VIA University College

Bidhub ~ Project Report

Group 7

Aleksandra Ignatova-Habibulina (293687)

Florina-Narcisa Mitigus (344692)

Mathias Rujan (344681)

Mariia Moskovko (344684)

Matej Palas (344682)

Software Technology Engineering, 2nd semester

Supervisors: Steffen Vissing Andersen, Allan Henriksen

Number of words: 16.475

Number of characters without spaces: 89.025

31.05.2024

Table of contents

Abstract

The primary objective of this project is to enhance the auction experience for both sellers and buyers by developing a user-friendly and efficient multi-user auction application. This platform addresses common challenges inherent in traditional auctions, including issues of transparency, accessibility, and security, by leveraging modern digital technologies to create a robust and scalable solution.

To achieve these goals, the technical choices for this project were informed by the latest industry standards and the academic guidance provided by VIA University College. The development stack includes Java for backend processing and PostgreSQL for reliable database management.

Through this comprehensive development approach, a functional auction client-server system was successfully created, which aligns with the defined project goals. The platform ensures that users can participate in auctions efficiently and securely, offering a transparent bidding process and broad accessibility. This project demonstrates the effective application of modern software development practices and technologies in addressing real-world problems, providing a reliable and innovative solution for online auctions.

1. Introduction

Auctions have a rich history dating back thousands of years, providing a fundamental mechanism for the exchange of goods. (Krishna, 2002, p.1). English auction stands as one of the most predominant forms, structured to facilitate efficient and competitive bidding (Krishna, 2002, p.2).

Despite their dynamism and efficiency in facilitating commerce, auctions present challenges for all involved parties (see **Figure 1.1** *Rich picture*).

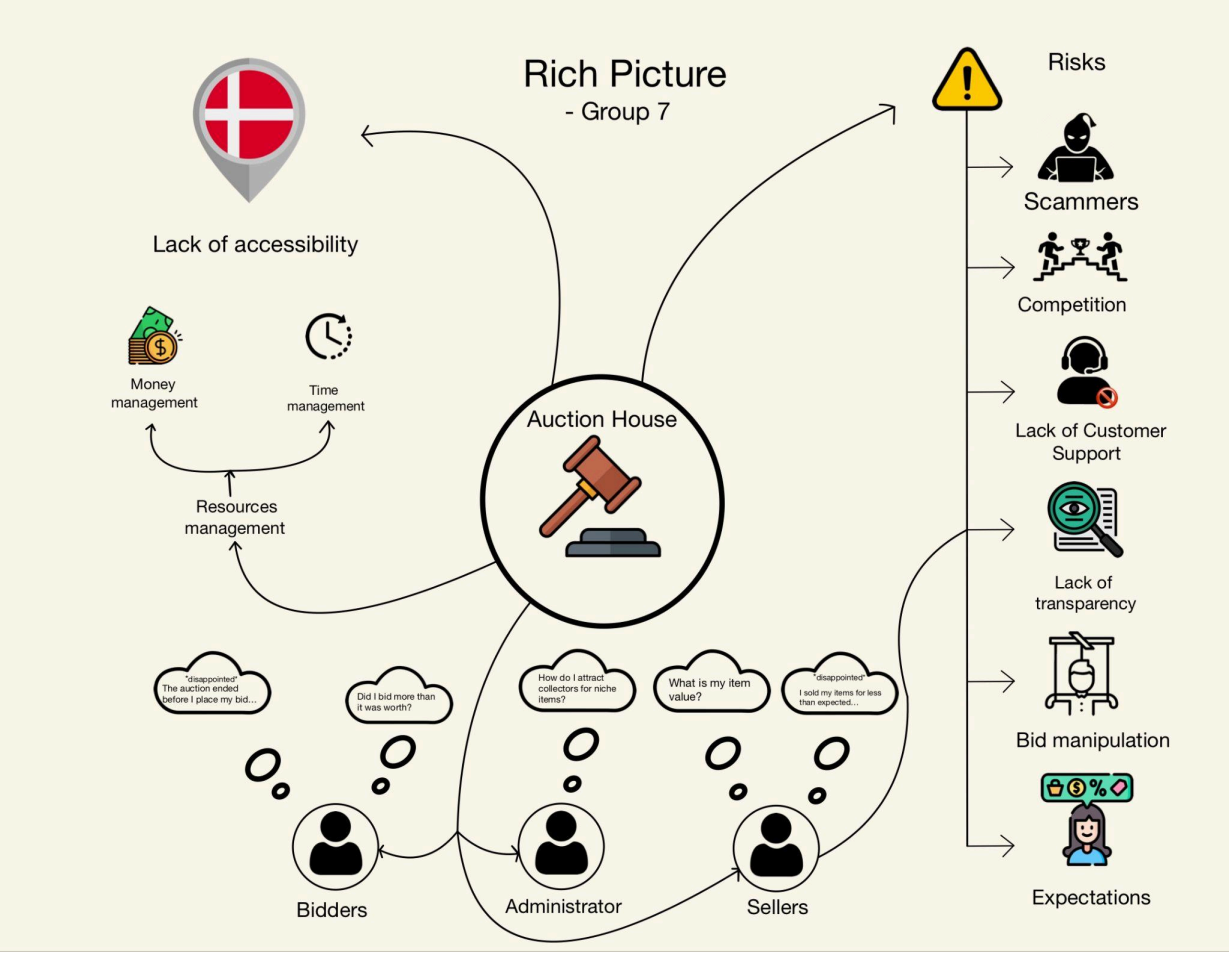


Figure 1.1 *Rich picture*, own production

Sellers may lack accurate evaluations of their items (Krishna, 2002, p.3) or they may have unrealistic expectations about them, leading to disappointment if the goods don't achieve the desired price.

Buyers face the responsibility of thorough item evaluation to avoid a common scenario of overpayment compared to alternative purchase methods (Davies et al.,

2017). Additionally, they have to supervise auctions closely and be available to place bids, especially in the final moments of an auction, when bidding activity tends to escalate (Federal Communications Commission, 2017).

The auction house owners must make sure that the items sold meet the standards and descriptions, so their reputation will not be affected. They must also be aware of the existence of scammers, sellers or bidders previously associated with fraudulent strategies during the auctions and take measures to enhance security and limit harmful practices, (Kas et al., 2023), by observing participants' activity, prohibiting participation, managing the items or informing the other participants about the lack of seriousness.

By addressing the inefficiencies of traditional auctions and leveraging modern digital tools, this project aims to create a superior auction experience that maximizes value for all stakeholders involved - auction moderators, sellers and potential buyers.

Traditional auctions face several challenges and they can be improved by allowing a better flow of information and more flexibility in terms of spatial-temporal requirements, resulting in a superior experience for participants, who are expecting fairness and accessibility.

The project delimits its scope by not including a payment system within the application, but allowing the participants to receive each other's contact information and close the deals independently. The platform will not implement advanced security measures beyond basic authentication with a password nor item evaluation services and it will not assume responsibility for any disappointing deals. The application will serve as a medium for connecting buyers and sellers, but the actual transactions and agreements reached between parties will be solely their responsibility and therefore, the project will not implement any return policy.

In order to solve any inefficiencies, the project was divided into several parts: analysis, design, implementation and testing, each of them playing a major role in the development of this project.

2. Analysis

A critical examination was conducted in order to develop a client-server system that enhanced the auction experience for all stakeholders. This system is designed to address prevalent challenges in traditional auction environments and therefore, this analysis aims to provide a comprehensive understanding of the system's architecture and functionality, highlighting how it meets the diverse needs of its users.

The following sections will systematically break down the various components and features of the auction platform, starting with the system's requirements, both functional and non-functional, to establish a clear context for the analysis and a thorough understanding of how the auction client-server system operates, the challenges it overcomes, and the benefits it offers to its users.

2.1. Requirements

Given the longstanding challenges in the auctioneering field, the stakeholders, such as the auction house and the auction participants have diverse needs and expectations, but all their requirements are fairness-oriented and together, they build a better understanding of the system. Six types of actors can be identified in the project and they have specific goals:

- Moderator
  - manages the auction system;
  - is responsible for ensuring that the platform operates smoothly and fairly by overseeing auctions and Participants;
  - has privileges in terms of accessibility
- Seller
  - the auction Participant who assumes the role of Seller by starting an auction after providing information on items;
- Bidder
  - the auction Participant who assumes the role of Bidder by participating in auctions (placing bids, buying items, browsing ongoing auctions);
- Participant
  - general actor, can be seller, bidder or both;
- Timer
  - is responsible for starting, updating and ending auction's countdown and for notifying its end;
- Guest

- someone who visits the auction system and has the option to create an account, transforming them into a Participant;

2.2. Functional Requirements

The functional requirements outlined in this section detail the specific behaviors and capabilities that the auction system must exhibit to meet the needs and expectations of its users. These requirements ensure that the system provides a comprehensive experience, supporting various roles. Each requirement is designed to facilitate seamless interactions within the auction platform, enhance security, and ensure regulatory compliance. The requirements are articulated to be Specific, Measurable, Achievable, Relevant, and Time-bound (SMART), serving as the foundational elements for the system's design and implementation. By adhering to these requirements, the auction system is well-equipped to handle the core functionalities necessary for efficient auction management and user satisfaction.

Functional requirements:

1. As a Seller, I want to start an auction deal that has an auto generated ID, upload a picture, fill out the information about my item (title, description), apply price constraints (reserve price - the minimum price I am willing to accept, buyout price - the price I am willing to sell the item right away for, minimum bid increment) and set the auction duration in hours (time left for participating from the moment I started the auction), so I will be able to list my items and sell them within the specified auction duration.

2. As a Bidder, I want to access the list of all ongoing auctions, showing the ID, title, current highest bid, picture and end time for each auction, so I can easily identify and participate in the auctions that interest me.

3. As a Bidder and Moderator, I want to open an auction and have access to all the necessary information about the item, including the title, description, reserve price, buyout price, minimum bid increment, auction time, picture, current highest bid and current highest bidder, so I can make informed decisions and effectively participate in the auction.

4. As a Bidder, I want to participate in an ongoing auction by offering at least the reserve price, if no other bids exist or increasing the highest bid (if I am not the current highest bidder) by at least the minimum bid increment, so I can compete for an item I am interested in, without outbidding myself.

5. As a Bidder, I want to receive a real-time notification with the auction ID when my bid is beaten for a specific item, so I can promptly increase my price in response and have a better chance of winning the auction

6. As a Participant, I want to access the chronological list of notifications, starting with the most recent ones, which includes timestamps and notification contents, allowing me to stay informed about relevant updates and actions, such as won items, deleted auctions or beaten bids.

7. As a Guest, I want to create an account by providing my first name, last name, date of birth, contact information (email address, phone number) and password so I can start to securely sell my items and participate in auctions.

8. As a Participant, I want to log in to my account using my email address and password, so that I can securely access my notifications, sell items, and participate in auctions at any time.

9. As a Bidder, I want to buy an auction item for the specified buyout price if no bids have been placed on it within a specified time period, so that I can secure the item and skip the bidding process.

10. As a Seller, I want to access all my ongoing and closed auctions, and see the ID, title, the highest bid, picture and the end time for each auction, so I will stay up to date with the status of my items.

11. As a Bidder and Seller, I want to be notified and receive the other party's contact information (including first name, last name, email address and phone number) and the auction ID after the auction time is over, so that we can communicate and close the deal independently.

Bidder - the Bidder with the highest bid after the auction end

Seller - the creator of the auction

12. As a Bidder and Moderator, I want to search for ongoing auctions using the IDs or words present in the titles, so I can easily find and access the items I'm interested in.

13. As a Bidder, I want to access all the ongoing and closed auctions where I previously placed a bid or bought the item and see the ID, title, the highest bid, picture and the end time for each auction, so I will stay up to date with information on deals I'm taking or took part in.

14. As a Participant and Moderator, I want to log out from my account, in order to prevent unauthorized access such as someone else using my device.

15. As the Moderator, I want to prohibit sellers from placing bids on their own auctions, to prevent them from artificially increasing the prices for the items.

16. As a Participant and Moderator, I want to reset my password by entering the old password and the new password, if I suspect unauthorized access or I want to choose a stronger password, so my account will be secured.

17. As the Moderator, I want to log in to my account using a provided email address and password, so that I can inspect Participants' activity and manage the auctions.

18. As the Moderator, I want to set and display my contact information (including first name, last name, email address and phone number), so the Participants can easily contact me for assistance with issues or navigating the platform.

19. As the Moderator, I want to access all the ongoing and closed auctions, including the ID, title, highest bid, picture and end time for each auction, in order to effectively evaluate whether they comply with the rules and standards, based on my determination.

20. As the Moderator, I want to search for a Participant by their email address, so I can see their first name, last name, phone number, ban or unban them.

21. As the Moderator, I want to ban a Participant, provide a reason for banning, and have the reason displayed to the Participant when they attempt to log in and get the system to automatically log them out, delete their auctions, regardless of status, and their bids, if I decide that the Participant does not respect the rules, based on my determination.

22. As the Moderator, I want to unban and allow a previously banned Participant to log back into their account if I determine that my initial reason for banning them was too weak.

23. As the Moderator, I want to delete an item from auction, regardless of status, by notifying and providing the Seller with a reason, if I decide that according to my determination, the item violates platform's standards.

24. As a Participant, I want to delete my account, and automatically delete my started auctions and my placed bids, regardless of status, if I decide I don't want to use the platform anymore.

25. As the Moderator, I want to access the list of accounts, with their first name, last name, email address and phone number, so I can investigate their activity and decide if they respect the rules.

26. As the Moderator, I want each auction to be open for at most 24 hours, so they will not be open for too long.

27. As a Participant, I want to edit the information in my account, including the first name, last name, email address, phone number and date of birth, to update it if the data is no longer actual or if I entered the information when I created my account.

2.3. Non-Functional Requirements

The following non-functional requirements (see **Table 2.3.1**) define the system's operation and ensure its usability, reliability, performance and supportability, providing the foundation for a robust and efficient auction platform.

No.	Summary	Non-Functional requirement	FURPS+ classification
28.	Age Restriction	Participants must be at least 18 years old to create an account.	Usability - ensures the user base is age-appropriate for the legal constraints
29.	Countdown Timer	A countdown timer must be used to specify the time left for each ongoing auction in the format HH:MM:SS and close the auction when the time is up.	Reliability - the system is required to perform correctly under specific conditions (ending the auction precisely when the time is up)
30.	Database Storage	The system must use a database (PostgreSQL) as storage strategy.	Performance - the choice of database impacts system performance in terms of data retrieval and storage efficiency
31.	Image Support	The system must support JPEG images when starting an auction.	Supportability - ensures that the system is compatible with widely used standards and formats

Table 2.3.1 *Non-functional requirements*

2.4. Use cases

The identified functional requirements are grouped in the use cases, depicted in **Figure 2.4.1** *Use case diagram*. This diagram provides a visual representation of the relationships and dependencies between the actors and different use cases, offering a clear overview of the system's functionality.

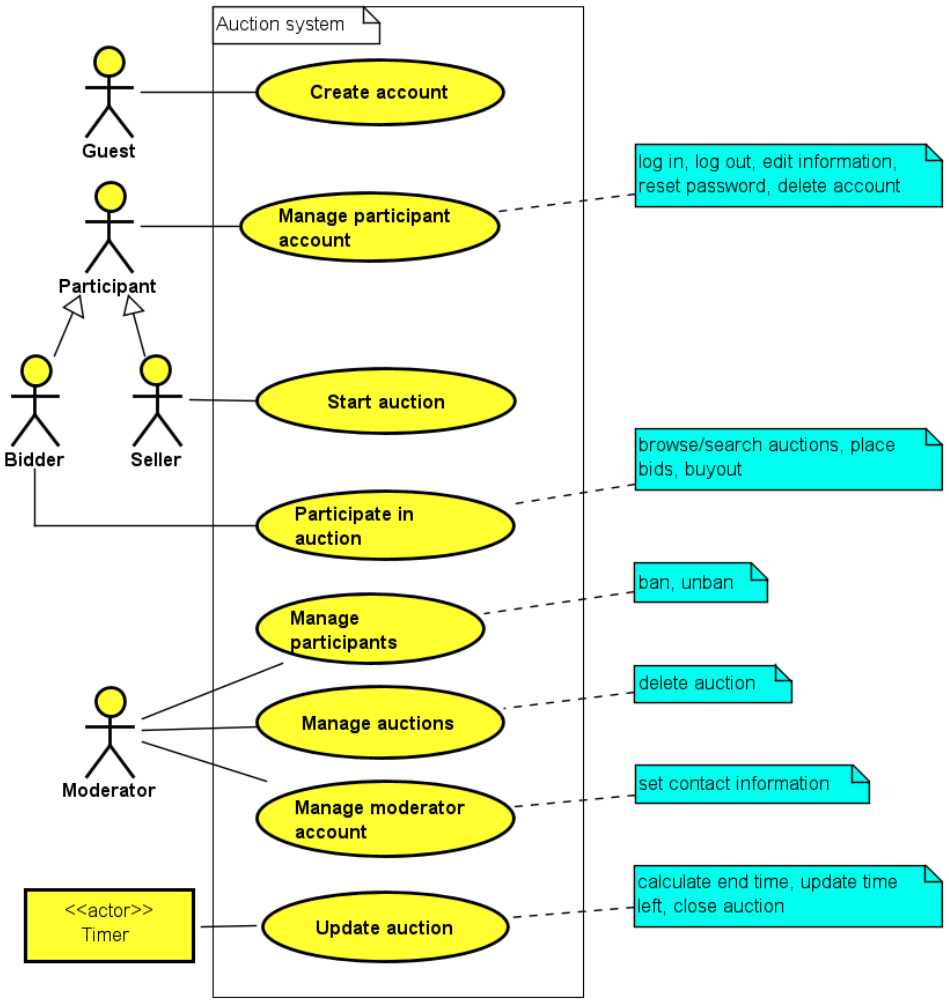


Figure 2.4.1 *Use case diagram*, own production

Use Cases:

1. Start auction (Actor - Seller)

- allows the Seller to:
    - initiate auctions by providing details such as title, description, reserve price, buyout price, minimum bid increment, auction duration and an image.

2. Participate in auction (Actor - Bidder)

- the Bidder can:

    - browse ongoing auctions;
    - search for ongoing auctions by id or words present in the title;
    - place bids in auctions;
    - buy items at the buyout price;
    - receive notifications when their previous bids have been beaten.

3. Create account (Actor - Guest)

- allows the Guest to:
    - create an account by providing personal information: first name, last name, email address, phone number and date of birth as well as a password.

4. Manage participant account (Actor - Participant)

- the Participant can:

    - log in and log out of their account;
    - reset their password;
    - edit or delete their account;
    - access the auctions they started or the auctions they participated in.

5. Manage participants (Actor - Moderator)

- the Moderator can:

    - ban a Participant after specifying the reason;
    - unban a banned participant;
    - view a list of all participants' contact information;
    - search for participants by email;

6. Manage auctions (Actor - Moderator)

- the Moderator can:

- browse both ongoing and finished auctions;
- search for any auction by id or words present in the title;
- delete auctions if they violate rules.

7. Manage moderator account (Actor - Moderator)

- the Moderator is allowed to:
  - set and display their own contact information for all participants;
  - log in and out of their account;
  - reset their password.

7. Update auction (Actor - Timer):

- the Timer:
  - calculates the moment when a started auction should end;
  - displays and updates the time left for bidders to participate in a specific auction.
  - notifies the end of the auction, which triggers the information exchange between the Seller and the final Bidder, if they exist;

Each use case can be traced back to the requirements:

Use Case	**Covered Requirements**
Start auction	1, 26
Participate in auction	2, 3, 4, 5, 9, 12, 15
Create account	7
Manage participant account	6, 8, 10, 13, 14, 16, 24, 27
Manage participants	20, 21, 22, 25
Manage auctions	3, 12, 19, 23
Manage moderator account	14, 16, 17, 18
Update auction	1, 3, 9, 11

Table 2.4.2 *Relation between use cases and requirements,* own production

The “Participate in auction” use case and its scenarios will be analyzed further in the following pages, to provide a detailed explanation of one representative use

case in the main body of the document (see **Table 2.4.3**), while the “Start auction” and the remaining use cases are documented in **Appendix A**.

<table>
<tr><td>Use case</td><td>Participate in auction</td></tr>
<tr><td>Summary</td><td>The actor can browse and search auctions they are interested in, buy or place bids on other’s items.</td></tr>
<tr><td>Actor</td><td>Bidder</td></tr>
<tr><td>Precondition</td><td>The actor must be logged in.</td></tr>
<tr><td>Postcondition</td><td>The actor found the auctions they were looking for, bought away items or placed bids for auctions.</td></tr>
<tr><td>Base sequence</td><td>
1. System displays all ongoing auctions, with ID, Title, Highest bid, Picture and auction End time.
2. IF SEARCH, go to Scenario A SEARCH, Step A1.
3. Select an auction.
4. System displays the selected auction and all its details: ID, Title, Description, Reserve price, Buyout price, Minimum bid increment, Picture, Highest bidder, Highest bid and auction Time left for participating.
5. If BID , go to Scenario B PLACE BID, Step B1.
6. If BUYOUT, go to Scenario C BUYOUT, Step C1.

<table>
<tr><td>Scenario A SEARCH</td><td>A1. Enter ID or at least one word for searching through the auction Titles.
A2. System displays all the ongoing auctions that contain the specified word in their Title or ID, including the ID, Picture, Title, Highest bid and the End time. Go to Step 3. [ALT1]</td></tr>
</table>

<table>
<tr><td>Scenario B PLACE BID</td><td>B1. Enter bid value
B2. Confirm the bid placing. [ALT2]
B3. System checks auction status. [ALT3][ALT4][ALT5]
B4. System validates the value [ALT6][ALT7]
B5. If the auction has a Highest bidder, System notifies them that their bid has been beaten, including the auction ID in the notification. Go to Step 4.
B6. System updates the Highest bidder to the actor's Email address and the Highest bid to their bid value.</td></tr>
</table>
</td></tr>
</table>

	Scenario C BUYOUT C1. Confirm the buyout. **[ALT8]** C2. System checks auction status. **[ALT3][ALT5] [ALT9]** C3. System updates the Highest bidder to the actor's Email address and the Highest bid to the Buyout price. C4. System marks the auction as closed.
Alternate sequence	**[*ALT0]** Processes can be canceled at any point with user interaction. The unconfirmed information will not be saved. Use case ends. **[ALT1]** If there are no auctions found searching by ID or by words in titles, no auctions are displayed. Go to Step 1. **[ALT2]** If the entered value is not confirmed, the bid is not placed. Go to Step 4. **[ALT3]** If the auction is not ongoing, the bid is not placed and the System shows an error message. Go to Step 4. **[ALT4]** If the actor is the already existing Highest bidder, the bid is not placed and the System shows an error message. Go to Step 4. **[ALT5]** If the actor is the Seller (auction's creator), System displays an error message. Go to Step 4. **[ALT6]** If the auction already has a Highest bid and the entered bid value is not greater than the sum of the Highest bid and the Minimum bid increment, the bid is not placed and the System shows an error message. Go to Step B1. **[ALT7]** If there are no bids and the value is not at least equal to the Reserve price, the System shows an error message. Go to Step B1. **[ALT8]** If the buyout is not confirmed, the item is not bought away. Go to Step 4. **[ALT9]** If there is at least one bid placed for the auction, System displays an error message. Go to Step 4.
Notes	This use case covers Requirements 2, 3, 4, 5, 9, 12, 15.

Table 2.4.3 *"Participate in auction" use case description,* own production

The "Participate in auction" use case description explains step by step how the actor can interact with the system, which handles the actor's input. **Figure 2.4.4** illustrates the same steps in a more dynamic manner, showcasing the system's behavior in each scenario and in case of invalid data.

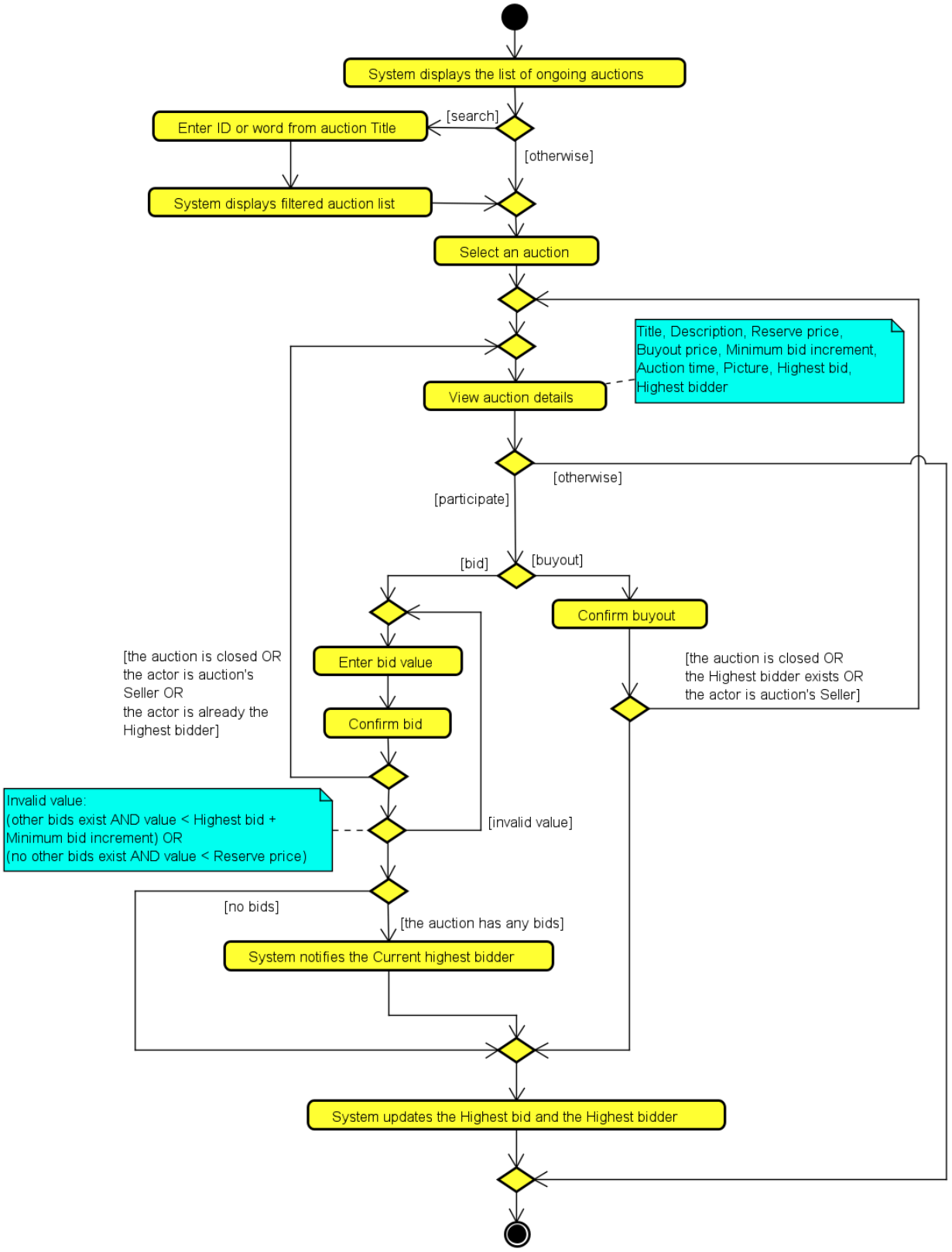


Figure 2.4.4 *Activity diagram for the "Participate in auction" use case,* own production

The activity diagram illustrates the sequential flow and decision-making process involved in participating in an auction. It ensures that all necessary conditions are checked, and appropriate actions are taken based on the actor's input and the auction's current status. The process starts with the actor (Bidder) viewing all the ongoing auctions with the option to search for the relevant ones. After selecting an auction, the actor can access detailed information and decide whether or not to participate.

If they participate, the user then decides to either place a bid or buyout the item. When opting for buyout, the item can be purchased immediately, ending the process, if the auction's state allows it. When placing a bid, the system validates the bid value, to ensure it meets the conditions. If the bid is invalid, the actor must re-enter a valid value (see **Figure 2.4.5**).

If the auction has any bids placed on it, the Current highest bidder will be notified that the Current highest bid for the auction has been updated. Similarly, when another user will place a bid for this item, the actor will be notified.

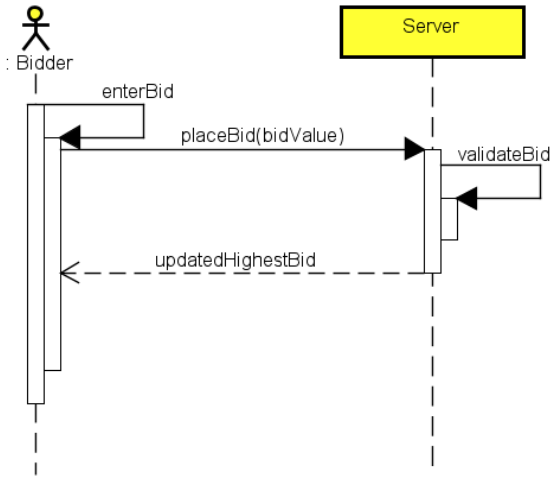


Figure 2.4.5 *System sequence diagram for the "PLACE BID" scenario - one Bidder*, own production

This use case is progressing in parallel along with the "Update auction" use case (see **Appendix A** for the use case description), of which the activity diagram is illustrated in **Figure 2.4.6**.

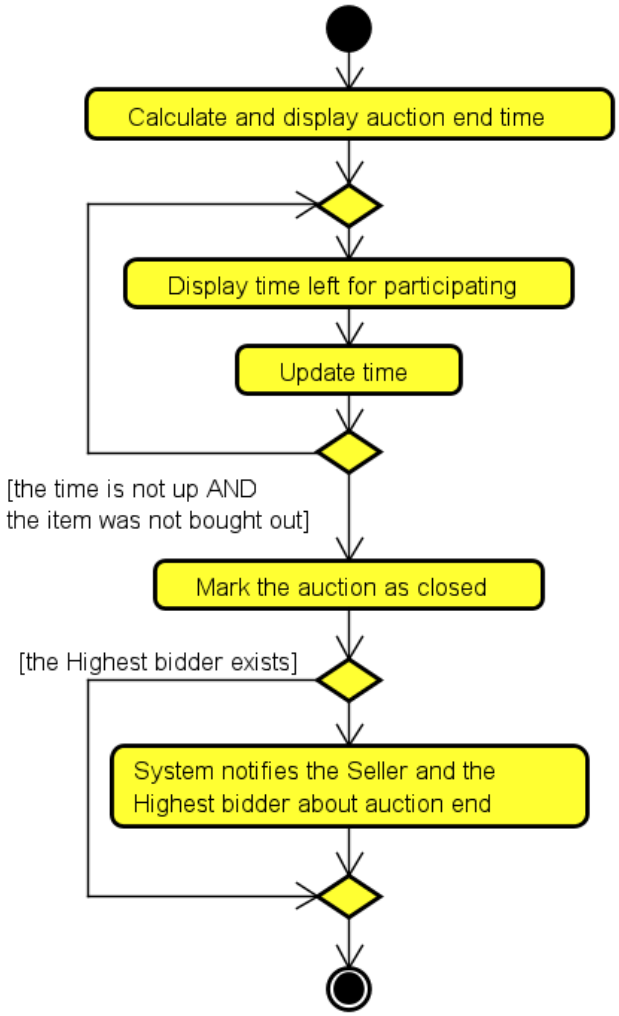


Figure 2.4.6 *Activity diagram for the "Update auction" use case*, own production

The process for the "Update Auction" use case begins with calculating and setting the auction end time, then updating the time left for participation until the auction time expires or someone buys out the item, thus maintaining a loop that ensures the auction runs for its intended duration. When the auction time is finally up, the auction's status is updated and notifications are sent out to relevant participants to inform them of the auction's conclusion. This marks the end of the process.

While the Bidders place bids and potentially buy out items, the Timer is continuously updating the auction time. If the auction time expires or the item is bought out, the system marks the auction as closed, preventing further bids, as shown in **Figure 2.4.7.**

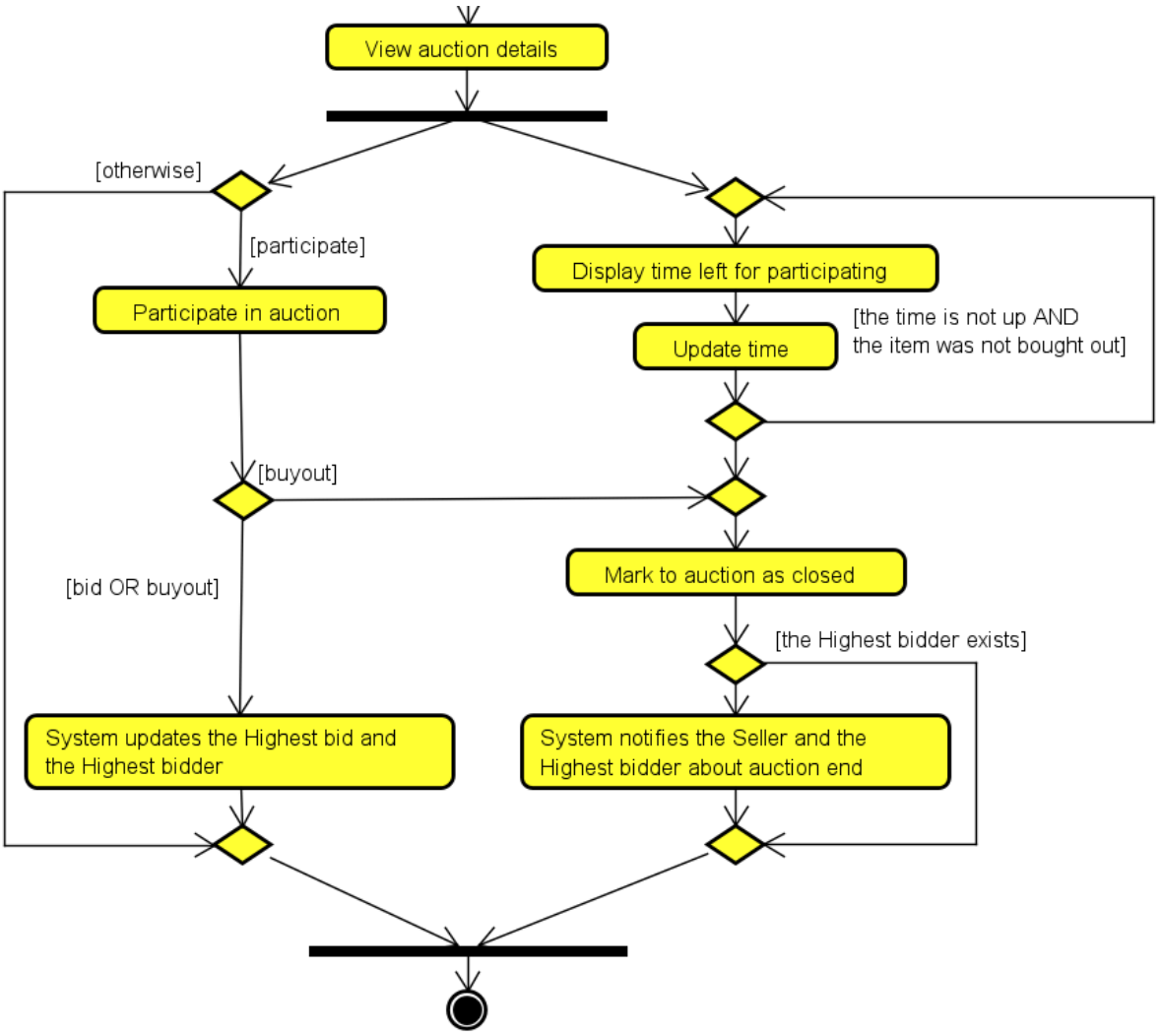


Figure 2.4.7 *Simplified visualization of the "Participate in auction" and "Update auction" use cases combination*, own production

2.5. Domain Model

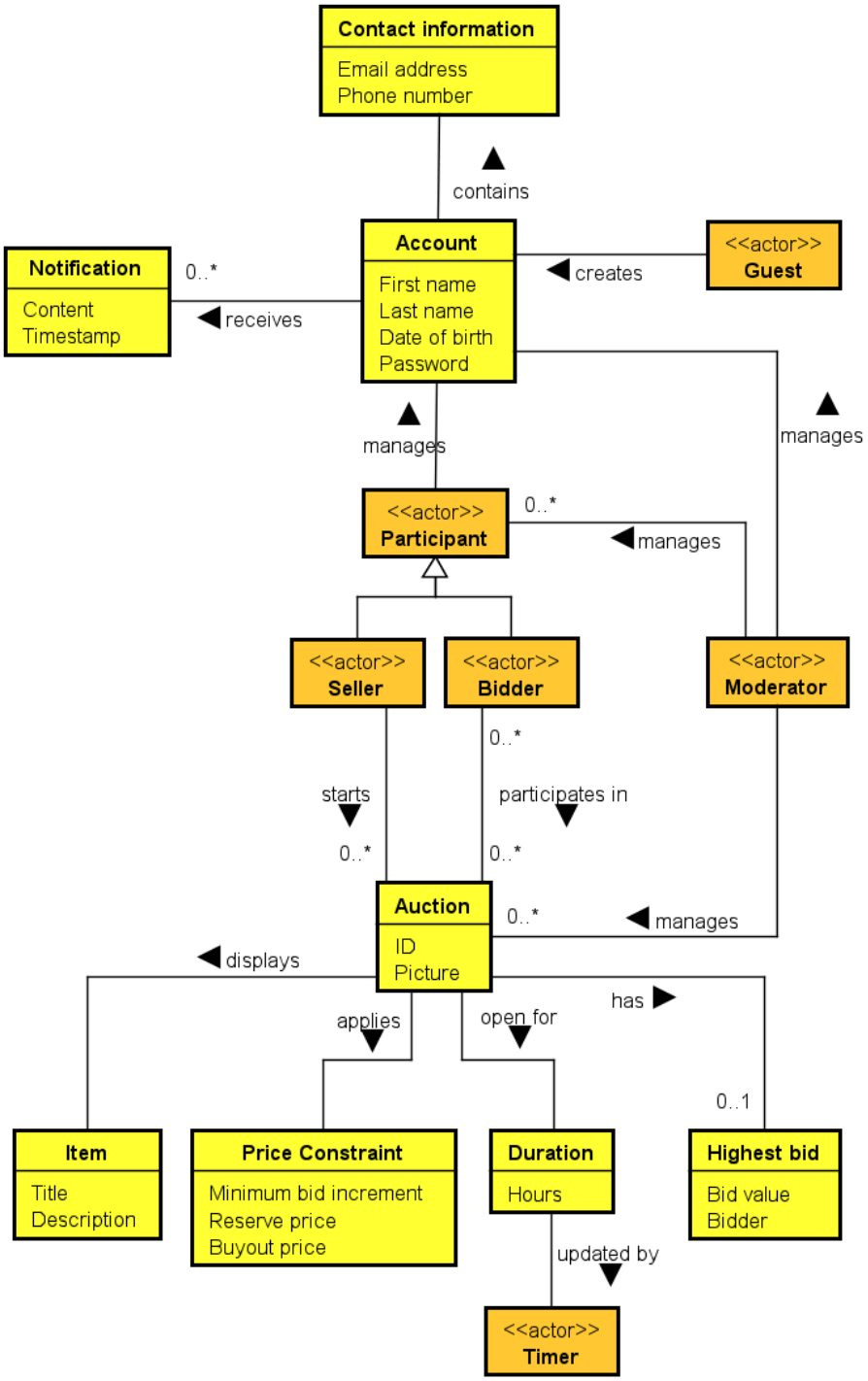


Figure 2.5.1 *Domain model*, own production

In **Figure 2.5.1** *Domain model*, the relationships and attributes of key entities within the auction system are depicted, as they are arising from the functional requirements. The model identifies several core classes, along with the actors: Account, created by a Guest (who assumes now the role of a Participant), which contains Contact information and receives Notifications. The Participant can now manage their account or undertake the position of a Seller and start Auctions or the role of a Bidder and participate in Auctions started by other Sellers. Each Auction:

- has a Picture and an unique ID
- is created for selling an Item, which has a Title and Description
- has Price restrictions for the incoming bids - Reserve price, Buyout price and Minimum bid increment
- is opened for a specified Duration, updated by the Timer
- may have a Highest bid, placed by a Bidder

The domain model serves as the foundation for designing the system architecture, user interfaces and detailed workflows. By building upon the domain model, the design phase aims to develop a robust and scalable blueprint that ensures the system's successful realization.

3. Design

The analysis phase provided a comprehensive understanding of the project's functionality. The Domain model facilitated the establishment of the needed design specifications, which will serve as a blueprint for the implementation phase. Therefore, significant focus was placed on ensuring that the system was designed to be scalable, maintainable, and extensible. This section will focus on the architectural decisions, design patterns, and principles utilized to create a robust and adaptable system capable of meeting future needs, so mostly the brief classes overview of model, domain, persistence, and mediator packages will be presented, for a detailed project representation go to **Appendix B**.

The system follows the SOLID principles we're taking into consideration in order to achieve a robust design:
Single Responsibility Principle - each class handles one aspect of the system.
Open-Closed Principle – inheritance and many interfaces are used to support the addition of new features without risking existing functionality.
Liskov Substitution Principle – each subclass can replace its superclass without affecting the system's quality.
Interface Segregation Principle – each class has access to the needed methods only, which are separated in one of the four levels:

- *Auction* level – starting an auction, placing a bid, buyout or deleting an auction
- *AuctionList* level – accessing the list of ongoing auctions, auctions created by one specific Seller or auctions where one specific Bidder placed a bid.
- *User* level - creating an account, logging in, resetting the password, editing personal information or deleting the account
- *UserList* level – accessing the list of notifications (for Participants), accessing the list of Participants, banning or unbanning a Participant (for Moderator)

Dependency Inversion Principle – the connection between two packages is always made through an interface.

3.1 Server side

The server side doesn't have a GUI, as it plays the computational role in the application. It is responsible for managing auctions and accounts, processing bids, and handling communication between clients and the database.

The first point to mention is Dependency Inversion and Interface Segregation, which are being applied to design the Remote, Model and Persistence interfaces (see **Figure 3.1.2)**. *AuctionModel, AuctionListModel, UserMode* and *UserListModel* interfaces define the core functionalities of the model package, providing a layer of abstraction that separates the implementation details from the rest of the system. The *AuctionPersistence, AuctionListPersistence* and *UserPersistence* interfaces abstract the persistence layer, ensuring that data storage and retrieval operations are decoupled from the business logic. *AuctionListRemote, AuctionRemote, UserListRemote* and *UserRemote* divides remote functionality into four different parts following the same principle.

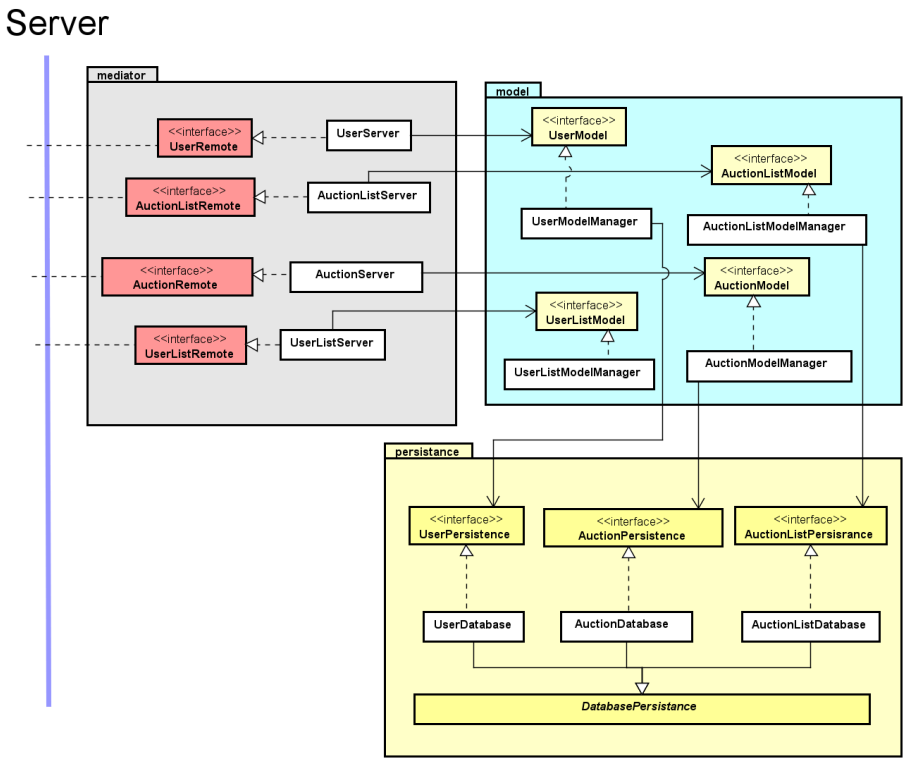


Figure 3.1.2 *Dependency Inversion, Interface Segregation on the server side, relevant part,* own production

Corresponding interfaces of mediator and model packages are defining the same methods. Their detailed representation can be found below, in **Figure 3.1.3**.

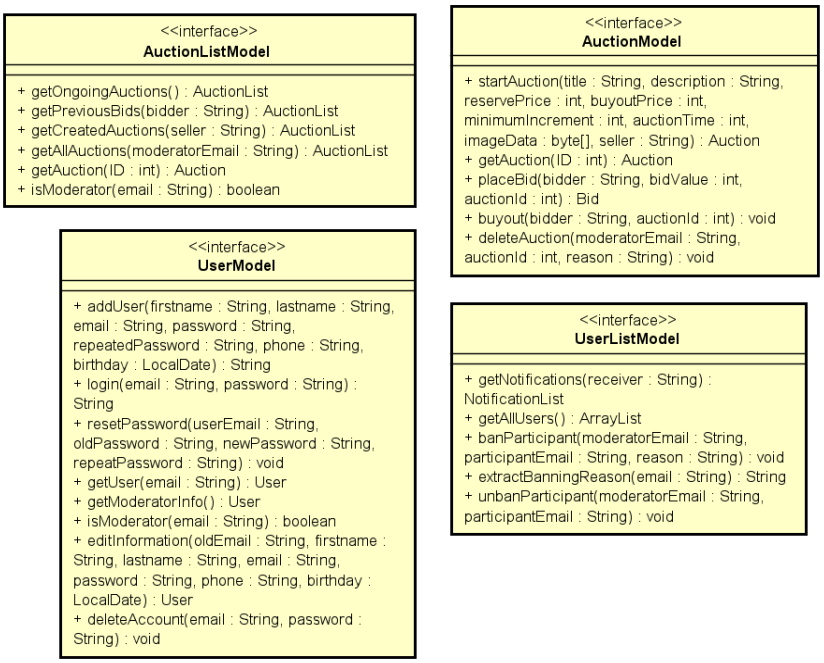


Figure 3.1.3 *Interfaces of mediator and model packages with their methods,* own production

The methods present in the persistence package interfaces can be seen below, in **Figure 3.1.4**.

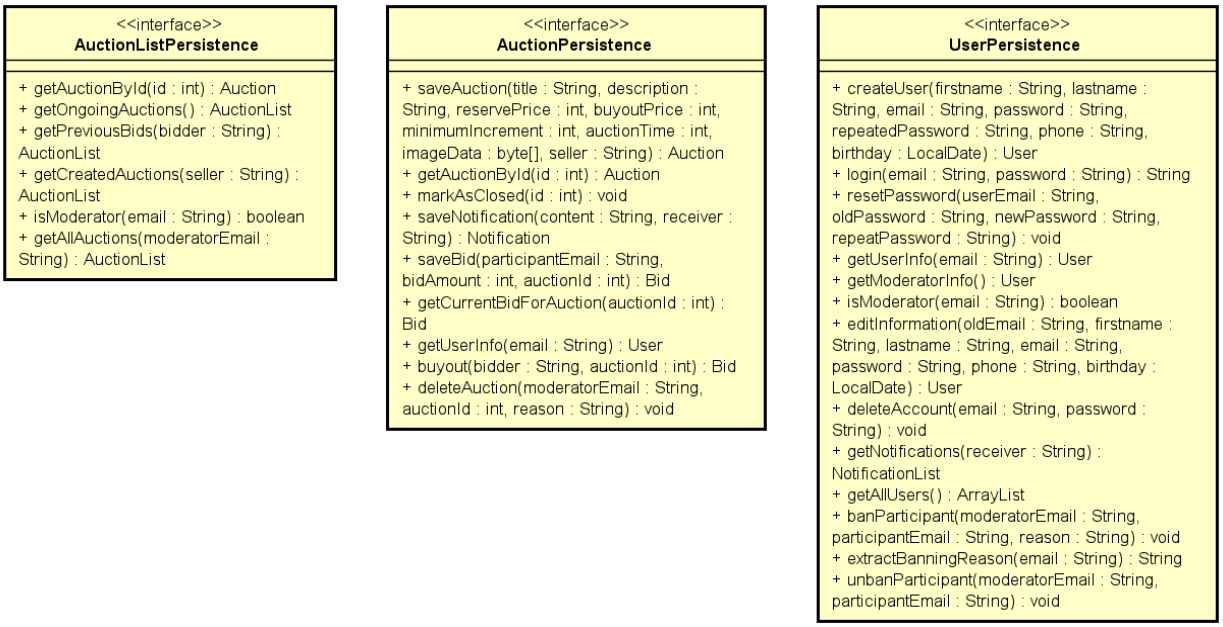


Figure 3.1.4 *Persistence interfaces,* own production

While designing the persistence package, *Protection Proxy* (see **Figure 3.1.5**) was used to enhance efficiency of database interactions. This design choice ensures that all updates to the database are checked before any query is performed, thereby reducing unnecessary load on the database.

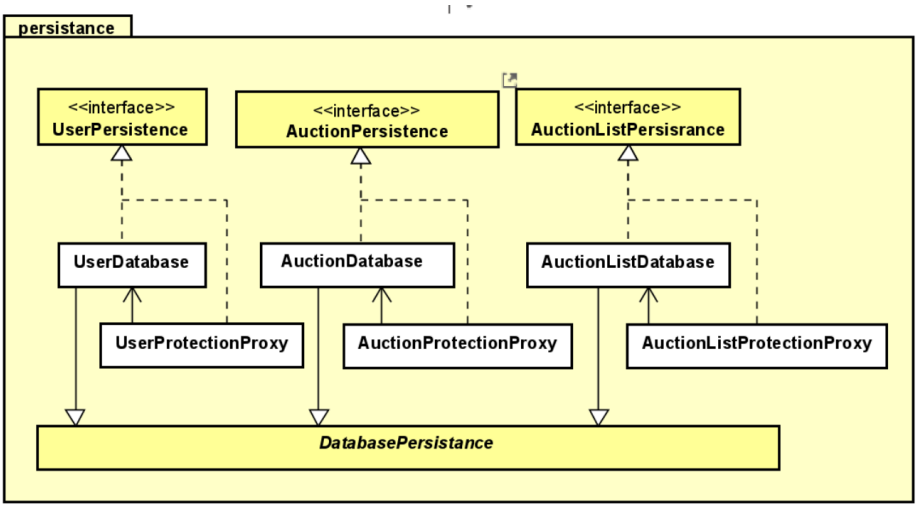


Figure 3.1.5 *Protection Proxy relevant part,* own production

Another core point is the usage of RMI. Remote interfaces in the mediator package are extending the *java.rmi.Remote* interface, which is crucial for enabling RMI, and facilitating communication between the server and remote clients. Representation of all the mediator classes and interfaces can be found below at **Figure 3.1.6**.

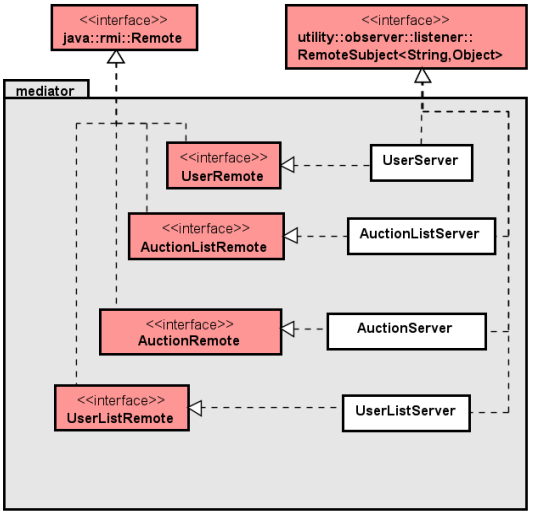


Figure 3.1.6 *Mediator package,* own production

The domain package is placed inside of the model package and represents domain model data, stated in the *Analysis* chapter, *2.5. Domain model*. Its relations to other classes are depicted in **Figure 3.1.7**.

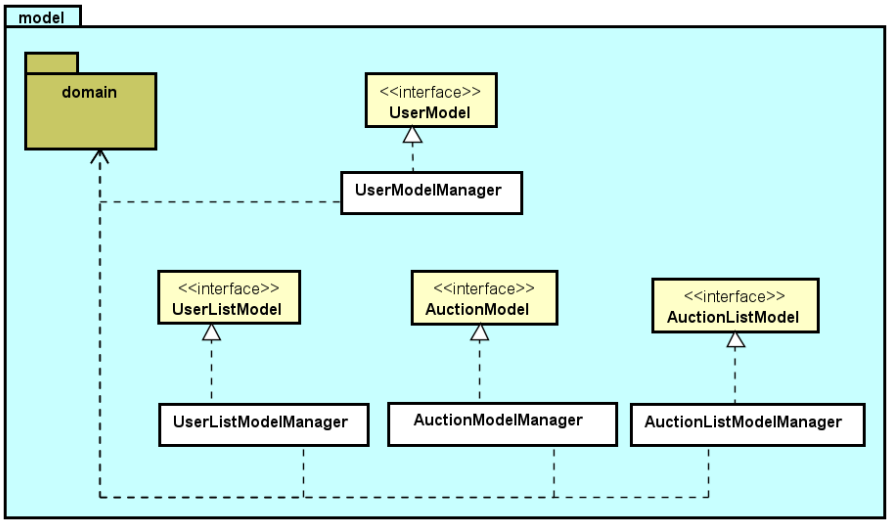


Figure 3.1.7 *Domain-model relations, relevant part,* own production

The *AuctionList, NotificationList, Auction, Item, PriceConstraint, Notification, User*, and *Bid* (see **Figure 3.1.8**) classes are implementing the *Serializable* interface,

enabling them to be easily encapsulated and transmitted over the network via RMI. The *ServerTimer* class implements *java.lang.Runnable*, being used for managing timed events within the auction system, such as updating auction countdowns and notifying closed auctions.

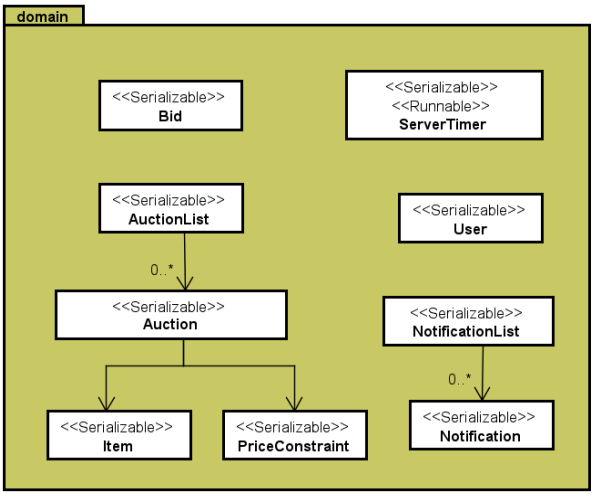


Figure 3.1.8 *Domain package,* own production

An important aspect of the design is the Observer pattern (see **Figure 3.1.9**) on the server side, utilizing libraries: *utility::observer::javaobserver:: NamedPropertyChangeSubject, java::beans::PropertyChangeListener,* and *utility::observer::listener::RemoteSubject*. To reduce coupling even more and complying with GRASP's Indirection pattern, the *ListenerSubject* class was created for those cases when one *Server* class has to be a listener of two *ModelManager* classes (example: *AuctionListServer* has to listen to *AuctionListModelManager* - the corresponding class, and to *UserListModelManager*, if a Participant is banned, all his auctions must be deleted from the all the auction lists). **Figure 3.1.9** below illustrates how the chained Observer pattern is designed, showcasing the connections between *subjects* and *listeners* across the server.

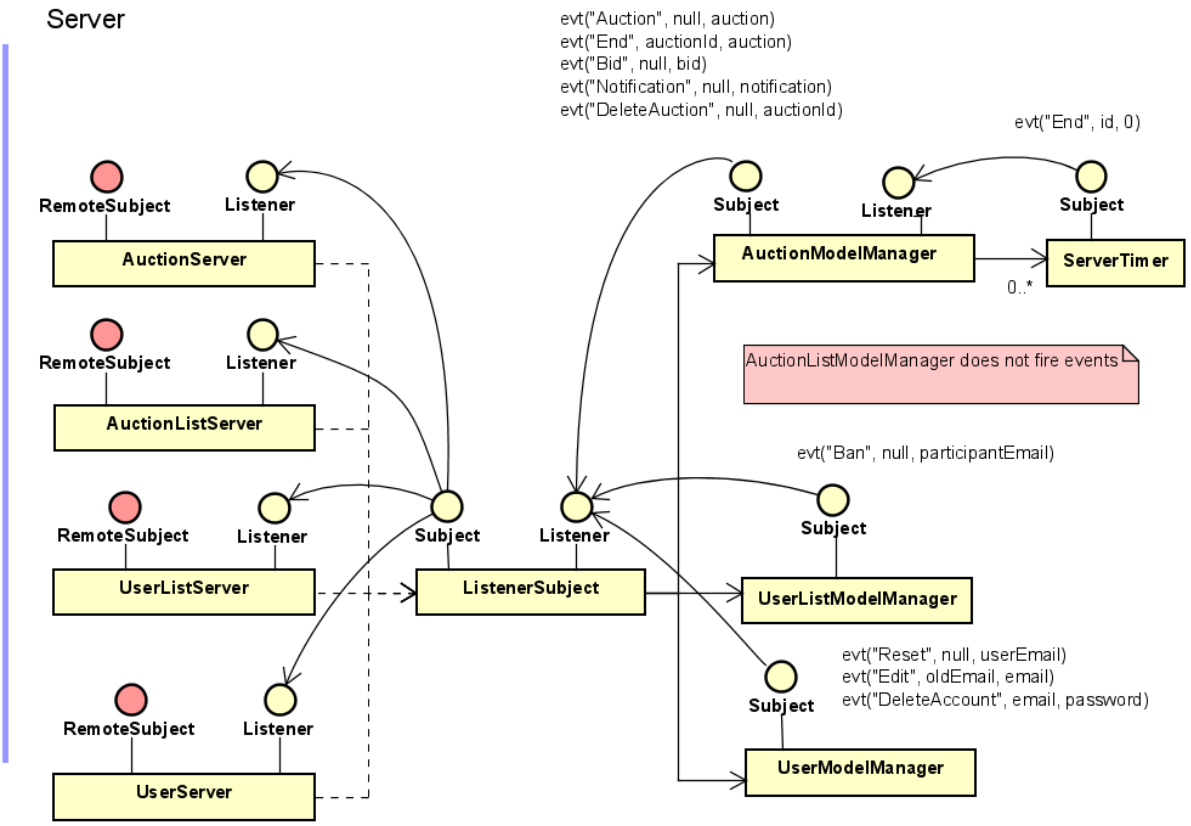


Figure 3.1.9 *Chained Observer Pattern relevant part, server package,* own production

So, the following events (of type *PropertyChangeEvent,* **Table 3.1.1**) at the can be fired from the server:

PropertyName	Methods firing	Notes
Auction	*startAuction(String title, String description, int reservePrice, int buyoutPrice, int minimumIncrement, int auctionTime, byte[] imageData, String seller)*	The *new value* of the event stores the created Auction object.
Bid	*placeBid(String bidder, int bidValue, int auctionId)*	The auction must update the displayed Highest bidder and Highest bid.
Notification	- *placeBid(String bidder, int bidValue, int auctionId)*	The “Notification” event is needed to inform:

	- *deleteAuction(String moderatorEmail, int auctionId, String reason)* - *sendContactInformation(int id)*	- the Bidder, whose bid was beaten; - the Seller, whose auction was deleted; - the Seller and the Bidder that the auction is closed and they can contact each other.
End	*startTimer(Auction auction)* *buyout(String current_bidder, int auctionId)*	Used to stop an ongoing auction.
Reset	*resetPassword(String userEmail, String oldPassword, String newPassword, String repeatPassword)*	The user who reset their password must be logged out from all devices.
Edit	*editInformation(String oldEmail, String firstname, String lastname, String email, String password, String phone, LocalDate birthday)*	The existing auctions may change the displayed Highest bidder;
Ban	*banParticipant(String moderatorEmail, String participantEmail, String reason)*	The banned Participant must be logged out from all devices and their auctions and bids must be deleted.
DeleteAuction	deleteAuction(String moderatorEmail, int auctionId, String reason)	Used to update the list of auctions.
DeleteAccount	deleteAccount(String email, String password)	The auctions and the bids of the user who deleted their account must be deleted as well.

Table 3.1.1 *Observer Pattern events, own production*

So, as mentioned above, one *server* class can listen to more than one *manager* class events through the *ListenerSubject* instance. Information about *server* package listeners is presented below in **Table 3.1.2.**

Server class	Property Name
AuctionServer	End, Bid, DeleteAuction, Ban, Edit, DeleteAccount
AuctionListServer	Auction, End, Bid, DeleteAuction, Ban, Edit, DeleteAccount
UserServer	Ban, Notification, Edit, Reset
UserListServer	Edit, DeleteAccount, Bid, Notification, Ban

Table 3.1.2 *Server listeners,* own production

3.2 Client side

For the client part of the Auction Application, a full Model-View-ViewModel (MVVM) architecture was implemented. This architectural pattern is particularly well-suited for *BidHub* Auction system as it requires a rich user interface. The only package that extends hierarchy is *mediator*. *AuctionRemote* and *AuctionClient* play a crucial role on the client side by facilitating communication between the client and the server through RMI mechanism.

Each package and its relations will be described separately. For detailed depiction, go to **Appendix B.**

As it can be seen in **Figure 3.2.1,** Interface Segregation principle was also followed on a client side. Moreover, *model* connection to the *mediator* is done in a similar to the server part way, *Domain* relations to *Manager* classes are the same as well.

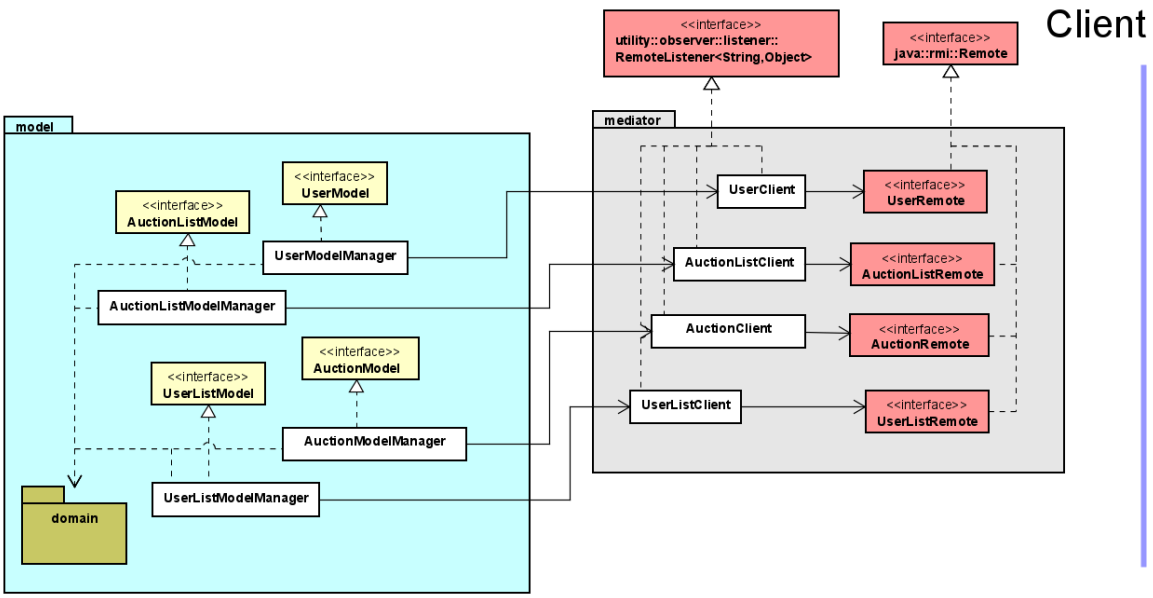


Figure 3.2.1 *Mediator-model on a client side, relevant part, own production*

Although the *model* package on the client side structurally is not changed, it operates differently compared to its server-side counterpart. Rather than handling computations and business logic locally, the client-side *ModelManager* relies on remote methods provided by the server to perform these tasks. This design ensures that all critical computations and data processing are centralized on the server, enhancing consistency, security, and performance.

To further optimize the client-side model, the package has been expanded to include a *Cache Proxy* design pattern (see **Figure 3.2.2**).

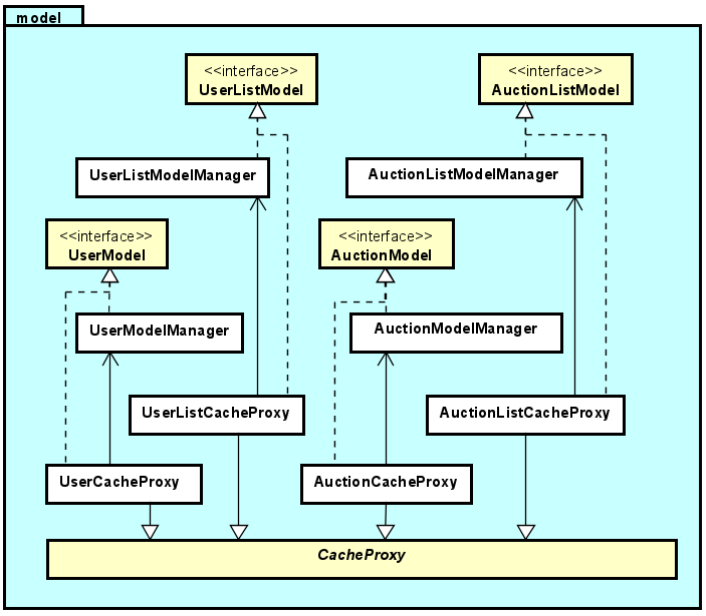


Figure 3.2.2 *Cache Proxy relevant part,* own production

Table **3.2.1** presents information on cached data and the classes where it is stored.

Cache class	**Stored cache**
AuctionListCacheProxy	All auctions (for moderator only), ongoing auctions, auctions created by the current user, and their previous bids
AuctionCacheProxy	Previously opened auctions
UserCacheProxy	Email of the logged in user
UserListCacheProxy	Notifications

Table 3.2.1 *Cache data storage, own production*

Proxy was designed in a way it has to be initialized only once. Cache updates are made through the Observer pattern. The Observer chain is constructed in the same way for all *Auction, AuctionList, User, UserList* logical levels. To follow the sequence, see **Figure 3.2.2.** All the updates from the server come in the form of events to the *Client* class, *ModelManager* re-fires them to the *CacheProxy*. After event processing, the needed cache is updated.

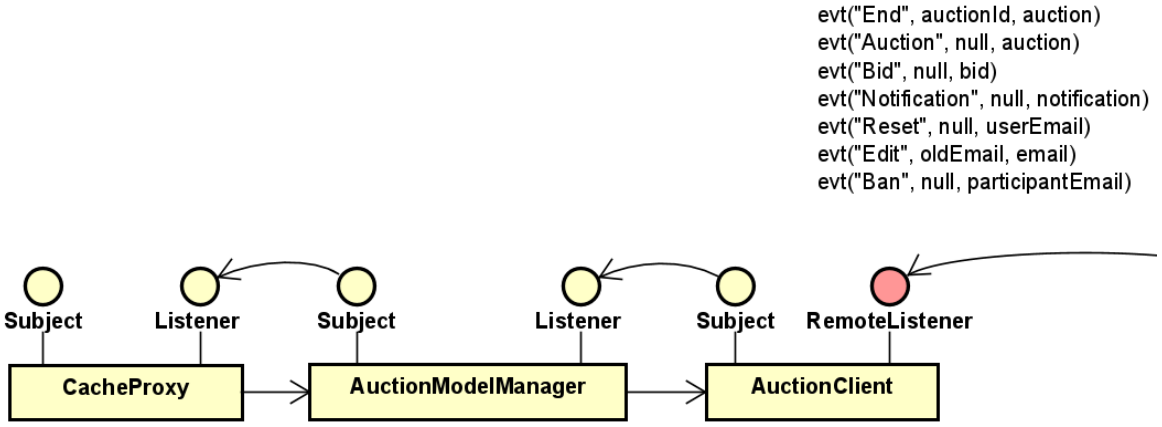


Figure 3.2.2 *Chained Observer Pattern, way to Cache Proxy,* own production

Initialization of *viewmodel* classes can be seen in **Figure 3.2.3.** *AllAuctions* and *AllAccounts_Notifications* viewmodels are responsible for windows with grid and table respectively, so additional viewmodels have been introduced.

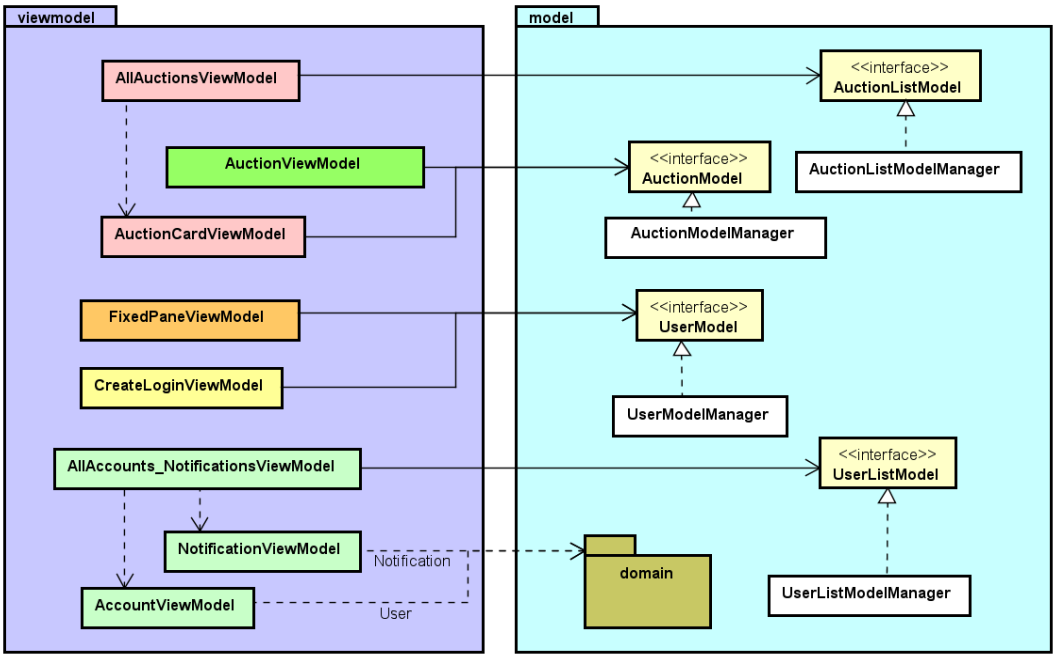


Figure 3.2.3 *Viewmodel-model relations relevant part,* own production

ViewModelFactory and *ViewModelState* classes (see **Figure 3.2.4**) were also introduced to the *viewmodel. ViewModelFactory* serves to initialize all the *viewmodel* classes and *ViewModelState* serves to store crucial information for window interaction.

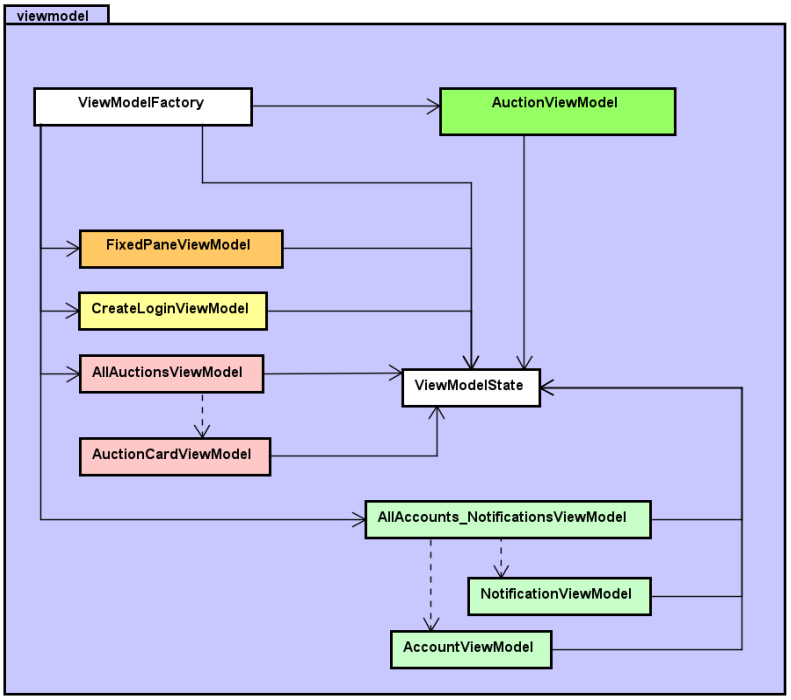


Figure 3.2.4 *ViewModelFactory, ViewModelState relevant part,* own production

The Observer Pattern chain ends at the *viewmodel* package (see **Figure 3.2.8**). Further communication between *view* and *viewmodel* is done through bindings (see **4. Implementation**).

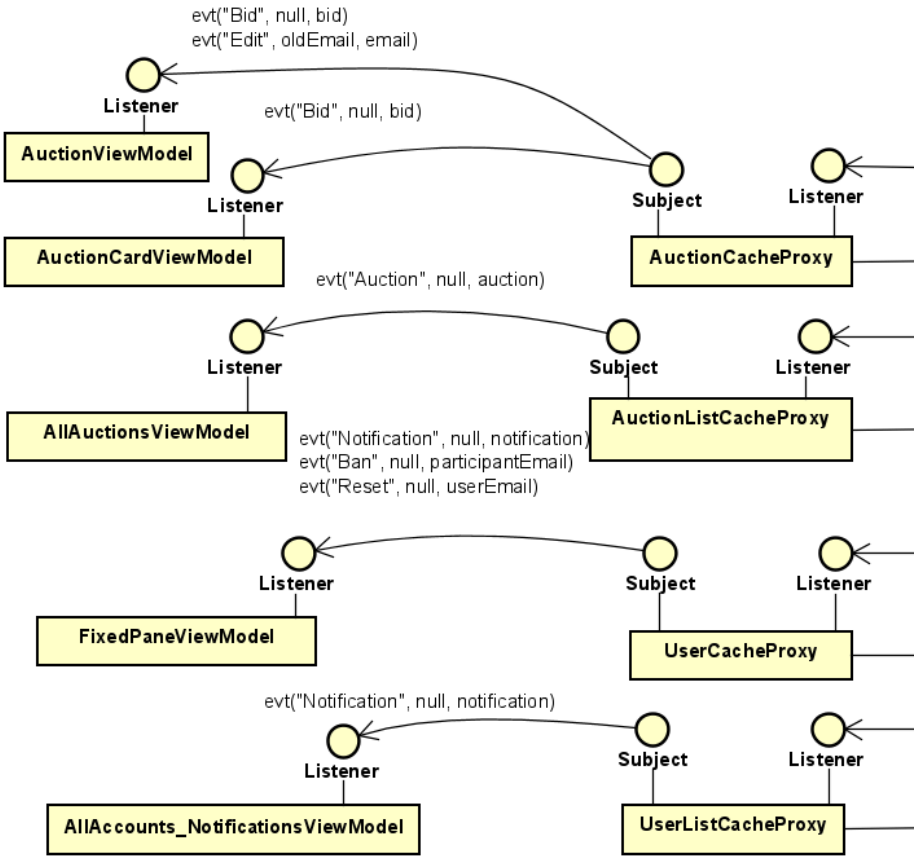


Figure 3.2.8 *Chained Observer Pattern, Cache Proxy-ViewModel,* own production

Initialization of *viewmodel* classes can be seen in **Figure 3.2.3.** As it was mentioned above, the *AllAuctions* window has a grid inside, each grid element has to be initialized separately. Therefore, it is needed to create controllers (complying with GRASP's Controller pattern) for each of those elements.

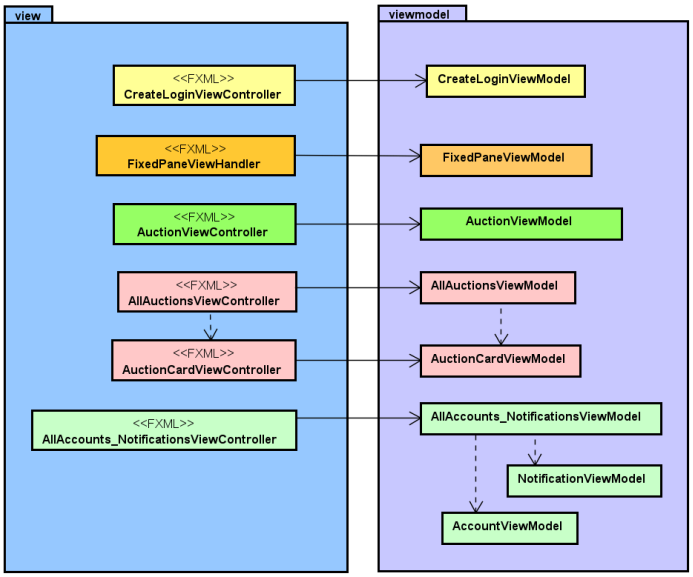


Figure 3.2.5 *View-viewmodel* relations relevant part, own production

To better understand controllers' relation to the actual view, a screenshot of the BidHub application is presented in **Figure 3.2.6.** *Login&CreateAccount* and *FixedPane* are two main windows that can be opened. *FixedPane* can be seen as the main window of the system, whereas *AllAuctions, Auction, AllAccount&Notification* are subwindows. They are changing the view of the black middle pane of a *FixedPane* window.

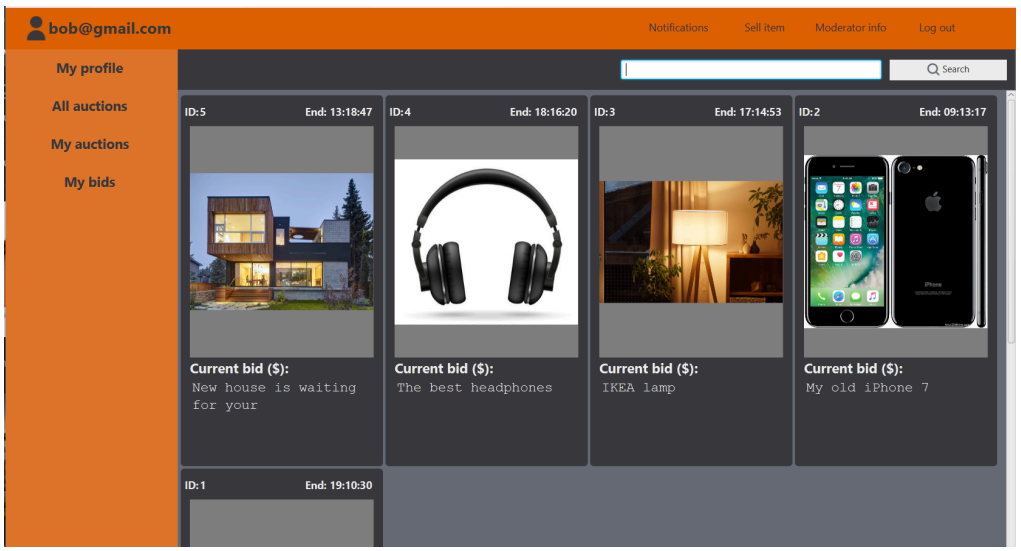


Figure 3.2.6 *FixedPane view,* own production

ViewHandler in the *view* package serves for switching the view between windows (see **Figure 3.2.6**). *FixedPane can* also be interpreted as a *ViewHandler class,* because all its elements play a window switching role.

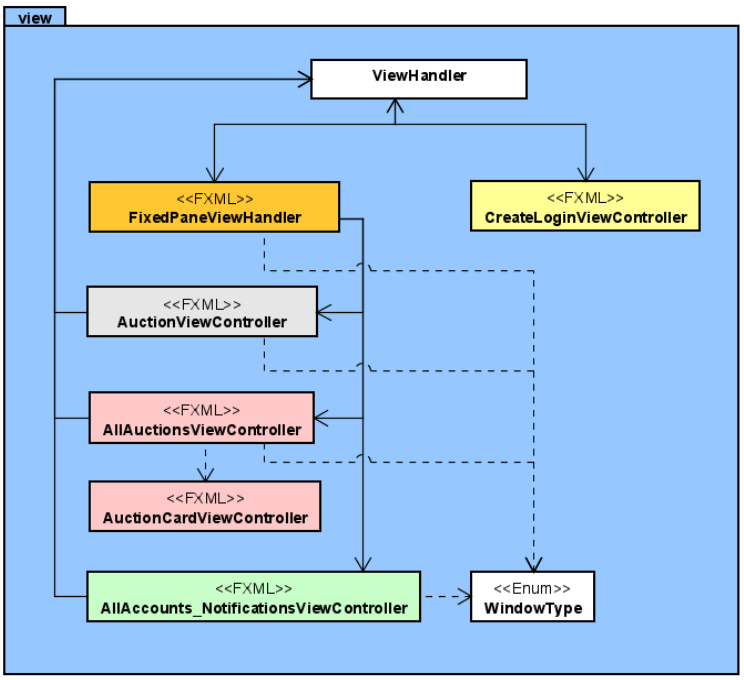


Figure 3.2.6 *ViewHandler relevant part,* own production

Some *.fxml* files are reused, as they play a similar role in the system (e.g. *Login* and *CreateAccount*), so the *WindowType* enum was introduced in order to reprocess the windows and achieve low coupling (see **Figure 3.2.6** and **Figure 3.2.7**).

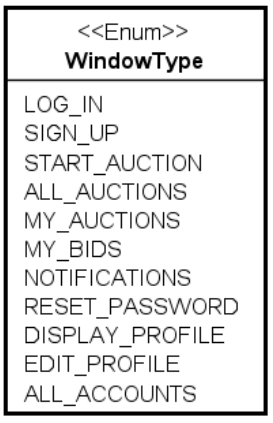


Figure 3.2.7 *WindowType enum,* own production

With all components comprehensively detailed, the next crucial aspect to address is the server-client communication. The client can establish the RMI

connection, facilitating the real-time exchange of data and execution of the necessary methods that support system's operational requirements.

3.3 Client-server communication

All requests from the client to the server have a similar path, illustrated in **Figure 3.3.1** The *ViewController* delegates the task to its corresponding *ViewModel*, which calls the *CacheProxy*, *ModelManager*, *Client* and *Server*.

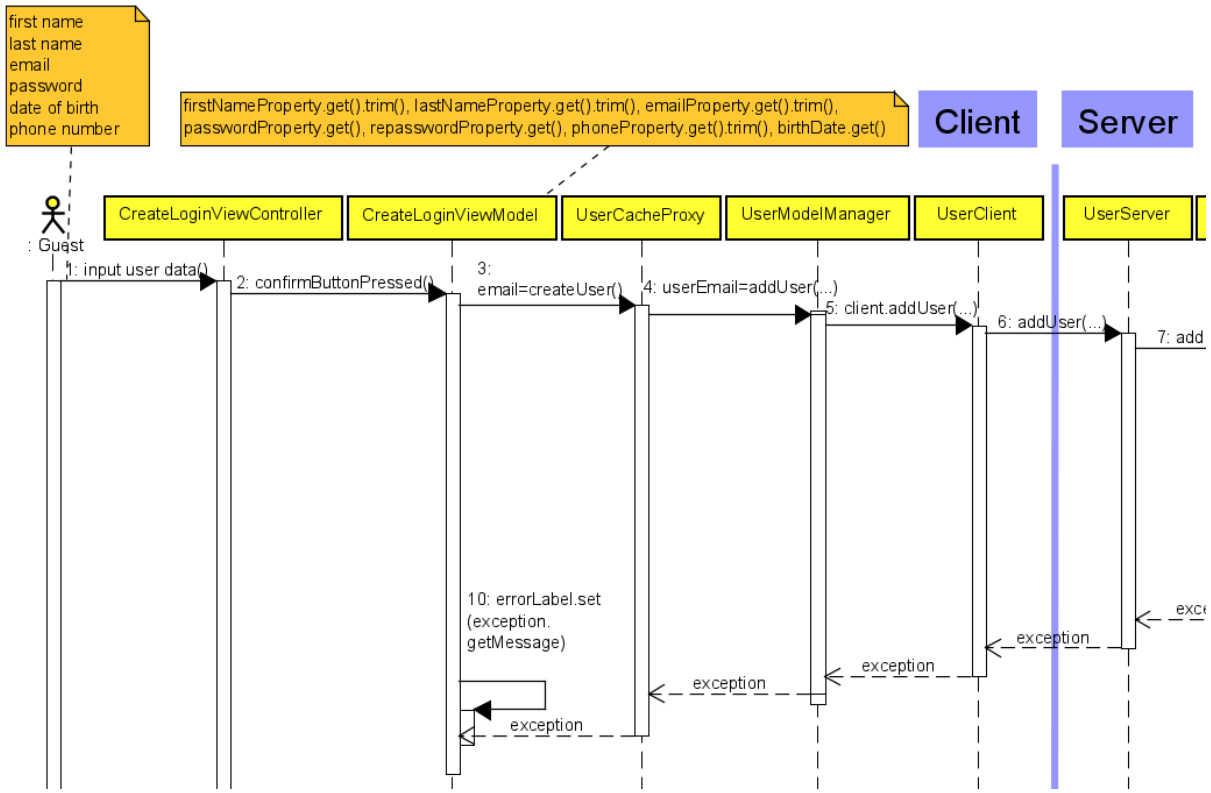


Figure 3.3.1 *Sequence diagram for the "createUser()" method, client side,* own production

The server delegates the task further, to the *ModelManager* and then, to the *ProtectionProxy*. The *ProtectionProxy* filters the calls for the database, as it can be seen in **Figure 3.3.2**.

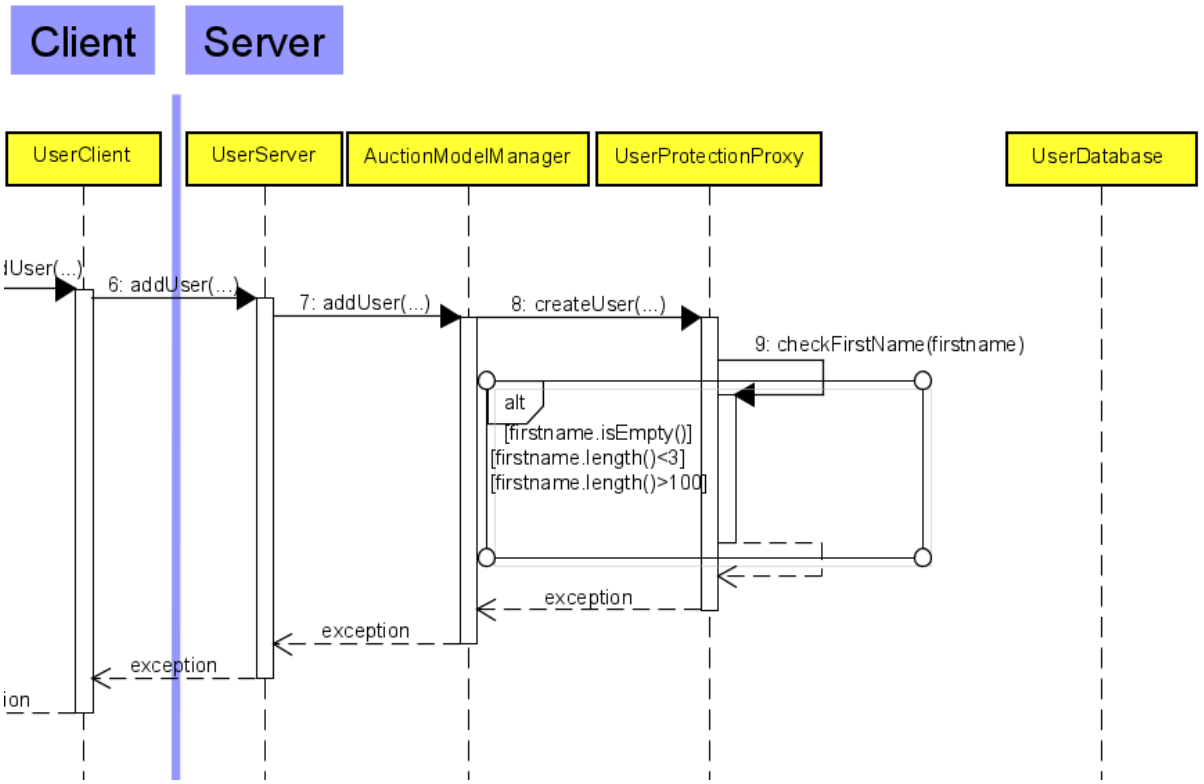


Figure 3.3.2 *Sequence diagram for the "createUser()" method, server side,* own production

When the data passes the regular validations, it is forwarded to the database (see **Figure 3.3.3** and **Figure 3.3.4**)

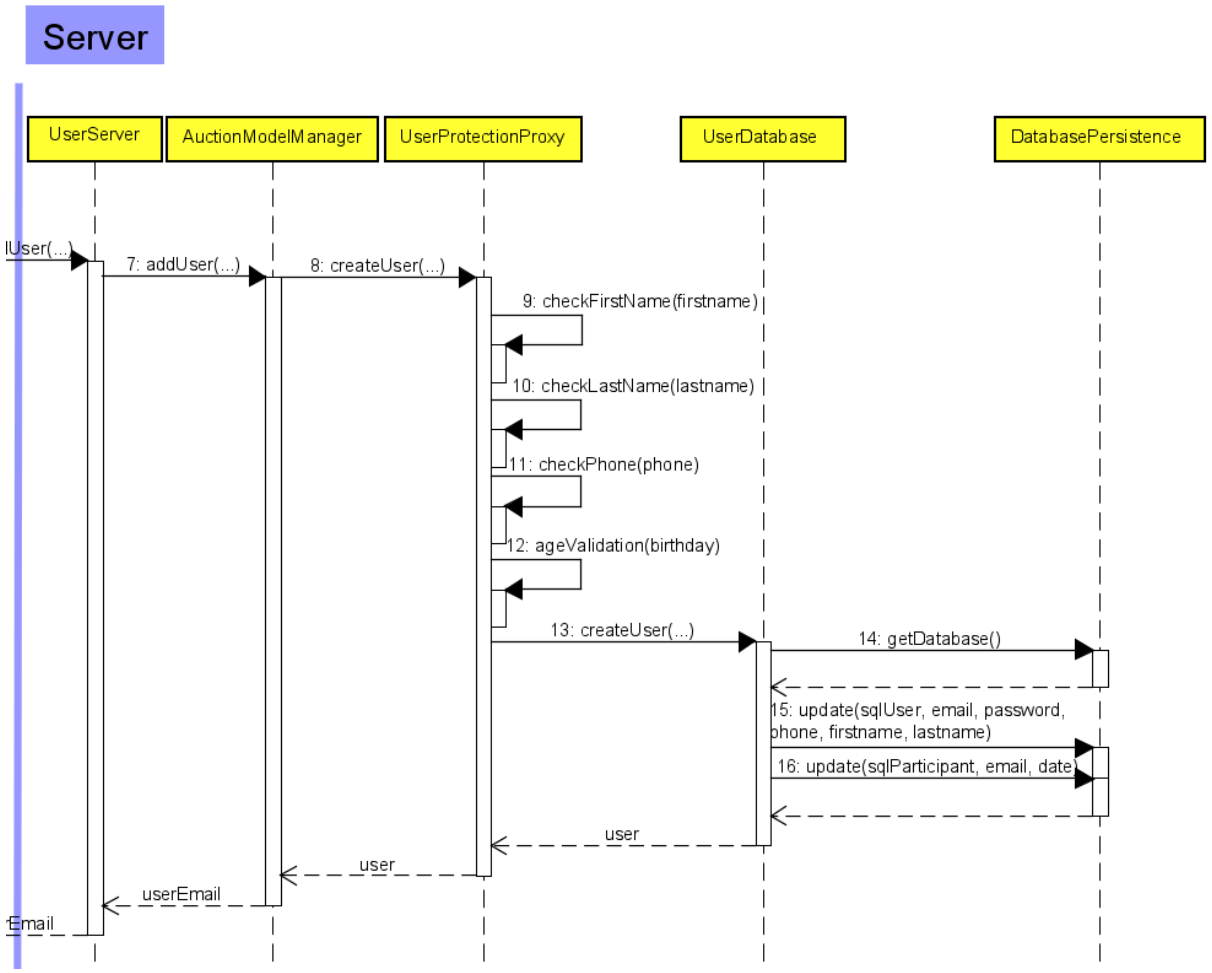


Figure 3.3.3 *Sequence diagram for the "createUser()" method, server side,* own production

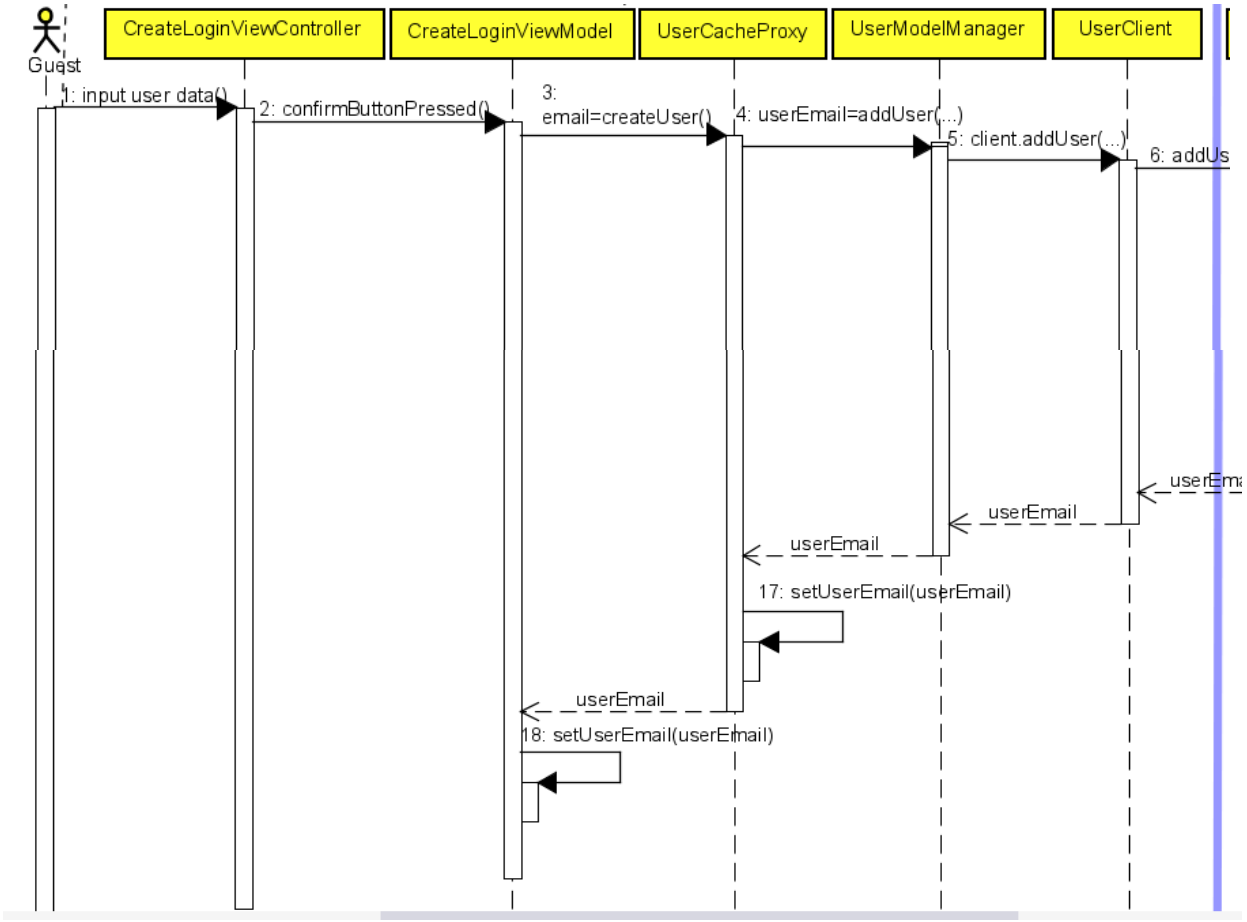


Figure 3.3.4 *Sequence diagram for the "createUser()" method, client side,* own production

When making requests to the server, the CacheProxy is updated automatically with the returned value of the call. This practice is also applied for logging in and it ensures efficient cache actualization.

Most updates are not isolated, but instead, they are broadcasted to all clients in order to allow them to access the actualized information. **Figure 3.3.5** shows that each Participant receives the updated information as result of the interaction of Bidder1 with the Server, by placing a bid. Moreover, when Bidder2 outbids Bidder1, the latter one is notified, as they previously participated in the same auction.

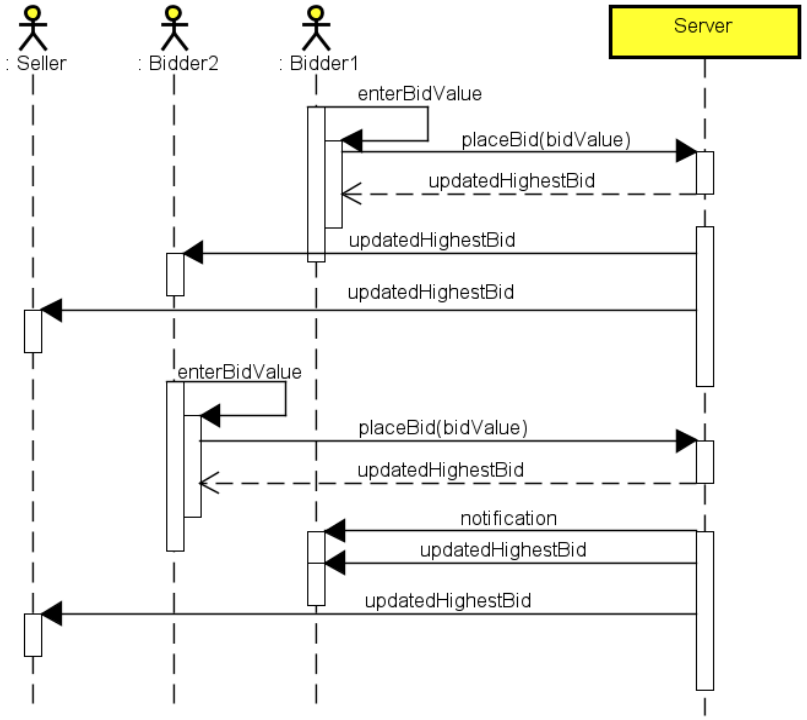


Figure 3.3.5 *Visualization of the process of bidding - two Bidders,* own production

3.4 Database

Figure 3.4.1 illustrates the structure of the database for the Auction Application, representing various entities, their attributes, and the relationships between them. At the core of the diagram is the User entity, which contains essential attributes such as *first_name, last_name, phone_number,* and *user_email.* Every user in the system must be either a *Moderator* or a *Participant*, with *Moderator* entities extending *User* and adding a *personal_email* attribute. Similarly, *Participant* entities, also derived from *User*, include a *birth_date* attribute.

The *Participant* can create multiple *Auctions*, which encompass attributes like *title, description, reserve_price, buyout_price, minimum_bid_increment, current_bid, current_bidder, image_data, status, start_time, end_time,* and *creator_email.* Participants also place multiple bids on an auction.

The system manages banned participants through the *banned_participant* entity, which holds a *reason* attribute for the ban. This relationship is optional, indicating that a participant might not always be banned. Additionally, participants can receive multiple *notification* entities, each containing *content, date,* and *time* attributes, representing a one-to-many relationship. This structure ensures comprehensive tracking of user roles, auction activities, bid interactions, and notifications, while also accommodating user management with roles and potential bans.

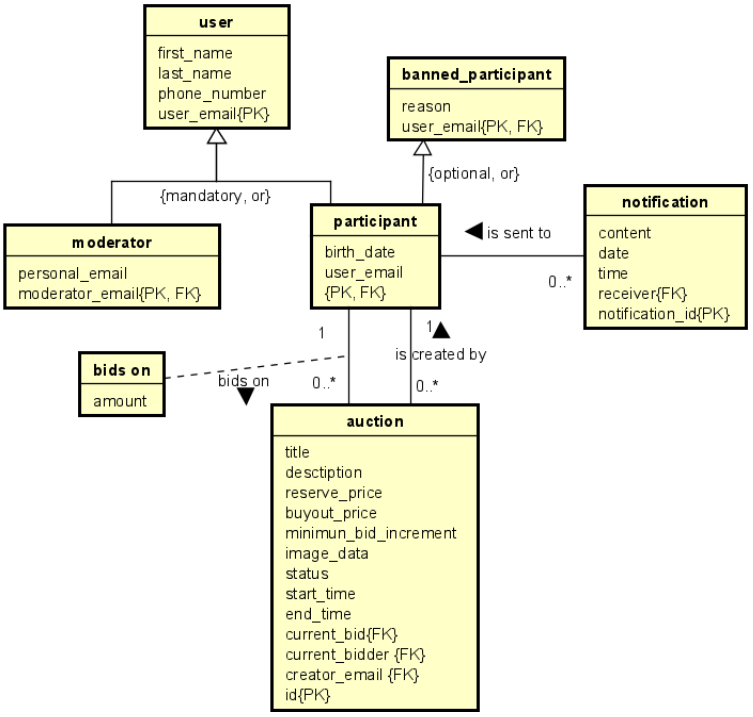


Figure 3.4.1 *EER diagram,* own production

In the EER diagram, inheritance was used to model the relationships between *User*, *Moderator*, and *Participant*. However, when converting this to a relational database schema, these are implemented as separate tables, to simplify the database structure and improve query performance. By creating distinct tables for *Moderator* and Participant`, each inheriting from the User table through foreign keys, we ensured that the specific attributes of `Moderator` and `Participant` are efficiently managed and accessed.

In the relational schema (**Figure 3.4.2**), the *User* table holds common attributes like *first_name, last_name, phone_number,* and *user_email*, which is the primary key. The *Moderator* table includes an additional *personal_email* attribute and references the *User* table via a foreign key (*user_email*). Similarly, the *Participant* table includes a *birth_date* attribute and also references the *User* table. This design allows to maintain the relationships and shared attributes while clearly separating the roles and their specific attributes, making it easier to handle role-specific operations and constraints.

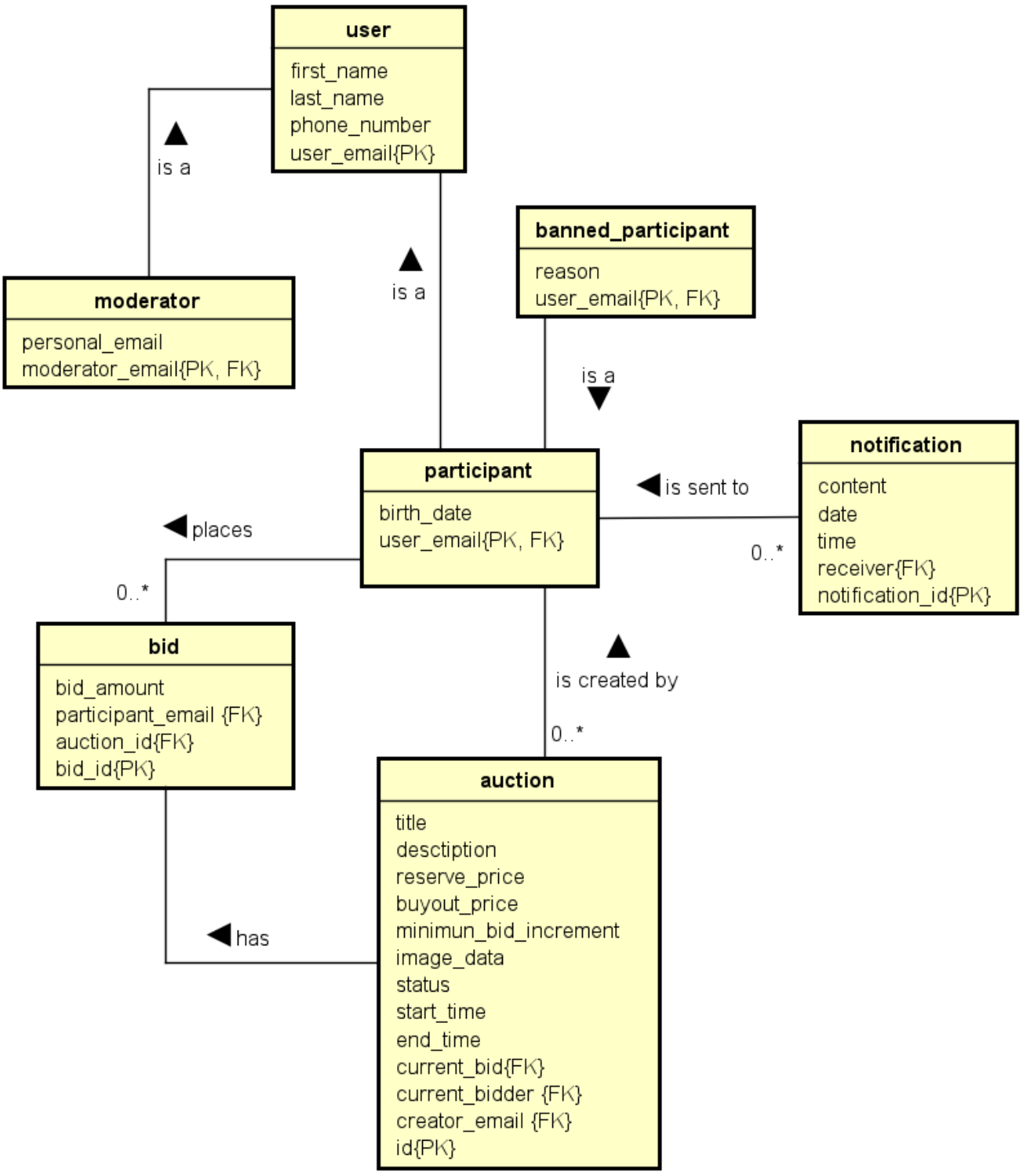


Figure 3.4.2 *GR diagram,* own production

4. Implementation

With the detailed class diagrams, data structures and system's architecture serving as a blueprint for the implementation, the design specifications are converted into actual code. The implementation phase will focus on building the system's functionalities and integrating components for a robust software system and the major elements will be presented in the following pages.

4.1 Bid placing feature

Within the context of an auction client-server system, placing a bid is a crucial ability for a Participant to have.

1. In order to place a bid for an item, the Bidder must select the auction, enter the bid value and Click the *Place bid* button (see the **User Guide** in **Appendix C**), which triggers the call of many methods:
- `placeBidButtonPressed()`, in the AuctionViewController - calls the `auctionViewModel.placeBid()` method, with all the fields being bound, so the *ViewModel* is able to access the bid value and the auction ID:

2. Method executed in *AuctionViewModel* will update the Highest bid and the Highest bidder with the values returned from the server, if no errors occur after calling the `model.placeBid(..)` with the Bidder's email stored in the *ViewModelState* after login. If exceptions are thrown from the server, the exception message is displayed in the error label and the bid field is cleared, to make room for a new bid.

```
public void placeBid() {
 errorProperty.set("");
 //  Creating an empty bid
 Bid bid = null;
 try {
   // Calling the model and storing the bid
bid = model.placeBid(state.getUserEmail(), incomingBidProperty.get(),
       idProperty.get());
 }
 //  If exception is caught, set the bid to 0 and display the message
 catch (SQLException e) {
   errorProperty.set(e.getMessage());
   incomingBidProperty.set(0);
 }
 //  if no errors, existing bid and id is matching with the id of auction
matches the one from bid
 //  Display the new highest bidder and their bid in the window
 if (errorProperty.get().isEmpty() && bid != null
```

```
      && idProperty.get() == bid.getAuctionId()) {
    currentBidProperty.set(bid.getBidAmount());
    currentBidderProperty.set(bid.getBidder());
    incomingBidProperty.set(0);
  }
}
```

3. The call from *AuctionViewModel* is passed down through *model*'s *CacheProxy* before *client's AuctionModelManager* receives the call. Further, *AuctionModelManager* passes it to the mediator's *AuctionClient* that is actively using the RMI Interface *AuctionRemote* to invoke methods on the identical *AuctionRemote* class, on the server side. This interface is implemented by *AuctionServer*, which calls the model's *AuctionModelManager*.

4. After the server's *AuctionModelManager* receives the call, the Highest bid is extracted for the specified auction, so the Highest bidder will receive a notification if the potential bid is validated. Lastly, the *"Bid"* event is being fired for the *mediator,* and broadcasted to all clients, so their values for the Highest bid and bidder will be updated for this auction.

```
@Override public synchronized Bid placeBid(String bidder, int bidValue,
    int auctionId) throws SQLException
{
  //  Existing bidder is being extracted
  Bid existingBid =auctionDatabase.getCurrentBidForAuction(auctionId);
  Bid bid = auctionDatabase.saveBid(bidder, bidValue, auctionId);
  //  if they exist, the notification is being sent in the event
  if (existingBid != null)
  {
    String n_content = "Your bid has been beaten for auction ID: " + auctionId
+ ".";
     Notification notification = auctionDatabase.saveNotification(n_content,
existingBid.getBidder());
    property.firePropertyChange("Notification", null, notification);
  }
  // Fires event for the server
  property.firePropertyChange("Bid", null, bid);
  return bid;
}
```

Before calling the actual *database,* the protection proxy is called to ensure that all the passed data is valid.

```
@Override public Bid saveBid(String participantEmail, int bidAmount,
    int auctionId) throws SQLException
{
  if (!isNotModerator(participantEmail))
    throw new SQLException("The moderator cannot place bids.");
```

```
 database.checkBid(bidAmount, participantEmail, auctionId);
 return database.saveBid(participantEmail, bidAmount, auctionId);
}
```

If the information passes the checks, the call is then forwarded to the database, which is being updated and the created Bid is being returned, to reach the ViewModel. In accordance with GRASP, the database classes are the Information experts and the Creators, as they can access the necessary data and methods to manage and create resources.

```
@Override public synchronized Bid saveBid(String participantEmail,
   int bidAmount, int auctionId) throws SQLException
{
 checkBid(bidAmount, participantEmail, auctionId);
   String   sql   =   "INSERT   INTO   sprint1database.bid   (participant_email,
auction_id, bid_amount) VALUES (?, ?, ?)";
 database.update(sql, participantEmail, auctionId, bidAmount);
 Bid bid = new Bid(auctionId, participantEmail, bidAmount);
 updateCurrentBid(bid);
 return bid;
}
```

The task of `checkBid(...)` method is to validate the bid according to the "PLACE BID" Scenario, from the "Participate in auction" use case (see chapter 2.4 *Use cases*). Overall, each element of the system is checked independently, in separate methods, in order to enhance code reusability (the `checkBid(...)` is also used for the buyout feature) and follow Curly's Law, as it is not the `saveBid(...)` method's responsibility to validate the bid.

```
public void checkBid(int bidAmount, String participantEmail, int auctionId)
   throws SQLException
{
 //  Statement preparation
 String retrieveSql =
                   "SELECT    auction.current_bid,    auction.current_bidder,
auction.reserve_price,     auction.minimum_bid_increment,     auction.status,
auction.creator_email\n"
         + "FROM auction\n" + "WHERE auction.ID=?;";
 //  Arraylist of object arrays is being received from the query
 ArrayList<Object[]> results = database.query(retrieveSql, auctionId);
 for (int i = 0; i < results.size(); i++)
 {
   Object[] row = results.get(i);
   //  Results from the query are being stored
   int currentBid = Integer.parseInt(row[0].toString());
   String currentBidder = null;

   if (row[1] != null)
```

```
    {
      currentBidder = row[1].toString();
    }

    int reservePrice = Integer.parseInt(row[2].toString());
    int increment = Integer.parseInt(row[3].toString());
    String status = row[4].toString();
    String seller = row[5].toString();

    //  Checks for the results from the query
    if (!status.equals("ONGOING"))
      throw new SQLException("The auction is closed.");
    if (participantEmail.equals(currentBidder))
      throw new SQLException("You are the current bidder.");
    if (participantEmail.equals(seller))
      throw new SQLException("You cannot bid for your item");
    if (currentBid > 0)
    {
      if (bidAmount <= currentBid + increment)
        throw new SQLException("Your bid is not high enough.");
    }
    if (bidAmount < reservePrice)
      throw new SQLException("Your bid must be at least the reserve price.");
  }
}
```

Lastly, the method `updateCurrentBid(Bid currentBid)` takes care of replacing the out-dated information about Highest bid and Highest bidder in the actual database.

```
private void updateCurrentBid(Bid currentBid) throws SQLException
{
  String sql = "UPDATE auction SET current_bid=?, current_bidder=? WHERE
auction.ID=?;";
database.update(sql,currentBid.getBidAmount(),currentBid.getBidder(),
     currentBid.getAuctionId());
}
```

Each call to the server has a similar path: *ViewController (view) > ViewModel (viewmodel) > CacheProxy (model) > ModelManager (model) > Client (mediator) >>RMI>> Server (mediator) > ModelMananger (model) > ProtectionProxy (persistence) > Database (persistence)*.

In most cases, the *ModelManager* (server side) is the one that fires event that reach the client (exception: the “End” event is fired in the *ServerTimer* class and catched by the *ModelManager*): *ModelManager (model) > ListenerSubject (model) > Server (mediator) >>RMI>> Client (mediator) > ModelManager (model) > CacheProxy (model)*(some events stop here to update the cache) > *ViewModel (viewmodel)*.

4.2 Cache proxy

When making calls from server and database, the performance can be improved by implementing a proxy design pattern which serves as local cache storage for the frequently accessed data and allows loading it when needed. This would result in less calls to the server.

Similarly to the AuctionModelManager on the client side, the *AuctionCache* class implements the *AuctionModel*, with the addition of a private instance variable `previousOpenedAuctions` which is of `AuctionList` type. If `previousOpenedAuctions` contains the requested auction, it will be loaded from cache, otherwise, a call to the AuctionModelManager will be made and the newly accessed auction is added in cache. Additionally, instead of starting the timer on the server side to update the auction, the timer is started locally, in the client's application, firing “Time” events. The next time a request will be made for this auction, it will be loaded from `previousOpenedAuctions` and the timer will not start, because there is already a running timer which updates this auction.

Similarly, the cache is used to load notifications and lists of auctions (ongoing auctions, auctions started by the logged user or the auctions they participated in).

```
@Override public Auction getAuction(int ID) throws IllegalArgumentException
{
 if (previousOpenedAuctions.contains(ID))
   return previousOpenedAuctions.getAuctionByID(ID);
 Auction auction = modelManager.getAuction(ID);
 startTimer(auction);
 previousOpenedAuctions.addAuction(auction);
 return auction;
}
```

The cache is updated via the events received. When the AuctionCache class receives a “Bid” event, the `previousOpenedAuctions` is updated by calling `updateBidIn(bid, previousOpenedAuctions);`:

```
private void updateBidIn(Bid bid, AuctionList cache)
{
 if (cache.contains(bid.getAuctionId()))
 {
   cache.getAuctionByID(bid.getAuctionId())
        .setCurrentBidder(bid.getBidder());
   cache.getAuctionByID(bid.getAuctionId())
        .setCurrentBid(bid.getBidAmount());
 }
}
```

It is worth mentioning that except the `previousOpenedAuctions` , the proxy only stores short versioned auctions (only the information needed for

displaying the cards), as they are loaded from the database. Instead of extracting all information related to all auctions, only the Title, End time, Picture and Highest bid are returned from the server as *AuctionLists*. In order to apply the DRY principle (Don't Repeat Yourself), an overloaded constructor for the *Auction* class has been used to extract the necessary data.

4.3 Self-validating database

The database is designed in a manner that allows the system's expansion (open for extension). Despite having the ProtectionProxy, the database realizes its own checks before making any updates. For example, a `title` could not be made with less than 5 or more than 80 characters. For each auction stored, the system needs to know whenever it is ongoing or not. This is stored in `status` and is constructed in a way that it could either be *"ONGOING"* or *"CLOSED".*

```
CREATE TABLE IF NOT EXISTS auction(
  ID SERIAL PRIMARY KEY,
  title VARCHAR(80) CHECK (length(title)>5) NOT NULL,
  description VARCHAR(1400) CHECK (length(description)>20) NOT NULL,
  reserve_price INTEGER NOT NULL,
  buyout_price INTEGER NOT NULL,
  minimum_bid_increment INTEGER NOT NULL,
  current_bid INTEGER,
  current_bidder email,
  image_data VARCHAR(250) NOT NULL ,
  status VARCHAR(7) CHECK (status IN ('ONGOING','CLOSED')) NOT NULL,
  start_time TIME NOT NULL,
  end_time TIME NOT NULL,
  creator_email email NOT NULL,
   FOREIGN KEY (creator_email) references participant(user_email) ON UPDATE
CASCADE,
   FOREIGN KEY (current_bidder) references participant(user_email) ON UPDATE
CASCADE
);
```

The system has a ProtectionProxy class that controls the entries to the database for each class: *AuctionDatabase, AuctionListDatabase, UserDatabase*. These classes have access to the same database, by extending the *DatabasePersistence* class, which created the actual database.

4.5 Auction card generation

A system displays auctions resembling cards on a grid, in a way that they will be placed next to each other 4 cards at most. Method `renderGridWithCards(...)` allows the system to dynamically render a grid of auction cards based on data that is provided. This action requires a usage of nested *for* loops for rendering each item accordingly using method

`addNewCardToGrid(...)`. This implementation will require for system to run this method at longest case *O(n)* time in Big O time complexity

```
private void renderGridWithCards(ObservableList<Auction> auctionCards)
{
 try // try-catch not affecting
 {
// Initialization and setting of variables are running in constant time
// Calculating numRow is arithmetic operation which also is running in O(1)
int totalElements = auctionCards.size();  // O(1)
int numRows = (totalElements + NUMBER_OF_COLUMNS - 1) / NUMBER_OF_COLUMNS;//
O(1)
int listIndex = auctionCards.size() - 1;  // O(1)

   // Outer loop runs 'numRows' times which is running in O(n)
   for (int row = 0; row < numRows; row++) // O(n) / n
   {
       // Inner loop runs 'NUMBER_OF_COLUMNS' times for each iteration of
       outer loop, NUMBER_OF_COLUMNS in this system's case always
     for (int column = 0; column < NUMBER_OF_COLUMNS; column++) // O(n)
     {
       // Check is happening in each iteration which is O(1)
       if (listIndex >= 0)
       {
         // auctionCards.get(listIndex) is called.
         // Accessing an element in an ObservableList by index is O(1)

         // Getter for auction by its index runs in O(1)
         // addNewCardToGrid method is running in O(1) as well
         addNewCardToGrid(auctionCards.get(listIndex), column, row);
         //  Decrementing listIndex is O(1)
         listIndex--;
       }
     }
   }
 }
 catch (IOException e) {}
 // Overall Complexity:
 // The outer loop runs approximately totalElements / NUMBER_OF_COLUMNS.
 // The inner loop runs NUMBER_OF_COLUMNS times for each iteration of the
       outer loop, which is currently set to 4 (n=4)
 // The overall complexity would be O(n^2), but since the inner loop always
//runs 4 times, it can be stated that the complexity is n*4, therefore
//O(n), where n is the number of auctions that have to be displayed.
}
```

Initialization of each card happens in a way similar to any other *view* in a system. Root node of the *fxml* file initialized in *AuctionViewController* is appended

and loaded into a `GridPane auctionsGrid`. Data is binded with *AuctionCardViewModel* and receives its data from a *model* package.

```
private void addNewCardToGrid(Auction auction, int column, int row)
    throws IOException
{
 AuctionCardViewController newCardController = initNewCard();
 newCardController.setData(auction);

 auctionsGrid.add(newCardController.getRoot(), column, row);
 GridPane.setMargin(newCardController.getRoot(), new Insets(-1));
}
```

Implementation of `initNewCard()` with auction cards loaded using *AuctionCardViewController* allows initialization bound to *AuctionCardViewModel* with active listening to all necessary events fired from the *model* package.

```
private AuctionCardViewController initNewCard() throws IOException
{
 FXMLLoader load = new FXMLLoader();
 load.setLocation(getClass().getResource("AuctionCardView.fxml"));

 Region innerRoot = load.load();
 AuctionCardViewController auctionCardViewController = load.getController();

 auctionCardViewController.init(viewHandler,
     viewModelFactory.getAuctionCardViewModel(), innerRoot);

 return auctionCardViewController;
}
```

5. Testing

The project incorporated numerous functionalities and features, necessitating continuous testing throughout the development process. Testing was systematically performed after the introduction of each new feature to ensure the software's reliability and performance. This practice aligns with established software engineering principles, which recommend repeated testing to quickly find and fix problems (Reis, S., Metzger, A., & Pohl, K., n.d.). Various methodologies were used, including White Box testing, Black Box testing, and JUnit testing with ZOMB+E. These methodologies were applied to ensure comprehensive validation and robust software performance.

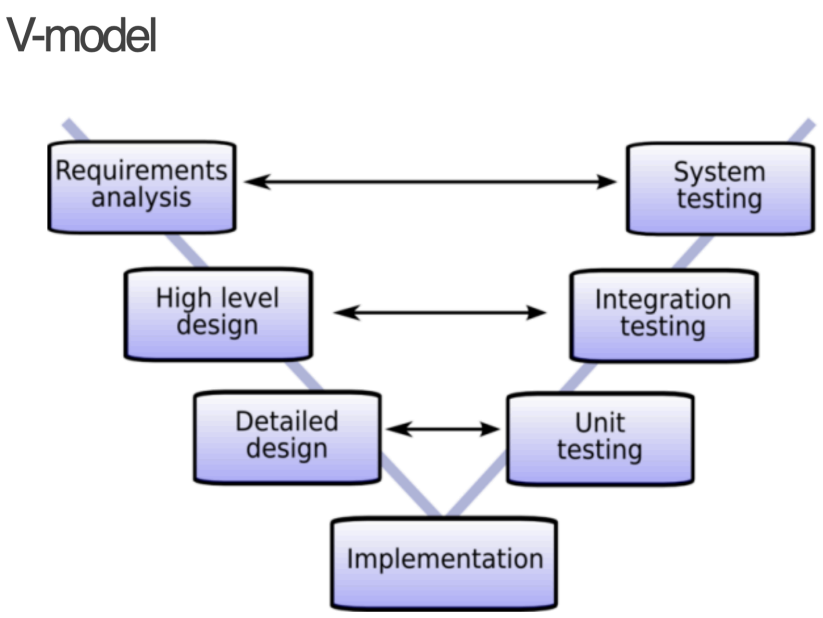


Figure 5.1.1. *V-model*, Source: Bruyninckx, 2008

To meet all the requirements, the V-Model was employed. The process began with defining system requirements and creating the design. This was followed by writing and compiling the code. Finally, the testing phase commenced.

5.1 jUnit testing

The project employed ZOMB+E testing methodology to ensure comprehensive test coverage. However, certain functionalities, such as auction creation and timer, posed challenges for direct testing due to their dependencies on external factors like image uploads and threading. For these functionalities, additional precautions were taken, such as synchronizing all the methods that may be invoked by more clients at the same time and storing the local path to images, transportation as arrays of bytes, followed by the reconstruction on the client side.

Only the most interesting methods of the Black box testing are presented below. They are categorized using the ZOMB+E methodology, covering zero, one, many, boundary, and exceptional cases:

Z - Zero:
Z1: Remove element at index 0;
Z2: Remove element with ID 0;
O - One:
O1: Remove element at index 1;
O2: Remove element with ID 1;
M - Many:
M1: Remove three auctions from list with five auctions;

M2: Remove three auctions from list with two auctions;
B - Boundaries:
B1: Remove at size()-1 index
B2: Remove at size() index
E - Exceptions: doesn't throw any;

```
class AuctionListTest
{

 private AuctionList auctions;
     private     Auction     auction1     =     new     Auction(1,     "teeest",
"teeeeeeeeeeeeeeeeeeeeest", 1, 1, 1, null, null, 1, null, new byte[] {},
      "test");
     private     Auction     auction2     =     new     Auction(2,     "teeest",
"teeeeeeeeeeeeeeeeeeeeest", 2, 2, 2, null, null, 2, null, new byte[] {},
      "test");
     private     Auction     auction3     =     new     Auction(3,     "teeest",
"teeeeeeeeeeeeeeeeeeeeest", 3, 3, 3, null, null, 3, null, new byte[] {},
      "test");

 @BeforeEach void setUp()
 {
   auctions = new AuctionList();

   auctions.addAuction(auction1);
   auctions.addAuction(auction2);
   auctions.addAuction(auction3);
 }

 // One - remove one
 @Test public void remove_One_Auction_Removes_From_List()
 {
   int sizeBefore = auctions.getSize();

   auctions.removeAuction(auction1);

   assertEquals(auctions.getSize(), sizeBefore - 1);
 }
```

The remove_One_Auction_Remove_From_List() removes auction from the list, in this case auction1.

The final size of the auctions list is compared to the initial size minus one using assertEquals. This ensures that exactly one auction has been removed. By focusing on the input-output relationship and ignoring the internal details, this test method follows black-box testing principles.

```
 // Many - remove many
 @Test public void remove_Two_Auctions_Removes_Two_Auctions_From_List()
 {
   int sizeBefore = auctions.getSize();

   auctions.removeAuction(auction1);
   auctions.removeAuction(auction2);

   assertEquals(auctions.getSize(), sizeBefore - 2);
 }
```

The removeAuction method is called twice, first to remove auction1 and then to remove auction2. The final size of the auctions list is compared to the initial size minus two using assertEquals. This ensures that exactly two auctions have been removed.

```
 // Zero - Remove auction with non-existing ID
 @Test
 void remove_Auction_NonExisting_ID() {
   int sizeBefore = auctions.getSize();
   auctions.removeAuction(100); // No auction with id 100
   assertEquals(sizeBefore, auctions.getSize());
 }

 // One - remove one auction with an existing ID
 @Test
 void remove_One_Auction_Existing_ID() {
   int sizeBefore = auctions.getSize();
   auctions.removeAuction(1);

   assertEquals(sizeBefore - 1, auctions.getSize());
   assertThrows(IllegalArgumentException.class, () ->
auctions.getAuctionByID(1));
 }
```

In the remove_Auction_NonExisting_ID() example, the removeAuction method is called with ID 100, which does not exist in the list.

The removeAuction method in the remove_One_Auction_Existing_ID() is called with ID 1, which exists in the list.

```
 @Test
 void remove_Auction_Invalid_ID() {
   assertThrows(IllegalArgumentException.class, () ->
auctions.removeAuction(-1));
   assertThrows(IllegalArgumentException.class, () ->
auctions.removeAuction(0));
   assertThrows(IllegalArgumentException.class, () ->
auctions.removeAuction(100));
 }
```

In the “Auction List” test example, the remove_Auction_Invalid_ID() method tests the behavior of the system when attempting to remove an auction with an invalid ID. This method tests for various scenarios such as attempting to remove an auction with a negative ID, zero ID, and an ID that does not exist. It ensures that the system throws the appropriate IllegalArgumentException in each of these cases.

As for the White Box testing, it has been decided to test the password resetting feature.

```
@Test
public void test_Reset_Password_Button_Pressed_EditProfile_Confirmation_Ok()
{
 when(headerLabel.getText()).thenReturn("Edit profile");
 when(errorLabel.getText()).thenReturn("");
 mockAlertConfirmation(true);

 controller.resetPasswordButtonPressed();

 verify(viewModel).deleteAccount();
 verify(viewModel).setForLogin();
 verify(viewHandler).openView(WindowType.LOG_IN);
 assertEquals("", errorLabel.getText());
}
```

The testResetPasswordButtonPressed_EditProfile_Confirmation_Ok method mocks headerLabel.getText() to "Edit profile," errorLabel.getText() to an empty string, and simulates clicking OK on the alert.

```
@Test
```

```
public void
test_Reset_Password_Button_Pressed_EditProfile_Confirmation_Cancel() {
 when(headerLabel.getText()).thenReturn("Edit profile");
 mockAlertConfirmation(false);

 controller.resetPasswordButtonPressed();

 verify(viewModel, never()).deleteAccount();
 verify(viewModel, never()).setForLogin();
 verify(viewHandler, never()).openView(WindowType.LOG_IN);
 assertNotEquals("", errorLabel.getText());
}
```

The testResetPasswordButtonPressed_EditProfile_ConfirmationCancel method mocks headerLabel.getText() to "Edit profile" and simulates clicking Cancel on the alert. It verifies that viewModel.deleteAccount(), viewModel.setForLogin(), and viewHandler.openView(WindowType.LOG_IN) are never called. Both of these tests are white-box tests, involving mocking internal methods and verifying internal logic and interactions within the method.

5.2 System testing

Testing of the "Create Auction" use case functionality took priority in alignment with the project's requirement order. For this functionality, there are a total of 250 scenarios (125+125). However, only the crucial examples will be presented to illustrate the key aspects of the testing process. Other Unit tests can be found in the *Appendices* folder.

Step	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
1. Enter data								
1 a) Title	**R**	**R**	**R**	**S**	**L**	**R**	**R**	**R**
1 b) Description	**R**	**R**	**R**	**R**	**R**	**S**	**L**	**R**
1 c) Reserve price	**R**	**E**	**R**	**R**	**R**	**R**	**R**	**R**
1 d) Buyout price	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
1 e) Minimum bid	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**R**
1 f) Duration	**R**	**R**	**N**	**R**	**R**	**R**	**R**	**R**
1 g) Image	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**E**

R - regular
E - empty
N - null
S - short
L - long

Table 5.2.1 *"Start Auction" Test cases*, own production

In the example above (**Table 5.2.1**) - R (regular) means that the value is within the given boundaries, for example title must contain minimum 5, maximum 80 characters. With these boundaries, it can be assumed that S(short) means less than 5 characters, and L(long) means more than 80 characters.

When creating an auction, the seller must take into account all alternate sequences from the “Start Auction” Use case description (**Appendix A**).

TC1 – Successful Auction creation

Step	Input Field	Value	Expected result	Actual result
1. Enter data				
	1 a) Title	Iphone 15	Valid	Valid
	1 b) Description	Iphone 15 in very good condition, bought in Bilka in January 2024	Valid	Valid
	1 c) Reserve price	100	Valid	Valid
	1 a) Buyout price	1200	Valid	Valid
	1 b) Minimum bid	5	Valid	Valid
	1 c) Duration	24	Valid	Valid
	1 d) Image	Uploaded	Valid	Valid
2.Confirm data		Confirm	Confirmed	Confirmed
3.System validates data			Validated	Validated
4.System creates an auction			Created	Created

Table 5.2.2 *Successful Auction creation*, own production

In the previous example (**Table 5.2.2**) the auction was successfully created, because all criteria were met and the system validated the values.

TC2 – Empty Reserve price value

Step	Input Field	Value	Expected result	Actual

				result
1. Enter data				
	1 a) Title	Iphone 15	Valid	Valid
	1 b) Description	Iphone 15 in very good condition, bought in Bilka in January 2024	Valid	Valid
	1 c) Reserve price		Not valid	Not Valid
	1 a) Buyout price	1200	Valid	Valid
	1 b) Minimum bid	5	Valid	Valid
	1 c) Duration	24	Valid	Valid
	1 d) Image	Uploaded	Valid	Valid
2.Confirm data		Confirm	Confirmed	Confirmed
3.System validates data			Not Valid	Not Valid
4.System creates an auction			Not Created	Not Created

Table 5.2.3 *Empty Reserve price*, own production

At the same time, in the example above (**Table 5.2.3**) the system did not confirm and validate the data due to the empty reserve price. When creating an auction all fields must contain information and there must be an image. So every time there is an empty field, the system shows an error “The (specific field) can not be empty”.

TC3 – Invalid Duration

Step	Input Field	Value	Expected result	Actual result
1. Enter data				
	1 a) Title	Iphone 15	Valid	Valid
	1 b) Description	Iphone 15 in very good condition, bought in Bilka in January 2024	Valid	Valid

	1 c) Reserve price	10 $	Valid	Valid
	1 a) Buyout price	1200 $	Valid	Valid
	1 b) Minimum bid	5 $	Valid	Valid
	1 c) Duration	25	Not Valid	Not Valid
	1 d) Image	Uploaded	Valid	Valid
2.Confirm data		Confirm	Confirmed	Confirmed
3.System validates data			Not Valid	Not Valid
4.System creates an auction			Not Created	Not created

Table 5.2.4 *Invalid duration*, own production

In the Invalid duration example, the system won't create an auction if the duration exceeds 24 hours. This check was made based on a customer's request, ensuring that auctions last no longer than 24 hours as desired by the moderator.

Use case	Actor	Expected Result	Actual Result
Create account	Guest	The actor should be able to create an account by providing first name, last name, date of birth, contact information (email address, phone number) and password.	Works as expected
Start auction	Seller	The actor should be able to start an auction with auto generated ID, by uploading a picture and filling in the information (title, description, reserve price, buyout price, minimum bid increment, duration).	Works as expected
	Seller	The actor should be able to access all their ongoing and closed auctions, and see the ID, title, the highest bid, picture and the end time for each auction.	Works as expected
	Moderator	The actor should be able to delete an item from auction by notifying and providing the Seller with a reason.	Works as expected
Participate in auction	Bidder	The actor should access the list of all ongoing auctions, showing the ID, title, highest bid, picture and end time for each auction.	Works as expected
	Bidder, Moderator	The actors should be able to open an auction and have access to all the necessary information about	Works as expected

		the item, including the title, description, reserve price, buyout price, minimum bid increment, auction time, picture, highest bid and highest bidder.	
	Bidder	The actor should be able to participate in an ongoing auction by offering at least the reserve price, if no other bids exist or increasing the highest bid (if they are not the highest bidder) by at least the minimum bid increment.	Works as expected
	Bidder	The actor should be able to buy an auction item at the specified buyout price if no bids have been placed on it within a specified time period.	Works as expected
	Bidder, Moderator	The actors should be able to search for ongoing auctions using the IDs or words present in the titles.	Works as expected
	Bidder	The actor should be able to access all the ongoing and closed auctions and see the ID, title, the highest bid, picture and the end time for each auction where they previously placed a bid or bought the item.	Works as expected
	Bidder	The actor should be able to receive a real-time notification with the auction ID when their bid is beaten for a specific item.	Works as expected
Manage participant account	Participant	The actor should be able to log in to their account using email address and password.	Works as expected
	Participant	The actor should be able to log out from their account.	Works as expected
	Participant	The actor should be able to reset their password by entering the old password and the new password.	Works as expected
	Participant	The actor should be able to delete their account.	Works as expected
	Participant	The actor should be able to edit the information in their account, including the first name, last name, email address, phone number and date of birth.	Works as expected
	Participant	The actor should be able to access the chronological list of notifications, starting with the most recent ones, which includes timestamps and notification contents.	Works as expected
Manage Moderator Account	Moderator	The actor should be able to log in to their account using a provided email address and password.	Works as expected

	Moderator	The actor should be able to log out from their account.	Works as expected
	Moderator	The actor should be able to reset their password by entering the old password and the new password.	Works as expected
	Moderator	The actor should be able to set and display their contact information (including first name, last name, email address and phone number).	Works as expected
Manage participants	Moderator	The actor should be able to search for a Participant by their email address.	Works as expected
	Moderator	The actor should be able to ban a Participant, provide a reason for banning, and have the reason displayed to the Participant when they attempt to log in and get the system to automatically log them out, delete their auctions, regardless of status, and their bids.	Works as expected
	Moderator	The actor should be able to access the list of accounts, with their first name, last name, email address and phone number.	Works as expected
	Moderator	The actor should be able to unban and allow a previously banned Participant to log back into their account.	Works as expected
Manage auctions	Moderator	The actor should be able to prohibit Sellers from placing bids on their own auctions, to prevent them from artificially increasing the prices for the items.	Works as expected
	Moderator	The actor should be able to access all the ongoing and closed auctions, including the ID, title, highest bid, picture and end time for each auction.	Works as expected
	Moderator	The actor should be able to delete an item from auction by notifying and providing the Seller with a reason.	Works as expected
Notify Participants	Bidder and Seller	The actors should be notified and receive each other's contact information (including first name, last name, email address and phone number) and the auction ID after the auction time is over.	Works as expected

Table 5.2.5 *Platform result overview*, own production

In conclusion, the testing phase played a pivotal role in ensuring the reliability, functionality, and user-friendliness of the following project. The detailed use cases provided a comprehensive overview of the testing process, demonstrating that each feature was thoroughly evaluated and met the expected outcomes.

All buttons were pressed, and all input fields were tested as outlined in the use case test cases, all features were tested after making new changes to the code. Of course, there were multiple bugs and errors while adding new features, as for example the Timer was not stopping while the Seller or Buyer wanted to check the auction that was Bought-away (its status was marked as CLOSED). According to **Table 5.2.5,** it can be stated that all requirements were successfully met.

6. Results and Discussion

The functional requirements specified for the auction application have been successfully met through comprehensive implementation. The system facilitates various types of users, including Guests, Sellers, Bidders, Participants and Moderators, to interact with the platform efficiently. Key results include:

Auction Creation and Management:

- **Auction Creation:** Sellers can easily create auction deals with auto-generated IDs, upload pictures, provide detailed item information (title, description) and set price constraints (reserve price, buyout price, minimum bid increment). Auctions have set durations, with an automatic countdown timer.
- **Auction Overview:** Sellers and Bidders can view detailed lists of ongoing and finished auctions, including auction ID, title, picture, highest bid and time left.
- **Auction Participation:** Bidders can place bids that meet or exceed the reserve price or the highest bid's value plus the minimum increment. They receive notifications when outbid.

Account Creation and Management:

- **Account Creation and Editing:** Guests over 18 can create accounts with personal details and edit their information as needed.
- **Login and Logout:** Participants can securely log in and out of their accounts.

Moderator Functions:

- **Auction Monitoring:** Moderators can view all auctions to ensure compliance with platform rules, ban participants, delete inappropriate items and provide reasons for their actions.

- **User Management:** Moderators can manage participants, investigating their activity and banning or unbanning users based on behavior and compliance.

System Features:

- **Search and Notifications:** Bidders can search for items using keywords or item IDs and receive notifications when outbid.
- **Buyout Option:** Bidders can buy items at the buyout price if no bids have been placed, bypassing the auction process.

The system's functionality aligns with the requirements and ensures a seamless user experience across different roles. The database storage strategy, support for JPEG images and the use of cache proxies enhance performance and contribute to a robust, user-friendly interface.

7. Conclusion

In conclusion, it can be stated that the expectations and needs of the various user roles—Guests, Sellers, Bidders, Participants, and Moderators—were successfully fulfilled. The system enables seamless auction creation, bidding processes, user account management, and moderator oversight, aligning well with the envisioned requirements. The needs of all user types were addressed, ensuring a functional and user-friendly multi-user auction platform.

The project was developed by applying the knowledge, methods, and theories learned during coursework. The system’s functionality, including the responsive design and database integration, ensures that it operates as intended across various devices, providing a smooth and efficient user experience.

It can be confidently stated that the primary goal of the project was achieved. The auction application not only meets the specified requirements but also provides a solid foundation for potential future enhancements. The successful implementation of this project demonstrates the effectiveness of the educational program in equipping students with the necessary skills for real-world applications.

8. Project Future

Even after meeting all the customer requirements, there is always room for improvement and enhancement. The auction application, while functional and meeting the initial specifications, can benefit from several future enhancements to improve user experience and expand its capabilities.

- **User Interface Improvements:** Minor improvements can be made to the user interface to ensure a professional appearance when displaying a large number of auctions, enhancing the visual appeal of the platform.
- **Rating System:** Introducing a detailed rating system for users can enhance trust and provide valuable feedback.
- **Category Management:** Adding refined categories for auctions can help users find items more easily, improving their overall experience.
- **Chat System Integration:** Incorporating a chat system within the platform can facilitate better communication between buyers and sellers, and provide immediate support to users.
- **Payment Method Integration:** A significant future improvement involves integrating a secure and efficient payment method within the app. Implementing an in-app payment gateway would streamline the process, allowing users to complete transactions directly within the platform. This feature would include:
    - Multiple Payment Options: Supporting various payment methods such as credit/debit cards, PayPal, and other popular payment gateways.
    - Secure Transactions: Ensuring that all financial transactions are encrypted and secure, protecting users' sensitive information.
    - Automated Payment Processes: Automatically handling payments once an auction is won, reducing the need for manual intervention and minimizing the risk of fraud.

Moreover, with some adjustments, the auction system can be easily transformed and used as a shopping platform:

- **Fixed Pricing**: Only allowing fixed prices (the buyout feature) for items instead of allowing users to place bids. Each item would only have the set price determined by the seller (buyout price), facilitating immediate purchase without waiting for an auction to end.
- **Product Listings**: Modify the product listing process to include more detailed descriptions, multiple pictures and possibly videos. This would help buyers make informed decisions without the competitive context of bidding.
- **Inventory Management**: Implement inventory management features for sellers to track available stock, update quantities, and manage out-of-stock items

By focusing on these improvements and future enhancements, the auction application can continue to evolve, providing an even better experience for its users and opening up new opportunities for expansion and innovation.

Sources of information

Bruyninckx, H. (2008, March 7). *English: V model from structured systems design*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:V-model.svg

Davies, M., Spalding, K., & Chubbock, T. (2017). *The Auction Process: Advantages and Disadvantages and the Key Steps*. (p. 13) https://www.globalprivatecapital.org/app/uploads/2017/10/The-Auction-Process.pdf

Federal Communications Commission. (2017, December 11) *How is an auction conducted?* . https://www.fcc.gov/how-auction-conducted

Kas, J., Corten, R., & van de Rijt, A. (2023). *Trust, reputation, and the value of promises in online auctions of used goods*. *Rationality and Society*, *35*(4), 387-419. https://doi.org/10.1177/10434631231170342

Krishna, V. (2002), *Auction Theory*, San Diego, USA: Academic Press (p. 1), ISBN 978-0-12-426297-3

Reis, S., Metzger, A., & Pohl, K. (n.d.). *Integration Testing in Software Product Line Engineering: A Model-Based Technique. Fundamental Approaches to Software Engineering*. https://doi.org/10.1007/978-3-540-71289-3_25

Appendices

Appendix A - Use case descriptions

Use case	Start auction
Summary	The actor is allowed to start a new auction sale by entering information about their item, setting the time auction and applying the price constraints.
Actor	Seller
Precondition	The Seller must be logged in.
Postcondition	The auction is started and open for participation.
Base sequence	1. Enter data: a) Title b) Description c) Reserve price d) Buyout price e) Minimum bid increment; f) Auction duration g) Picture 2. Confirm entered data. **[ALT1]** 3. System validates data. **[ALT2][ALT3][ALT4][ALT5][ALT6][ALT7][ALT8]** 4. System creates the auction with given data and displays the new auction with the generated ID and stores it in the database.
Alternate sequence	**[*ALT0]**The process can be canceled at any point with user interaction. The unconfirmed data will not be saved. Use case ends. **[ALT1]** If the data is not confirmed, the auction is not started and the data is not saved. Use case ends. **[ALT2]** If the Title is shorter than 5 or longer than 80 characters, the System shows an error message. Go to Step 1a. **[ALT3]** If the Description is shorter than 20 or longer than 1400 characters, the System shows an error message. Go to Step 1b.

	[ALT4] If the Reserve price is not given as a positive integer, the System shows an error message. Go to Step 1c. **[ALT5]** If the Buyout price is not given as a positive integer or it is not greater than the reserve price, the System shows an error message. Go to Step 1d. **[ALT6]** If the Minimum bid increment is not given as a positive integer, the System shows an error message. Go to Step 1e. **[ALT7]** If the Auction duration is not set or it is not given as a positive integer between 1 and 24, the System shows an error message. Go to Step 1f. **[ALT8]** If no Picture was entered, the System shows an error message. Go to Step 1g.
Notes	This use case covers requirements 1, 26.

Use case	Create account
Summary	The actor is allowed to create an account after entering personal information.
Actor	Guest
Precondition	none
Postcondition	The account is created and the actor is automatically logged in.
Base sequence	1. Enter data: a) First name b) Last name c) Email address d) Phone number e) Date of birth f) Password 2. Confirm the entered data. **[ALT3]** 3. System validates data. **[ALT4][ALT5][ALT6][ALT7][ALT8][ALT9]** 4. System creates the account with the specified data and stores it in the database. 5. The actor is logged in.
Alternate sequence	**[*ALT0]** Processes can be canceled at any point with user interaction. The unapproved information will not be saved. Use case ends. **[ALT1]** If the information is not confirmed, go to Step 1. **[ALT2]** If the account with the given information doesn't exist, the System shows an error message. Go to Step A1. **[ALT3]** If the entered data is not confirmed, the account is not created and the data is not saved. Go to Step 1.

	[ALT4] If the First name is shorter than 3 or longer than 100 characters, the System shows an error message. Go to Step 1a. **[ALT5]** If the Last name is shorter than 3 or longer than 100 characters, the System shows an error message. Go to Step 1b. **[ALT6]** If the Email address doesn't have a valid format (*local-part@domain*) or an account linked to it already exists, the System shows an error message. Go to Step 1c. **[ALT7]** If the Phone number doesn't have a valid format (length between 6 and 20 characters, all digits) or it is already linked to another account, the System shows an error message. Go to Step 1d. **[ALT8]** If the Date of birth is not a legal date with day, month and year, which determines that the actor is at least 18 years old, the System shows an error message. Go to Step 1e. **[ALT9]** If the Password is shorter than 8 or longer than 255 characters, the System shows an error message. Go to Step 1f.
Note	This use case covers Requirement 7.

Use case	Manage participant account
Summary	The actor can log in, log out, edit their information, reset their password, delete their account or access the list of notifications and the history regarding their auctions and bids
Actor	Participant
Precondition	none
Postcondition	The actor logged in or out, edited or deleted their account, reset their password or accessed the history or the list of notifications.
Base sequence	1. If LOGIN, go to **Scenario A LOGIN**, Step A1. 2. If LOG OUT, go to **Scenario B LOG OUT**, Step B1. 3. If EDIT, go to **Scenario C EDIT INFO**, Step C1. 4. If RESET, go to **Scenario D RESET PASSWORD**, Step D1. 5. If DELETE, go to **Scenario E DELETE ACCOUNT**, Step E1. 6. If AUCTIONS, System displays all ongoing and closed auctions started by the actor, including the ID, Title, Highest bid, End time and Picture for each auction. 7. If BIDS, System displays all ongoing and closed auctions where the actor placed a bid or bought out the item, including the ID, Title, Highest bid, End time and Picture for each auction. 8. If NOTIFICATIONS, System displays the list of the notifications received by the actor, including the Timestamps and the Contents, starting with the most recent ones.

Scenario A LOGIN	A1. Enter the Email address and the Password A2. Confirm the entered data. **[ALT1]** A3. System validates data **[ALT2][ALT3]** A4. The actor is logged in. Go to Step 2.

Scenario B LOG OUT	B1. Confirm the logging out action. **[ALT4]** B2. The actor is logged out. Go to Step 1.

Scenario C EDIT INFO	C1. System displays all information in the account: First name, Last name, Email address, Phone number or Date of birth. C2. Confirm the intention to edit **[ALT5]** C3. Edit the existing fields for First name, Last name, Email address, Phone number or Date of birth C4. Enter the Password. C5. Confirm the changes. **[ALT6]** C6. System validates data.**[ALT7][ALT8][ALT9][ALT10][ALT11][ALT12]** C7. System updates the account information in the database. If the email address has been changed, the System updates all auctions where the actor is the Seller or the Highest bidder. Go to Step 2.

Scenario D RESET PASSWORD	D1. System displays all information in the account: First name, Last name, Email address, Phone number or Date of birth. D2. Confirm the intention to reset the password **[ALT13]** D3. Enter the Old password and the New password and repeat the New password. D4. Confirm the reset. **[ALT14]** D5. System validates data. **[ALT15][ALT16][ALT17]** D6. System updates the password linked to the account in the database and automatically logs out the actor on all devices. Go to Step 1.

Scenario E DELETE ACCOUNT	E1. System displays all information in the account: First name, Last name, Email address, Phone number or Date of birth. E2. Confirm the intention to delete the account. **[ALT18]** E3. Enter the Password. E4. System validates the password.**[ALT19]**

	E5. Confirm the account deletion.**[ALT20]** E6. System deletes the account, all the started auctions and updates all auctions where the actor previously placed a bid in the database. Use case ends.
Alternate sequence	**[*ALT0]** Processes can be canceled at any point with user interaction. The unconfirmed information will not be saved. Use case ends. **[ALT1]** If the entered information is not confirmed, the actor will not log in. Use case ends. **[ALT2]** If the Email and the Password do not match any account, System shows an error message. Go to Step A1. **[ALT3]** If the actor has been banned by the Moderator, System shows an error message with the reason. Use case ends. **[ALT4]** If the actor doesn't confirm the logging out intention, the actor will not be logged out. Go to Step 2. **[ALT5]** If the actor doesn't confirm the intention to edit, go to Step 2. **[ALT6]** If the changes are not confirmed, the account information is not updated. Go to Step C3. **[ALT7]** If the password doesn't match the account, System shows an error message. Go to Step C4. **[ALT8]** If the First name is shorter than 3 or longer than 100 characters, the System shows an error message. Go to Step C3. **[ALT9]** If the Last name is shorter than 3 or longer than 100 characters, the System shows an error message. Go to Step C3. **[ALT10]** If the Email address doesn't have a valid format (*local-part@domain*) or an account linked to it already exists, the System shows an error message. Go to Step C3. **[ALT11]** If the Phone number doesn't have a valid format (length between 6 and 20 characters, all digits) or it is already linked to another account, the System shows an error message. Go to Step C3. **[ALT12]** If the Date of birth is not a legal date with day, month and year, which determines that the actor is at least 18 years old, the System shows an error message. Go to Step C3. **[ALT13]** If the intention to reset the password is not confirmed, go to Step 2. **[ALT14]** If the data is not confirmed, the password is not reset, go to Step D1. **[ALT15]** If the Old password doesn't match the account, System displays an error message. Go to Step D3. **[ALT16]** If the New password doesn't match the repeated New password, System displays an error message. Go to Step D3. **[ALT17]** If the New password is the same as the Old password, System shows an error message. Go to Step D3. **[ALT18]** If the intention to delete the account is not confirmed, go to Step 2. **[ALT19]** If the Password doesn't match the account, System displays an error message. Go to Step E3.

	[ALT20] If the account deletion is not approved, the account is not deleted. Go to Step 2.
Notes	This use case covers Requirements 6, 8, 10, 13, 14, 16, 24, 27

Use case	Manage participants
Summary	The actor can access the list of Participants, search for a Participant by email, ban or unban them.
Actor	Moderator
Precondition	The actor must be logged in.
Postcondition	The actor viewed the list of accounts, searched for a Participant, banned or unbanned them.
Base sequence	1. System displays the list of all Participants, with their First name, Last name, email address and phone number. 2. If SEARCH, go to **Scenario A SEARCH**, Step A1. 3. If BAN, go to **Scenario B BAN**, Step B1. 4. If UNBAN, go to **Scenario C UNBAN**, Step C1.

Scenario A SEARCH	A1. Enter the Email address of the Participant. A2. System displays the Participant with the matched email address. Go to Step 2.

Scenario B BAN	B1. Select the account to be banned. B2. Enter the Reason. B3. Confirm the banning. **[ALT1]** B4. System validates the data. **[ALT2][ALT3]** B5. System deletes all auctions and the bids placed by the selected Participant and adds the Participant to the list of the banned users in the database, along with the reason. B6. System shows a message that the Participant has been banned.

Scenario C UNBAN	C1. Select the account to be unbanned. C2. System displays the reason for banning. C3. Confirm the unbanning. **[ALT4]** C4. System validates data.**[ALT5]** C5. System deletes the selected Participant from the list of banned users in the database.

	<table><tr><td></td><td>C6. System displays a message that the Participant has been unbanned.</td></tr></table>
Alternate sequence	**[*ALT0]** Processes can be canceled at any point with user interaction. The unconfirmed information will not be saved. Use case ends. **[ALT1]** If the banning is not confirmed, the Participant is not banned. Go to Step 1. **[ALT2]** If the reason is not between 3 and 600 characters, System shows an error message. Go to Step B2. **[ALT3]** If the selected Participant is already banned, System displays an error message. Go to Step 1. **[ALT4]** If the unbanning is not approved, the Participant is not unbanned. Go to Step 1. **[ALT5]** If the selected Participant is not banned, System displays an error message. Go to Step C1.
Notes	This use case covers Requirements 20, 21, 22, 25.

Use case	Manage auctions
Summary	The actor can access the list of ongoing and closed auctions, search for an auction by ID or word present in the titles, see all information about an auction or delete an item.
Actor	Moderator
Precondition	The actor must be logged in.
Postcondition	The actor viewed all auctions, searched for one or deleted items from sale.
Base sequence	1. System displays the list of all ongoing and closed auctions, including the ID, Title, Picture, Highest bid and End time. 2. If SEARCH, go to **Scenario A SEARCH**, Step A1. 3. Select an auction. 4. System displays the selected auction and all its details: ID, Title, Description, Reserve price, Buyout price, Minimum bid increment, Picture, Highest bidder, Highest bid, auction Time left for participating and Seller's email address. 5. If DELETE, go to **Scenario B DELETE ITEM**, Step B1. <table><tr><td>**Scenario A SEARCH**</td><td>A1. Enter ID or at least one word for searching through the auction Titles.</td></tr></table>

	A2. System displays all auctions that contain the specified word in their Title or ID, including the ID, Picture, Title, Highest bid and End time. Go to Step 3.
	Scenario B DELETE ITEM: B1. Enter the Reason for deleting the auction. B2. Confirm the deletion. **[ALT1]** B3. System validates the data. **[ALT2]** B4. System deletes the auction from the database and sends a notification to the Seller, including the auction ID and the Reason for deletion. Go to Step 1.
Alternate sequence	**[*ALT0]** Processes can be canceled at any point with user interaction. The unconfirmed information will not be saved. Use case ends. **[ALT1]** If the deletion is not confirmed, the auction is not deleted. Go to Step 4. **[ALT2]** If the Reason is not between 3 and 600 characters, System shows an error message. Go to Step B1.
Notes	This use case covers Requirements 3, 12, 19, 23.

Use case	Manage moderator account
Summary	The actor can log in, log out, set and display their contact information or reset their password.
Actor	Moderator
Precondition	none
Postcondition	The actor logged in or out, set their information or reset their password.
Base sequence	1. If LOGIN, go to **Scenario A LOGIN**, Step A1. 2. If LOG OUT, go to **Scenario B LOG OUT**, Step B1. 3. If SET, go to **Scenario C SET INFO**, Step C1. 4. If RESET, go to **Scenario D RESET PASSWORD**, Step D1. **Scenario A LOGIN**: A1. Enter the Email address and the Password A2. Confirm the entered data. **[ALT1]** A3. System validates data **[ALT2]** A4. The actor is logged in. Go to Step 2.

<table>
<tr><td></td><td>
<table>
<tr><td>Scenario B LOG OUT</td><td>B1. Confirm the logging out action. [ALT3]
B2. The actor is logged out. Go to Step 1.</td></tr>
</table>
<table>
<tr><td>Scenario C SET INFO</td><td>C1. System displays all information in the account: First name, Last name, Email address and Phone number.
C2. Confirm the intention to set the contact information. [ALT4]
C3. Edit the existing fields for First name, Last name, Email address or Phone number.
C4. Enter the Password.
C5. Confirm the changes. [ALT5]
C6. System validates data.[ALT6][ALT7][ALT8][ALT9][ALT10]
C7. System updates the account information in the database and makes it available for the Participants. Go to Step 2.</td></tr>
</table>
<table>
<tr><td>Scenario D RESET PASSWORD</td><td>D1. System displays all information in the account: First name, Last name, Email address and Phone number.
D2. Confirm the intention to reset the password [ALT11]
D3. Enter the Old password and the New password and repeat the New password.
D4. Confirm the reset. [ALT12]
D5. System validates data. [ALT13][ALT14][ALT15]
D6. System updates the password linked to the account in the database and automatically logs out the actor on all devices. Go to Step 1.</td></tr>
</table>
</td></tr>
<tr><td>Alternate sequence</td><td>[*ALT0] Processes can be canceled at any point with user interaction. The unconfirmed information will not be saved. Use case ends.
[ALT1] If the entered information is not confirmed, the actor will not log in. Use case ends.
[ALT2] If the Email and the Password do not match the moderator account, System shows an error message. Go to Step A1.
[ALT3] If the actor doesn’t confirm the logging out intention, the actor will not be logged out. Go to Step 2.
[ALT4] If the actor doesn’t confirm the intention to set the contact information, go to Step 2.
[ALT5] If the changes are not confirmed, the account information is not updated. Go to Step C3.
[ALT6] If the password doesn’t match the account, System shows an error message. Go to Step C4.</td></tr>
</table>

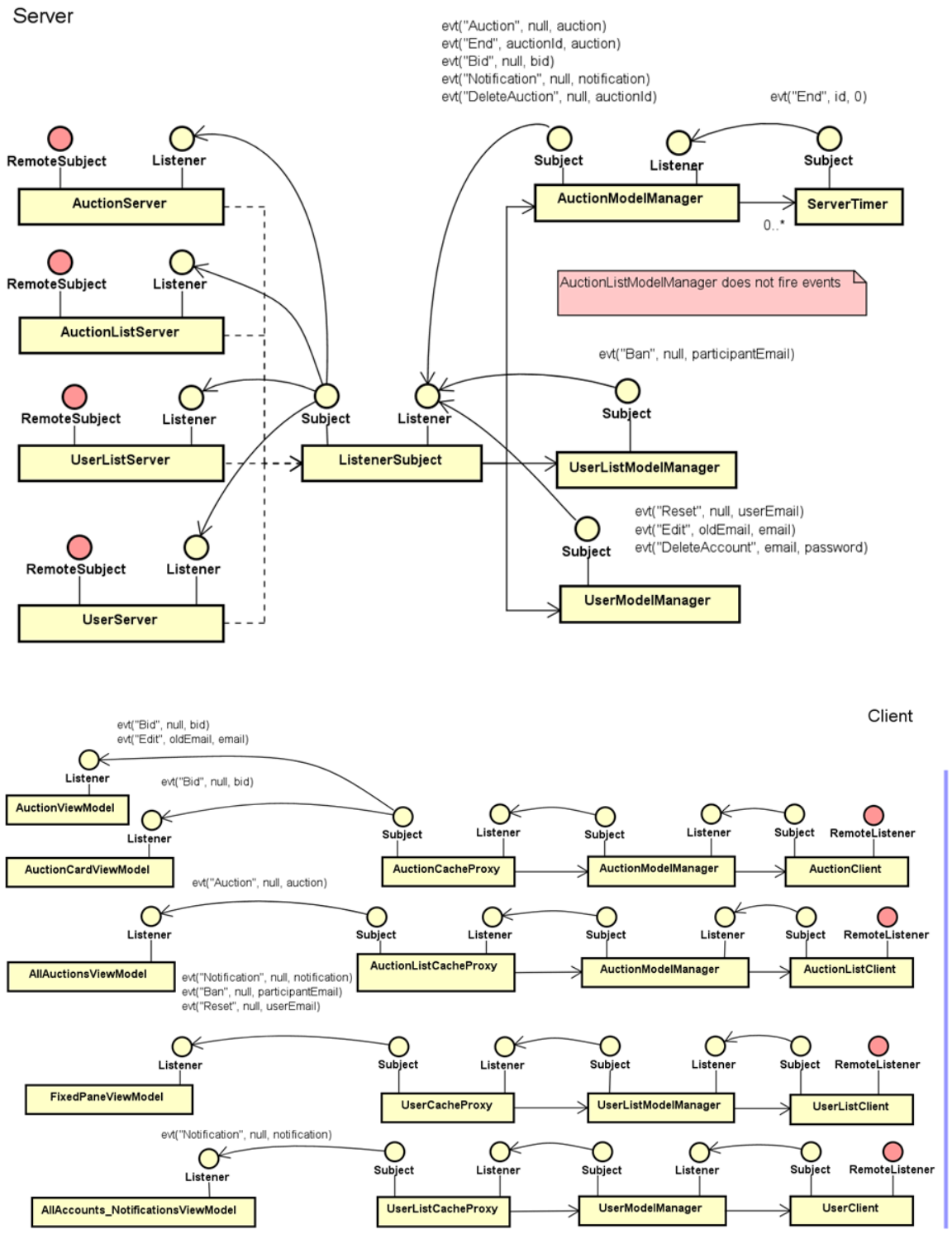
	[ALT7] If the First name is shorter than 3 or longer than 100 characters, the System shows an error message. Go to Step C3. **[ALT8]** If the Last name is shorter than 3 or longer than 100 characters, the System shows an error message. Go to Step C3. **[ALT9]** If the Email address doesn't have a valid format (*local-part@domain*) or an account linked to it already exists, the System shows an error message. Go to Step C3. **[ALT10]** If the Phone number doesn't have a valid format (length between 6 and 20 characters, all digits) or it is already linked to another account, the System shows an error message. Go to Step C3. **[ALT11]** If the intention to reset the password is not confirmed, go to Step 2. **[ALT12]** If the data is not confirmed, the password is not reset, go to Step D1. **[ALT13]** If the Old password doesn't match the account, System displays an error message. Go to Step D3. **[ALT14]** If the New password doesn't match the repeated New password, System displays an error message. Go to Step D3. **[ALT15]** If the New password is the same as the Old password, System shows an error message. Go to Step D3.
Notes	This use case covers Requirements 14, 16, 17, 18.

Use case	Update auction
Summary	The actor calculates the auction end time, updates the auction time each second and notifies the auction's end.
Actor	Timer
Precondition	The System created an auction based on Seller's input
Postcondition	The auction has been updated from the moment it was started to the moment it ended and the auction Participants (if relevant) has been notified.
Base sequence	1. Calculate the moment when the auction ends, based on the auction's Duration set when created. 2. Display updated time left until the auction closes. 3. If it is not the time for the auction to end and the item was not bought out, go to Step 2. 4. Update auction status. 5. If the auction has a Current highest bidder, System notifies auction's creator (Seller) and the Current highest bidder that the auction ended, including each other's contact information in the notifications and saves the notifications in the database.

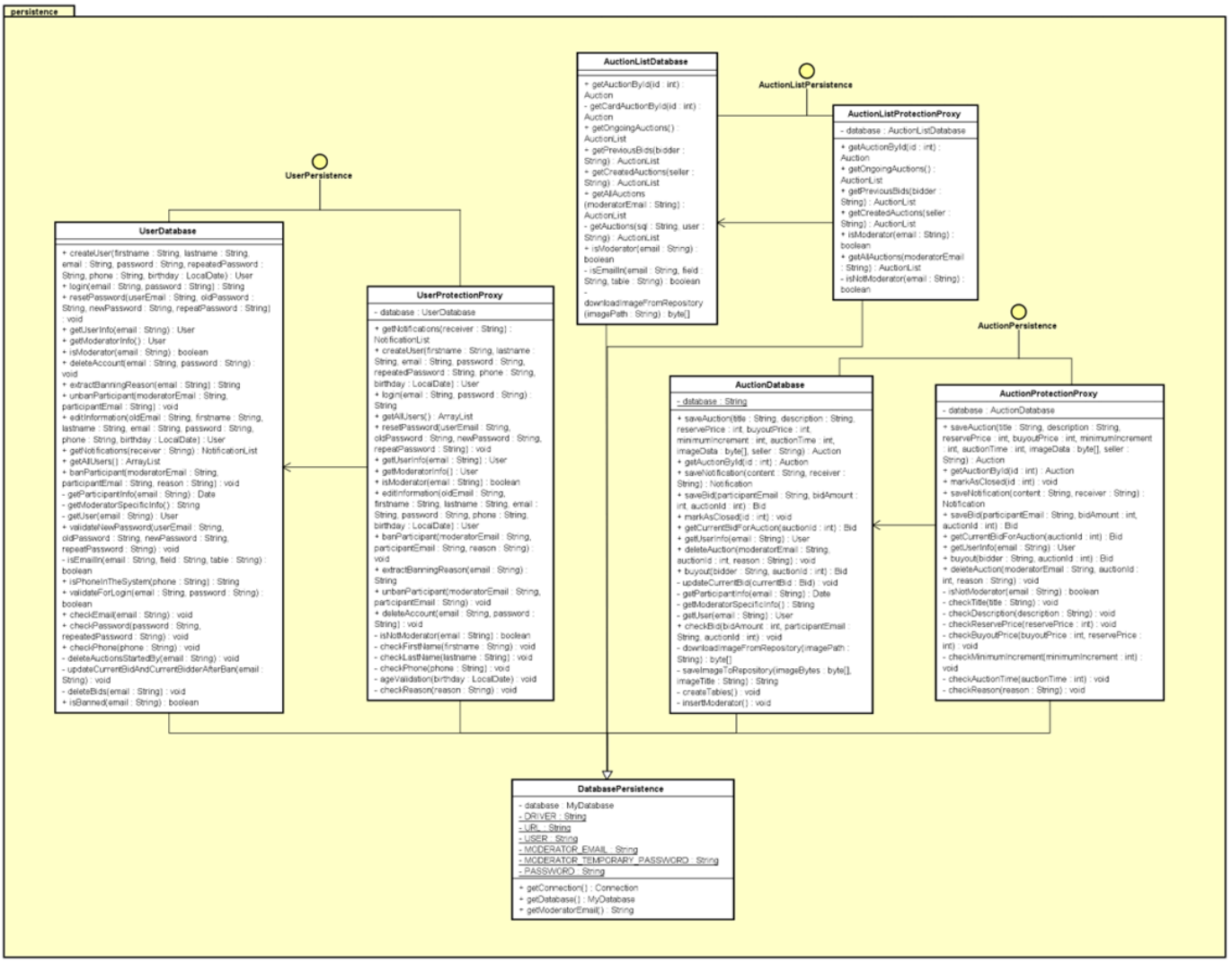
Alternate sequence	
Note	This use case covers Requirements 1, 3, 9, 11.

Appendix B - Detailed class diagrams

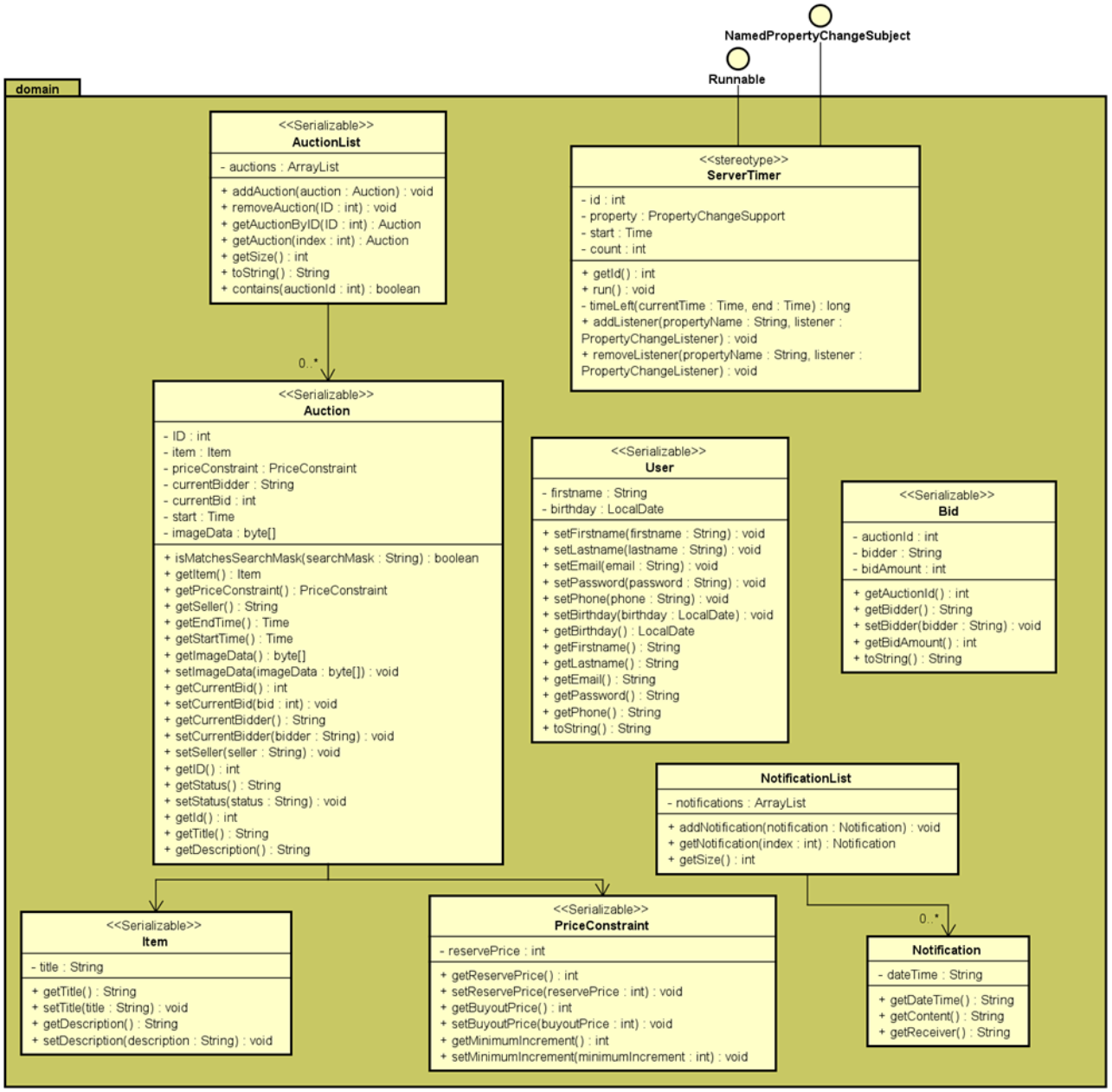
Observer Pattern



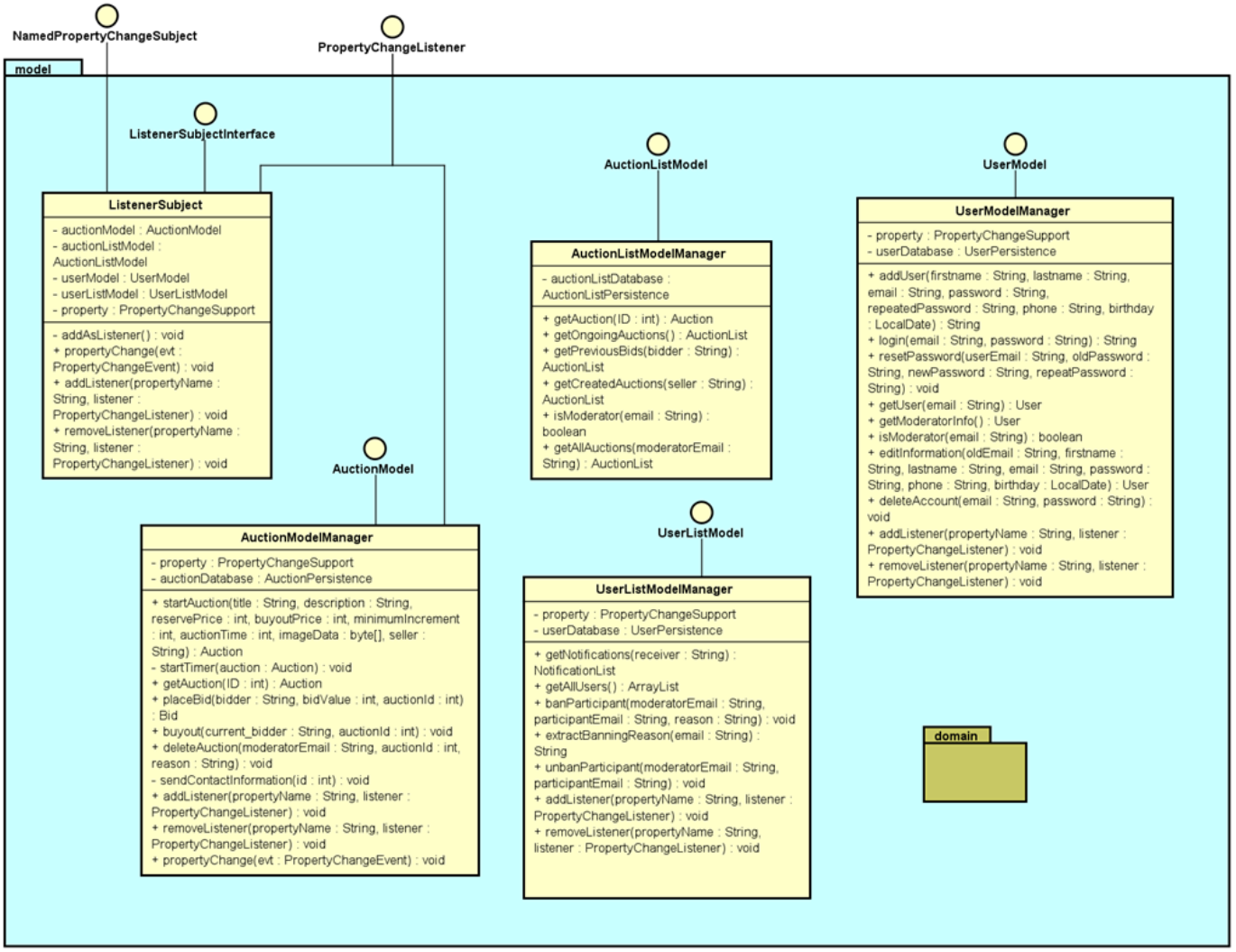
Persistence Package



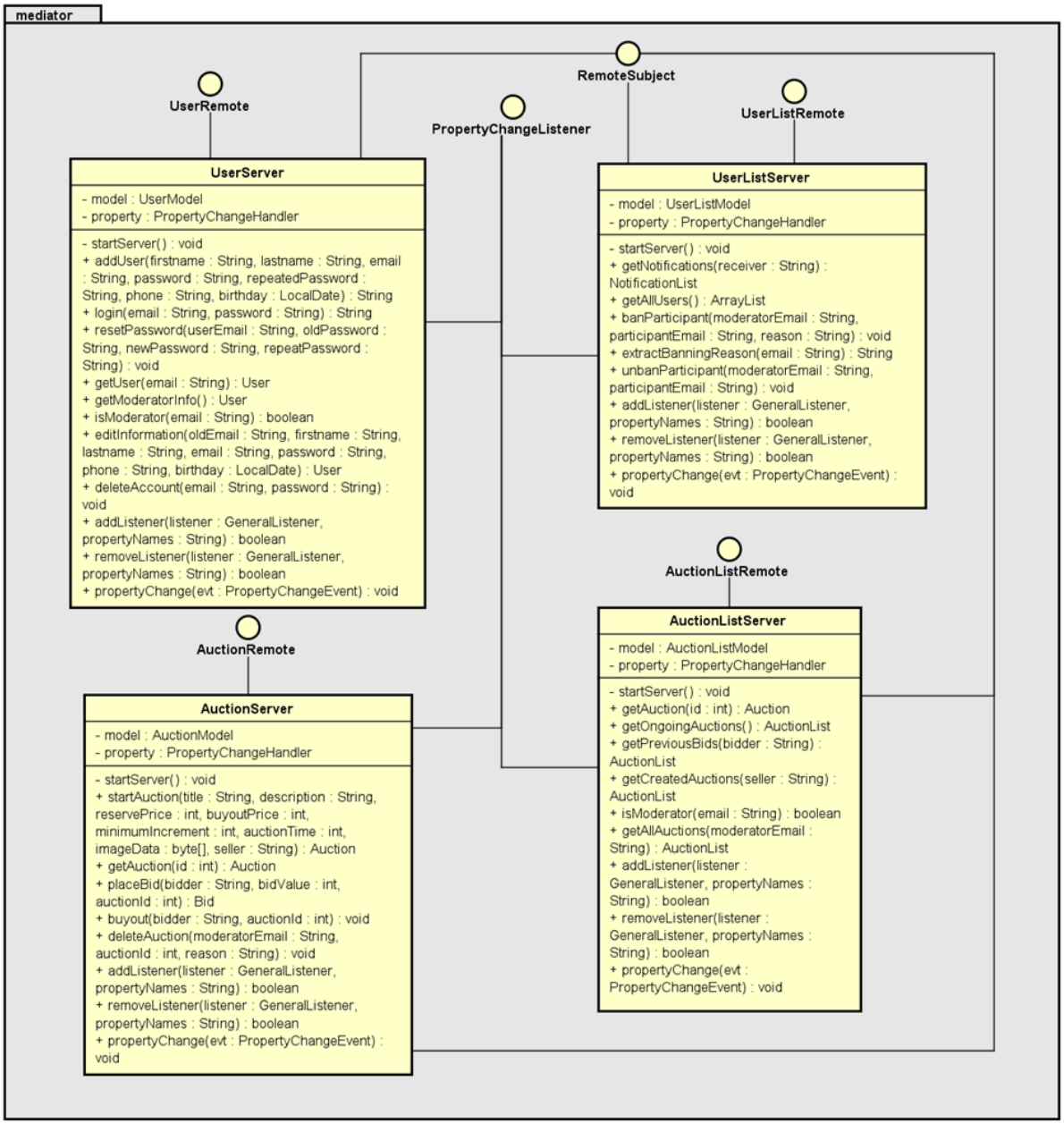
Domain Package



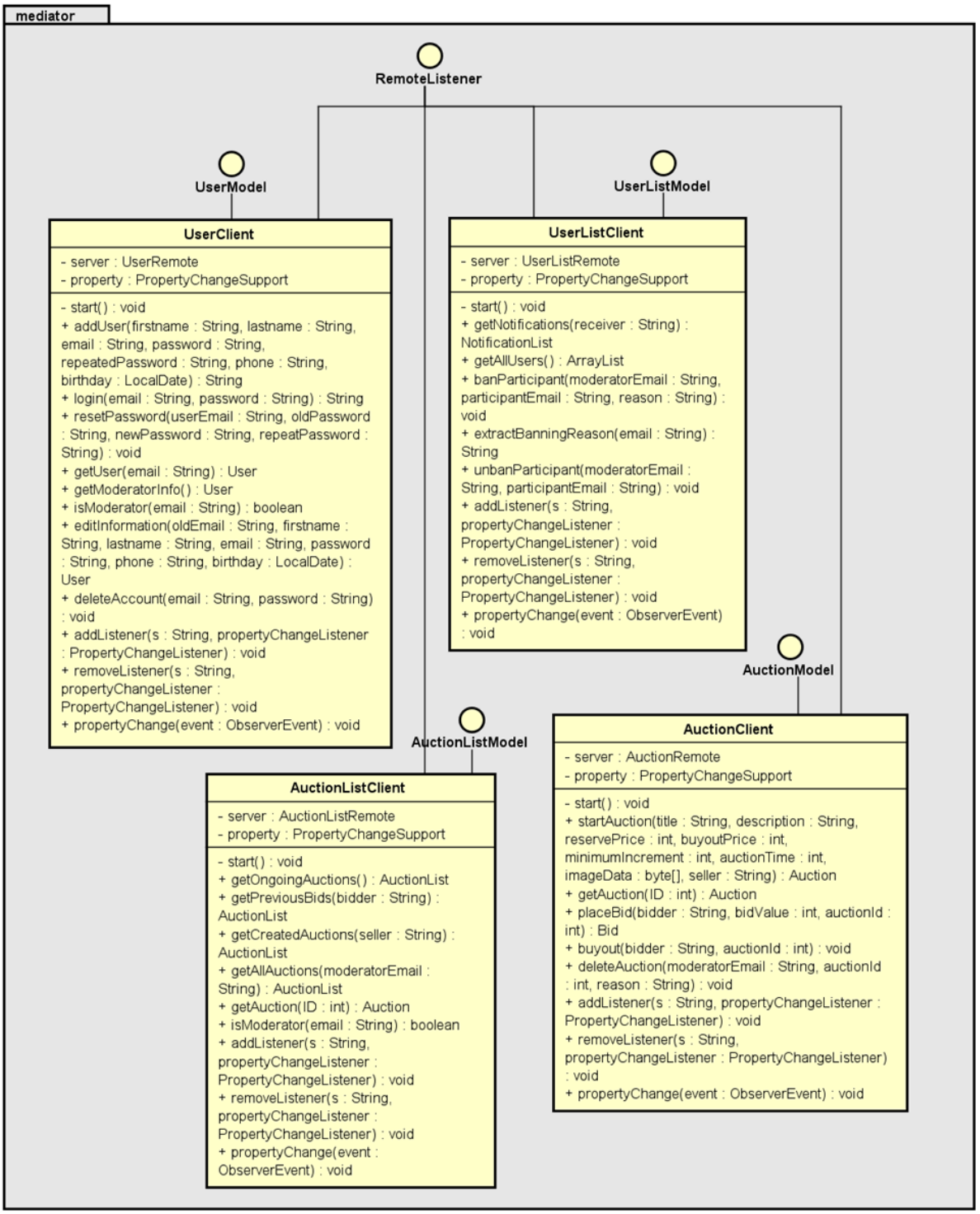
Server-Side Model



Server-Side Mediator



Client-Side Mediator



* rest of the classes can be found in the resource folder

Appendix C - User Guide

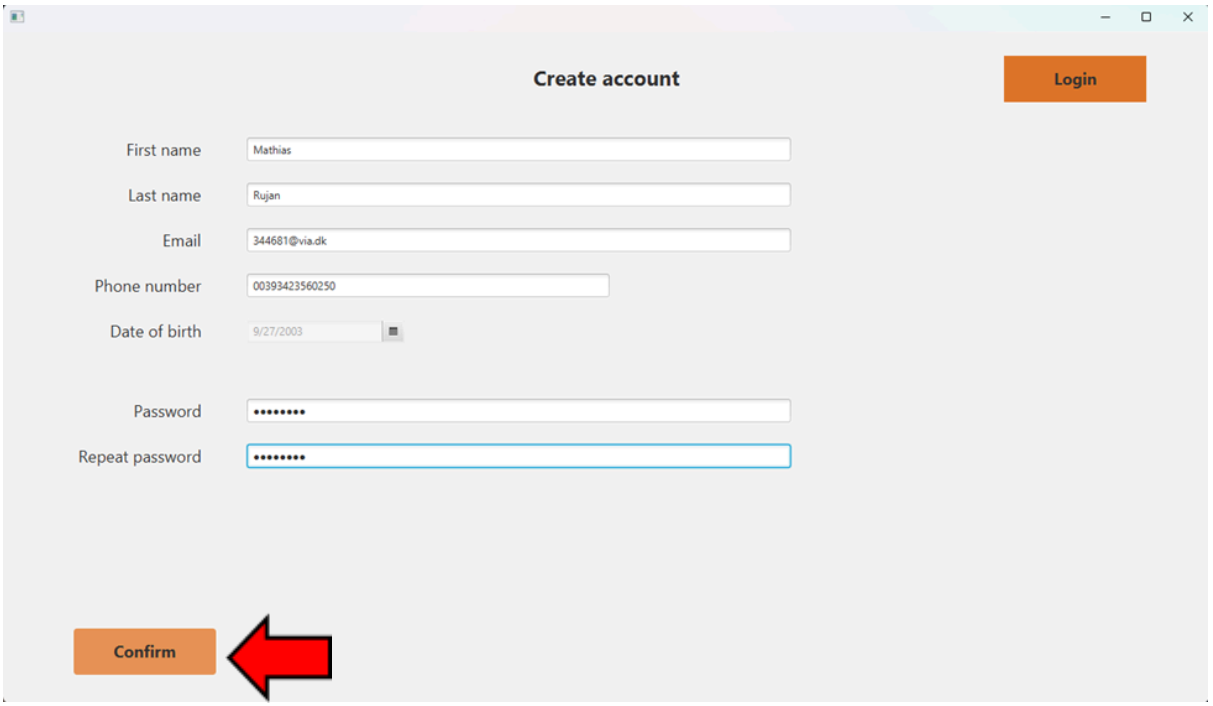
Bidhub ~ User Guide

1. Create Account

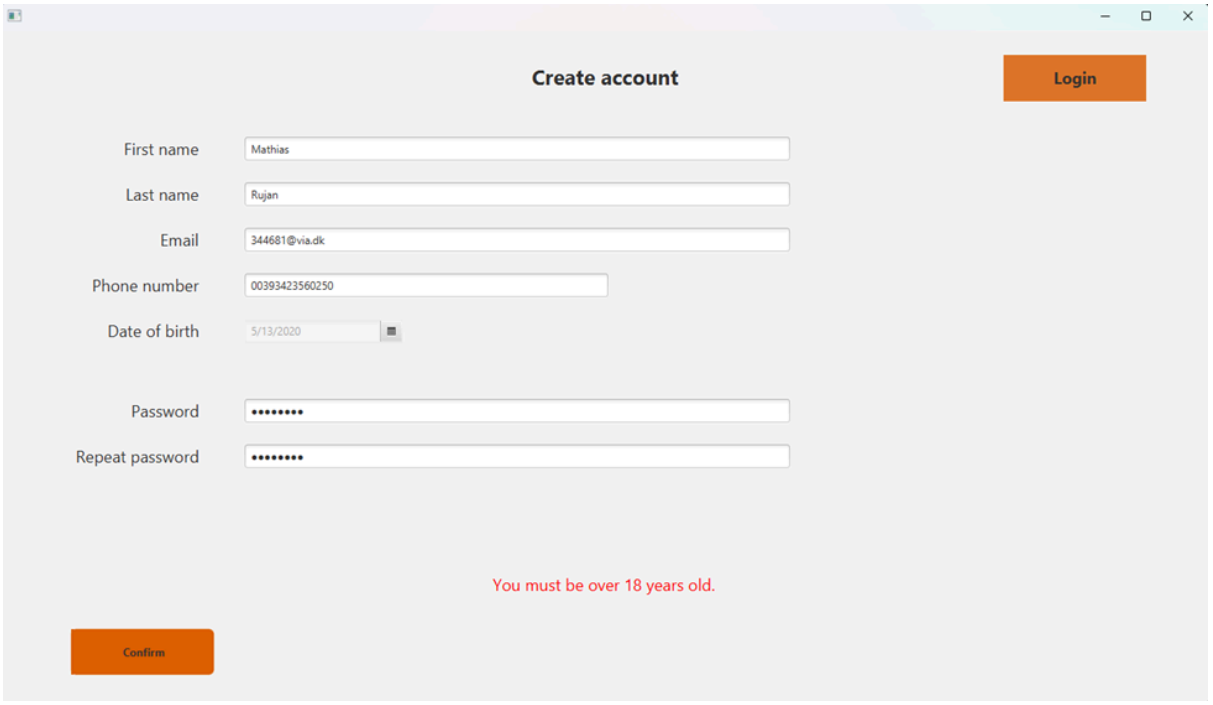
To create an account, follow these steps:

1. Open the application and the "Create Account" pane will pop up.
2. Fill in the personal information fields:
    - First name
    - Last name
    - Email
    - Phone number
    - Date of birth
    - Password
    - Repeat the password

3. Click the “Confirm” button, located at the bottom left.



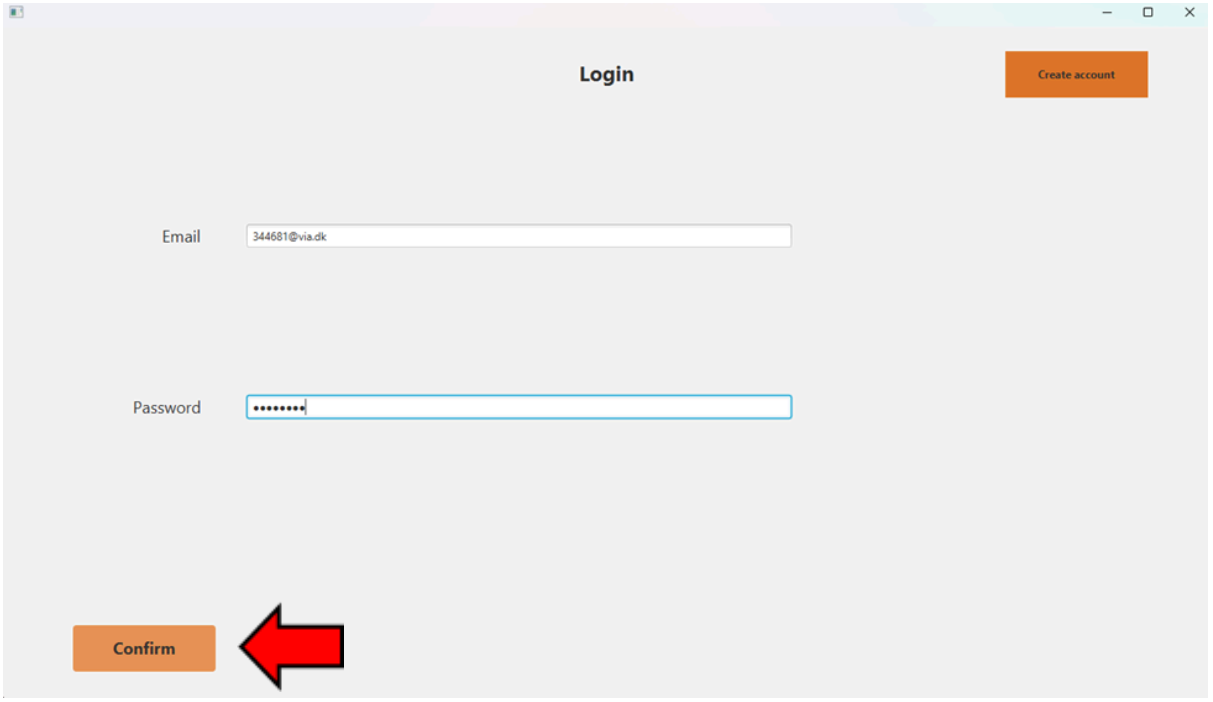
If you enter invalid data, the system will display an error and allow you to fix the values in question. (F.e: the age of the user does not meet the app requirements)



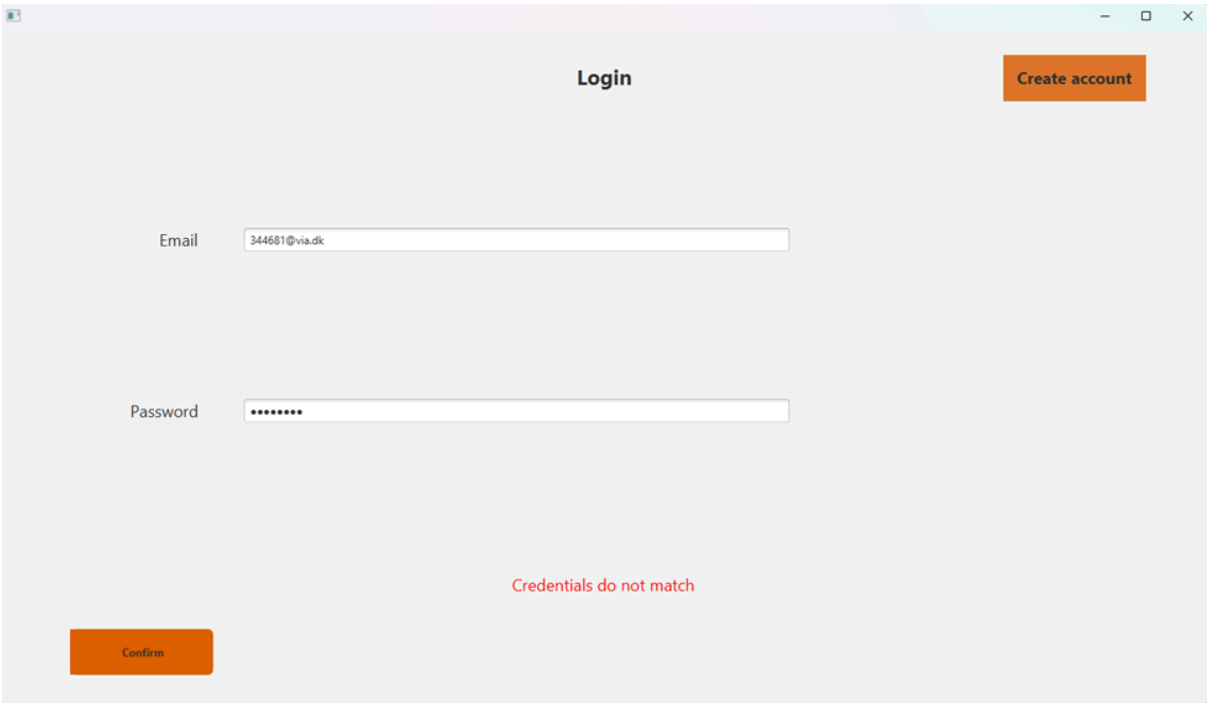
2. Login

To log in to your account:

1. Enter your email and password.



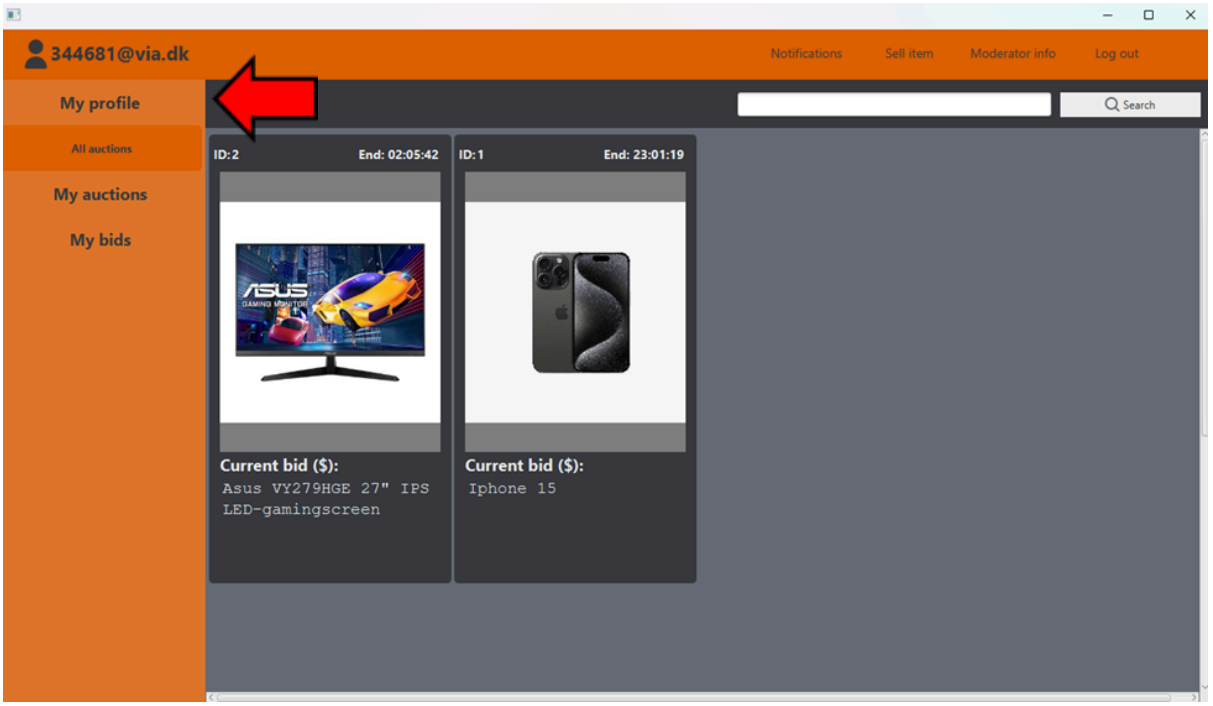
2. Click the “Confirm” button. If you enter invalid data, the system will display an error and allow you to fix the values in question. (F.e: invalid password).



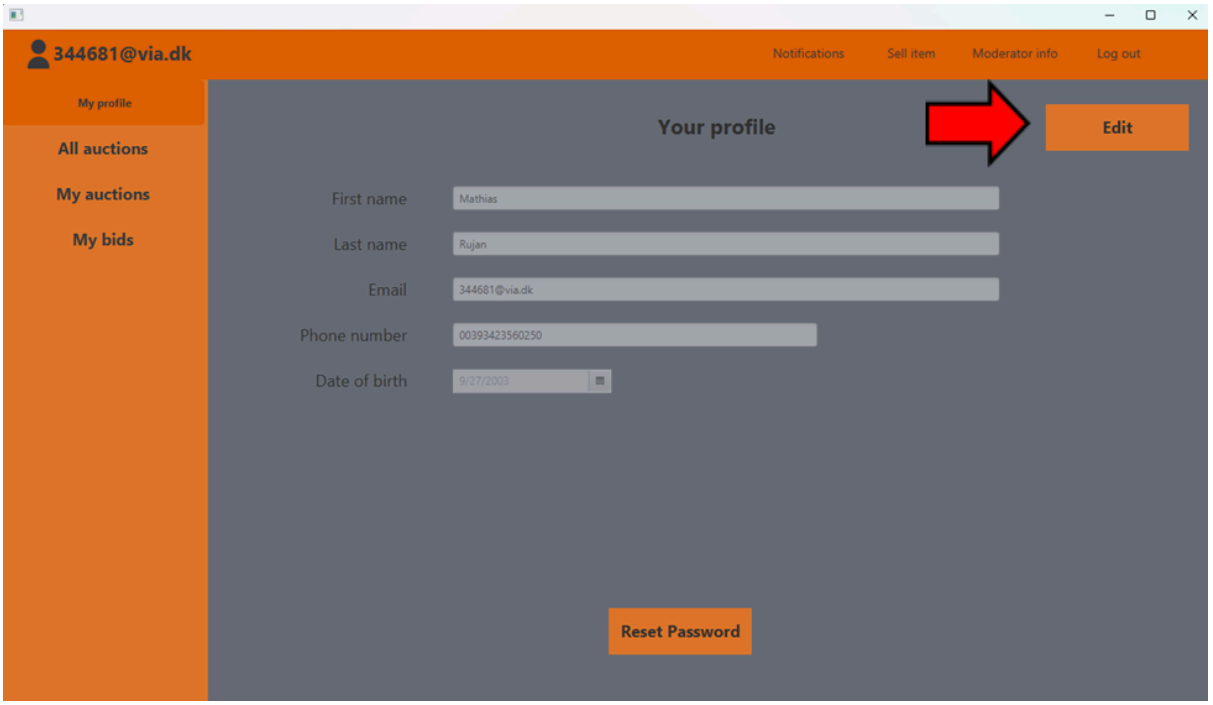
3. Edit Profile

To edit your profile:

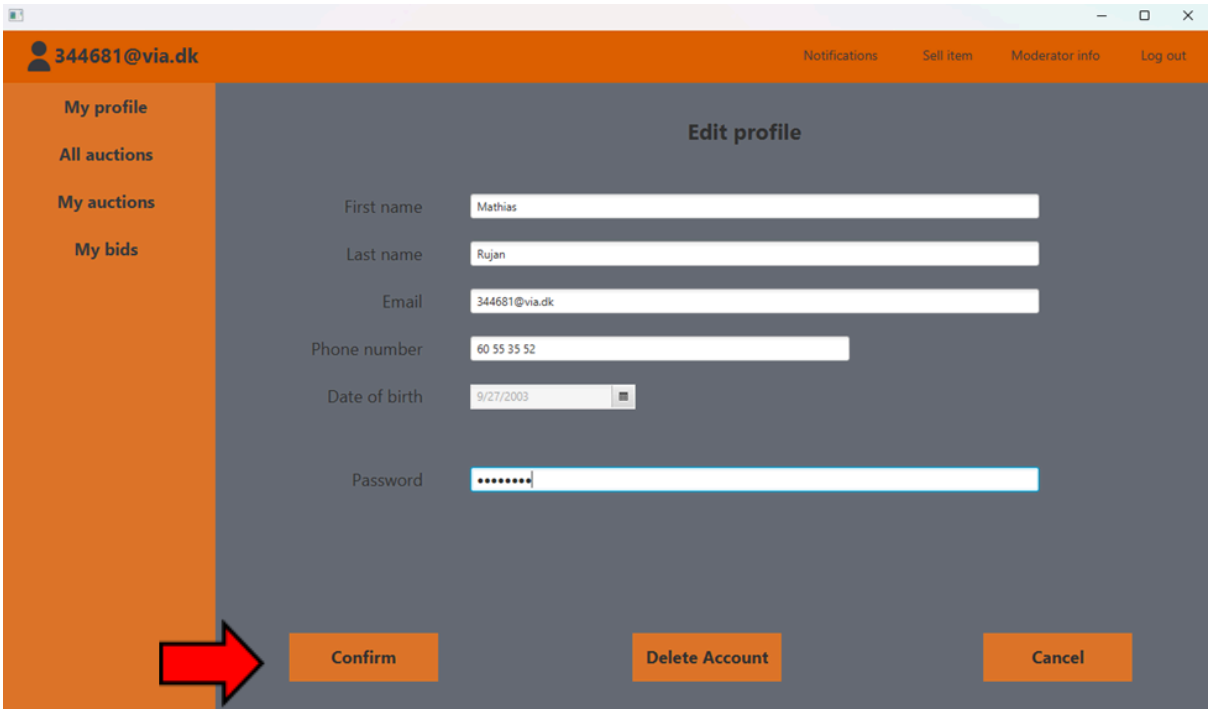
1. Login (see point 2. Login).



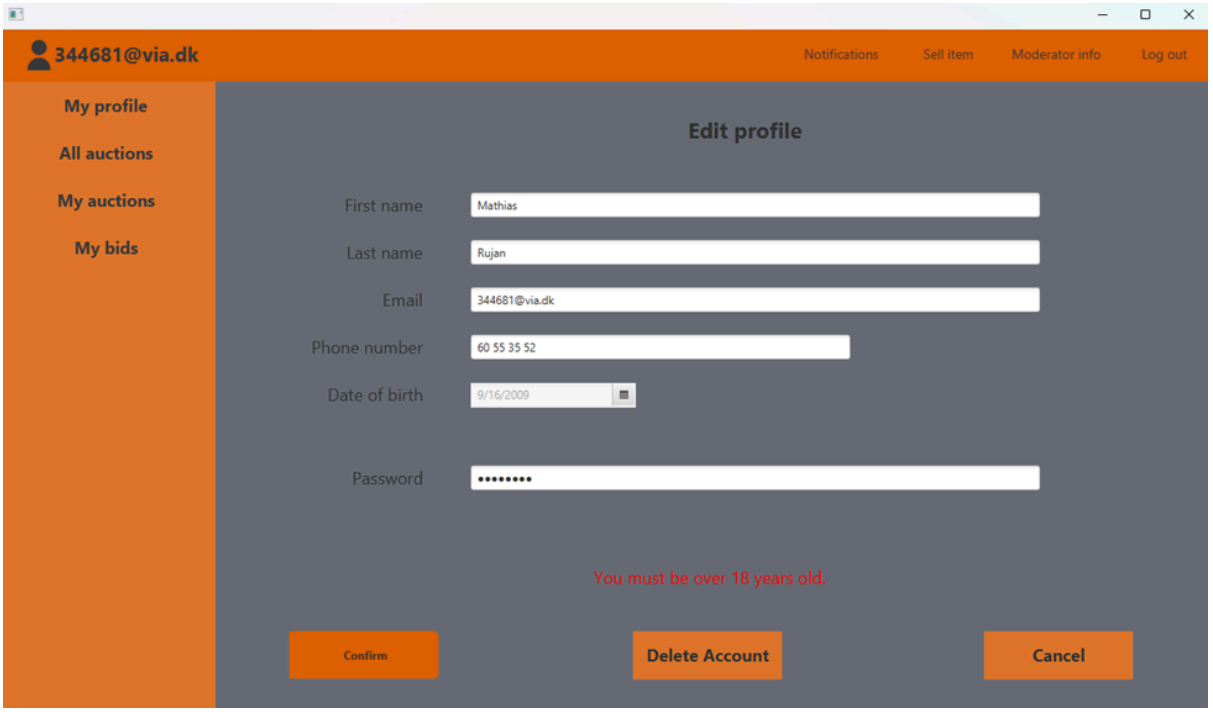
2. Click the “My Profile” button.
3. Click the “Edit” button.



4. Edit the necessary fields. (f.e: changing the phone number)
5. Enter your password.
6. Click the “Confirm” button.



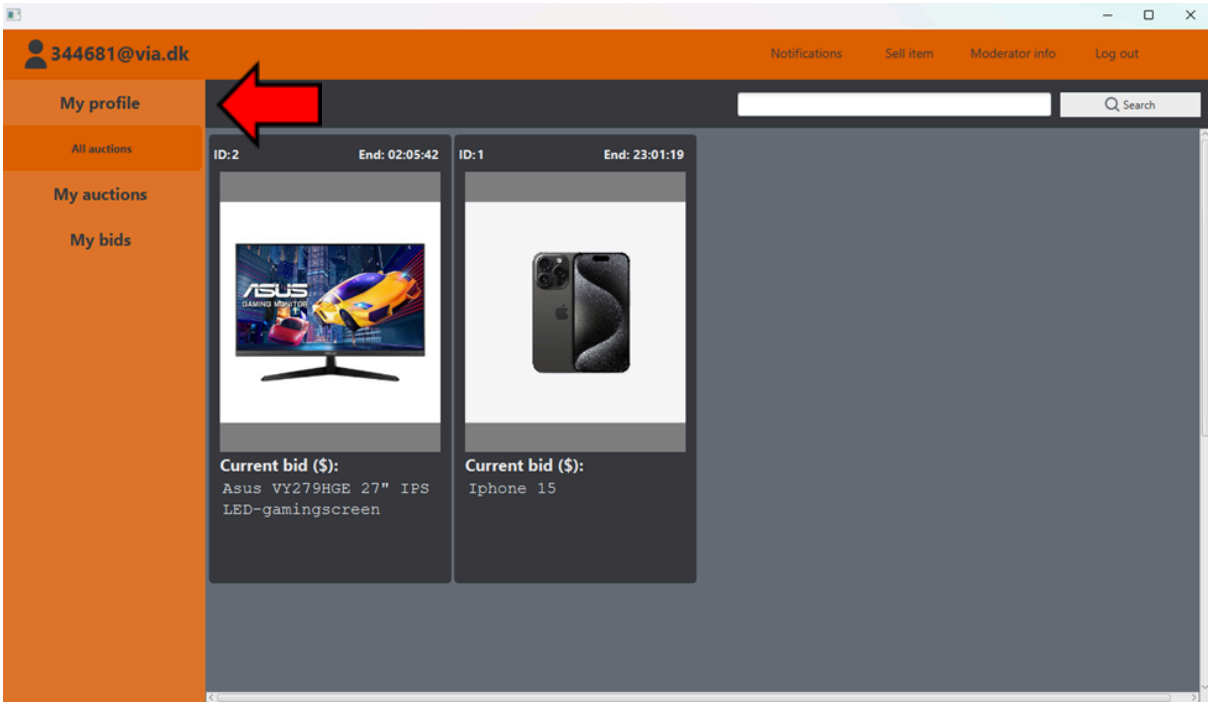
If you enter invalid data, the system will display an error and allow you to fix the values in question. (f.e: age doesn’t meet the requirements)



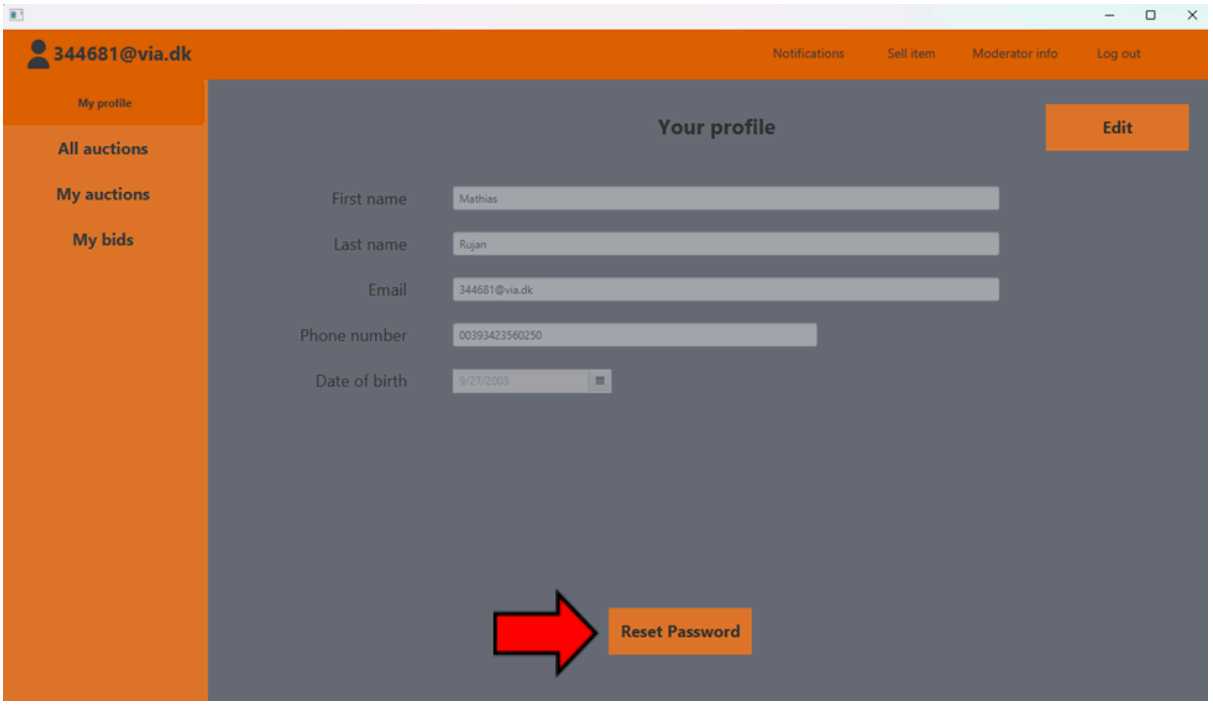
4. Reset Password

To reset your password:

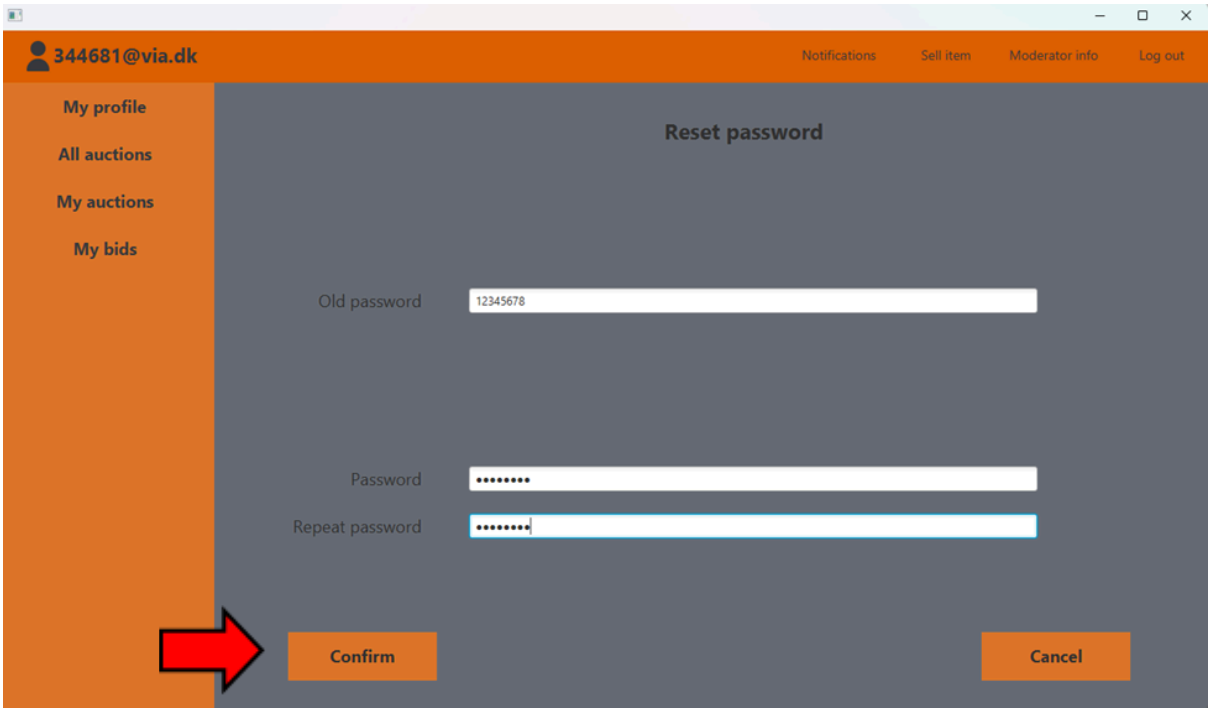
1. Login (see point 2. Login).



2. Click the “My Profile” button.
3. Click the “Reset password” button.



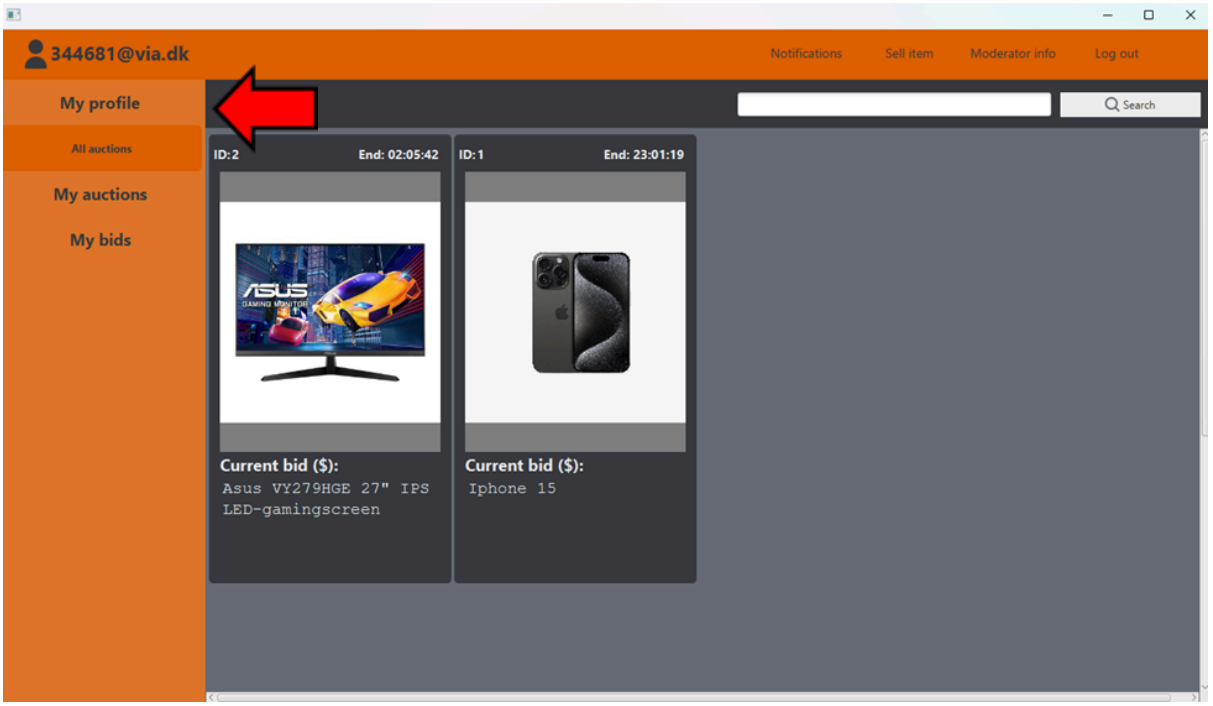
4. Enter your old password.
5. Enter your new password and confirm it by entering a second time; afterwards click “Confirm”



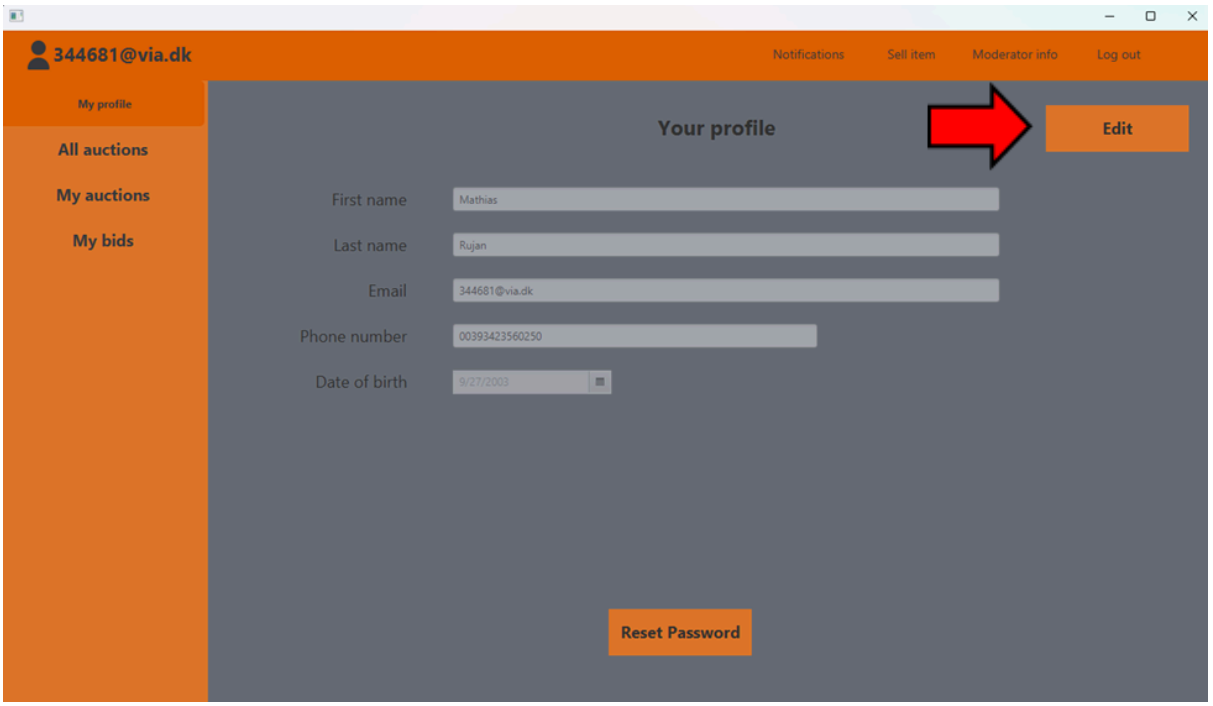
5. Delete Account

To delete your account:

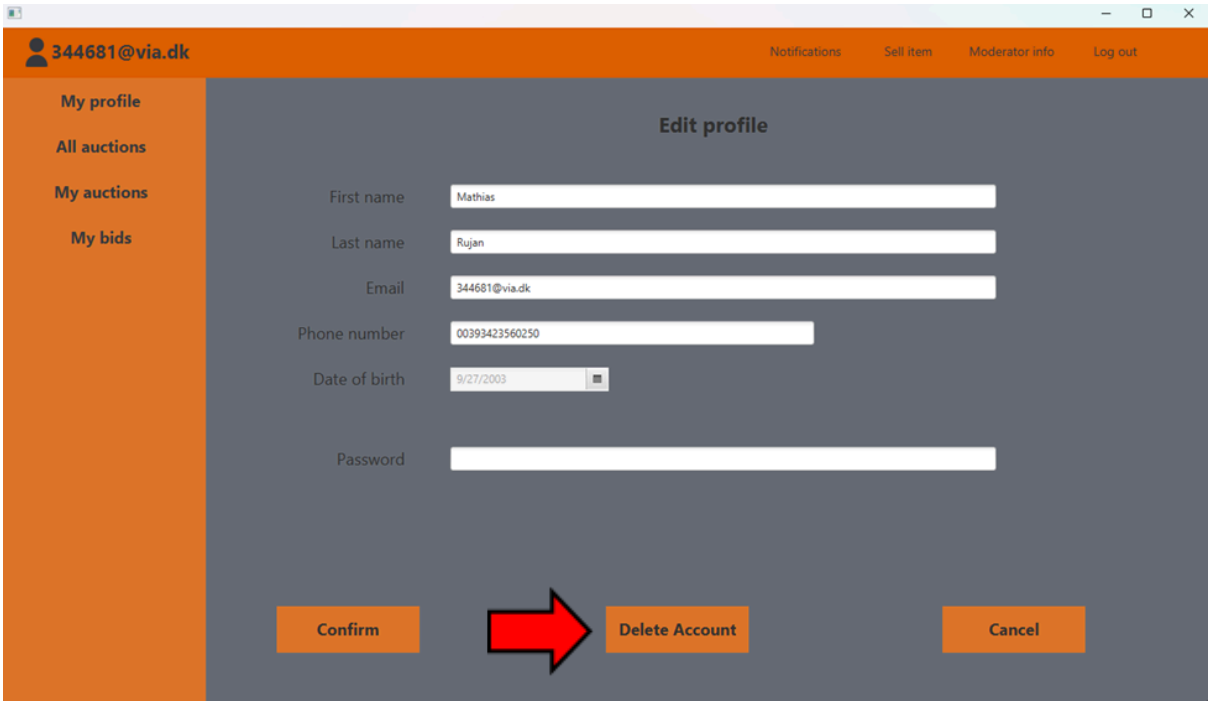
1. Login (see point 2. Login)



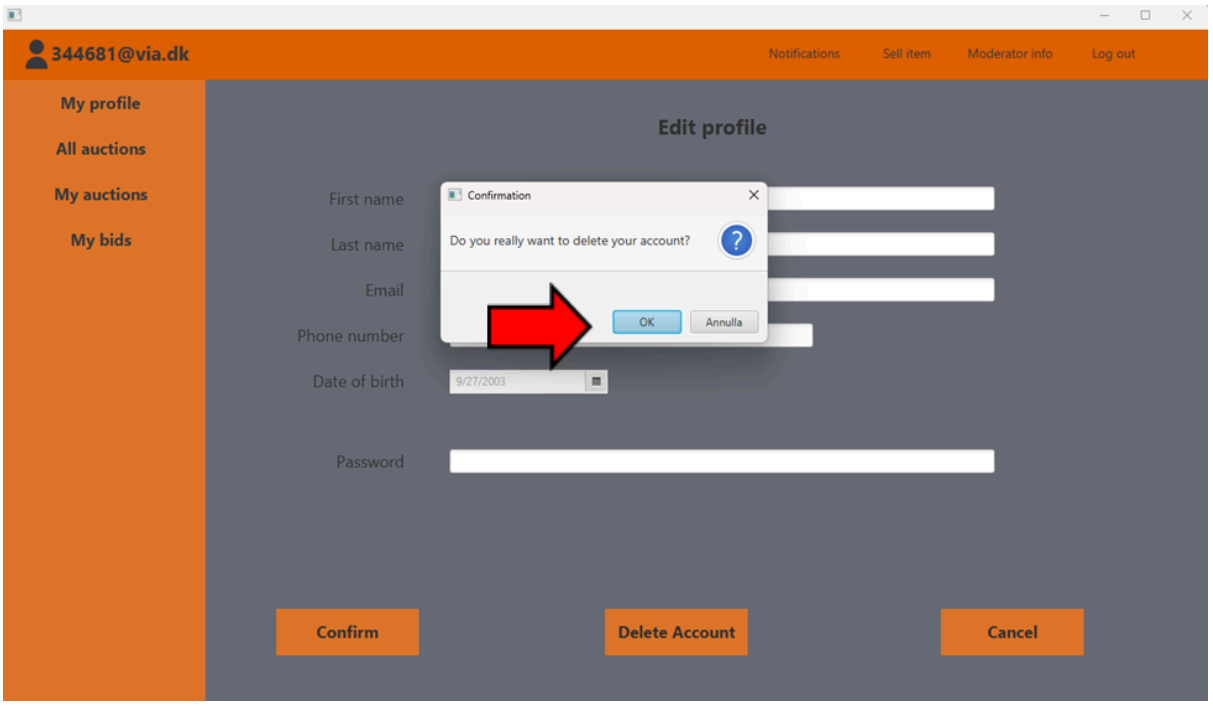
2. Click the “My Profile” button.
3. Click the “Edit” button.



4. Enter your password and click the “Delete Account” button.



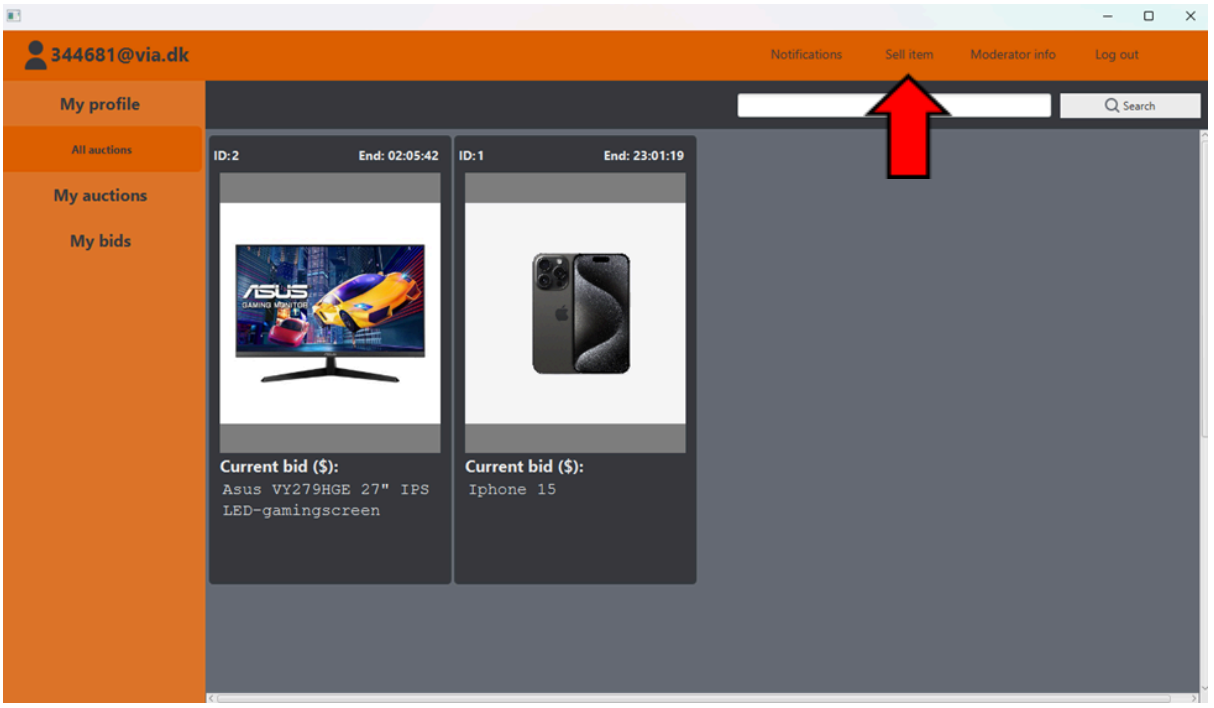
5. Confirm the action in the pop-up window.



6. Start Auction

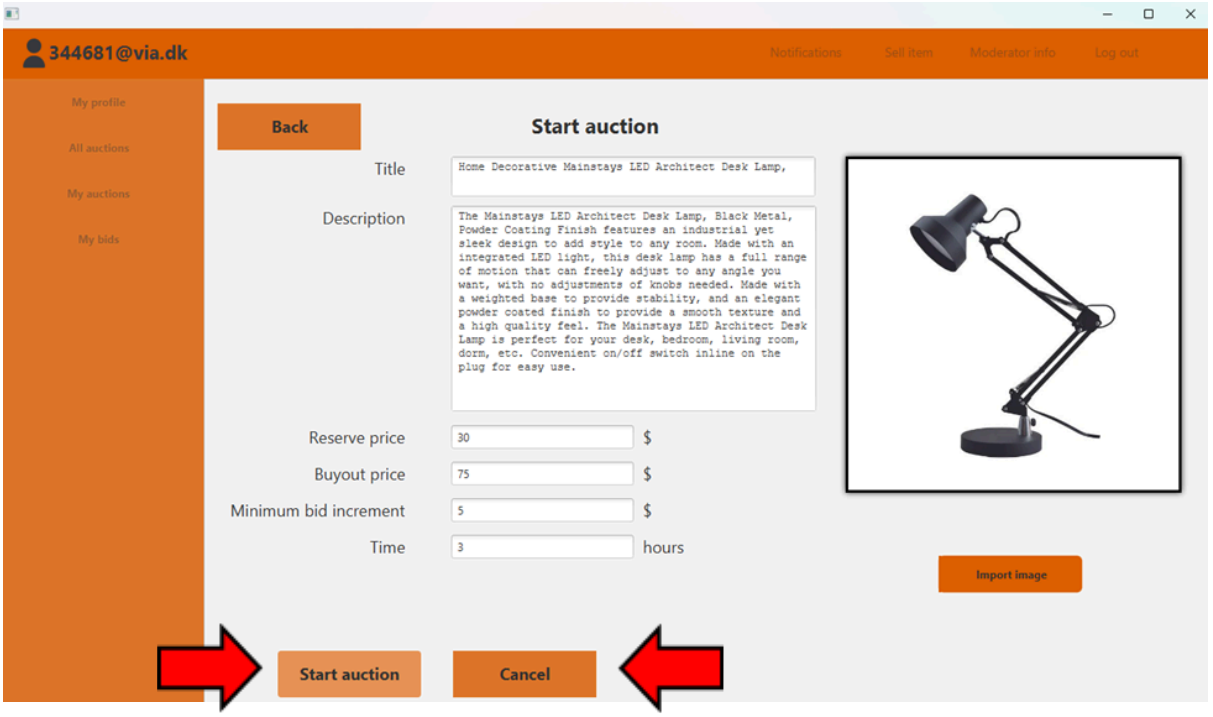
To start a new auction:

1. Login (see point 2. Login)
2. Click the “Sell Item” button.



3. Fill in the item details:
    - Title
    - Description
    - Reserve price
    - Buyout price
    - Minimum bid increment
    - Auction duration
    - Image

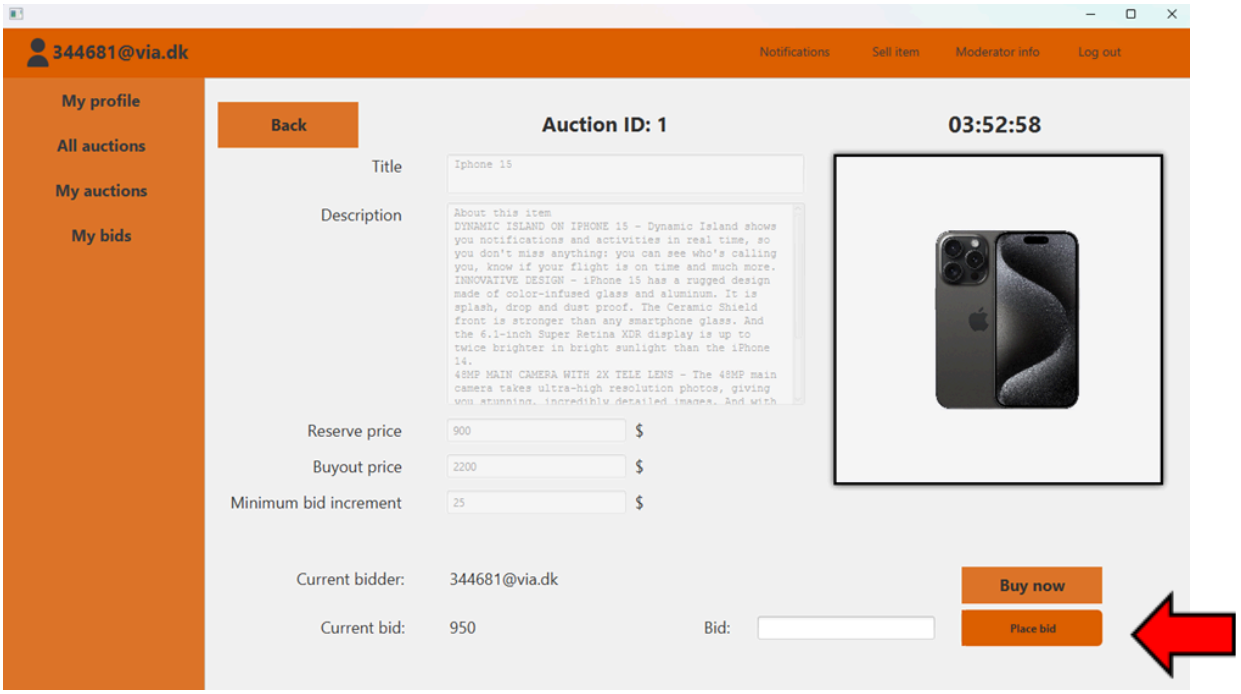
4. Click “Start Auction” to list the item or “Cancel” to discard the changes.



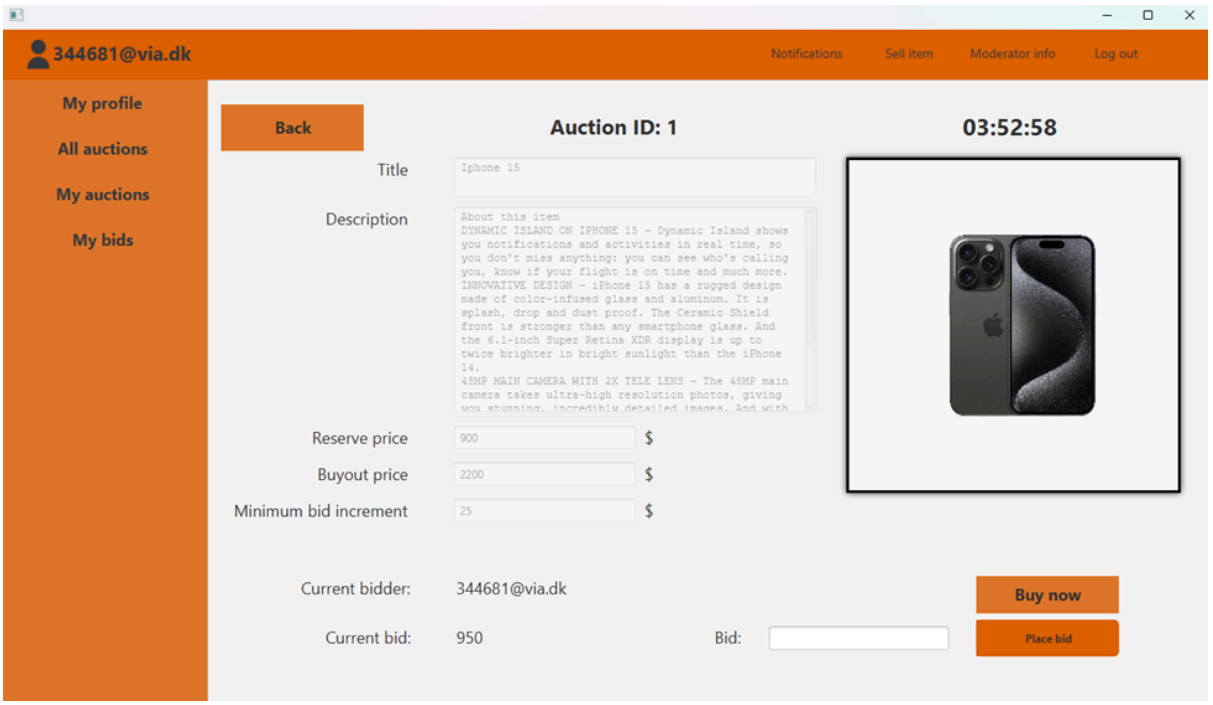
7. Place Bid

To place a bid on an auction:

1. Login (see point 2. Login).
2. Navigate to the "All Auctions" window.
3. Select the item you want to bid on.
4. Enter your bid amount next to "Bid".



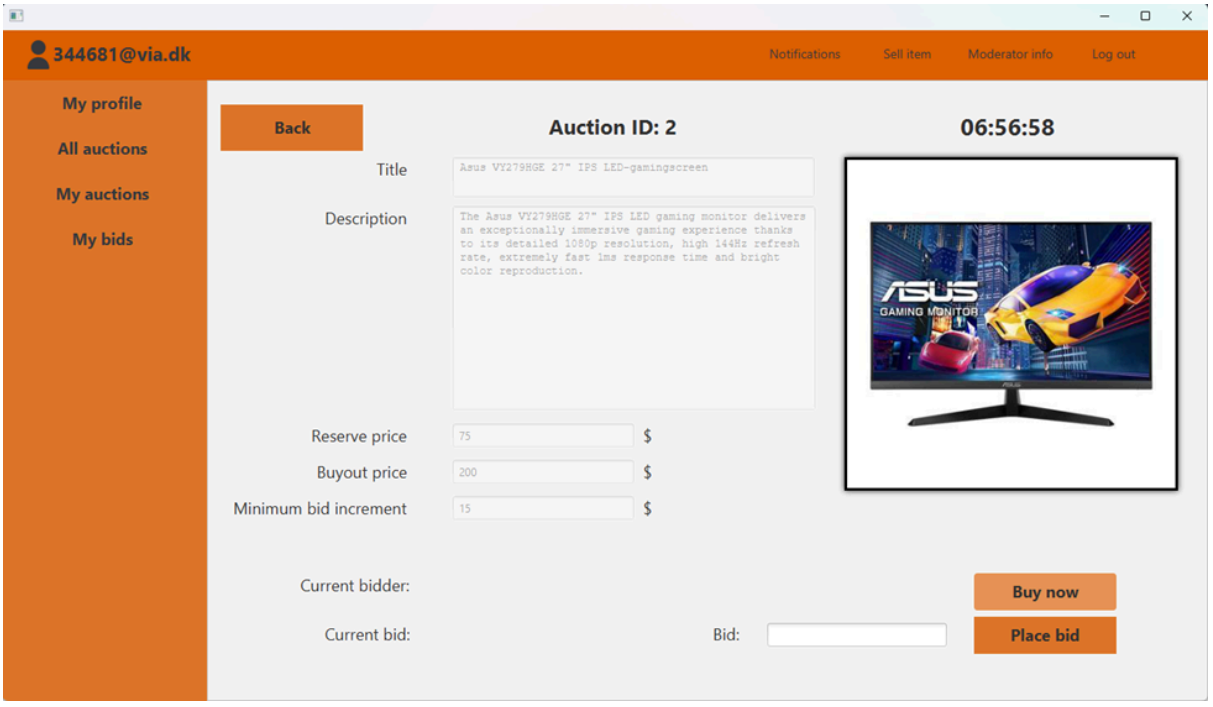
5. Click “Place Bid”



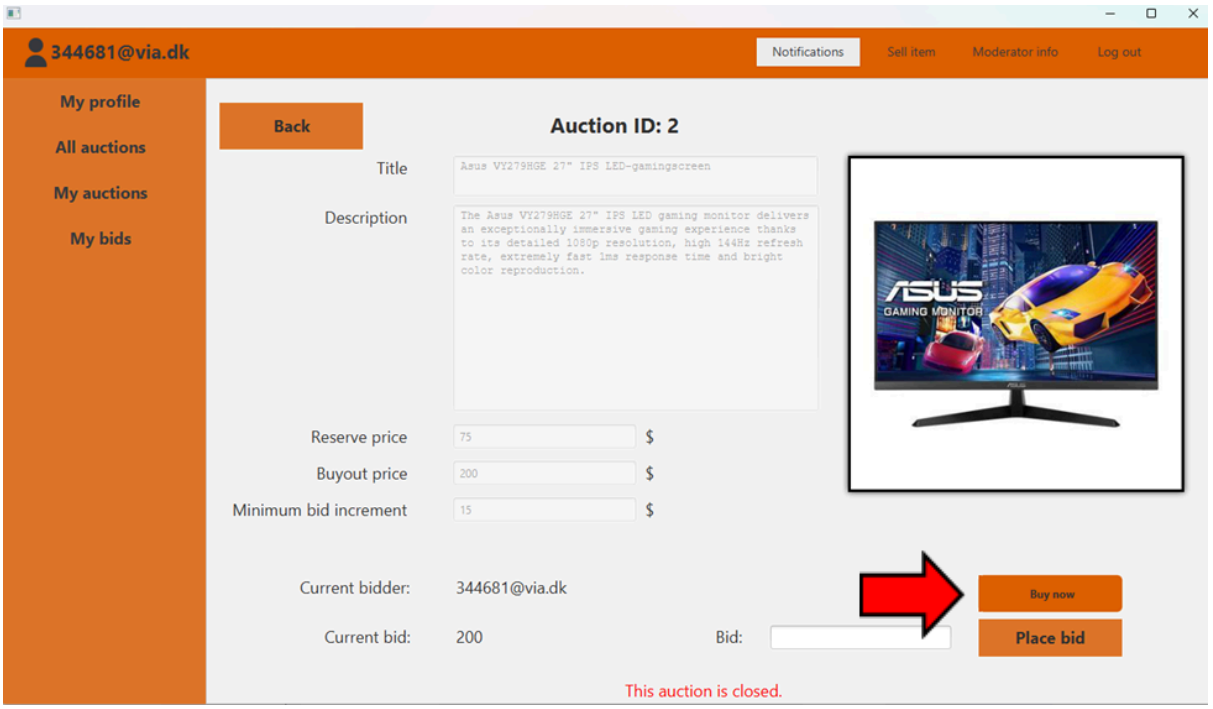
8. Buyout

To buy an item immediately:

1. Login (see point 2. Login)
2. Navigate to the "All Auctions" window.
3. Select the item you want to buy.



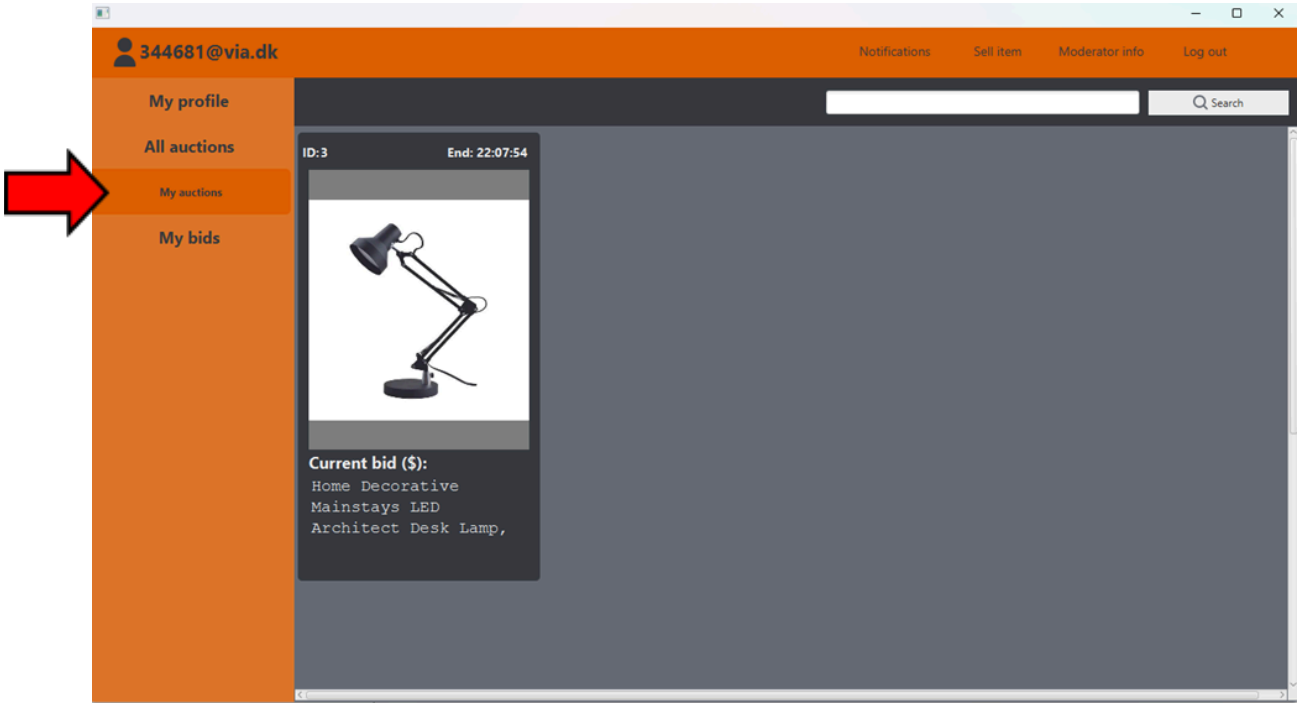
2. Click the “Buy now” button (only if there are no existing bids).



9. Access Personal Auctions

To view your personal auctions:

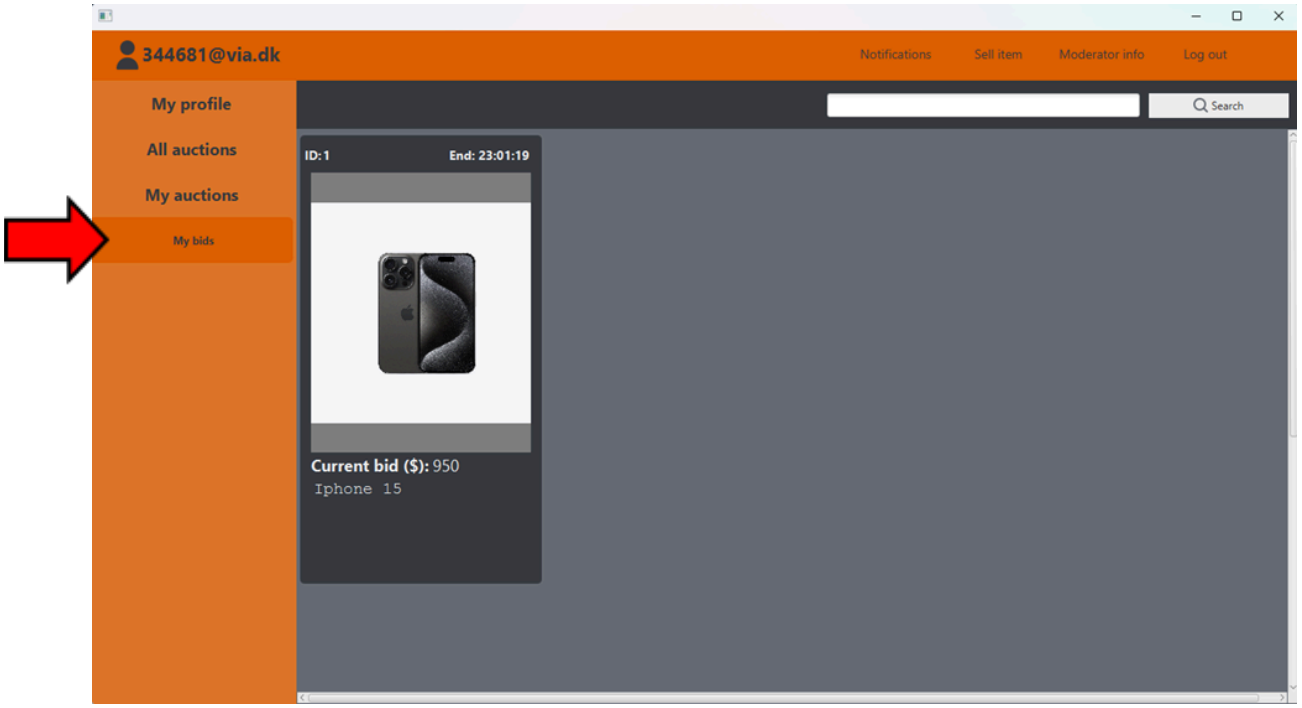
1. Login (see point 2. Login)
2. Click the “My Auctions” button.



10. Access Personal Bids

To view items you have bid on:

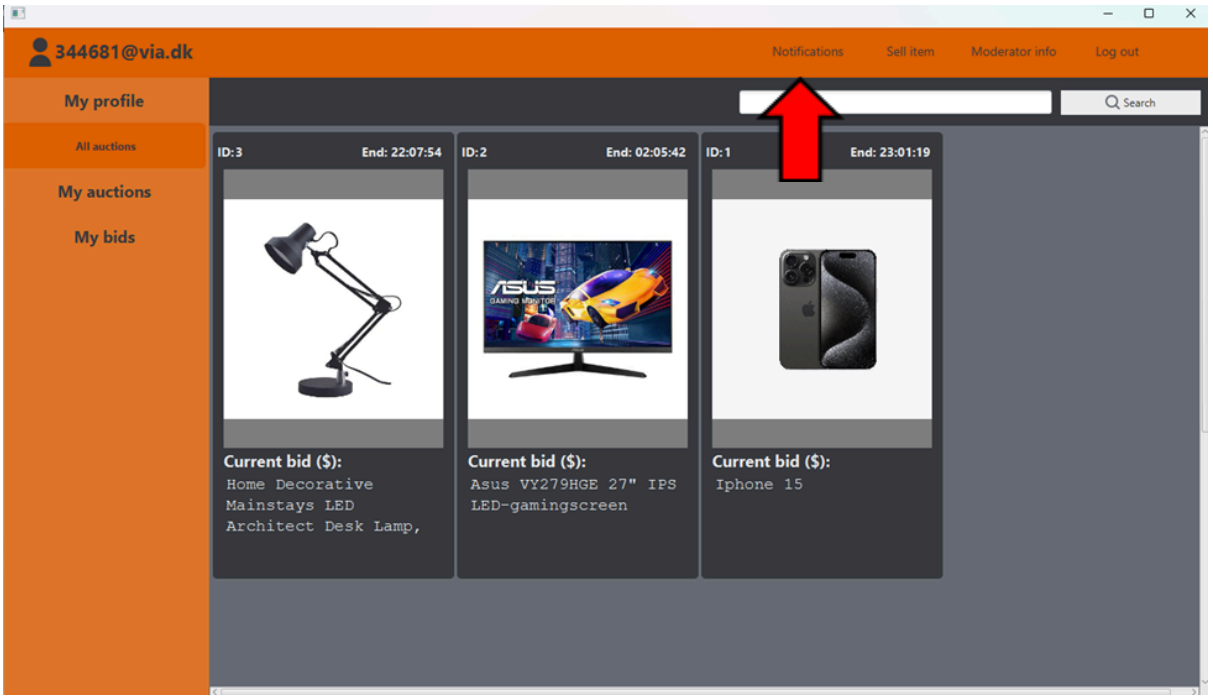
1. Login (see point 2. Login)
2. Click the “My Bids” button.

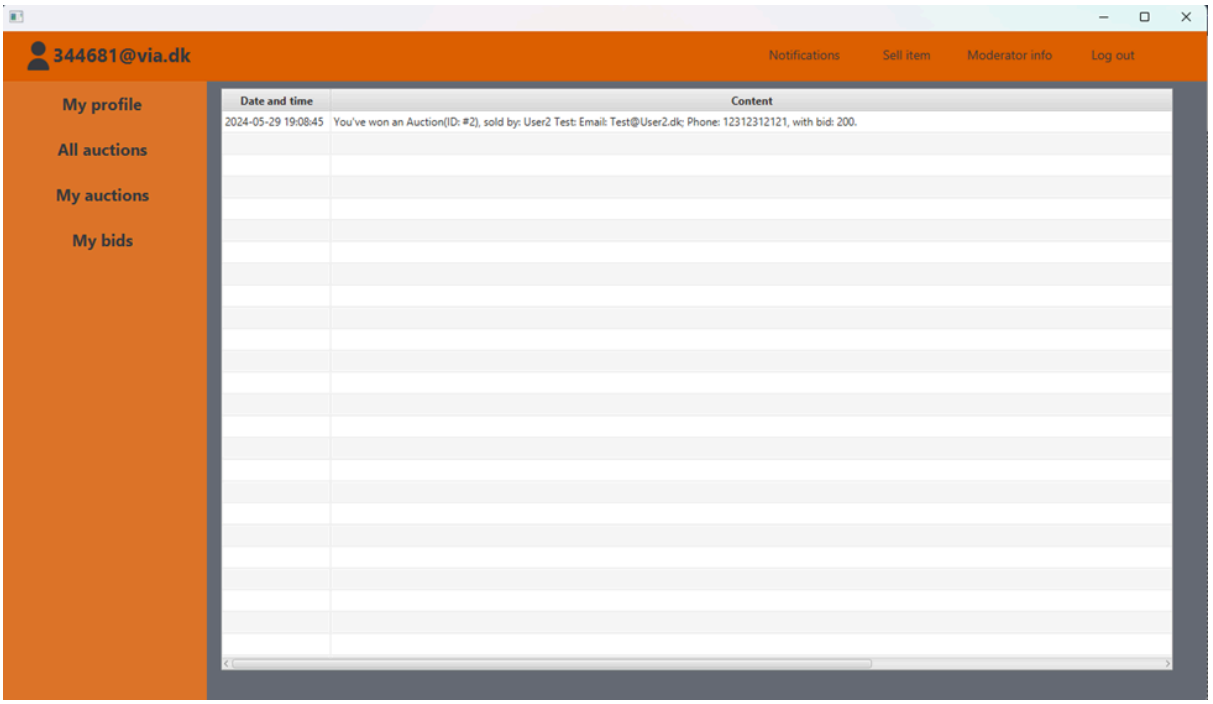


11. Access Notifications

To view your notifications:

1. Login (see point 2.Login)
2. Click the “Notifications” button.

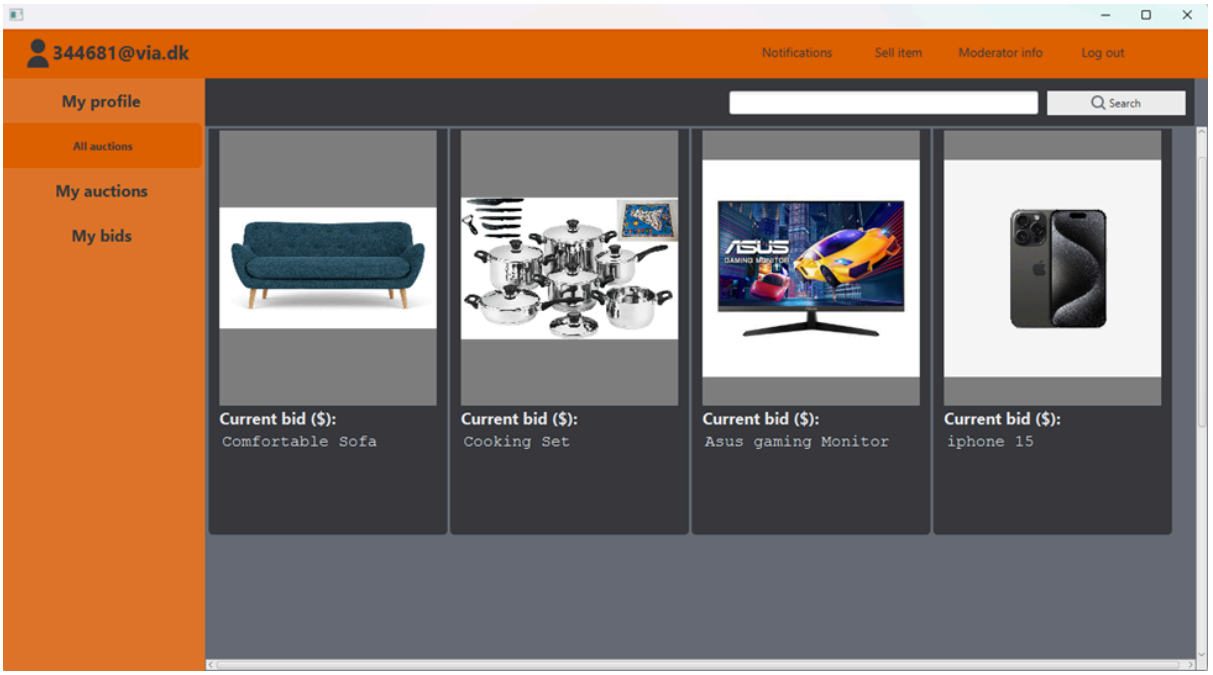




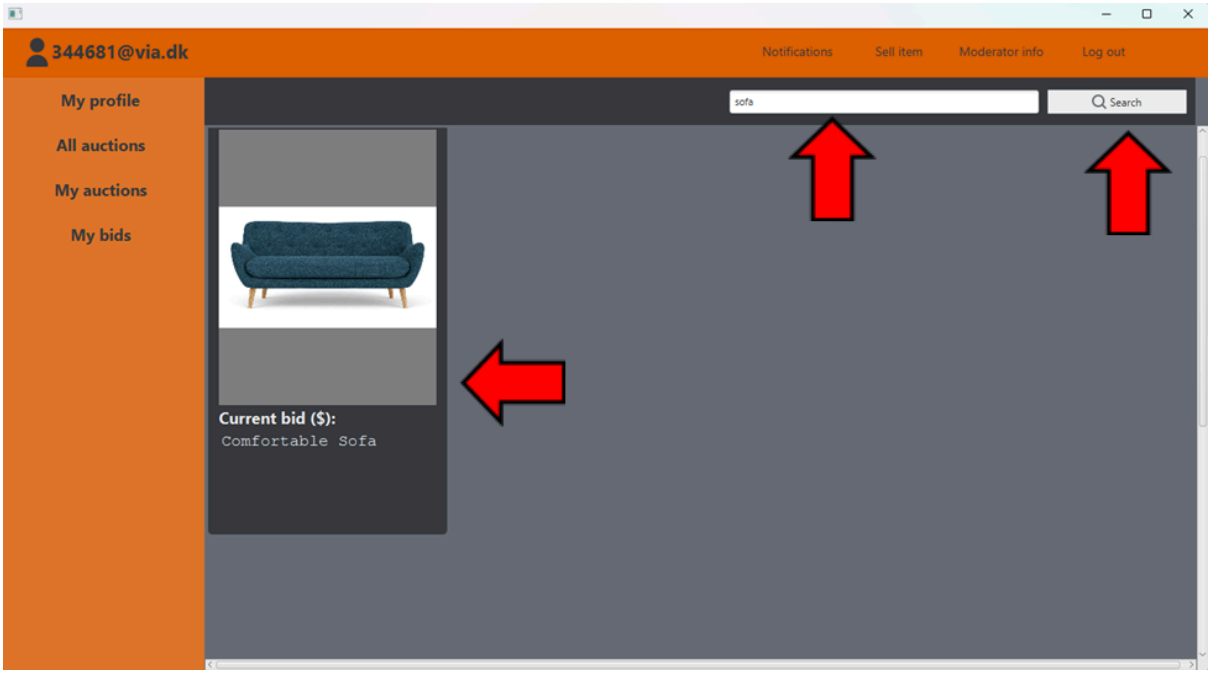
12. Search Auction by ID or Title

To search for an auction:

1. Login (see point 2.Login)
2. Click on “All auctions”.



2. Enter the auction ID or title in the search bar and click the “Search” button.

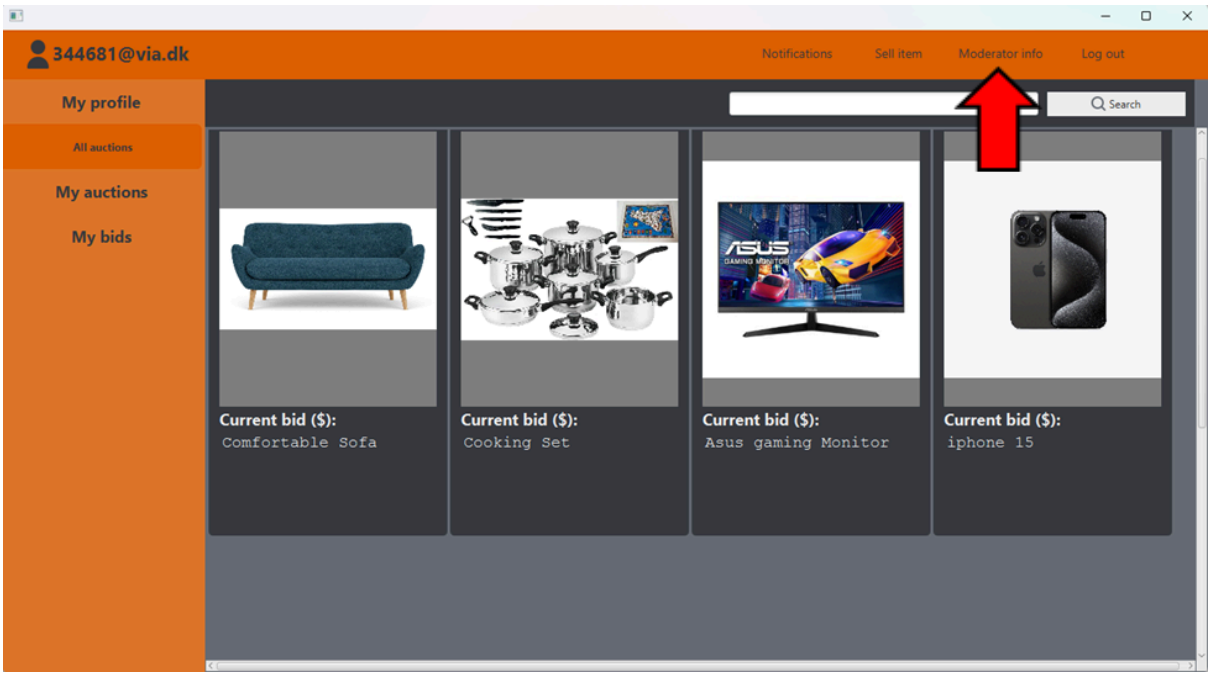


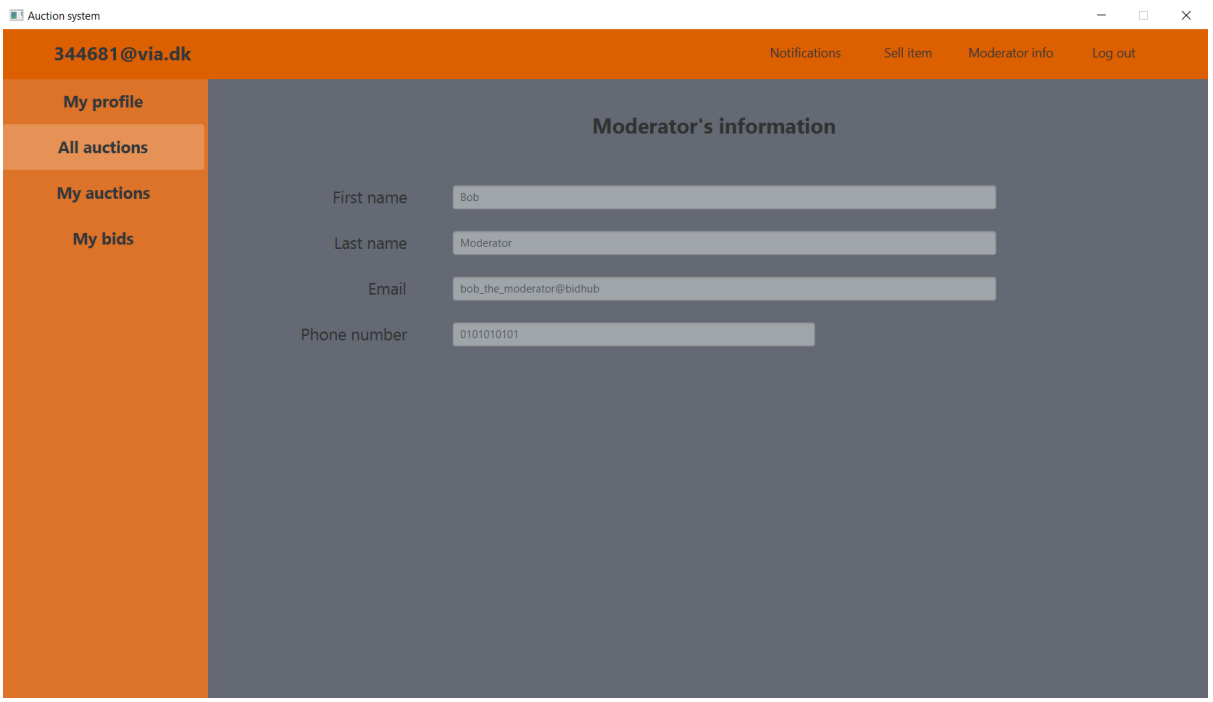
3. View the search results.

13. Access Moderator's Information

To access moderator information (moderators only):

1. Login (see point 2.Login)
2. Click the "Moderator info" button.

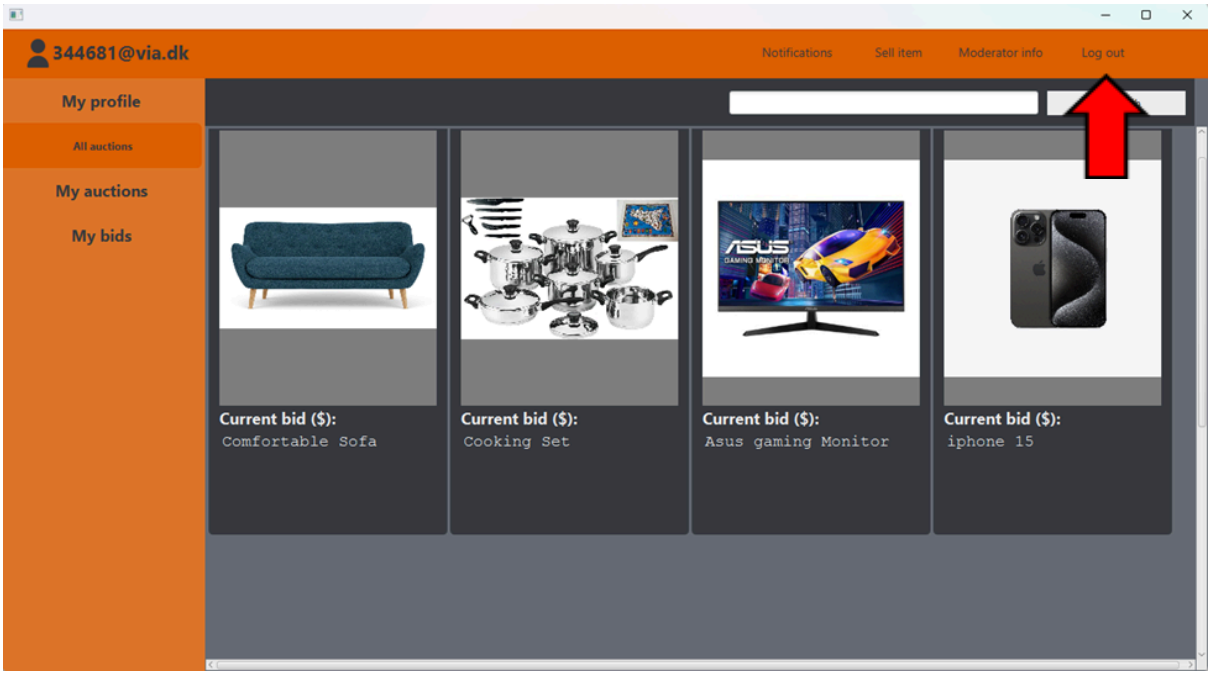




14. Log Out

To log out of the system:

1. Click the “Log out” button.

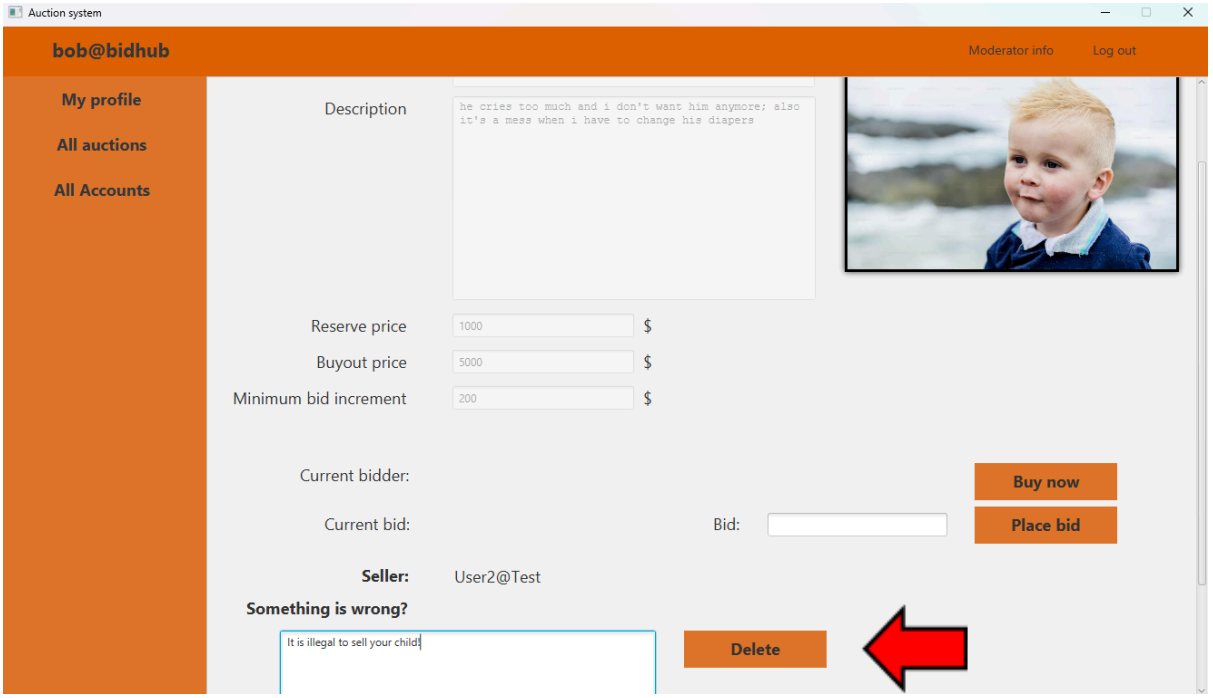


FOR MODERATORS

15. Delete Auction

To delete an auction:

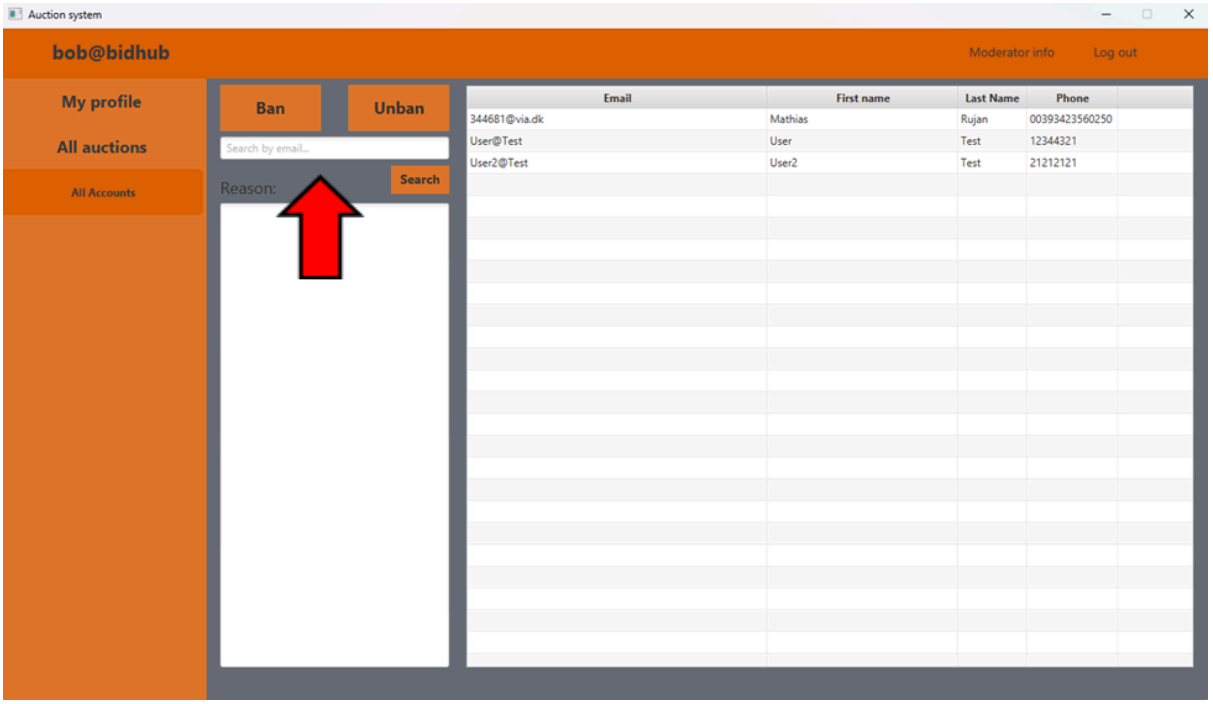
1. Login (see point 2.Login)
2. Navigate to the auction you want to delete.
3. Insert a reason for deletion.
4. Click the “Delete” button.



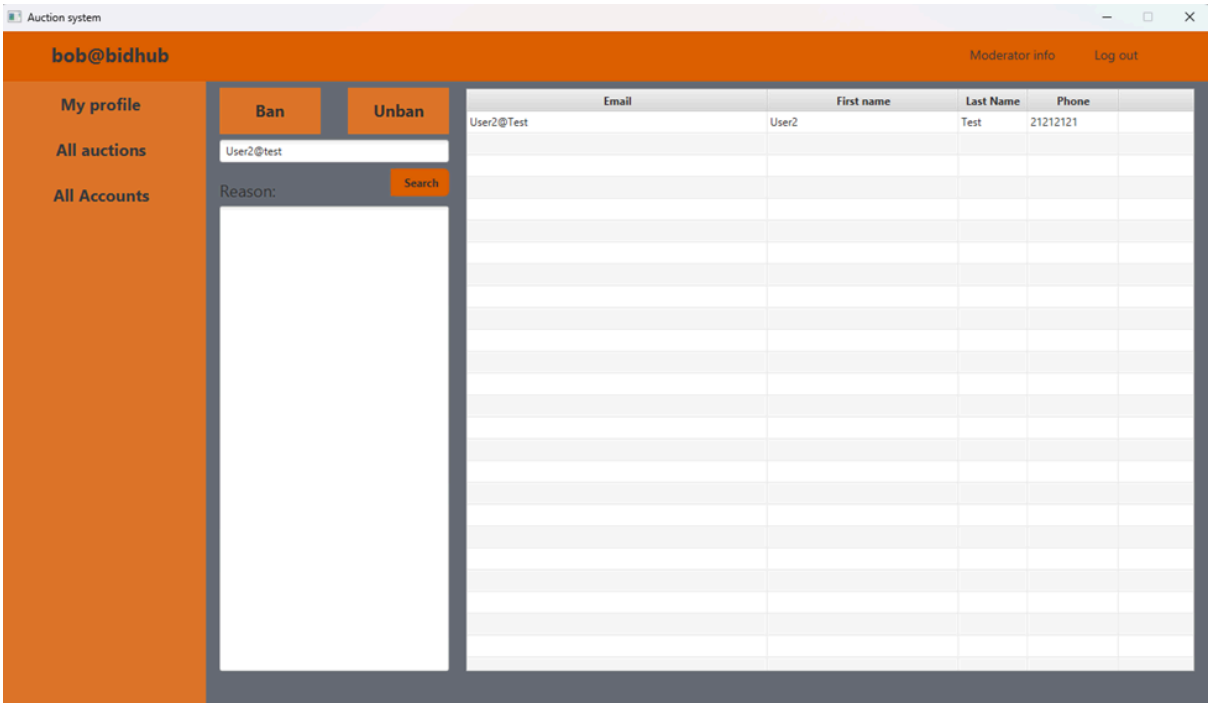
16. Search for Participant

To search for a participant:

1. Login (see point 2.Login)
2. Click the “All Accounts” button.
2. Enter the participant's name or email in the search bar and click “Search”.



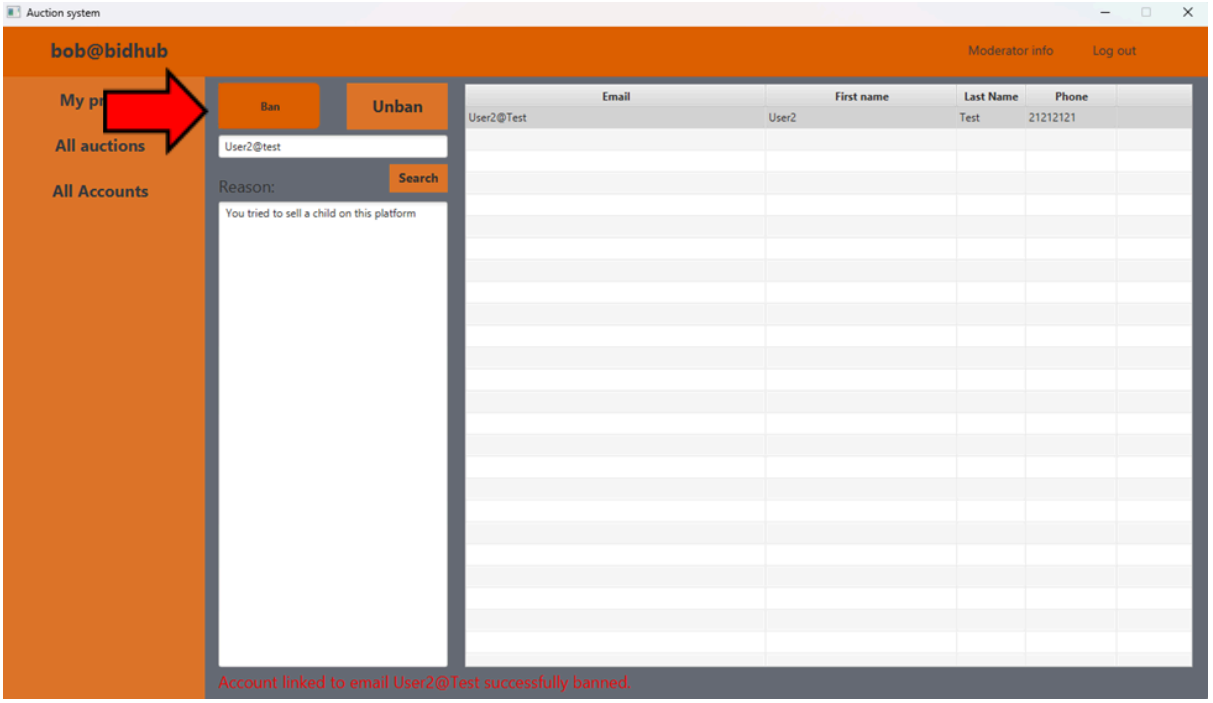
3. View the search results.



17. Ban Participant

To ban a participant:

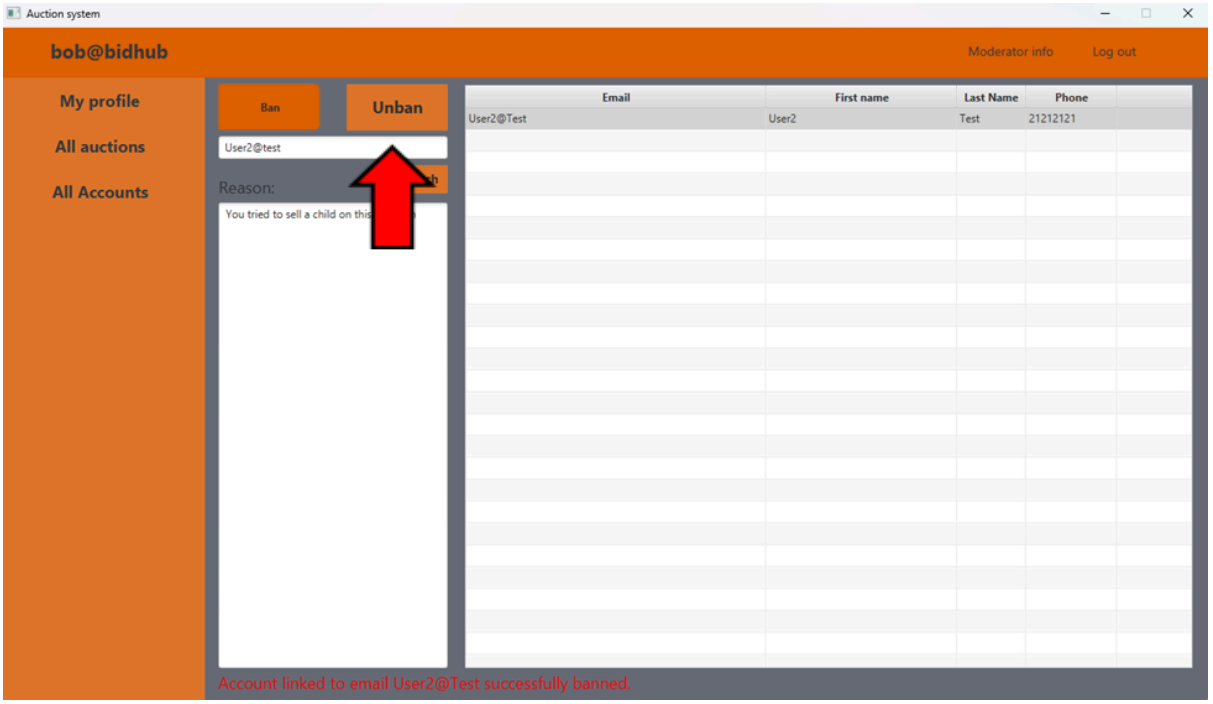
1. Login (see point 2.Login)
2. Navigate to the participant's profile in the “All accounts” list.
3. Insert a reason for the ban.
4. Click the “Ban” button.



18. Unban Participant

To unban a participant:

1. Login (see point 2.Login)
2. Navigate to the banned participant's profile.
3. Click the “Unban” button



19. Set Contact Information

To set your contact information:

1. Login (see point 2.Login)
2. Click the “My profile” button.
3. Enter or edit your contact details.
4. Enter your password.
5. Click “Confirm”.