Software Design Specifications

for

Online Auction for students to sell and buy their personal items

Prepared by: BBS
SE22UCSE004 Abhilash
SE22UCSE010 Adithi
SE22UCSE065 Surya
SE22UCSE256 Sreeja
SE22UCSE258 Sreeni
SE22UCSE305 Vijay
SE22UCSE274 Koushik

Document Information

Title: Online Auction for students to sell and buy their personal items	
Project Manager:	**Document Version No: 3**
	Document Version Date: 9th April 2025.
Prepared By: BBS	**Preparation Date: 9th May 2025.**

Table of Contents

1 Introduction

The Bid Buy Save (BBS) website is an online auction platform where users can register, list products for sale, place bids, and buy items. This document explains how the system is designed, covering the structure, components, and how everything works together. It gives a complete overview of the system's purpose, what it will include, and the important terms used in the design.

1.1 Purpose

The objective of this Software Design Specification (SDS) document is to define the design framework for the Bid Buy Save (BBS) web application. It details the structural composition of the system and the interactions between its components to facilitate core functionalities such as product listings, bidding processes, and user account management.

This document is intended for use by developers, testers, UI/UX designers, and project stakeholders. Developers will reference it to comprehend the system architecture and implement individual modules accordingly. Testers will use it to verify the logical coherence and completeness of the design. Designers can consult the document to ensure interface consistency with the system's functional and structural requirements. Stakeholders may use it to gain insight into the expected behaviour and composition of the final product.

1.2 Scope

This Software Design Specification applies to the entire Bid Buy Save (BBS) web application. It includes the design of all core modules such as user registration and login, product listing, bidding system and admin panel.

The document influences how the system will be implemented by providing technical guidelines for developers and ensuring consistency in functionality and design across different modules. It serves as a reference for understanding how individual components fit together to achieve a fully functional, secure, and user-friendly online auction platform.

1.3 Definitions, Acronyms, and Abbreviations

Acronym		Description
BBS	-	Bid Buy Save – the web-based auction platform being developed
UI	-	User Interface – the front-end interface through which users interact with the system
UX	-	User Experience – overall experience of a user while using the system
DB	-	Database – stores all the data related to users, products, bids, and transactions
API	-	Application Programming Interface – allows different software components to communicate
Admin	-	Administrator – user with highest-level access to manage platform activities
Bidder	-	A user who places bids on products listed for auction
Seller	-	A user who lists a product for sale or auction
Buyer	-	A user who purchases a product either through direct buy or by winning a bid
HTTPS	-	Hypertext Transfer Protocol Secure – ensures secure communication over the web

1.4 References

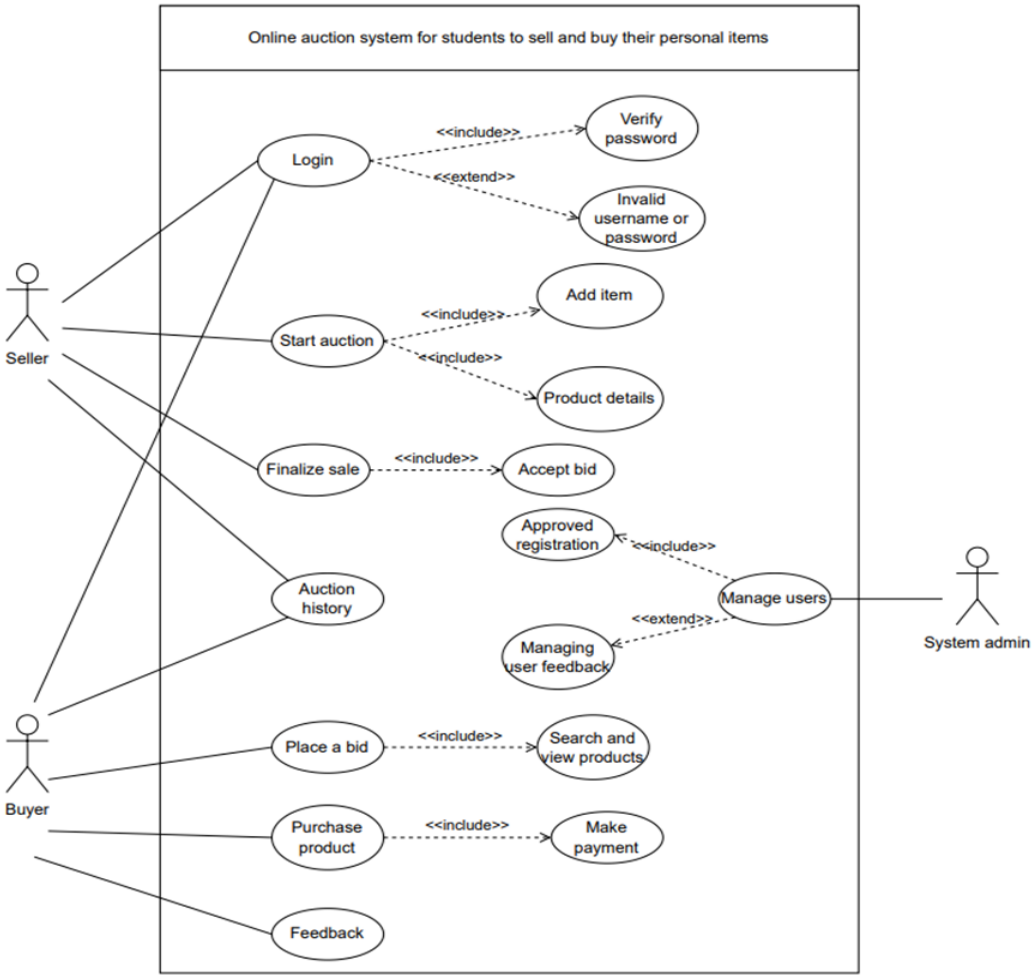
This part lists all the documents that were referred to while creating this Software Design Specification. Wherever possible, links to the original sources are provided to make them easy to find and access.

1) https://www.tandfonline.com/doi/abs/10.1080/0144929021000050256
2) https://www.sciencedirect.com/science/article/pii/S1877050917329800
3) https://link.springer.com/chapter/10.1007/978-3-7091-7504-0_16

2. Use Case View

This section identifies and describes the key use cases that drive the software design of the BBS (Bid, Buy, and Save) platform. These use cases represent the core functionalities provided to users primarily students and administrators to buy, sell, bid, rent, and manage products within the system.

The use cases were derived from the Software Requirements Specification (SRS) and help define how the system architecture should be developed to support user interactions



2.1 List of Core Use Cases

1. User Registration & Login
2. Item Listing & Management
3. Bidding on Items
4. Admin Moderation & User Management

2.2 Use Case Descriptions

2.2.1 Use Case: User Registration & Login

Description:

Allows university students to register and securely log in using verified email credentials.

Actors:

- Student
- System

Preconditions:

- The user must have a valid university email.
- The platform must be accessible.

Postconditions:

- Account is successfully created and activated.
- The student can log in and access the dashboard.

Basic Flow:

1. Student navigates to the registration page.
2. Enters name, email, and password.
3. System verifies if the email is valid.
4. If valid, account is created, and confirmation mail is sent.
5. Student logs in using credentials.

Alternative Flow:

- If the email is invalid, the system displays: "Invalid email. Please use your university email."

Exceptions:

- If the mail server is down, system shows: "Verification email could not be sent. Please try again later."

2.2.2 Use Case: Item Listing & Management

Description:

Enables students to list, edit, or delete items for sale.

Actors:

- Seller (Student)
- System
- Admin

Preconditions:

- User must be logged in.
- The description and images of the product should be clear.

Postconditions:

- Item is listed or updated in the platform after admin approval.

Basic Flow:

1. Seller accesses "Add Item" section.
2. Inputs title, description, images, price, and category.
3. Submits for admin approval.
4. Admin reviews and publishes item.

Alternative Flow:

- If images are missing or file types are invalid, system prompts: "Please upload valid images."

Exceptions:

- If database is unavailable, user sees: "Listing failed. Try again later."

2.2.3 Use Case: Bidding on Items

Description:

Allows users to place bids on active items.

Actors:

- Buyer (Student)
- System

Preconditions:

- The item is listed for bidding and not expired.
- User is logged in.

Postconditions:

- Bid is recorded and updated as the current highest bid if valid.

Basic Flow:

1. Buyer views product page.
2. Enters bid amount.
3. System checks if it’s higher than current bid.
4. If valid, bid is recorded and updated.

Alternative Flow:

- If bid is lower than the current highest bid, system prompts: "Bid too low. Please enter a higher amount."

Exceptions:

- In case of database error, system shows: "Bid failed. Try again later."

2.2.4 Use Case: Admin Moderation & User Management

Description:

Enables admins to approve listings and manage user accounts.

Actors:

- Admin

Preconditions:

- Admin is authenticated.

Postconditions:

- Listings are either approved or rejected.
- Suspicious accounts are removed.

Basic Flow:

1. Admin logs into dashboard.
2. Reviews newly listed items or user reports.
3. Approves or deletes items and manages flagged users.

Alternative Flow:

- If no items are pending, system displays: "No new submissions."

Exceptions:

- If admin session times out, system forces re-login.

3 Design Overview

This section outlines the overall design of the Bid Buy Save (BBS) website, detailing how the software architecture aligns with project requirements and module decomposition. The BBS system follows a modular, layered architecture ensuring scaability, maintainability, and separation of concerns.

3.1 Design Goals and Constraints

Goals:

- **Maintainability:** The design should enable developers to easily comprehend, update, and enhance the system, supporting long-term sustainability and adaptation.
- **Usability:** The interface should prioritize user-centered design principles, offering a seamless and accessible experience that minimizes the need for extensive training or support.
- **Scalability:** The system must be designed to accommodate increasing demands, including higher user traffic, larger data sets, and the integration of new features over time.
- **Security:** Robust security practices must be implemented to safeguard sensitive information, encompassing strong authentication protocols, role-based access control, and secure data handling.
- **Performance:** The application is expected to deliver low-latency responses and operate efficiently within defined resource constraints, ensuring smooth user experience under normal and peak conditions.
- **Modularity:** The architecture should consist of independent, well-structured components that facilitate code reuse and simplify ongoing maintenance.

Constraints:

- **Team Structure:** Divided into frontend, backend, and QA sub teams.
- **Legacy Code:** Integration with existing components in specific modules.
- **Technology Stack:** Java, HTML, CSS, SQL database.
- **Schedule Constraints:** Strict deadlines requiring phased, prioritized development.
- **Development Tools:** Visual Studio Code, draw.io, GitHub, MySQL workbench.

3.2 Design Assumptions

- The system assumes users will have stable and consistent internet access.
- Access is restricted to students with valid university email addresses.
- User roles and permissions are predefined and limited to known categories.
- Product listings will be moderated manually or through basic automated filters.
- The server is expected to handle moderate traffic, with scalability planned for future updates.
- Third-party APIs and external services are assumed to be consistently available and function as documented.
- Payment and delivery processes will be conducted offline and are not handled within the system.

3.3 Significant Design Packages

The software design for the BBS platform is organized into layered packages, each handling a specific responsibility. The presentation layer manages user interfaces such as registration forms, login screens, product listings, and bidding dashboards, providing students with a seamless and responsive user experience. The business logic layer handles core processing like user authentication, bidding validation, auction timing, and admin moderation workflows, ensuring that all operations follow the platform's rules and logic. The data access layer interacts directly with the SQL database, performing CRUD operations on users, products, bids, and reviews while maintaining data consistency and integrity.

Additional supporting packages include an admin module that enables administrative control over listings, reports, and user accounts. Dependencies among these modules are structured hierarchically to maintain modularity and ensure a clean separation of concerns. This design promotes scalability, simplifies debugging, and allows parallel development by the frontend, backend, and QA sub-teams.

3.4 Dependent External Interfaces

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	***Module Using the Interface***	***Functionality/Description***
Authentication API	Authentication Module	Used to validate user credentials during login and maintain secure access control.
Email Service API	Notification Module	Sends bid updates, account alerts, and purchase confirmations.
Location API	Product Module	(Optional) To show location-based products or listings.

3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

Interface Name	***Module Implementing the Interface***	***Functionality/Description***
User API	User Management Module	Handles user login, registration, and provides user profile information to external systems upon request.
Product Listing API	Product Module	Allows adding, viewing, and updating products in the BBS.
Bidding API	Bidding Module	Accepts and validates bids, updates product status, and manages bid history.
Admin API	Admin Module	Provides tools to view reports, manage users/listings, and control platform rules.

4 Logical View

The logical view describes the functionality, structure, and behaviour of the BBS system from a user perspective. This section outlines the system's major components, their interactions, and key design patterns implemented.

Layered Architecture Overview

1. **Presentation Layer**
   The topmost layer of the system; it provides the interface through which users interact with the application. It supports activities like login, registration, browsing, and inventory management, capturing user inputs and displaying the corresponding responses from the system.
2. **Database Layer**
   This layer acts as the permanent data repository for the system. It stores structured records for users, products, orders, and categories. It ensures reliable data persistence and provides the required information to the other layers when needed.
3. **Data Access Layer**
   Serving as an intermediary between the business logic and the database, this layer handles all data operations such as retrieval, insertion, updating, and deletion. It ensures that data-related tasks are separated from the application's core logic.
4. **Business Logic Layer**
   This layer contains the main functionality of the application. It processes user inputs, manages key operations like user authentication, order handling, and product updates, and applies the necessary business rules. It coordinates the flow of data between the user interface and the data layer.

4.1 Design Model

The design model presents a class-based view of the **Online Auction System for Students**. It identifies the system's major functional modules, and the primary classes involved in each. The model supports object-oriented design principles such as encapsulation, inheritance, and modularity, ensuring that the system is well-structured and maintainable.

Module Decomposition

The system is logically divided into the following modules:

- Student/User Management
- Auction & Bidding Management
- Product Listing
- Authentication Control
- Admin Operations

Each module contains classes that work together to support the respective functionality.

Key Classes and Responsibilities

1. **Student**
    a. **Attributes:** studentID, name, email, password, contactInfo
    b. **Responsibilities:** Represents a registered student user; can view, list, bid on products, and manage profile information.
2. **Product**
    a. **Attributes:** productID, title, description, basePrice, categoryID, sellerID, status
    b. **Responsibilities:** Represents an item listed for auction; supports posting, updating, and viewing details.
3. **Bid**
    a. **Attributes:** bidID, productID, bidderID, bidAmount, bidTime
    b. **Responsibilities:** Represents a student's bid on a product; maintains bid history and highest bid logic.
4. **Admin**
    a. **Attributes:** adminID, name, email, password
    b. **Responsibilities:** Oversees the platform; manages user accounts, categories, and resolves disputes or issues.
5. **Auction Service**
    a. **Responsibilities:** Coordinates bidding processes, closing auctions, selecting winning bids, and handling auction timings.
6. **Authentication Service**
    a. **Responsibilities:** Manages login, registration, and access control for students and administrators.

Relationships Between Classes

- A Student can place multiple Bids.
- A Product receives multiple Bids from different Students.
- A Product is listed under one Category.
- An Admin manages Products and Categories.
- Each Bid is associated with a Product and a Student.
- The Auction Service oversees the bidding activity for all Products.

Class Diagram Representation

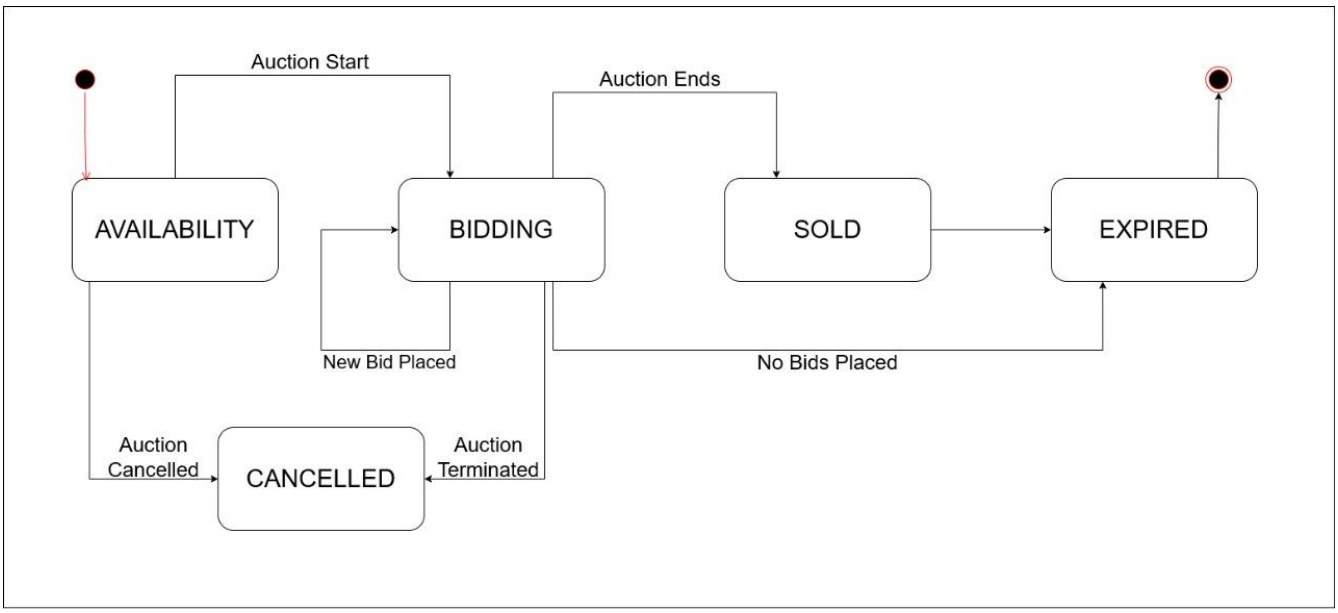
A class diagram for this system can visually represent:

- Associations (e.g., Student ↔ Bid, Product ↔ Category)
- Aggregations (e.g., Product → Bids)
- Attributes and Methods within each class
- Controller classes such as Auction Service or Authentication Service
- User roles, where Admin and Student may inherit from a base User class if applicable

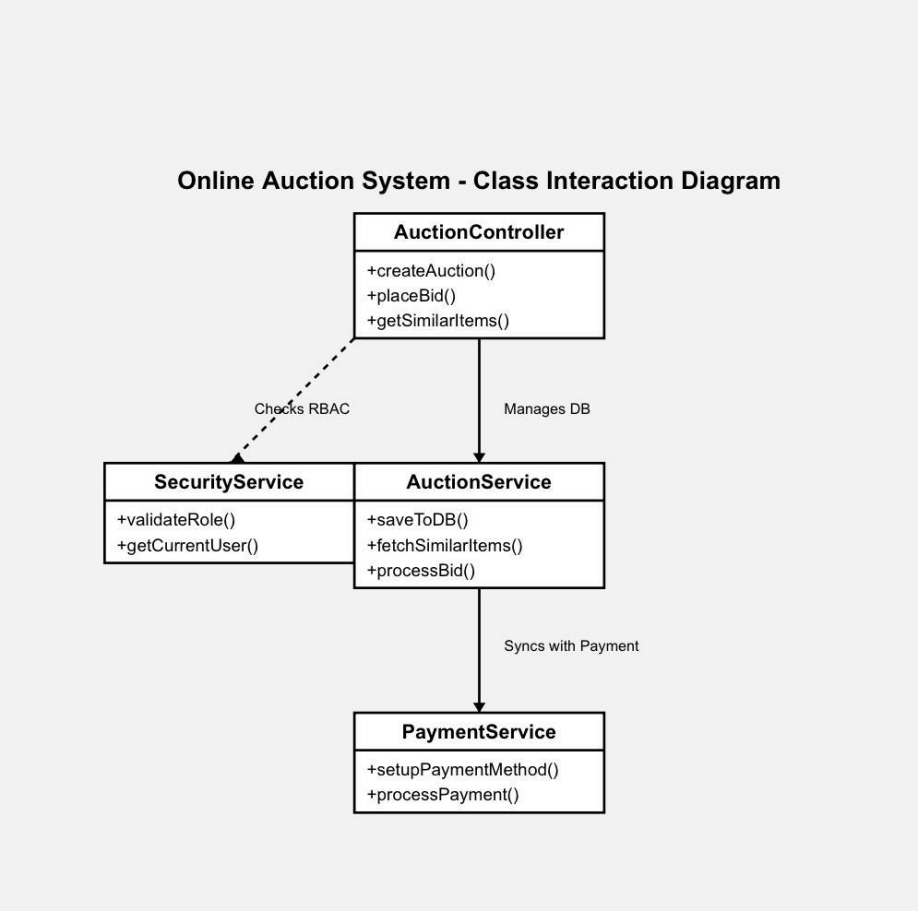
4.2 Use Case Realization

Use Case 1: User registration and login

State diagram:



Class interactions:

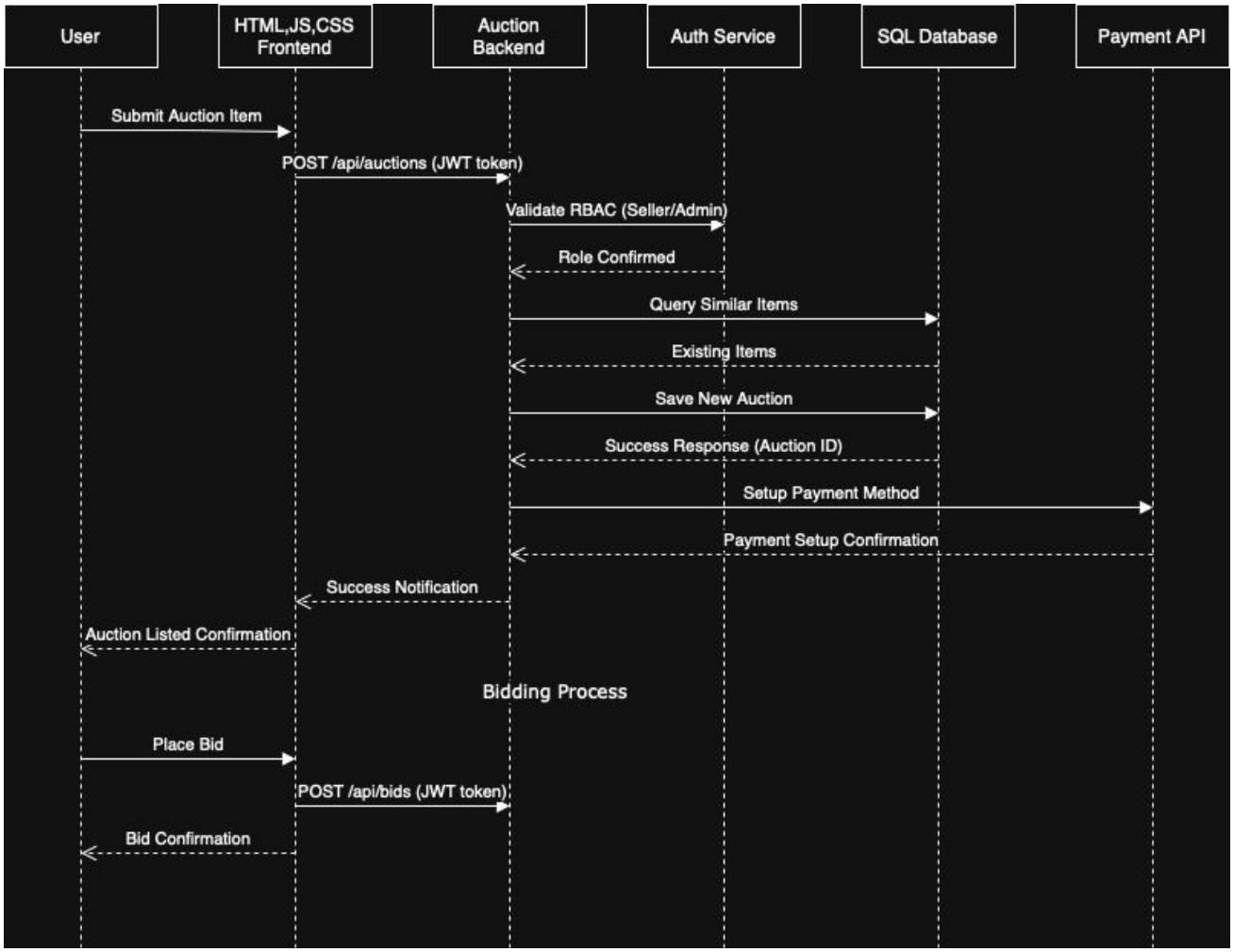


Key Design Elements

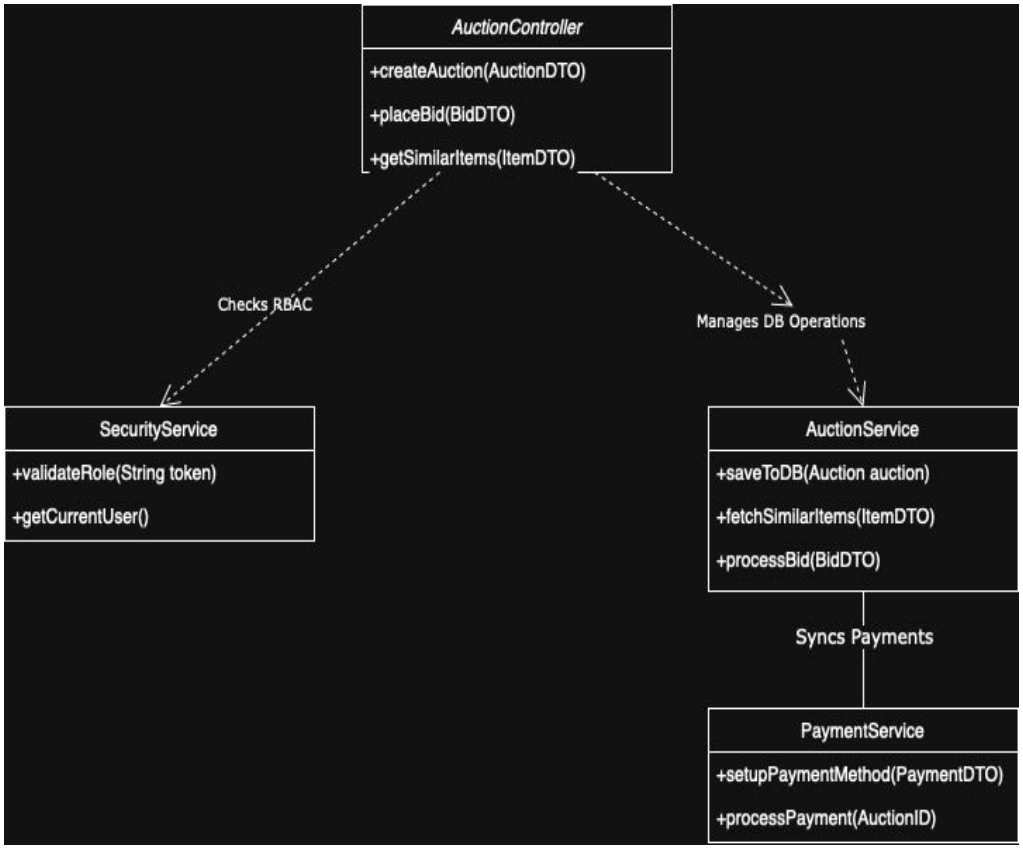
- **RBAC Enforcement:** User roles (Seller/Admin) are validated via Security Service
- **Modular Services:** Core logic is split into Auction Service, Payment Service, and Security Service.
- **Auction Flow:** Auction creation involves role check, saving to DB, querying similar items, and setting up payment.
- **Bidding Flow:** Bids are placed via secure API calls and processed by the backend.

Use Case 2: Item listing and Management

High level sequence diagram:



Low level sequence diagram:

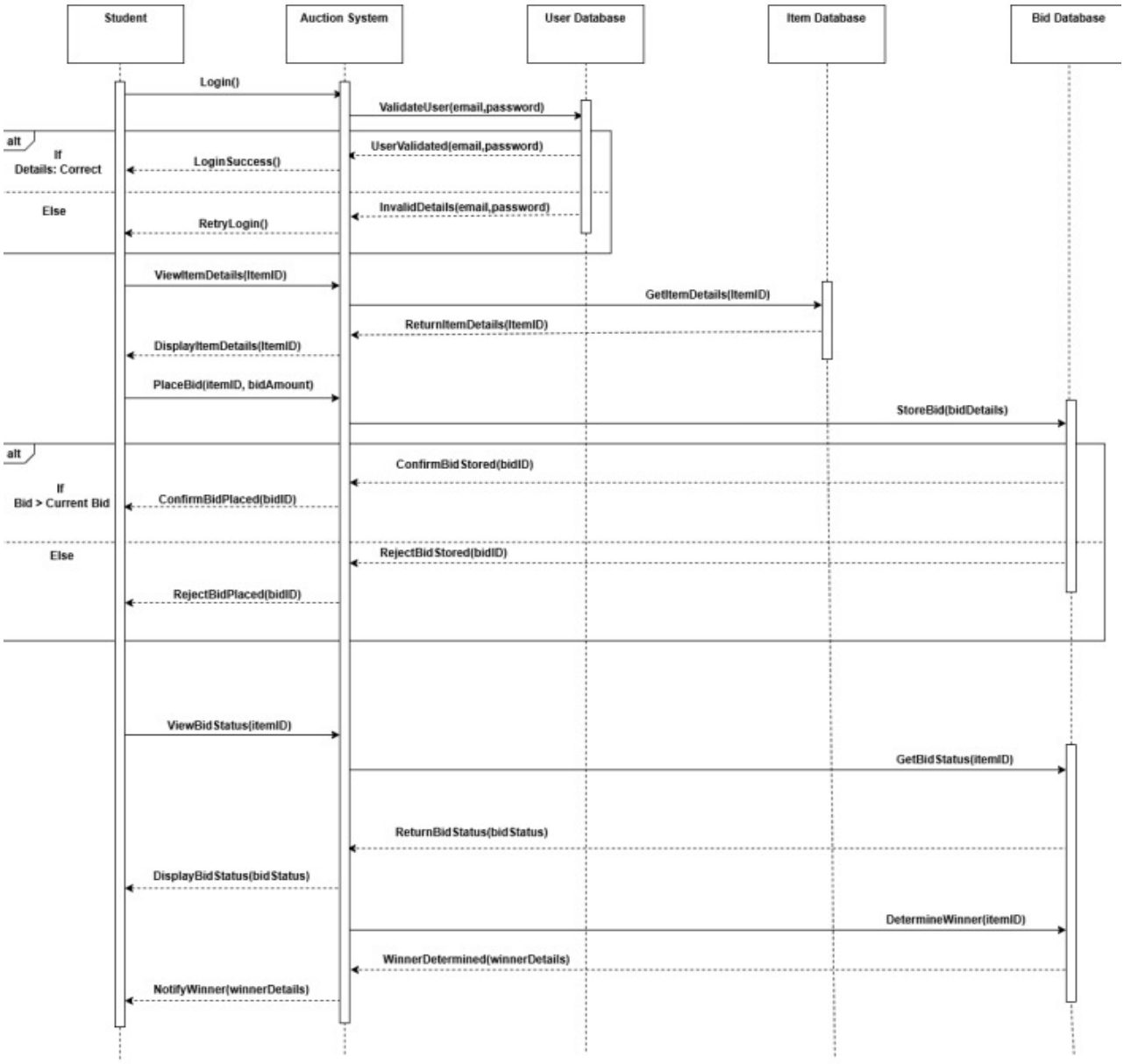


Key Design Elements

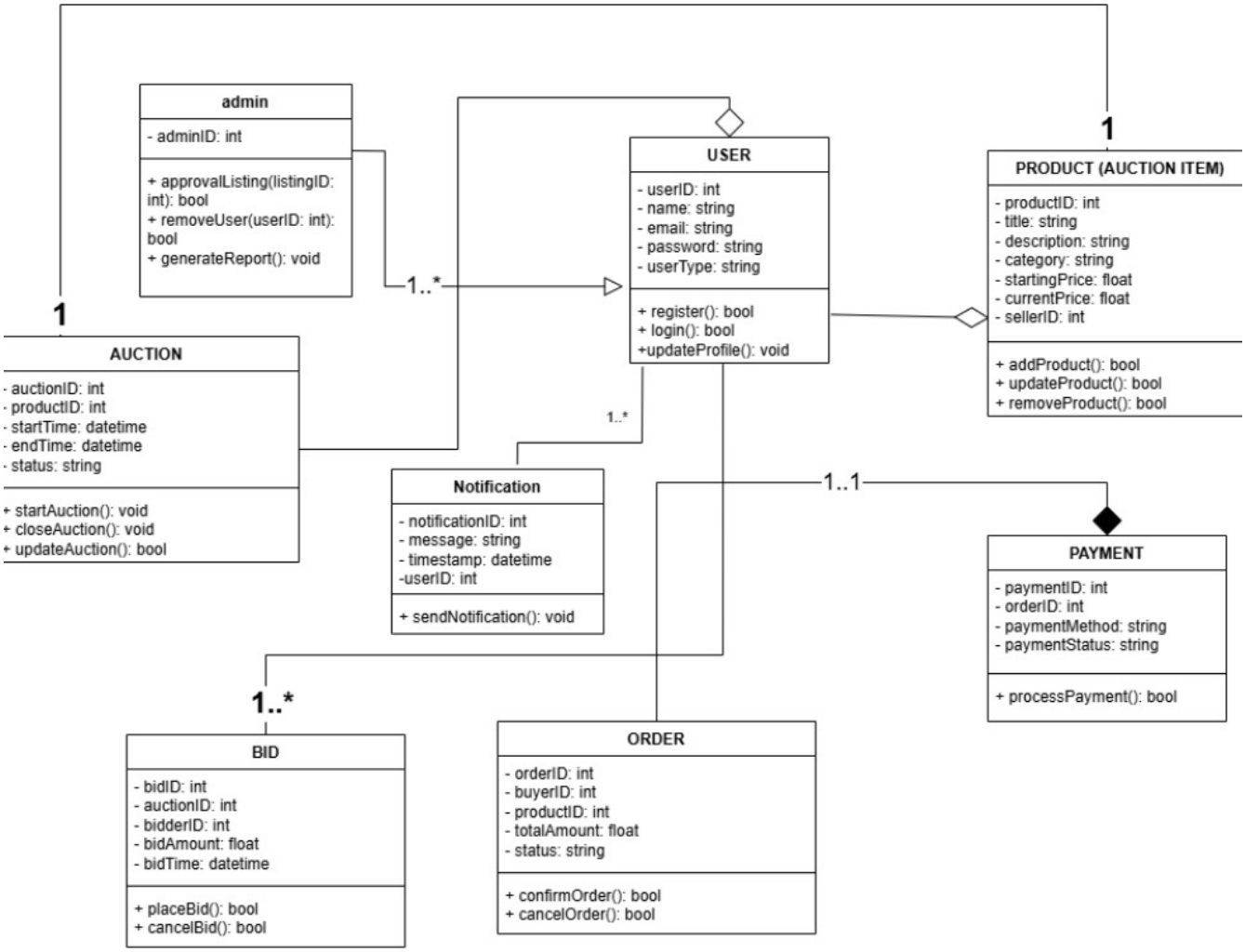
- **RBAC Enforcement:** User roles (Seller/Admin) are validated via `Security Service`
- **Modular Services:** Core logic is split into Auction Service, Payment Service, and Security Service.
- **Auction Flow:** Auction creation involves role check, saving to DB, querying similar items, and setting up payment.
- **Bidding Flow:** Bids are placed via secure API calls and processed by the backend.

Use Case 3: Bidding on items

Sequence diagram:



Class diagram:

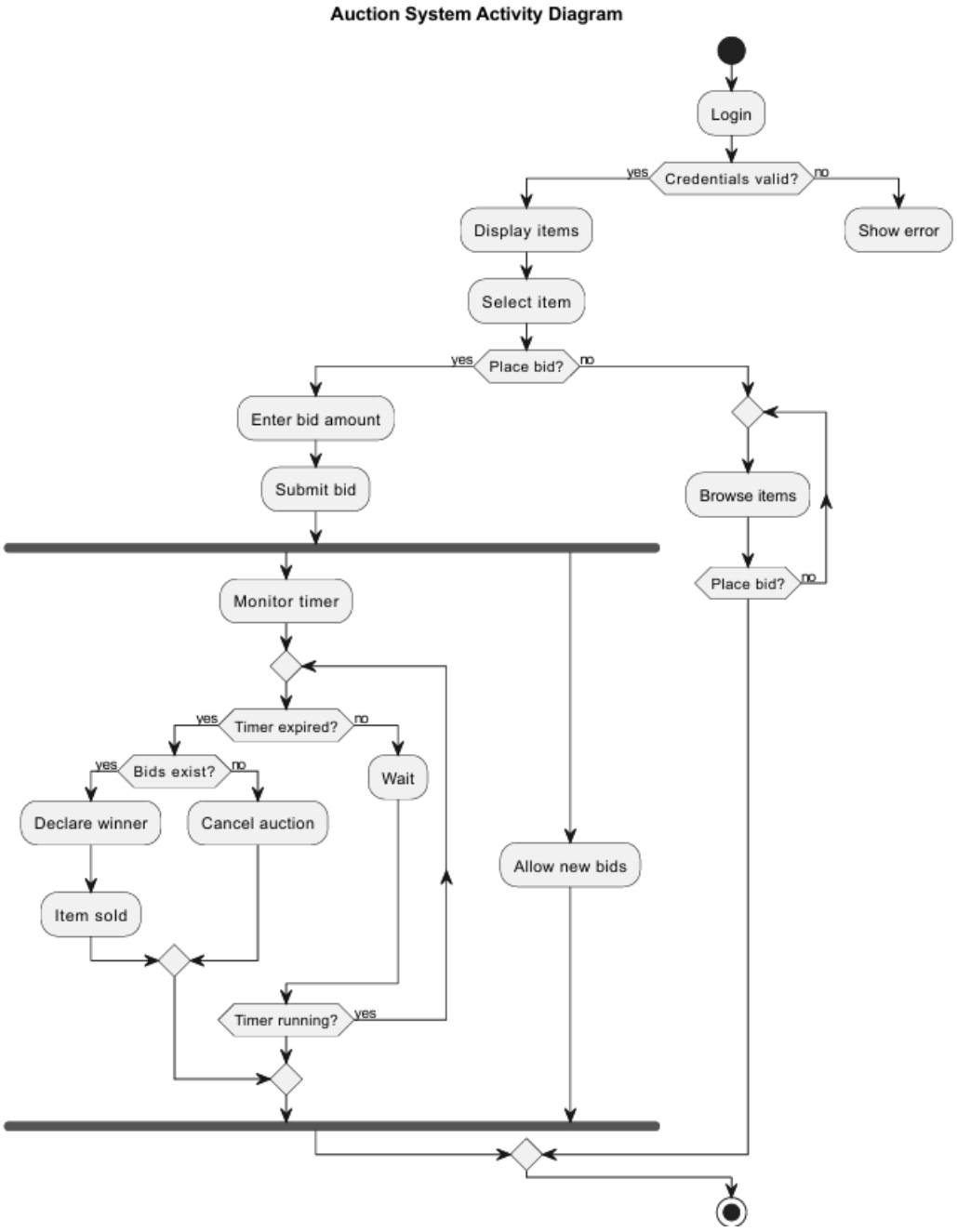


Key Design Elements

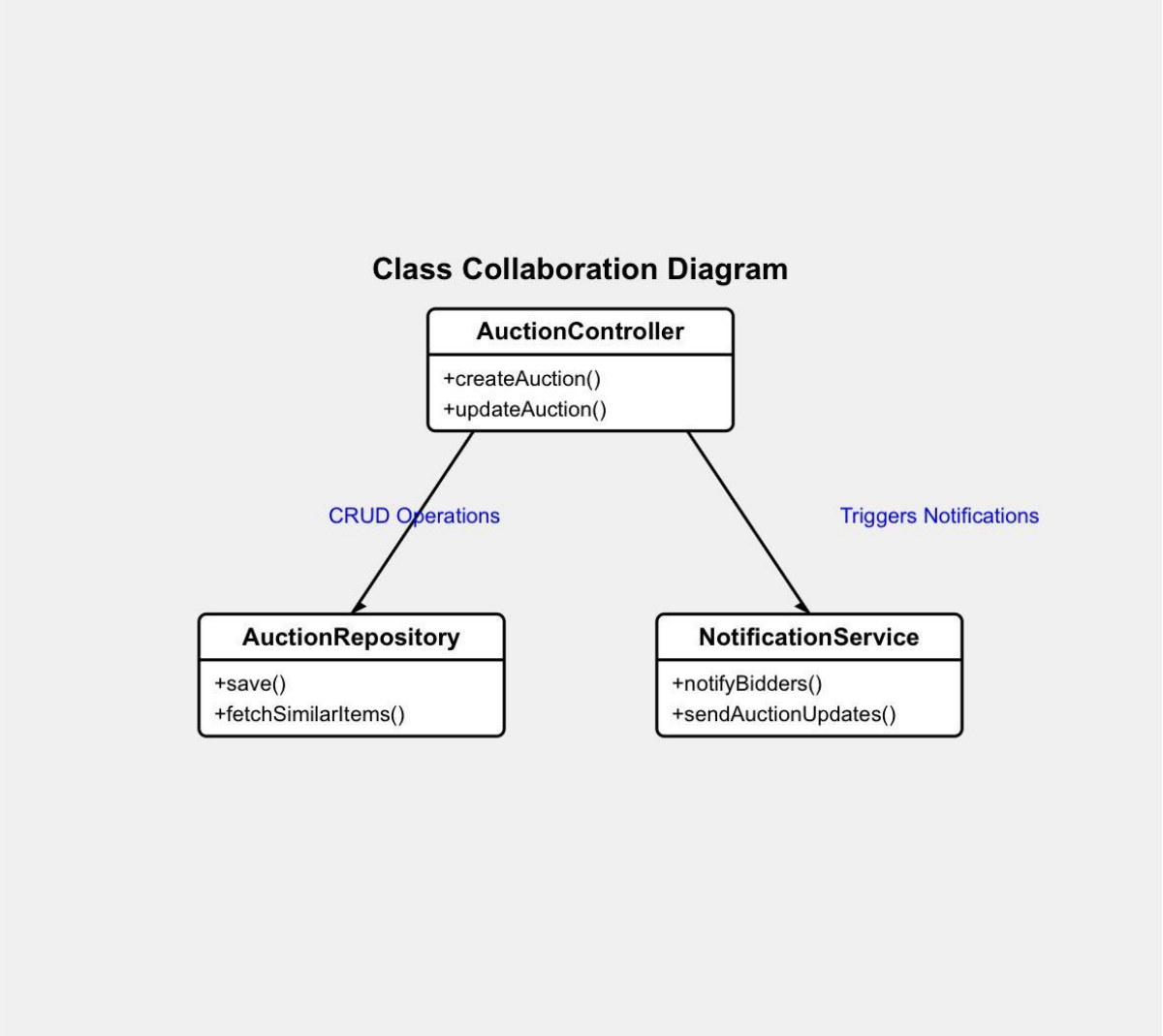
- **Concurrent Bidding:** Supports multiple users bidding at the same time using timestamp checks.
- **Session Validation:** Only authenticated sessions can place bids or create auctions.
- **Timed Auctions:** Automatically closes auctions and declares winners when the timer ends.
- **Service Coordination:** Controllers interact with services for bid processing and auction updates.

Use Case 4: Admin Moderation and user management

Activity diagram:



Class collaborations:



Key design elements

- **Auction Handling**: Auction Controller manages creation, bid placement, and coordination with services.
- **Database Operations**: Auction Service and Auction Repository handle data saving and retrieval (e.g., similar item queries)
- **Timer Monitoring**: The system actively checks auction timer status; upon expiry, it evaluates bids to declare a winner or cancel the auction.
- **Error Handling**: Invalid actions (like bidding without login or during inactive auctions) prompt error messages.

5 Data View

The data view defines how information is structured, stored, and maintained within the BBS platform. The system relies heavily on persistent data for users, products, bids, and transactions, thus necessitating a well-defined data schema

5.1 Domain Model

The Domain Model represents core entities in the BBS platform and how they interact with one another.

Entity Relationship Overview:

- **User**: Can act as a Buyer, Seller, or Admin.
- **Product**: Listed by a User (Seller) can receive multiple Bids.
- **Bid**: Placed by a User (Buyer) on a Product.
- **Transaction**: Generated when a Bid is accepted.
- **Notification**: Sent to Users for various events (e.g., new bid, bid accepted).

Entity Relationships:

- One **User** can list multiple **Products**.
- One **Product** can have many **Bids**.
- One **Bid** is linked to one **User** (Buyer).
- One **Transaction** involves one **Product** and one winning **Bid**.
- **Notifications** are sent to **Users**.

5.2 Data Model (persistent data view)

This section outlines the tables and how persistent data is stored using relational mapping. Below is a simplified schema.

5.2.1 Data Dictionary

Entity	Field	Data Type	Description
User	userId	String	Unique identifier for each user
	username	String	User's display name
	email	String	Verified student email address
	password	Encrypted String	User's login credentials
Item	itemId	String	Unique identifier for each item
	itemName	String	Name of the product or service
	itemDescription	Text	Description of the item
	startingPrice	Decimal	Initial bid price
	currentBid	Decimal	Latest bid amount
	status	String	Available, Sold
	listedBy	String (FK)	Reference to userId of the seller
Bid	bidId	String	Unique identifier for each bid
	itemId	String (FK)	Reference to the bid item
	userId	String (FK)	Reference to the bidder
	bidAmount	Decimal	Amount placed by the user
	bidStatus	String	Active, Outbid, Won
	timestamp	DateTime	When the bid was placed
Review	reviewId	String	Unique ID for the review
	itemId	String (FK)	Reviewed item
	userId	String (FK)	Reviewer’s ID
	rating	Integer (1–5)	Numeric star rating
	comment	Text	Optional review comment

6 Exception Handling

This section outlines the exceptions that may occur within the BBS application, the scenarios in which they are triggered, how they are handled, and the corresponding follow-up actions. A robust exception handling strategy ensures system stability, user-friendly error messages, and detailed logging for debugging and auditing purposes.

1. Authentication and Authorization Exceptions

Exception Name	***Description***	***Handling***	***Follow-up Action***
InvalidCredentialsException	Thrown when login credentials are incorrect.	Return error message: "Invalid username or password."	Prompt user to retry login.
AccessDeniedException	Thrown when a user tries to access a restricted resource.	Return error message: "You are not authorized."	Redirect user to homepage/login.

2. Data Validation Exceptions

Exception Name	***Description***	***Handling***	***Follow-up Action***
MissingFieldException	Required input fields are missing.	Return message: "All fields are required."	Highlight missing fields in UI.
InvalidDataFormatException	Incorrect format for input data (e.g., email, bid amount).	Return: "Invalid format provided."	Highlight incorrect field and guide.

3. Database Exceptions

Exception Name	***Description***	***Handling***	***Follow-up Action***
DataNotFoundException	Queried data does not exist in the database.	Log error; show: "Requested data not found."	Prompt user to recheck input.
DatabaseConnectionException	Unable to connect to the database.	Log error and show: "Internal server error. Try later."	Alert admin, retry after delay.
DuplicateEntryException	Attempt to insert data that violates uniqueness.	Show: "This entry already exists."	Suggest alternative input.

4. Bidding Exceptions

Exception Name	***Description***	***Handling***	***Follow-up Action***
LowBidAmountException	Bid amount is less than current highest bid.	Return message: "Bid must be higher than the current bid."	Prompt user to increase bid.

BidClosedException	Attempt to bid on a closed/expired product.	Show: "Bidding for this product has ended."	Disable bid button on UI.

5. File and Network Exceptions

Exception Name	***Description***	***Handling***	***Follow-up Action***
FileUploadException	File format or size issue during product upload.	Show: "Invalid file type or size exceeds limit."	Instruct user on acceptable formats.
NetworkTimeoutException	Timeout in API or service call.	Show: "Request timed out. Please try again."	Retry request after short delay.

6.2 Exception Logging

All exceptions are:

- Logged using a centralized logging mechanism (e.g., Log4j or Winston).
- Stored with timestamps, user IDs (if available), stack traces, and exception messages.
- Logs are reviewed periodically by the DevOps/admin team.

6.3 Follow-Up Action

After catching exceptions:

- Non-critical errors return friendly messages and UI prompts.
- Critical exceptions (e.g., database failure) trigger email/SMS alerts to administrators.
- Unhandled exceptions are routed to a fallback error page, ensuring no sensitive data is exposed.

7 Configurable Parameters

This section describes the application-level parameters that are configurable in the BBS system. These parameters allow flexibility for system tuning, deployment-specific customization, and feature toggling. Some parameters can be updated at runtime without restarting the application (marked as *Dynamic = Yes*), while others require a restart.

7.1 Simple Configuration Parameters

Configuration Parameter Name	*Definition and Usage*	*Dynamic?*
MAX_BID_RETRY_ATTEMPTS	Number of times a user can retry submitting a failed bid before being blocked.	No
SESSION_TIMEOUT_MINUTES	Duration (in minutes) before an inactive user session is terminated.	Yes
DEFAULT_IMAGE_UPLOAD_PATH	File path where uploaded product images are stored.	No
ALLOWED_IMAGE_FORMATS	Comma-separated list of image types allowed for product uploads (e.g., .jpg, .png).	Yes
MAX_IMAGE_UPLOAD_SIZE_MB	Maximum size (in MB) for each product image file upload.	Yes
BID_CLOSING_BUFFER_MINUTES	Buffer time before the actual bid end-time during which new bids are no longer allowed.	No
SUPPORT_EMAIL_ADDRESS	Email address used for customer support queries.	Yes
EMAIL_VERIFICATION_REQUIRED	Boolean flag to enforce email verification during registration.	Yes
ENABLE_2FA	Toggle to enable or disable two-factor authentication for users.	Yes
DATABASE_CONNECTION_URL	JDBC URL for connecting to the application database.	No
LOG_LEVEL	Controls the verbosity of system logs (e.g., INFO, DEBUG, WARN, ERROR).	Yes
CACHE_EXPIRY_MINUTES	Duration (in minutes) for which in-memory cache entries remain valid.	Yes

8 Quality of Service

This section outlines how the BBS platform is designed to ensure high availability, secure user access, responsive performance under load, and system monitoring for reliable operation in production environments.

8.1 Availability

- **Business Requirement Reference**: The BBS platform is expected to be available 99.5% of the time, excluding planned maintenance.
- **Design Support for Availability**:

    - Load-balanced architecture with failover support.
    - Stateless backend services to allow easy horizontal scaling and recovery.
    - Graceful shutdown and restart mechanisms.
    - Use of health check endpoints for each microservice to allow Kubernetes/ELB to manage availability.
- **Potential Downtime Factors**:
    - Mass product import/export operations during off-peak hours.
    - Database schema changes during maintenance windows.
    - Housekeeping jobs such as archiving expired listings.
    - Bi-weekly maintenance window scheduled for updates and patches.

8.2 Security and Authorization

- **Authentication & Authorization**:
    - Supports secure login with hashed passwords and optional two-factor authentication (2FA).
    - OAuth 2.0 support for third-party integrations.
    - Role-based access control (RBAC) for Admin, Seller, and Buyer functionalities.
    - JWT-based authentication for secure API access.
    - Encrypted passwords (128-bit) for data protection.
- **Data Security Measures**:
    - HTTPS enforced throughout the application.
    - Input validation and sanitization to prevent XSS and SQL injection.
    - Access control checks at both API and UI levels.
    - Only verified students with valid university email domains allowed.
- **User Access Management**:
    - Admin panel to assign or revoke user roles.
    - Account lockout after multiple failed login attempts.
    - Email verification and password reset workflows.
    - Audit logs of user activities for compliance.
    - Admins validate new listings for policy compliance.

8.3 Load and Performance Implications

- **Expected Usage Load**:
    - Peak load: ~500 concurrent users during bidding hours.
    - Daily product listings: ~5,000 new entries.
    - Average bid submissions per hour: ~20,000.
    - Supports 100+ concurrent users in initial implementation.
- **Design Adaptations**:
    - Asynchronous queuing for bid processing using message brokers (e.g., RabbitMQ/Kafka).
    - Indexed DB tables for fast search and bids resolution.
    - CDN for static content (images, stylesheets).
    - Cache for frequently accessed data (e.g., product details, current bids).
    - Database sharding planned for long-term scalability.
    - Backend optimized specifically for auction/bid transaction scalability.

8.4 Monitoring and Control

- **Monitoring Tools**:
    - Integration with Prometheus and Grafana for system metrics.
    - Custom health endpoints (/health, /status) for uptime checks.
    - Logging via ELK (Elasticsearch, Logstash, Kibana) or Loki for centralized log monitoring.
    - Admin dashboard for log tracking and user analytics.
- **Controllable Processes**:
    - Scheduled background jobs for data cleanup, email notifications.
    - Resilient retry logic in case of third-party service failures.
    - Alerts for system failures or suspicious bidding patterns.
- **Published Metrics**:

- API latency, bid submission rates, user login frequency.
- Error rates, CPU/RAM usage, queue size, and DB response time.