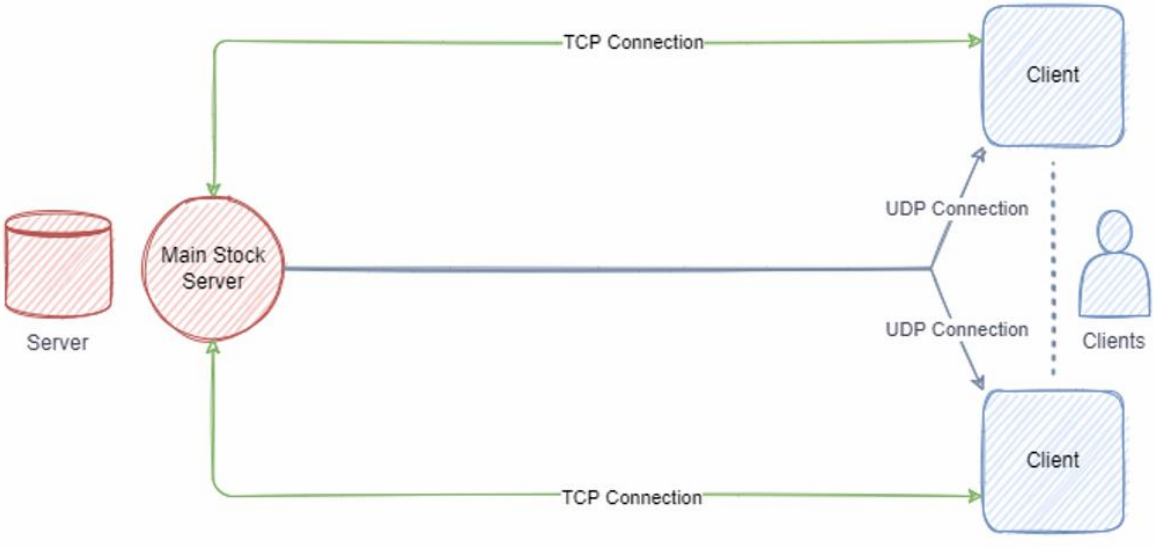
Lab 6:

Real – Time
Stock Market



- ***What is our project supposed to do?***

  Our project is a representation of a real-time stock market application.
  Clients will be able to register to an open server with a unique username and password and will then be assigned a unique stock portfolio that belongs to them.
  Clients that have already registered will be able to reconnect and continue with their portfolio state as long as the server wasn't shut down in the process.

  Every client will be able to buy and sell stocks with his money, and observe the best current offers and price per unit of each stock.
  Every transaction in a stock triggers a change in the stock price (price of unit per stock), a sell and a buy with a higher unit price than the current unit increases the stock price and vice versa.

  All Sell/Buy requests are a client-initiative and all stocking prices are sent in real-time from the server to the client.

- ***How will it manage several clients simultaneously?***

  Our Server will manage connections with several clients by using threads, where each thread will handle a TCP socket between the server and each client, this socket will be used for sending and receiving reliable data.
  updates will be sent to all clients using multicast over UDP, every client connected to the server will receive the multicast IP 239.0.0.1.

- ***What are the communication protocols between the client and the server?***

  Both TCP and UDP will be used for each client-server connection,
  We will use TCP's reliability to preform reliable transactions.
  By using TCP, we ensure QoS, each transaction will (eventually) be handled without any losses, every offer is added to the relevant array.
  We will use UDP to perform real-time updates using multicast, these updates include changing stock prices and best current prices for them.
  we will add a sequence number to the UDP packets in order to reliably keep the clients up to date with relevant new updates and not ones that were for some reason delayed (they will be dropped).

- ***What timeouts are part of this protocol?***

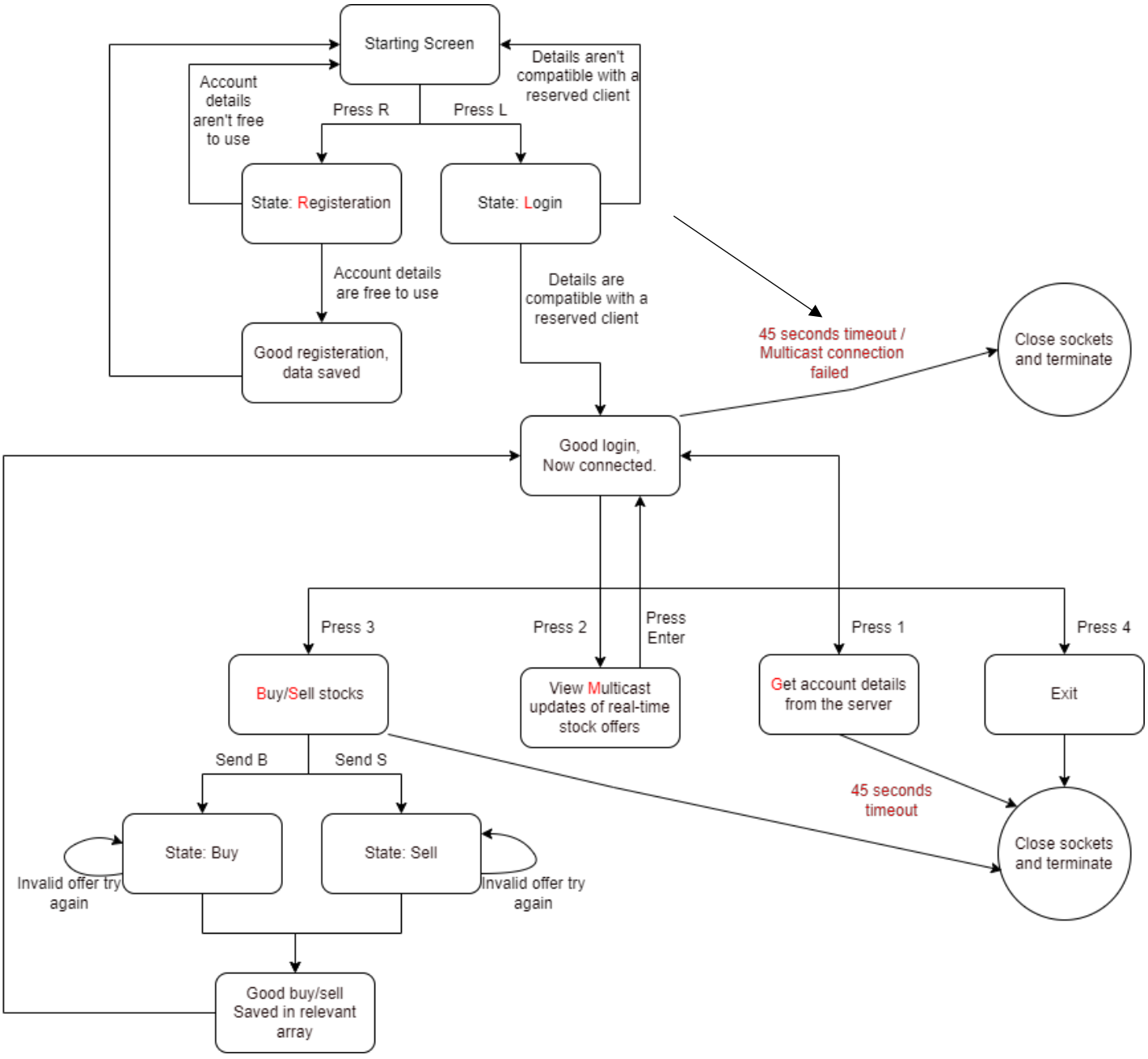
  Each client will be kicked out if idle for more that 45 seconds, disregarding the state of observing the multicast updates, at this case he can be idle for as long as he pleases, we use select function to measure the time between messages sent by the client, if no message was sent to the server after 45 seconds the client is dropped.

- ***Which tests do we need to run in order to test our program?***

  We will look at extreme cases such as:

  1. **Server crashing unexpectedly** – Once the server shuts down, by sampling the value returned from the recv function, we identify a server crash and close the sockets.
  2. **Client crashing unexpectedly** - Once a client shuts down unexpectedly, by sampling the value returned from the recv function, we identify the crash and close the client's sockets.
  3. **Out-of-order Updates** – By adding sequence numbers to the multicast updates we ensure that if a client receives an out-of-order update it is dropped and ignored.
  4. **Incompatible message types** – By checking the first letter in each control packet we ensure that the message type is compatible with our pre-defined agreements, if not, the socket is closed, and the specific client is dropped.
  5. **Client idle for too long** – as mentioned above, an idle-timeout is checked and handled.
  6. **Authentication** - Although this is more of a feature of our code, we can check and see that logging in with wrong details and trying to register with an already used username is not possible.
     In addition, if username is currently logged in then if another client tries to log in with the same username and password he is blocked from

Client-Side State Machine:



Server-Side State Machine: