CIS 501: Software Architecture and Design

Final Project (v1.1 f19)

Teamwork.

(30 points) Design is due on October 31, 2019 11:59 PM US Central Time.

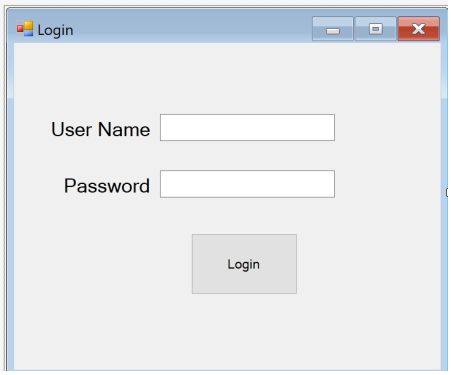
(30 Points) Implementation is due on November 30, 2019 11:59 PM US Central Time.

Introduction

You are to design and later implement a GUI-based bidding application Bid501. Your implementation must reflect your design. If you make changes at implementation time, be sure to update your design documents (keep the different versions).

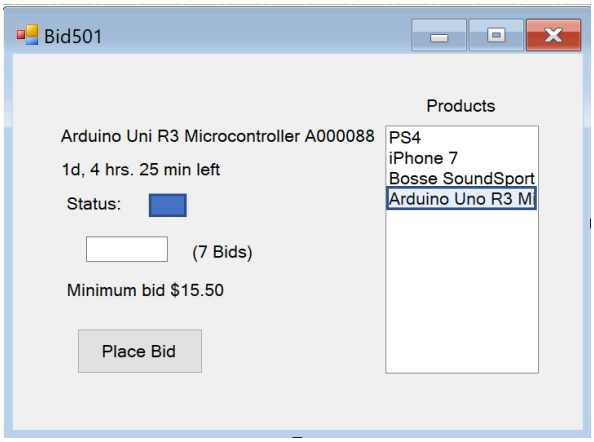
Bid501Server & Bid501Client

- The Client and the Server must run in different computers
- Bid501Client and Bid501Server (solution) must conform with the **MVC** architecture
- All classes and methods must comply with the **SRP**
- All interfaces must comply with the **ISP**
- Client and Server should comply with the **OCP** in reference to:
    - different types of items that can be auctioned
    - different types of transactions, such as traditional bid and buy transaction
- The user/admin must clear credentials before accessing his/her account. (Server and Client)



- After the user/admin enters username and password, he/she will be able to click on the “Login” button. If the user enters the right credentials, he/she will be able to access his/her account. If credentials are not right, a pop-up message notifying the user will be displayed. We will not implement a recovery mechanism at this time. No purchase or bidding history is kept at this time.

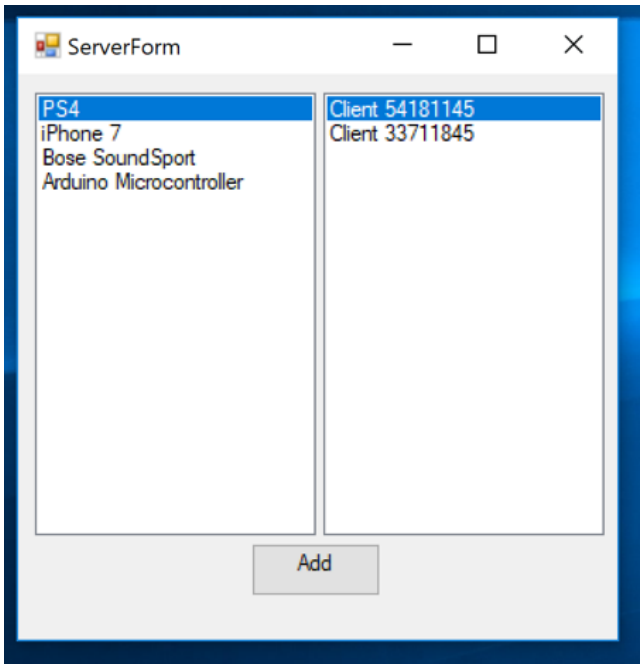
Bid501Client's GUI and Functional Requirements



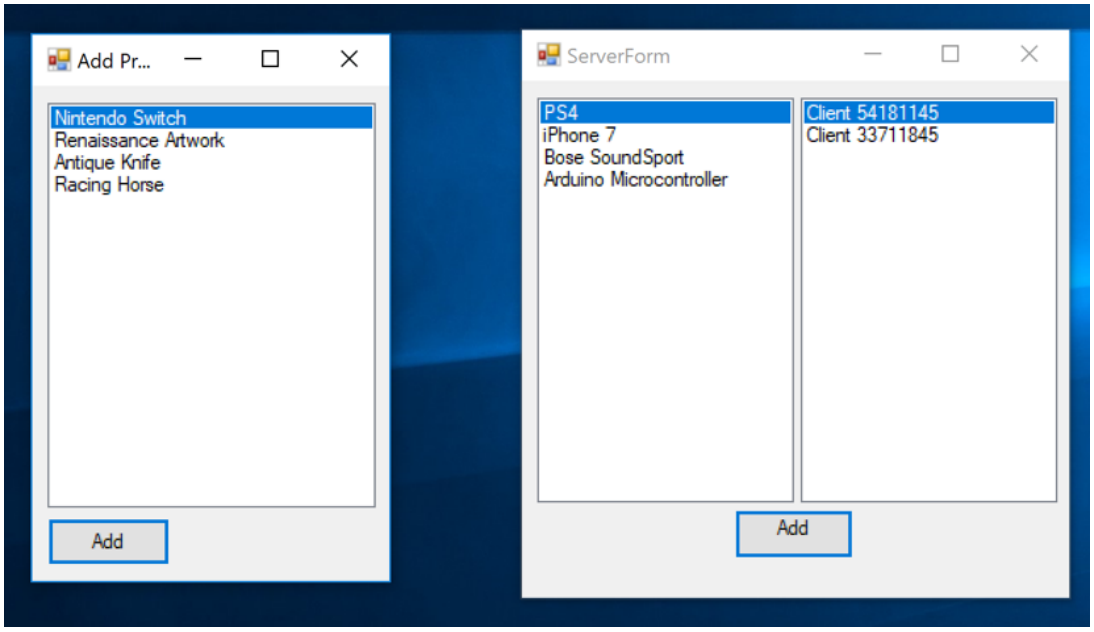
- Maintain a list of products being actioned in the client side. Allow the user to place bids on a selected item. Removes not allowed.
- On start, connect to the Bid501Server to pull the current products being auctioned
- When a product is selected in the list of products, the detailed product info is displayed in the client GUI
- When the user places a valid bid, notify the Bid501Server
- Update the list of products and their status when server sends updates
- Use a Proxy to minimize the communication from the client to the server

Bid501Server's GUI and Functional Requirements

- Maintain a database of users with their password (No encryption is needed at this time)
    - When the server starts, it should lad the users to eb ready to validate credentials and establish connections
- Maintain a database of products to be auctioned
- Provide the ability to Add and Modify* the products to be actioned (not Remove)
- *When a product is modified, notify all the clients:
    - When a bid time expires, notify the Bid501Clients, the one that won, and all the other bidders as well. (Do not implement a countdown timer, implement a way to simulate the auction-time has expired)
    - When a client enters a bid, handle the bid appropriately
- Communicate any change in the database to the Bid501Clients
- On start, load three (3) products from a file and have the product ready to be auctioned when a Bid501Client connects
- Allow a user (admin) to add new products by clicking on the button "Add" (See below)
- When a product is added, the server notifies all the clients of the new product
- When you run the Server, you should see the currents products being auctioned and the clients connected to the server.



- If the user clicks on the “Add” button, a form “Add Product” will pop-up. As shown below:



- When the user clicks on the “Add” button in the “Add Product” form, the next product in the stack should be added into the list of current products being actioned in the Server, and the Server should update all Clients connected
- In this version of Bid501 app, we do not auction, simultaneously, the same type of product more than once, that is, we do not want duplicated products in our list. (two products are the 'same' if their name and description match)

Design Phase

In this phase, you are to write use case realizations (UCR), create class diagrams (following the MVC), sequence diagrams, and state diagrams for your **server and client** apps. The goal of your architecture design is to increase/benefit coupling and cohesion. Here is the list of artifacts expected in your design:

Use Cases Realizations for:

- **UC-01: Log in/Sign in.** A user *A* launches the Client app (C_A) and enters a name (n_A) and password (p_A) to log into the Server (*S*). Server *S* can respond to C_A in two ways:
    - Login is successful – if p_A matches what is in n_A's account record, or if n_A is a new name, in this case, *S* creates a new account with name n_A and password p_A.
    - Login is unsuccessful – if p_A does not match the information in n_A's account records.
- **UC-02: Place bid.** A user selects a product form the list and places a valid bid.
- **UC-03: Add a new product.** A user clicks on the "Add" button in the AddForm after selecting a product to auction

Class Diagrams for:

- **Class Diagrams** (for the whole **Bid501Client and Bid501Server Apps**). Your solution must follow the MVC architecture with interfaces, delegates (or equivalent), and patterns as appropriate. Note that your class diagram should include classes (whose objects will be de/serialized) for client/server communication. You can submit your diagrams as PDF files generated from a computer program (e.g. Draw.io).

Sequence Diagrams for use-case realizations UCR-02 and UCR-03

State Diagrams for the controllers in your MVCs.

Submission

1. Create:
    - A solution in VS, call it **Bid501**,
    - A Project and call it **Bid501-Server**
    - A Project and call it **Bid501-Client**
    - Create three source folders FOR EACH project (Client and Server), and name them ModelSrc , ViewSrc, and CntrlSrc. Populate this folders with the classes you design and code for the model, view, and controller respectively.
    - A folder called "Diagrams" where you upload PDF versions of all the diagrams.
2. Push your initial solution to GitHub (it will be empty - no source code/diagrams)
3. Work on your diagrams in LucidCharts or Draw.io.
4. Upload the PDF version of your diagrams and push it to GitHUb.

****No email submission will be accepted for this assignment. ****