Client Requirements

Server-side core functionality

Only the IT Administration team is authorised to add/edit the requirements in the table below. The data must all be stored in the server-side database.

Requirement	Descriptions	Priority
Admin users (default)	Always at least 1 default admin user (config file)	Must have
Create usernames (users)	As an IT admin I want to create new users with unique usernames	Must have
Generate passwords	As an IT admin I want to generate passwords for new users	Should have
Assign admin privileges	As an IT admin I want to specify/change each account's privileges	Should have
Assign user to unit	As an IT admin I want to assign users to a unit	Must have
Credit system	As an IT admin I want to add the budget assigned to a unit	Must have
Create new asset types	As an IT admin I want to add any asset type	Should have
Create organisational unit	As an IT admin I want to assign asset types, quantity & budget	Must have
Store outstanding trade info	System stores BUY/SELL, unit, asset, quantity, price & date	Must have
Store executed trade info	System permanently stores the outstanding data	Must have
Hash passwords	System hashes passwords with a SALT	Should have
Adjust unit's credits	As an IT admin I want to adjust a unit's balance	Should have
Adjust unit's asset quantity	As an IT admin I want to adjust a unit's asset's quantity	Should have
Update unit's assets	As an IT admin I want to add/remove assets from units	Should have
Read config file	System reads config file to get Client's currently connected port	Could have
Database	All data stored in one place for easy back-up (SQLite)	Must have

Briefly recapping the server-side core functionality table above.

- **Usernames** must be unique
- **Passwords** should be unique, complex & hashed with a SALT
- **Account type** is either IT admin v user (access level)
- **Credit system** is our mini internal economy with a new currency
- **Organisational units** are like special accounts that hold the sensitive information (current budget, current assets, quantity etc). This information can be ACCESSED by users assigned to the unit (changing the values through BUY/SELL trades). Only the IT Administration team can directly alter this sensitive information.
- **Asset types**. The full range of assets is not known in advanced. Therefore, the system needs to be flexible to universally accept any future assets.
- **Outstanding trade info** needs to temporarily be stored as the data is crucial in order for us to match and execute successful trades. This data also temporarily decreases a unit's budget/asset quantity so they do not oversell/overspend. This data can also be used to generate the CURRENT average price so that user's can make a better judgement on orders. The reason this info is temporary is because users can cancel trades at any time and the current average price is always fluctuating.
- **Executed trade info** should be stored permanently if the trade was executed. It is best practice to keep this data on record in case anything goes wrong (a receipt of sorts). This info also allows us to generate graphs of an asset's historical trade value (stock-like graph).
- **Hash passwords** should be done automatically when sending passwords over the network & storing in database.
- **Read config file** would allow the IT admins to easily move the server around. By only modifying a config file, the Server can easily retrieve the Client's currently connected port number.

Client-side core functionality

Requirements that are needed for the Client-side to be functional.

Requirement	Description	Priority
Set SELL order	As a user I want to sell assets by setting quantity & price	Must have
Set BUY order	As a user I want to buy assets by setting quantity & price	Must have
Cancel an order	As a user I want to cancel outstanding orders	Should have
BUY/SELL restrictions	Client checks that quantities match & BUY price >= SELL price	Should have
Frequent checks	Client frequently checks to match BUY/SELL orders	Must have
Executing trades	Client permanently stores trade info & updates units' credit/asset quantity changes	Must have
Request current orders	As a user I want to view my unit's outstanding orders	Should have
Request available quantity	As a user I want to view my unit's assets and quantities	Must have
Request available credits	As a user I want to view my unit's available credits	Must have

Request average BUY/SELL	As a user I want to view an asset's current average price	Should have
Graphical User Interface	A more user friendly UI instead of the terminal	Could have
Generate visual graph	A (stock-like) graph showing an asset's value fluctuating	Could have
Password Self Service	As a user I want to be able to change my own passwords	Could have
Trade executed alert	Push an alert to (all?) members of that organisational unit	Could have
Client to read config file	Changes the Client's server IP address and port to connect to	Could have

Briefly recapping the client-side core functionality table above.

- **SELL orders** are successful when available quantity >= trade quantity. Temporarily decrease that asset's quantity for the unit.
- **BUY orders** are successful when available credits >= trade credit. Temporarily decrease that unit's credits.
- **CANCEL orders**. If the trade is still outstanding, allow users to cancel the trade. This deletes all trade info that was stored temporarily and increases the unit's asset quantities and credits using the trade info. User must be able to request to see their unit's current outstanding trade orders (as the GUI is not crucial).
- **BUY/SELL restrictions** prevents outstanding trades from executing. No unit will pay more or sell for less than they specified. As long as the BUY price is higher than the asset's SELL price AND quantities match, Then the trade can be executed at the LOWEST price (aka the SELL price).
- **Frequent checks** are made by the client in an attempt to match outstanding BUY/SELL orders and execute if requirements are met.
- **Executed trades** will permanently store the trade info & update a unit's credits/asset quantity.
- **Requesting current orders, assets & credits** will be done at the user's request. Requesting current orders pulls the outstanding trade info while assets and credits are pulled from their unit's account.
- **Requesting average BUY/SELL price** is done per asset. The client will use the (temporary) outstanding trade data to generate the current average price.
- **A user-friendly GUI** would "be nice to have", but the system can be completely functional without it. This means that users must be able to request the average prices, their available assets' quantities, and credits from the command line.
- **Stock-like graph** for each asset shows its historical value. Historical graphs of time (x-axis) and price (y-axis) using data from executed trades.
- **Password Self Service** is a could have, allowing users to change their passwords WITHOUT input from the IT Admin team.
- **Trade successfully executed alert** is a simple gesture that would notify members of a particular organisational unit that a change has been made (credits and an asset's quantity).
- **Read config file** would be modified by the IT Administration team. Allowing the Client to use a new server IP address and connect to a new port. The same config file should be read by both the Server and Client.

Functionality

The application must have the following items implemented for basic functionality:

- Organisational Units
- Credit system
- Place orders
- Reconcile orders
- Login with unique credentials (different accounts)
- Assets
- Admin User
- Ability to create new users/units/assets
- Assign assets/users/units to eachother

For extended functionality the application should have:

- GUI
- Trade history
- Current offers for a given asset (Buy/Sell Average Price)
- Historical Trades graph

SERVER SIDE

The application and associated Databases are to be stored on a singular Client-Server. The server will contain 5 Database tables:

- User Accounts
- Organisational Units
- Asset types
- Current orders
- Historical orders

The databases are to be in one of the following database formats:

- MariaDB
- PostgreSQL
- SQLite3

The **User Information** database is to store the following Individual User information:

Username	Hashed Password	SALT	Account privileges (User/Administrator)	Organisational Unit

The **Organisational Unit Information** database is to store the following Organisation information:

Organisation Name	Credit balance	Holdings (Assets)	Quantity of individual Assets	Organisational Unit

The **Asset Information** database is to store the following asset information:

Asset ID	Asset description

The **Current orders Information** database is to store the following current orders information:

Order ID	Order type	Associated unit	Asset ID	Asset quantity	Order price	Date placed

The **Historical orders Information** database is to store the following historical orders information:

Order ID	Order type	Associated unit	Asset ID	Asset quantity	Order price	Date placed

CLIENT-SIDE

Organisational units

Organisational Units are equivalent to departments or companies and must have the following feature:

- Credit Balance
- Assets
- Individual Asset Quantities
- Orders
- Members

Credits

Credits are used as a common currency between Organisational Units. The quantity of credits should be stored per individual Organisational Unit.

Assumption: Credits should be a whole number, no decimal places, to allow for easy storing and manipulation. Client & user feedback on how credits are used should be used to form this data type.

Assets

Assets are the items that are traded between organisations. There is no limit to the number of assets that an organisation can have or the quantity of a given asset.

Orders

Orders are used to track the trades between organisational units. There is no limit to the number of orders that can be listed by either a user or organisation. There are to be two (2) types of orders, Buy orders & Sell orders.

Buy Orders are a request from one organisation to another to *Buy* a *Quantity* of an *Asset* for a certain number of *Credits*

Sell Orders are a request from one organisation to another to *Sell* a *Quantity* of an *Asset* for a certain number of *Credits*

The system will periodically check for outstanding trades (buy and sell offers concerning the same asset). The system will attempt to reconcile these trades. For this to succeed, the sell price must be equal to or lower than the buy price, and there must be enough assets for sale to satisfy the buy order. If the buy price is lower than the sell price, nothing will happen.

If the quantities match, both orders will be automatically removed when the trade is reconciles. If the sell order is offering a greater number of the asset than the buy order is requesting, the sell order will not be removed but instead have its quantity reduced.

If a buy order exceeds the total credits of the buyer's organisational unit, or a sell order exceeds the quantity of that asset possessed by the seller, the system must ***not*** process the order and should give some

feedback to the user to explain why the order was not processed.

Orders - Nice to Haves

Users should be able to view a list of current orders that have not yet been processed. Using this list users should be able to cancel an order.

A graph showing the historical trading data for the given asset. The graph should contain the following items:

- X Axis - Orders
- Y Axis - Price

Current offers including:

- Buy/Sell price
- Quantity

Individual Users

Individual users are the personnel that will be placing & requesting the trades. There is to be no limit to the number of users. Users will need to be able to complete the following functions:

- Placing Buy/Sell Orders
- View current organisational orders
- Cancelling orders
- View current offers for a given asset
- View historical trading information for a given asset

Each user will be part of an Organisational Unit. There can be no users without an associated Organisational Unit. Users will inherit their organisations credits. As such, the cost of trades will be subtracted from the organisations total, and the profit from sales will be added to the organisations total.

Each user will have their own unique username and password. Users should be allowed to change their own passwords without external assistance.

Administrators

There should also be a second user type for administrators. Administrators are to have the ability to:

- Edit the number of credits that an organisation has (Confirm if this is any or if it is only their specific organisation)
- Edit the Quantity of a given asset that an organisation has
- Add a new asset to an organisation
- Add new users
    - Assign Usernames & Passwords
    - Assign Organisational Units to Users
- Create new Administrator Users with the same privileges

GUI Requirements

- Users to receive a message upon fulfilment of an order.
- Current orders graph
- Historical trades Graph

Additional Documentation

Configuration File

The configuration file for the application is to contain the IP address and Port Number of the server that the application is hosted on. Additionally, the configuration file for the Server is to also contain these details. You may also configure the schema to use a fake database or the real database. You must specify the default admin user (username & password).