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

**Group:**

* **Adrian Ash: N1064937**
* **Matthew Biggs**
* **Christopher Paterson**
* **Adrian Roy**

**Contents**

[**Requirements** 2](#_Toc68301667)

[**Must have** 2](#_Toc68301668)

[**Should have** 3](#_Toc68301669)

[**Nice to have** 3](#_Toc68301670)

[**Detailed Design** 3](#_Toc68301671)

[**Design of Classes** 3](#_Toc68301672)

[**Interaction of Classes/Methods** 3](#_Toc68301673)

[**Design and Dataflow (GUI)** 4](#_Toc68301674)

[**Database Schema** 4](#_Toc68301675)

[**Network Protocol** 4](#_Toc68301676)

[**Sprint Planning (for milestones)** 4](#_Toc68301677)

[**Adrian Ash** 4](#_Toc68301678)

[**Matthew Biggs** 4](#_Toc68301679)

[**Christopher Paterson** 4](#_Toc68301680)

[**Adrian Roy** 4](#_Toc68301681)

[**Deployment (final submission) (might be in readme, unsure)** 4](#_Toc68301682)

[**Database Setup** 4](#_Toc68301683)

[**Java Frameworks** 4](#_Toc68301684)

[**……** 4](#_Toc68301685)

[**References** 4](#_Toc68301686)

# **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

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

* Public Methods and Fields (where they are called)
* Arguments and Returns of each method
* Any Assumptions
* Exceptions

\*\*\*\* We will used JavaDoc for this and reference it \*\*\*\*

## **Interaction of Classes/Methods**

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

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

* Create user interface mock-up (drawing tools)
* Server GUI Mock-up (if we have one)

## **Database Schema**

* Tables and table columns (names and data types)
* Connections of tables (primary and foreign keys)
* Object-Role Modelling (if we want)

## **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 (for milestones)**

## **Adrian Ash**

## **Matthew Biggs**

## **Christopher Paterson**

## **Adrian Roy**

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

## **Database Setup**

## **Java Frameworks**

## **……**

# **References**

EndNote to pool references?

Update test