

***UE21CS352B - Object Oriented Analysis & Design using Java***

**Mini Project Report**

**“AUCTIONING APPLICATION”**

***Submitted by:***

|  |  |
| --- | --- |
| **Sri Harshitha A**  **Vijapura Akibbhai**  **Aditi M**  **Srimanish C** | **PES1UG21CS615**  **PES1UG22CS847**  **PES1UG21CS901**  **PES1UG21CS936** |

*6th Semester K**Section*

|  |
| --- |
| **Prof. Bhargavi Mokashi**  Asst. Professor |

**January - May 2024**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

FACULTY OF ENGINEERING

**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Table of Contents

1. Problem Statement
2. Abstract
3. Design Models
   1. Use Case
   2. Class
   3. State
   4. Activity
4. Modules
5. Design Patterns & Principles
6. Output GUI
7. Contributions
8. GitHub Repository

**Problem Statement:**

The traditional method of conducting auctions often involves physical presence, limiting participation to a local audience and creating logistical challenges. In response to this, there is a growing demand for online auctioning systems that provide a convenient platform for both sellers and buyers to engage in auction activities remotely. However, existing solutions may lack certain functionalities or may not be tailored to specific needs.

**Abstract:**

The Online Auctioning System is a web application designed to facilitate auction activities in a virtual environment. The system caters to two primary user roles: administrators and customers. Administrators have the authority to post items for auction, manage auctions, and oversee the entire process. On the other hand, customers can view available items, place bids, and participate in auctions.

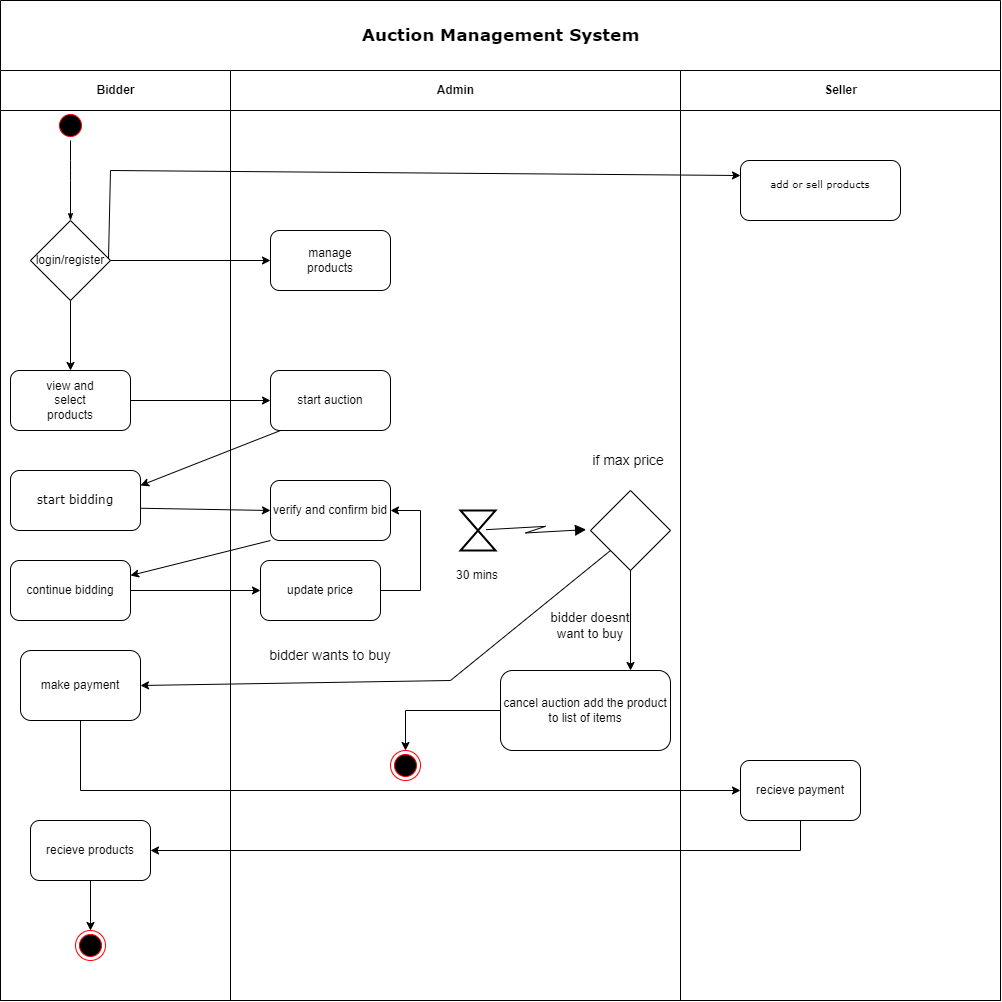
Key features of the system include:

* User and functionality classification: Differentiating between administrators and customers, ensuring secure access to the platform.
* Item management: Administrators can add, edit, and remove items for auction, specifying details such as starting price, auction duration, and images.
* Auction functionality: Automated timers regulate the duration of auctions, allowing customers to place bids within the specified timeframe. The customer with the highest bid at the auction's conclusion wins the item.
* Real-time updates: Customers receive notifications about new items, auction status, and bidding activity.
* Reporting and analytic: Administrators can generate reports on auction performance, including statistics on bids, item popularity, and revenue generation.

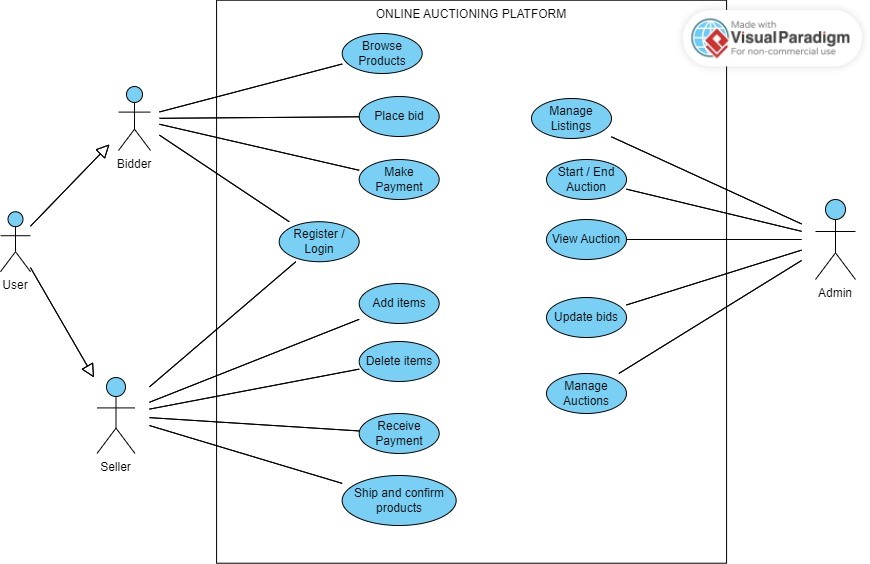
Through the Online Auctioning System, we aim to provide a seamless and user-friendly platform for conducting auctions online, enhancing accessibility and efficiency for both sellers and buyers.

**Design Models:**

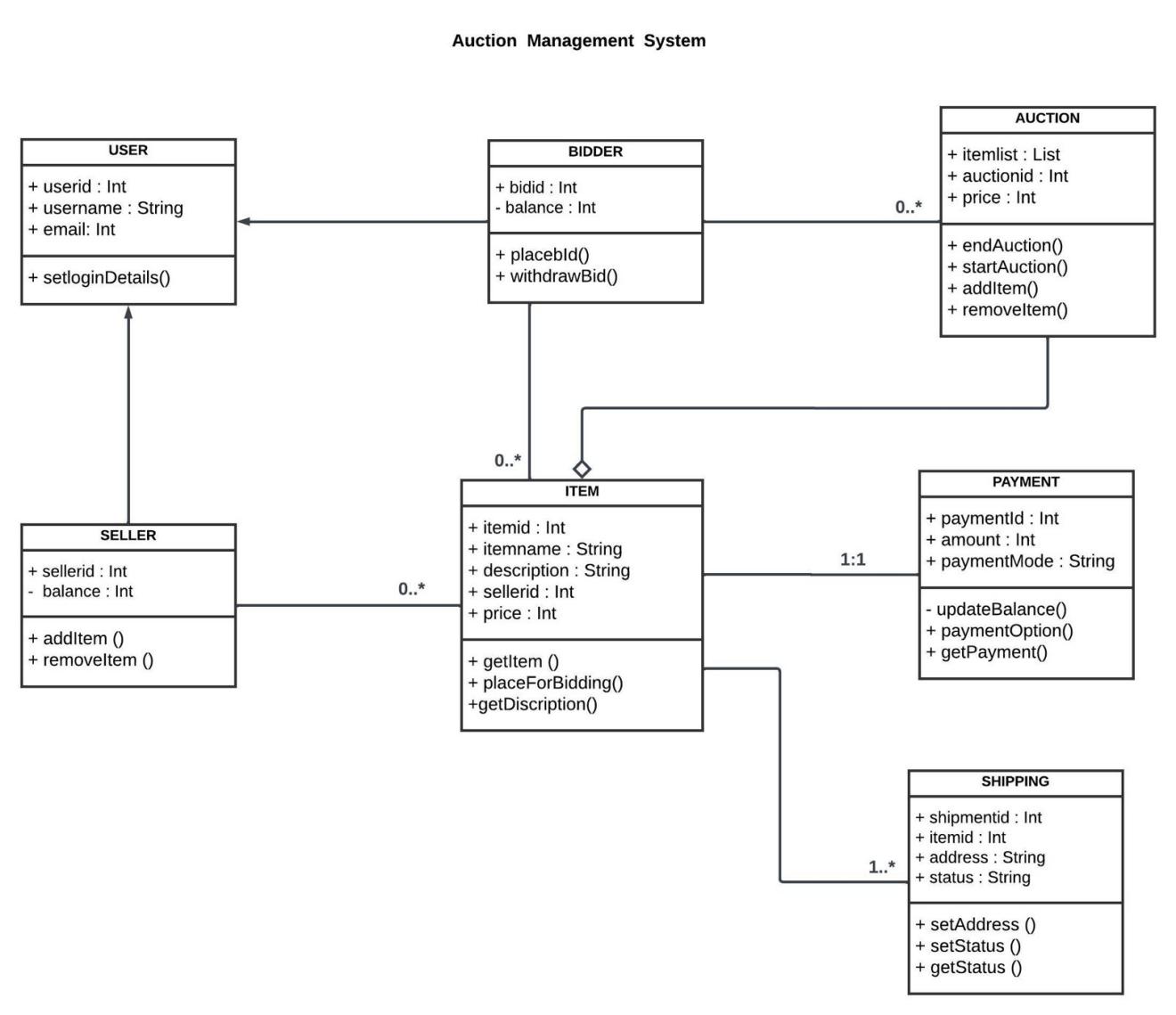
Activity Diagram:

****

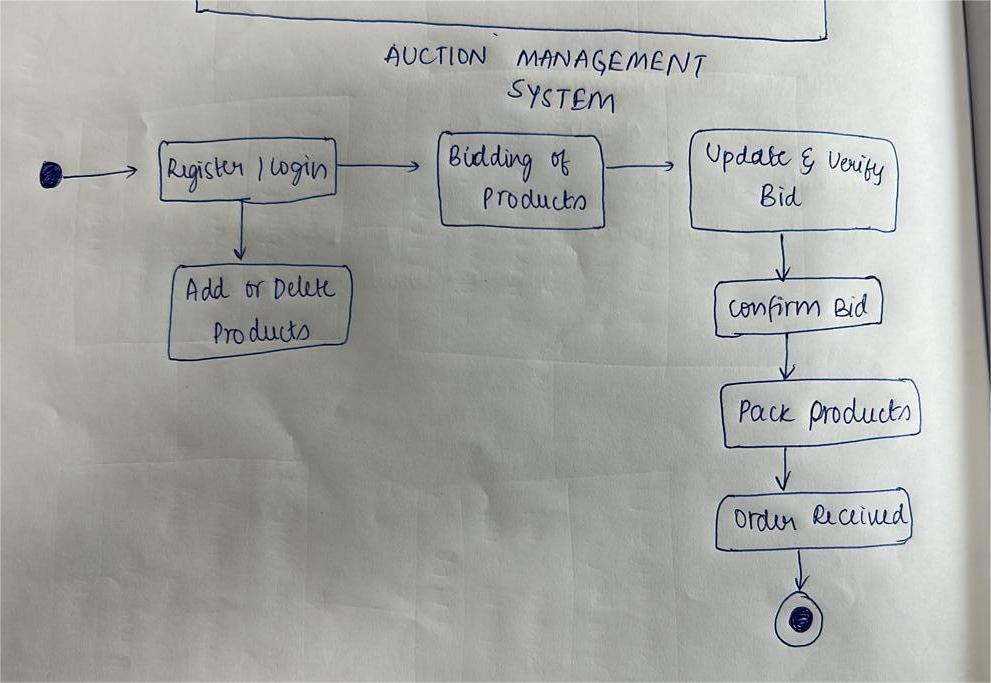
Use case Diagram:



Class Diagram (Proposed):



State Diagram (Proposed):



**Modules:**

Main Screen Module:  
  
The MainScreen module serves as the entry point for the online auctioning system. It is responsible for initializing the main user interface, which allows users to choose between accessing the system as a customer or an administrator. This module primarily consists of a graphical user interface (GUI) implemented using Swing components.

Functionality:

1. Initialization of GUI: Upon instantiation, the MainScreen class initializes the main JFrame (auctionF) and sets up the auctionPanel, which contains two buttons for accessing the system as a customer or an administrator.
2. Button Actions:

* CUSTOMERButton ActionListener: When the "Customer" button is clicked, it triggers an ActionListener, which instantiates a new Customer object. This allows users to access the system's functionalities from a customer's perspective.
* ADMINButton ActionListener: Similarly, when the "Admin" button is clicked, It triggers an ActionListener, which instantiates a new Admin object, granting access to administrative functionalities.
* Main Method: The main method of this module is responsible for starting the GUI within the Event Dispatch Thread (EDT) to ensure thread safety and proper GUI initialization.

Usage:

The MainScreen module is invoked when the application is launched. It presents users with the option to log in either as a customer or an administrator, facilitating access to the respective functionalities of the online auctioning system.

Integration:

The MainScreen module is integrated into the overall system architecture as the initial interface through which users interact with the system. It is interconnected with other modules, such as Customer and Admin, to provide a seamless transition and access to various features based on user roles.

Dependencies:

This module relies on the javax.swing package for implementing GUI components and handling user interactions. It also utilizes the ActionListener interface from the java.awt.event package to capture button click events.

Enhancements:

* Implementing user authentication and authorization mechanisms to ensure secure access to the system based on user roles.
* Enhancing the GUI layout and design to improve user experience and visual appeal.
* Adding error handling and validation to handle unexpected user inputs or system errors gracefully.

Customer Module:

Description:

The Customer module is responsible for providing the user interface and functionalities specific to customers within the online auctioning system. It allows customers to view auction items, place bids, and participate in auctions.

Functionality:

GUI Initialization: Upon instantiation, the Customer class sets up the main JFrame (customerF) with components such as timerLabel, bidDetails table, bidName and bidPrice text fields, ADDBIDButton, itemName label, price label, image label, and close button.

1. Timer Management:

The startTimer method initializes a timer that counts down the remaining time for the auction. It updates the timerLabel accordingly.

When the auction ends (timer reaches 0), the timer stops, and the highest bidder's details are retrieved from the database. The auctioned item's status is updated, and the bid history is cleared.

1. Bidding Functionality:

The ADDBIDButton ActionListener adds a bid to the auction. It validates the bid price and inserts the bid details into the database. If the bid price is lower than the current price, an error message is displayed. After adding the bid, the bid history table is updated.

1. Database Interaction:

The module interacts with a MySQL database to retrieve bid history and update auction status.

It uses JDBC for database connectivity, executing SQL queries to insert bids, retrieve bid history, and update auction status.

1. Table Data Display:

The tableData method retrieves bid history from the database and populates the bidDetails table.

1. Close Button:

The close button ActionListener closes the customer interface JFrame.

Usage:

Customers utilize this module to view auction items, place bids, and monitor auction status in the online auctioning system.

Integration:

The Customer module integrates with the overall system architecture to provide a seamless experience for customers. It interacts with other modules such as Admin for accessing shared data and functionalities.

Dependencies:

Dependent on javax.swing and java.sql packages for GUI implementation and database interaction, respectively.

Enhancements:

Implement user authentication to ensure only registered customers can participate. Enhance GUI design for improved aesthetics and usability. Implement error handling for database connectivity issues and user input validation.

Admin Module:

Description:

The Admin module provides administrative functionalities within the online auctioning system. It allows administrators to manage auction items, start auctions, and monitor auction status. The module includes a graphical user interface (GUI) implemented using Swing components.

Functionality:

1. GUI Initialization: Upon instantiation, the Admin class sets up the main JFrame (adminF) with components such as startButton, timerLabel, ADDITEMButton, SELECTIMAGEButton, CLOSEButton, table1 (for displaying auction items), and input fields for item details.
2. Auction Management:

The startButton ActionListener starts the auction timer, allowing bidding to begin if the item is not already sold. The ADDITEMButton ActionListener adds new items to the auction by inserting their details into the database.

1. Image Selection:

The SELECTIMAGEButton ActionListener allows administrators to select images for auction items using a file chooser dialog. The selected image is displayed in the GUI.

1. Database Interaction:

The module interacts with a MySQL database to insert new auction items and retrieve existing items. It utilizes JDBC for database connectivity, executing SQL queries to perform database operations.

1. Table Data Display:

The tableData method retrieves auction item details from the database and populates the table1 component for display.

1. Timer Management:

The startTimer method initializes a timer that counts down the remaining time for the auction. It updates the timerLabel accordingly.

1. Item Selection:

Clicking on a row in the table1 component selects the corresponding item and displays its details in the input fields.

1. Image Resizing:

The resize method resizes images to fit within the GUI for display.

1. Item Sold Check:

The isItemSold method checks if an item is already sold by querying the database for its sold status.

Usage:

Administrators utilize this module to manage auction items, start auctions, and monitor auction status within the online auctioning system.

Integration:

The Admin module integrates with other system modules to provide a comprehensive administrative interface. It interacts with the Customer module for bid processing and with the database module for data storage and retrieval.

Dependencies:

Dependent on javax.swing and java.sql packages for GUI implementation and database interaction, respectively.

Enhancements:

Implement user authentication to restrict access to administrative functionalities. Improve error handling for database operations and user inputs. Enhance GUI design for improved usability and aesthetics.

**Design Patterns and Principles:**

**Design Patterns:**

Observer Pattern:

The project utilizes an event-driven architecture, akin to the Observer Pattern. ActionListener interfaces are employed to handle user interactions, allowing components to subscribe to events and respond accordingly. For instance, when a user clicks a button, registered action listeners are notified, triggering the associated actions.

Factory Method Pattern:

The project showcases the concept of object instantiation and creation through cohesive class structures. Instances of GUI components and database connections are created within appropriate classes, adhering to the Factory Method Pattern's principles of encapsulating object creation.

Model-View-Controller (MVC) Pattern:

The project follows a structured MVC architecture, separating concerns between the presentation layer (View), application logic (Controller), and data model (Model). MainScreen acts as the controller, facilitating user interactions and coordinating between the view and model components.

Singleton Pattern:

The project demonstrates singleton-like behavior in certain contexts. For example, database connection instances could be managed as singletons to ensure only one connection exists throughout the application, optimizing resource utilization and maintaining consistency.

Command Pattern:

The project exhibits elements of the Command Pattern through the use of ActionListener interfaces to encapsulate user actions as objects. Each ActionListener represents a specific command triggered by user input, enabling the execution of diverse actions while promoting decoupling between invokers and receivers.

Strategy Pattern:

The project showcases flexibility in selecting and applying different strategies for event handling and database operations. By utilizing interfaces and polymorphism, classes can switch between alternative implementations, facilitating code reuse and promoting maintainability.

Template Method Pattern:

The project employs a template-like structure for GUI initialization and event handling. For instance, ActionListener interfaces define a template method (actionPerformed) that subclasses override to provide specific implementations, ensuring consistent behavior across different actions while allowing for customization.

**SOLID Principles:**

Single Responsibility Principle (SRP):

While some classes like Admin and Customer handle multiple responsibilities such as GUI initialization and database interaction, they still maintain a degree of cohesion within their respective functionalities.

Open/Closed Principle (OCP):

The project adheres to the OCP. However, the classes can be extended to add new functionalities without directly modifying existing code, suggesting a degree of openness to extension.

Liskov Substitution Principle (LSP):

Subclasses such as Customer and Admin can be used interchangeably with their base classes (JFrame), maintaining the expected behaviors of JFrame.

Interface Segregation Principle (ISP):

The project demonstrates ISP principles. It avoids unnecessary dependencies between classes by utilizing interfaces for event handling (ActionListener) and database operations (ResultSetMetaData).

Dependency Inversion Principle (DIP):

The project demonstrates a degree of dependency inversion by relying on interfaces and abstract classes (e.g., ActionListener) to decouple higher-level modules from specific implementations.

**GRASP Principles:**

Creator:

The MainScreen class serves as a creator for instances of Customer and Admin classes, aggregating the initialization of these objects.

Controller:

Admin and Customer classes act as controllers, handling user interactions and coordinating the flow of control within their respective functionalities.

Information Expert:

Admin and Customer classes contain relevant information and behaviors related to their responsibilities, acting as information experts within the system.

High Cohesion:

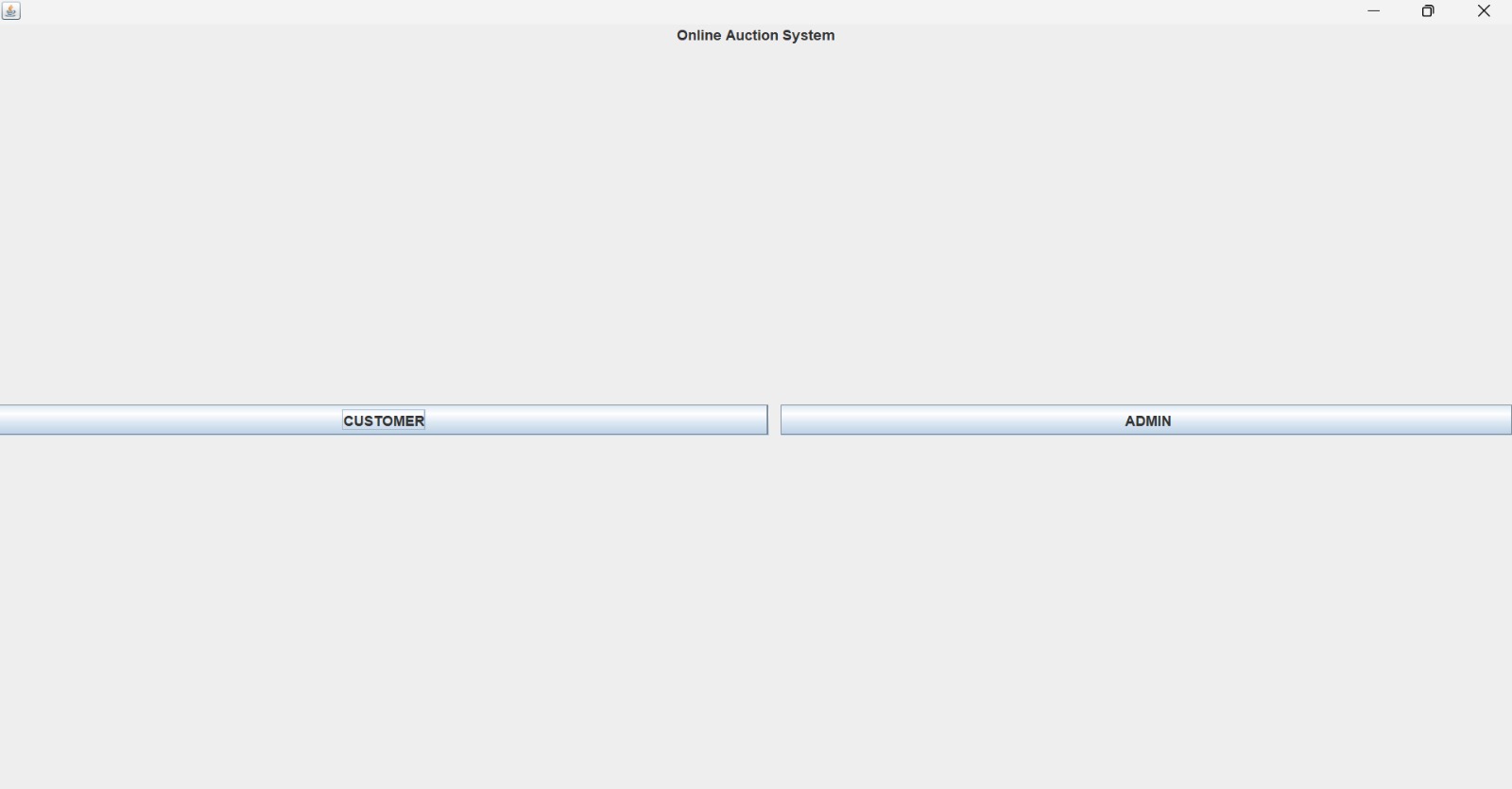
Methods within Admin and Customer classes exhibit high cohesion, focusing on related functionalities such as GUI initialization and database interaction.

Low Coupling:

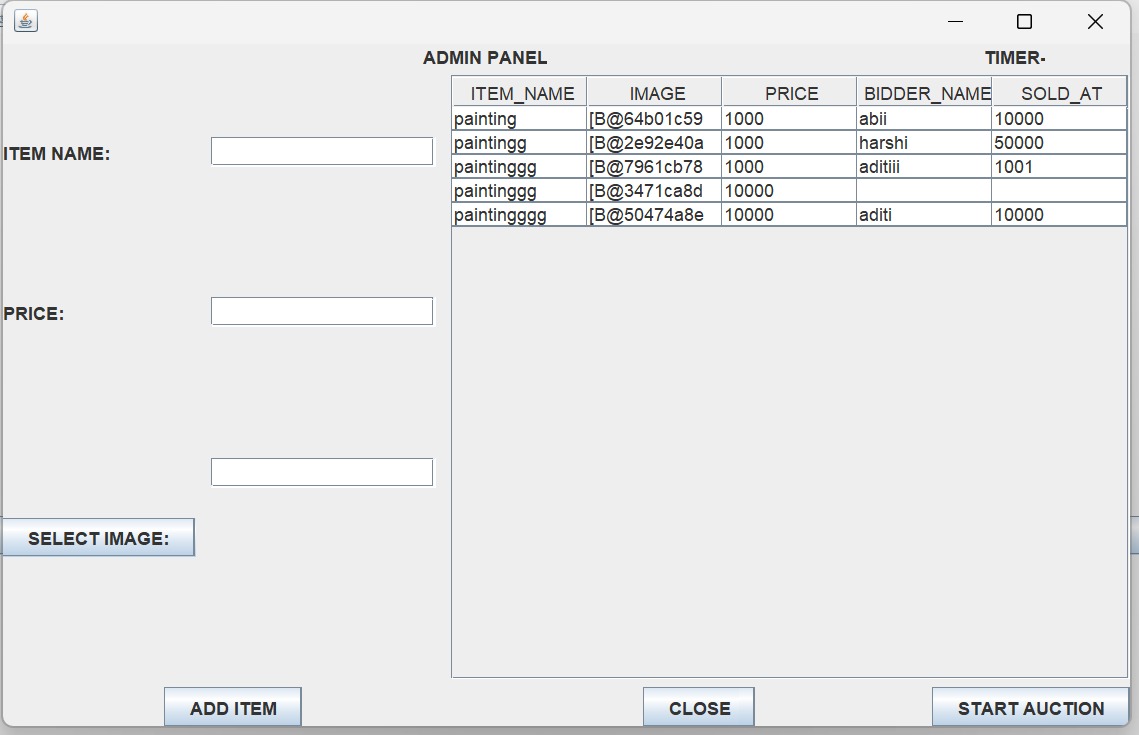
While there are dependencies between classes, efforts are made to minimize direct dependencies on concrete implementations, promoting flexibility and maintainability within the codebase.

**Output:**

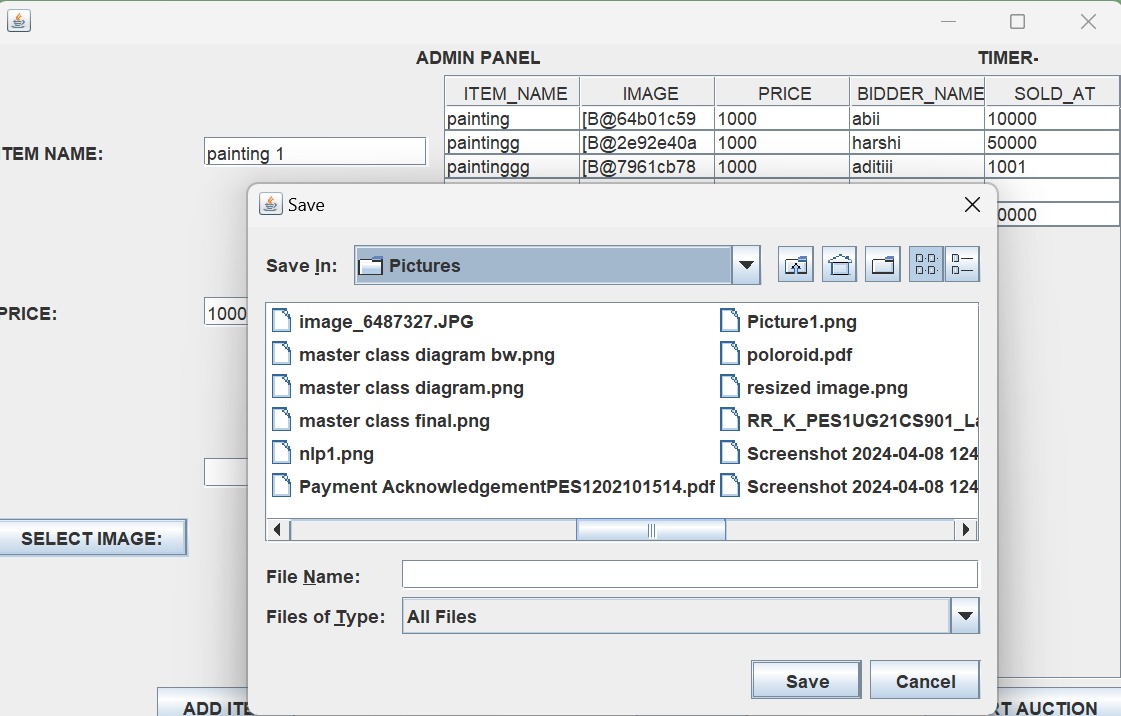
Main Home Screen:

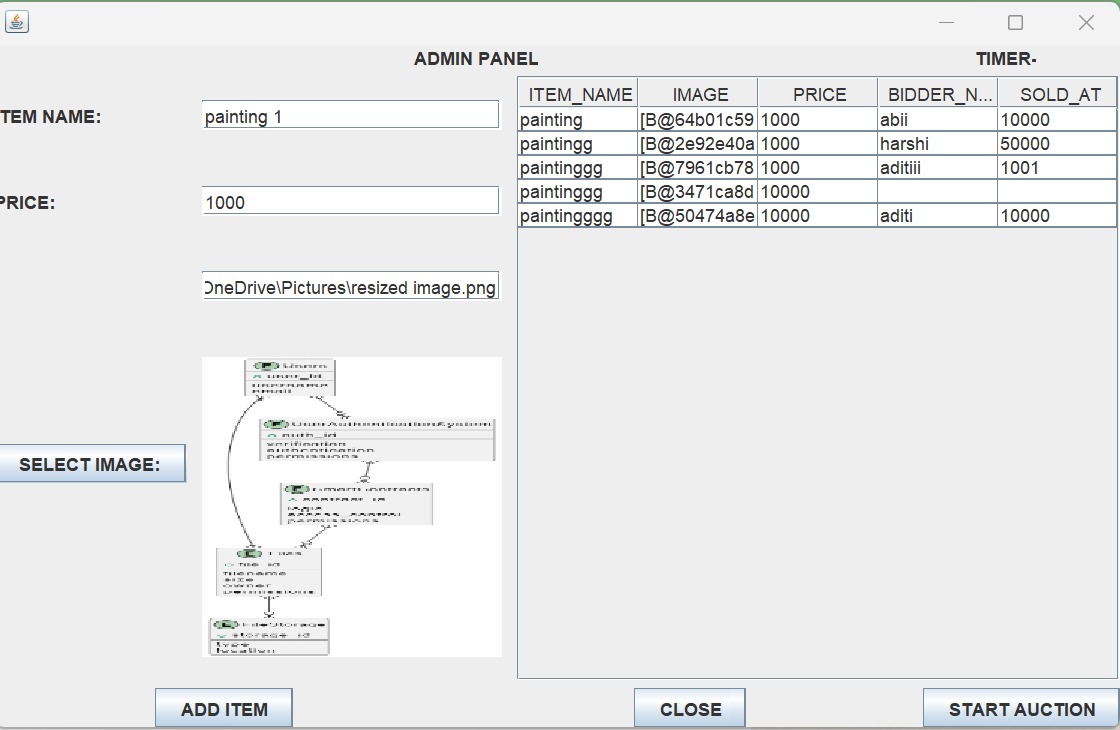
****

Admin Panel:

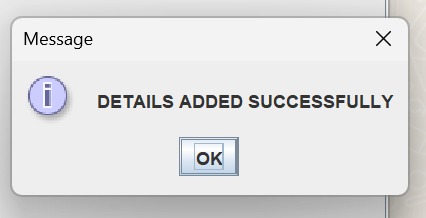


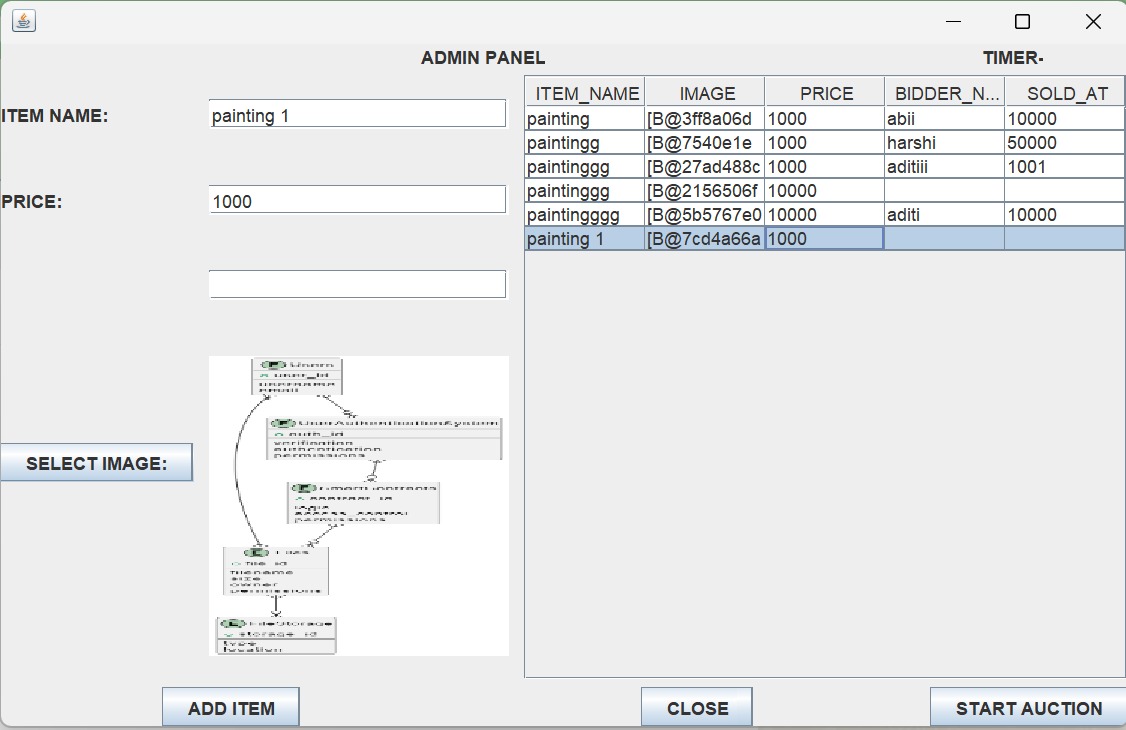
Setting up Product for Auction:



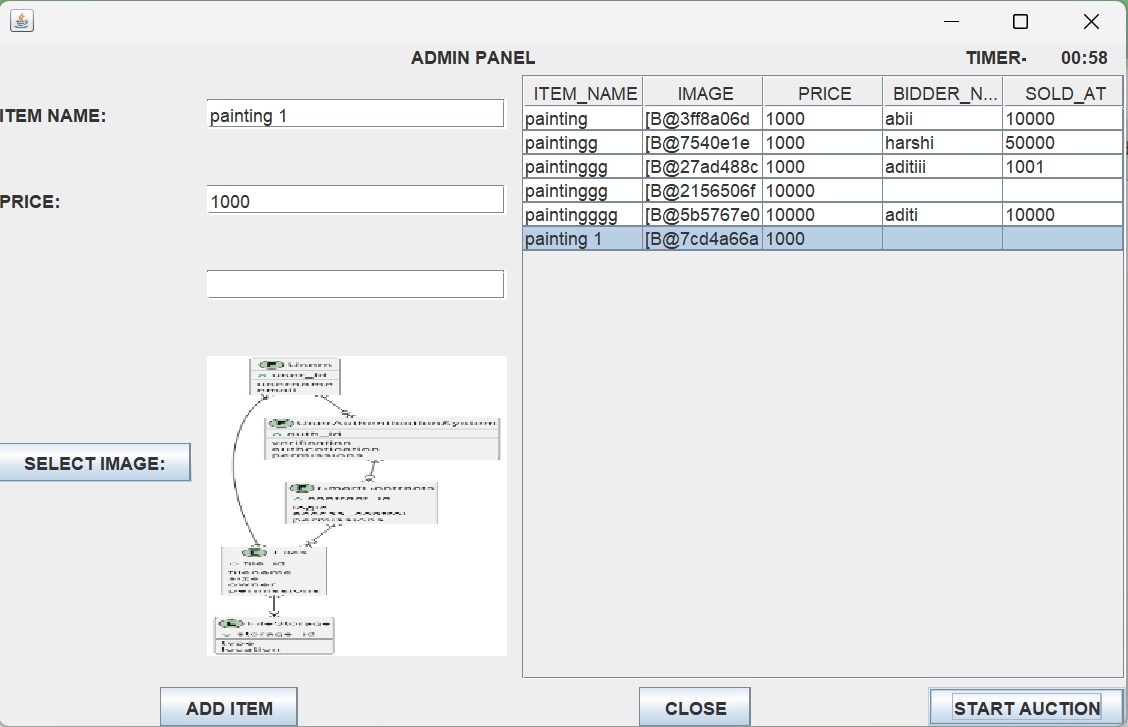


Product successfully added for Auctining:

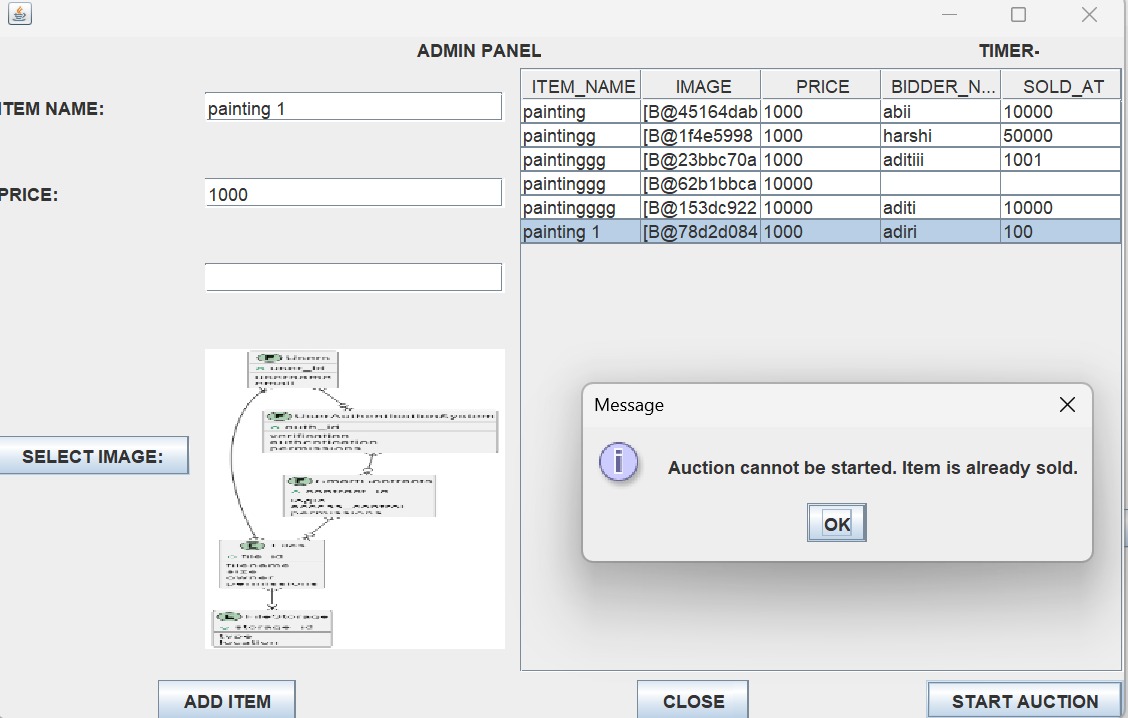




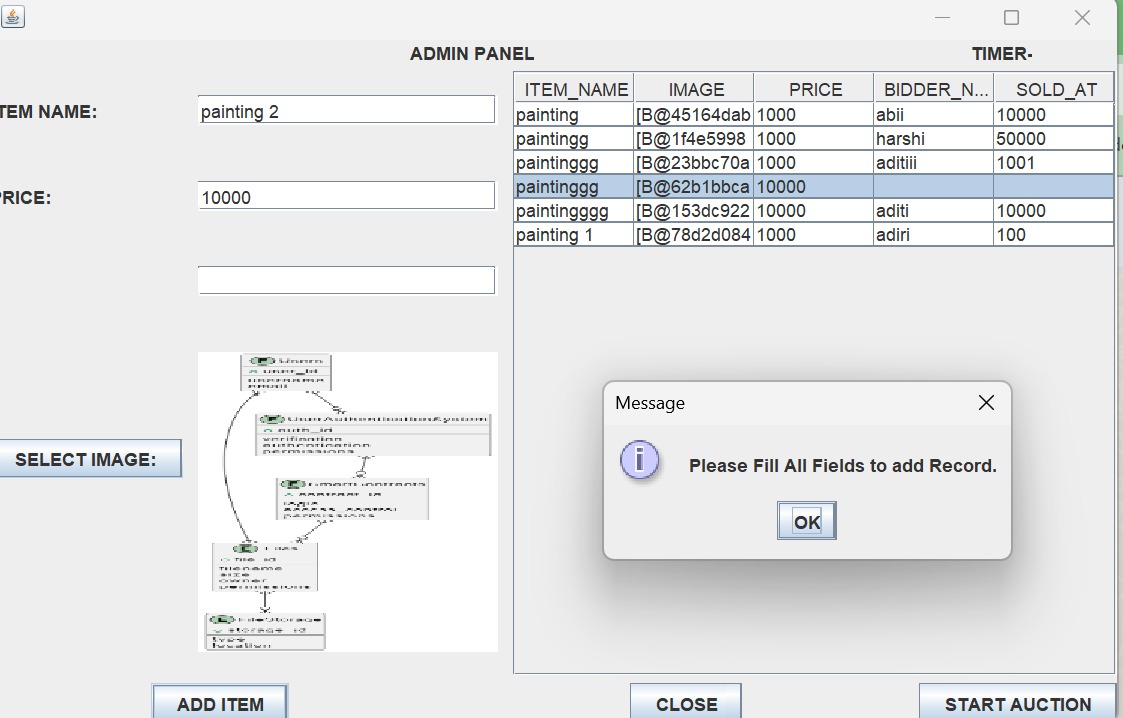
Auction started and Timer Running:



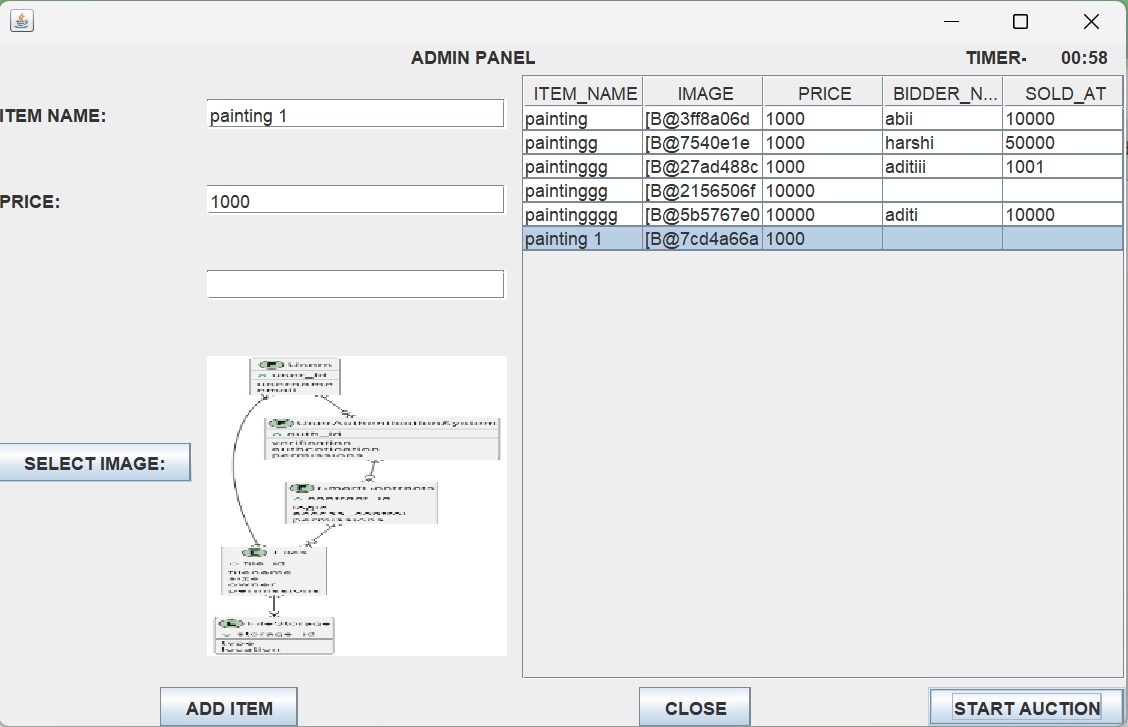
Constraint to prevent Re-Auctioning of Sold Products:



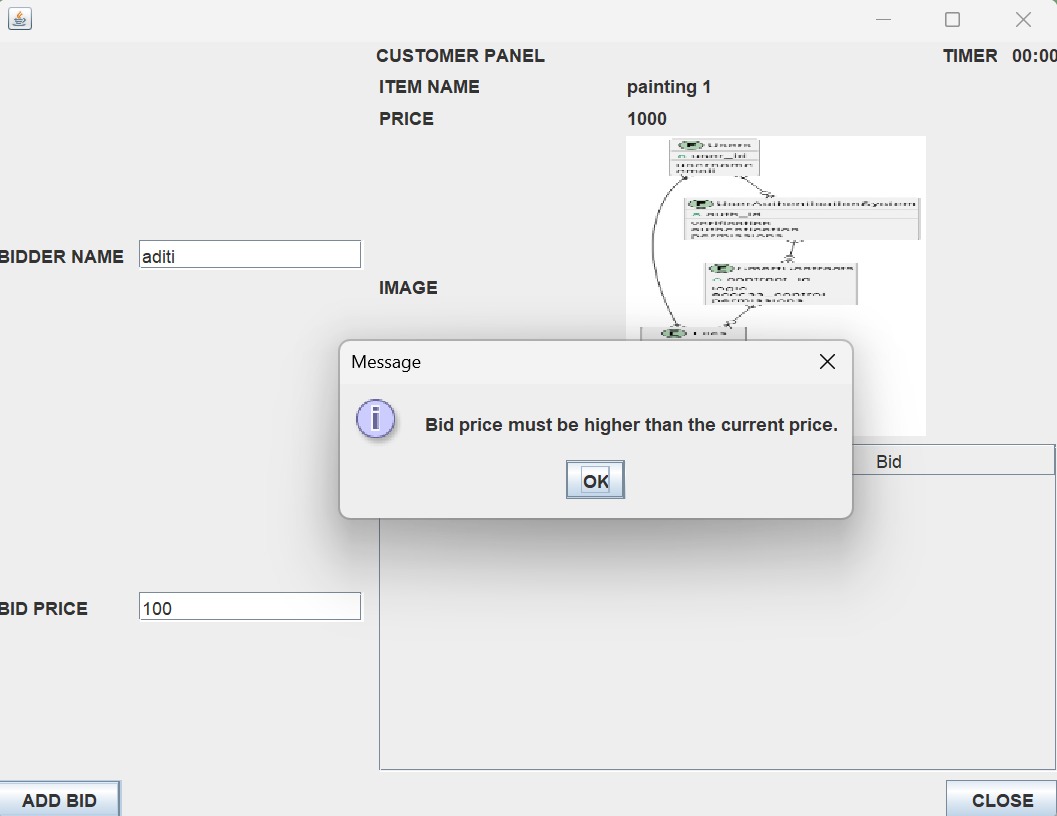
Constraint to prevent incomplete Item for Auctioning:



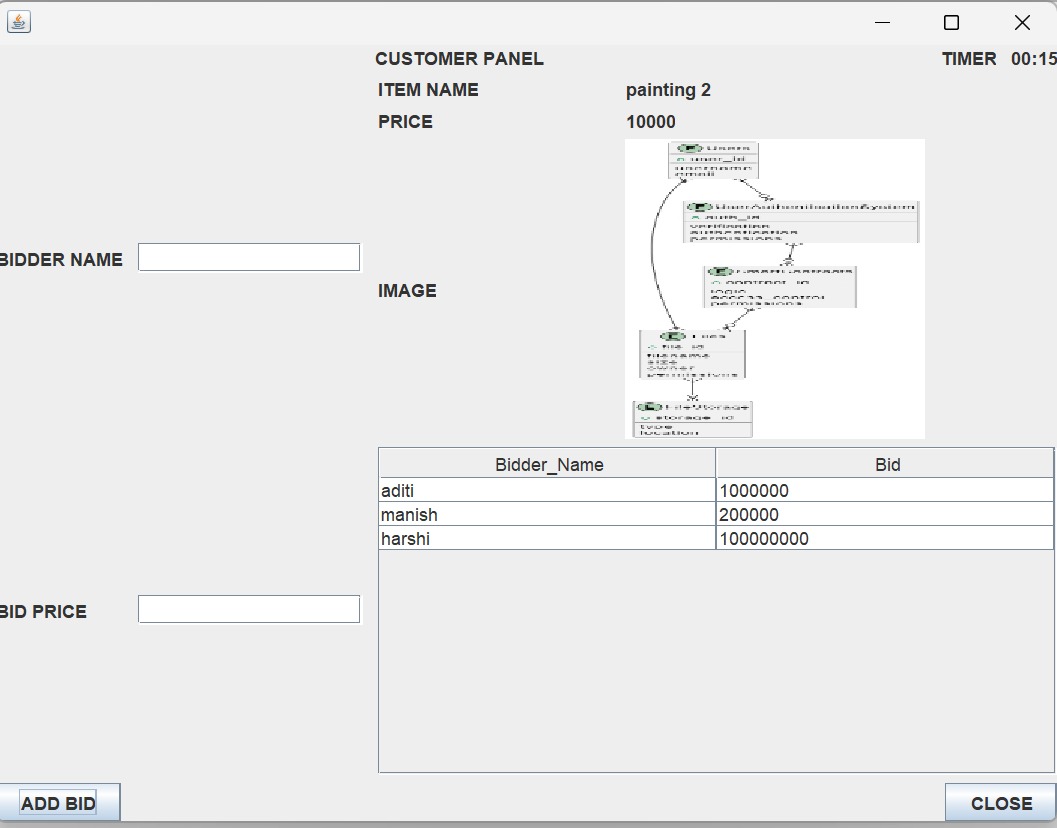
Customer Panel:



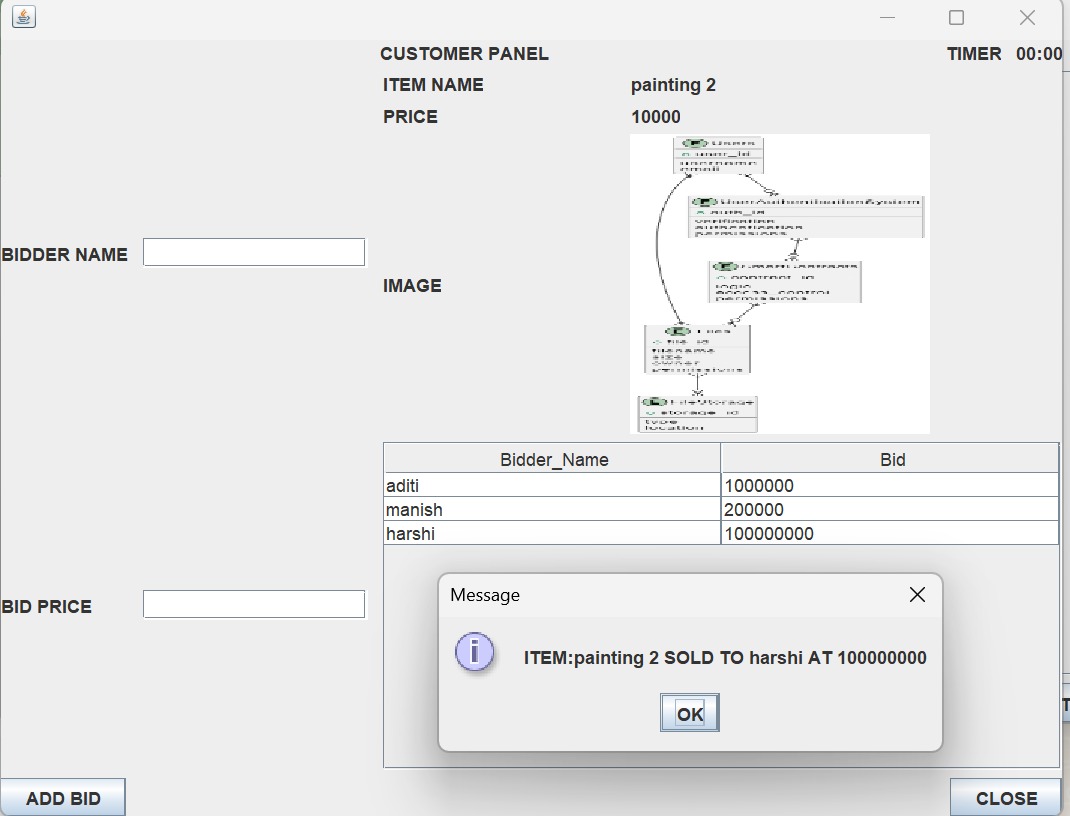
Buyer Bidding and Constraints:



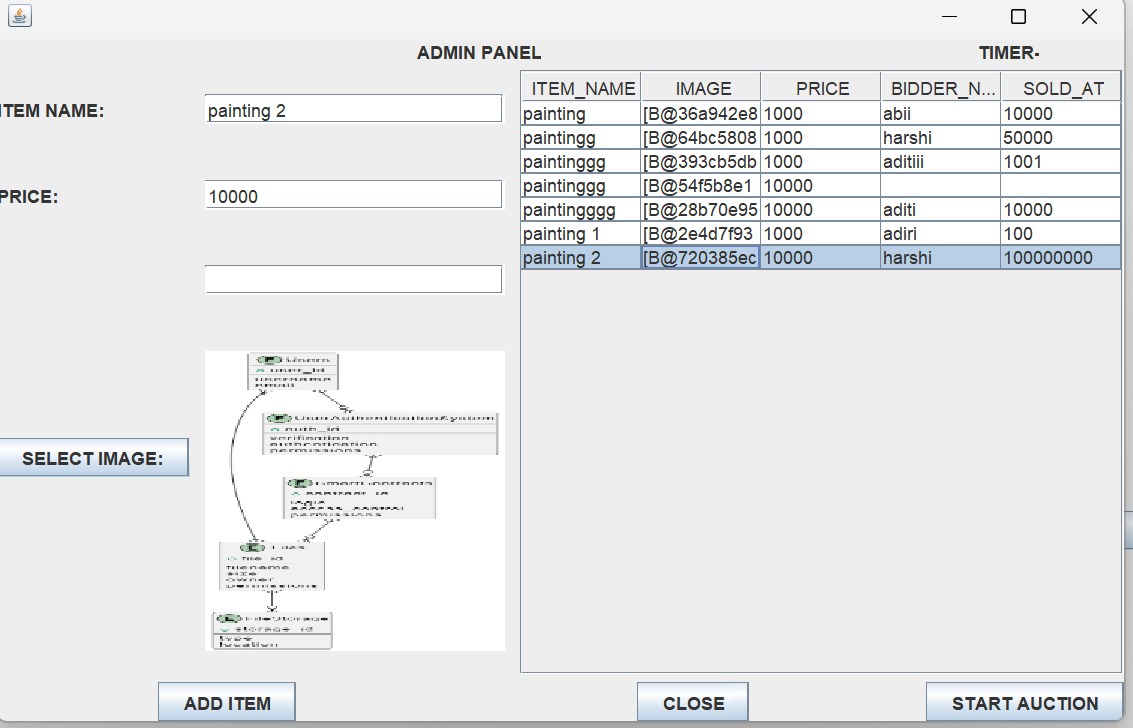
Viewing All Auction Bidders:



Auction Over and Product Marked as Sold:



Admin viewing Product as Sold:



**Contributions:**

Team Member 1: Sri Harshitha A

Frontend Focus:

Designed and implemented the user interface using Swing, ensuring a visually appealing and intuitive design.

Optimized GUI components for responsiveness and consistency across various screens.

User Experience Enhancement:

Conducted user testing sessions to gather feedback and iteratively improve the GUI design based on user preferences and behavior.

Team Member 2: Aditi M

Backend Expertise:

Developed backend logic and functionalities, including auction management, bid processing, and user authentication.

Implemented robust error handling and exception management to ensure system stability under various conditions.

Database Management:

Designed and optimized database schemas to efficiently store and retrieve auction data, considering scalability and performance.

Team Member 3: Srimanish

Architectural Design:

Defined the overall architecture of the Online Auctioning System, selecting appropriate design patterns and principles.

Established clear interfaces and communication protocols between frontend and backend components to facilitate seamless integration.

Technical Guidance:

Provided technical guidance and mentoring to team members on architectural best practices and design decisions.

Team Member 4: Akibbhai

Integration Management:

Oversaw code integration from multiple team members, ensuring consistency and compatibility across different modules.

Managed version control systems to track changes, merge branches, and resolve conflicts effectively.

Dependency Management:

Managed project dependencies, including libraries, frameworks, and external APIs, to ensure smooth integration and minimize conflicts.

Stayed updated with industry best practices and emerging technologies, recommending relevant tools and libraries to enhance project development.

**GitHub Repository for the project:**