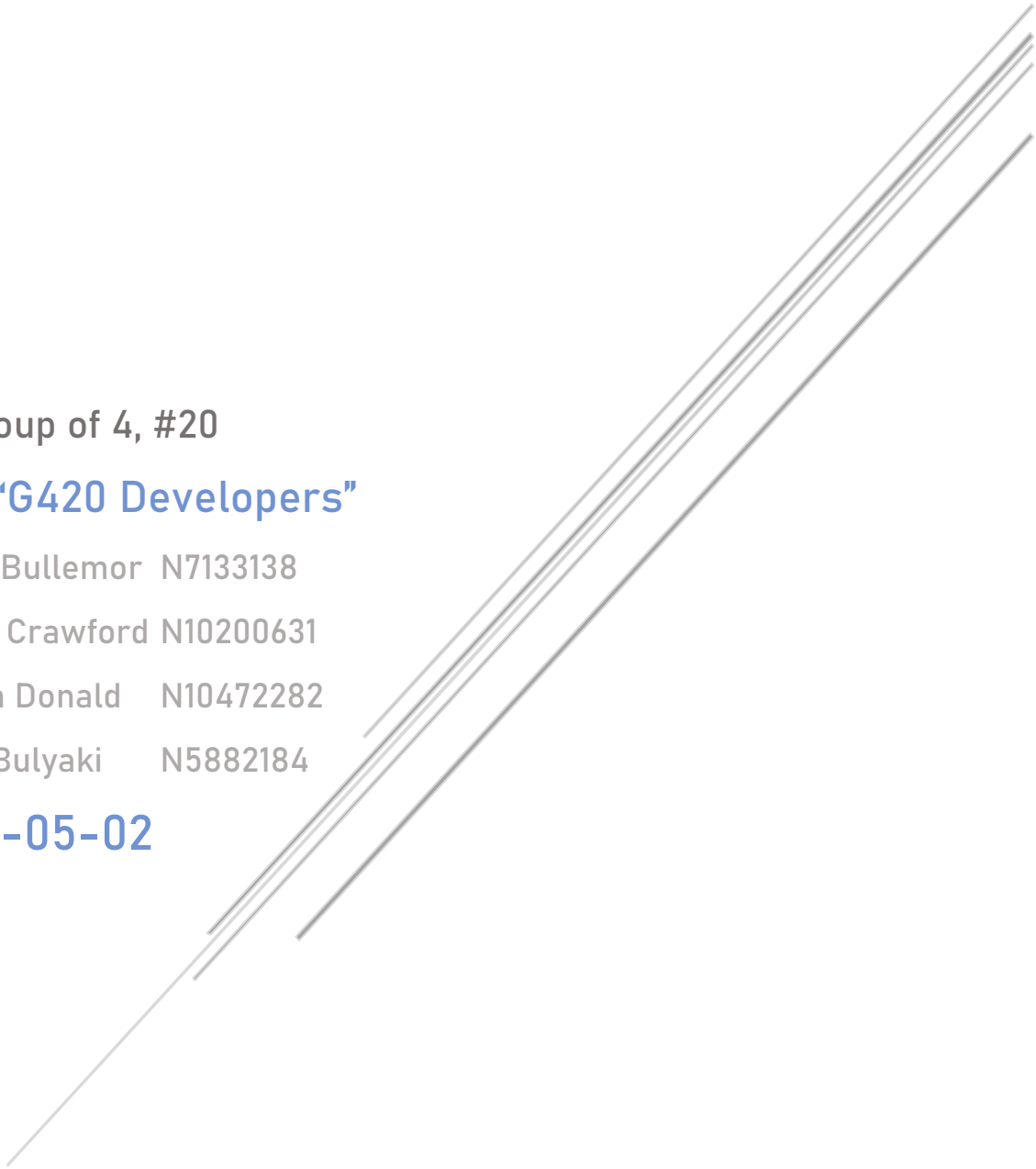
CAB302

Detailed Design Document

Online Trading Platform



By Group of 4, #20

AKA "G420 Developers"

Robyn Bullemor N7133138

James Crawford N10200631

Nathan Donald N10472282

David Bulyaki N5882184

2020-05-02

Contents

0 - Overview

Legitimate Corporation Pty Ltd approached G420 Developers to design an online trading platform. The objective was to facilitate an efficient exchange of internal resources, known as commodities (or an 2.1 - Asset). The proposition was that each company department was allocated a certain budget of universally-recognised credits, which they could then use to barter for commodities with anyone who was active on the online trade platform.

The idea is that users of this system do not tediously micromanage their transactions. Instead, they create buy or sell orders, where they nominate the type of commodity they wish to buy or sell, specify their desired unit price (to buy or sell), and finally the quantity thereof. Once a Buy or Sell order has been created, autonomous processes cross examine the compatibility of the existing Buy & Sell orders.

Transactions whose Buy & Sell terms are evenly matched will process automatically without the need for a user to intervene. To help avoid human users needing to micromanage Buy & Sell orders, trades will also occur automatically when a buy order is willing to purchase commodities for a greater unit price, than what existing Sell orders are listing. In this case, the program will protect the interest of the buyer by only paying for the nominated Sell price (the lower value), and not the original, higher Buy price. The whole concept for this online trade platform is to help streamline interdepartmental resource-sharing. It then makes sense to create a program that has the convenience of inbuilt, automated functionality, to free employees up to do their regular, daily duties.

Equally important is the user interface. The client has expressly conveyed that they refuse to accept a terminal-based application, and would much rather interact with a GUI. Special considerations for the GUI design include for its' need to be intuitive, pleasant to use and appealing to the user (as well as maintaining the scope of the features requested). There has been some inspiration drawn from existing online trading platforms.

A database schema has been prepared to help manage the relationships between the data objects. Laying a sound foundation for the database will significantly aid in the effectiveness of the online trading platform to retrieve and store relevant data. The online trading platform must be flexible enough to not limit the number of commodity types (2.1 - Asset), users (2.5 - User) and admin members (2.3 - ITAdminUser). The online trading platform also incorporates network connectivity. It can host multiple users, ensuring that their login credentials are hash encrypted to protect their security.

The following sections provide a more detailed overview of the application's programming logic and structure.

1 - Javadoc

G420 Developers has taken the time to describe the online trade platform's methods and classes within an exported Javadoc. "overview-summary.html" (Figure 1) Is a good starting point for examining the Javadoc content, as it lists the main packages that contain the application's code content.

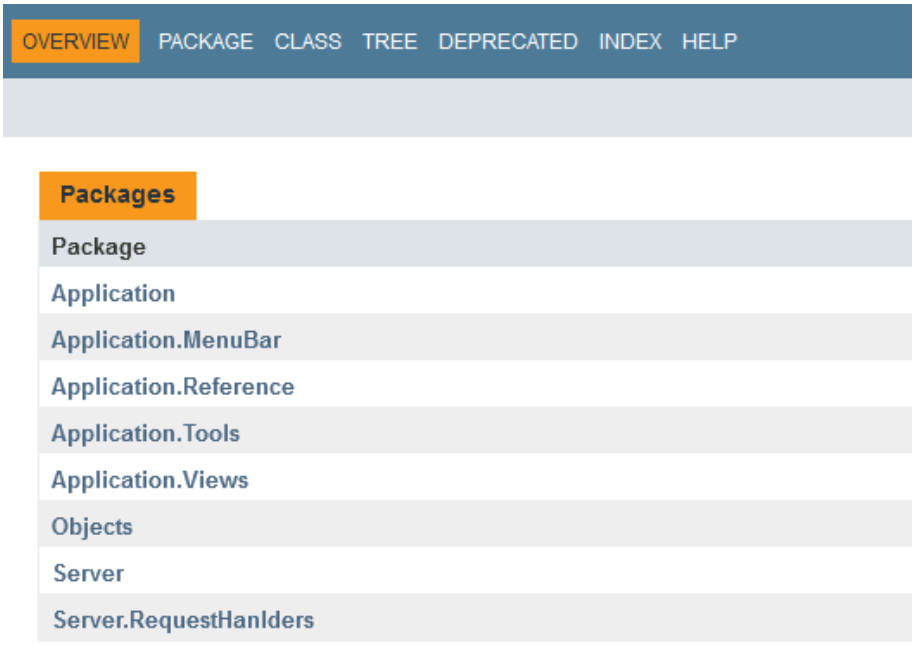


Figure 1: Javadoc "overview-summary.html".

2 - Backend Classes (Brief Overview)

Whilst the Javadoc does contain adequate descriptions for the backend classes (Figure 2) relating to the application's use of data objects, some overview for the backend functionality is provided here in section 2 for reader convenience.

Class Summary

Class	Description
Asset	The class that represents the Asset types in the system.
Department	The class that represents the Departments in the system.
ITAdminUser	The class that represents the IT Admin User by extending the User class.
OnlineTrade	The class that represents the Online Trade Listings of the system.
User	The class that represents the Users of the system.

Figure 2: Backend Classes.

2.1 - Asset

The class used for representing the "Asset" or the types of corporate resources or commodities to trade for credits.

2.2 - Department

The class used for representing the corporate department.
Also contains the number of credits currently available for the *Department* to spend.

2.3 - ITAdminUser

The class used for representing the users who are enabled with special privileges to manage the existing types of *Asset*s, as well as the delegation of credits to corporations, and managing users. An *ITAdminUser* can also do everything a normal *User* can.

2.4 - Online Trade

The *OnlineTrade* objects are the primary means of representing the necessary data relating to the Buy and Sell orders. The database relationships are used to associate the relatively simple *Department* objects with the more, comprehensive *OnlineTrade* objects. The *OnlineTrade* objects are designed to be flexible, where appropriately setting their *tradeType* will determine whether they're representing a Buy or Sell order.

2.5 - User

The class used for representing the users who are enabled with regular privileges to create Buy or Sell orders. A User cannot add more credits to a Department, nor manage commodities (*Asset*) or other *User*s.

3 - Graphical User Interface

3.1 - GUI Design

The GUI draws inspiration from an existing and popular online trading platform, the Steam Community Market (Figure 3). This example is relatively simple, but provides some good insight regarding the appeal of intuitive icons representing the resources being traded, along with other stats such as quantity and unit price.

Figure 3: Steam Community Market

An initial GUI concept depicted how a similar concept can be implemented with the notion of trading corporate resource commodities for a given unit price in credits. A central Box component is loaded with the content of all the active OnlineTrade objects being referenced from the database. The idea here is that the database contents can be converted into some sort of *Java ArrayList<>*, and then be displayed in a user-intuitive format.

Figure 4: GUI Initial Prototype showing active OnlineTrades

3.2 - GUI Classes / Package Overview

GUI design can become very complex very quickly, and easily runs the risk of losing maintainability as Java components are incorporated into an ever-increasing feature set. It was therefore important to determine a sound foundation as to the methodology for organising the GUI classes in a way that was intuitive, and maintainable from the start. This way the code will tend to progressively accumulate in content without accumulating technical debt [3].

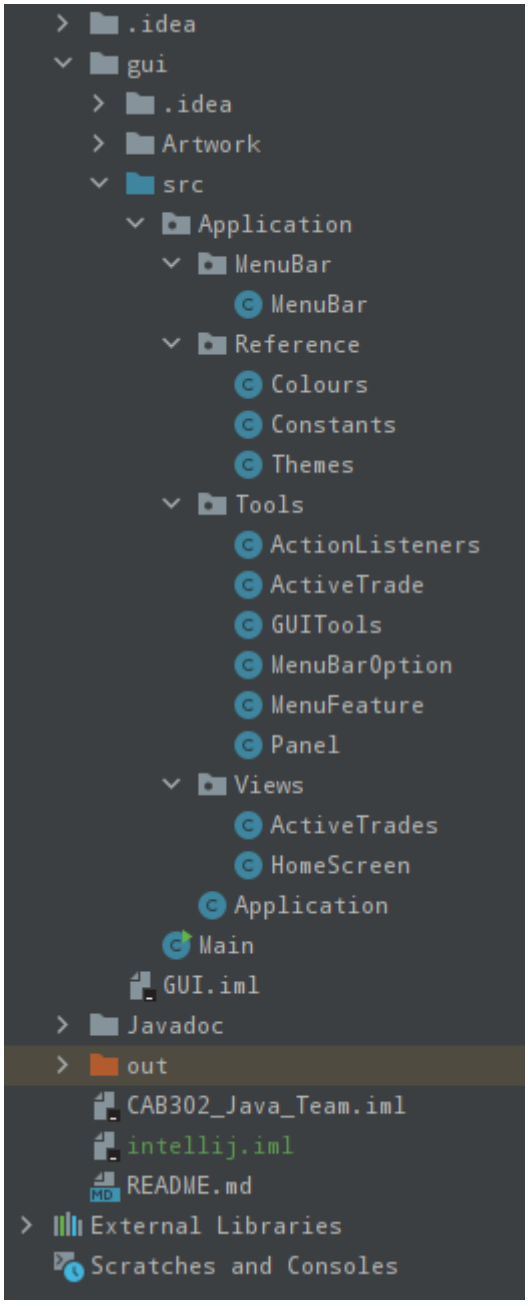


Figure 5: GUI Classes

Figure 5 demonstrates the compartmentalisation of each GUI class. The overarching package Application incorporates a number of sub-packages *MenuBar*, *Reference*, *Tools* and *Views*.

3.2.1 - MenuBar

As per its' name, *MenuBar* contains the necessary logic to arrange the Java Components that constitute the Menu bar that appears at the top of the GUI. This specifically relates to the menu buttons such as "File", "View" and "Help", along with all of their options (See Javadoc).

3.2.2 - Reference

Contains a number of classes, each of which primarily contain constants IE Java variables initialised with the *final* keyword. They contain the programmer-defined variables that determine the desired GUI characteristics (See Javadoc).

3.2.3 - Tools

Is a very foundational package that serves as the backend workhorse for the GUI. It houses the object classes and methods necessary for the intuitive and maintainable composition of the GUI code, particularly to do with features and parsing of data (See Javadoc).

3.2.4 - Views

Contains classes which mainly combine the functionality of the existing classes in order to produce the visual representation of a particular user view. Classes within the *Views* package do not bring any novel functionality into the program, in terms of new object types or methods, but rather they possess substantial methods which in turn call a combination of smaller methods to create the new View for the GUI to display (See Javadoc).

3.3 - General Functionality

Initial concepts depict the home screen (Figure 6) having large user icons to conveniently access the features they want without wasting time. Again, this program is being developed for workers in a corporate environment who might be busy, and engaged in multi-tasking duties. Having large, coloured buttons to navigate to different parts of the program may help make mental associations to features, and should help enable users to become more efficient over time.

Figure 6: Home Screen Concept

Whilst this is only a crude concept, the button captions do reveal the core set of features that G420 Developers aim to deliver for the client. These include a provision to create new Buy or Sell orders, manage existing Buy or Sell orders, create new commodity types, view active orders (*OnlineTrade* objects) that exist the market (server), inspect commodity trends (to determine competitive unit prices for Buy or Sell orders), and for *ITAdminUser*s only, managing *User*s, Credits and Commodities (*Asset*s).

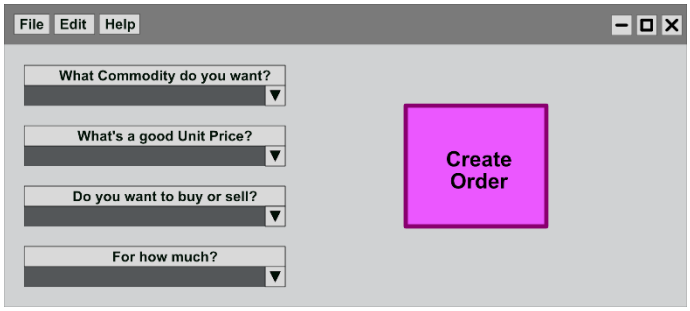


Figure 7: Create new Order

Figure 7 depicts an insight as to what one of these features "Create new Commodity" might look like. Here we see 4 x drop-downs that intuitively prompt the user regarding the specifics of their Buy or Sell order. Although this depiction is rather crude, and it's highly likely a better solution will be developed for the final version of this feature.

4 – Database Schema

4.1 – Component design description

Data is split into 7 tables: *Orders*, *Assets*, *Inventories*, *Trades*, *Users*, *OrganisationalUnits* and *Notifications*.

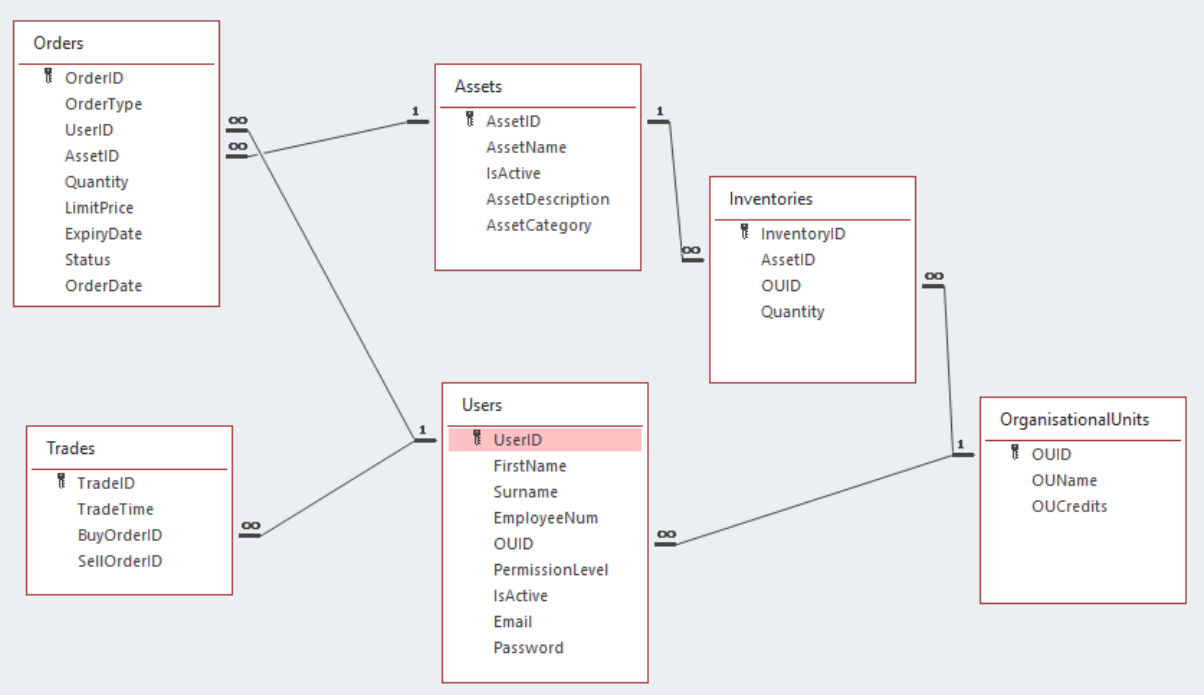


Figure 8: Database Relationships

Further details to be added once plan is more certain.

See Appendix 1 (Figure 10) for a full Object Role Model.
See Appendix 2: Database Script for full database script.

4.2 – Component interfaces

Information can be entered into the database either by the user, via the GUI or programmatically, as triggered by certain events.

An initial database has been populated based on information provided by Legitimate Corporation Pty Ltd including *Assets*, *Users*, *OrganisationalUnits* and *Inventories*. To maintain system stability, the *Trades* table is only modifiable via the built-in algorithms or manually by CAB302 Java Team.

4.3 – Permissions

Excluding primary keys, *Assets*, *Users*, *OrganisationalUnits* and *Inventories* tables can be created and modified manually by Database Administrators (2.3 - ITAdminUser). Records can be made inactive but cannot deleted to maintain system integrity.

*User*s (2.5 - User) will be able to edit their own password and will have read only access to their user information as well as active trade, order and asset information, inventory and details of their own Organisational Unit. Users can also edit and cancel orders which they have placed up until the time that the order is approved.

Organisational Unit Managers will have User level permissions, as well as the ability to modify their own OU Inventories, approve or cancel pending orders from within their OU.

4.4 – Workflows and algorithms

To be confirmed once coding begins

4.5 Triggers

When a new record is added to the Orders table a notification is sent to the originating OUs Manager for the order to be approve. Upon approval/non-approval, a notification will be sent to the originating User, advising them of the order status.

If an order has been approved, a search is performed on active orders to find a matching buy/sell order. If a matching order is found, a check is performed to ensure that the buying OU still has enough credits to complete the trade. Another check is performed to ensure that the selling OU has a sufficient Inventory. If either of these conditions cannot be met, the failing Order is skipped and the search will continue until a suitable Order is found. If there are no suitable Orders, no further action is taken.

Once a suitable match is found, a new record is added to the Trades table and both Order records are marked as complete. A notification is sent to both order users to advise that the trade has been completed. The selling OUs Inventory record Quantity is decreased by the required amount while the buying is increased. Similarly, the selling OUs *OUCredits* is increased by the required amount while the buying OUs is decreased.

At 12:01am, a search will be run on the Orders table to check the *ExpiryDate* of each active record. If the *ExpiryDate* has passed, the record will be marked as inactive. If the *ExpiryDate* is within the next two days, a notification is sent to the Order User advising them that their order is due to expire.

TBC - Partial fill orders ….. to be confirmed.
TBC - How will we handle linked records being made inactive? E.g. If an OU is made inactive, is a user still able to trade?

4.6 – Software requirements mapping

MariaDB has been used for the database server as it provides “performance and stability, especially in high load environments” [1] in an open-source platform [1].

5 - Networking Protocols

5.1 - Networking Design

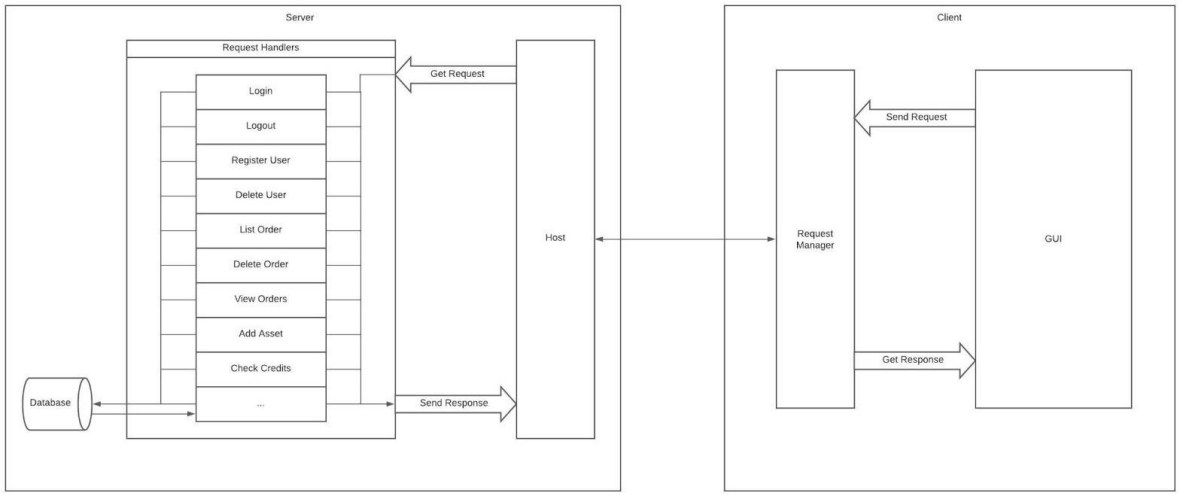


Figure 9: Network Design Diagram

On start up the client's program will read a simple configuration file which will give the servers IP and Port (the project will use a single server). Once connected, the user will be able to interact with the GUI to view, edit, and delete a multitude of data stored on the server's database. When the program requires data from the server, an appropriate request will be sent via a request manager. The server will use the appropriate request handler to decide if the request is eligible to be fulfilled, if so then the server will complete the request and send an appropriate response. This flow is visible in the network diagram (Figure 9)

5.1.1 – Server Classes

The public server classes and their descriptions can be seen in the Javadoc. All classes relating to the server are in the 'Server' package. Within this package there is a package, *RequestHandlers*, which holds each of the unique request handlers. Each handler has been given its own class rather than just a method. This is to make them easier to edit and flexible to add or remove handlers in the future.

6 - Appendix

Appendix 1: Database Object Role Model

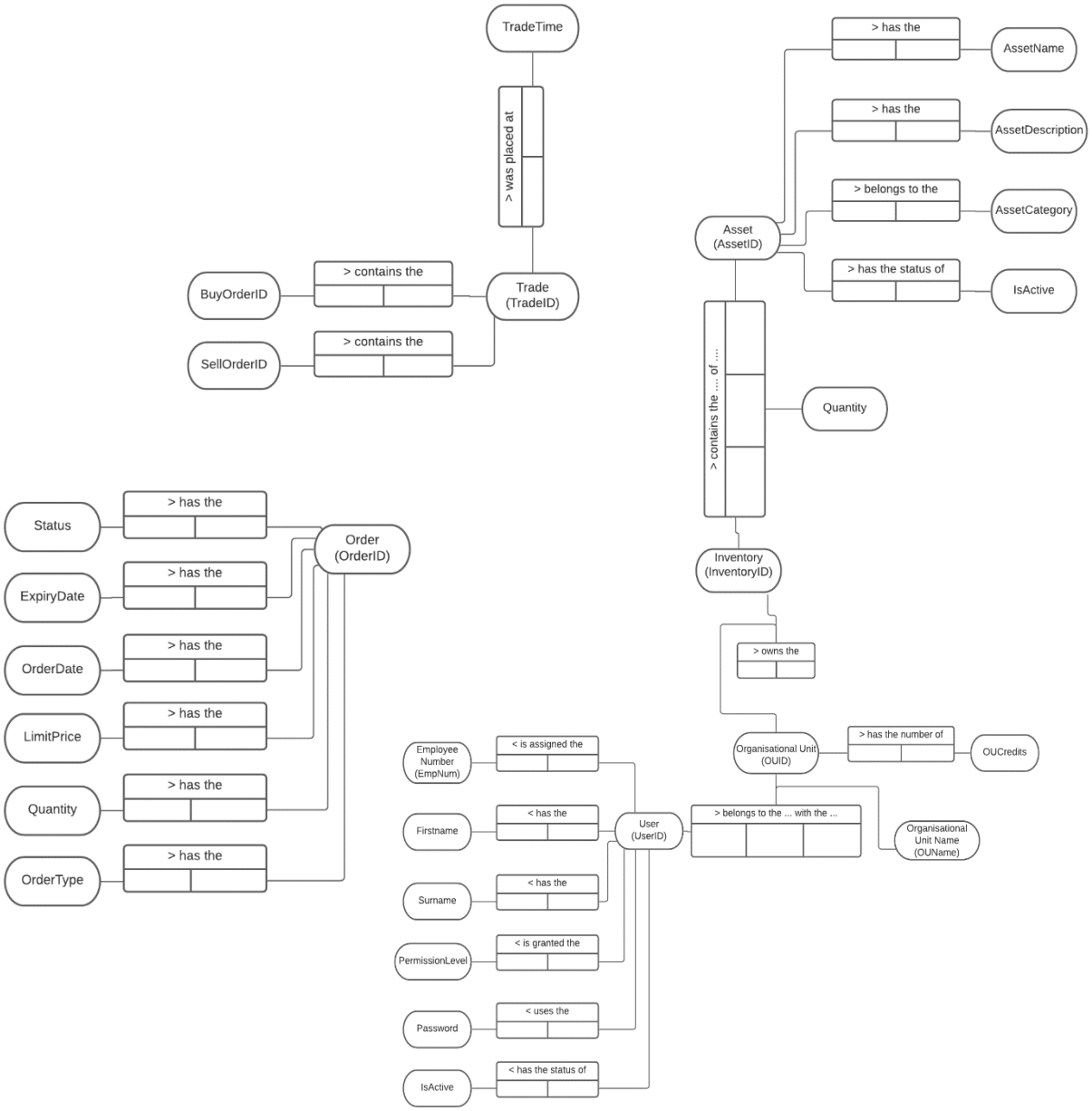


Figure 10: Database Object Role Model

Appendix 2: Database Script

```
-- -----------------------------------------------------------------
--  DATABASE cab302db
-- -----------------------------------------------------------------

CREATE DATABASE cab302db
   CHARACTER SET `latin1`
   COLLATE `latin1_swedish_ci`;
```

```
-- -----------------------------------------------------------------
--  TABLE assetcategories
-- -----------------------------------------------------------------

CREATE TABLE cab302db.assetcategories
(
   `AssetCatID`        INT(20) NOT NULL COMMENT 'Unique Asset Category ID',
   `AssetCatName`      VARCHAR(30)
                         CHARACTER SET latin1
                         COLLATE latin1_swedish_ci
                         NOT NULL
                         COMMENT 'Asset Category name',
   `AssetCatActive`    TINYINT(1)
                         NOT NULL
                         DEFAULT 1
                         COMMENT 'Is the Asset Category currently active',
   PRIMARY KEY(`AssetCatID`)
)
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------

--  TABLE assets

-- -----------------------------------------------------------------


CREATE TABLE cab302db.assets

(

   `AssetID`             INT(20) NOT NULL COMMENT 'Unique Asset ID',

   `AssetName`           VARCHAR(30)

                           CHARACTER SET latin1

                           COLLATE latin1_swedish_ci

                           NOT NULL

                           COMMENT 'Asset name',

   `AssetDescription`    VARCHAR(100)

                           CHARACTER SET latin1

                           COLLATE latin1_swedish_ci

                           NOT NULL

                           COMMENT 'Further information about the asset',

   `AssetCategory`       INT(20) NOT NULL COMMENT 'Asset category',

   `AssetActive`         TINYINT(1)

                           NOT NULL

                           DEFAULT 1

                           COMMENT 'Is the Asset currently active',

   PRIMARY KEY(`AssetID`)

)

ENGINE MYISAM

COLLATE 'latin1_swedish_ci'

ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------
--  TABLE inventories
-- -----------------------------------------------------------------

CREATE TABLE cab302db.inventories
(
   `InventoryID`        INT(20) NOT NULL COMMENT 'Unique Inventory ID',
   `AssetID`            INT(20) NOT NULL COMMENT 'Asset ID',
   `OUID`               INT(20) NOT NULL COMMENT 'OU ID',
   `Quantity`           INT(20)
                          NOT NULL
                          COMMENT 'Quantity of particular asset held by OU',
   `InventoryActive`    TINYINT(1)
                          NOT NULL
                          DEFAULT 1
                          COMMENT 'Is the Inventory currently active',
   PRIMARY KEY(`InventoryID`)
)
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------
--  TABLE orders
-- -----------------------------------------------------------------


CREATE TABLE cab302db.orders
(
   `OrderID`        INT(20) NOT NULL COMMENT 'Unique Order ID',
   `OrderType`      TINYINT(1)
                      NOT NULL
                      COMMENT 'Buy(true) or Sell(false) order',
   `UserID`         INT(20) NOT NULL COMMENT 'User ID',
   `AssetID`        INT(20) NOT NULL COMMENT 'Asset ID',
   `OUID`           INT(20) NOT NULL COMMENT 'OU ID',
   `Quantity`       INT(20)
                      NOT NULL
                      COMMENT 'Quantity of particular asset to be bought/sold',
   `LimitPrice`     INT(20)
                      NOT NULL
                      COMMENT 'Limit price for an individual asset',
   `OrderDate`      DATETIME(0)
                      NOT NULL
                      DEFAULT CURRENT_TIMESTAMP()
                      COMMENT 'The Timestamp for when the order was placed',
   `OrderStatus`    INT(1) NOT NULL COMMENT 'The current status of the order',
   `ExpiryDate`     DATETIME(0)
                      NOT NULL
                      COMMENT 'Timestamp for when the order will expire if not filled.',
   PRIMARY KEY(`OrderID`)
)
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------

--  TABLE orderstatus

-- -----------------------------------------------------------------


CREATE TABLE cab302db.orderstatus

(

   `OrderStatusID`      INT(20) NOT NULL COMMENT 'Unique Order Status ID',

   `OrderStatusDesc`    VARCHAR(20)

                          CHARACTER SET latin1

                          COLLATE latin1_swedish_ci

                          NOT NULL

                          COMMENT 'Description for the order status',

   PRIMARY KEY(`OrderStatusID`)

)

ENGINE MYISAM

COLLATE 'latin1_swedish_ci'

ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------
--  TABLE ous
-- -----------------------------------------------------------------


CREATE TABLE cab302db.ous
(
   `OUID`          INT(20) NOT NULL COMMENT 'Unique Organisational Unit ID',
   `OUName`        VARCHAR(30)
                     CHARACTER SET latin1
                     COLLATE latin1_swedish_ci
                     NOT NULL
                     COMMENT 'Organisational Unit name',
   `OUManager`     INT(30)
                     NOT NULL
                     COMMENT 'UserID for Organisational Unit Manager',
   `Credits`       INT(30)
                     NOT NULL
                     COMMENT 'Number of credits held by the Organisational Unit',
   `OUActive`      TINYINT(1)
                     NOT NULL
                     DEFAULT 1
                     COMMENT 'Is the Organisational Unit currently active',
   PRIMARY KEY(`OUID`)
)
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------
--  TABLE permissions
-- -----------------------------------------------------------------


CREATE TABLE cab302db.permissions
(
   `PermissionsID`            INT(20) NOT NULL COMMENT 'Unique Permissions Level ID',
   `BuyPermissions`           TINYINT(1)
                                NOT NULL
                                DEFAULT 0
                                COMMENT 'Does user have permissions to buy',
   `SellPermissions`          TINYINT(1)
                                NOT NULL
                                DEFAULT 0
                                COMMENT 'Does user have permissions to sell',
   `ApprovePermissions`       TINYINT(1)
                                NOT NULL
                                DEFAULT 0
                                COMMENT 'Does user have permissions to approve orders',
   `EditUserPermissions`      TINYINT(1)
                                NOT NULL
                                DEFAULT 0
                                COMMENT 'Does user have permissions to edit users',
   `CreateUserPermissions`    TINYINT(1)
                                NOT NULL
                                DEFAULT 0
                                COMMENT 'Does user have permissions to create users',
   PRIMARY KEY(`PermissionsID`)
)
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------
--  TABLE trades
-- -----------------------------------------------------------------


CREATE TABLE cab302db.trades
(
   `TradeID`        INT(20) NOT NULL COMMENT 'Unique Trade ID',
   `BuyOrderID`     INT(20) NOT NULL COMMENT 'Buy Order ID',
   `SellOrderID`    INT(20) NOT NULL COMMENT 'Sell Order ID',
   `AssetID`        INT(20) NOT NULL COMMENT 'Asset ID',
   `TradeTime`      DATETIME(0)
                      NULL
                      DEFAULT NULL
                      COMMENT 'Timestamp for when trade was completed',
   PRIMARY KEY(`TradeID`)
)
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

```
-- -----------------------------------------------------------------
--  TABLE users
-- -----------------------------------------------------------------


CREATE TABLE cab302db.users
(
   `UserID`              INT(20) NOT NULL COMMENT 'Unique user ID Employee ID',
   `FirstName`           VARCHAR(30)
                           CHARACTER SET latin1
                           COLLATE latin1_swedish_ci
                           NOT NULL
                           COMMENT 'Users first name',
   `Surname`             VARCHAR(30)
                           CHARACTER SET latin1
                           COLLATE latin1_swedish_ci
                           NOT NULL
                           COMMENT 'Users surname',
   `Email`               VARCHAR(50)
                           CHARACTER SET latin1
                           COLLATE latin1_swedish_ci
                           NOT NULL
                           COMMENT 'Users email address',
   `UsersOU`             INT(20) NOT NULL COMMENT 'Users Organisational Unit',
   `PermissionLevel`     INT(20) NOT NULL DEFAULT 0 COMMENT 'Permission Level',
   `Password`            VARCHAR(20)
                           CHARACTER SET latin1
                           COLLATE latin1_swedish_ci
                           NOT NULL
                           COMMENT 'Users Password',
   `UserIsActive`        TINYINT(1)
                           NOT NULL
                           DEFAULT 1
                           COMMENT 'Is the user active',
   PRIMARY KEY(`UserID`)
)
```

```
ENGINE MYISAM
COLLATE 'latin1_swedish_ci'
ROW_FORMAT DEFAULT;
```

7 - Bibliography

1. MariaDB. 2014. "Why MariaDB? Advantages over MySQL | MariaDB." Last modified October 29, 2014. https://mariadb.com/resources/blog/why-should-you-migrate-from-mysql-to-mariadb/#:~:text=First%20and%20foremost%2C%20MariaDB%20offers,large%20organizations%20and%20corporate%20users..

2. Steam. “Community Market”. https://steamcommunity.com/market/

3. Wikipedia. “Technical Debt”. https://en.wikipedia.org/wiki/Technical_debt