

    "startingBid": 30,

    "endDate": "2024-12-01T00:00:00.000Z"

}

* **PUT** /api/auctions/:id: Updates an auction item.

{

    "title": "Updated Title",

    "description": "Updated description",

    "startingBid": 600

}

* **DELETE** /api/auctions/:id: Deletes an auction item.

**Bidding**

* **POST** /api/:auctionId/bid: Places a bid on a specified auction item.

{

    "amount": 920

}

* **GET** /api/:auctionId/bids: Retrieves bid history and current highest bid for an auction item.

**5.2.2 Authentication Flow**

* Users will authenticate via **JWT**. Upon successful login, the backend will issue a signed JWT token, which the frontend will store (typically in local storage or cookies).
* Each API request (except public ones like login/register) will include this token in the Authorization header.

**5.2.3 Database Design (MongoDB)**

**Collections**

* **Users**: Stores user details such as username, email, hashed password, and profile information.
* **Auctions**: Stores auction items with fields such as title, description, starting bid, and end date.
* **Bids**: Stores bid details, linked to both users and auction items.

**Schema Design**

1. **User Schema**:
2. // models/User.js
3. const mongoose = require('mongoose');
4. const userSchema = new mongoose.Schema({
5. name: { type: String, required: true },
6. email: { type: String, required: true, unique: true },
7. password: { type: String, required: true },
8. }, {
9. timestamps: true, // Adds createdAt and updatedAt fields
10. });
11. const User = mongoose.model('User', userSchema);
12. module.exports = User;
13. **Auction Schema**:
14. const mongoose = require('mongoose');
15. const auctionSchema = new mongoose.Schema({
16. title: {
17. type: String,
18. required: true,
19. trim: true,
20. },
21. description: {
22. type: String,
23. required: true,
24. },
25. imageName: {
26. type: String,
27. required: true,
28. },
29. startingBid: {
30. type: Number,
31. required: true,
32. min: 0,
33. },
34. endDate: {
35. type: Date,
36. required: true,
37. },
38. user: {
39. type: mongoose.Schema.Types.ObjectId,
40. ref: 'User', // Reference to the user who created the auction
41. required: true,
42. },
43. bids: [
44. {
45. user: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
46. amount: { type: Number, required: true },
47. createdAt: { type: Date, default: Date.now }
48. }
49. ]
50. }, { timestamps: true });
51. module.exports = mongoose.model('Auction', auctionSchema);

**6. Non-Functional Requirements**

**6.1 Performance**

* The system must be capable of handling a large number of concurrent users and bids with low latency, particularly during peak times.

**6.2 Security**

* User authentication will be handled using **JWT tokens**.
* Passwords will be hashed and salted using **bcrypt** before storing in the database.
* API endpoints that require authentication will be protected using middleware to verify the JWT token.

**6.3 Scalability**

* The system is designed to scale horizontally, with the possibility of adding additional servers and replicating the database as needed.

**6.4 Usability**

* The frontend will be designed with a focus on a simple, intuitive user interface.
* Responsive design principles will be followed to ensure accessibility across devices (desktops, tablets, and mobiles).

**7. Conclusion**

This Software Design Document outlines the key architectural decisions, components, and interactions for the **Full-Stack Online Bidding System**. By following this design, we aim to create a secure, scalable, and user-friendly platform that meets the functional and non-functional requirements outlined above.