# Implemented Functionality Report

## Implemented Functionality Table

| **Function** | **C#** | **Java** |
|---|---|---|
| Client establishes a connection with the server | Yes | Yes |
| Client is assigned a unique ID when joining the market | Yes | Yes |
| Client displays up-to-date information about the market state | Yes | Yes |
| Client allows passing the stock to another player | Yes | Yes |
| Server manages multiple client connections | Yes | Yes |
| Server accepts new connections while traders are exchanging stock among themselves | Yes | Yes |
| Server correctly handles clients leaving the market | Yes | Yes |
| Client is compatible with the server in the other language | Yes | Yes |
| **Additional tasks:** | | |
| Client GUI | No | No |
| Server GUI | No | No |
| Server restarts are correctly implemented | No | No |
| Unit tests | No | No |

## Protocol

- The application is a client-server application. This means that the client initiates a connection to the server, which is created once the server accepts the client's connection.
- The application uses a non-binary, text-based protocol; meaning that the bit stream between processes is organised using textual message as opposed to a compact sequence of bits.

- All exchanges of data are initiated by messages from the client. The server never initiates any exchange of data with the client. Once a client requests data, the server will immediately respond to each request.
- IDs are generated incrementally and do not require authentication from the user. These IDs are not reused and clients cannot reconnect using the same ID once they have disconnected.
- Each request from the client is sent on a single text line. This is done using the input and output streams attached to the socket the client has connected to the server with, and requests are sent in the form of commands.
- The server is able to respond with either a single line of text, or multiple lines of text in response to a single request from the client.
- In the case of exceptions and errors, the server responds by catching the error, and returning it to the client.

## Client Threads

The client only contains a single thread, which is used to communicate with the server once their connection to the server socket via a supplied port has been accepted. This thread communicates with the server via the input and output streams attached to the connected socket; allowing the process to receive command and return data.

This thread is created once the client process is started, and is terminated on the termination of the client process.

## Server Threads

The main server thread is used to run the method “RunServer()” once the server process has been started. This thread waits for new client connections, and then creates additional threads for each individual client. This thread is created once the server process has started, and is terminated on termination of the server process.

Additional threads running within the server process are used for the individual clients’ “ClientHandler” class. Each instance of this class is used to connect single client to the server using a supplied port; and allows for commands to be sent back and forth between the server and the client individually which allows the transfer and communication of data between processes. This is done by sending commands via the input and output streams for the clients socket; which can then be read and have the following method call or function related to the received command carried out.

Each thread is created within the “RunServer()” method once a connection with the client to the server socket has been accepted, and is terminated once the connection with the client process has been terminated.

## Project Review

In reflection I believe that my project went well. I have managed to get all of the required functionality working as it should do, with the correct data being returned to the client on request and the correct handling of the clients by the server. The client and server processes are also able to communicate between both C# and Java correctly; regardless of the language used in that process.

However I do know that I could have improved some areas of it. The main area I could have improved would be the automatic client updates once a change in the market occurs; whether it be a new trader connecting, a trader disconnecting, or a change of stockholder. I initially wanted to implement this as a command to the server, however I had difficulty with the output streams and commands would often get picked up accidentally by other methods.

Because of this, I resorted to using an if/else statement for the client in an infinite loop. As clients cannot use the command prompt to trade if they are not the stockholder, they will receive an update as soon as the state of the market changes. However, the market should not update if the trader is in the middle of a trade prompt. The trader has an option to update the market manually in this prompt if they wish to do so, before being prompted to trade again.

This works as it should do, and considering the difficulties I had with the input and output streams, I am happy with my solution. I would however in the future like to implement it a better way, as this would allow me to call the update on a separate thread; allowing the updating to occur regardless of the user command prompt and would make it easier to develop a client-side GUI as updates can be supplied independently to the functioning of said GUI.