Constellation Auction House
Design Document

Version:	2.0
Print Date:	2023/10/20
Release Date:	2023/10/20
Approval State:	Approved
Approved by:	Constellation Team
Prepared by:	Constellation Team
Reviewed by:	Constellation Team
File Name:	Deliverable2_EECS4413.pdf
Document No:	2

Version	Date	Authors	Summary of Changes
1.0	2023/10/20	Ali, Alex, Dwumah, Mate (The Constellation Team)	Initial Release
2.0	2023/11/11	Ali, Alex, Dwumah, Mate	Deliverable 2 Requirmenets

Name	Signature	Date
Mate Korognai		2023/10/20
Alex Arnold		2023/10/20
Ali Sheikhi		2023/10/20
Dwumah Anokye		2023/10/20

Table of Contents

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 organized 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 CatalogueServerApp focuses solely on item management, while the AuctionServerApp handles bids. This high cohesion simplifies maintenance and minimizes 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 organize 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 organization and manageability of the user interface.

3   Sequence Diagrams

The Sequence diagrams are listed in the hyperlink below. 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

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.
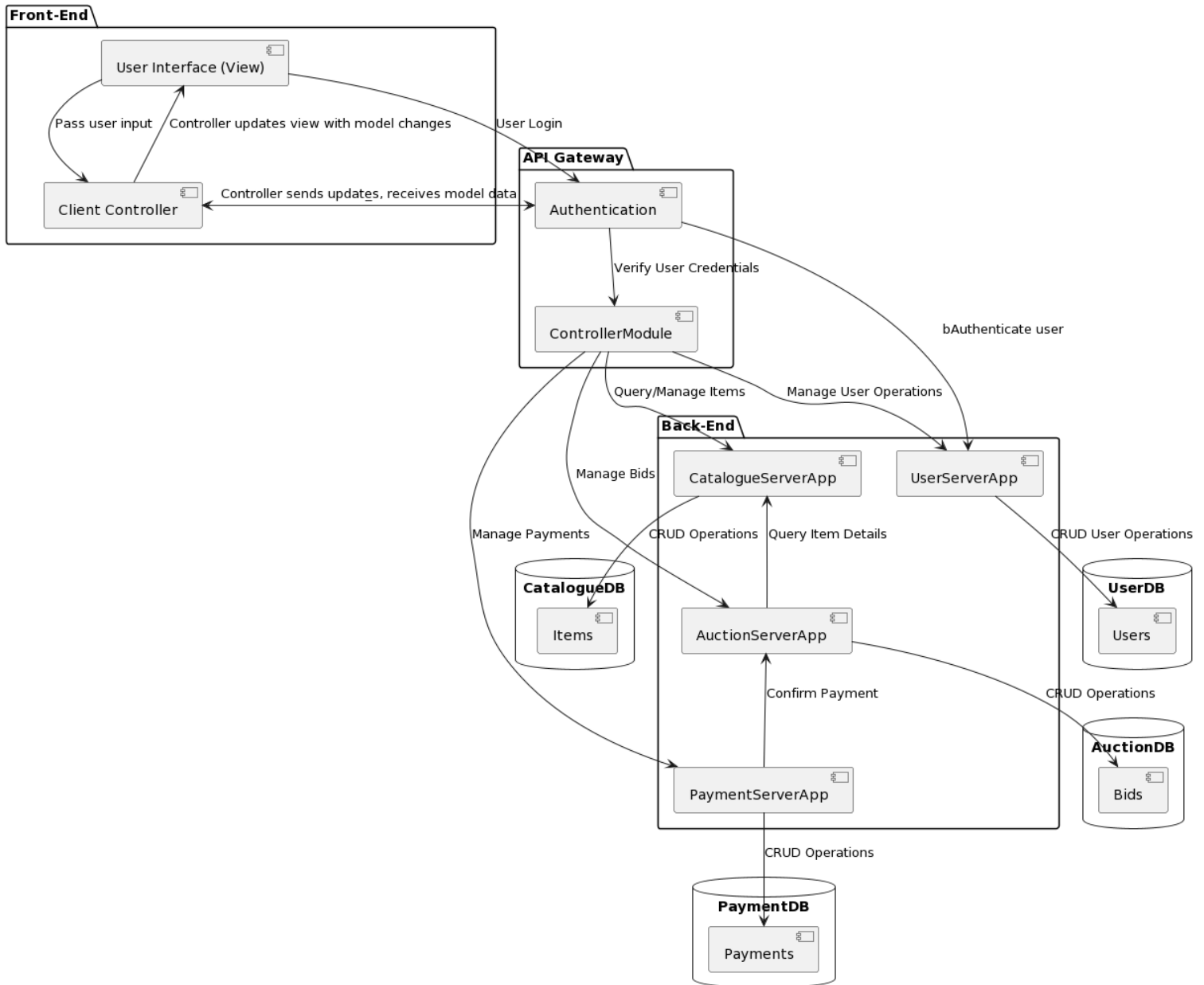

Figure 1: Model Diagram of preliminary design

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
- ➔ Introduction
- ➔ Test Cases 1-16
- ➔ Individual Designs
- ➔ Gantt Chart

- ➔ Major Design Decisions
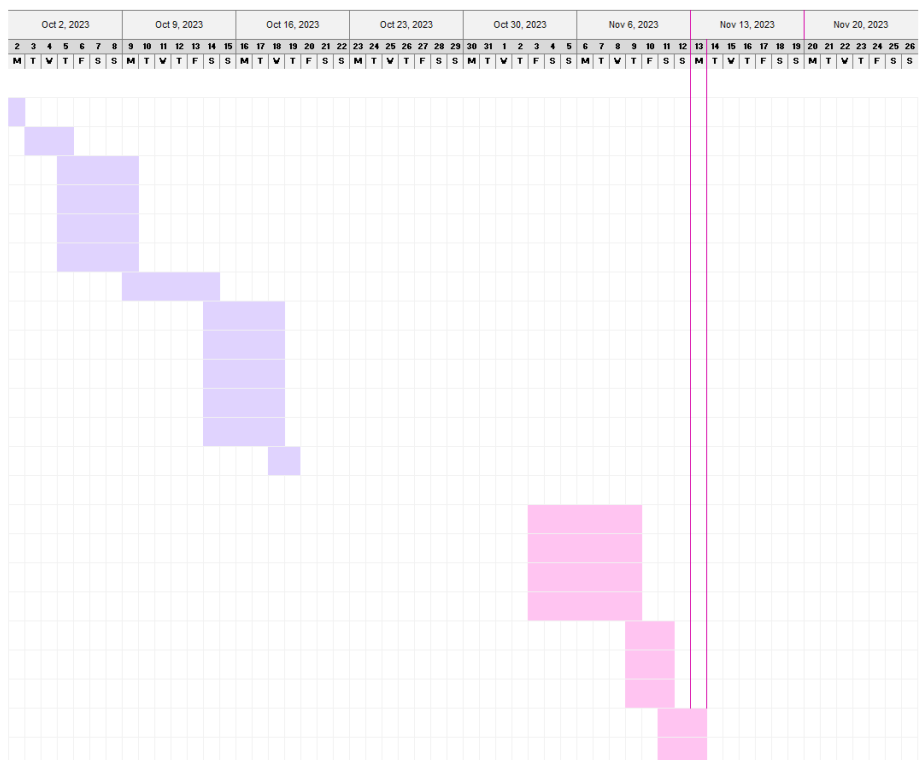- ➔ Sequence Diagrams
- ➔ Activity Diagrams
- ➔ Architecture
- ➔ Submission

Gantt Chart

6.2   Group Meeting Logs

2023/10/05	Everyone Attended (Ali, Alex, Dwumah, Mate) Got together for the first time and began discussing the project Talked about how to begin working on the project. Things like what languages to use, what architecture style to approach this problem with. Talked about the software tools we will use for the design phase.Talked about using MVC for the primary architecture of the project. Decided to work on Test Driven development, each UC has a person that works on those tests: ➔ Dwumah - Use Cases 6 and 7 ➔ Ali - Use Cases 2 and 3 ➔ Alex - Use Cases 4 and 5 ➔ Mate - Use Case 1 **Next Meeting 2023/10/09 6:00pm**
2023/10/09	Everyone Attended (Ali, Alex, Dwumah, Mate) Looked over the test cases Discussed Architecture that we will use 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. **Next Meeting 2023/10/14 6:00pm**
2023/10/14	Everyone Attended (Ali, Alex, Dwumah, Mate) 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. Tasks were assigned: ➔ Dwumah - Sequence Diagrams ➔ Ali - Activity Diagram ➔ Alex - Architecture & Major Design Decisions ➔ Mate - Gantt Chart & Tables for Architecture **Next Meeting 2023/10/18 8:30pm**
2023/10/18	Everyone Attended (Ali, Alex, Dwumah, Mate) Discuss the tasks that need to be done before submission. List of things to do: ➔ 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	Everyone Attended (Ali, Alex, Dwumah, Mate) Began working on Deliverable 2 Discussed how to split up the work Allocated Tasks:

	➜ 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) **Next Meeting 2023/11/9 9:00pm**
2023/11/9	Everyone Attended (Ali, Alex, Dwumah, Mate) 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 Allocated Tasks: Everyone must make a simple html page for the usecases that relate to their code from the previous meeting ➜ 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 **Next Meeting 2023/11/11 8:00pm**
2023/11/11	Everyone Attended (Ali, Alex, Dwumah, Mate) Alex showcased how he went about connecting frontend and backend modules so that the login and signup pages 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 Tasks were assigned as follows ➜ 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 **Next Meeting 2023/11/12 8:00pm**
2023/11/12	Everyone Attended (Ali, Alex, Dwumah, Mate) Issues: Login not needed to access everything else Showcased existing pages and how they are connected. Done the catalogue list page, as well as the login, signup and main menu Tasks were assigned as follows ➜ 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 **Next Meeting 2023/11/13 8:00pm**
2023/11/13	Everyone Attended (Ali, Alex, Dwumah, Mate) Went over project details and looked at the product, submitted github link

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](#)

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/adapter 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:
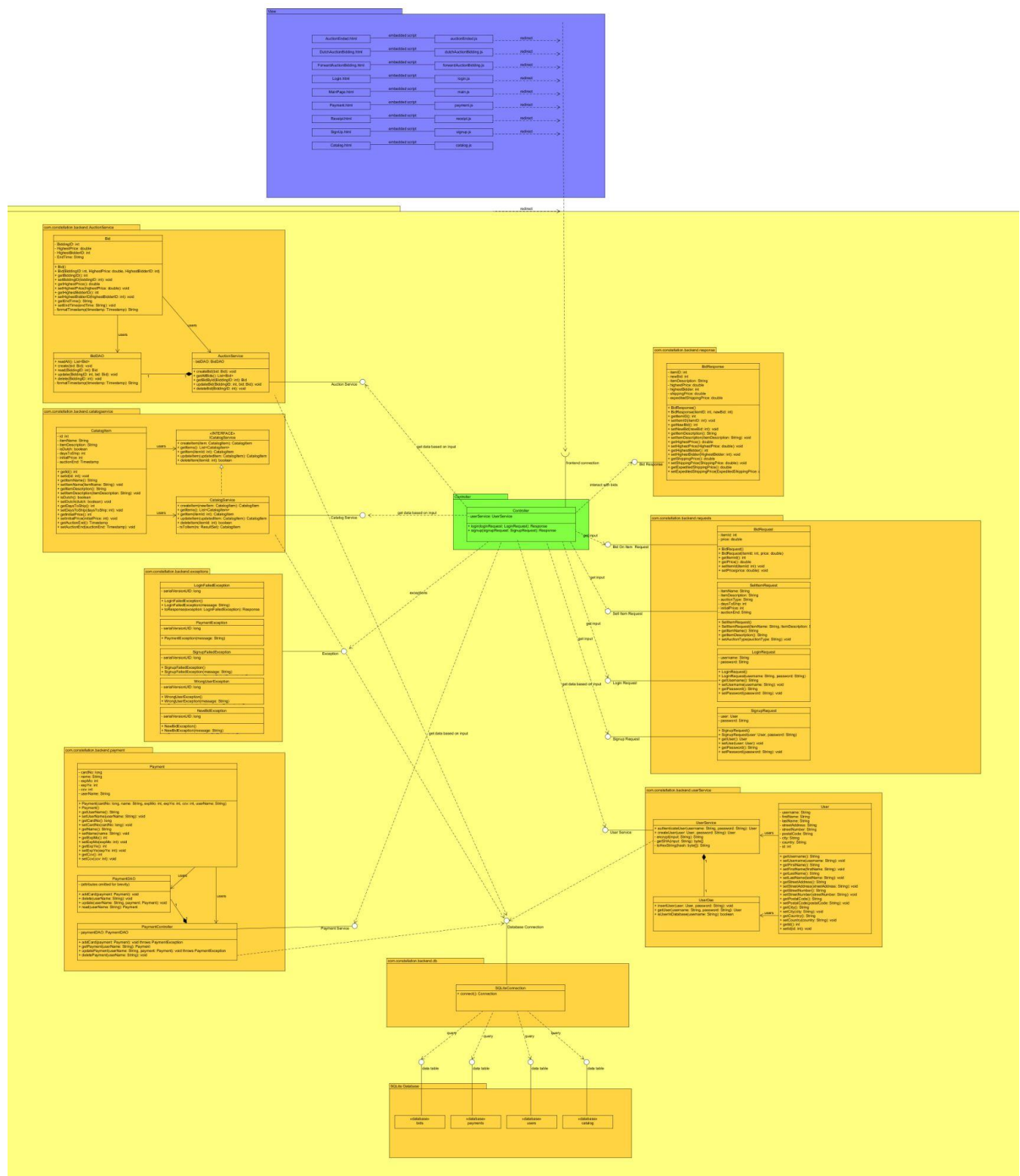
➔ [UML Diagram of Constellation Auction House](#)

Figure 2: UML Diagram based on Deliverable 2 Implementation

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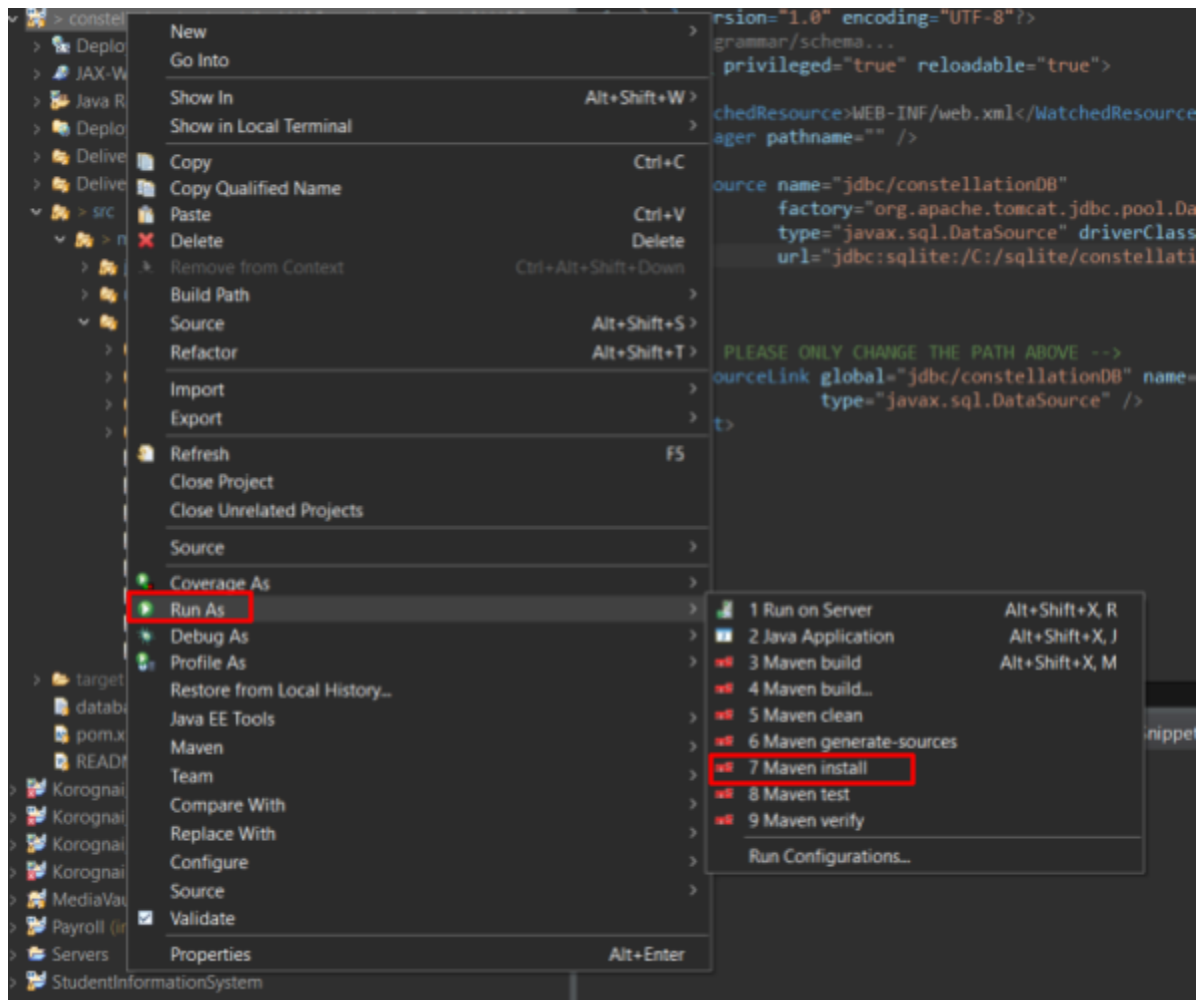
First and foremost, make sure that the system is present on your device by cloning the repository 'git clone https://github.com/tens-ml/4413ConstellationTeam'.
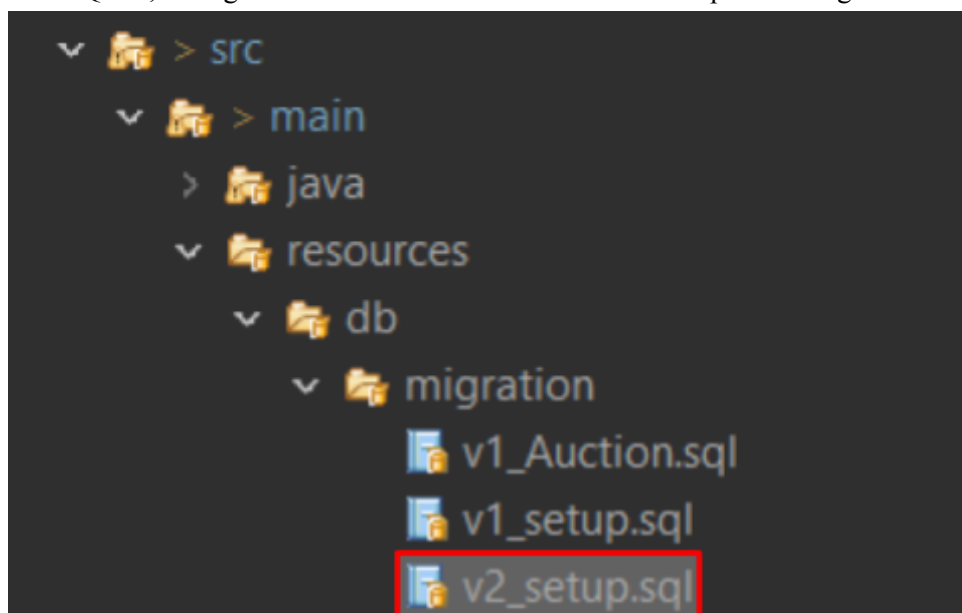

Next, import the project to your workspace by clicking on File > Import > Existing Maven Projects.
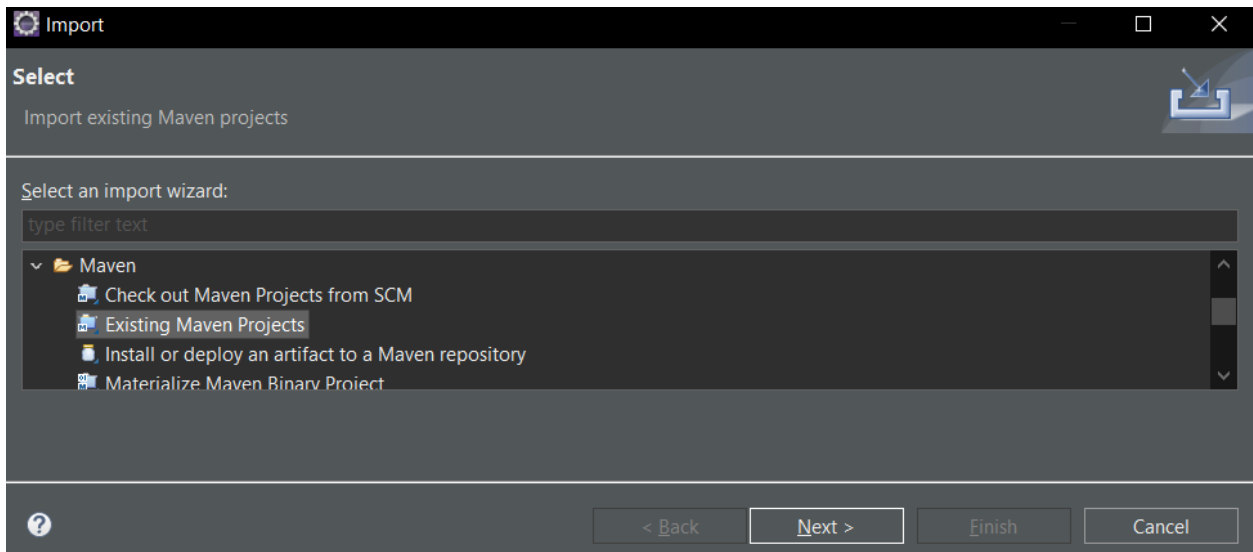
Once the project is in the workspace, click on the project > Run As > Maven install.

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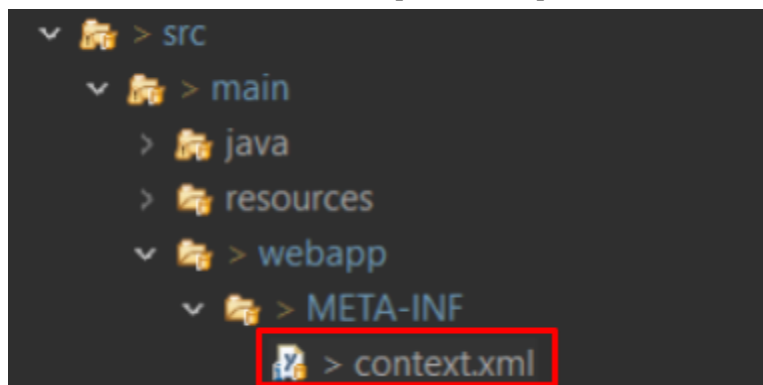
Once the previous step is done, the installation is complete and the persistence layer has been established, all that is left is to run the program on the tomcat server by clicking on the project > Run As > Run on Server.