UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3112: Software Engineering Lab

BidCraft: Online Bidding Platform

Software Design Document (SDD)

Submitted By:

Name : Tahsin Ahmed

Roll No : 03

Name : Md. Atikur Rahman Hridoy

Roll No : 19

Name : Md. Sakib Ur Rahman

Roll No : 37

Submitted On :

April 16, 2024

Submitted To :

Dr. Saifuddin Md. Tareeq

Redwan Ahmed Rizvee

Contents

1 Introduction

1.1 Purpose of the System

BidCraft: Online Bidding Platform is an innovative web-based platform designed to revolutionize the way individuals and businesses engage in online auctions. This platform provides a user-friendly interface, enabling seamless participation in auctions, bid placement, and efficient management of listings. BidCraft aims to create a dynamic and engaging online marketplace where users can discover unique items, competitive pricing, and a variety of bidding strategies. With its intuitive design and robust features, BidCraft offers a seamless experience for both buyers and sellers, fostering transparency, trust, and excitement in the online auction process. Moreover, BidCraft continually evolves by incorporating user feedback and integrating cutting-edge technologies, ensuring a constantly improving platform tailored to meet the evolving needs of its users.

1.2 Scope of the System

The scope of the **BidCraft Online Bidding Platform** encompasses various aspects related to its functionality, features, target audience, and intended usage. Here's an outline of the scope:

I. Functionality:

- Online Auction Management: Facilitating the creation, management, and monitoring of auctions.
- Allowing users to place bids on items within auctions.
- User Authentication and Profiles: Managing user accounts and profiles securely.
- Notifications: Sending alerts for bid updates, auction status changes, and other relevant events.
- Payment Integration: Integrating secure payment methods for transactions between buyers and sellers.

II. Features:

- User-Friendly Interface: Intuitive design to enhance user experience and ease of navigation.
- Search and Discovery: Tools for users to discover items of interest easily
- Reporting and Analytics: Providing insights into auction performance, bidder behavior, and listing effectiveness.
- Secure Transactions: Implementing robust security measures to protect user data and financial transactions.

III. Target Audience:

- Individual Consumers: People interested in finding unique items or securing deals through online auctions.
- Businesses: Entities looking to sell surplus inventory, unique products, or specialized goods through an online platform.
- Auction Enthusiasts: Individuals who enjoy participating in bidding and collecting items of interest.

IV. Intended Usage:

- Buying and Selling: Providing a platform for users to buy and sell items through the auction format.
- Participation and Engagement: Encouraging active participation and engagement among users through bidding and interaction.
- Community Building: Fostering a community of buyers and sellers who share common interests in unique items and auctions.
- Convenience and Efficiency: Offering a convenient and efficient alternative to traditional auction formats, accessible from anywhere with an internet connection.

Overall, the scope of BidCraft is to provide a comprehensive online auction platform that facilitates smooth transactions, fosters engagement, and offers a dynamic marketplace for users to discover, bid, and sell a wide range of items.

1.3 Objective and success criteria of the project

The primary objectives of BidCraft include:

- Creating a user-friendly online bidding platform that caters to users of all experience levels.
- Facilitating seamless auctions for a diverse range of products and services.
- Attracting a broad user base of both sellers and buyers through innovative features.
- Ensuring secure and transparent transactions to build trust among users.
- Implementing features for efficient bid management and comprehensive auction tracking.
- Providing a dynamic marketplace for unique and in-demand items to captivate users.
- Enhancing user engagement through interactive bidding features and personalized recommendations.

1.4 Definitions and Acronyms and abbreviations

Here some Definitions and Acronyms and abbreviations that i use in my project

- Software Design Document (RAD) is a software development methodology that prioritizes rapid prototyping and iterative development over traditional planning processes.
- Unified Modeling Language (UML) is a standardized modeling language used in software engineering to visually represent a system's architecture, design, and implementation.

1.5 References

References

[1] Medium : https://medium.com/technical-writing-is-easy/what-is-a-software-design-document-sdd-6a3ea3a953b2

[2] Kinsta : https://kinsta.com/blog/fastapi/

[3] LucidChart : https://www.lucidchart.com/pages/

1.6 Overview

The BidCraft Online Bidding Platform is a new way to do auctions on the internet. It's easy to use for both people selling things and people buying things. You can find cool stuff, bid on it, and see how the bidding is going in real-time. BidCraft is all about making sure everyone feels safe and can trust the process. It's like a fun online market where you can find unique items and enjoy bidding on them. And the best part is, BidCraft is always getting better thanks to feedback from users and new technology. So get ready to have fun bidding and discovering amazing things with BidCraft!

2 System Description

2.1 Product Perspective

The product perspective of the BidCraft project is to provide a client-to-client e-commerce platform where a user can buy and sell products by making a bid. On this platform, registered users can list their old or new products, setting a minimum selling price and a specified time range. Other registered users can then place bids on the product. After the designated period, the person with the highest bid will be the successful buyer. The platform will be accessible in English and include an interactive interface for posting, bidding, and a secure payment system. Additionally, we will maintain user profiles to allow buyers to provide ratings, enhancing the trustworthiness of sellers. The platform will also feature a Q&A option, allowing users to inquire about products, and sellers can respond to these queries. A search option will assist buyers in finding specific products, while a filter system will enable users to narrow down their search based on price ranges. Furthermore, product categories will be available, offering a convenient way for users to locate specific types of products in one place.
In the long term, our commitment is to enhance the security and services offered on the platform continually. To ensure the authenticity of product categories, quality, and sellers, we have established and managed a dedicated team responsible for meticulously verifying products. The listing process is conducted by our team members, instilling confidence in buyers to place bids with trust. As a client-to-client application, our platform currently does not facilitate delivery or doorstep services. However, we have a dedicated team in place to efficiently manage these aspects. Additionally, a comprehensive warranty is provided, categorizing products into two types: BidCraft Verified and Unverified. We provide a warranty on verified products. Our assurance is to deliver 100% security, reliable services, and products of verified quality to the buyers.
The platform will cater to two distinct user types: public users and registered users. Public users will have the capability to view products and navigate the site, while registered users will enjoy a more comprehensive set of features. Registered users will be able to enlist products, place bids, provide feedback, upvote or downvote products, and engage in communication through contact/chat functionalities. In the long term, our vision includes the establishment of specialized staff roles to enhance user experience. This will involve a dedicated team for product verification, technicians to address warranty-related and other technical concerns, as well as delivery personnel responsible for ensuring the seamless delivery of purchased products to buyers.
Overall Bybid is a buyer-to-buyer e-commerce site. The platform is committed to providing a secure and trustworthy environment for buyers and sellers, with a focus on product authenticity, quality assurance, and reliable services. As the platform evolves, we strive to continually improve security measures and overall service quality to meet the needs of our users.

3 Design Overview

3.1 Design Rationale

In our front-end development, we've opted for React due to its ability to accelerate and streamline the development process, offering essential components without the need for intricate handling of host device operating system APIs. With React, we eliminate the requirement for native frameworks or separate codebases. Additionally, React's built-in data persistence functionality fulfills a crucial need for our application. Leveraging React alongside JavaScript also positions us well for expanding to other platforms, as both are inherently geared towards cross-platform development from the outset.

On the backend, we're using FastAPI framework to create the server.FastAPI, a lightweight web framework designed for crafting modern APIs with Python 3.6 and above, has garnered acclaim in the developer community for its speed, ease of use, and comprehensive documentation. Renowned for its rapid performance comparable to Go and Node.js, FastAPI offers pre-configured features that allow developers to create production-ready APIs with minimal code. Leveraging OpenAPI standards, FastAPI dynamically generates detailed documentation, streamlining the API development process. Its robust data validation system, supporting custom validation, significantly reduces bugs, potentially decreasing errors by up to 40%. Additionally, FastAPI's integration of type hints enhances code quality and productivity by providing better support and error prediction in IDEs. As one of the top open-source frameworks of 2021, FastAPI continues to revolutionize API development, attracting developers seeking efficient and robust solutions for their projects.

We're using Firebase for Authenticates users. Firebase Authentication offers a streamlined solution for robust and scalable user authentication. With its easy integration, secure methods, and built-in features like identity verification and scalability, developers can quickly implement a reliable authentication system.

Taking all of these considerations into account, it can be inferred that the most suitable architectural pattern for the application is the Model-View-Controller (MVC) pattern. This pattern effectively separates the business logic from the presentation logic, thereby improving the application's maintainability and flexibility.

3.2 System Architecture

The system architecture of our application website follows a modular program structure to achieve complete functionality. Designing an online bids platform requires careful consideration of various components to ensure scalability, security, and efficiency. Here Here's a high-level overview of the system architecture:

3.2.1 Client Interface:

- Interface: A user-friendly interface allowing users to browse, search, and place bids on items.
- APIs: Restful APIs for communication between the client interface and the server.

3.2.2 Server Side:

- Web Server: Handles incoming requests from clients, manages sessions, and serves dynamic content.

- Application Server: Implements the business logic, manages user authentication, authorization, bidding processes, and communicates with the database.

3.2.3 Database Layer:

- Database Server Integration: Stores user data, item details, bidding history, and other relevant information.

3.2.4 External Services:

- Payment Gateway: Integrates with a third-party payment gateway for secure payment processing.
- Email and Notification Service: Sends notifications for bid updates, auction status, payment confirmation, etc.

3.2.5 Development Tools:

- Version Control: Git for managing source code revisions and collaboration among developers.
- Continuous Integration/Continuous Deployment (CI/CD) Pipeline: Automates the build, test, and deployment processes, ensuring rapid and reliable software delivery.

This architecture provides a robust foundation for building an online bids platform that can handle high traffic loads, ensure data security, and deliver a seamless user experience.

3.3 Assumption and Dependencies

3.3.1 Assumption

- There will be no unexpected difficulties with the application's backend, which might cause substantial delays or even failure in development.
- There will be no schedule conflicts or time limits during the development process, which might adversely impact the project's timeframe.
- The device that will run the program must fulfill the minimal hardware requirements.
- To use the program without interruptions, the user must have access to a reliable internet connection.
- The user will have basic computer literacy and be able to navigate and use the platform's features effectively.
- Users will be able to accurately and clearly articulate their questions and problems, which will allow other users to provide helpful and relevant answers.

3.3.2 Dependencies

3.3.2.1 Front-end dependencies (React):

- `react`: The core library for building React components.
- `react-dom`: This package serves as the entry point to the DOM and server renderers for React.

- `react-scripts`: Includes scripts and configuration used by Create React App.
- `axios`: For making HTTP requests, providing more features than the native fetch API.
- `react-router-dom`: For managing routing in React web applications.
- `@material-ui/core`: Material UI framework for React components.
- `@material-ui/icons`: Icons from Material UI for use in React projects.
- `styled-components`: Allows the use of component-level styles using tagged template literals.
- `redux`: A state management library often used in React apps for managing application state.
- `react-redux`: Official React bindings for Redux.
- `@testing-library/react`: For DOM-based component testing in React.
- `jest`: A JavaScript Testing Framework with a focus on simplicity.

3.3.2.2 Back-end dependencies (FAST API)

- **Database Connection:** A dependency to establish and manage connections to a database. This can include functions or classes for connecting to the database, executing queries, and handling transactions.
- **Authentication:** Dependencies for handling user authentication, including functions or classes to verify user credentials, generate tokens, and manage session data.
- **Authorization:** Dependencies to enforce access control and permissions. This can include functions or classes to check if a user has the necessary permissions to access certain resources or perform specific actions.
- **Data Validation:** Dependencies for validating incoming data against predefined schemas or constraints.
- **Logging:** Dependencies for logging application events and errors. This can include functions or classes to configure logging settings, format log messages, and store log entries in different destinations (e.g., files, databases, or external services).
- **Caching:** Dependencies for caching frequently accessed data to improve performance.
- **External Services Integration:** Dependencies for integrating with external services or APIs. This can include functions or classes to make HTTP requests, handle responses, and manage API authentication and authorization.
- **Configuration Management:** Dependencies for managing application configuration settings. This can include functions or classes to load configuration files, parse environment variables.
- **Dependency Injection:** Dependencies for injecting dependencies into endpoint functions or other components.
- **Request Rate Limiting:** Dependencies for limiting the rate of incoming requests to prevent abuse or excessive resource consumption.

4 Object Model

4.1 Object Description

4.1.1 GuestUserController <<controller>>

ClassName	GuestUserController
Brief Description	Controller object handling all functionalities related to non-registered and unauthorized user
Methods	**Description**
accessLandingPage()	Allows non-registered users to access the landing page
viewPosts()	Allows non-registered users to view posts
searchPosts()	Allows non-registered users to search posts
accessSignUp()	Allows non-registered users to access signup
viewComment()	Allows non-registered users to view posts comment
viewQ&A()	Allows non-registered users to view posts Q&A

4.1.2 SignupController <<Controller>>

Class Name	SignUpController
Brief Description	Controller object handling user registration
Methods	**Description**
userRegistration(email, pass-word, name)	allows users to register using their email address and pass-word and name. They can also register using their google or facebook account through Fire-base

4.1.3 SigninController Controller

Class Name	SignInController
Brief Description	Controller object handling user authentication
Methods	**Description**
userLogin(email, password)	Allows registered users to log in to their accounts using their email address and password

4.1.4 AuthenticationController <<Controller>>

Class Name	FirebaseAuthentication
Brief Description	Object representing Firebase Authentication
auth	The Firebase Authentication instance
user	The currently authenticated user object
Methods	**Description**
signUpWithEmail(email: Email, password: string)	Signs up a new user with the provided email address and password
signInWithEmail(email: Email, password: string)	Signs in a user with the provided email address and password
signInWithGoogle()	Signs in a user with their Google account

Class Name	FirebaseAuthentication
signInWithFacebook()	Signs in a user with their Facebook account
signOut()	Signs out the current user
resetPassword(email: Email)	Sends a password reset email to the provided email address
verifyEmail()	Sends an email verification email to the current user's email address
getCurrentUser()	Returns the currently authenticated user object
isAuthenticated()	Returns true if a user is currently authenticated, false otherwise

4.1.5 User `<<entity>>`

Class Name	User
Brief Description	Object representing the user's information
userid	Unique identifier for the user
role	Role Authorization for the user
name	Name of the user
email	email address of the user
userpic	Image of the user
description	Optional description provided by the user
posts	List of posts that the user has created
bids	List of items that the user did bid
comments	List of comments that the user did
card	List of item that the user save
buy	List of item that the user buy
sell	List of item that the user sell
searches	List of searches that the user did
que	List of question that user did
ans	List of answer that user did
Methods	**Description**
addPost(post)	Add a post to the user's list of posts
removePost(post)	Remove a post from the user's list of posts
addbid(item)	Place a bid in a item
removebid(item)	Remove bid from item within in a limited time
addComment(comment)	Add a comment to the user's list of comments
removeComment(comment)	Remove a comment from the user's list of comments
addQuetion(question)	Add a question to the user's list of question
removeQuestion(Question)	Remove a comment from the user's list of question
addAnswer(Answer)	Add a question to the user's list of Answer
removeAnswer(Answer)	Remove a comment from the user's list of Answer
addCard(item)	Add a item to the user's list of Card
removeCard(item)	Remove a item from the user's list of Card
addSearch(search)	Add a search to the user's list of searches
removeSearch(search)	Remove a search from the user's list of searches
addTag(tag)	Add a tag to the user's list of tags
removeTag(tag)	Remove a tag from the user's list of tags
getPosts()	Get a list of all posts created by the user
getComments()	Get a list of all comments made by the user
showCard()	Get a list of all item in Card

Class Name	User
showShoping()	Get a list of all item in Buy this user
show-Selling()	Get a list of all item in sell this user
getSearches()	Get a list of all searches made by the user
getQues()	Get a list of all question did by the user
getAns()	Get a list of all answer did by the user
editProfile()	Edit profile info

4.1.6 DatabaseController <<controller>>

Class Name	Database Connection
Brief Description	Object representing a connection to a database
host	Host name of the database server
port	Port number on which the database server is listening
database	Name of the database to connect to
user	Username to use for authentication
password	Password to use for authentication
Methods	**Description**
connect()	Establishes a connection to the database
disconnect()	Closes the connection to the database
execute(query)	Executes a SQL query on the connected database
fetchall()	Returns all rows from the most recent query as a list of tuples
fetchone()	Returns the next row from the most recent query as a tuple
commit()	Commits the current transaction to the database
rollback()	Rolls back the current transaction

4.1.7 Staff <<entity>>

Class Name	Staff
Brief Description	Object representing the staff's information
name	Name of the staff
role	Role of the staff in the business
email	Email address of the staff
Method	**Description**
approvePost(post)	Approve a post for publication
takeAction(post)	Take necessary action on a post (e.g., moderation)
reviewPost(post)	Review a post before publication
escalateIssue(issue)	Escalate an issue for further attention or action

4.1.8 DeliveryMan <<Entity>>

Class	DeliveryMan
Brief Description	Object representing the staff's information and functionality
deliveryman_id	Unique identifier for the delivery man
name	Name of the delivery man
email	Email address of the delivery man
phone_number	Phone number of the delivery man

Class	DeliveryMan
vehicle_type	Type of vehicle used for delivery (e.g., car, bike, truck)
current_location	Current location of the delivery man
pending_orders	List of orders assigned but not delivered
delivered_orders	List of orders successfully delivered
failed_orders	List of orders failed to be delivered
Method	**Description**
acceptOrder(order)	Accept a new order for delivery
updateLocation(location)	Update the current location of the delivery man
markAsDelivered(order)	Mark an order as delivered
markAsFailed(order)	Mark an order as failed to be delivered
viewPendingOrders()	View the list of pending orders
viewDeliveredOrders()	View the list of successfully delivered orders
viewFailedOrders()	View the list of failed orders
contactCustomer(order)	Contact the customer associated with a specific order

4.1.9 Postcontroller `<<controller>>`

Class	PostController
Brief Description	Controller object handling all functionalities related to user post
post_id	Unique identifier for the post
user_id	ID of the user who created the post
title	Title of the post
description	Description or details of the post
images	Images associated with the post
starting_bid	Initial bid amount set by the user
current_bid	Current highest bid on the post
bidding_end_time	Deadline for bidding on the post
status	Status of the post (e.g., active, closed)
category	Category or type of the post
Method	**Description**
createPost(user_id, title, description, images, starting_bid, bidding_end_time, category)	Create a new post
editPost(post_id, title, description, images, starting_bid, bidding_end_time, category)	Edit an existing post
deletePost(post_id)	Delete a post
viewPost(post_id)	View details of a specific post
browsePosts()	Browse all posts available in the system
searchPosts(query)	Search for posts based on a query
getMyPosts(user_id)	Get all posts created by a specific user
placeBid(post_id, user_id, bid_amount)	Place a bid on a post
closeBidding(post_id)	Close bidding on a post and determine the winner
updatePost(post_id)	Update post if need.

4.1.10 PostElementController <<controller>>

Class	PostElementController
Brief Description	PostElementController that manages comments, questions, answers, and other elements related to a post
post_id	ID of the post associated with the elements
comments	List of comments associated with the post
questions	List of questions associated with the post
answers	List of answers associated with the post
other_elements	List of other elements associated with the post
Method	**Description**
addComment(post_id, comment)	Add a comment to the specified post
removeComment(post_id, comment_id)	Remove a comment from the specified post
getComments(post_id)	Retrieve all comments associated with the post
addQuestion(post_id, question)	Add a question to the specified post
removeQuestion(post_id, question_id)	Remove a question from the specified post
getQuestions(post_id)	Retrieve all questions associated with the post
addAnswer(post_id, question_id, answer)	Add an answer to the specified question
removeAnswer(post_id, question_id, answer_id)	Remove an answer from the specified question
getAnswers(post_id, question_id)	Retrieve all answers associated with the specified question
addElement(post_id, element)	Add another element to the specified post
removeElement(post_id, element_id)	Remove a specific element from the post
getElements(post_id)	Retrieve all other elements associated with the post

4.1.11 UserController <<controller>>

Class	PostElementController
Brief Description	PostElementController that manages comments, questions, answers, and other elements related to a post
landing_page_data	Data related to the landing page
user_data	Data related to the user
Method	**Description**
displayLandingPage()	Display the landing page with relevant data
updateUserProfile(userData)	Update user profile data
getUserData()	Retrieve user-specific data
signIn()	Load sing in and sing up page
updateLandingPageData(data)	Update landing page data with new information
search(query)	Perform a search based on the given query
addPost(postData)	Add a new post to the landing page

4.1.12 SearchController <<controller>>

Class	SearchController
Brief Description	Object representing the search functionality of the system
search_id	Unique identifier for the search
user_id	ID of the user performing the search
search_query	Text query entered by the user for the search
search_results	List of search results returned to the user
Method	**Description**
search(query)	Perform a search based on the given query
saveSearchResult(result)	Save a search result to the database
getUserSearchHistory(user_id)	Retrieve the search history for a specific user
clearSearchHistory(user_id)	Clear the search history for a specific user

4.1.13 Chatcontroller <<controller>>

Class	ChatController
Brief Description	Object representing a chat conversation between two user
chat_list	List of chats involving the user
chat_id	Unique identifier for the chat
Method	**Description**
getChatList(user_id)	Retrieve the list of chats for the specified user
createChat(user_id, recipient_id)	Create a new chat between the user and another recipient
sendMessage(chat_id, sender_id, message)	Send a message in the specified chat
getChat(chat_id)	Retrieve details of the specified chat
deleteChat(chat_id)	Delete the specified chat from the user's chat list
deleteConversation(user_id)	Delete the conservation with the specific user's

4.2 Object Collaboration Diagram

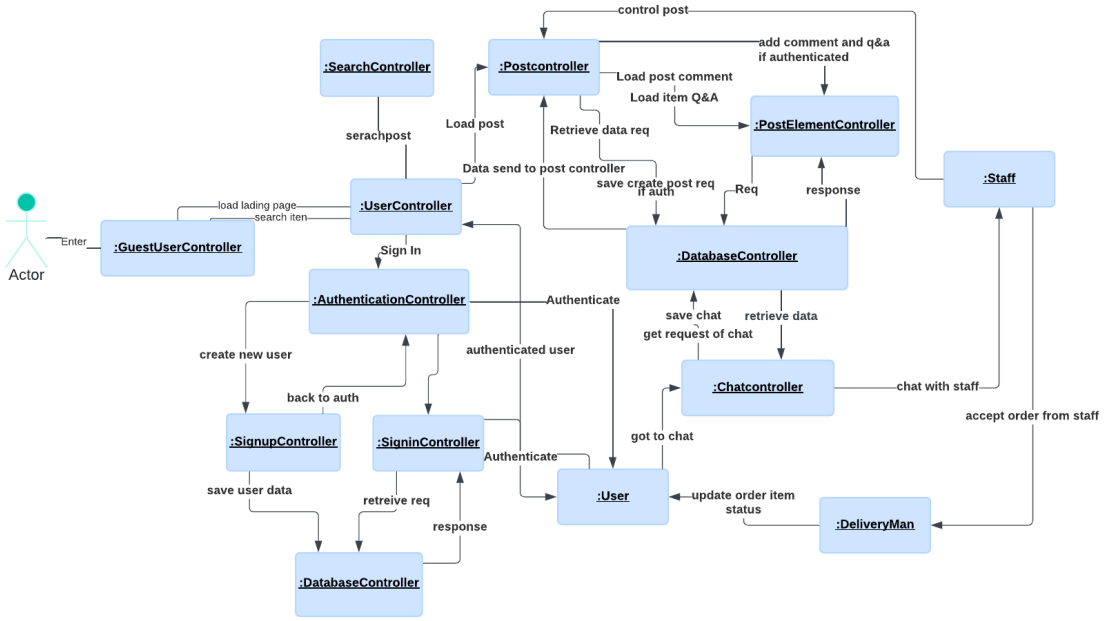

Figure 1: Object Collaboration Diagram

5 Subsystem Decomposition

5.1 Complete Package Diagram

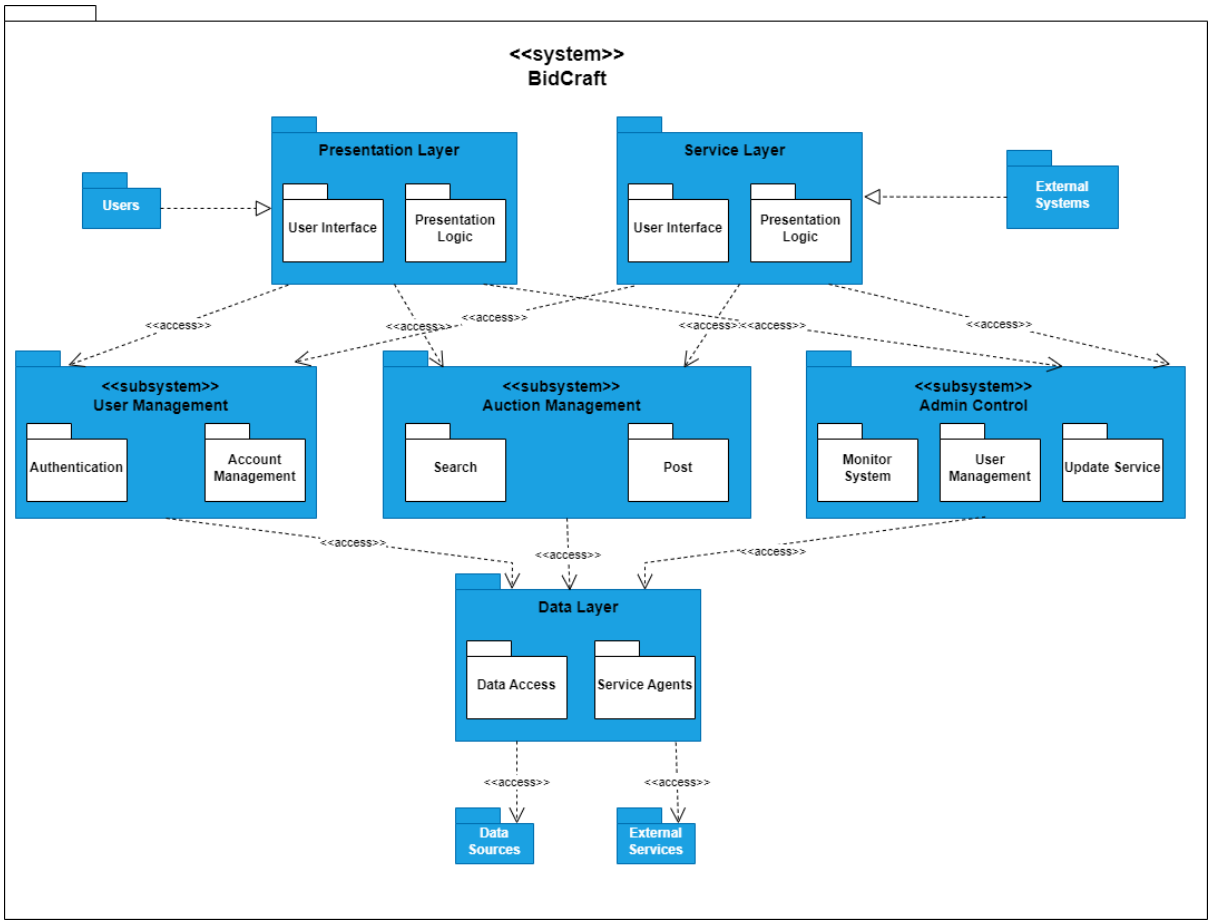

Figure 2: Package Diagram

5.2 Subsystem Detail Description

5.2.1 User Management

5.2.1.1 Module Description This module will handle user authentication, registration, and profile management. It will be responsible for user authorization and access control.

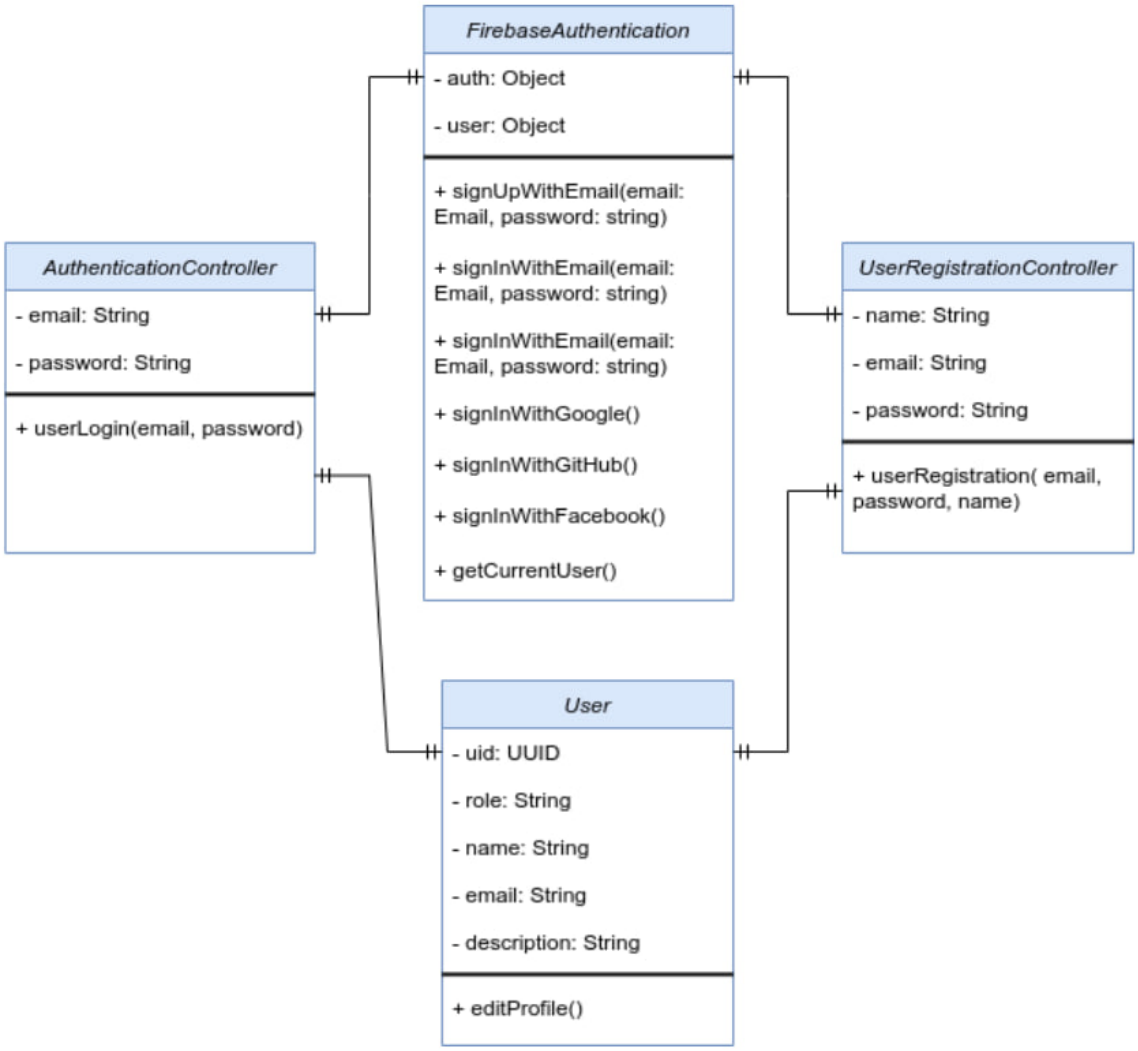

Figure 3: User Management Class Diagram

5.2.1.2 Subsystem Interfaces From the user's perspective, the User Management module will provide a seamless experience for user authentication, registration, and profile management. Users will be able to create an account, log in, and access their profile information from a single interface.

To register, users will need to provide their basic information, including their name, email address, and password. Once registered, users will be able to log in to the system using their email and password credentials. In case of incorrect login credentials, the system will display an error message to inform the user about the issue.

Users will also be able to manage their profile information, including their contact information, profile picture, and other relevant details. They will be able to update their profile information and save the changes in the system. Feedback information will be displayed to the user to confirm that their changes have been saved successfully.

The User Management module will also handle user authorization and access control. Users with appropriate permissions will be able to access certain features of the system, while others will not. Feedback information will be displayed to inform users if they don't have sufficient permissions to access specific features.

Overall, the User Management module will provide a secure and user-friendly interface for managing user authentication, registration, and profile information.

5.2.2 Post Management

5.2.2.1 Module Description The module will provide an interface for users to post and create an auction. It will It will also allow users to search for previously posted products and their current bid amount. Staffs can approve or remove posts and delivery man will deliver the product to the buyer.

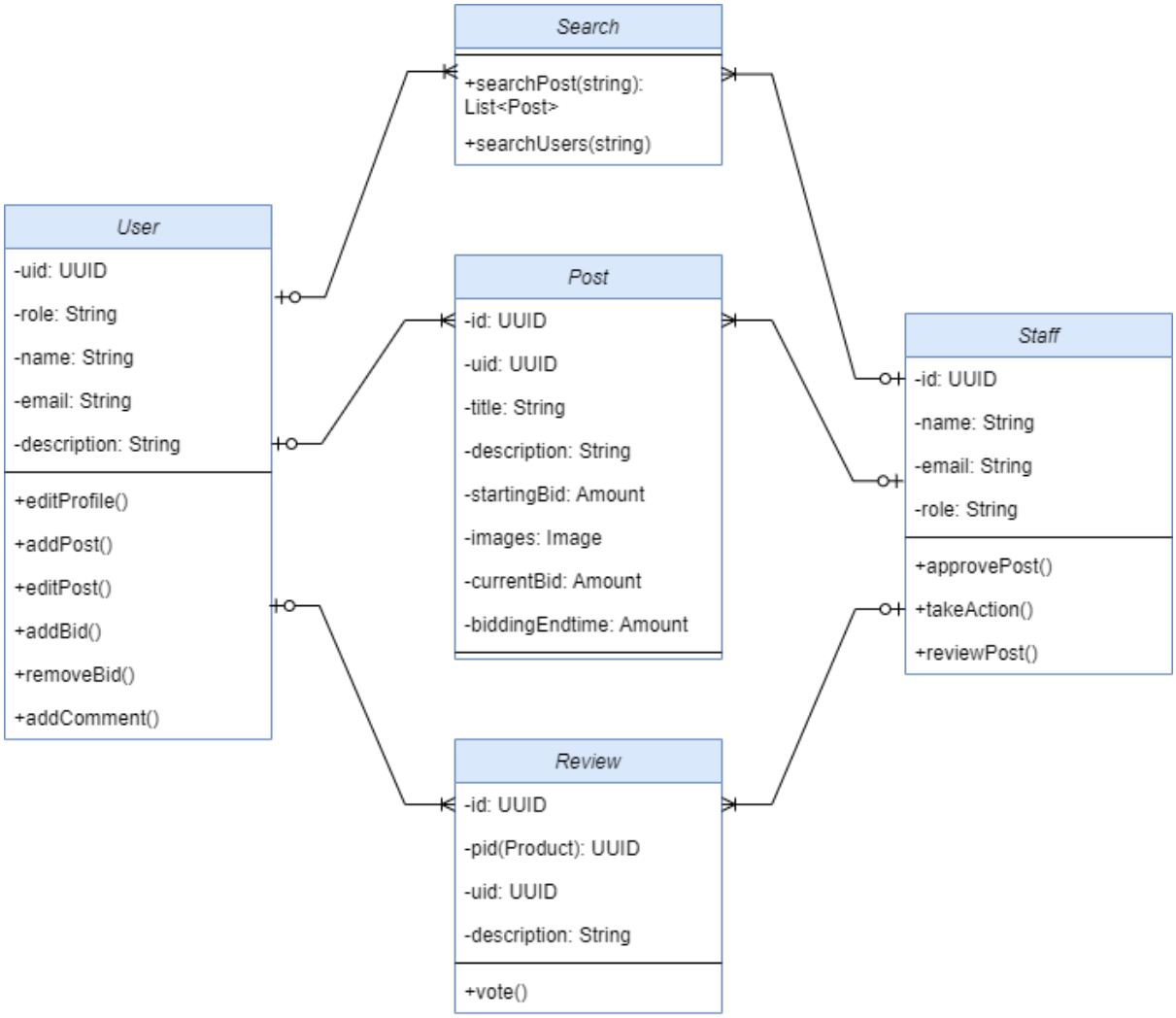

Figure 4: User Management Class Diagram

5.2.2.2 Subsystem Interfaces The post management subsystem will provide users ability to create, search, and review posts, each of which may represent an item or service available for bidding. When creating a post, users can provide details such as descriptions, starting bid amounts, and images, which are critical for attracting bids. Additionally, users can search for posts based on various criteria like categories, price ranges, or keywords, enhancing the user experience by making it easier to find relevant items. They can also leave reviews and feedback on posts, influencing the credibility and attractiveness of sellers within the marketplace.

On the administrative side, staff members play a pivotal role in maintaining the quality and security of the marketplace. They have the authority to approve posts, ensuring that each item meets the platform's standards and guidelines before becoming visible to other users. Staff can also remove posts if they violate terms of service or show misleading information, thereby protecting the integrity of the bidding environment.

5.2.3 Admin Control

5.2.3.1 Module Description The module will handle all admin operations. Verifying Service provider, monitoring overall system will be handled by this module. Only admin users have the access to this subsystem.

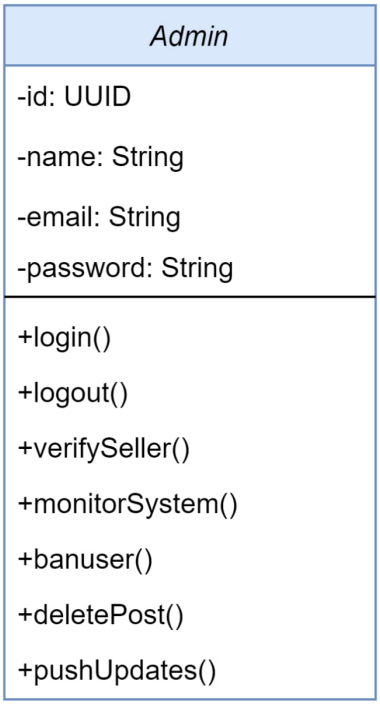

Figure 5: Admin Control Subsystem Class Diagram

5.2.3.2 Subsystem Interfaces The Admin Control subsystem is responsible for managing the overall system and ensuring that all operations are running smoothly. Only admin users have access to this subsystem, which includes features such as verifying technical experts, monitoring system performance, and managing user accounts.

The Admin Control subsystem is responsible for managing the overall system and ensuring that all operations are running smoothly. Only admin users have access to this subsystem, which includes features such as verifying technical experts, monitoring system performance, and managing user accounts.

The user account management section will allow the admin user to manage user accounts, including adding new users, deleting accounts, and modifying user details. Admin users can also reset passwords and update user permissions.

The system monitoring section will provide the admin user with real-time information about system performance and usage. This will include metrics such as the number of active users, the number of auctions posted, and auction times.

The Admin Control subsystem will provide feedback information to the user through the dashboard interface. Notifications will be displayed if there are any auctions with the system, such as server downtime or problems with user accounts. The dashboard will also display information about the status of service provider verification and user account management tasks.

In summary, the Admin Control subsystem will provide admin users with the necessary tools and features to manage the overall system, including verifying sellers, managing user accounts, and monitoring system performance. The dashboard interface will display real-time feedback information to the user, allowing them to make informed decisions and take action quickly to resolve any issues that may arise.

6 Data Design

6.1 Data Description

The BidCraft Online Bidding Platform utilizes a relational database to store and manage its data efficiently. The data structures are designed to accommodate the major entities: `User`, `Auction`, `Bid`, and `Item`.

- **User**: Stores information about registered users, including their unique identifiers, usernames, email addresses, hashed passwords for security, and the date when their accounts were created.
- **Auction**: Represents individual auctions listed on the platform, storing details such as the item name, description, starting price, start date, end date, and the identifier of the user who listed the auction.
- **Bid**: Records bids placed by users on auctions, capturing details like the bid amount, the auction being bid on, the user placing the bid, and the timestamp of the bid.
- **Item**: Manages individual items listed by users, storing information such as the item name, description, category, and the identifier of the user listing the item.
- **Review**:It manages review for an item.Actually it represents the review for the item that is already sold by an user.

The relational database ensures data integrity and allows for efficient querying and retrieval of information based on various parameters such as user activity, auction status, and bid history.

6.2 Data Dictionary

User

- **Type**: Entity
- **Description**: Represents registered users on the platform.
- **Attributes**:
  - `userId`: Unique identifier for the user (Integer)
  - `username`: User's username (String)
  - `email`: User's email address (String)
  - `passwordHash`: Hashed password for security (String)
  - `createdDate`: Date when the user account was created (Date)

Auction

- **Type**: Entity
- **Description**: Represents individual auctions listed on the platform.
- **Attributes**:
  - `auctionId`: Unique identifier for the auction (Integer)
  - `itemName`: Name of the item being auctioned (String)

  - `description`: Detailed description of the item (String)
  - `startingPrice`: Initial bidding price (Float)
  - `startDate`: Date when the auction starts (Date)
  - `endDate`: Date when the auction ends (Date)
  - `sellerId`: Identifier for the user who listed the auction (Foreign Key - Integer)

Bid

- **Type**: Entity
- **Description**: Represents bids placed by users on auctions.
- **Attributes**:
  - `bidId`: Unique identifier for the bid (Integer)
  - `auctionId`: Identifier for the auction being bid on (Foreign Key - Integer)
  - `userId`: Identifier for the user placing the bid (Foreign Key - Integer)
  - `bidAmount`: Amount of the bid (Float)
  - `bidTime`: Timestamp when the bid was placed (Date)

Item

- **Type**: Entity
- **Description**: Represents individual items listed by users.
- **Attributes**:
  - `itemId`: Unique identifier for the item (Integer)
  - `itemName`: Name of the item (String)
  - `description`: Detailed description of the item (String)
  - `category`: Category of the item (String)
  - `sellerId`: Identifier for the user listing the item (Foreign Key - Integer)

Review

- **Type**: Entity
- **Description**: Represents reviews posted by users for items listed on the platform.
- **Attributes**:
  - `reviewId`: Unique identifier for the review (Integer)
  - `itemId`: Identifier for the reviewed item (Foreign Key - Integer)
  - `sellerId`: Identifier for the owner of the reviewed item (Foreign Key - Integer)
  - `sellingPrice`: Selling price of the item reviewed (Float)
  - `reviewerId`: Identifier for the user who bought it.
  - `comment`: Comment provided by the reviewer (String)

Functions:

- `placeBid(auctionId, userId, bidAmount)`: Places a bid on a specific auction.
  - **Parameters**:
    - `auctionId`: Identifier of the auction (Integer)
    - `userId`: Identifier of the user placing the bid (Integer)
    - `bidAmount`: Amount of the bid (Float)
- `createAuction(userId, itemName, description, startingPrice, startDate, endDate)`: Creates a new auction listing.
  - **Parameters**:
    - `userId`: Identifier of the user listing the auction (Integer)
    - `itemName`: Name of the item (String)
    - `description`: Detailed description of the item (String)
    - `startingPrice`: Initial bidding price (Float)
    - `startDate`: Date when the auction starts (Date)
    - `endDate`: Date when the auction ends (Date)

6.3 Entity Relationship Diagram

The Entity Relationship Diagram (ERD) for the BidCraft Online Bidding Platform is illustrated in Figure 6 below.

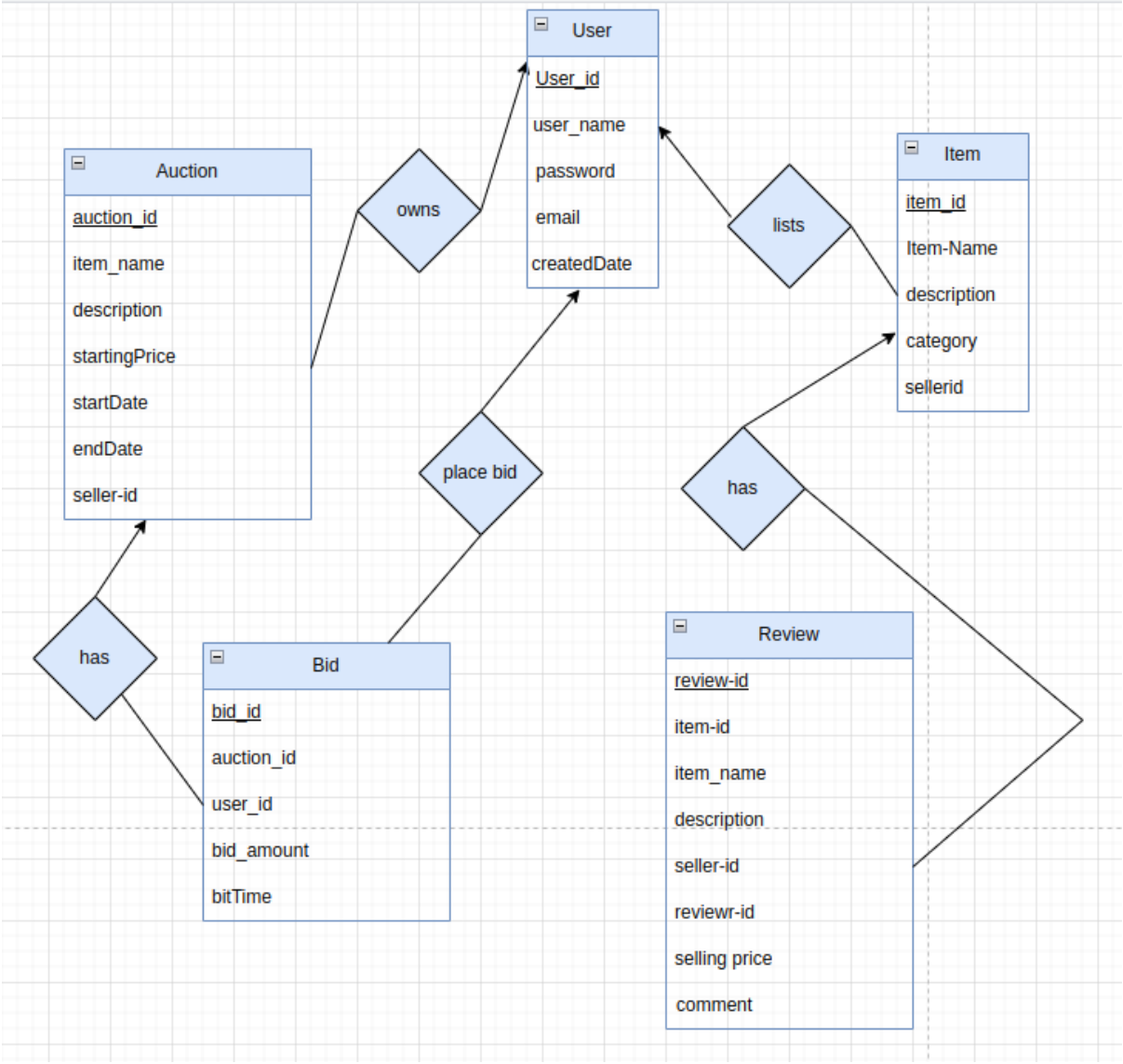

Figure 6: ER Diagram for Database Model

7 User Requirement and Component Traceability Matrix

User Requirement	User Management	Product Management	Staff Management	Data Management
User Visits as Guest	x	x		
Authentication	x	x	x	x
Listing Products	x	x	x	x
Bidding System	x	x		x
Search		x		x
Product Categories		x		x
User Engagement	x		x	
User Profiles	x			x
Q&A Option	x	x	x	x
Technical Support Request	x		x	

8 Supporting Information

This document does not need any supporting information from other documents. All the diagrams presented in this document were created with LucidChart[3] Online Diagram maker.