**Student Online Auction System**

**-Software Design Specification (SDS)**

**Team – 07**

**Team Members:**

|  |  |
| --- | --- |
| Manaswitha Reddy Singam | - [se22uari201@mahindrauniversity.edu.in](mailto:se22uari201@mahindrauniversity.edu.in) |
| Gautham C Reddy Musuku | - [se22uari103@mahindrauniversity.edu.in](mailto:se22uari103@mahindrauniversity.edu.in) |
| M Abhilash Reddy | - [se22uari083@mahindrauniversity.edu.in](mailto:se22uari083@mahindrauniversity.edu.in) |
| N Sai Shivan Raj | - [se22uari106@mahindrauniversity.edu.in](mailto:se22uari106@mahindrauniversity.edu.in) |
| Soma Mrudula | - [se22uari163@mahindrauniversity.edu.in](mailto:se22uari163@mahindrauniversity.edu.in) |
| Challa Mokshagna Reddy | - [se22uari032@mahindrauniversity.edu.in](mailto:se22uari032@mahindrauniversity.edu.in) |

**1. Introduction**

The introduction of the Software Design Specifications provides an overview of the **Student Online Auction System** software design. It includes the purpose, scope, definitions, references, and an overview of the design document.

**1.1 Purpose**

The purpose of this Software Design Specification (SDS) is to present a detailed design of the **Student Online Auction System**. It serves as a guide for the development team to implement the system and for stakeholders to understand the design decisions.  
This document is intended for:

-Developers

-Testers

-Project Managers

-Clients

Each group is expected to use this document to understand how the system works behind the scenes and how different components interact.

**1.2 Scope**

This SDS outlines the design of the **Student Online Auction System**, a web-based platform where students can buy and sell second-hand items. The system supports user registration, item listing, bidding functionality, and an admin module for managing users and items.

The backend is built with **Node.js and Express**, and the database used is **MongoDB**. The frontend is a separate React or HTML/CSS interface that communicates with the backend via RESTful APIs.

**1.3 Definitions, Acronyms, and Abbreviations**

| **Term** | **Definition** |
| --- | --- |
| SDS | Software Design Specification |
| API | Application Programming Interface |
| UI | User Interface |
| CRUD | Create, Read, Update, Delete |
| HTTP | Hypertext Transfer Protocol |
| JSON | JavaScript Object Notation |
| DB | Database |
| ODM | Object Document Mapper (Mongoose) |

**1.4 References**

* Node.js Documentation
* [Express.js Documentation](https://expressjs.com/)
* [MongoDB Documentation](https://www.mongodb.com/docs/)
* [React Documentation (Frontend)](https://react.dev/)
* Project Requirements Document (Internal)
* Software Engineering Class Notes / Book References

**2. Use Case View**

This section provides the key use cases of the system. These use cases represent the core functionalities of the Student Online Auction System such as user registration, item listing, bidding, and administrative control.

**2.1 Use Case**

Key use cases include:

**-User Registration and Login**  
Students create accounts and log in using secure credentials.

**-Item Listing by Sellers**  
A student can list an item for auction by providing details such as title, description, base price, and image.

**-Bidding by Buyers**  
Buyers can view items and place bids within a set bidding window.

**-Item Search and View**  
All users can browse or search for active listings.

**-Admin Controls**  
Admin users can manage user accounts, edit or delete inappropriate listings, and monitor overall platform activity.

**3. Design Goals and Constraints**

The **Student Online Auction System** is designed with the following goals and constraints:

-Provide a secure and user-friendly online platform for students to auction and bid on second-hand items.

-Ensure real-time updates for bids and listings using efficient backend processing.

-Maintain data consistency and integrity using MongoDB.

-Deliver a responsive user interface compatible with desktops and mobile devices.

-Use RESTful APIs to ensure seamless communication between frontend and backend.

-Constraints include limited development time, student-specific user authentication, and using Node.js, Express, and MongoDB stack.

* 1. **Design Assumptions**

-All users are verified students of the university.

-Users will have internet access to interact with the system.

-Admins will periodically review flagged listings or suspicious activity.

-MongoDB is assumed to be hosted on a reliable cloud platform (e.g., MongoDB Atlas).

-The frontend and backend will be deployed separately and communicate over HTTP.

**3.3 Significant Design Packages**

The system is divided into the following major packages:

**-User Module**: Manages registration, login, and profile updates.

**-Item Listing Module**: Handles listing of items for auction.

**-Bidding Module**: Controls bidding logic and winner selection.

**-Admin Module**: Provides features to manage users and monitor listings.

**-Notification Module**: Sends email or in-app alerts for bid status, item expiry, etc.

**-API Layer**: Routes frontend requests to backend services.

**-Database Access Layer**: Mongoose-based models that interact with MongoDB.

**3.4 Dependent External Interfaces**

|  |  |  |
| --- | --- | --- |
| **External Application and Interface Name** | **Module Using the Interface** | **Functionality / Description** |
| Cloud Email API (e.g., SendGrid or SMTP) | Notification Module | Sends bid updates, alerts, and confirmations to users. |
| MongoDB Cloud Database | All modules | Stores all user, item, and bid-related data. |
| Payment Gateway (Future Extension) | Checkout Module (future) | Process payments for auction items (if needed). |

**3.5 Implemented Application External Interfaces (and SOA Web Services)**

| **Interface Name** | **Module Implementing the Interface** | **Functionality / Description** |
| --- | --- | --- |
| /api/register | User Module | Registers a new student user. |
| /api/login | User Module | Authenticates a user and generates a session token. |
| /api/items | Item Listing Module | Adds, updates, or deletes auction items. |
| /api/bid/:itemId | Bidding Module | Places a bid on an item. |
| /api/admin/users | Admin Module | Allows admins to manage user accounts. |

**4 Logical View**

This section outlines how the application modules interact to implement key use cases, broken down into modules and classes with responsibilities.

**4.1 Design Model**

The system is decomposed into the following modules and classes:

**User Module**

**-User Manager**

* + *Responsibilities*: User registration, login, and authentication.
  + *Attributes*: userID, username, email, passwordHash, token.
  + *Operations*: registerUser(), loginUser(), authenticateToken()

**Item Module**

**-Item Manager**

* + *Responsibilities*: Add/edit/delete item listings.
  + *Attributes*: itemID, title, description, price, images, sellerID, endDate.
  + *Operations*: createItem(), updateItem(), deleteItem(), getItemsByUser()

**Bidding Module**

**-Bid Manager**

* + *Responsibilities*: Place bids, track highest bidder, close auction.
  + *Attributes*: bidID, itemID, bidderID, amount, timestamp.
  + *Operations*: placeBid(), getHighestBid(), finalizeAuction()

**Admin Module**

**-Admi Manager**

* + *Responsibilities*: Manage users and monitor activity.
  + *Operations*: viewAllUsers(), blockUser(), removeItem()

**Notification Module**

* **Notifier**
  + *Responsibilities*: Send notifications via email.
  + *Operations*: sendEmail(to, subject, body)

The classes interact through service calls and API endpoints, using MVC pattern and RESTful interfaces.

**4.2 Use Case Realization**

**Use Case: Post an Item**

**Sequence of Interactions**:

1. User submits item form via frontend.
2. Frontend calls /api/items endpoint.
3. Backend ItemManager.createItem() validates and saves item to DB.
4. Response sent back to user with item details.

**Use Case: Place a Bid**

1. User submits bid via item page.
2. API call to /api/bid/:itemId.
3. BidManager.placeBid() checks bid amount, updates if valid.
4. Notify previous highest bidder via Notifier.sendEmail().

Sequence and activity diagrams can be added to show flow of data through controllers, services, and DB layers for each use case.

**5 Data View**

This section explains the persistent data storage, including entities and schema.

**5.1 Domain Model**

**Entities**:

* **User**
  + userID, username, email, passwordHash, isAdmin
* **Item**
  + itemID, title, description, images, sellerID, startPrice, endDate
* **Bid**
  + bidID, itemID, bidderID, amount, timestamp

**Relationships**:

* One **User** can list many **Items**.
* One **Item** can have many **Bids**.
* Each **Bid** belongs to one **User** and one **Item**.

**5.2 Data Model (Persistent Data View)**

**5.2.1 Data Dictionary**

| **Entity** | **Attribute** | **Type** | **Description** |
| --- | --- | --- | --- |
| User | userID | String | Unique identifier |
|  | email | String | User’s email |
|  | passwordHash | String | Encrypted password |
| Item | itemID | String | Unique item identifier |
|  | title | String | Item name |
|  | endDate | DateTime | Auction expiration date |
| Bid | bidID | String | Unique bid ID |
|  | amount | Decimal | Amount placed |
|  | timestamp | DateTime | When bid was placed |

**6 Exception Handling**

The system handles exceptions at multiple layers:

| **Exception** | **Thrown When** | **Handling Strategy** | **Logging & Follow-up** |
| --- | --- | --- | --- |
| Authentication Error | Invalid login credentials | Returns 401 Unauthorized | Logged in auth logs |
| AuthorizationError | User accesses restricted resource | Returns 403 Forbidden | Logged with userID |
| ValidationError | Missing or invalid input data | Returns 400 Bad Request | Logged with input context |
| DatabaseError | MongoDB insert/update fails | Returns 500 Internal Server Error | Logged in DB error logs |
| NotFoundError | Item or User not found | Returns 404 Not Found | Logged for audit trail |

**7 Configurable Parameters**

This table describes the simple configurable parameters (name/value pairs) used in the application. These parameters are adjustable to tailor system behavior without modifying code. Some parameters are dynamic and can be updated at runtime, while others require application restart.

| **Configuration Parameter Name** | **Definition and Usage** | **Dynamic?** |
| --- | --- | --- |
| MAX\_BID\_RETRY | Maximum number of retry attempts for placing a bid in case of temporary failure. | Yes |
| SESSION\_TIMEOUT | Time in minutes before an inactive user session expires. | No |
| EMAIL\_NOTIFICATIONS | Toggle to enable/disable email notifications to users. | Yes |
| MAX\_IMAGE\_SIZE\_MB | Maximum image upload size per item in megabytes. | No |
| CURRENCY | Default currency used for listing items and placing bids. | Yes |
| AUCTION\_DURATION\_LIMIT | Maximum auction duration allowed for item listings in days. | No |
| DATABASE\_URI | MongoDB connection URI for database operations. | No |
| ADMIN\_EMAIL | Email address to which system errors and alerts are sent. | Yes |

**8 Quality of Service**

This section outlines how the system maintains high availability, enforces secure access, manages load efficiently, and ensures monitoring and control in a production environment.

**8.1 Availability**

The system is designed to maintain high availability to meet the business requirement of **99.5% uptime**, especially during peak auction periods.

Key design decisions supporting availability include:

**-Redundant Deployment**: Hosted on cloud infrastructure with load balancing to avoid single points of failure.

**-Auto-scaling**: Application servers are configured for horizontal scaling during traffic spikes (e.g., near auction closings).

**-Graceful Failures**: Non-critical components (e.g., email dispatchers) fail silently without affecting core bidding operations.

**-Scheduled Maintenance**: System supports configurable maintenance windows for database backup, cache clearance, and log rotation.

**-Failover Mechanism**: In the event of a server crash, the session data and in-progress bids are stored in a shared cache (like Redis) to allow recovery.

**8.2 Security and Authorization**

The system follows **Role-Based Access Control (RBAC)** to enforce security requirements:

**-User Roles**:

* + *Student*: Can register, list items, place bids, and track auctions.
  + *Admin*: Can manage users, monitor suspicious activity, and control system parameters.

Security features include:

**-Authentication**: Email-based login with secure password hashing (bcrypt or similar).

**-Authorization**: Middleware-based access control ensuring restricted endpoints are accessed only by authorized roles.

**-Data Security**:

* + Encrypted communications using HTTPS.
  + Input validation and sanitization to prevent SQL/NoSQL injections and XSS.

**-Audit Logs**: All login attempts, parameter changes, and admin actions are logged.

**-User Management Interface**: Admins can create/modify user roles, deactivate accounts, and reset passwords via a secure dashboard.

**8.3 Load and Performance Implications**

Design considerations made to meet performance requirements (target: <2s response time per transaction under 1000 concurrent users):

**-Database Optimization**: Use of indexed fields (e.g., item ID, bid timestamp) to support fast lookups.

**-Caching Layer**: Frequently accessed data like item details and current bids are cached using Redis or Memcached.

**-Asynchronous Processing**: Email notifications and image compression handled via background job queues (e.g., Celery, RabbitMQ).

**-Pagination & Lazy Loading**: Applied to avoid loading all auctions or bids at once.

**-Expected Load**:

* + Peak concurrent users: ~1000
  + Auction listing growth: ~5000 items/month
  + Message volume (bids/notifications): ~100,000/day

**8.4 Monitoring and Control**

The system includes built-in and external monitoring tools:

**-Process Monitoring**:

* + Background services like job queues and schedulers are monitored for uptime.
  + Status endpoints expose application health (/health, /status).

**-Metrics Published**:

* + Average response time per endpoint
  + Number of bids per hour
  + Memory/CPU usage
  + Auction closure success rate

**-Alerting & Logging**:

* + Integrated with tools like Prometheus + Grafana or ELK Stack.
  + Alerts on system errors, unusual activity, or downtime are sent via email/SMS to admins.

**-Admin Control Panel**: Provides visibility into auctions, users, bid logs, and system parameters with the ability to trigger maintenance mode.