

Process MeNtOR 3.0

Uni-SEP

<BiddingBlitz> **Design Document**





| Group # | Name, Surname, email |
|----------|----------------------------------|
| Member 1 | James, Le, jamesmql@my.yorku.ca |
| Member 2 | Brandon, La, brandla@my.yorku.ca |
| Member 3 | Ryan, Luk, ryankluk@my.yorku.ca |
| Member 4 | Eric, Cha, eric905@my.yorku.ca |

| | |
|--|--|
| Version: 4 | |
| Submission Date: 2024-10-14 | |
| File Name: EECS4413-Deliverable-1-Report | |

Document Change Control

| Version | Date | Authors | Summary of Changes |
|---------|------------|----------------------------|--|
| 1 | 2024-10-07 | James, Ryan, Brandon, Eric | Architecture |
| 2 | 2024-10-10 | James, Ryan | Added Test Case 1-8, Group Meeting Logs. |
| 3 | 2024-10-13 | Brandon | Added Test Case 12-16, Gantt Diagram, UCS2 sequence and activity diagrams. |
| 4 | 2024-10-14 | James Ryan, Brandon | Finalized all sections for deliverable 1. |

Document Sign-Off

| Name (Position) | Signature | Date |
|-----------------|--|------------|
| James Le |  | 2024/10/10 |
| Ryan Luk |  | 2024/10/10 |
| Eric Cha |  | 2024/10/12 |
| Brandon La |  | 2024/10/10 |

Contents

| | | |
|----------|------------------------------------|----------|
| 1 | INTRODUCTION | 4 |
| 1.1 | Purpose | 4 |
| 1.2 | Overview | 4 |
| 1.3 | Resources - References | 4 |
| 2 | MAJOR DESIGN DECISIONS | 4 |
| 3 | SEQUENCE DIAGRAMS | 4 |
| 4 | ACTIVITY DIAGRAMS | 4 |
| 5 | ARCHITECTURE | 4 |
| 6 | ACTIVITIES PLAN | 5 |
| 6.1 | Project Backlog and Sprint Backlog | 5 |
| 6.2 | Group Meeting Logs | 5 |
| 7 | TEST DRIVEN DEVELOPMENT | 6 |

1 Introduction

1.1 Purpose

This document details the requirements of the system <System Name>.

For small projects this document may contain the complete design model of a system. In larger projects this document will cover only a particular subject area.

This project aims to provide users with an easy and intuitive system to bid for auction items. The system comes equipped with a very simple and easy-to-navigate GUI and a robust system designed to handle anything a user could ask for. The system will include two types of auctions: the Forward Auction and the Dutch Auction. Underneath the simple interface comes the complex yet organized middle and back-end systems, designed to be hidden and work out of sight of the user.

1.2 Overview

Text providing a roadmap to the sections and diagrams in your document

This design document for the BiddingBlitz auction system outlines key sections guiding its development. It begins with major design decisions, including modularization criteria for scalability. We then provide sequence and activity diagrams, each linked to specific use cases to illustrate system workflows. The architecture section details modular components, interfaces, and their interactions. An activities plan, visualized in a Gantt chart, maps out project milestones. Lastly, a test-driven development approach is outlined with test cases to ensure robustness. This document serves as a blueprint for BiddingBlitz from design to implementation and testing.

1.3 Resources - References

Include references to other documents that may assist in the understanding of this document. (i.e. past SRS and SDDs)

2 Major Design Decisions

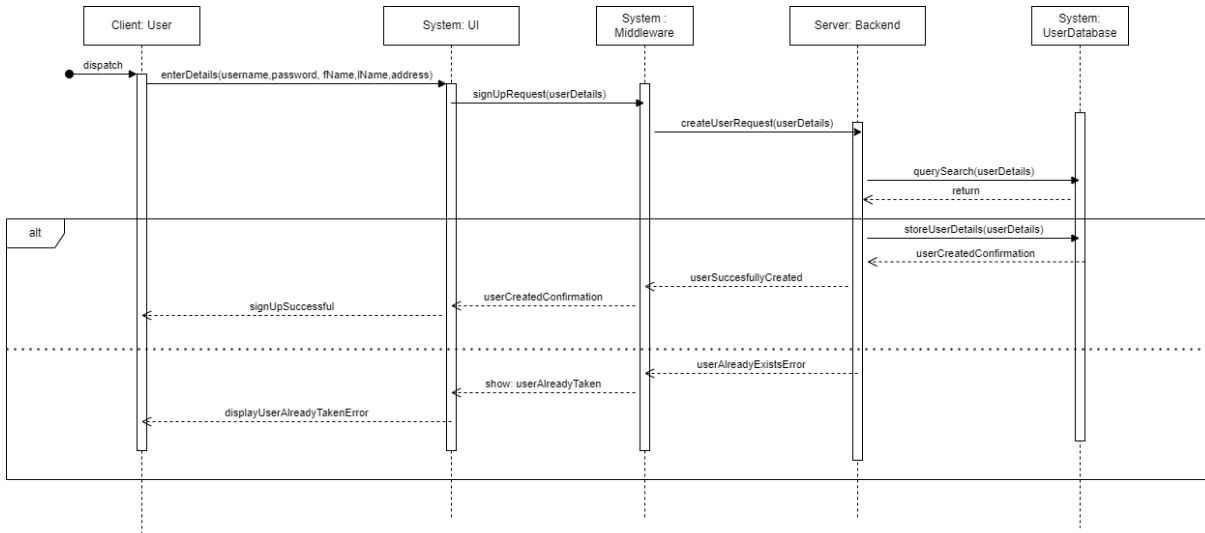
Description of significant design choices, and modularization criteria.

Our design approach centers on standard design patterns to achieve high cohesion and low coupling within the code. We organized the backend server into four distinct modules: catalog, auction, payment, and user authentication. This separation allows us to maintain a low-coupled, modularized structure, making it easier to manage and scale. To accomplish our goals, we implemented several design patterns, including Singleton, Factory, Observer, and Facade.

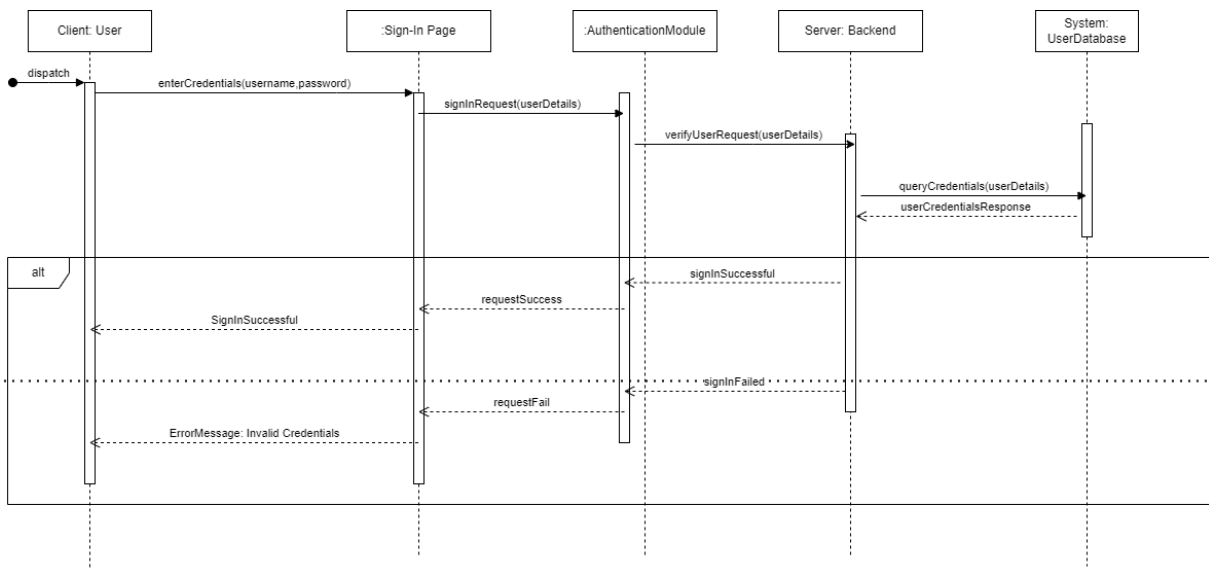
The Singleton pattern ensures only one instance of each module is created, adding simplicity and reliability to our code. We used the Factory pattern to manage auction types—currently, Forward and Dutch auctions—by defining shared functionality and enabling easy expansion for future auction types. The Observer pattern enhances user experience by making the webpage reactive to bid updates and auction status changes, providing real-time notifications. Finally, the Facade pattern is central to our project; it simplifies user interactions by hiding the underlying complexity, which is encapsulated within the controller module.

3 Sequence Diagrams

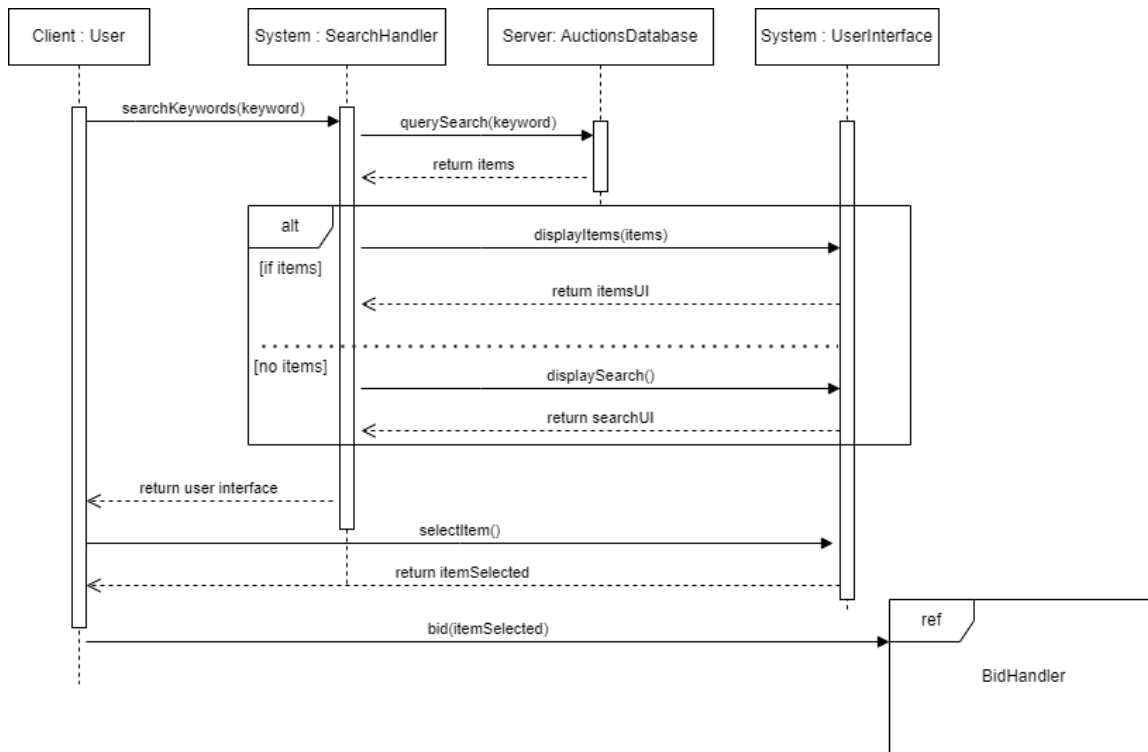
UC1.1: Sign Up



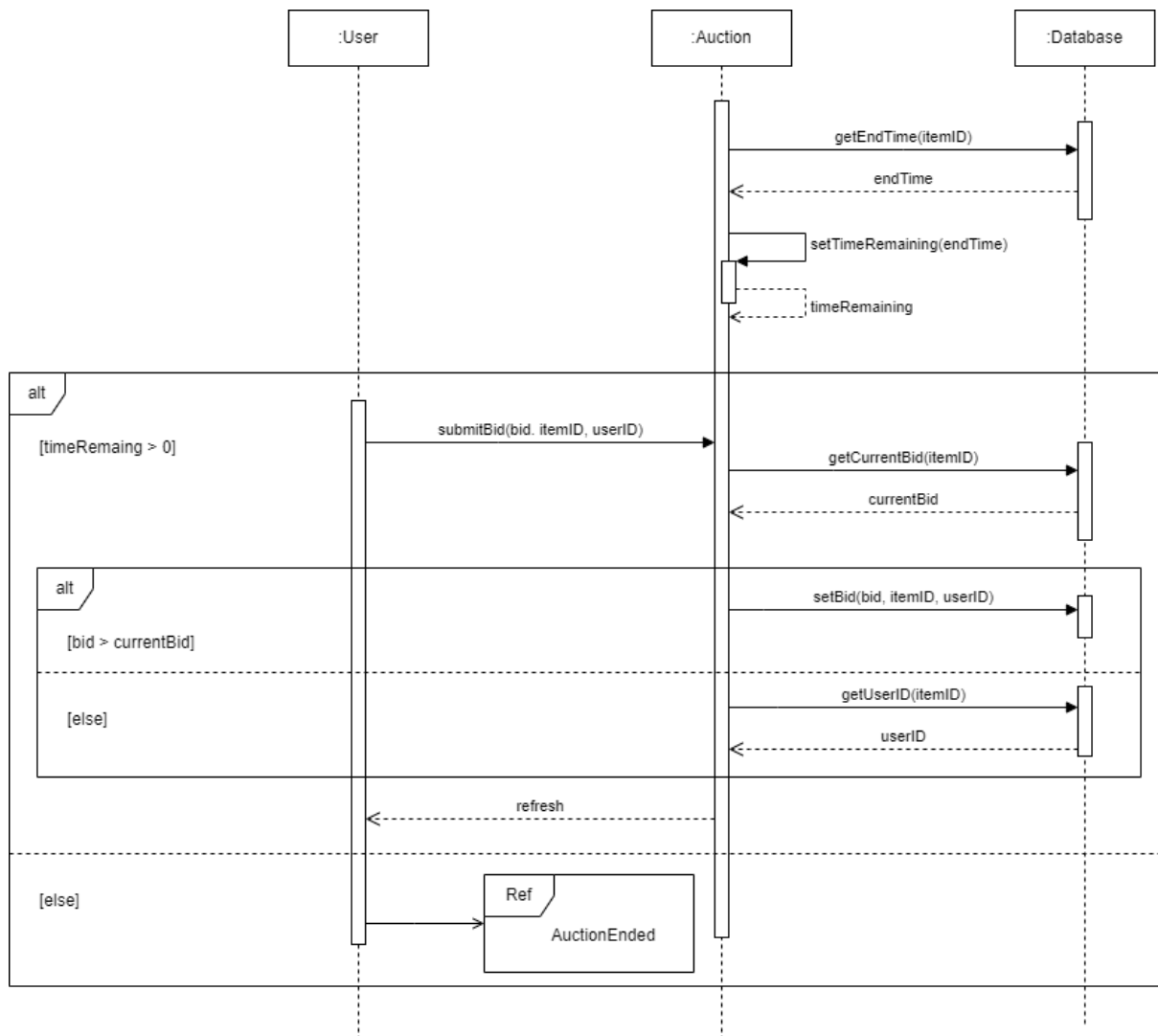
UC1.2: Sign In



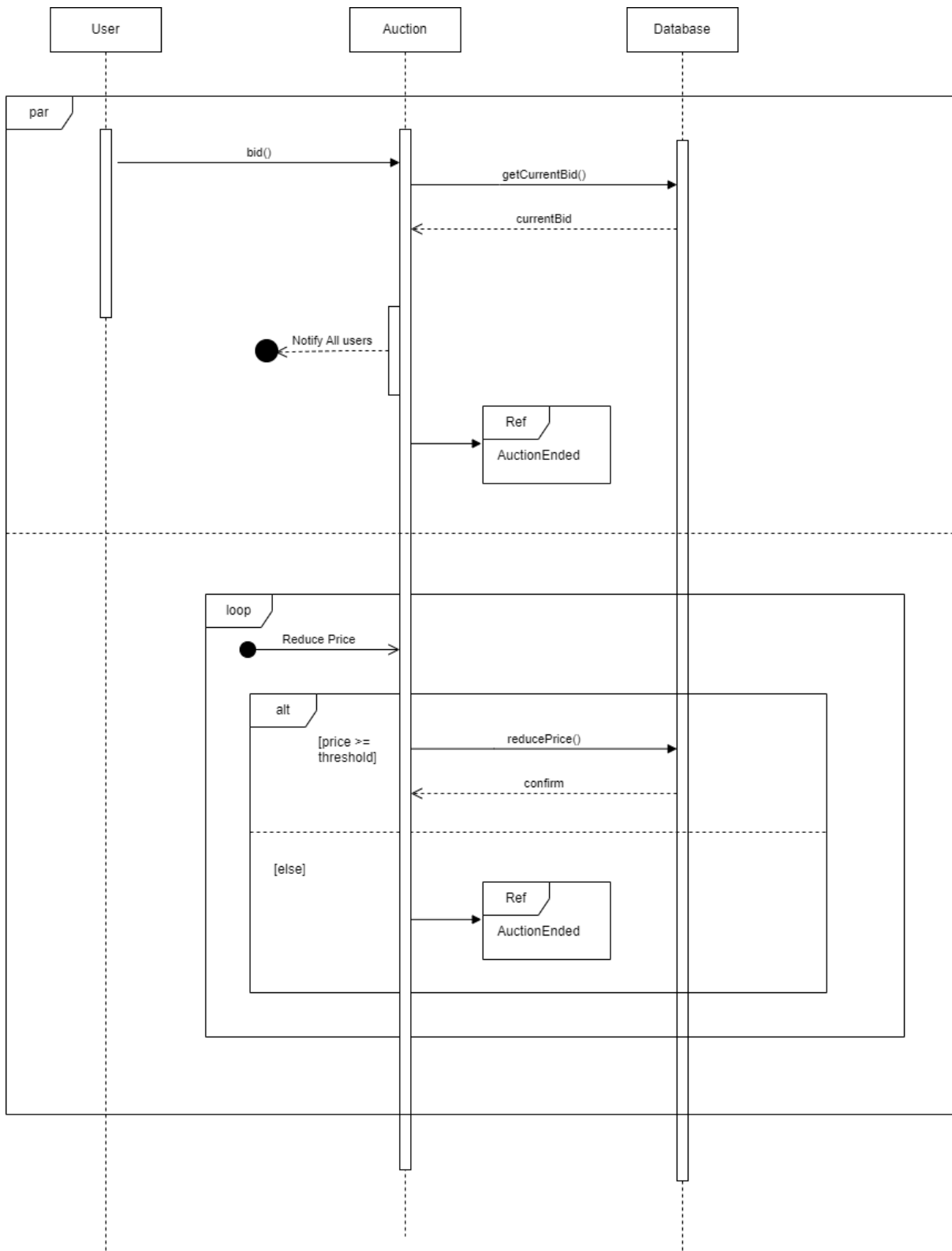
UC2: Browse Catalogue of Auctioned Items



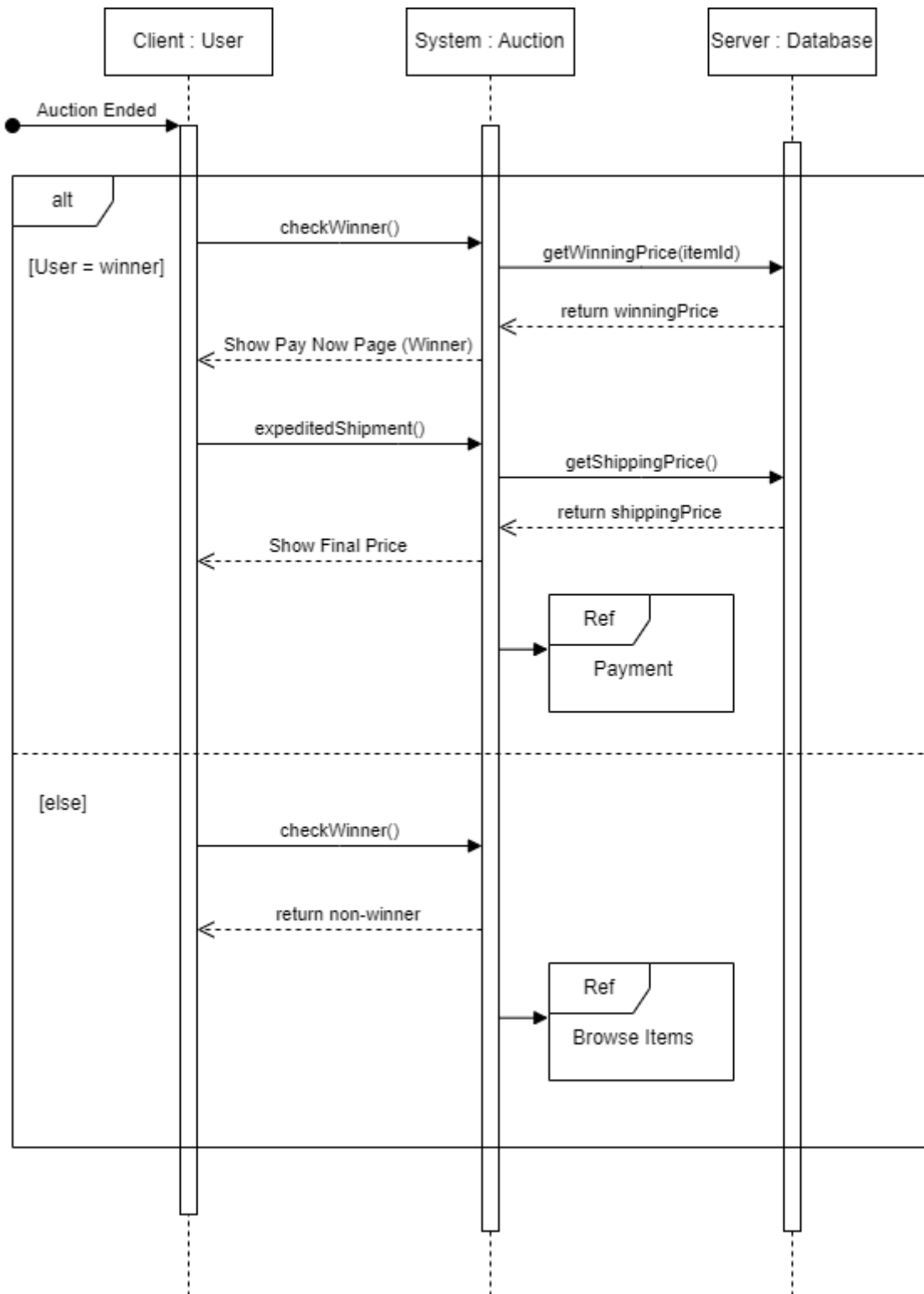
UC3.1: Forward Auction Bidding



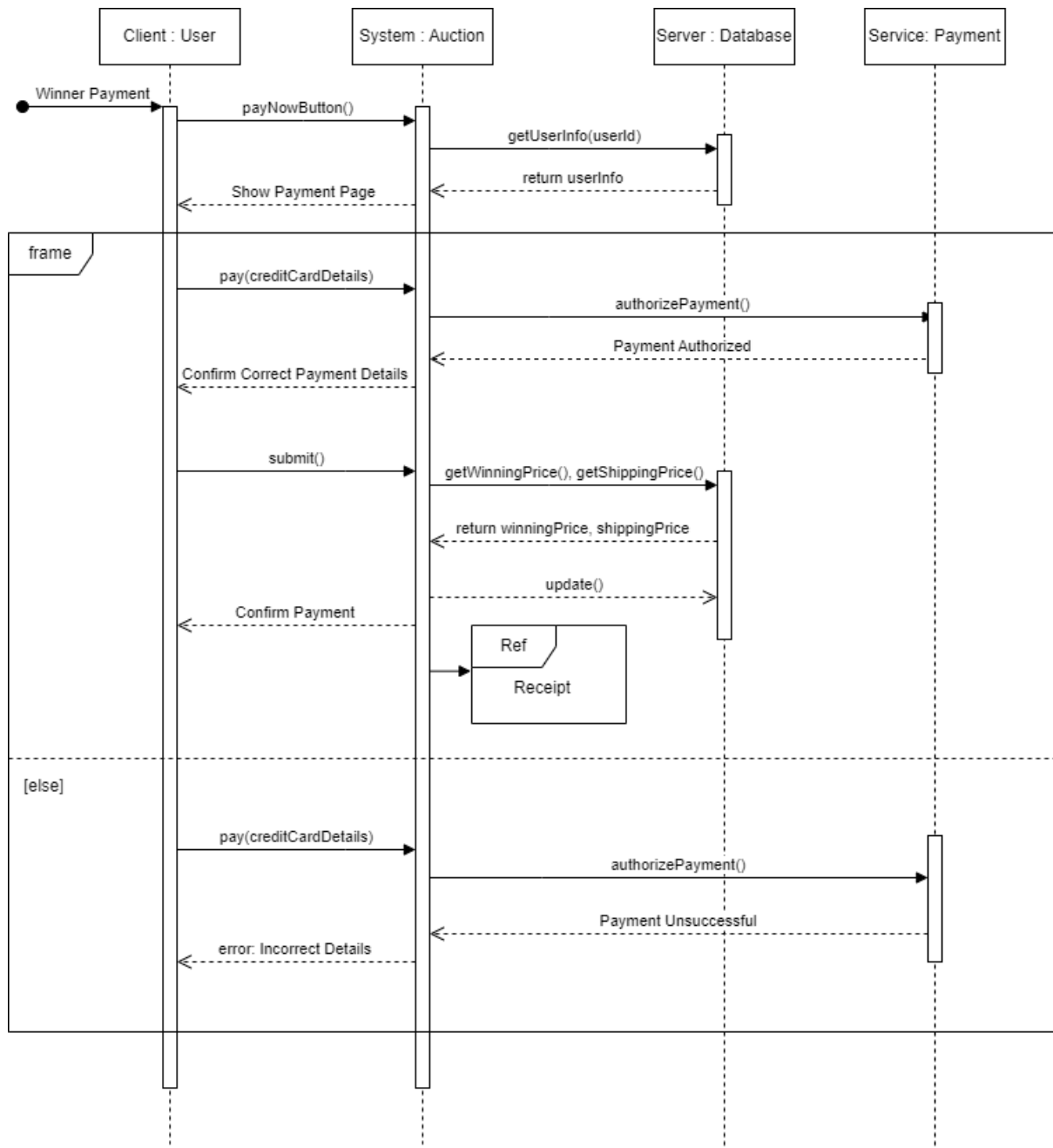
UC3.2: Dutch Auction Bidding



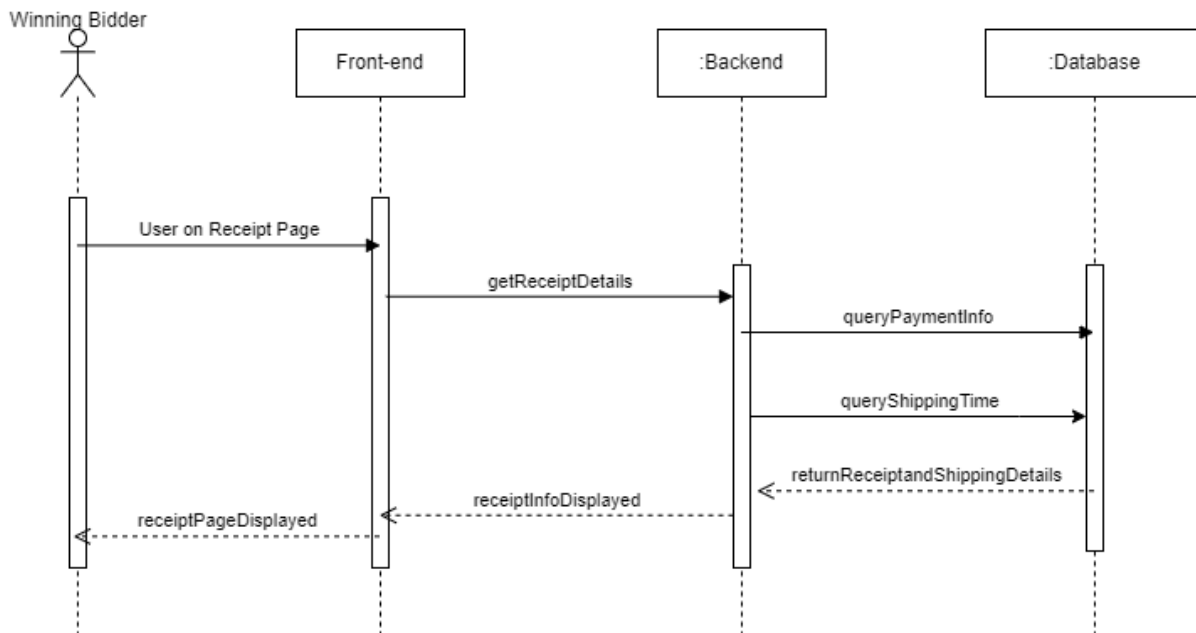
UC4: Auction Ended



UC5: Payment



UC6: Receipt Page and Shipment Details

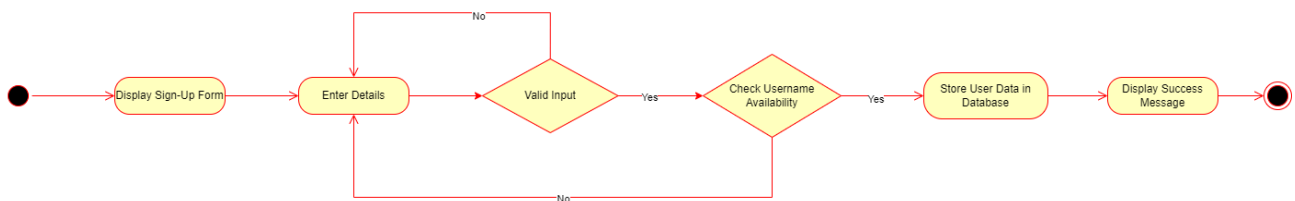


4 Activity Diagrams

For each use case introduce the corresponding activity diagram(s). Make sure you identify and associate each activity diagram with the proper use case by maintaining unique Identifiers for use cases. Refer to the project description for which of the scenario you need to write collaboration diagrams.

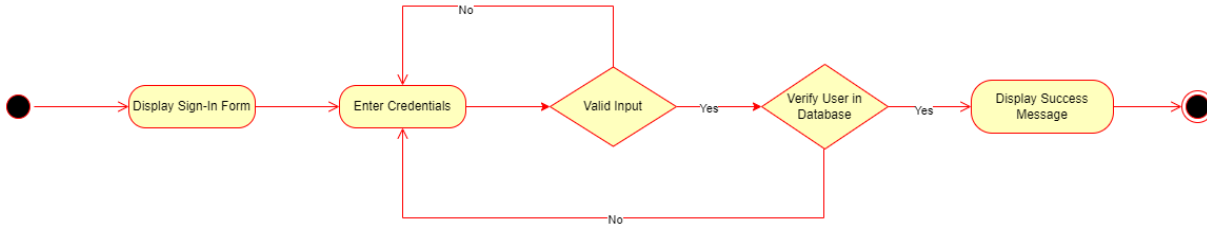
UC1.1: Sign Up

Signing-up requires the users to provide a username, password, first name, last name, and shipping address (street name, street number, city, country, postal code). This information is then stored to a user database and will be used to verify consequent sign-ins



UC1.2: Sign In

Signing in requires the user to provide their username and password. If the user can sign in if the username and password are correct.



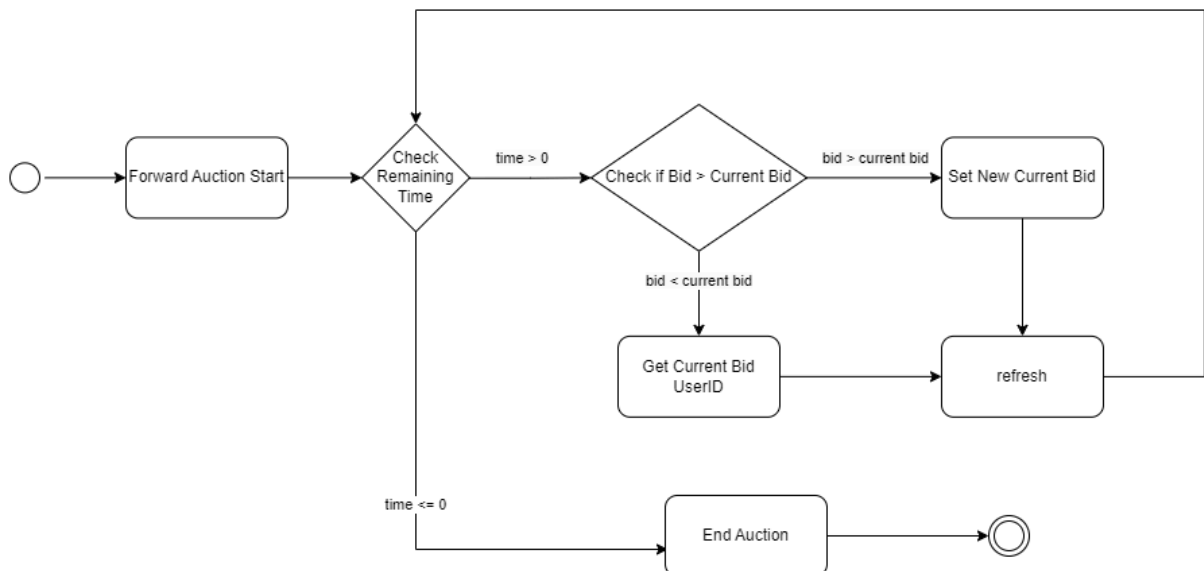
UC2: Browse Catalogue of Auctioned Items

The user can search the catalogue of auctioned items by typing a keyword in the search and select to bid on it given it is still available.



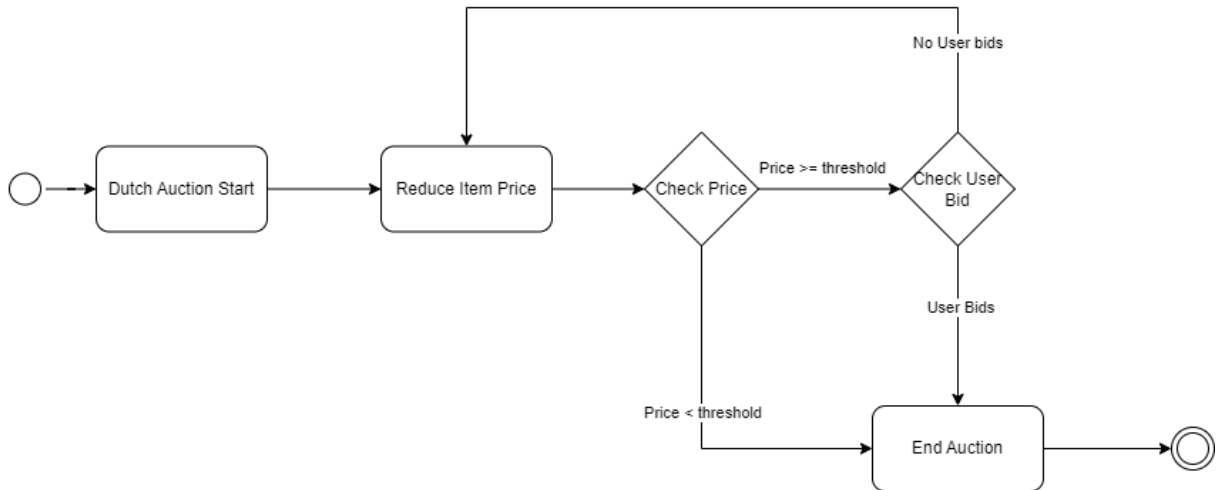
UC3.1: Forward Auction Bidding

When a Forward Auction starts, a timer is set, and users can continue bidding until the auction timer runs out. Each time a bid is placed, the price updates for each user who is currently watching (observers) the auction.



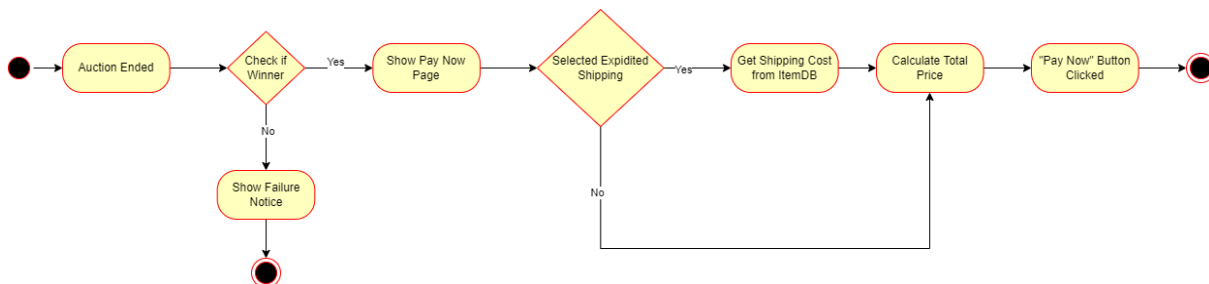
UC3.2: Dutch Auction Bidding

When a Dutch Auction starts, a starting price is set, along with a threshold price (minimum price the item will go for). This price will go down as time goes on, until the first bid is made or hits the threshold price. Once one of these is met, the auction ends.



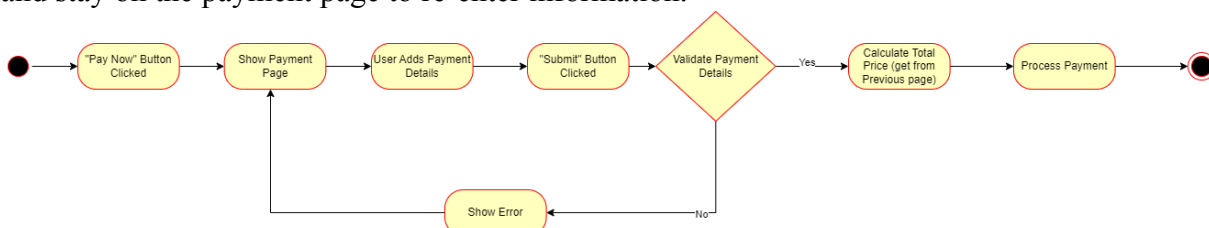
UC4: Auction Ended

After Auction ends, the winning bidder is presented with a confirmation page saying they have won the auction and the option to continue to payment, along with the option of shipping. The page provides the cost of the item as well as item details. Non-winners do not get this page, and are notified instead that they have not won that specific option.



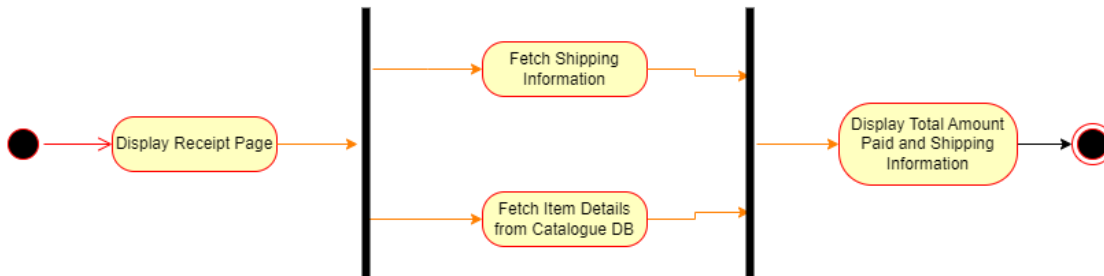
UC5: Payment

Once the user clicks on the “pay now” button (which only the winner of an auction should be able to click), they are routed to the payment page, where payment details are shown. Additionally, the user has available fields to enter credit card information, then click the “Submit” button to process the payment. If the information is correct, payment will go through, otherwise, user will see an error and stay on the payment page to re-enter information.



UC6: Receipt Page and Shipment Details

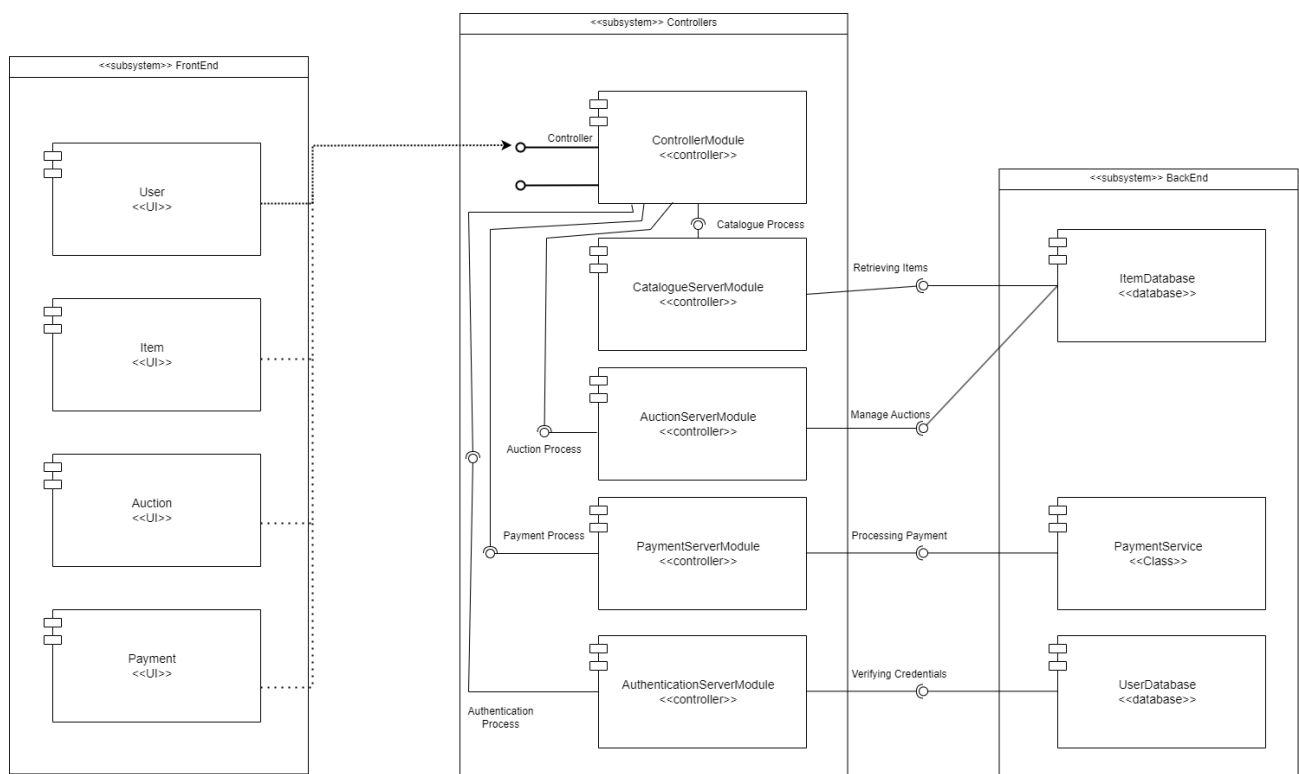
Once payment is cleared, the winning bidder is served with a receipt and shipment information. The page provides the total amount paid as well as the shipping details. You can consider that each item has a separate shipping time.



5 Architecture

Provide an initial decomposition of your system as a collection of interacting modules. If you need nested diagrams please use nesting levels. Include explanations on the functionality of each component. Provide the exposed interfaces of each component and list and briefly describe the functionality (one sentence) of the operations included in each such interface.

Provide a description of each module and its interaction with other modules in the form of the following tables.



| Modules | | | |
|-------------|---|---|--|
| Module Name | Description | Exposed Interface Names | Interface Description |
| User | <p>The User Module manages all aspects of user information within the system. It encompasses user data representation, database interactions, and HTTP request handling.</p> <p>The module allows users to register, log in, update their profiles, and manage their accounts securely. It ensures the integrity and confidentiality of user data while providing a smooth interface for user-related operations.</p> | User:User User:UserDatabase User:UserController | <p>User : User</p> <p>Represents an individual user, encapsulating attributes like email, username, and password. Provides methods for accessing and modifying user data..</p> <p>User : UserDatabase</p> <p>Handles database operations for user data, enabling CRUD functionality for user records.</p> <p>User : UserController</p> <p>Manages HTTP requests related to user operations, exposing API endpoints for sign-up, login, and profile management.</p> |
| Item | <p>The Item Module is responsible for managing all items listed for auction within the system. It encompasses item representation, database interactions, and HTTP request handling.</p> <p>The module facilitates the creation, retrieval, updating, and deletion of auction items, ensuring accurate data management and providing a smooth experience for users participating in auctions.</p> | Item:Item Item:ItemDatabase Item:ItemController | <p>Item : Item</p> <p>Represents an individual item listed for auction, encapsulating attributes such as item ID, name, description, and price. It provides methods for accessing and modifying item data.</p> <p>Item : ItemDatabase</p> <p>Manages database operations for items, enabling CRUD functionality to create, retrieve, update, and delete item records in the auction system.</p> <p>Item : ItemController</p> <p>Handles HTTP requests related to item operations, exposing API endpoints for adding, updating, and retrieving item information in the auction system.</p> |
| Auction | <p>The Auction Module is responsible for managing the auction bidding process within the system. It encompasses auction creation, bidding operations, and auction lifecycle management.</p> <p>The module ensures fair and efficient</p> | Auction:Auction Auction:AuctionController | <p>Auction : Auction</p> <p>Manages the auction bidding process, including functionalities for starting, bidding on, and ending auctions. It encapsulates auction-related data and logic, ensuring proper auction lifecycle management.</p> |

| | | | |
|----------------|---|--|--|
| | handling of auctions, allowing users to participate in forward and Dutch auctions while providing an interface for seamless interaction through HTTP requests. | | Auction : AuctionController Handles HTTP requests related to auction operations, providing API endpoints for starting auctions, placing bids, and retrieving auction details. It coordinates user interactions with the auction system. |
| Payment | The Payment Module is responsible for managing the payment process within the system, ensuring secure transactions and the confidentiality of user financial information. It facilitates the collection of payments for auction items, maintains records of transactions, and provides a user-friendly interface for interacting with payment-related operations through HTTP requests. | Payment:Payment Payment:PaymentController | Payment : Payment Manages payment processing and storage of payment records. It handles the collection of payment details, ensuring secure transactions and confidentiality of user financial information. Payment : PaymentController Handles HTTP requests related to payment operations, providing API endpoints for collecting payments and retrieving transaction history. It facilitates user interactions with the payment system. |
| Authentication | The Authentication Module is responsible for managing user authentication and authorization processes within the system. | Authentication:Authentication | Authentication : Authentication Manages user sign-in and registration processes, ensuring the confidentiality of login credentials. It provides methods for authenticating users, validating session tokens, and managing user sessions securely. |
| Controller | The Controller Module is responsible for managing and routing all HTTP requests in the system. It serves as the interface between users and the application's backend, coordinating interactions with various modules to ensure efficient processing and response delivery. | Controller:Controller | Controller : Controller Manages all HTTP requests within the system, acting as the central point for directing user interactions to the appropriate modules. It orchestrates the flow of data between the user interface and the underlying services, ensuring high performance and efficient request handling. |

| Interfaces | | |
|---------------------|--|--|
| Interface Name | Operations | Operation Descriptions |
| User:User | <String> getUsername() <void> setUsername(String username) | getUsername(): Retrieves the username of the user setUsername(String username) sets or updates the name of the user |
| User:UserDatabase | <void> createUser(String username, String password) <void> updateUsername(<User> users, String username) <Integer> getId(String) | createUser(String username, String password): creates a new user in the database updateUsername(<User> user, String username): updates the username in the database getId(String username): gets the Id from the database by username |
| User:UserController | <HttpResponse> addUser(HttpRequest) <HttpResponse>updateUser(HttpRequest) <HttpResponse>deleteUser(HttpRequest) | addUser(HttpRequest): Processes an incoming HTTP request to create a new user. This operation extracts user details (like email and password) from the request body, validates the information, and interacts with the UserDatabase to save the new user record. Upon successful creation, it returns an HTTP response indicating success, along with the created user's details. updateUser(HttpRequest): Handles HTTP requests to update an existing user's information. It retrieves user details from the request, validates the changes, and then updates the user's record in the UserDatabase. The response includes a confirmation of the update and the updated user information. deleteUser(HttpRequest): Processes a request to delete a user account. It identifies the user to be deleted based on provided identifiers (like user ID or email) in the request. After validating permissions and confirming the user exists, it removes the user from the UserDatabase and returns an HTTP response confirming the deletion. |
| Item:Item | <Integer> getId() <String> getName() <String> getDescription() <Double> getPrice() <void> setName(String name) <void> setDescription(String description) <void> setPrice(Double price) | getId(): Retrieves the unique ID of the item. getName(): Retrieves the name of the item. setName(String name): Sets or updates the name of the item. getDescription(): Retrieves a description of the item. setDescription(String description): Sets or updates the item's description. |

| | | |
|----------------------------|--|--|
| | | <p>getPrice(): Retrieves the price of the item.</p> <p>setPrice(Double price): Sets or updates the price of the item.</p> |
| Item:ItemDatabase | <p><Item> createItem(String name, Double price, String description)</p> <p><Item> getItemById(Integer id)</p> <p>List<Item> getAllItems()</p> <p><void> updateItem(Item item)</p> <p><void> deleteItemById(Integer id)</p> | <p>createItem(String name, Double price, String description): Creates a new item in the database with the specified name, price, and description, and returns the Item object.</p> <p>getItemById(Integer id): Retrieves an item from the database by its unique ID.</p> <p>List<Item> getAllItems(): Retrieves all items from the database.</p> <p><void> updateItem(Item item): Updates the given item in the database.</p> <p><void> deleteItemById(Integer id): Deletes the item with the specified ID from the database.</p> |
| Item:ItemController | <p><HttpResponse> addItem(HttpRequest request)</p> <p><HttpResponse> updateItem(HttpRequest request)</p> <p><HttpResponse> deleteItem(HttpRequest request)</p> | <p>addItem(HttpRequest request): Adds a new item to the catalog based on data from the HTTP request and returns an appropriate response.</p> <p>updateItem(HttpRequest request): Updates an existing item with the information provided in the request and returns an appropriate response.</p> <p>deleteItem(HttpRequest request): Deletes an item based on the item ID provided in the request and returns an appropriate response.</p> |
| Auction:Auction | <p><void> startAuction()</p> <p><void> endAuction()</p> <p><boolean> isActive()</p> <p><void> placeBid(Integer bidAmount, Integer itemId, Userbidder)</p> | <p>startAuction(): Initiates a new auction, setting the necessary parameters such as start time, end time, and item details.</p> <p>endAuction(): Concludes the auction, finalizing the bids and determining the winner.</p> <p>isActive(): Checks if the auction is currently active.</p> <p>placeBid(Integer bidAmount, User bidder): Accepts a bid from a user and updates the auction's highest bid if the new bid is valid.</p> |
| Auction: AuctionController | <p><HttpResponse> createAuction(HttpRequest request)</p> <p><HttpResponse> startAuction(HttpRequest request)</p> <p><HttpResponse> endAuction(HttpRequest request)</p> <p><HttpResponse> placeBid(HttpRequest request)</p> | <p>createAuction(HttpRequest request): Handles requests to create a new auction, processes the input data, and returns a response with auction details or errors.</p> <p>startAuction(HttpRequest request): Accepts a request to start an existing auction and returns a response indicating success or failure.</p> <p>endAuction(HttpRequest request): Accepts a request to end an ongoing</p> |

| | | |
|-------------------------------|--|---|
| | | <p>auction and returns the auction results or status.</p> <p>placeBid(HttpRequest request): Accepts a bid from a user, processes it, and returns the updated auction status or errors.</p> |
| Payment:Payment | <Boolean> processPayment(PaymentDetails) | <p>processPayment(PaymentDetails): Initiates a refund for a specified transaction identified by payment details. Returns true if the refund is successful, false otherwise.</p> |
| Payment:PaymentController | <HttpResponse> processPayment(HttpRequest request) | <p>processPayment(HttpRequest request): Handles incoming HTTP requests to process a payment. Extracts payment details from the request body and calls the processPayment method of the Payment interface.</p> |
| Authentication:Authentication | <boolean> authenticate(String username, String password) <boolean> authenticate(Token token) <User> login(String username, String password) <User> signup(String username, String password, String passwordConfirmation) <Token> generateToken(User user) <void> invalidateToken(Token token) | <p>authenticate(String username, String password): Validates the provided username and password against the stored credentials. Returns true if valid, otherwise false.</p> <p>authenticate(Token token): Validates a session token to check if it is still valid. Returns true if valid, otherwise false.</p> <p>login(String username, String password): Logs in a user with the provided username and password. Returns the associated User if valid; throws an error if invalid.</p> <p>signup(String username, String password, String passwordConfirmation): Registers a new user with the provided credentials. Returns the created User if successful; throws an error if validation fails.</p> <p>generateToken(User user): Generates a new session token for a logged-in user. Returns the generated token.</p> <p>invalidateToken(Token token): Invalidates the specified session token, preventing further use.</p> |
| Controller:Controller | <HttpResponse> startServer() <HttpResponse> stopServer() <HttpResponse> handleRequest(HttpRequest request) | <p>startServer(): Initializes and starts the HTTP server, making the application accessible for incoming requests.</p> <p>stopServer(): Shuts down the HTTP server, terminating all active connections and halting request processing.</p> <p>handleRequest(HttpRequest request): Processes an incoming HTTP request, routing it to the appropriate module or service based on the request type and parameters.</p> |

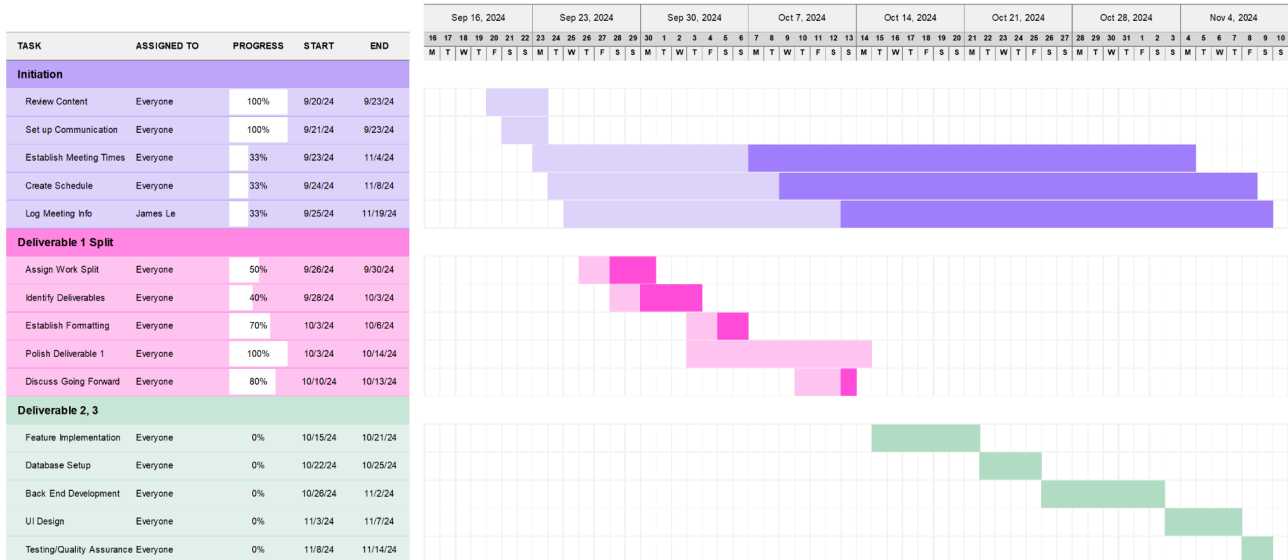
6 Activities Plan

6.1 Gantt Chart

Bidding Blitz

Project start: **Fri, 9/20/2024**

Display week: **1**



6.2 Project Backlog and Sprint Backlog

In this Section, and assuming you follow a Scrum process model, provide a list of product backlog items so that you can select items for your Sprint backlog. Make sure the product backlog list and the tasks in each product backlog item are consistent with the Gantt Chart in Section 6.1. above.

Product Backlog

| Backlog Item | Description | Priority | Est. Effort |
|-------------------------|---|-------------|-------------|
| User Module Development | Implement user registration, login, and profile management using Spring Boot and integrate with a database. | High | 8 points |
| Item Module Development | Develop item creation, retrieval, update, and deletion functionalities for auction items. | High Medium | 10 points |
| Auction Module | Implement auction | High | 15 points |

| | | | |
|----------------------------|--|--------|-----------|
| Development | management, including start, bidding, and end auction processes. | | |
| Payment Module Development | Integrate payment processing functionality, including secure transaction handling and record-keeping. | Medium | 12 points |
| Authentication Module | Build user authentication and authorization functionalities, including token-based security measures. | High | 10 points |
| Controller Module | Design and implement the main controller for handling HTTP requests and routing. | Medium | 6 points |
| Database Schema Design | Develop database schema for all modules (User, Item, Auction, Payment) | Medium | 5 points |
| API Documentation | Create documentation for all API endpoints exposed by the system, covering all modules. | Medium | 3 points |
| Unit Testing | Write unit tests for all modules to ensure functionality and reliability. | Medium | 8 points |
| Integration Testing | Perform integration testing to verify interaction between modules. | Medium | 6 points |
| Frontend Development | Build basic UI for the auction system, with forms for registration, login, item creation, and bidding. | Medium | 12 points |
| Deployment Setup | Prepare environment | Low | 5 points |

| | | | |
|--|---|--|--|
| | for deployment, configure Spring Boot, and set up server. | | |
|--|---|--|--|

6.3 Group Meeting Logs

In this Section you write minutes of each meeting, listing the attendance, what the topics of discussion in the meeting were, any decisions that were made, and which team members were assigned which tasks. These minutes must be submitted with the project report in each deliverable and will provide input to be used for the overall assessment of the project.

| Present Group Members | Meeting Date | Issues Discussed / Resolved |
|----------------------------|--------------|---|
| James, Ryan, Brandon, Eric | 2024/09/25 | Time: 30 minutes Topics of Discussion: Team introductions and discussion of Deliverable 1. Decisions Made: The team decided to split Deliverable 1 into two 1-week sprints. Assigned Tasks: <ul style="list-style-type: none"> James: Responsible for Use Cases 1 and 6 - Sequence and Activity Diagrams Ryan: Responsible for Use Cases 4 and 5 Brandon: Responsible for Use Case 2 Eric: Responsible for Use Case 3 |
| James, Brandon, Eric | 2024/10/02 | Time: 30 minutes Topics of Discussion: <ul style="list-style-type: none"> Review of current diagrams Discussion of services and modules for system architecture Decision on whether to pre-populate auction items or assume a fixed number Decisions Made: <ul style="list-style-type: none"> Diagrams need further refinement. Team members need to coordinate on overlapping use cases during implementation. Next meeting will be held in person to collaboratively work on system architecture. Assigned Tasks: <ul style="list-style-type: none"> Everyone: Complete their sequence and activity diagrams from the last meeting. Begin research on system architecture for the upcoming meeting. |
| James, Ryan, Brandon, Eric | 2024/10/07 | Time: 2 hours |

| | | |
|--|--|---|
| | | <p>Topic of Discussion:</p> <ul style="list-style-type: none"> System architecture (including design patterns and modularization) Test case development <p>Decisions Made:</p> <ul style="list-style-type: none"> Finalized the system architecture and selected design patterns to use Divided test case responsibilities among team members <p>Assigned Tasks:</p> <ul style="list-style-type: none"> Create test cases for each assigned use case Complete documentation for interfaces and modules Clean up sequence diagrams Research the implementation of the ControllerModule Develop the UML component diagram |
|--|--|---|

1 Test Driven Development

Test cases will be provided in the form of a table as follows:

| | |
|-----------------------|--|
| Test ID | TC1 (Ryan) |
| Category | End Auction |
| Requirements Coverage | UC4-Auction-Ended-Winner |
| Initial Condition | Auction Ends |
| Procedure | <ol style="list-style-type: none"> Auction Ends System checks if user is the winner of the auction System confirms user is the winner The Auction UI closes User is presented with the Auction Ended UI, which has the item information and button to proceed to pay now |
| Expected Outcome | User is presented with the Auction Ended UI, which has the item information and button to proceed to pay now |
| Notes | The UI should also include a radio button for the shipping option. |

| | |
|---------|------------|
| Test ID | TC2 (Ryan) |
|---------|------------|

| | |
|-----------------------|---|
| Category | End Auction |
| Requirements Coverage | UC4-Auction-Ended-Non-Winner |
| Initial Condition | Auction Ends |
| Procedure | <ol style="list-style-type: none"> 1. Auction Ends 2. System checks if user is the winner of the auction 3. System finds that user is not a winner 4. The Auction UI closes and the User gets a message saying they did not win and are routed back to the catalog page |
| Expected Outcome | User is routed back to the catalog page |
| Notes | User should get a clear message letting them know that they did not win the auction. |

| | |
|-----------------------|---|
| Test ID | TC3 (Ryan) |
| Category | Payment |
| Requirements Coverage | UC5-Payment-Success |
| Initial Condition | User clicks the pay now button from the Auction Ended Page |
| Procedure | <ol style="list-style-type: none"> 1. User clicks on the pay now button 2. System gets all user information from database, as well as auction item cost, then brings user to the payment page 3. User fills in payment information 4. User clicks on submit to confirm payment 5. System authenticates payment information and user is presented with the Receipt page |
| Expected Outcome | Payment is confirmed and user is presented with the Receipt page |
| Notes | Total cost is calculated at this stage, when user gets routed to the payment page, and the total cost is rendered in |

| | |
|-----------------------|--|
| Test ID | TC4 (Ryan) |
| Category | Payment |
| Requirements Coverage | UC5-Payment-Failure |
| Initial Condition | User clicks the pay now button from the Auction Ended Page |
| Procedure | <ol style="list-style-type: none"> 1. User clicks on the pay now button 2. System gets all user information from database, as well as auction item cost, then brings user to the payment page 3. User fills in payment information 4. User clicks on submit to confirm payment |

| | |
|------------------|---|
| | 5. Payment information is incorrect, and System outputs an error to the user, keeping them on the same payment page |
| Expected Outcome | Payment is denied, and the user stays on the same payment page |
| Notes | Once payment is denied, all the values that the user input are gone, so they have to fill in all values again |

| | |
|-----------------------|--|
| Test ID | TC5 (James) |
| Category | User Sign-Up |
| Requirements Coverage | UC1-UserSignup-Success |
| Initial Condition | The user is on the Signup Page |
| Procedure | <ol style="list-style-type: none"> 1. The user fills in all required fields (username, password) with valid data. 2. The user clicks “Sign Up” button 3. System verifies the data and checks for any existing username conflicts. 4. If verification is successful, the system creates a new account and redirects the user to the Bidding Page. |
| Expected Outcome | User is successfully registered and redirected to the Welcome Page. |
| Notes | All fields are validated before submission to ensure accuracy and compliance with field requirements (e.g., password strength, valid email format). |

| | |
|-----------------------|---|
| Test ID | TC6 (James) |
| Category | User Sign-Up |
| Requirements Coverage | UC1-UserSignup-Failure |
| Initial Condition | The user is on the Signup Page |
| Procedure | <ol style="list-style-type: none"> 1. The user attempts to submit the signup form without filling in one or more requirements. 2. System detects the missing required fields and displays error messages next to each incomplete field. 3. User fills in the missing information and clicks “Sign Up” again. |

| | |
|------------------|--|
| Expected Outcome | System highlights the missing fields with error messages and prevents form submission until all required fields are completed. |
| Notes | The system should guide the user by indicating which fields are missing or need correction. |

| | |
|-----------------------|---|
| Test ID | TC7 (James) |
| Category | User Login |
| Requirements Coverage | UC1-UserLogin-Success |
| Initial Condition | The user is on the Login Page |
| Procedure | <ol style="list-style-type: none"> 1. User enters valid credentials (username and password) 2. User clicks the “Login” button 3. System verifies the credentials and upon validation, grants access and redirects the user to their Dashboard. |
| Expected Outcome | User is logged in successfully and redirected to the Dashboard. |
| Notes | The system should show a loading spinner while verifying the credentials. |

| | |
|-----------------------|--|
| Test ID | TC8 (James) |
| Category | User Login |
| Requirements Coverage | UC1-UserLogin-Failure |
| Initial Condition | The user is on the Login Page |
| Procedure | <ol style="list-style-type: none"> 1. User enters an existing username but an incorrect password. 2. User clicks the “Login” button. 3. System detects that the password is incorrect and displays an error message, such as “Incorrect username or password”. 4. User is prompted to try again or reset their password if needed. |
| Expected Outcome | Login is denied, and an error message is displayed, keeping the user on the Login Page. |
| Notes | The system should prevent multiple failed login attempts in a row by locking the account temporarily after three consecutive failures. |

| | |
|-----------------------|--|
| Test ID | TC9 |
| Category | Forward Auction |
| Requirements Coverage | UC3-Forward-Auction-Bid |
| Initial Condition | The user is on a Forward Auction bidding page of an item that has not timed out |
| Procedure | <ol style="list-style-type: none"> 1. The user selects the bidding input text box 2. The user inputs a bidding price 3. The user selects the 'Bid' button |
| Expected Outcome | The page refreshes. If the submitted bid is higher than the current price of the item, the 'current price' and 'highest bidder' entries are updated on the page. |
| Notes | The user should only provide numbers for the bidding price. |

| | |
|-----------------------|--|
| Test ID | TC10 |
| Category | Forward Auction |
| Requirements Coverage | UC3-Forward-Auction-Time-Out |
| Initial Condition | The user is on Forward Auction bidding page of an item |
| Procedure | <ol style="list-style-type: none"> 1. The user remains on the page as the time for the item expires |
| Expected Outcome | The user is presented with a page indicating that the auction has ended |
| Notes | |

| | |
|-----------------------|--|
| Test ID | TC11 |
| Category | Dutch Auction |
| Requirements Coverage | UC3-Dutch-Auction-Bid |
| Initial Condition | The user is on a Dutch Auction bidding page for an item |
| Procedure | <ol style="list-style-type: none"> 1. The user selects the 'Buy Now' button |
| Expected Outcome | The user is presented with the auction ended page |
| Notes | |

| | |
|----------|---------------|
| Test ID | TC12 |
| Category | Dutch Auction |

| | |
|-----------------------|---|
| Requirements Coverage | UC3-Dutch-Auction-End |
| Initial Condition | The user is on a Dutch Auction bidding page for an item |
| Procedure | 1. A different user selects 'Buy Now' button |
| Expected Outcome | The user is presented with the auction ended page |
| Notes | |

| | |
|-----------------------|---|
| Test ID | TC13 (Brandon) |
| Category | Search Keyword |
| Requirements Coverage | UC2-Search-Valid-Keyword |
| Initial Condition | The user is on the search menu, searching an item |
| Procedure | <ol style="list-style-type: none"> 1. User searches for an item by typing keyword 2. System checks for that keyword in item database 3. System returns the items matching that keyword 4. System displays the items to the user |
| Expected Outcome | User is presented with the catalogue UI containing the matching items |
| Notes | The UI should display those items highly similar first, and the rest with the keyword in the name after. |

| | |
|-----------------------|---|
| Test ID | TC14 (Brandon) |
| Category | Search Keyword |
| Requirements Coverage | UC2-Search-Invalid-Keyword |
| Initial Condition | The user is on the search menu, searching an item |
| Procedure | <ol style="list-style-type: none"> 1. User searches for an item by typing keyword 2. System checks for that keyword in the item database 3. System returns no items matching that keyword 4. System displays no items, and prompts user to search again |
| Expected Outcome | User is prompted to search for an item again |
| Notes | User should get notified they did not find any matching items |

| | |
|-----------------------|--|
| Test ID | TC15 (Brandon) |
| Category | Select item to bid |
| Requirements Coverage | UC2-Select-Item-Bid |
| Initial Condition | User has searched a query for items and finds results. |
| Procedure | <ol style="list-style-type: none"> 1. The user clicks on the radio button of the item they want to bid on |

| | |
|------------------|--|
| | <ol style="list-style-type: none"> 2. The user clicks the bid button 3. The user is prompted to a bidding UI which shows the details of the item bidding |
| Expected Outcome | The user is brought to a bidding page where they can bid on the selected item |
| Notes | The user only has one item selected and clicks bid |

| | |
|-----------------------|---|
| Test ID | TC16 (Brandon) |
| Category | Multiple items selected to bid |
| Requirements Coverage | UC2-Multiple-Items-Bid |
| Initial Condition | User has searched a query for items and finds results. |
| Procedure | <ol style="list-style-type: none"> 1. The user clicks on the radio button of the items they want to bid on 2. The user clicks the bid button 3. The user is prompted that they must only select one item to bid on 4. The user remains on the same page |
| Expected Outcome | Bidding is denied and the user must select only one item to place a bid on |
| Notes | |