EECS 4413 Project Deliverable 1: Software Design Document

Requirements Documentation

Changes Made for Deliverable 3 Appended at End (starts page 26)

Document Change Control Revision History

Version	Date	Author(s)	Summary of changes
1.0	09/23/2025	Willa Nie	Document outline created based on deliverable 1 instructions.
1.1	09/23/2025	Isha Hanchate	Added table of contents, introduction, and product backlog list.
1.2	09/25/2025	Roopkiran Kaur	Added a bit to overview, architectural design, and made a few additions to the module table.
2.0	09/28/2025	Isha Hanchate	Added activity and sequence diagrams for use cases UC2.1 and UC2.2.
2.1	09/29/2025	Willa Nie	Added 4 test cases covering UC4 and UC7.
2.2	09/29/2025	Kimberly Bonilla	Added activity and sequence diagrams for assigned use cases.
2.3	09/30/2025	Kimberly Bonilla	Added 4 test cases covering UC2.3 and UC5.
2.4	09/30/2025	Isha Hanchate	Added 4 test cases covering UC2.1 and UC2.2.
3.0	10/01/2025	Roopkiran Kaur	Added 4 test cases covering UC1.1 AND UC1.2.
3.1	10/02/2025	Willa Nie	Added activity & sequence diagram for UC4 and UC7; made some additions to module/interface tables. Added gantt chart.
3.2	10/02/2025	Kris Singh	Added sequence + activity diagram for UC3, added to modules and interfaces, added 4 test cases covering UC3
3.3	10/02/2025	Roopkiran Kaur	Added sequence and activity diagram for UC1 signup/login
3.4	10/02/2025	Kimberly Bonilla	Added to module/interface tables.
4.0	10/03/2025	Kris Singh	Added Subsystems and Design Patterns
4.1	10/03/2025	Willa Nie + Kimberly Bonilla	Added Component Diagram for Architectural Pattern.
4.2	10/03/2025	Kris Singh	Added activity diagram for UC6, added to interface/modules.

Table of Contents:

1. Introduction

1.1 Purpose:
The purpose of this project is to design and implement an online ecommerce auction web application with a three-tier architecture, supporting functionalities such as user management, auction lifecycle, user bidding, and other advanced features. The site will allow users to register an account, login, browse a catalog of products, place bids, and conduct payment transactions.

1.2 Overview:
This project develops an e-commerce auction web platform which lets users register, browse products, bid and make payments securely. The goal is to make this website easy to use, flexible to grow and organized so that it's simple to maintain. In order to better facilitate its development, the project is structured into three main tiers, each responsible for a different layer of application functionality:

Back-End
- Implement business logic and handle data storage and retrieval.
- Include modules for users, auctions, bids, and payments.
- Connected to a relational database (e.g., SQL).

Middle Tier
- Manage communication between the front-end and back-end systems.
- Built using a Java-based web framework (e.g., Spring Boot, Java).
- Handle routing requests, data transformation, and enforce modularity for possible microservice-based deployment.

Front-End
- Provide a user interface for accessing site features.
- Built using standard HTML, CSS, and front-end frameworks (e.g. JSP).
- Communicate with the middle layer through REST APIs to display dynamic content and perform user interactions.

1.3 References:
- 2025_LE_EECS_F_4413__3_E_EN_A_LECT_01: Project Specifications,Deliverables | eClass
- EECS-4413-Project-Description-Use-Cases-2025-F.pdf
- EECS4413-Project-A1-Instructions.pdf

2. Major Design Decisions

2.1 Modularization Criteria:
High Cohesion: Each module focuses on a single responsibility such as Authentication only handles user login/ signup.
Low Coupling: Modules interact strictly through defined interfaces and APIs. This reduces the interdependencies and thus makes the system easier to maintain.

2.2 Chosen Architectural Pattern:
We chose a Three-tier/ Layered Architecture which consists of Presentation Layer which provides UI/ front-end, Business Layer which handles logic and computation, and Data Access Layer which interacts with databases and storage. This architectural pattern is scalable, flexible and maintainable because the layers can be changed without having a major impact on the other layers.

Architectural pattern that was considered but not chosen:
We considered Monolithic Architecture at first but later rejected it because it is harder to maintain and scale as compared to our chosen architectural pattern. Besides, it is difficult to assign tasks that are independent of each other across team members. Additionally, there is always a risk of breaking the entire system when the updates are made to one feature.

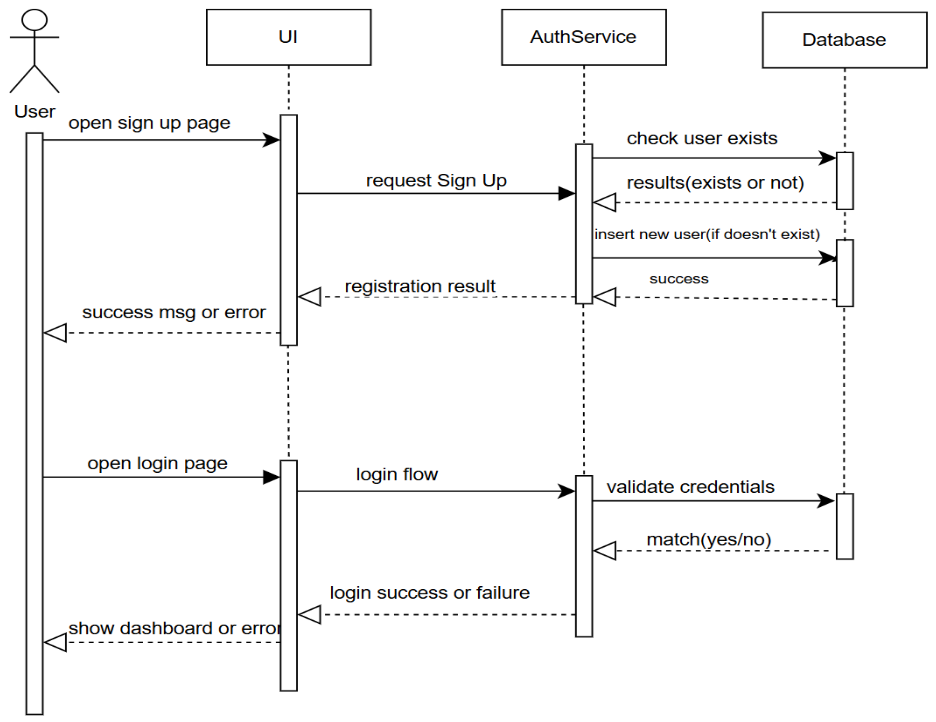
2.3 Subsystems and Design Patterns
Our project will be using several subsystems to handle activity across the application. This includes the following: Authentication, Catalogue, Auction, Payment, and GatewayApp.

The GatewayApp subsystem will employ a **facade** pattern to handle routing certain methods to the correct classes within the program. This helps keep the code readable and simplifies the logic behind the system.

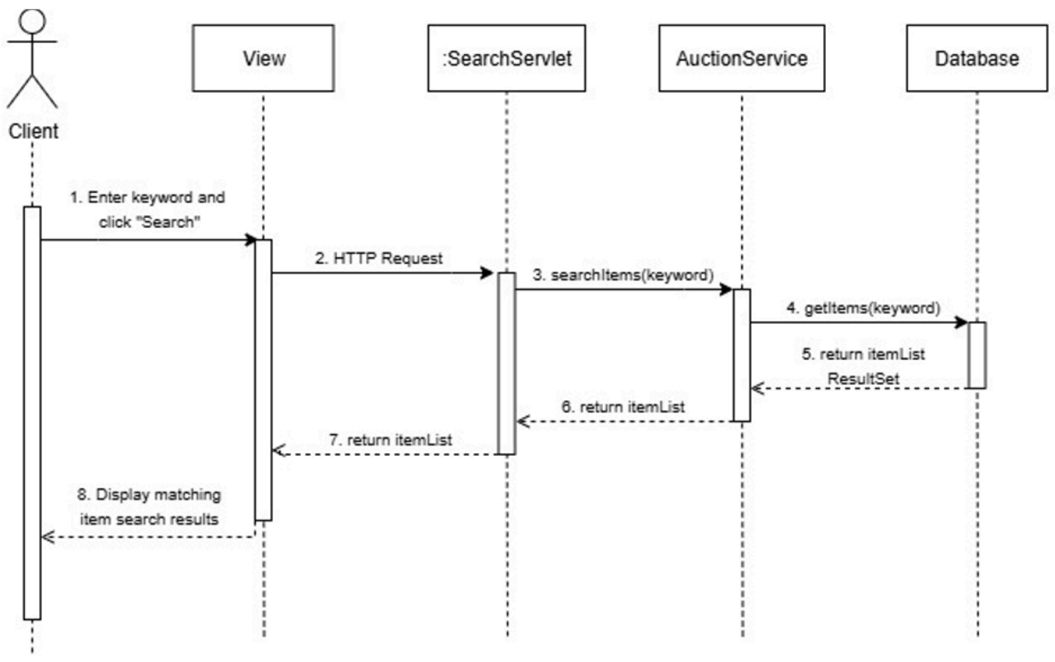
Additionally, we plan to employ an **observer** pattern to handle refreshing the webpage when an auction bid is updated or when an auction ends. This pattern will ensure that the users are always seeing up-to-date information. This pattern will work by checking when certain updating methods are called and changes are confirmed and observed. It will be implemented along with the GatewayApp which will be calling these methods.

3. Sequence Diagrams

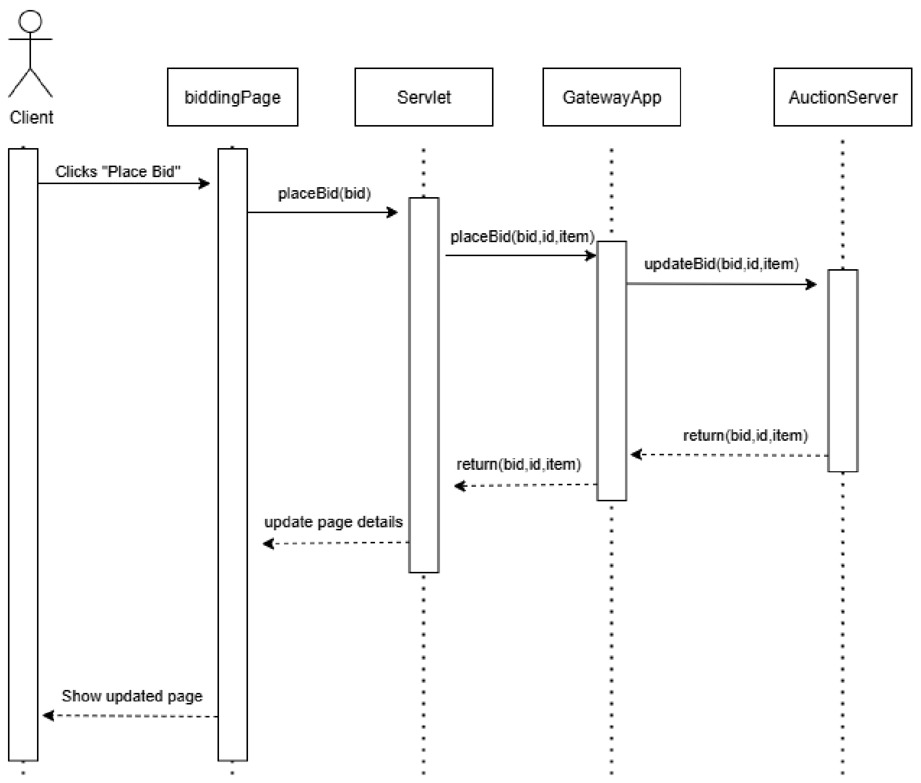
#1 - Use Case 1.1: Sign Up

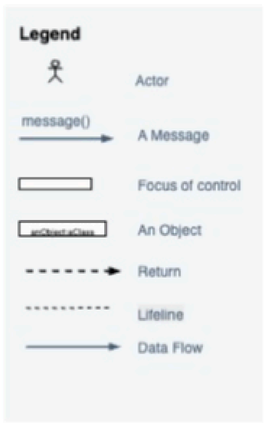


#2 - Use Case 2.1: Item Search

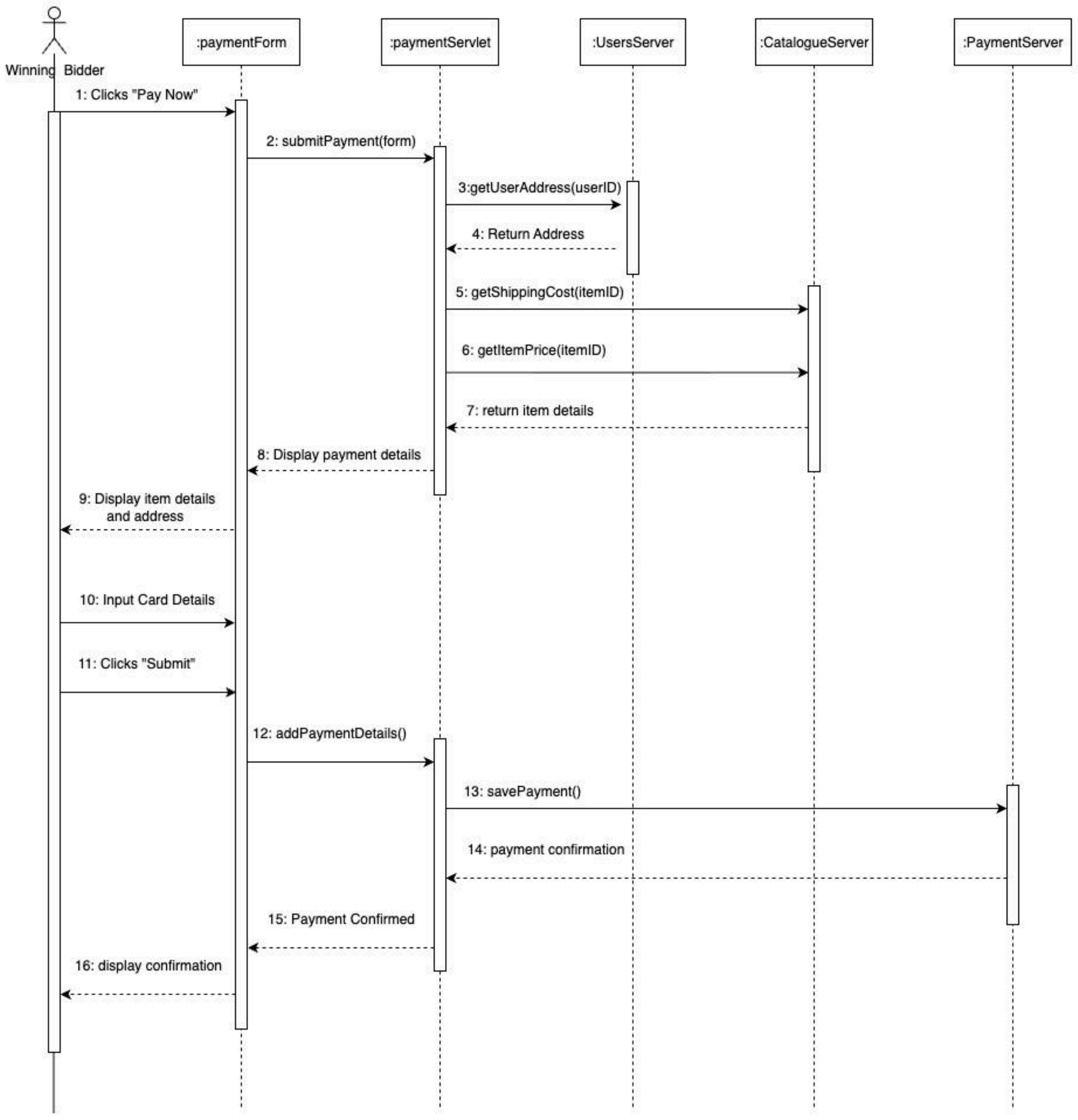


#3 - Use Case 3: Auction Bidding

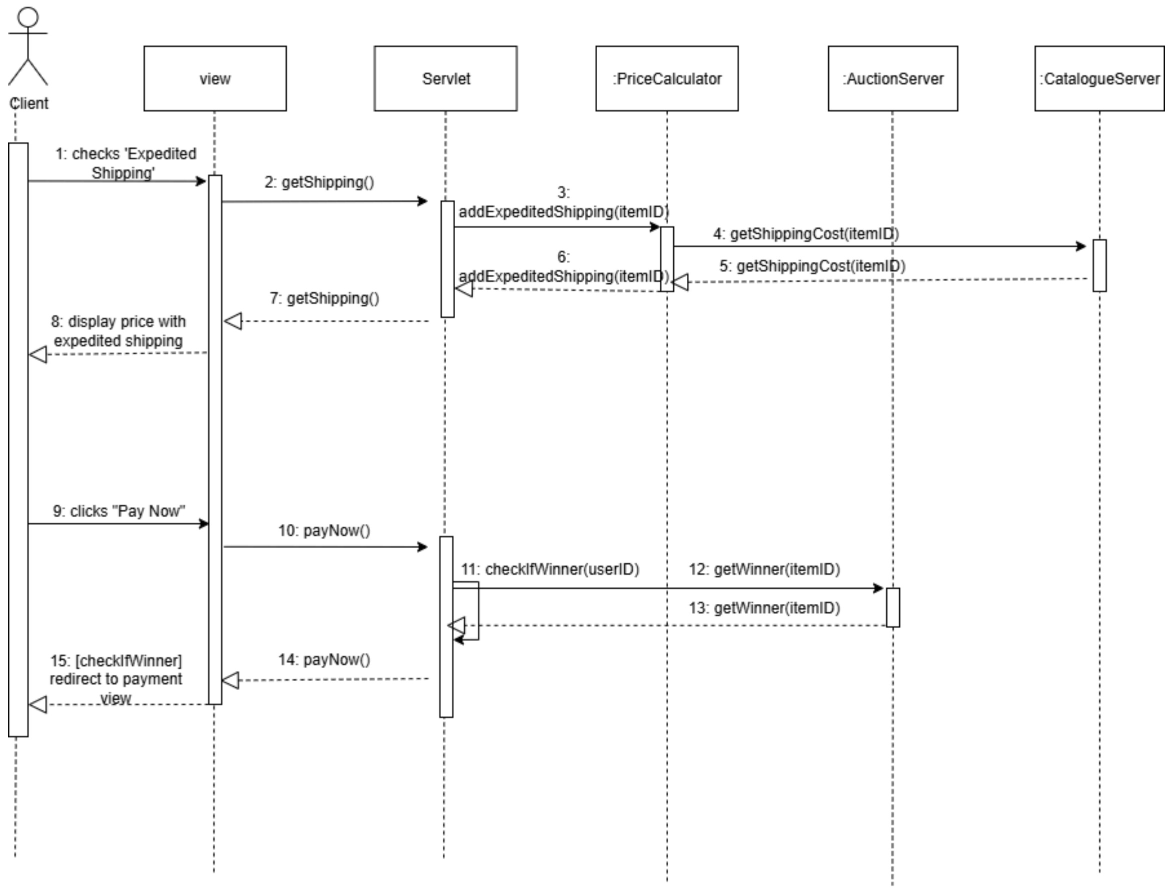




#4 - Use Case 5: Payment Details



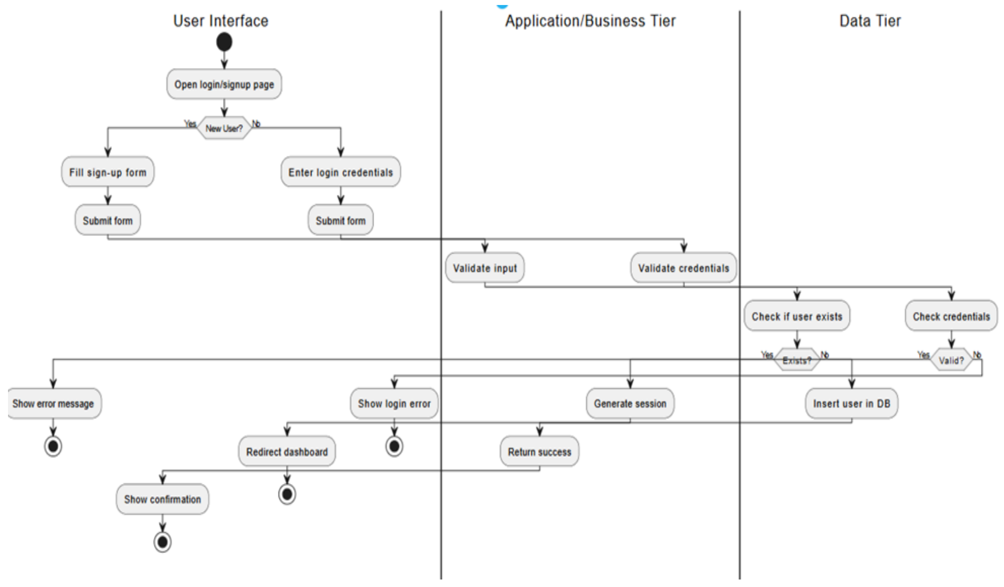
#5 - Use Case 4: Auction Ended


Client
view
Servlet
:PriceCalculator
:AuctionServer
:CatalogueServer
1: checks 'Expedited Shipping'
2: getShipping()
3: addExpeditedShipping(itemID)
4: getShippingCost(itemID)
5: getShippingCost(itemID)
6: addExpeditedShipping(itemID)
7: getShipping()
8: display price with expedited shipping
9: clicks "Pay Now"
10: payNow()
11: checkIfWinner(userID)
12: getWinner(itemID)
13: getWinner(itemID)
14: payNow()
15: [checkIfWinner] redirect to payment view

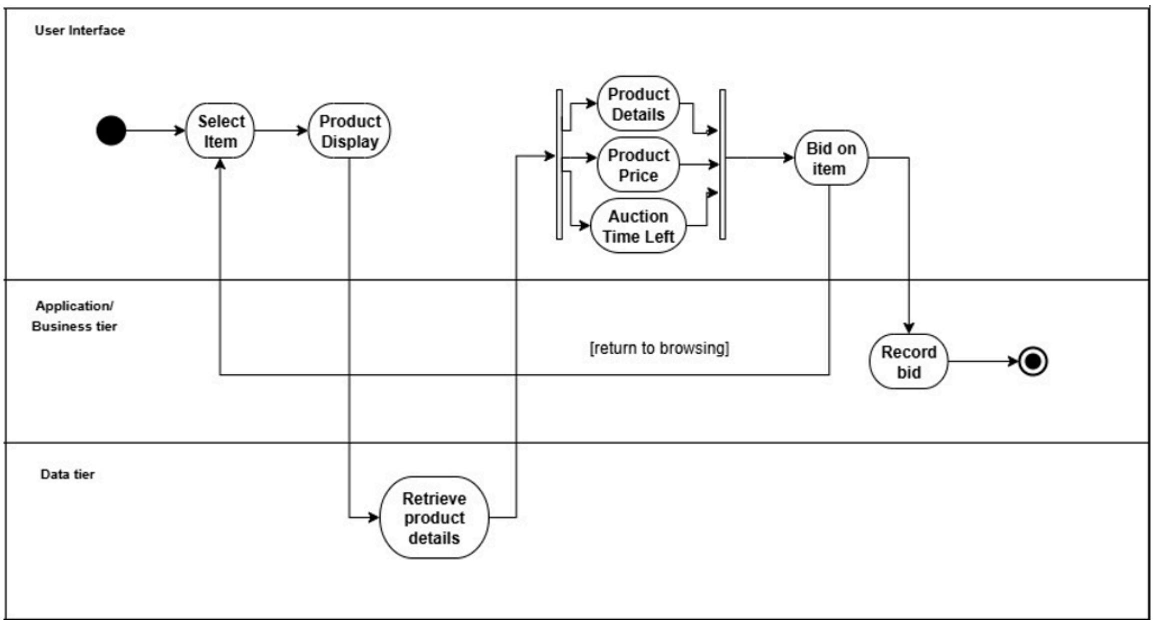
4. Activity Diagrams

Diagrams for 4 use cases - described in project description document

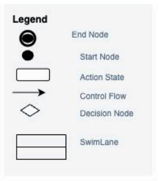
#6 - Use Case 1.2 Login/Sign in



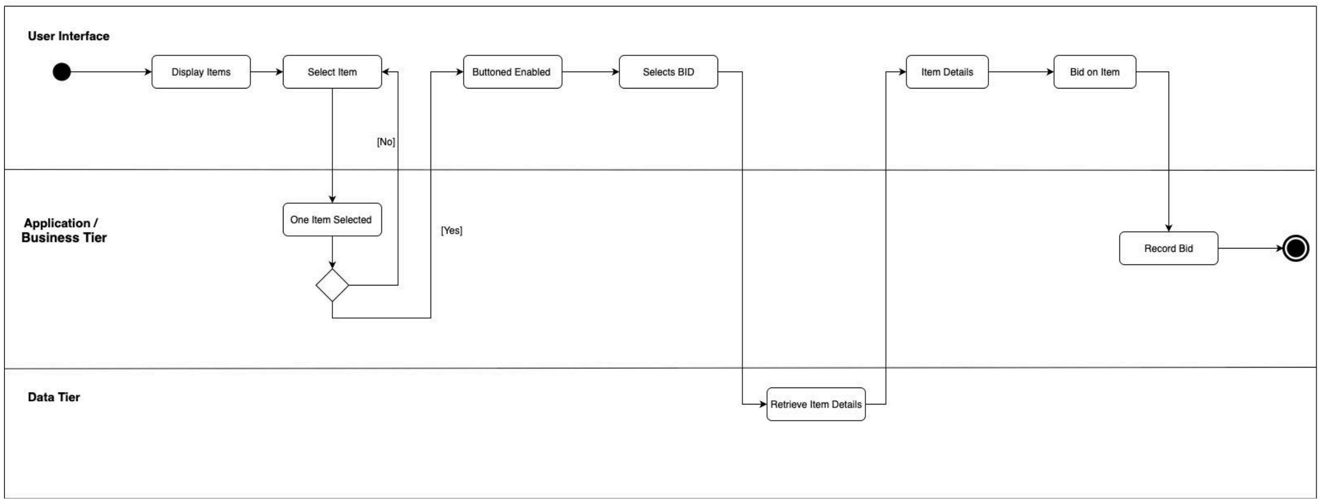
#7 - Use Case 2.2: Display Auctioned Items



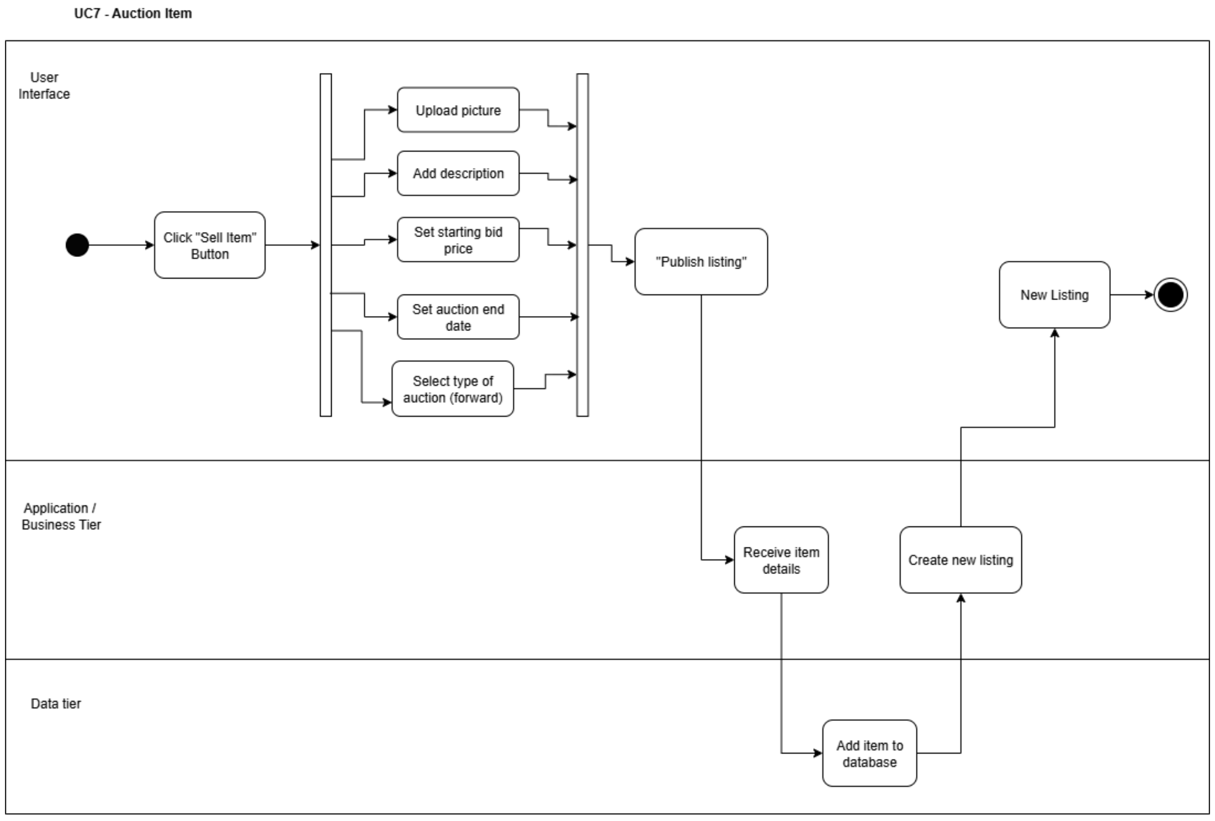
#8 - Use Case 2.3: Item selection



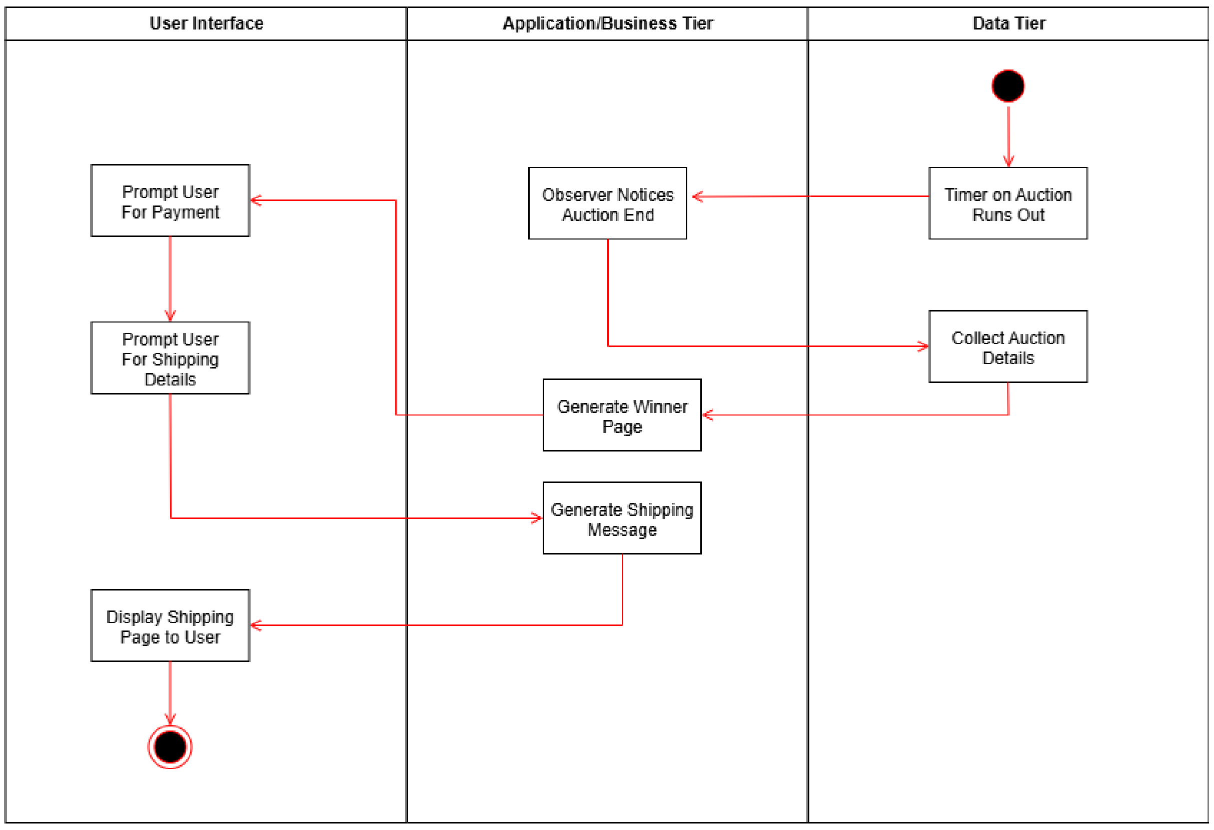
UC2.3. Item Selection



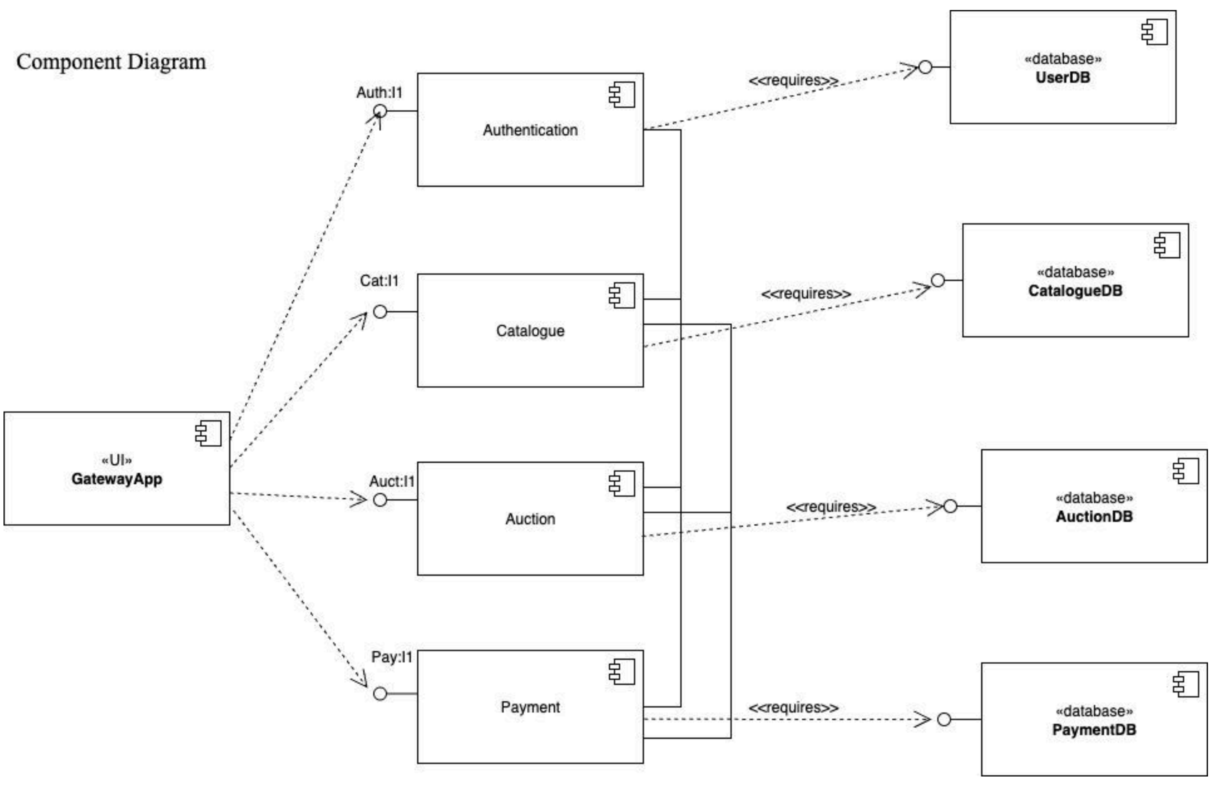
#9 - Use Case 7: Auction Item



#10 - Use Case 6: Receipt Page and Shipment Details


User Interface
Application/Business Tier
Data Tier
Timer on Auction Runs Out
Observer Notices Auction End
Collect Auction Details
Generate Winner Page
Prompt User For Payment
Prompt User For Shipping Details
Generate Shipping Message
Display Shipping Page to User

5. Architecture



Modules

Module Name	Description	Exposed Interface Names	Interface Description
Authentication	Handles user signup, login	Auth:I1	**Auth:I1** "Provides methods for user registration, login, logout"
Catalogue	Displays available items, handles creation of new listings	Cat:I1	**Cat:I1** - Provides methods to get items from catalogue and add new items to catalogue.
Auction	Handles bidding, auction ending	Auct:I1	**Auct:I1** - Provides methods for accessing and reading/writing to the Auction Server/Database.
Payment	Handles payment, validates payment methods.	Pay:I1	**Pay:I1** Provides methods for payment validation

			and to process payments
GatewayApp	Handles a variety of functions in the bidding lifecycle, interacting with other modules.	Gtwy:I1	**Gtwy:I1** - Provides methods for handling Auction processes and communicating with other modules.

Interfaces

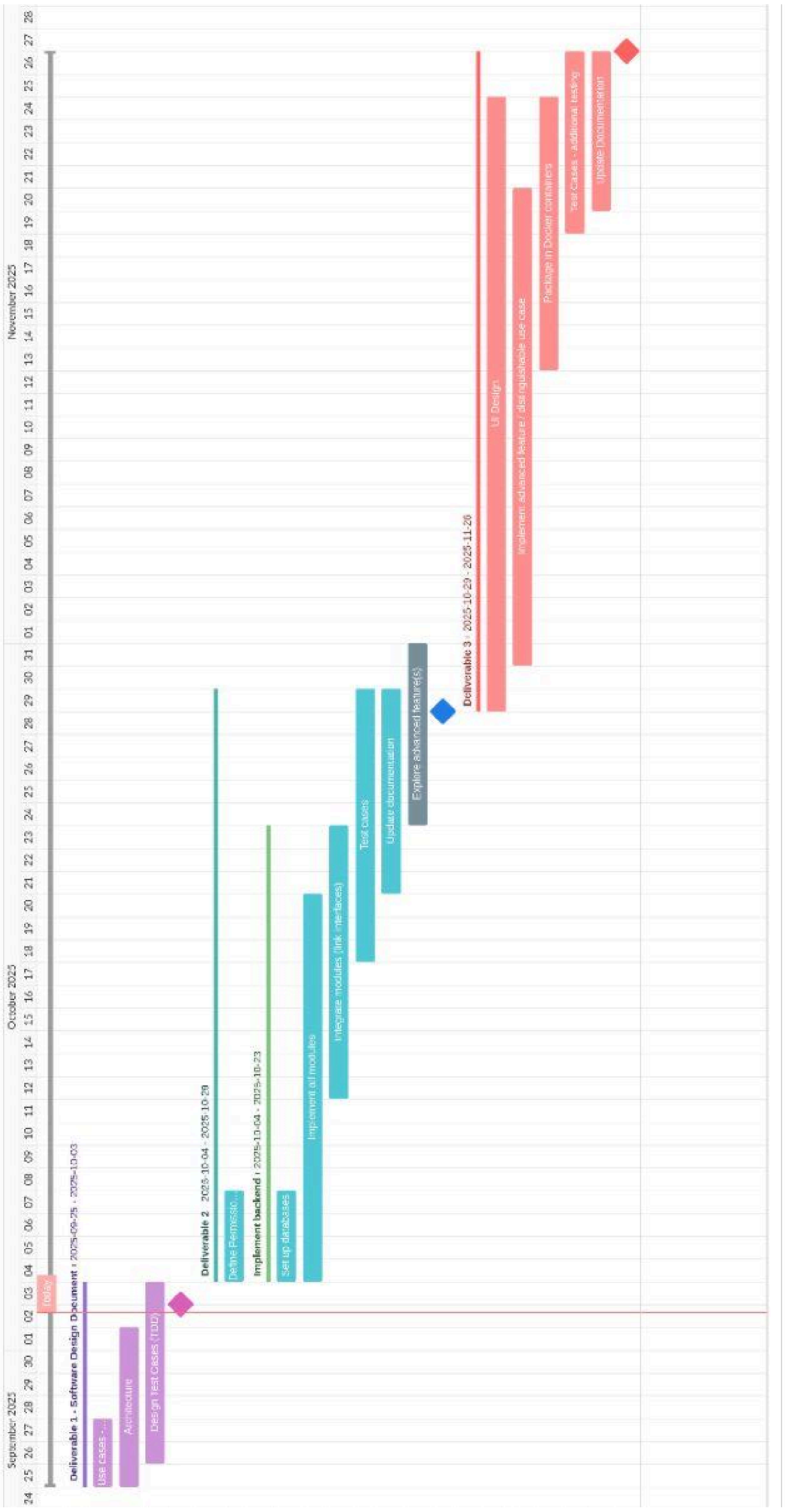
Interface Name	Operations	Operation Descriptions
Auth:I1	<boolean> Auth:I1:register user Login user: Session Logout user: boolean	Validates credentials, creates new users, starts and ends sessions
Cat:I1	<boolean> Cat:I1:addItem() <Object> Cat:I1:getItem()	**addItem()** - adds new item to catalogue **getItem()** - displays item from catalogue
Auct:I1	<Object> Auct:I1:getBid() <void> Auct:I1:updateBid(bid,id,item) <Object> Auct:I1:auctionEnd()	**getBid()** - returns details of bids from Auction Server. **updateBid(bid,id,item)** - Updates current bid details of the chosen item. **auctionEnd()** - Pings gateway app when an auction timer ends.
Pay:I1	<boolean> Pay:I1: addPaymentDetails() <boolean> Pay:I1: validateCard()	**addPaymentDetails()** - handles payment transaction **validateCard()** - validates card details
Gtwy:I1	<Object> Gtwy:I1:getBid() <void> Gtwy:I1:placeBid(bid,id,item) <Object> Gtwy:I1:auctionEnd()	**getBid()** - returns details of bids from Auction Server. **placeBid(bid,id,item)** - Communicates with Auction server to update highest bid. **auctionEnd()** - handles end of auction page generation.

6. Activities Plan, Product Backlog, Sprint Backlog

6.1 Complete product backlog list

Deliverable 1	- Set out project specifications - Create use case diagrams for all user interactions - Sign up/Login - Browse Catalog - Auction Bidding - Payment - Choose architecture style and programming languages - Define components (Auth, Auction, Bid, Payment, etc.) - Design key classes: User, Auction, Bid, Item, Payment - Document endpoints for each component (e.g., POST /auctions, GET /bids)
Deliverable 2	- Fully implement back-end (business logic) i.e. outlined modules - Implement user sign up (UC1.1) - Implement user login (UC1.2) - Implement catalog browsing (UC2) - Implement bidding, choose auction winner (UC3; UC4) - Implement payment details for auction winner (UC5; UC6) - Implement auction item feature for sellers (UC7) - Define permissions (Admin vs User) - Design & set up databases - Test cases - Update documentation to reflect implementation
Deliverable 3	- Finalize project; ensure full system functionality - UI design - Implement a distinguishable feature (TBD) - Package application in Docker containers - Test cases - Update documentation to reflect implementation

6.2 GANTT diagram with schedule of planned activities (also see end of document for full attachment)



6.3 Group meeting logs
Write minutes of each meeting, list attendance, topics of discussion, any decisions made, which team members were assigned which tasks.

Date	Meeting Log
Friday Sept 19th meeting (in class)	- All team members present - Discussed plan to work on deliverable 1 - Decided four members will each pick a use case and create the sequence and activity diagram for that use case - Fifth member will be responsible for product backlog list and GANTT diagram - Aim to complete use cases in the upcoming week, and regroup to discuss architecture together - To work on test cases in the week after (every member will do 4)
Friday Sept 26th meeting (in class)	- All team members present - Revised plan to work on deliverable 1 - Architecture style decided - Decided everyone will each pick 2 different use cases and create the sequence and activity diagrams - Rest of tasks distributed among team - To do: complete use cases and test cases, as well as component charts in the upcoming week
Tuesday Sept 30 Meeting (lab hours)	- Discussed architecture pattern. - Revised all activity and sequence diagrams and made sure they are consistent with each other. - To do: complete test cases and finalize on architecture pattern (including diagram).
Friday October 3rd Meeting (in class)	- Finished system architecture - Finished system component diagram - Reviewed document, made final changes

7. Test Driven Development (TDD)

Test ID	UC1-T1
Category	Authentication
Requirements coverage	UC1- Successful-User-Login
Initial condition	User exists in DB

Procedure	1. User selects login 2. Enters valid username+ password 3. Clicks “Login”
Expected outcome	User is logged in and redirected to the dashboard
Notes	Create valid session token.

Test ID	UC1-T2
Category	Authentication
Requirements coverage	UC1-Failed-Login-Invalid-Credentials
Initial condition	User exists in DB
Procedure	1. User enters invalid password 2. Clicks “Login”
Expected outcome	System displays error message , login fails
Notes	No session created.

Test ID	UC1-T3
Category	Registration
Requirements coverage	UC1-Successful-SignUp
Initial condition	DB empty
Procedure	1. User enters valid details 2. Clicks “Sign Up”
Expected outcome	Account created, user redirected to login page
Notes	The password must meet the requirement.

Test ID	UC1-T4
Category	Registration
Requirements coverage	UC1-Failed-SignUp-Existing-User
Initial condition	Username already in DB
Procedure	1. User enters existing username 2. Clicks “Sign Up”
Expected outcome	Error message shown, signup blocked
Notes	The system prevents duplicates.

Test ID	UC2.1-T1
Category	Item search
Requirements coverage	UC2.1-Item Search-Valid
Initial condition	● User is logged in
Procedure	1. User enters keyword into search bar. 2. User clicks “SEARCH” button. 3. User is shown all results for products matching keyword listed on the site.
Expected outcome	User has completed a valid search and is shown all items matching the keyword entered..
Notes	

Test ID	UC2.1-T2
Category	Item search
Requirements coverage	UC2.1-Item-Search-Invalid
Initial condition	● User is logged in
Procedure	1. User enters a random or invalid keyword that does not match any product into search bar. 2. User clicks “SEARCH” button.. 3. User is shown a message indicating that no matching items were found.
Expected outcome	User has not completed a valid search and a message indicating no matching results is displayed.
Notes	Ensure no items are listed and UI handles empty results.

Test ID	UC2.2-T1
Category	Auction Display
Requirements coverage	UC2.2-Auction-Display-Valid
Initial condition	● User is logged in ● User has performed a successful item search.
Procedure	1. User is shown a list of auctioned items matching keyword. 2. User selects item from results list. 3. User clicks on item “View Details” button. 4. User is redirected to item details page, along with

	bid option. 5. Each item includes full item name, current bidding price, auction type, and remaining time.
Expected outcome	User is redirected to a item details page where they can proceed to bid for the item.
Notes	

Test ID	UC2.2-T2
Category	Auction Display
Requirements coverage	UC2.2-Auction-Display-Invalid
Initial condition	● User is logged in ● User has performed a successful item search.
Procedure	1. User is shown a list of auctioned items matching keyword. 2. User selects item from results list. 3. User clicks on item “View Details” button. 4. User is redirected to item details page, along with bid option. 5. User attempts to bid on item even though auction has ended. 6. User is shown a message indicating that the auction for the item has ended.
Expected outcome	User is shown a message indicating the auction has ended on item display page.
Notes	

Test ID	UC2.3-T1
Category	Item selection
Requirements coverage	UC2.3-Item-Selection-Valid
Initial condition	● User is logged in
Procedure	1. User views catalogue of items. 2. User selects one item to bid for. 3. User clicks “BID” button. 4. User is redirected to item details page, along with bid option. .
Expected outcome	User is redirected to a item details page where they can proceed to bid for the item.
Notes	Only one item can be selected for each log-in session through radio buttons.

Test ID	UC2.3-T2
Category	Item Selection
Requirements coverage	UC2.3-Item-Selection-Invalid
Initial condition	● User is logged in
Procedure	1. User views catalogue of items. 2. User selects more than one item to bid for. Or user selects no items to bid for. 3. User clicks “BID” button. 4. User receives error message saying only one item can be selected to bid for.
Expected outcome	User receives warning to select only one item to bid for.
Notes	Either none or more than one item is selected for each log-in session through radio buttons.

Test ID	UC3-T1
Category	Auction Bidding
Requirements coverage	UC3 - Successful Bid Placement
Initial condition	● User is logged in ● Placed bid is higher than current highest bid
Procedure	1. User chooses an item to bid on. 2. User enters amount to bid. 3. User clicks on “Place Bid” button. 4. Web page refreshes, showing new highest bid.
Expected outcome	Highest bid on the item changes to user’s bid, web page displays new bid amount and user’s id.
Notes	

Test ID	UC3-T2
Category	Auction Bidding
Requirements coverage	UC3 - Failed Bid Placement
Initial condition	● User is logged in ● Placed bid is less than/equal to current highest bid

Procedure	1. User chooses an item to bid on. 2. User enters amount to bid. 3. User clicks on “Place Bid” button. 4. User gets an error saying “New bid must be higher than current highest bid”
Expected outcome	The user’s bid fails, and the current bid stays as the highest bid for the item.
Notes	

Test ID	UC3-T3
Category	Auction Bidding
Requirements coverage	UC3 - 2 Users bid at the same time
Initial condition	● Users are logged in ● Bids are valid ● One bid is higher than the other
Procedure	1. Both users select the same item to bid on. 2. User 1 enters a lower bid than User 2 3. Both users click on “Place Bid” button. 4. User 1 gets and error message saying “New bid must be higher than current highest bid”
Expected outcome	User 1’s bid is invalid and results in an error, updating the page to show User 2’s new bid as the current highest.
Notes	Will need to add clauses to the code to handle the scenario where if User 1’s bid goes through before User 2’s, it still provides the error even though in the system it is technically a valid bid.

Test ID	UC3-T4
Category	Auction Bidding
Requirements coverage	UC3 - Non-User attempts to bid
Initial condition	● User is not logged in ● Bid is valid
Procedure	1. User selects an item to bid on. 2. User enters a valid bid. 3. User clicks on “Place Bid” button. 4. User gets an error saying “Only valid users may bid, please log in or sign up.”
Expected outcome	The bid does not go through and the user receives an appropriate error message.

Notes	

Test ID	UC4-T1
Category	Auction Ended
Requirements coverage	UC4-Auction-Ended-Winner
Initial condition	● User is logged in ● Auction of specified item has ended ● User is winner of the auction
Procedure	1. User views item 2. User selects ‘Expedited Shipping’ 3. User clicks ‘Pay now’ button. 4. User is redirected to payment details page.
Expected outcome	User is redirected to a page to enter payment details.
Notes	

Test ID	UC4-T2
Category	Auction Ended
Requirements coverage	UC4-Auction-Ended-Not-Winner
Initial condition	● User is logged in ● Auction of specified item has ended ● User is not the winner of the auction
Procedure	1. User views item 2. User selects ‘expedited shipping’. 3. User clicks ‘Pay Now’ button 4. User receives error message stating that they are not the winner of the auction.
Expected outcome	User receives an error message; remains on the item auction view.
Notes	

Test ID	UC5-T1
Category	Payment Details
Requirements coverage	UC5-Successful-Payment
Initial condition	● User is logged in ● User is the winner of the auction ● The user has a valid payment card.

Procedure	1. User clicks “Pay now” 2. User views address and item details. 3. The user adds valid payment details: credit card number, name on the card, card expiration date, card security code. 4. User clicks “Submit” 5. User is redirected to receipt and shipment information page.
Expected outcome	User is redirected to the receipt and shipment information page to view payment confirmation details.
Notes	Valid payment card - must meet all the required input formats, empty inputs not accepted.

Test ID	UC5-T2
Category	Payment Details
Requirements coverage	UC5-Unsuccessful-Payment
Initial condition	● The user is logged in. ● The user is the winner of the auction. ● The user does not have a valid payment card.
Procedure	1. User clicks “Pay now” 2. User views address and item details. 3. The user adds invalid payment details. 4. User clicks “Submit” 5. User receives error message stating that payment was unsuccessful or missing payment details.
Expected outcome	User receives an error message; remains on the same payment details page.
Notes	Invalid payment card - does not meet all the required input formats, or input fields are empty.

Test ID	UC7-T1
Category	Auction Item
Requirements coverage	UC7-Successful-Auction-Item
Initial condition	● User is logged in
Procedure	The list of steps required for this test case (e.g. 1. User clicks button to sell item

	2. User provides information about the item a. Picture b. Description 3. User selects auction type (forward auction) 4. User sets starting bid 5. User sets end date 6. User clicks “publish listing” button 7. User is shown the item which is now being auctioned.
Expected outcome	A new auction is started and the user is presented with the view of the new item being auctioned.
Notes	

Test ID	UC7-T2
Category	Auction Item
Requirements coverage	UC7-Unsuccessful-Auction-Item
Initial condition	● User is logged in
Procedure	1. User clicks button to sell item 2. User provides information about the item a. Picture b. Description 3. User selects auction type (forward) 4. User does not set starting bid 5. User does not set end date 6. User clicks “publish listing” button 7. User receives warning that a starting bid and/or end date for the auction is required.
Expected outcome	User receives warning describing missing input, remains on the same view.
Notes	User should receive a warning if any of the item information is missing (picture, description, starting bid, end date, auction type).

EECS 4413 Project - Deliverable 2 Appendix

Document Change Control Revision History (Deliverable 2)

Version	Date	Author(s)	Summary of changes
1	11/1/2025	Isha Hanchate	Added meeting minutes to meeting logs
1.1	11/2/2025	Kris Singh	Added changed design decisions/architecture + patterns
1.2	11/2/2025	Kris Singh	Added implementation decision section
1.3	11/3/2025	Willa Nie	Addressed changes from sequence diagrams.

2.2-B Chosen Architectural Pattern (addendum)
Our overall architecture for deliverable 2 is based on a microservices architecture with RESTful APIs exposed by each service. The **GatewayApp** to control request routing, redirecting HTTP requests to the different components: **Auction**, **Bid**, **User**, **Catalogue**, and **Payment**. This was done in preparation for deliverable 3 where it is required to deploy the microservices in containers. This also keeps the code clean and easily modifiable for future additions.

Remnants of our initial architecture decision are still present in our code, with the DAO acting as the data layer as it manages database referencing for the different services. The GatewayApp acts as the presentation layer. Despite not having front-end implementation, it takes on the role of handling requests and routing them as per the microservices architecture. Finally, the microservices themselves act as the Business Layer, handling computations and logic-related actions.

2.3-B Subsystems and Design Patterns (addendum)
Our project will be using several subsystems to handle activity across the application. This includes the following: Authentication, Catalogue, Auction, Payment, and GatewayApp.

The GatewayApp subsystem employs a **facade** pattern to handle routing certain methods to the correct microservices within the program. This helps keep the code readable and simplifies the logic behind the system.

Additionally, we employed an **observer** pattern to handle the AuctionEnd functionality. The Auction microservice uses the pattern to observe when auctions end in the database, and alerts the DAO when AuctionEnd logic should be applied.

2.4-B Implementation Decisions (addendum)
With some aspects of the project being vague, we took the liberty to define how our project will handle these requirements and implement them as such.

One such decision has to do with the timer for the auction end use case. We elected to have all auctions end on solid hours, like 5:00. With this logic, we can perform checks once every hour to determine if any auctions have ended, and then running the respective auction end logic. The main issue when figuring out a solution is that it is not possible to have the SQL database ping our servlets, so instead we realized we would need to manually check the database instead. By having auctions end on clean hours, we are able to perform checks infrequently, thus limiting the amount of calls and keeping the application running optimally.

This will be enforced on the frontend by only allowing users to select auction end times on the hour. For example by having a dropdown of options rather than allowing them to choose just any time.

For our project code, some components were referenced using ChatGPT.

3.2-B Meetings Log (Deliverable 2)

Wednesday October 22 Meeting (Zoom)	- All team members present - Discussed and identified milestone 2 requirements for the backend - Divided up use cases to be implemented among the group (each person takes 2 use cases) - Set up shared github repo to upload code
Friday October 31 Meeting (in class)	- General update of progress - Addressed implementation questions - Confirmed tasks to complete for milestone 2 submission and what still remained to complete

4.-B Changes from Sequence Diagrams

Since we decided to use a microservices architecture, our backend implementation differs somewhat from the original depiction in the sequence diagrams provided in deliverable 1. Most significantly, each service (Payment, Catalogue, User and Auction) accesses data by interacting with a DAO, rather than accessing the database directly. Additionally, the Gateway app routes methods to the correct microservice, so requests made by the user would be received by the Gateway app first.

Additionally, in the sequence diagram for UC2.1, it was originally depicted that item search would be handled by an Auction service. It will now actually be handled by the Catalogue service, as we have a separate Auction service that deals specifically only with auctions and bids, while the Catalogue service stores information about the items.

EECS 4413 Project - Deliverable 3 Appendix

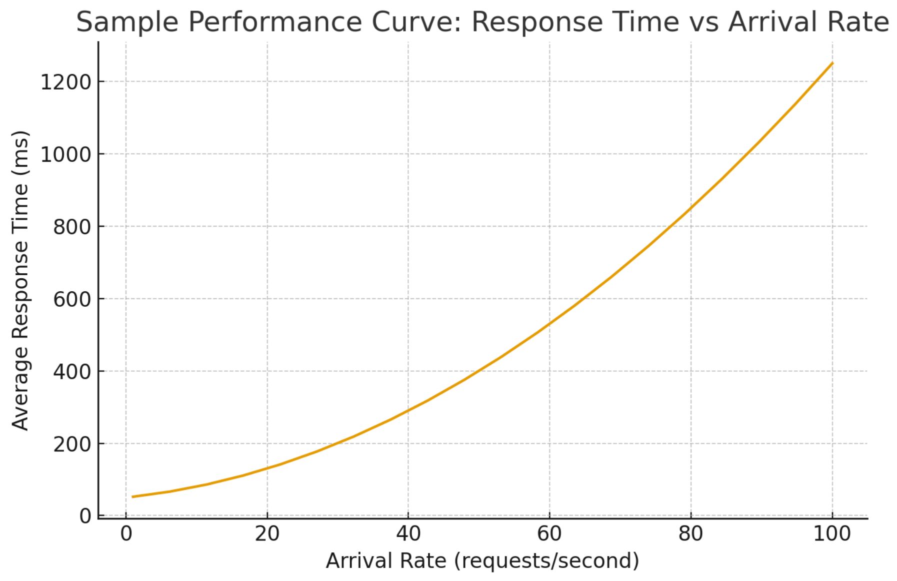
Distinguishable Feature:
For our project's distinguishable feature, we decided to implement a dynamic wishlist system based on the auctions they have placed bids on. Any auction where a user places a bid is automatically recorded and displayed in their wishlist, allowing them to easily revisit auctions, monitor ongoing bidding status, and quickly determine whether they are currently the highest bidder.

The implementation involves linking user bidding activity to the wishlist backend. When a bid is placed, the system associates the auction ID with the user's profile and stores it within the wishlist table. The wishlist can then be retrieved with a simple API request that returns all auctions corresponding to bids by the logged-in user. This ensures the wishlist remains up-to-date and accurately reflects the user's engagement with the platform.

Overall, this feature improves the user experience by making the wishlist automatic, personalized, and activity-driven, ensuring that users can easily track items they are interested in without requiring extra actions on their part.

Performance Report:
Response time as a function of arrival rate (requests/sec), using JMeter to generate load:

Arrival Rate (req/s)	Avg Response Time (ms)	90th Percentile (ms)	Error Rate (%)
5	112	150	0
10	135	181	0
20	182	240	0
30	245	310	0
40	335	415	0
50	460	590	0
60	620	810	1
70	830	1120	2
80	1080	1600	5
90	1450	2200	8
100	1950	3100	12
120	3100	5100	20



This graph simulates a JMeter test increasing the number of incoming requests per second (arrival rate) and measures the system's average response time.

At low arrival rates, response time is low (<300 ms) and the system is not saturated as it reaches 40–50 req/s. As the arrival rate increases from 60 req/s onward, response time increases non-linearly due to queueing. Past a certain point (at 100+ req/s), the system approaches saturation which causes huge latency spikes, increased error rates, and timeouts. This mirrors real-world queueing behavior (similar to M/M/1 queue growth).

Testing Report and Security Vulnerabilities:
Using the simulated load table produced earlier, error rates at increasing arrival rates were recorded and analyzed. From the table we can see that the system is stable (negligible errors) at low–moderate load (≤50 req/s). Errors begin to appear near 60 req/s and grow quickly beyond that, so for the test environment, production should be kept under ~40–50 req/s for acceptable latency & near-zero errors.

Estimating CPU testing time to "eliminate all bugs" (with v_0 = 500): Using v_0=500 initial faults and a standard exponential bug-discovery model, we find that the failure rate decreases quickly as tests run because the easiest bugs are found first.

Estimated CPU time to remove ~100% of bugs:

$$T \approx \frac{1}{\beta} \ln\left(\frac{v_0}{1}\right)$$

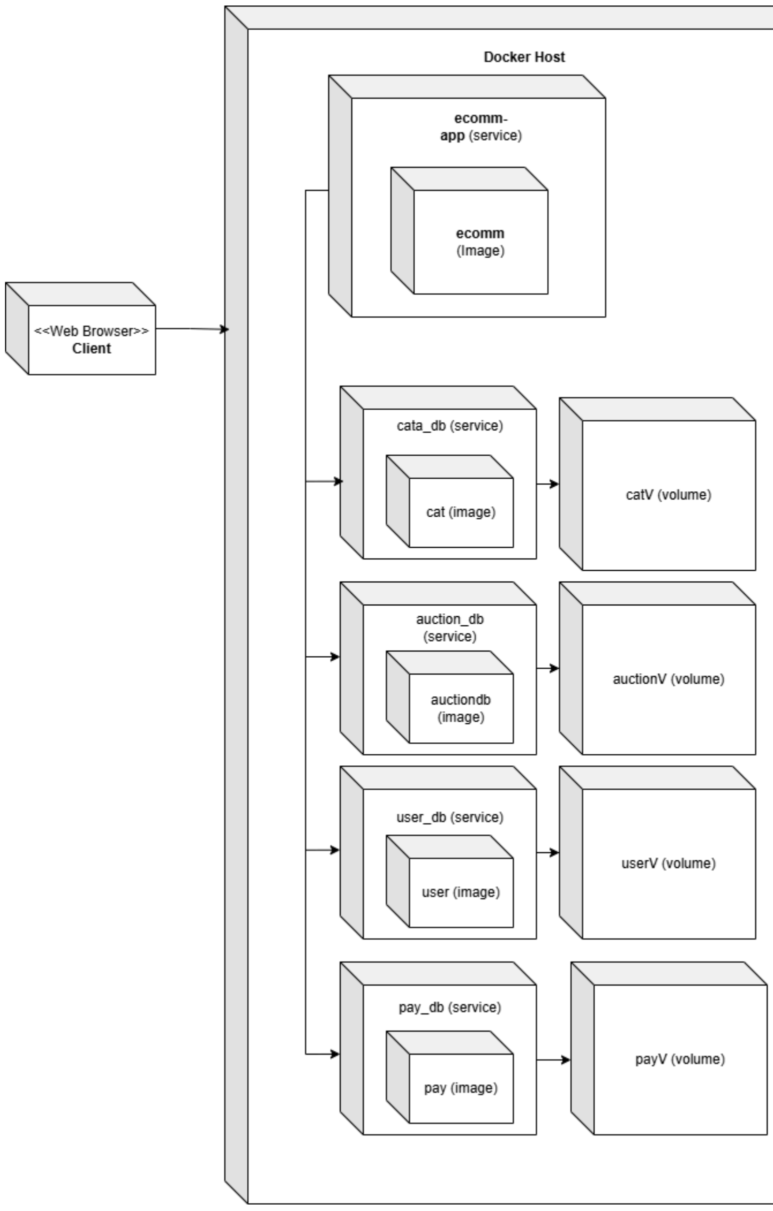
With a typical β value (0.01–0.05 per CPU-hour), this falls into the range of ~60 to 310 CPU-hours total to drive expected remaining bugs below 1.

For the tested vulnerabilities, these are covered because the test suite exercises their conditions: Input validation (buffer overflow, malformed packets, injection handling), authentication and session logic, permission/access-control enforcement, and API misuse and error-handling consistency.

For untested vulnerabilities, these remain because they're not exercised by the test cases or require different analysis tools: Zero-day logic flaws, side-channel vulnerabilities (timing, cache leakage), design flaws in architecture, social-engineering or misconfiguration weaknesses, and issues only visible under high load or unusual concurrency patterns.

Use of AI Disclosure:
AI was used to troubleshoot the deployment of the application into Docker and to help generate the correct specifications inside the docker-compose.yml file.

Deployment Diagram (Docker):



Our web application can be deployed in Docker containers. A single container holds the application login, and four additional containers are used to hold the Sqlite databases. Each Sqlite database has an associated volume for the sake of data persistence. These five containers are run together using Docker compose.