|  |
| --- |
| **CAB302: Software Development** |
| Electronic Asset Trading Platform: Report |
| Semester 1, 2021 |

**Group:**

* **Adrian Ash: N1064937**
* **Matthew Biggs: N9181717**
* **Christopher Paterson: N9175920**
* **Adrian Roy: N9748792**

**Contents**

[**Requirements** 2](#_Toc73468668)

[**Must have** 2](#_Toc73468669)

[**Should have** 3](#_Toc73468670)

[**Nice to have** 3](#_Toc73468671)

[**Detailed Design** 4](#_Toc73468672)

[**Design of Classes** 4](#_Toc73468673)

[**Interaction of Classes/Methods** 4](#_Toc73468674)

[**Design and Dataflow (GUI)** 4](#_Toc73468675)

[**Database Schema** 7](#_Toc73468676)

[**Network Protocol** 8](#_Toc73468677)

[**Deployment (final submission) (might be in readme, unsure)** 8](#_Toc73468678)

[**Database Setup** 8](#_Toc73468679)

[**Java Frameworks** 8](#_Toc73468680)

[**……** 8](#_Toc73468681)

# **Requirements**

## **Must have**

General:

* Make application to buy and sell organisational assets
* Budget: electronic credits (assigned to each organisation)
* Marketplace Model: add buy orders and sell orders. Both need to input; BUY/SELL, organisational unit, asset name, quantity, price, date
* Orders are successful if the there is a sell order that is less than or equal to the buy order price.
  + Eg. BUY order: 100 CPU hours at 10 credits each.
  + SELL order = 50 CPU hours at 5 credits each.
  + RESULT = SELL order complete. BUY Order: 50 hours at 10 credits. First 50 hours are bought at 5 credits each.
* Client connects to server to list trades.
* There should be no artificial limit to number of commodities in the database, no limit to number of trades that are listed and no limit to number of users in the system. (will be implement by default)
* Users will be trading as part of an organisational unit. They will have access to their organisational unit’s credit balance and assets.
* Restrict ability to be able to buy for more credits or sell for more assets than the organisational unit has (cannot go below 0)
* GUI

GUI:

* List their own organisational units offers
* Add/Remove/Edit their own organisational units’ orders

IT Admin Team:

* Able to create new organisational units
* Edit the number of credits each unit has
* Edit the number of assets each unit has
* Create new asset types
* Able to add new users and assign them passwords and assign them to organisational units
* Admin Accounts (for IT admin team) (roles)
* Able to create new admin accounts for new IT admin people
* Able to do these from the GUI with a special admin account

Security:

* Login System: Each user must have their own username and password so that only authorised users from each organisational unit are able to trade
* No plain text passwords should be sent over the network, at least hash the password before sending it over.
* No plaintext passwords in the database either

Database:

* Use MariaDB/PostgreSQL/SQLite
* Stores User Info (username, password, account type, organisational unit)
* Organisational unit info (organisational unit name, credits, assets, quantity of each asset)
* Asset types (asset names)
* Current trades (BUY/SELL, organisational unit, asset name, quantity, price, date)
* Trade history (same as current trade info) (only if trade is successful?)

## **Should have**

GUI:

* Nice friendly GUI.
* List current buy and sell orders (sort by date)
* List price history of each asset

System Administrators:

* Configuration file: server Ip address and port to connect to
* Configuration file: for the server to get port

## **Nice to have**

General:

* User should be able to change their own password without needing the IT team

GUI:

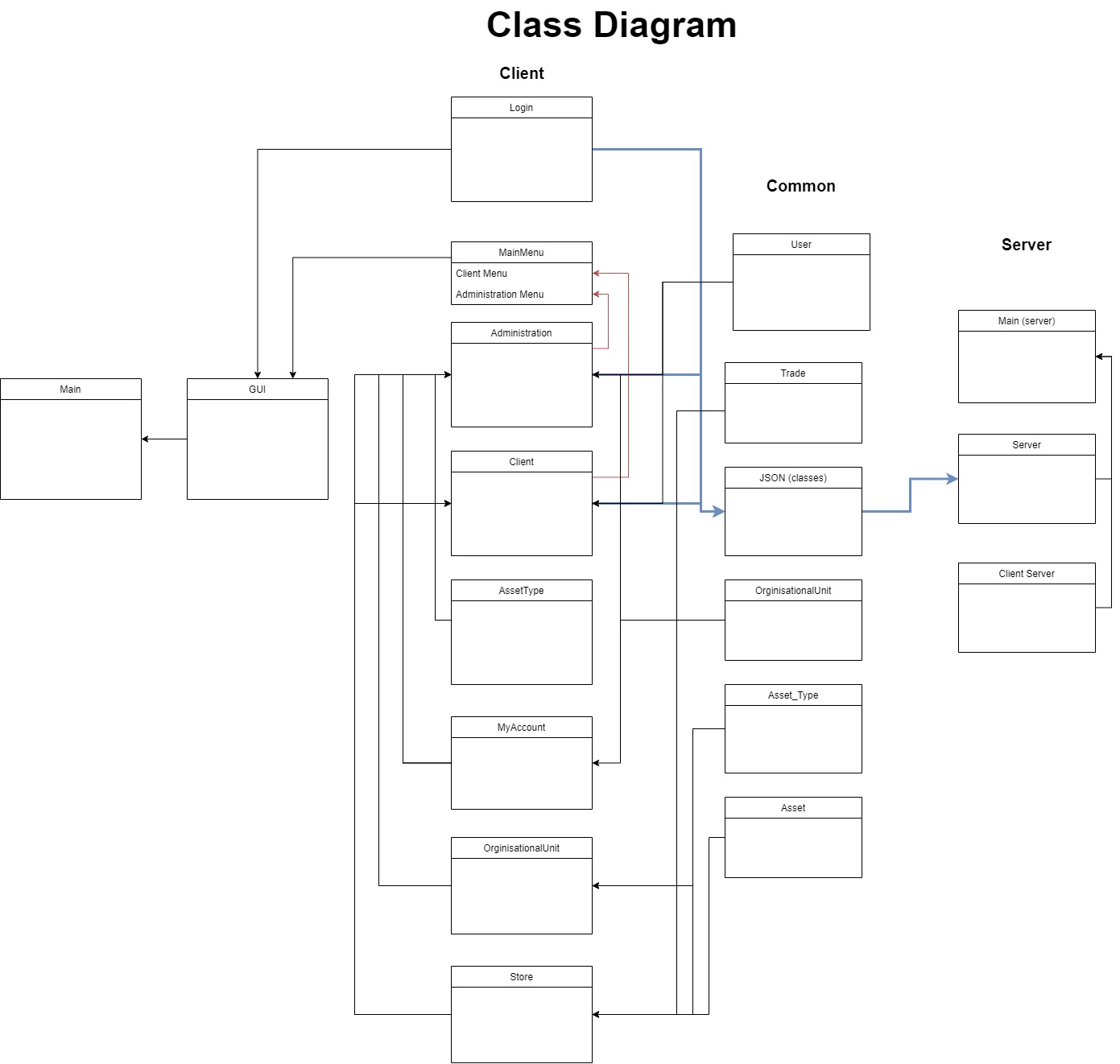
* List current buy and sell: sort by any
* Graph: Shows price over time (has search feature) (time consuming, complete last)
* User feedback: When a trade is reconciled, show a little message somewhere that the trade is fulfilled

# **Detailed Design**

## **Design of Classes**

[**Javadoc**](../Javadoc/index.html)

## **Interaction of Classes/Methods**



## **Design and Dataflow (GUI)**

**Screen 1: Login**

When first opening the program, the user is brought to the log in screen.

Graphical user interface, application

Description automatically generated

**Screen 2: Unit Asset/Trades**

After logging in, they are brought to the Unit Asset/Trades page which is like a homepage which shows all the current assets the organisation owns, and the trades it currently has in progress. Also allows them to cancel or edit their orders

Graphical user interface

Description automatically generated

**Screen 3: Store**

When the user selects the store button on the left panel, they are brought to the store, which shows all the assets they can place an order to either buy or sell.

A screenshot of a computer

Description automatically generated with low confidence

**Screen 4: Asset Selected from Store**

Once the user selects an asset from the store, they are brought to a page where they can place a buy or sell order. They are also shown the price history and current orders of that specific asset.

Graphical user interface, application

Description automatically generated

**Screen 5: Admin Page**

If the user has admin privileges, an admin button will appear on the bottom left

Graphical user interface, application

Description automatically generated

**Screen 6: Edit Organisational Units**

Graphical user interface

Description automatically generated

**Screen 7: Create Organisational Units**

Graphical user interface

Description automatically generated

**Screen 8: Create Asset Types**

Graphical user interface

Description automatically generated

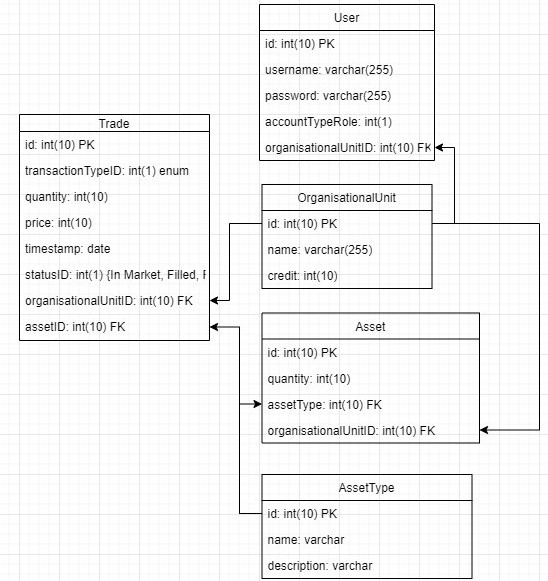
**Screen 9: Create Users**

Graphical user interface

Description automatically generated



## **Database Schema**



## Network Protocol

Both the server and client sides operate on separate threads. When a request is made on the client side, through the Client class, a new socket connection is established to the Server. The Server waits until a socket connection is established, then invokes a request handler on a separate thread. The Client then sends a payload request object, containing the payload object to be sent to the server, a request payload type as an enumerator, an object type stored as a string and a string checksum. The payload request object is then sent to the request handler by an object output stream.

The request handler on the server side receives the stream, reassembles the payload request object and uses the object type string to determine what the object type of the payload object is. The request handler then interfaces with the database through a Hibernate session, and its actions are determined by the request payload type. A “Buy” or “Sell” type request will cause the request handler to cast the request payload object as a Trade class object and create a new entry in the database’s trade table. A “Get” type request will cause the request handler to pass a select query to the Hibernate session to return an object matching the request payload object or null, if there is no matching entry in the database. A “List” type request will cause the request handler to pass a select query to the Hibernate session to return a list of objects matching some property of the payload object, or null if no such entry exists in the database. The “Create”, “Update” and “Delete” type requests will create, update or delete entries, respectively, from the table matching the object type.

Once the request handler has processed the request object, a response object is created and passed back to the client by an object output stream. The request handler socket is then closed and the request handler process ends. The Client reads in the response object, then closes the socket and returns the response.

# **Deployment (final submission) (might be in readme, unsure)**

## **Database Setup**

## **Java Frameworks**

## **……**