**Proposal:**

Buying and selling are essential activities every person needs to do, while for most of the commonly bought and sold items, trading shares at companies isn't something everyone knows how to do, our aim is to develop a simple platform from which buyers can buy or sell shares at companies they want.

**Test Plan:**

The web application was stress-tested to ensure that at least 1000 users can connect at the same time (we could increase this number even more, but as the application is hosted on a server with limited resources, we didn’t), to do this stress-testing, we used DirBuster for sending requests to the server from multiple parallel threads.

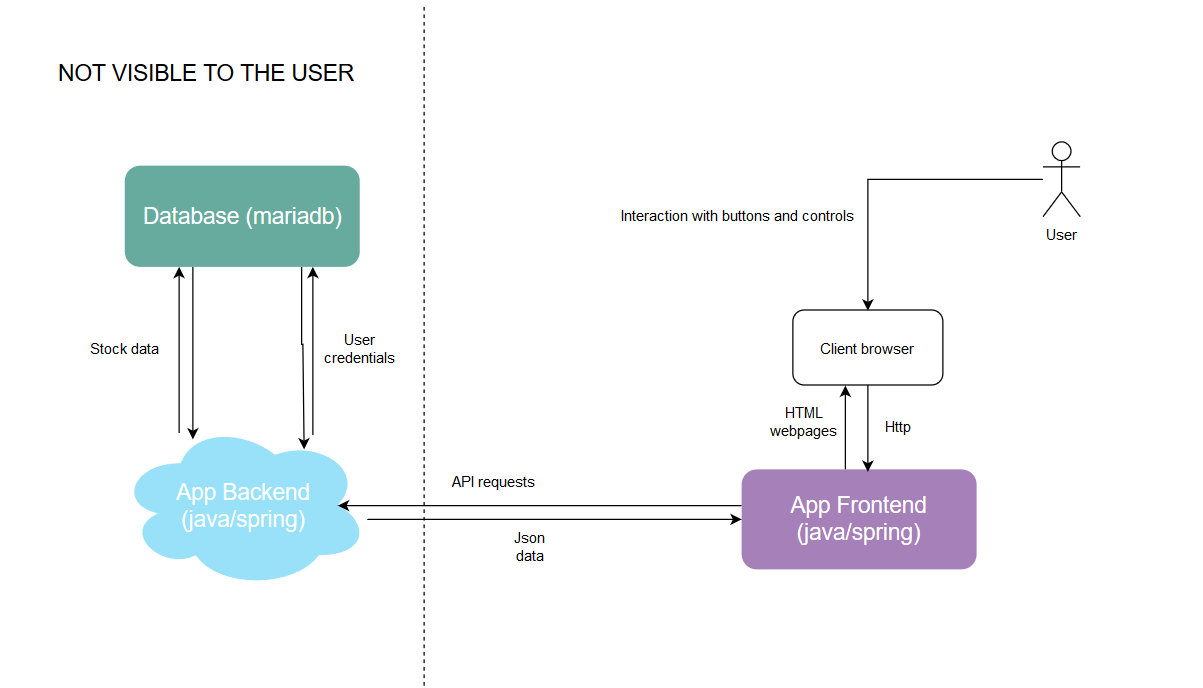
Testing individual components was simplified by the design of our solution, we were able to test the front-end independently of the back-end, for the front-end, we had a script acting as backend and returning placeholder data, on authentication requests, and on other database queries.

For the backend, we tested the API using postman, and curl (to send GET/POST requests and see the response).

When we merged the two together, we used Burp Suite, to ensure the front-end sends the requests our backend is expecting, we were able to identify and fix issues very quickly this way.

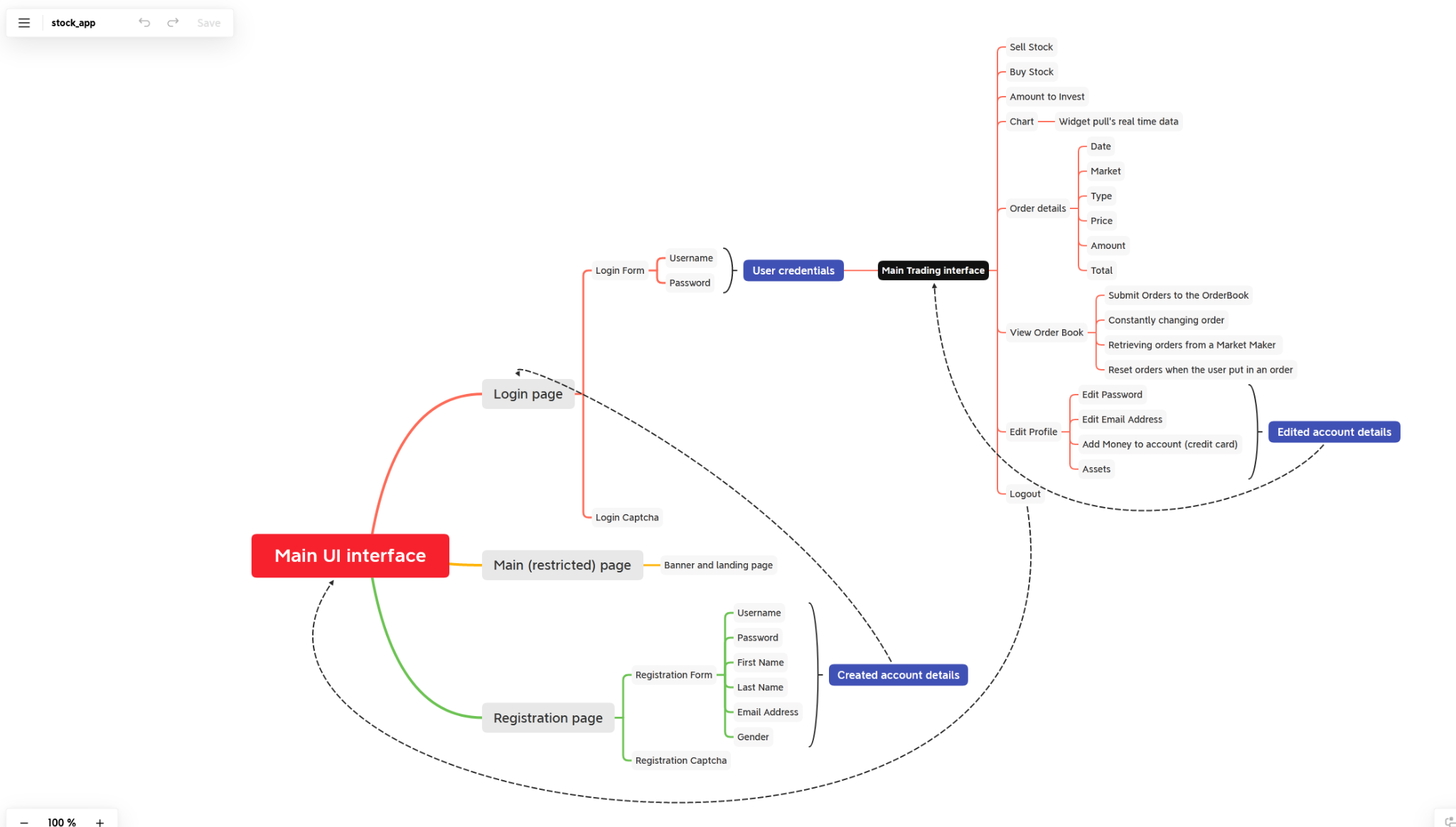
We did not implement unit tests mainly because the platform is not going to get more commits from different contributors, unit tests ensure that new contributions don’t break the logic.

**Project Diagram:**



High-level diagram showing the components of the application

**Node Graph:**



**Requirements / Specifications of the software environment:**

Docker and docker-compose (for deployment)

Inside the Docker containers:

Maven (for building the project, handles dependencies and allows building the project in a simple command)

Java 9 or greater (for running the jar file that results from Maven)

**Database:**

There is a single database named marketapp.

The database contains the following tables:

|  |
| --- |
| users(id integer auto\_increment, username varchar(20), password varchar(32), firstname varchar(20), lastname varchar(20), gendermale boolean, money float, primary key id) |

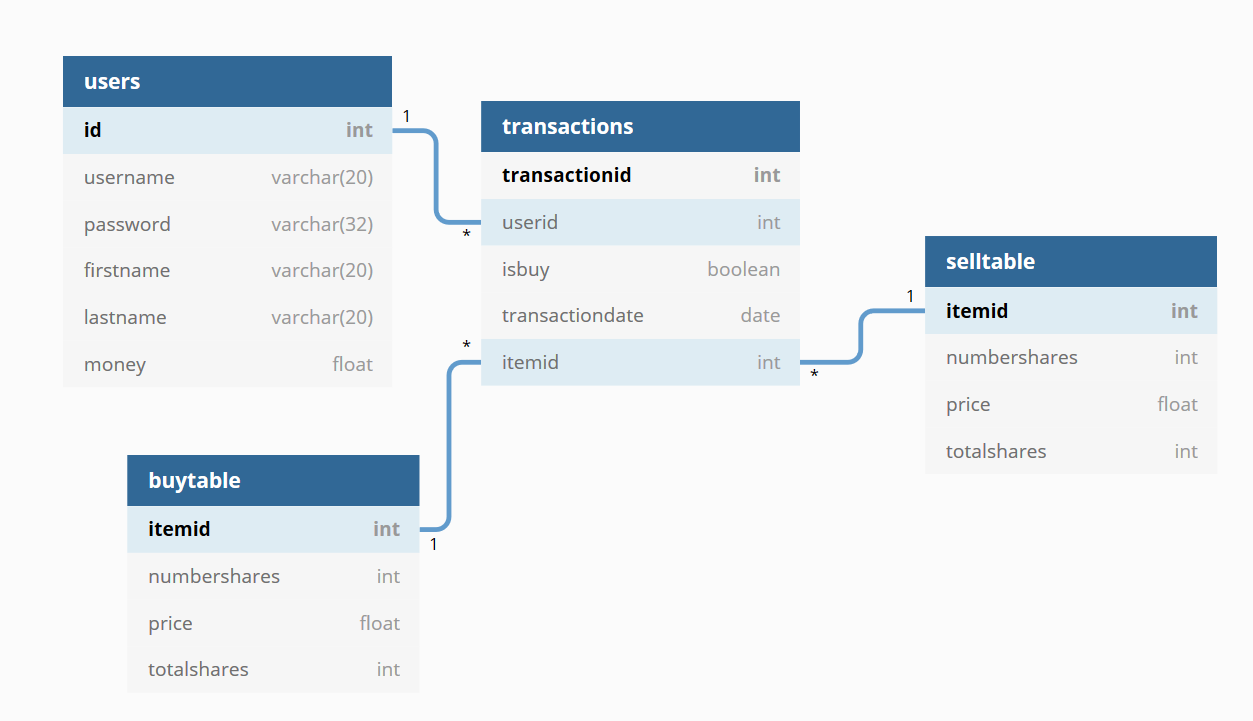
id is a numeric identifier of a user, password is the hashed password value (hashed with sha256).

|  |
| --- |
| buytable(itemid integer, numbershares integer, price float, totalshares integer)  selltable(itemid integer, numbershares integer, price float, totalshares integer) |

The tables containing informations on shares, with prices for buying and selling, itemid is the Identifier of the item, the other informations are exactly the informations displayed to users.

|  |
| --- |
| transactions(transactionid integer, userid integer, isbuy boolean, transactiondate date), userid references users.id, primary key transactionid |

This table contains transactions made, can be used for monitoring, easily rolling-back transactions if needed.



A diagram of the Entity-Relationship modeling of the database

**Features table:**

|  |  |
| --- | --- |
| Feature | Details |
| User registration and secure login. | Made in Java / Spring, secure against SQL injection and other common vulnerabilities. |
| Buying or selling from within the account | Made in Java / Spring, each buy or sell action saves a transaction in the database, and updates stock informations, transactions are saved in their own table, in case there is the need to roll-back one for fraud or whatever reason. |
| Getting stock informations in real-time | Uses XMLHttpRequests to fetch data from the API, in case another user buys or sells, the table gets updated. |
| Editing profile / adding money to the account | Editing profile informations like the password, simulating adding money to the account through a credit card. |

**Project Planning:**

Since the project is being conducted individually, it was hard to follow git best practices for teamwork, creating issues and pull-requests, and using different branches, in order to plan the project accordingly, I separated the App frontend and backend (made them two different repositories), and used docker for testing the deployment of the whole project with a single command, by using two projects, planning was easier, finding specific commits is faster, and keeping the project clean also.

**Conclusion:**

I really believe such a project would be useful, and while I wasn't able to accomplish every feature listed above (I'm still working on it), using real banking credentials, and integrating a proper user interface would have been great, unfortunately, I think that would require a team of developers, with people specializing in front-end dev, web design and so on.