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

**Milestone 2 Video:** [**https://www.youtube.com/watch?v=OxrmQS30-WU**](https://www.youtube.com/watch?v=OxrmQS30-WU)

**Group:**

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

**Contents**

[**Requirements** 2](#_Toc72065365)

[**Must have** 2](#_Toc72065366)

[**Should have** 3](#_Toc72065367)

[**Nice to have** 3](#_Toc72065368)

[**Detailed Design** 4](#_Toc72065369)

[**Design of Classes** 4](#_Toc72065370)

[**Interaction of Classes/Methods** 4](#_Toc72065371)

[**Design and Dataflow (GUI)** 4](#_Toc72065372)

[**Database Schema** 6](#_Toc72065373)

[**Network Protocol** 6](#_Toc72065374)

[**Sprint Planning** 7](#_Toc72065375)

# **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**

* Class Diagram (UML) (if you guys want)
* Or a description of interconnectivity

## **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

Need to do admin page



## **Database Schema**

Graphical user interface, application, table, Excel

Description automatically generated

## **Network Protocol**

* Description of data to and from the server and user
* How that data should be interpreted???
* Basically, a description of everything required to recreate a compatible server

# **Sprint Planning**

|  |  |
| --- | --- |
| **Adrian Ash** | Documentation:   * Interaction of classes/methods   GUI elements:   * Main Layout * Unit Asset/Trades Page * Admin panels (create new users, create new asset types)   Object Logic:   * Logic related to GUI pages * User logic for admin users |
| **Matthew Biggs** | Backend (server):   * Ensuring hibernate works correctly * Multi-threading (allow multiple connections) * API development (client-server calls)   Unit Testing:   * Creating unit tests for Asset class * Creating unit tests for Asset Type class   Object Logic:   * Logic involving hibernate |
| **Christopher Paterson** | Unit Testing:   * Creating unit tests for User class * Creating unit tests for Organisational Unit class * Creating unit tests for Trade class   GUI elements:   * Store Page   Object Logic:   * Creating and display Asset Types on Store Page * Linking and passing required information to the connected Buy/Sell Page |
| **Adrian Roy** | Documentation:   * Network Protocol * Remaining GUI mock-ups   GUI elements:   * Buy/Sell Page of store (requires a lot of object logic) * Settings Page (reset password and display user information) * Creating the colour scheme (if time permits) * Price History Graph (if time permits) |

(all members will share all areas of the workload as its part of specification however members will focus on certain elements that pertain to skills)