

Constellation Auction House

Design Document

Version:	3.0
Print Date:	2023/12/06
Release Date:	2023/12/06
Approval State:	Approved
Approved by:	Constellation Team
Prepared by:	Constellation Team
Reviewed by:	Constellation Team
File Name:	Deliverable3 EECS4413.pdf
Document No:	3

Version	Date	Authors	Summary of Changes
1.0	2023/10/20	Ali, Alex, Dwumah, Mate	Initial Release
2.0	2023/11/11	Ali, Alex, Dwumah, Mate	Deliverable 2 Requirements
3.0	2023/12/06	Ali, Alex, Dwumah, Mate	Deliverable 3 Requirements



Name	Signature	Date
Mate Korognai		2023/12/06
Alex Arnold		2023/12/06
Ali Sheikhi		2023/12/06
Dwumah Anokye		2023/12/06

Table of Contents

1 Introduction.....	1
1.1 Purpose.....	1
1.2 Overview.....	1
1.3 References - GitHub.....	1
1.5 Deliverable 1 Feedback.....	1
2 Major Design Decisions.....	1
2.1 Design Choices.....	1
2.2 Architectural Patterns Chosen.....	2
2.3 Other Patterns Considered.....	2
3 Sequence Diagrams.....	2
4 Activity Diagrams.....	8
5 Architecture.....	10
6 Activities Plan.....	13
6.1 Project Backlog and Sprint Backlog.....	13
6.2 Group Meeting Logs.....	14
7 Test Driven Development.....	17
8 Deliverable 2 Implementation.....	26
9 Deliverable 2 Installation Guide.....	28
10 Deliverable 3 Additional Use Case: OAuth.....	31
10.1 OAuth Use Case.....	31
10.2 OAuth Implementation.....	31
11 Deliverable 3 Architecture.....	32
12 Deliverable 3 Deployment/Installation.....	33
13 Deliverable 3 Security Vulnerabilities.....	33
14 Deliverable 3 Demo.....	33
15 Deliverable 3 Testing.....	33

1 Introduction

1.1 Purpose

This document details the requirements of the system Constellation Auction House. It explores many aspects of the project, providing both technical and practical explanations of its components. The guidelines for this system were taken from the project description as posted on eClass for the course LE/EECS4413 under 'Project Specification'.

1.2 Overview

The report will start by explaining the key design choices made in accordance with the project specifications. Subsequently, it will present sequence and activity diagrams corresponding to user cases one through seven. Following that, a comprehensive list of architectural components will be provided in the form of tables and a component diagram. The subsequent section will summarise the collaborative efforts undertaken by the group during the document's creation, followed by a set of sixteen test cases aimed at verifying the system's performance.

1.3 References - GitHub

There will be heavy use of github links in this document. They will be the references to the descriptions that some of the following sections will use. This is to retain the quality of diagrams and test cases as well as to shorten the length of the document. The github page now contains the actual files for the project as well.

→ [4413 Constellation Team](#)

1.5 Deliverable 1 Feedback

This section is dedicated to discussing Deliverable 1 Feedback review. First and foremost, the feedback stating "Diagrams are not included in the text." is unfortunately unfixable at this time. Due to the length and size of each image, the report would be unnecessarily long and the images would be of low quality due to the limited page size. So to spare the reader from both of those factors they will remain a link. However, two pieces of feedback that are closely related to this "Diagrams are not discussed in the text." and "The only included diagram has no caption." have been fixed by adding a more in-depth explanation as to what the diagrams mean and how they are represented in sections 3, 4 and 5. Added a references page, mainly talking about github as per the feedback "No references are listed.". The feedback stating " All use cases are displayed in a combined mode in a single activity diagram." will not be fixed for deliverable 2. It would be possible however the team's attention for this deliverable has been focused on coding a functional minimal browsing client.

2 Major Design Decisions

2.1 Design Choices

The major design decisions for the e-commerce app encompass modularity, a three-tier architecture, database separation, and the inclusion of a Controller Module, with an additional focus on the Model-View-Controller (MVC) architectural pattern specifically for the client/user interface side. These choices have been driven by a commitment to achieving high cohesion and low coupling, core criteria for modularization.

The application has been thoughtfully organised into separate modules/packages, each assigned distinct responsibilities. This modular approach enhances the app's maintainability and offers scalability options as the system evolves. High cohesion is achieved by ensuring that each module has a well-defined responsibility. For example, the CatalogueService focuses solely on item management, while the BidService handles bids. This high cohesion simplifies maintenance and minimises unintended side effects.

2.2 Architectural Patterns Chosen

The architectural pattern chosen for the client/user interface side of the application is the Model-View-Controller (MVC) pattern. This pattern is employed for rendering views to the front-end user using React. Within the client/user interface, React components act as the View, managing the presentation of data and the user interface. The Controller aspect is represented by the logic within React components that handle user interactions, such as form submissions or button clicks. These interactions trigger requests to the back-end services following the three-tier architecture.

On the back-end side, the application continues to follow the principles of the Microservices Architecture with separate services for Catalogue, Auction, Payment, and User management. Each of these services can internally organise its components and logic as needed, and the loose coupling between them allows for independent development and scaling.

2.3 Other Patterns Considered

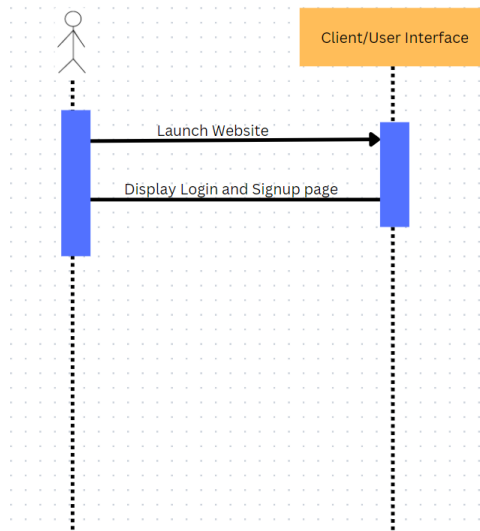
One architectural pattern considered but not chosen is Monolithic Architecture. In a Monolithic Architecture, the entire application is developed as a single, interconnected unit. However, this pattern was not chosen as it doesn't align with the project's requirements for scalability and maintainability, especially given the complexity of an e-commerce system. The modular and microservices-based approach better addresses these needs while the MVC pattern enhances the organisation and manageability of the user interface.

3 Sequence Diagrams

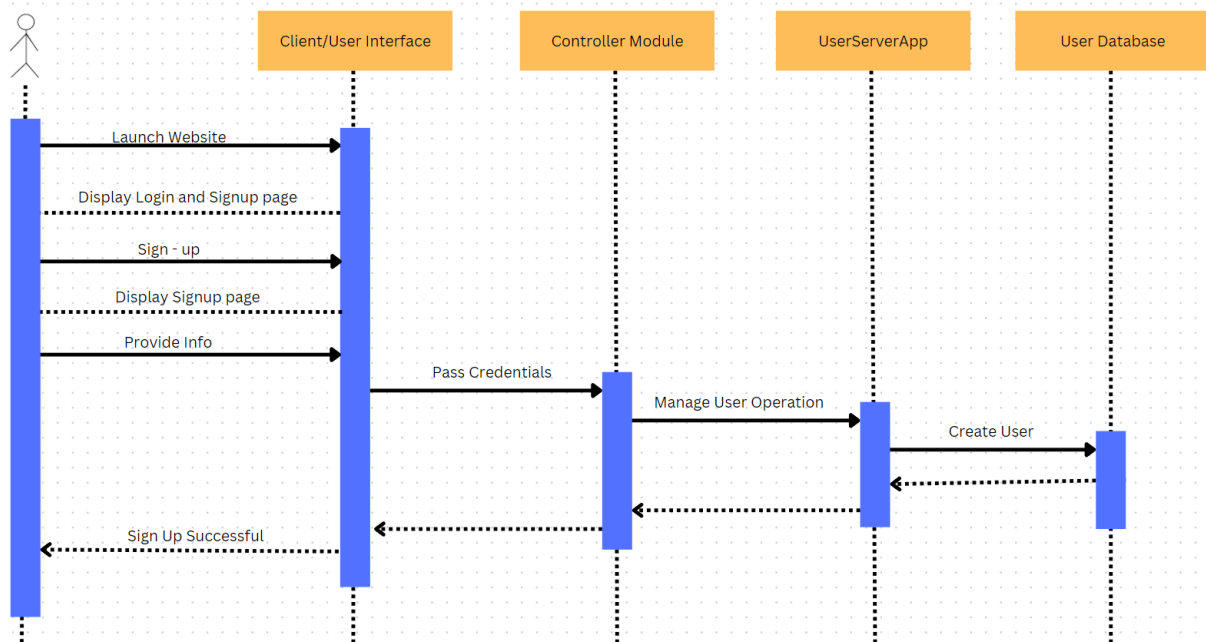
The Sequence diagrams are listed in the hyperlink below in case the images lost quality on the document. There is a sequence diagram for each user case listed in the project description. They go into depth on how the system handles the internal exchanges of information between modules and how they will be controlled to showcase the output to the user. The user should only interact with the front end and every other transfer of data should be hidden from the user between the model and the controller.

→ [Sequence Diagrams](#)

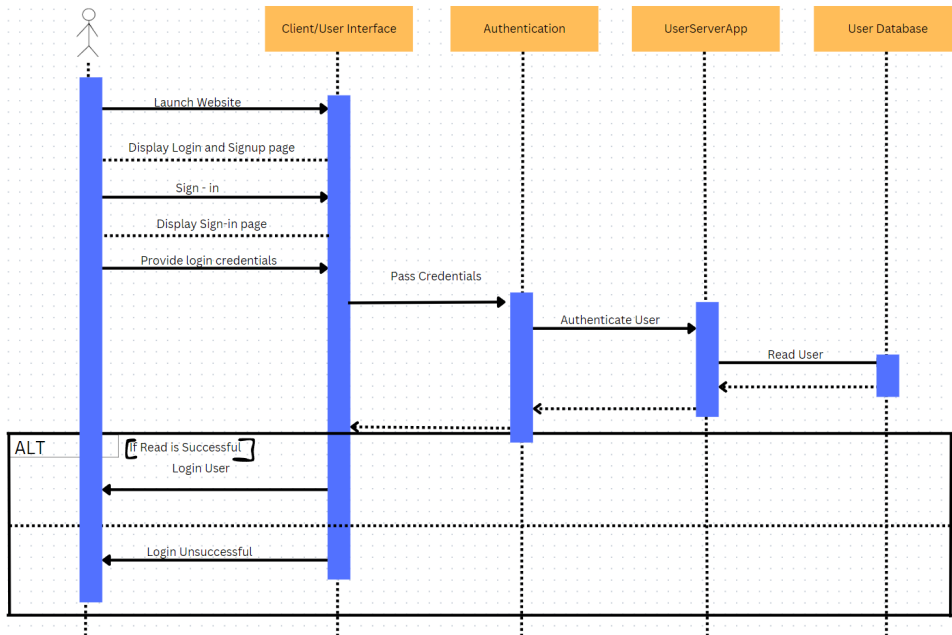
Use Case 1: Sign-in and Sign-up



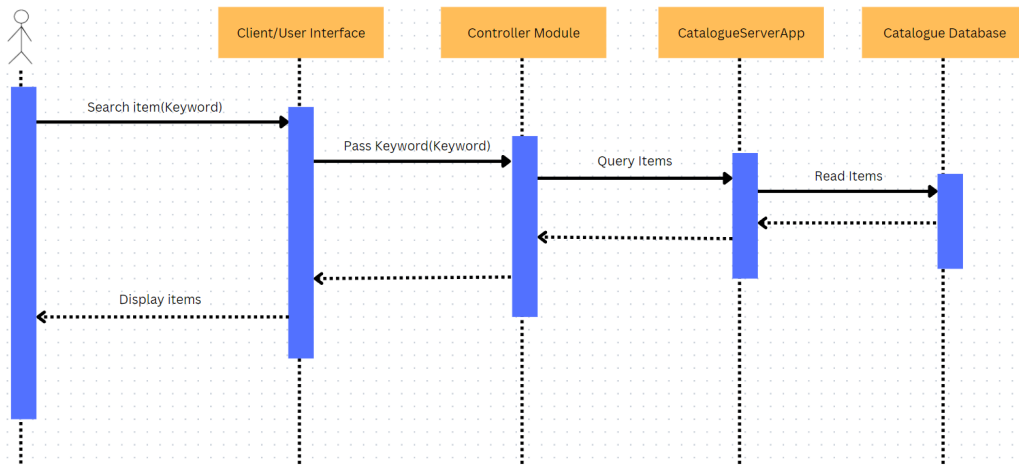
Use Case 1.1: Sign-up



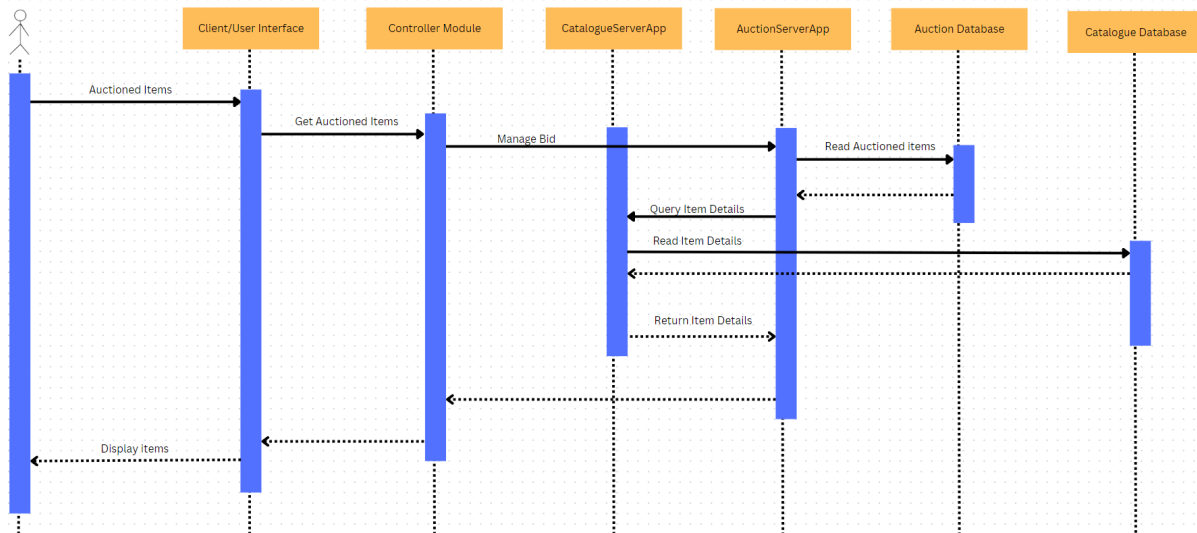
Use Case 1.2: Sign-In



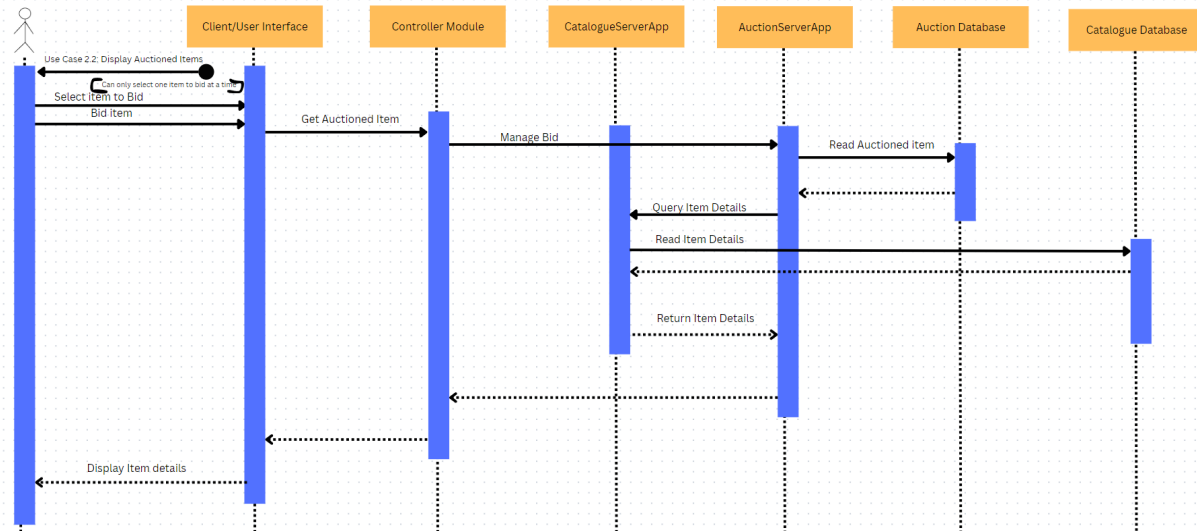
Use Case 2.1: Item Search



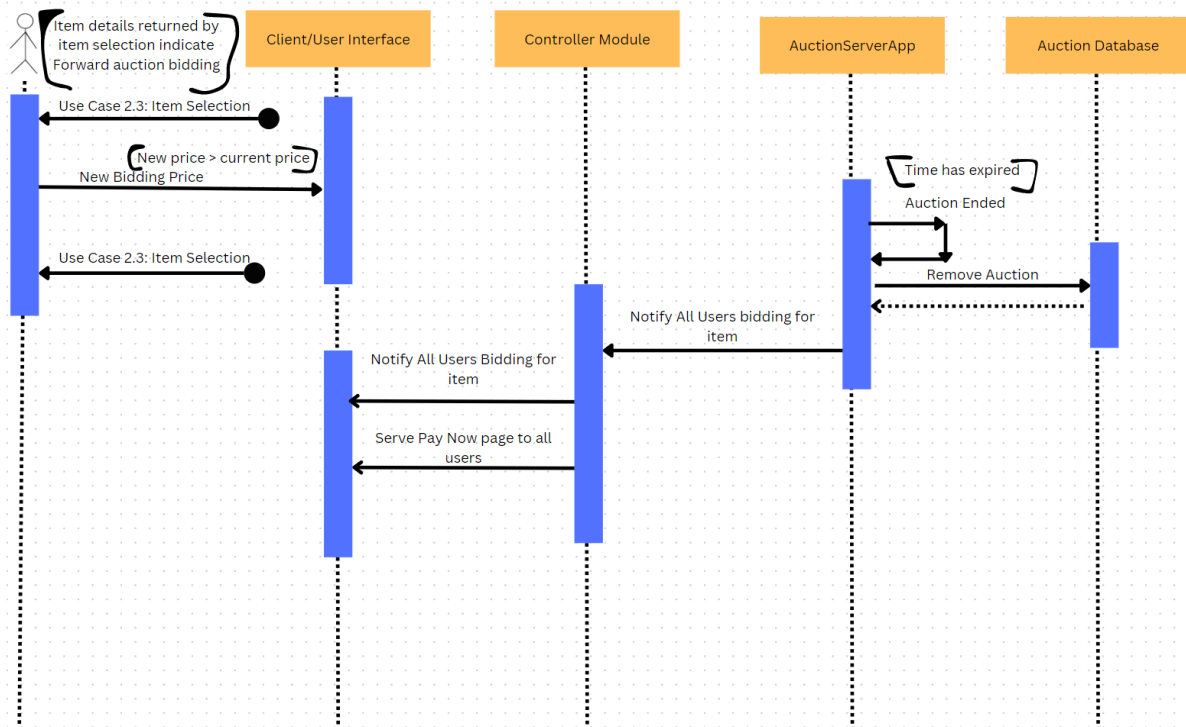
Use Case 2.2: Display Auctioned items



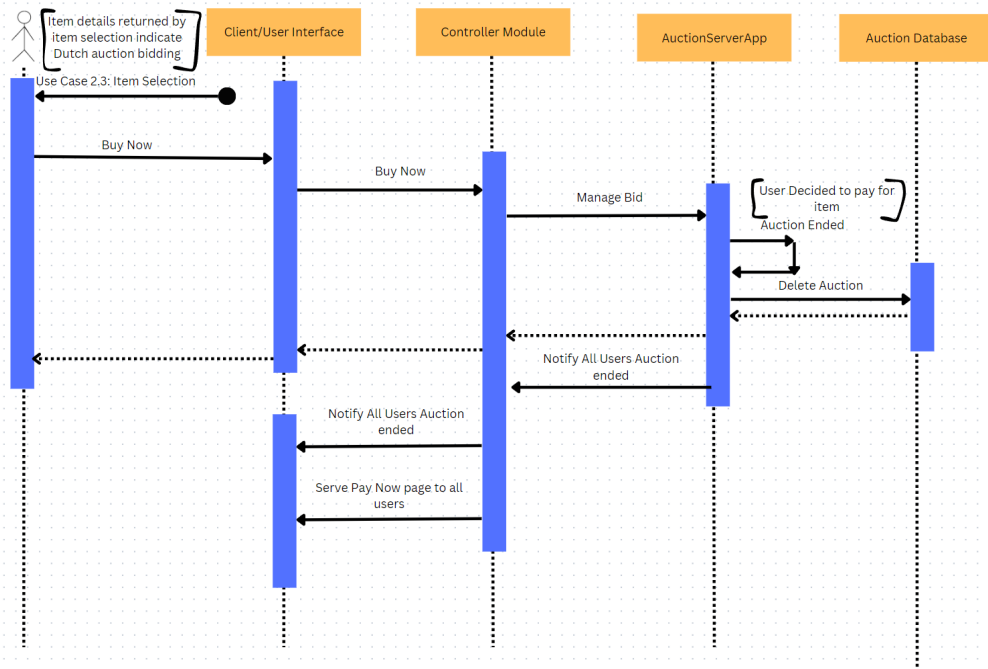
Use Case 2.3: Item Selection



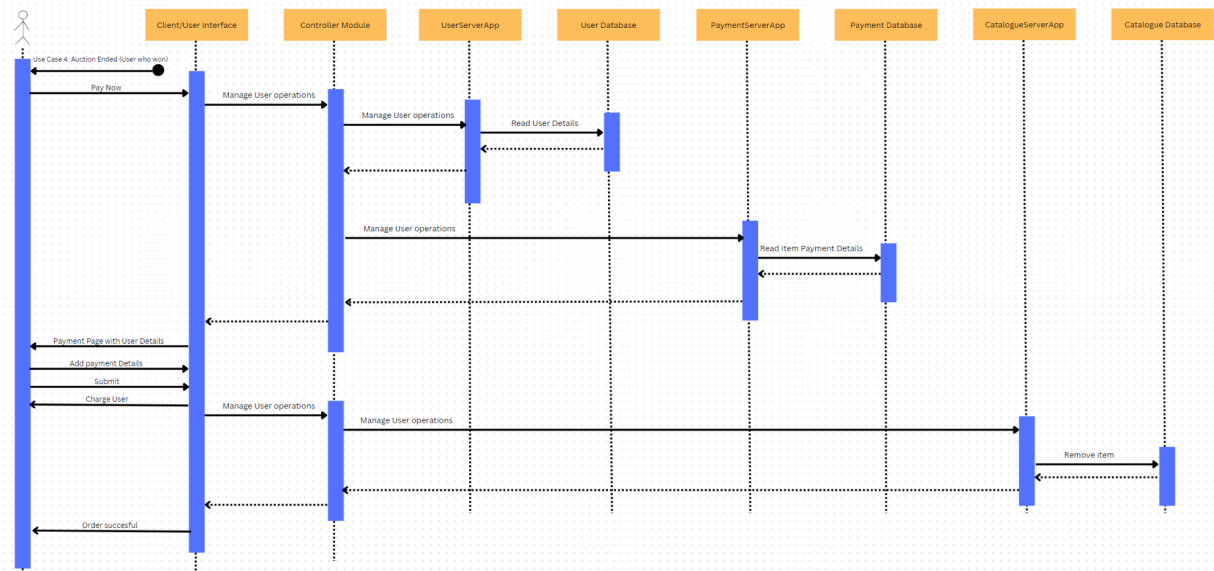
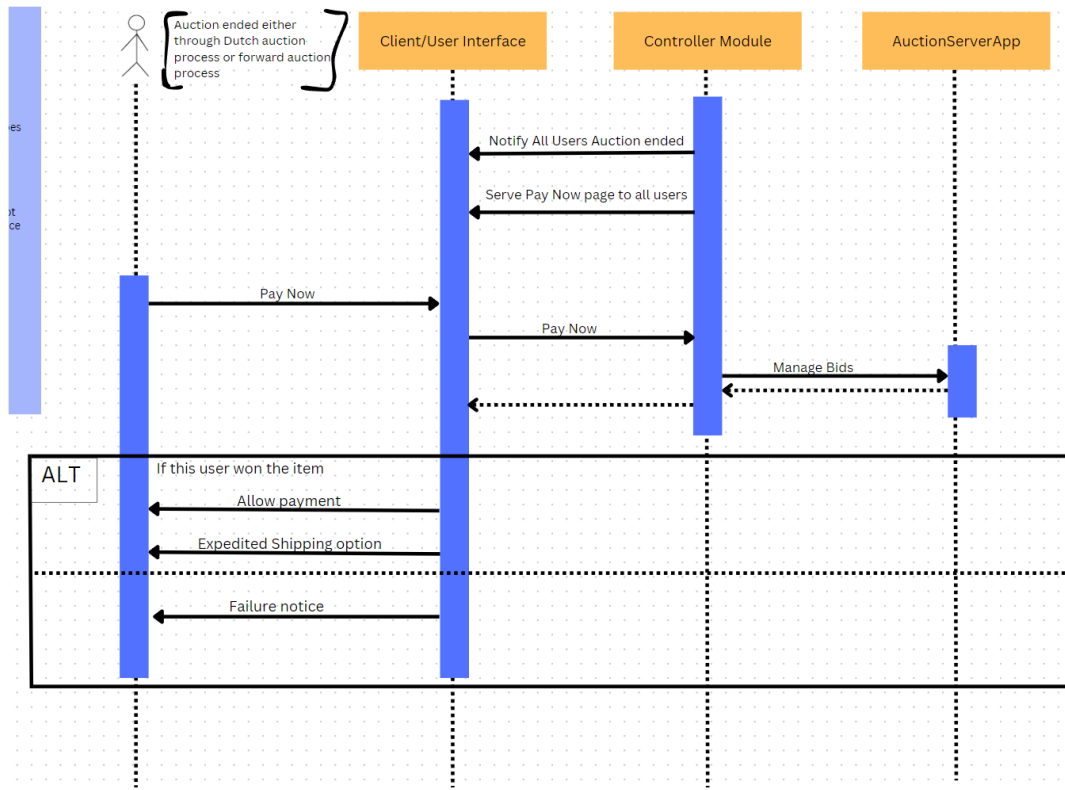
Use Case 3.1: Forward Auction Bidding

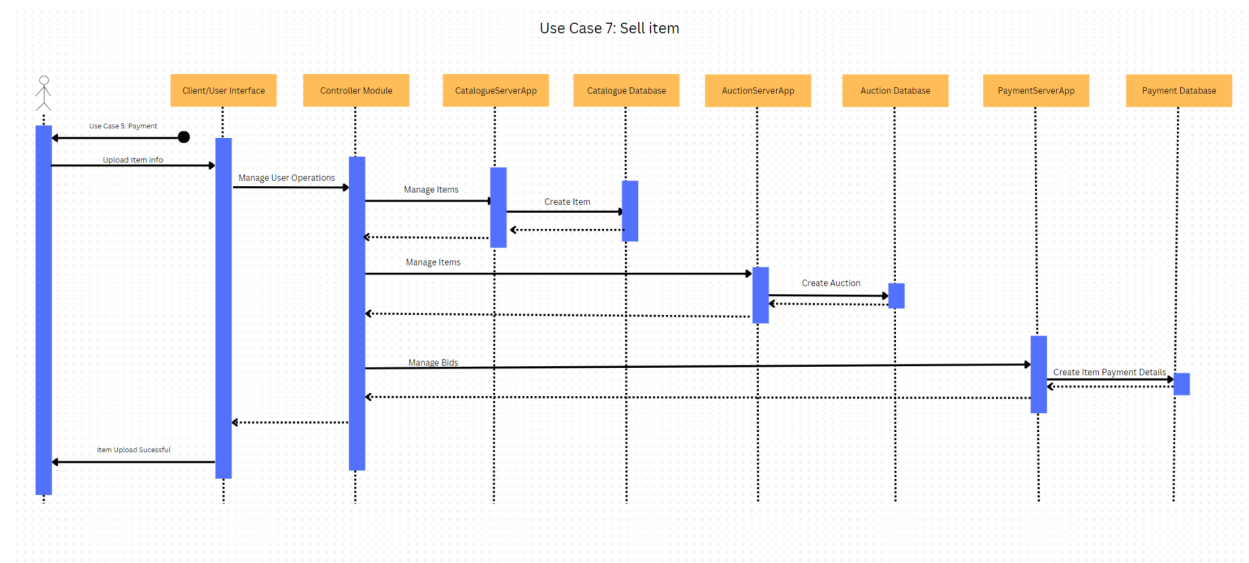
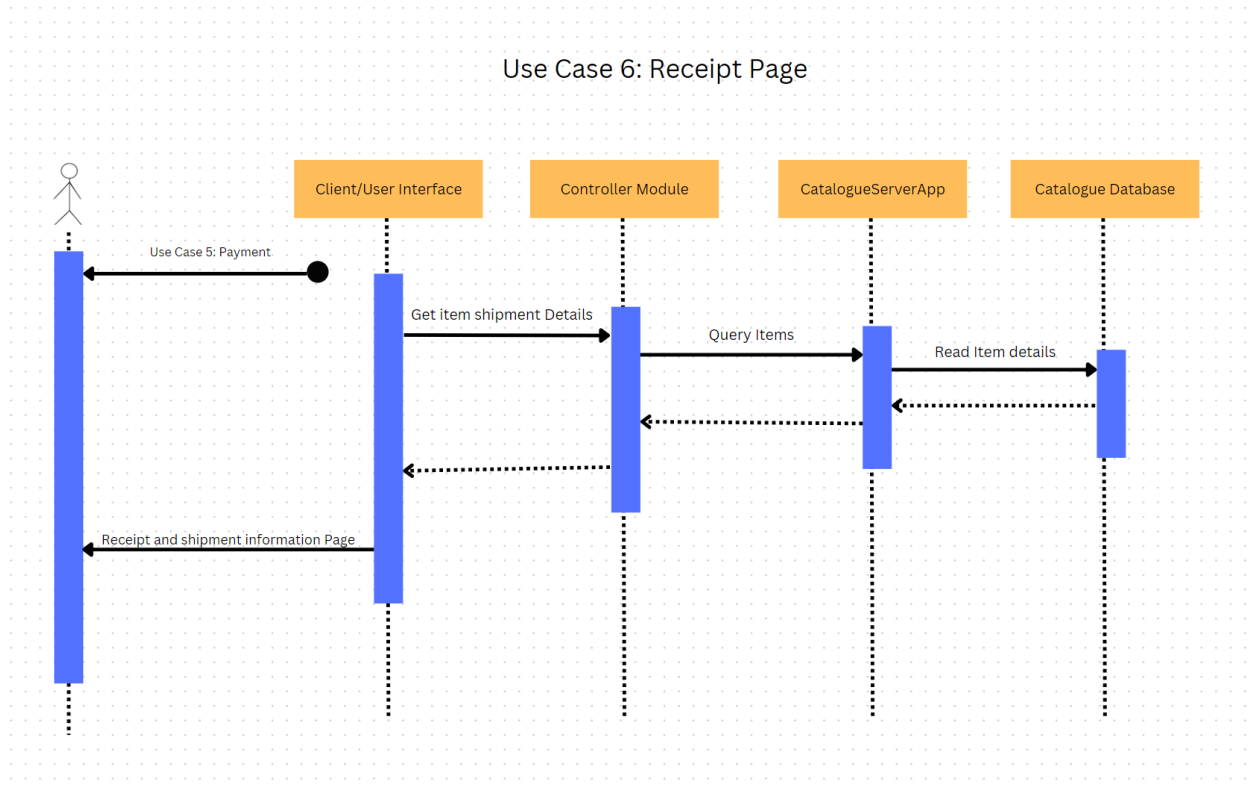


Use Case 3.2: Dutch Auction Bidding



Use Case 4: Auction Ended

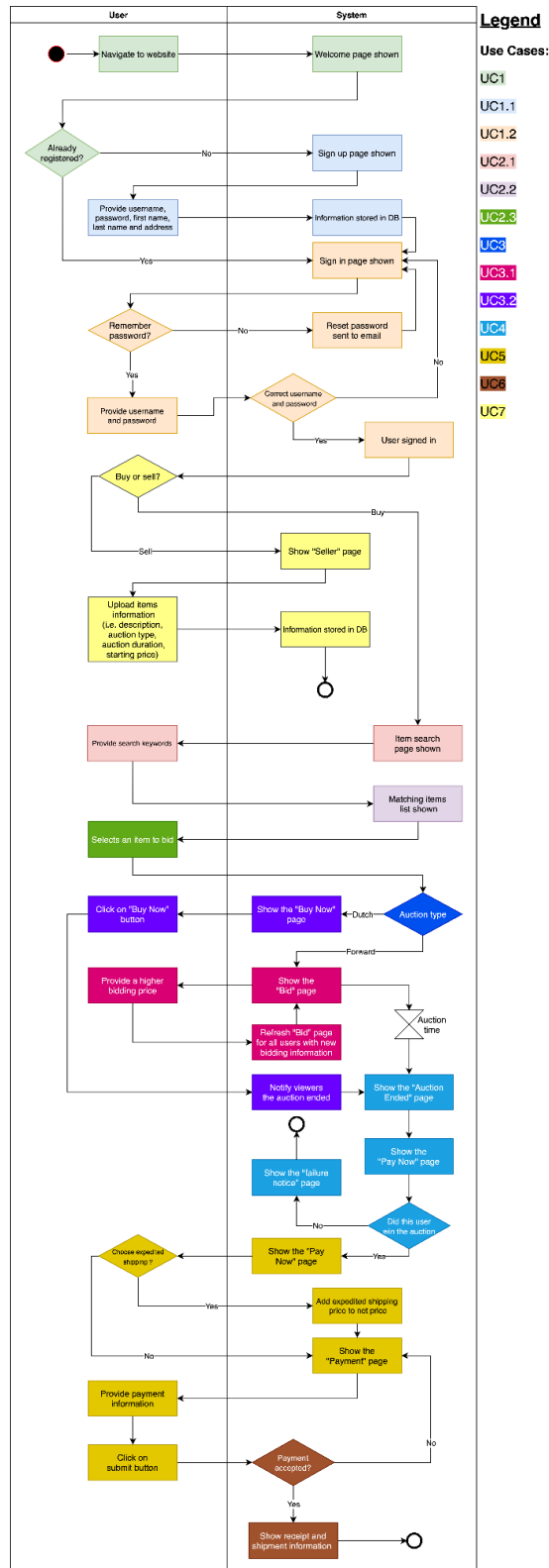




4 Activity Diagrams

There is a single activity diagram that includes all use cases. There is a legend indicating colour sections of the activity diagram corresponding to user cases.

→ [Activity Diagram](#)



5 Architecture

Below is the architecture chosen for the execution of this project. It has a front-end component that is responsible for the user interface through a controller and a bunch of views that are most likely going to be in the form of HTML files. The authenticator is used to verify user information and the controller will handle all data transfers between the front end and the back end. There is a database containing fields for users, items, bids as well and payments which will be interacted with using the services described in the backend, that will be feeding the controller.

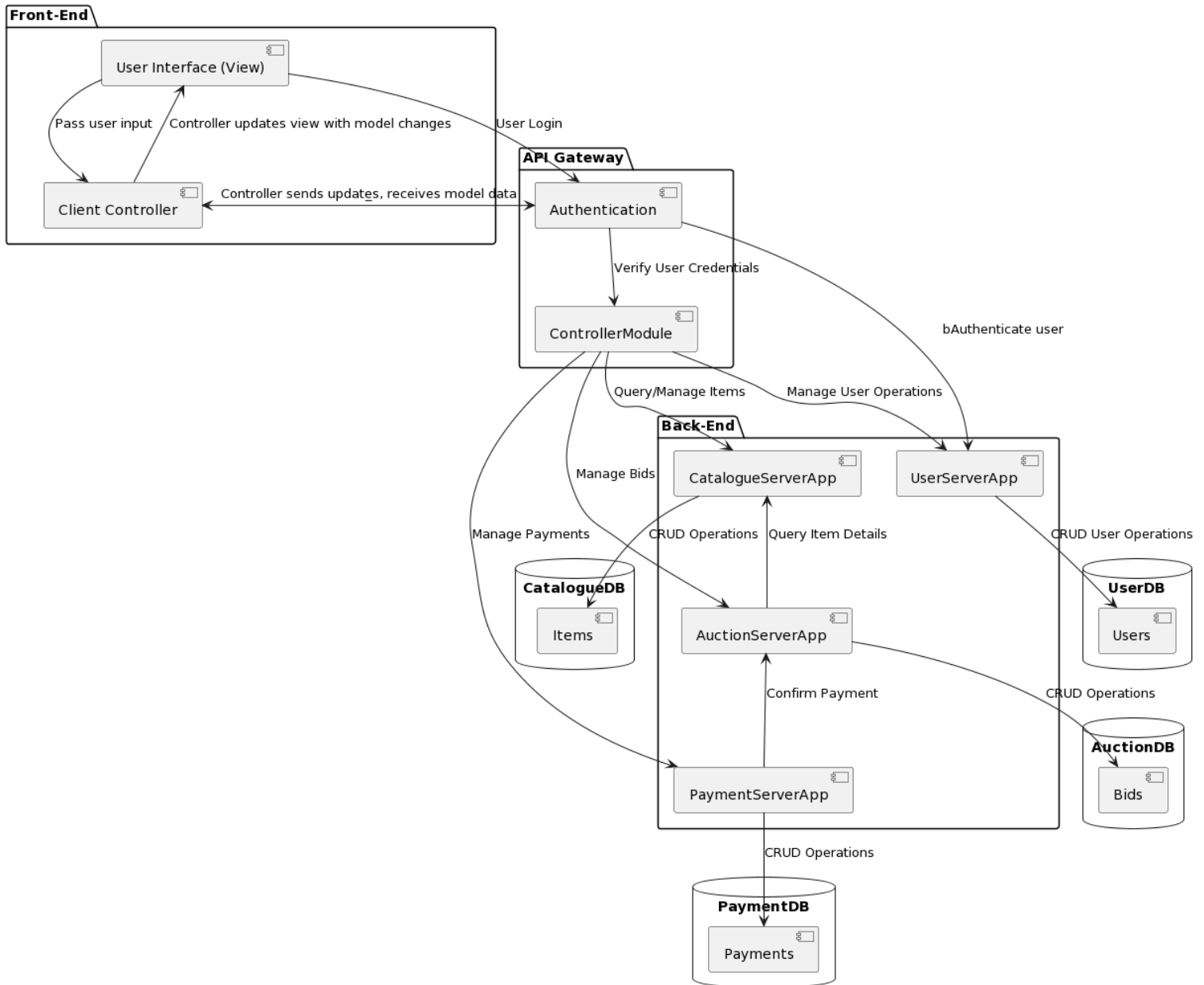


Figure 1: Model Diagram of preliminary design

Below is a table of all modules. Each module is corresponding to a box in the above architecture diagram. They each have a short description that explains the purpose of the modules. There is a list of exposed interfaces that will be the main modes of interaction between modules. The interfaces also all have a short description.

Modules			
Module Name	Description	Exposed Interface Names	Interface Description
Front-end	Manages user interactions/input, updates interface accordingly	UserInterface ClientController	UserInterface - Manage user input ClientController - Update interface
API Gateway	Coordinates communication and authentication between front-end and back-end	Authentication ControllerModule	Authentication - Authenticates and authorises requests ControllerModule - Acts as a facade to control interactions with back-end
Back-end	Manages business logic, process data, interact with database	CatalogueServerApp AuctionServerApp PaymentServerApp UserServerApp	Each app controls the business logic and crud interfacing with the database layer
Databases	Store and manage all of the data pertaining to the auction system	CatalogueDB AuctionDB PaymentDB UserDB	Each separate database stores data for only 1 purpose as is typical for a microservices architecture

Table 1: Modules of the chosen design

The table below has all the interfaces named in the previous table. Each interface goes more in-depth with its operations, indicating the purpose of the operations through a short description. This table will dictate how the modules will interact with their exposed interfaces and through what operations.

Interfaces		
Interface Name	Operations	Operation Descriptions
User Interface	Display, UserInput	Display auction items/statuses, receive and send user events
Client Controller	UpdateView, GetUserInput	Update the ui view, receive inputs to forward to gateway
Controller Module	ManageRequests	Routes requests back to the appropriate service/database
Authentication	VerifyUserCredentials, UserLogin	Verify that user is still logged in and has access to the requested resource; handle user login process and return auth token to client
CatalogueServerApp	CatalogueCreate, CatalogueRead, CatalogueUpdate, CatalogueDelete	Allows for adding new items to the site, viewing available items in the catalogue, updating properties of those items, and deleting them when needed
AuctionServerApp	BidCreate, BidRead, BidUpdate, BidDelete	Manage operations related to auction bids
PaymentServerApp	PaymentCreate, PaymentRead	Create payment records and store them in a database. Also view payment records.
UserServerApp	UserCreate, UserRead, UserUpdate, UserDelete	Manage user profiles

Table 2: Interfaces of the chosen design

6 Activities Plan

6.1 Project Backlog and Sprint Backlog

Scrum Product Backlog To-Do List for Deliverable 1

- Meet The Team
- Major Design Decisions
- Introduction
- Sequence Diagrams
- Test Cases 1-16
- Activity Diagrams
- Individual Designs
- Architecture
- Gantt Chart
- Submission

Gantt Chart

Constellation Auction House

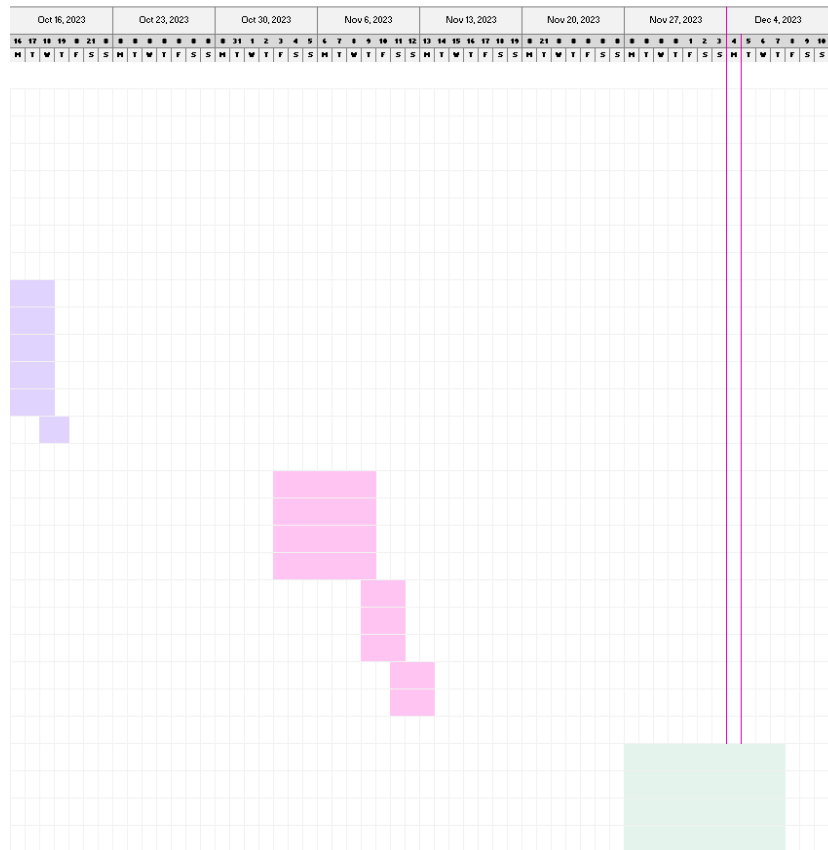
LE/EECS4413 Building E-Commerce Systems

Project start: Sun, 10-1-2023

Display week: 3

SIMPLE GANTT CHART by Vertov42.com
https://www.vertov42.com/ExcelTemplates/simple-gantt-chart.html

TASK	ASSIGNED TO	PROGRESS	START	END
Deliverable 1				
Meet The Team	Team	100%	10-1-23	10-2-23
Introduction	Team	100%	10-3-23	10-5-23
Test Cases 1-4	Mate	100%	10-5-23	10-9-23
Test Cases 5-8	Ali	100%	10-5-23	10-9-23
Test Cases 9-12	Alex	100%	10-5-23	10-9-23
Test Cases 13-16	Diumah	100%	10-5-23	10-9-23
Individual Designs	Team	100%	10-9-23	10-14-23
Gantt Chart	Mate	100%	10-14-23	10-18-23
Major Design Decisions	Alex	100%	10-14-23	10-18-23
Sequence Diagrams	Diumah	100%	10-14-23	10-18-23
Activity Diagrams	Ali	100%	10-14-23	10-18-23
Architecture	Mate & Alex	100%	10-14-23	10-18-23
Submission	Team	100%	10-18-23	10-19-23
Deliverable 2				
PaymentServerApp	Ali	100%	11-3-23	11-9-23
AuctionServerApp	Diumah	100%	11-3-23	11-9-23
ItemServerApp	Alex	100%	11-3-23	11-9-23
UserServerApp	Mate	100%	11-3-23	11-9-23
Review Deliv. 1 Feedback	Mate	100%	11-9-23	11-11-23
HTML Pages	Everyone	100%	11-9-23	11-11-23
Controller	Alex	100%	11-9-23	11-11-23
Documentation	Mate	100%	11-11-23	11-13-23
Connecting all Modules	Alex, Diumah, Ali	100%	11-11-23	11-13-23
Deliverable 3				
Separate Services	Mate	100%	11-27-23	12-7-23
Connect back to front	Alex	100%	11-27-23	12-7-23
Testing	Ali & Diumah	100%	11-27-23	12-7-23
Documentation	Everyone	100%	11-27-23	12-7-23



6.2 Group Meeting Logs

2023/10/05	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Got together for the first time and began discussing the project what architecture style to approach this problem with.</p> <p>Talked about how to begin working on the project. Things like what languages to use, Talked about the software tools we will use for the design phase. Talked about using MVC for the primary architecture of the project.</p> <p>Decided to work on Test Driven development, each UC has a person that works on those tests:</p> <ul style="list-style-type: none"> → Dwumah - Use Cases 6 and 7 → Ali - Use Cases 2 and 3 → Alex - Use Cases 4 and 5 → Mate - Use Case 1 <p>Next Meeting 2023/10/09 6:00pm</p>
2023/10/09	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Looked over the test cases</p> <p>Discussed Architecture that we will use</p> <p>Goal for next meeting: Everyone should create a design which they feel fits the project description. Next meeting will be devoted to looking at all the suggested designs and then mashing them all into one solid design.</p> <p>Next Meeting 2023/10/14 6:00pm</p>
2023/10/14	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Everyone brought their own design and we discussed the pros and cons of each design. We talked about the advantages of MVC, Three-tiered, client-server and repository architecture and ultimately decided on a combination of MVC and three-tiered design.</p> <p>Tasks were assigned:</p> <ul style="list-style-type: none"> → Dwumah - Sequence Diagrams → Ali - Activity Diagram → Alex - Architecture & Major Design Decisions → Mate - Gantt Chart & Tables for Architecture <p>Next Meeting 2023/10/18 8:30pm</p>
2023/10/18	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Discuss the tasks that need to be done before submission.</p> <p>List of things to do:</p> <ul style="list-style-type: none"> → Put all diagrams onto doc → Create github and put the Test Cases PDF in there and put the link in the doc (Alex) → Update table of contents (Mate) → Finish modules/interfaces table (Alex/Mate) → Update diagram to properly show use of MVC pattern (Alex) → Final Formatting (Mate)
2023/11/03	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Began working on Deliverable 2</p> <p>Discussed how to split up the work</p> <p>Allocated Tasks:</p>

	<ul style="list-style-type: none"> → Dwumah - AuctionServerApp, Auction DB, related curl/postman commands → Ali - PaymentServerApp & PaymentDB and other thing needed for payment → Alex - Itemservice → Mate - UserServerApp & UserDB & Documentation (Login, Register, Bidding) <p>Next Meeting 2023/11/9 9:00pm</p>
2023/11/9	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Reviewed everyone's submitted code. Had trouble with github merging, debugged as a group and then we talked about the code that everyone made. Everyone described their design choices and showcased how their code worked in comparison to the requirements</p> <p>Allocated Tasks:</p> <p>Everyone must make a simple html page for the usecases that relate to their code from the previous meeting</p> <ul style="list-style-type: none"> → Dwumah - Dutch auction, regular auction, auction ended → Ali - Payment page, receipt page → Alex - Controller, connecting html pages → Mate - landing / login / sign up, review deliverable 1 feedback, update gantt <p>Next Meeting 2023/11/11 8:00pm</p>
2023/11/11	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Alex showcased how he went about connecting frontend and backend modules so that the login and signup pages</p> <p>Everyone finished making their html pages, so now all that remains is to have each user case described in the provided requirements included into the system</p> <p>Tasks were assigned as follows</p> <ul style="list-style-type: none"> → Dwumah - Connecting Dutch auction, regular auction, auction ended from end to front → Ali - Connecting Payment page, receipt page, from end to front → Alex - Connecting Auction List Page, creating main page connection, home page now auction list → Mate - Documentation and UML diagram design. looking over requirements <p>Next Meeting 2023/11/12 8:00pm</p>
2023/11/12	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Issues: Login not needed to access everything else</p> <p>Showcased existing pages and how they are connected. Done the catalogue list page, as well as the login, signup and main menu</p> <p>Tasks were assigned as follows</p> <ul style="list-style-type: none"> → Dwumah - Connecting Dutch auction, regular auction, auction ended from end to front → Ali - Connecting Payment page, receipt page, from end to front → Alex - Connecting Auction List Page, creating main page connection, home page now auction list → Mate - Documentation and UML diagram design. looking over requirements, postman test cases for signup, login, catalog, bid id <p>Next Meeting 2023/11/13 8:00pm</p>
2023/11/13	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Went over project details and looked at the product, submitted github link</p>

2023/11/27	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>Split up tasks</p> <p>Began to organize existing classes into each service</p> <p>Tasks were assigned as follows:</p> <ul style="list-style-type: none"> → Dwumah - Writing Postman and curl test cases → Ali - Writing Postman and curl test cases → Alex - containerizing, connecting front to back & Documentation → Mate - Separating microservices & Documentation <p>Next Meeting 2023/12/06</p>
2023/12/06	<p>Everyone Attended (Ali, Alex, Dwumah, Mate)</p> <p>One of the last meetings before submission.</p> <p>Discussed how the project is set up and talked about alternative architectural styles that can be used.</p> <p>The existing structure is an MVC + Repository</p> <p>Spring and Google Cloud were used for auto deployment</p> <p>Made an architecture diagram</p>

7 Test Driven Development

The test cases have been moved to github to shorten document length. There is a test case for each user interface each with their unique id representing their purpose. The Github page can be accessed using the following hyperlink:

→ [Test Cases](#)

Test ID	01
Category	Evaluation of system response upon sign up of an already existing user.
Requirements Coverage	UC1.1-Unsuccessful-Signup
Initial Condition	The System has been initiated and a user with username 'user1' already existing within the system.
Procedure	<ol style="list-style-type: none">1. User starts system2. User clicks sign-up3. In the sign up menu, the user inputs the already existing 'user1' username into the username field4. The user inputs a password, can be anything that is accepted by the system5. The user clicks the sign-up button6. The user is unable to create a new user with username 'user1'
Expected Outcome	The user should be blocked from creating a user with username 'user1', and should be prompted to use another username. They should be back at the signup menu after the prompt.
Notes	The name user1 is arbitrary, this test requires that an already existing username is inputted into the username field of the sign-up menu

Test ID	02
Category	Evaluation of system response to a successful sign up from a user
Requirements Coverage	UC1.1-Successful-Signup
Initial Condition	The System has been initiated, some users can exist, but no username 'newUser' should be in the system
Procedure	<ol style="list-style-type: none"> 1. User starts system 2. Users clicks sign-up 3. User inputs 'newUser' as their username 4. User inputs a password of their choice (arbitrary as long as its remembered) 5. The user clicks the sign-up button 6. The user has successfully created a new user and is automatically logged in
Expected Outcome	A new user with the name 'newUser' is registered to the System with the inputted password. The should then be logged into the 'newUser' account
Notes	The username chosen can be anything as long as it does not already exist within the list of already existing usernames

Test ID	03
Category	Evaluation of incorrect password response during login procedures
Requirements Coverage	UC1.2-Unsuccessful-Login
Initial Condition	The System has been initiated and there exist a user with username 'user1' and password '111'
Procedure	<ol style="list-style-type: none"> 1. User starts system 2. Users clicks login 3. User inputs 'user1' as their username 4. User inputs '222' as their password 5. User clicks the login button 6. The user is told that the password does not work for the username
Expected Outcome	The user should be told that either the username or password are wrong. They should be put back to the login screen

Test ID	04
Category	Evaluation of correct username and password during login procedures
Requirements Coverage	UC1.2-Successful-Login
Initial Condition	The System has been initiated and there exist a user with username 'user1' and password '111'
Procedure	<ol style="list-style-type: none"> 1. User starts system 2. Users clicks login 3. User inputs 'user1' as their username 4. User inputs '111' as their password 5. User clicks the login button 6. The user is at the main menu logged in as 'user1'
Expected Outcome	The user should be told that their login was successful then shortly after they should be taken to the main menu

Test ID	05
Category	UC2: Browse Catalogue of Auctioned Items
Requirements Coverage	UC2.1-Item-Search
Initial Condition	The user should be already logged in.
Procedure	<ol style="list-style-type: none"> 1. The user clicks on the item search text box. 2. The user provides a keyword in the item search text box. 3. The user clicks on the search button.
Expected Outcome	The list of items up for auction or currently being auctioned matching the keywords are displayed.
Notes	If no item matches the keyword "no item found" or similar should be displayed.

Test ID	06
Category	UC2: Browse Catalogue of Auctioned Items
Requirements Coverage	UC2.2-Display-Auctioned-Items
Initial Condition	The logged in user should use the item search function to search for a keyword.
Procedure	<ol style="list-style-type: none"> 1. The search function should be used to search for an item.
Expected Outcome	UI should display a list of items matching the searched keyword with full item name, the current bidding price, the type of auction, and the remaining time should be displayed for each item.
Notes	The remaining time should only be displayed for forward auctions or other appropriate auctions that have a remaining time.

Test ID	07
Category	UC2: Browse Catalogue of Auctioned Items
Requirements Coverage	UC2.3-Item-Selection
Initial Condition	The user should be logged in and searched for a keyword of an available item and the list of the items should be displayed.
Procedure	<ol style="list-style-type: none"> 1. Next to each item should be a radio button that the user can select. 2. The user should only be able to choose 1 item with the radio buttons. 3. Once the selection is made, the user can push the BID button.
Expected Outcome	The UI displays the item details and the option to bid.
Notes	<p>A bidder bids for only one item per session.</p> <p>The user may have multiple sessions in parallel in different browsers.</p>

Test ID	08
Category	UC3: Bidding
Requirements Coverage	UC3.1-Forward-Auction-Bidding
Initial Condition	The auction selected is a forward auction.
Procedure	<ol style="list-style-type: none"> 1. The user can provide a new bidding price which must be higher than the current price and press the BID button. 2. Once a bid is submitted, the page is refreshed (by the server). 3. The new highest bidding price and the highest bidder are displayed to all users bidding for this item. 4. The current highest price and the ID of the current highest bidder are always displayed. 5. When the remaining time expires and the auction ends, all users bidding for this item get to a new page indicating that the auction has ended.
Expected Outcome	The highest bidder has the option to pay and is served a page with a “Pay Now” button.

Test ID	09
Category	UC3: Bidding
Requirements Coverage	UC3.2-Dutch-Auction-Bidding
Initial Condition	The auction selected is a Dutch auction.
Procedure	<ol style="list-style-type: none"> 1. The users can select the “Buy Now” button. 2. The auction terminates immediately. 3. All users on this auction’s page get to a new page indicating that the auction has ended.
Expected Outcome	The bidder has the option to pay and is served a page with a “Pay Now” button.

Test ID	10
Category	UC4: Auction Ended
Requirements Coverage	Non-winner cannot view Pay Now
Initial Condition	Auction ends and the user is served with the “Pay Now” button. The user is not a winner of the bid.
Procedure	<ol style="list-style-type: none"> 1. User clicks “Pay now” button.
Expected Outcome	User receives a failure message that their payment didnt go through because they are not the winner of the bid

Test ID	11
Category	UC5. Payment
Requirements Coverage	Successful Payment
Initial Condition	User should be logged in and the pay now button is pressed, and user is winning bidder
Procedure	<ol style="list-style-type: none"> 1. Form opens for user to submit payment details 2. Complete payment details are entered 3. Form is submitted
Expected Outcome	User should be moved on to the next step where they are given a receipt and shipment details

Test ID	12
Category	UC5. Payment
Requirements Coverage	Failed Payment
Initial Condition	User should be logged in and the pay now button is pressed, and user is winning bidder
Procedure	<ol style="list-style-type: none"> 4. Form opens for user to submit payment details 5. Incomplete (1 random field missing) payment details are entered 6. Form is submitted
Expected Outcome	User should receive an indication that their payment did not go through due to missing fields

Test ID	13
Category	UC6: Receipt Page and Shipment Details
Requirements Coverage	UC6-Receipt-Generation-Dutch-Auction
Initial Condition	System has been initiated and runs. A user that has signed up already logs in.
Procedure	<ol style="list-style-type: none"> 1. Dutch auction 2. Auction ended due to purchase 3. Winner decided to pay 4. Payment has been cleared
Expected Outcome	Html page should be generated and shown to the user that has a receipt and address. The receipt should have all the correct information about the user that was indicated in the user sign up
Notes	User should have indicated their personal information when signing up

Test ID	14
Category	Evaluation of Receipt page
Requirements Coverage	UC6-Main-Page-Button-Receipt-Page
Initial Condition	System has been initiated and runs. A user that has signed up already logs in.
Procedure	<ol style="list-style-type: none"> 1. Dutch auction 2. Auction ended due to purchase 3. Winner decided to pay 4. Payment has been cleared 5. Receipt page was generated 6. Winner presses back to home page button
Expected Outcome	The user should be returned back to the home page screen while still being logged in (session still kept).

Test ID	15
Category	Evaluation of sell item button on main page
Requirements Coverage	UC7-Sell-Item-Button
Initial Condition	System has been initiated and runs. A user that has signed up already logs in.
Procedure	<ol style="list-style-type: none"> 1. User clicks a button on the home page to sell an item.
Expected Outcome	An html page should be shown that allows the seller to upload information about the item to sell.

Test ID	16
Category	Evaluation of selling procedure
Requirements Coverage	UC7-Missing-Item-Information-Sell-Item-Procedure
Initial Condition	System has been initiated and runs. A user that has signed up already logs in.
Procedure	<ol style="list-style-type: none"> 1. User clicks a button on the home page to sell an item. 2. An html page pop ups that allows the seller to input information about the item 3. User fills in fields; forward auction, duration time, starting bid price 4. User does not put information about the item 5. User presses the confirm button
Expected Outcome	A prompt should be shown to the user that tells them that item information is missing. The item should not be added to the database

8 Deliverable 2 Implementation

The implementation of the Constellation Auction House system is very similar to the design mentioned in Section 2 Major Design Decisions, and Section 5 Architecture. It also has the interaction between modules as described in Section 3 Sequence Diagrams, Section 4 Activity Diagrams and Section 5 Architecture (specifically the model description tables Table 1 and Table 2). The below UML diagram was made by hand based on the system interactions that are present within the packages.

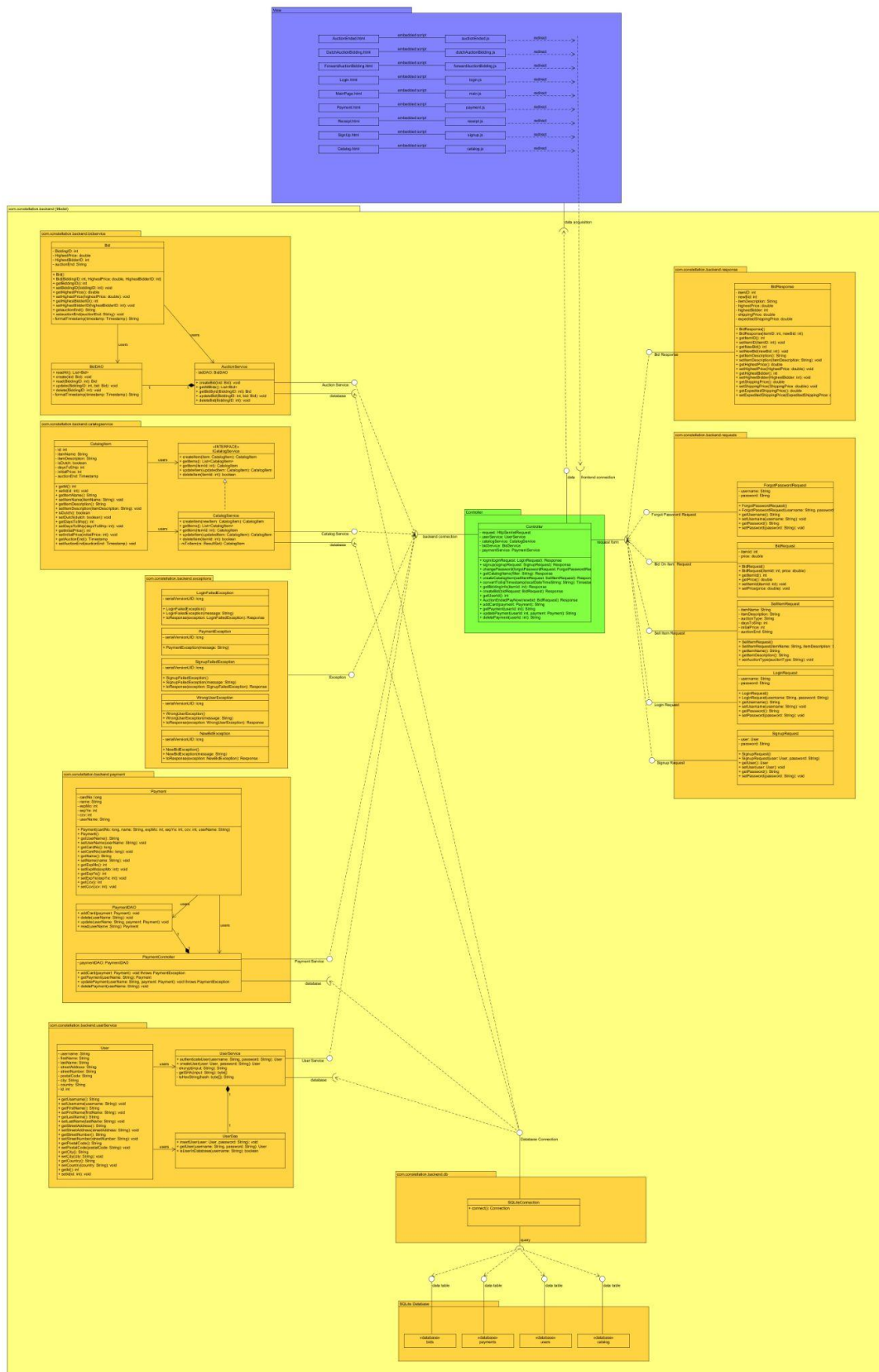
In terms of design decisions, several were made for this project. While not implemented with a `getInstance` method, our service classes all act as singletons in the context of this project, only 1 should ever exist. Once we migrate to a scalable microservices architecture, this will no longer be the case. We have also used data access objects (implementation of the facade/adaptor pattern) to provide interfaces to our database that are independent of the business logic. Lastly, we have used the facade pattern for our controller module which orchestrates all of the different services in one central API.

A major thing to note is that the design is split into three distinct parts, Model, View and Controller. The View is responsible for the user interface, and it mainly consists of HTMLs for each page, as well as an accompanying javascript file that helps in the course of redirecting to new pages from the current page. The Model is responsible for representing the internal state of the system by making abstractions for every data related component of the system. This is also the layer where persistence is maintained through an SQLite database connection located in `com.constellation.backend.db` package under `SQLiteConnection.java`. The Controller is the most vital component of the MVC design (although the other two are just as indispensable). It facilitates the interaction between the View and the Model, allowing them to work together seamlessly. The Controller also does exception handling through exceptions from `com.constellation.backend.exceptions`. We were told to consider a pub/sub system in the last deliverable however considering the added complexity and lack of requirement for scaling we determined this to be a bad choice.

For our design, the workflow of the controller is as described below. The Controller gather input data from respective HTML views through the use of request bodies that are located in `com.constellation.backend.requests`. The controller then performs operations using interfaces given in the Model. For example, when a login request is made from `Login.html > login.js`, it is passed to the controller, and the controller takes the request in the form of a `LoginRequest` object, which it then uses to get the username and password to pass it to the `UserService` provided in `com.constellation.backend.user service`.

If Figure 2 below is of low quality, please use the following link to access the UML diagram representation of the system:

→ [UML Diagram of Constellation Auction House](#)



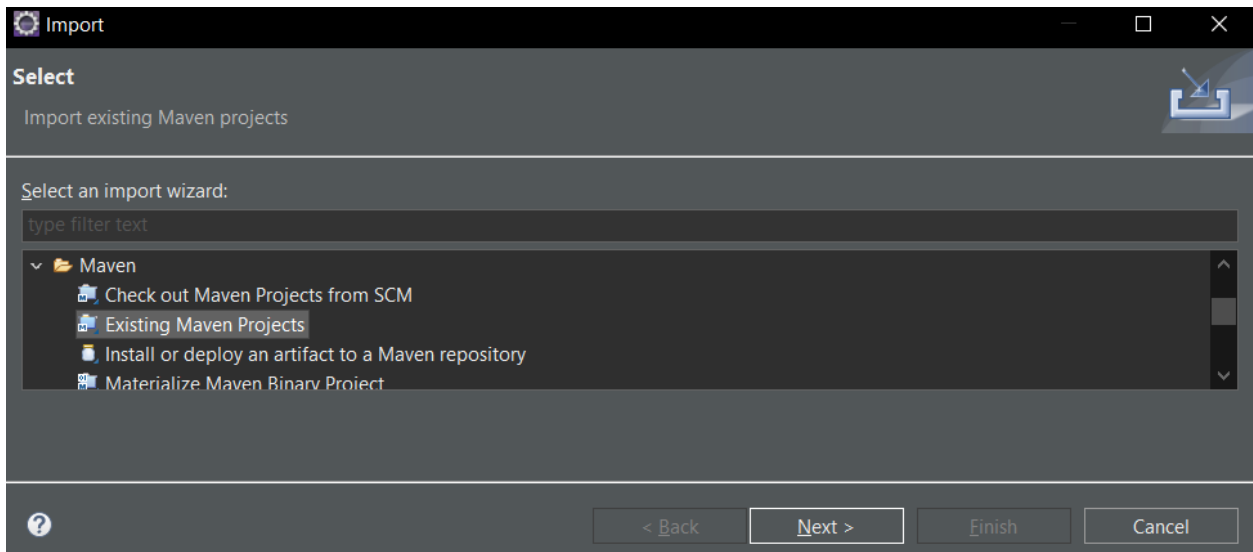
9 Deliverable 2 Installation Guide

The installation process for the system is quite straight-forward, and requires the user to take 4 steps. Screenshots will be included to help guide the user. However, please note that this demonstration uses Eclipse, and hence screenshots will be using Eclipse interface.

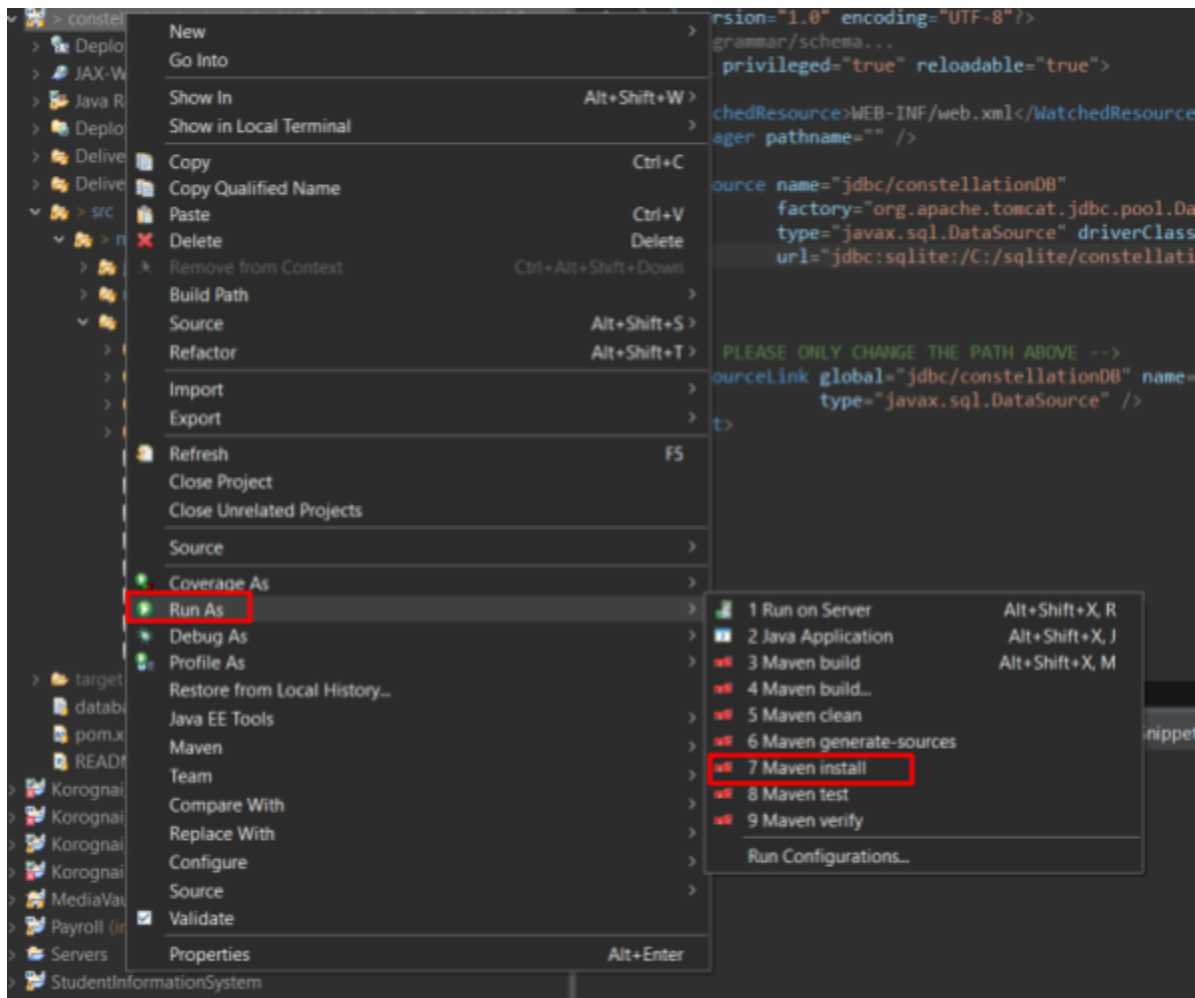
First and foremost, make sure that the system is present on your device by cloning the repository 'git clone <https://github.com/tens-ml/4413ConstellationTeam>'.

```
matekoro@MatePC:/mnt/d/github/installation_guide$ git clone https://github.com/tens-ml/4413ConstellationTeam
Cloning into '4413ConstellationTeam'...
remote: Enumerating objects: 830, done.
remote: Counting objects: 100% (248/248), done.
remote: Compressing objects: 100% (157/157), done.
remote: Total 830 (delta 99), reused 165 (delta 38), pack-reused 582
Receiving objects: 100% (830/830), 4.96 MiB | 11.39 MiB/s, done.
Resolving deltas: 100% (304/304), done.
Updating files: 100% (75/75), done.
matekoro@MatePC:/mnt/d/github/installation_guide$
```

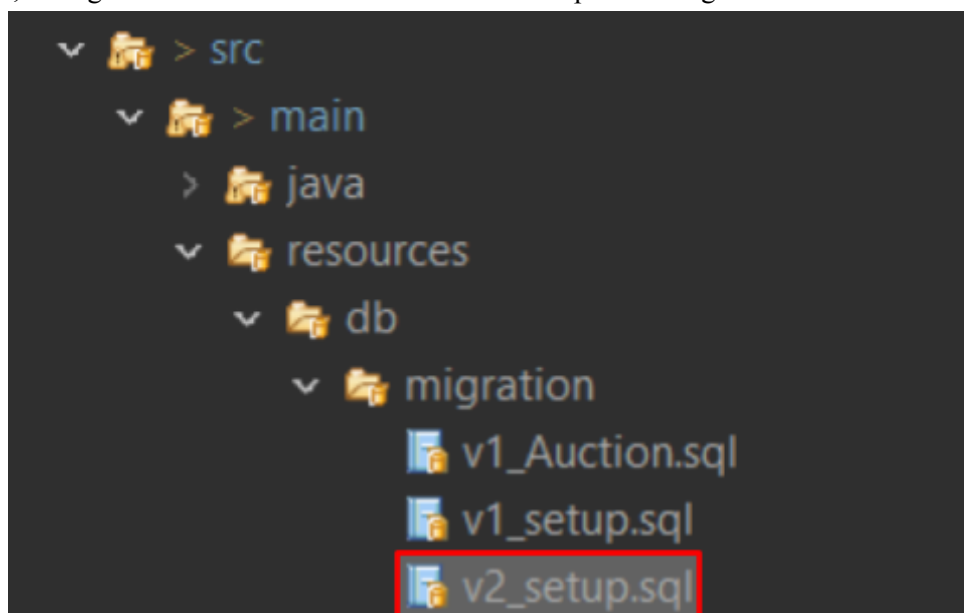
Next, import the project to your workspace by clicking on File > Import > Existing Maven Projects.



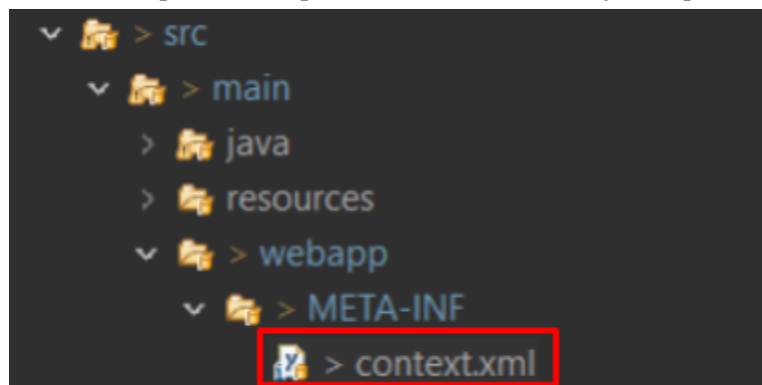
Once the project is in the workspace, click on the project > Run As > Maven install.



For the next step, SQLite is required. Open the script under `src > main > resources > v2_setup.sql` and run it on SQLite, noting the location of the database that the script is writing to.

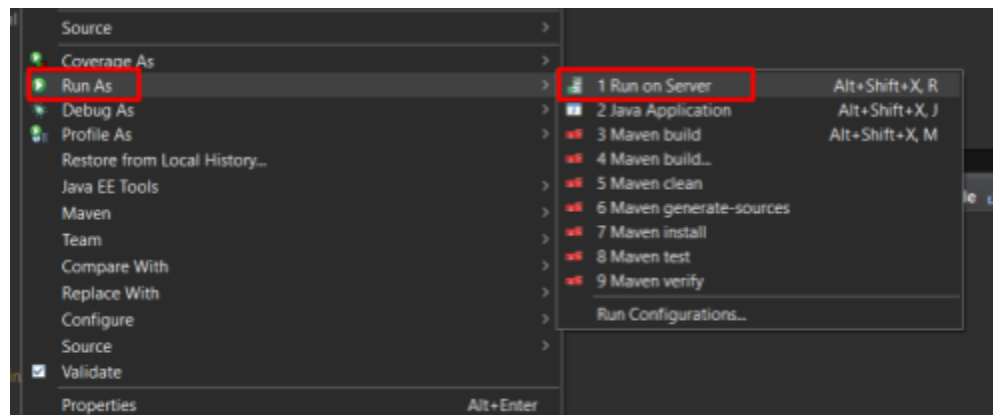


Using the location of the database that you have created your tables in using v2_setup.sql, go to src > main > webapp > META-INF > context.xml and within the xml file, change the highlighted section to the location of the database from the previous step. The url should be url="jdbc:sqlite:<YOUR PATH>">



```
context.xml ×
1 <?xml version="1.0" encoding="UTF-8"?>
  Bind to grammar/schema...
2 <Context privileged="true" reloadable="true">
3
4   <WatchedResource>WEB-INF/web.xml</WatchedResource>
5   <Manager pathname="" />
6
7   <Resource name="jdbc/constellationDB"
8             factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
9             type="javax.sql.DataSource" driverClassName="org.sqlite.JDBC"
10            url="jdbc:sqlite:/C:/sqlite/constellation.db"/>
11
12
13
14   <!-- PLEASE ONLY CHANGE THE PATH ABOVE -->
15   <ResourceLink global="jdbc/constellationDB" name="jdbc/constellationDB"
16                type="javax.sql.DataSource" />
17 </Context>
```

Once the previous step is done, the installation is complete and the persistence layer has been established, all that is left is to run the program on the tomcat server by clicking on the project > Run As > Run on Server.



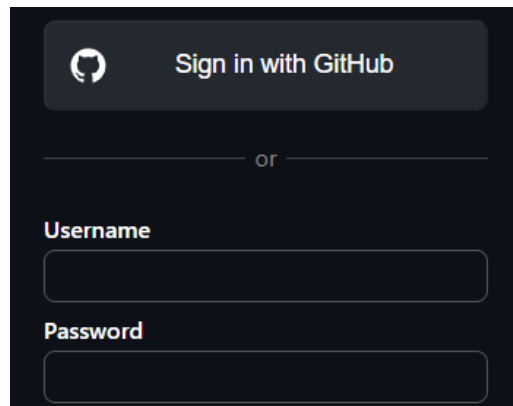
10 Deliverable 3 Additional Use Case: OAuth

10.1 OAuth Use Case

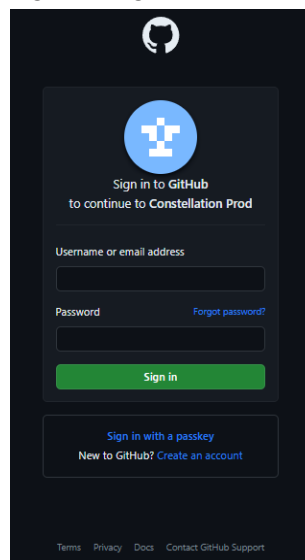
Open Authorization, henceforth called OAuth, is an authorization framework that allows third-party applications to obtain access to a service on behalf of the user. For successful OAuth, a user must be registered with a third-party application (Github in the case of this project) and be able to log in. The OAuth option should become available on the login screen of the application prompting the user to log in via a method that is not just a simple username and password. The application should accept any valid Github login and allow the user to successfully log in. If the OAuth account login fails, then so should the login for the application. Upon failure to login, the user should be taken back to the login page. Upon success of login, the user should be taken to the main bidding page.

10.2 OAuth Implementation

For the purposes of this project, the OAuth implementation involves Github. The application allows valid github users to log in. The username upon login would become the username that is associated with the github account. Below is an image of what this looks like.



After this is done the user goes onto the Github login page. Wherein they can input their username and password to successfully log in using OAuth.



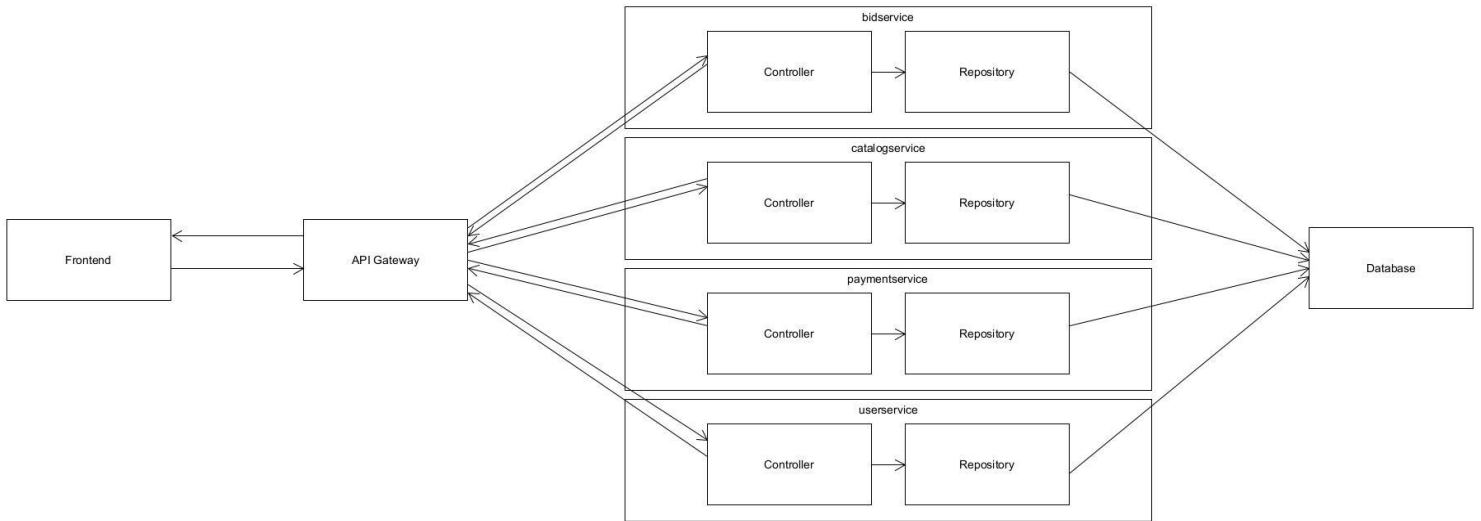
11 Deliverable 3 Architecture

The architecture of the system has changed since Deliverable 2 to accommodate the changes required to deploy each service as an independent container. A combination of MVC , Repository, and microservices architecture was used in our system. The base architecture on the front-end, Model View Controller (henceforth MVC), is still heavily prevalent as it allows for an easy-to-follow understanding of the project, as well as allowing scalability through modularity. On the backend we opted to use the facade design pattern in our implementation of API gateway to effectively manage interactions between our front and back end. To add additional features the existing structure would not have to change, as creating the backend and frontend and then connecting it with a correct gateway function would suffice. The current structure is in the diagram below.

The project uses Spring as well as Google Cloud (discussed further in Section 12) to allow for continuous deployment whenever changes are made in the main branch of the GitHub repository. Since there is an existing structure for database access using Spring, a repository was added to each service. This allows each service to write to their own tables exclusively, as well as allowing each service to read from all present tables. This is a Repository architecture, and while often in a microservice architecture you may see separate databases, for the purpose of this project we decided that the additional databases were unneeded complexity. When doing software engineering, one key thing to remember is that simplicity is the most important thing for any system's success.

Other architectural styles were also considered. Service-Oriented Architecture was considered as it would fit well with each independent service, however a key requirement for this deliverable was to have independently deployed microservices which would not have the normal coupling found with service oriented architecture. Another architectural style we considered was event based architecture, wherein we would use something like Kafka or RabbitMQ to broker messages between the back-end microservices. Given the minimal communication between microservices, we decided it would be much easier to just stick with using HTTP REST communication between the services. Should we expect our system to get more complex, we would likely implement some kind of event based messaging to better decouple and manage this backend communication.

The team feels that the chosen architecture for the Constellation Auction House fits well. It allows the project to be highly adaptable, and it also allows for fast automatic deployment. Furthermore, the current architecture allows for the seamless addition of new services, thus highlighting its scalability. If new requirements were added, the team would be able to add them to the existing structure without disturbing what is already in place.



12 Deliverable 3 Deployment/Installation

For deploying our application, we opted to automate the building and deployment process using cloud continuous deployment tools. To do this, we first set up the repository with separate modules/directories for each one of our system components. For our front-end, we are using a React-based javascript front-end framework called Next.js. Next is responsible for routing and static page generation, and its integration with Vercel allows for a seamless continuous deployment process. With Vercel you can simply select the repository that contains a NextJS application, and it will handle all of the setup, environment management, deployment management, logging, CI/CD, and many other useful features to control and monitor the front end. In order to deploy to Vercel, you must give Vercel the URL of the gateway via the environment variables. This is done so that when developing locally we can pass in the local addresses of the services we are developing, and in production, we can configure Vercel to pass in the production API gateway. With Vercels automated CD, making changes to the front-end is as simple as pushing an update to the constellation-frontend repository. From there, a trigger will automatically start a new deployment on Vercel. If this build is successful, it is immediately promoted to production replacing the previous deployment.

Each one of the back-end microservices are deployed as containerized Spring Boot applications. Our back-end services all are hosted via Google Cloud Platform (GCP), utilising Cloud Build for automating the deployment once again (CD), Cloud SQL for the data persistence, and Cloud Run for hosting our containers as services. Within Cloud Build, we set up a trigger for each of our back-end microservices to watch their respective directories for changes. When changes occur (i.e. when changes are pushed to the main branch), Cloud Build will automatically begin building a new container, pushing it to the Google Container Registry, and then deploying them via Cloud Run. Cloud Run simply takes an image from the container registry and deploys it as a service. After this initial setup, deployment is as simple as pushing changes to main. Any changes to the front-end or back-end services will automatically trigger a redeploy of whatever

was changed. We then have static urls, a publicly accessible gateway mapping them all, and a completely functional front-end.

13 Deliverable 3 Security Vulnerabilities

In terms of security vulnerabilities we opted for 2 authentication protocols, credential based authentication and OAuth2 based authentication. For credential based authentication, the login system is secure. One thing that could use further improvement is developing a system to generate tokens for each credentialed user as well as OAuth users, and then using that token to authenticate any further requests from that user. As written right now, the backend does not perform this kind of authentication because it was outside the scope of this course. From the front-end however, any page that is accessed without authentication will simply redirect back to the login page. For us to further secure the dashboard we would want to generate JWTs for each user login and use that to authenticate via the gateway. While the website itself is secure, malicious users could access the backend without authentication. To fix this we could set up Spring Security and write some token generation logic within the user service.

14 Deliverable 3 Demo

The demo will be given on zoom on Tuesday 2023/12/12.

15 Deliverable 3 Testing

Testing for Deliverable 3 is a combination of curl commands as well as a Postman test suite involving many requests. The requests can evaluate the error codes presented. Request response 404 not found, 401 not authorised, 400 bad requests and others are used to evaluate if the backend is correctly connected to the frontend while ensuring that any unauthorised access or invalid requests are handled as such. As well, each error returned should have a message identifying what went wrong on the backend. The test cases can be found under the directory 'Tests'.

One quick note regarding validation, while we have implemented some of the validation on the backend, all of the input is fully validated on the frontend as well. When our system is deployed in production, the only server that may access the backend is through our frontend, hence why all of the validation is being done there. So when you attempt to submit forms with invalid information, you may be prevented in some cases on the frontend where you are not prevented from doing so on the backend. In all critical cases, we have made sure to properly validate the backend when needed.