CSC4402 Database Management

Aidan Eiler, Andrew Schacht, Ford Morris, James Power, Jonathan Morse, Joseph Curtis, Lydia Parsa, Nate Canterbury

Problem Statement

Not knowing how to invest your money can be devastating for the livelihoods of people. Learning how to invest with the money you have can be risky, and the consequences that follow can be life ending.

The goal of our project is to allow users to safely practice making investments without putting them into financial ruin. To do this, we create a stock market simulation with fluctuating prices. The user is able to input new stocks with starting values as well as a volatility value that can predict the magnitude of price fluctuations over time. The user is also able to create a portfolio in order to interact with the stock market they've created.

Methods

We plan on creating a website with a standard account login procedure. Once the user has logged into their account, they can examine the stocks they currently own and view the change in price of the stock live. Similar to real Online Brokerage Platforms (OBPs), users will be able to view the price changes of various stocks in real time. For the purpose of maintaining simplicity and a low cost project, the stock prices are pseudorandom.

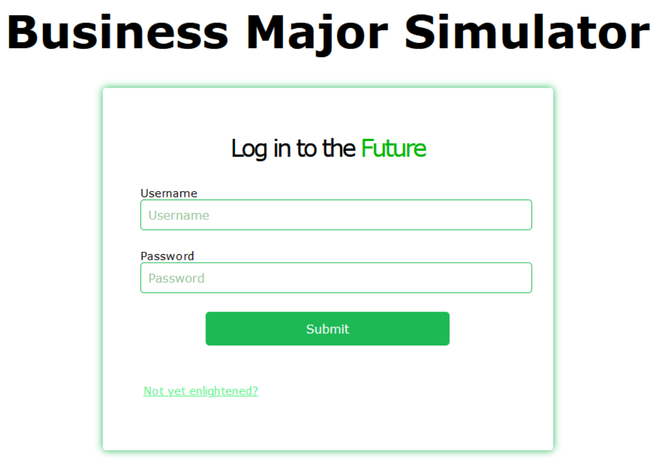
Experiments

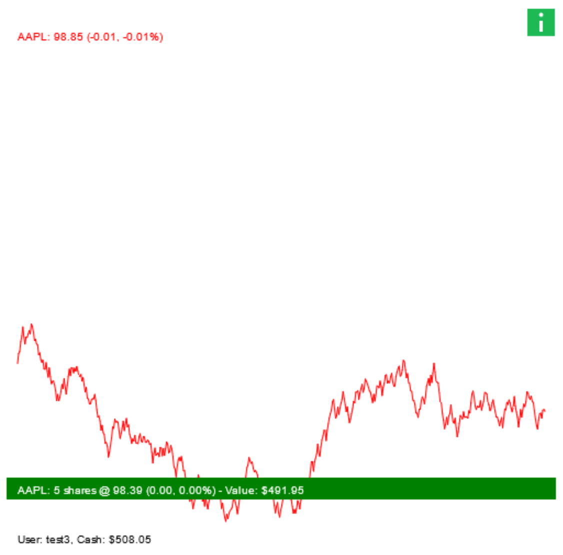
The project was built using JavaScript and the p5.js library, which made incorporating a front-end into the project much simpler. The generated webpage will prompt the user to login, then display the user's portfolio as well as a constantly updating line graph representing a stock's current price and fluctuation (volatility). The user's saved stocks will be stored in a corresponding SQL database, which is connected to a server that's being hosted by another machine, using a Node.js backend.

The database schema consists of the the user's account credentials, their account balance (in USD), and the stocks they own, which is indicated by the stock ticker, the number of shares they have, and when they bought the stock. If the stock has a stop-loss order, that is also provided in the user portfolio.

The stocks are also stored in the database, where the ticker and current price is stored. The volatility is also stored in the database, which is found by analysing the stock's quarterly price change.

Results



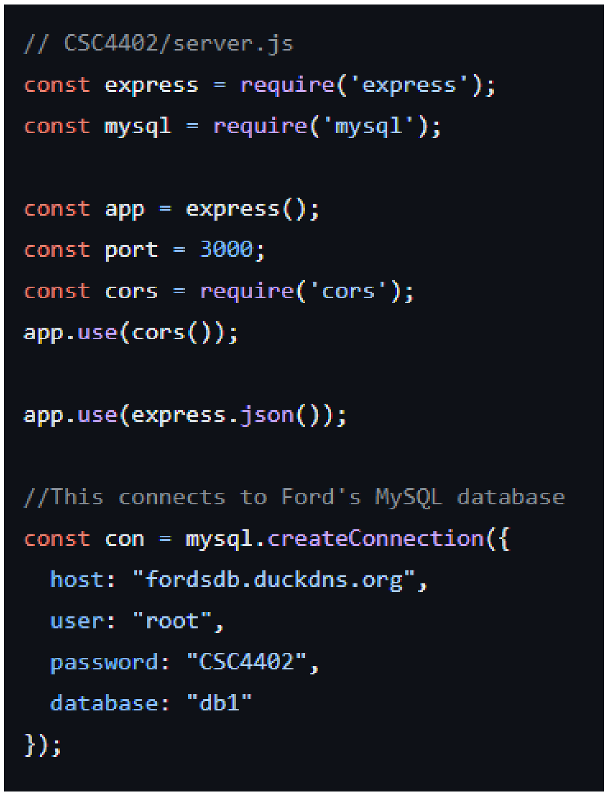


AAPL: 98.85 (-0.01, -0.01%)

AAPL: 5 shares @ 98.39 (0.00, 0.00%) - Value: $491.95

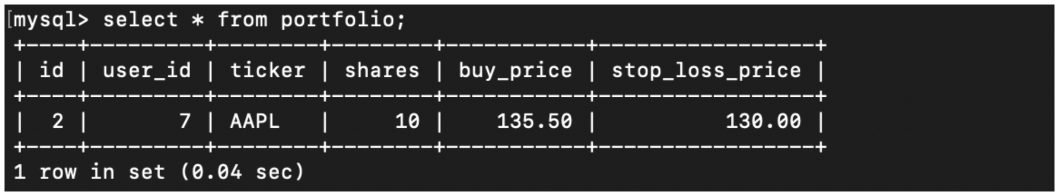
User: test3, Cash: $508.05

The login/register page (left) and the stock ticker (right).

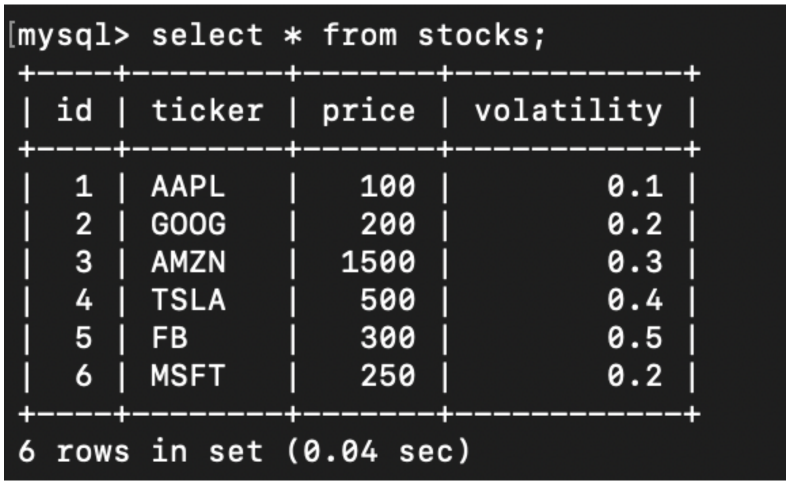


```
// CSC4402/server.js
const express = require('express');
const mysql = require('mysql');

const app = express();
const port = 3000;
const cors = require('cors');
app.use(cors());

app.use(express.json());

//This connects to Ford's MySQL database
const con = mysql.createConnection({
  host: "fordsdb.duckdns.org",
  user: "root",
  password: "CSC4402",
  database: "db1"
});
```

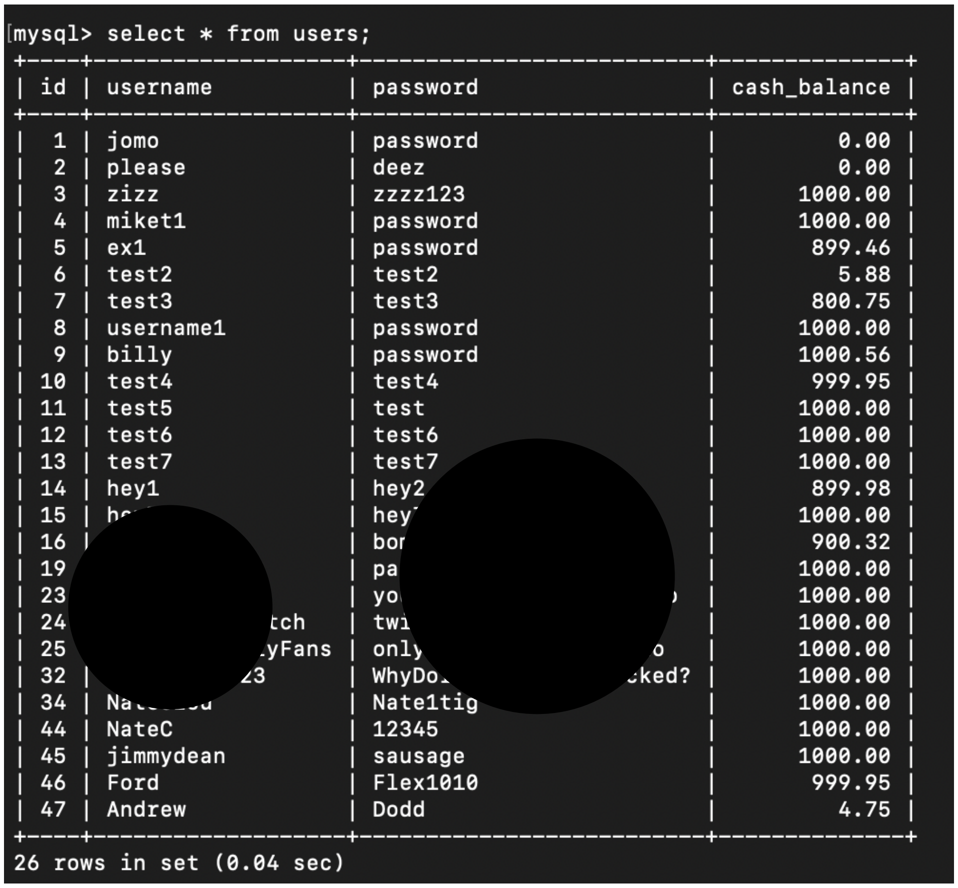
Connecting to our database.



```
[mysql> select * from portfolio;
+----+---------+--------+--------+-----------+-----------------+
| id | user_id | ticker | shares | buy_price | stop_loss_price |
+----+---------+--------+--------+-----------+-----------------+
|  2 |       7 | AAPL   |     10 |    135.50 |          130.00 |
+----+---------+--------+--------+-----------+-----------------+
1 row in set (0.04 sec)
```

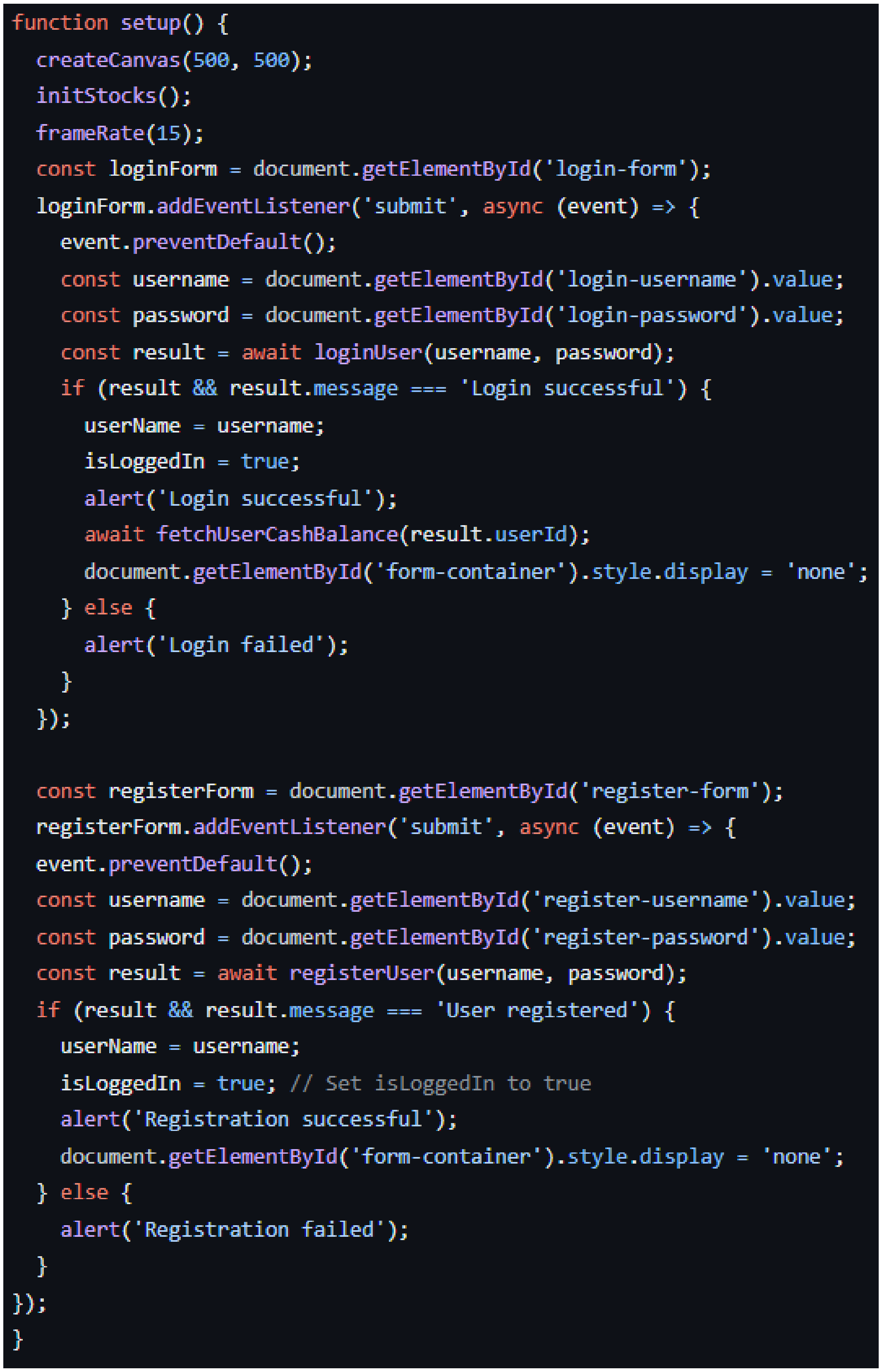


```
mysql> select * from stocks;
+----+--------+-------+------------+
| id | ticker | price | volatility |
+----+--------+-------+------------+
|  1 | AAPL   |   100 |        0.1 |
|  2 | GOOG   |   200 |        0.2 |
|  3 | AMZN   |  1500 |        0.3 |
|  4 | TSLA   |   500 |        0.4 |
|  5 | FB     |   300 |        0.5 |
|  6 | MSFT   |   250 |        0.2 |
+----+--------+-------+------------+
6 rows in set (0.04 sec)
```



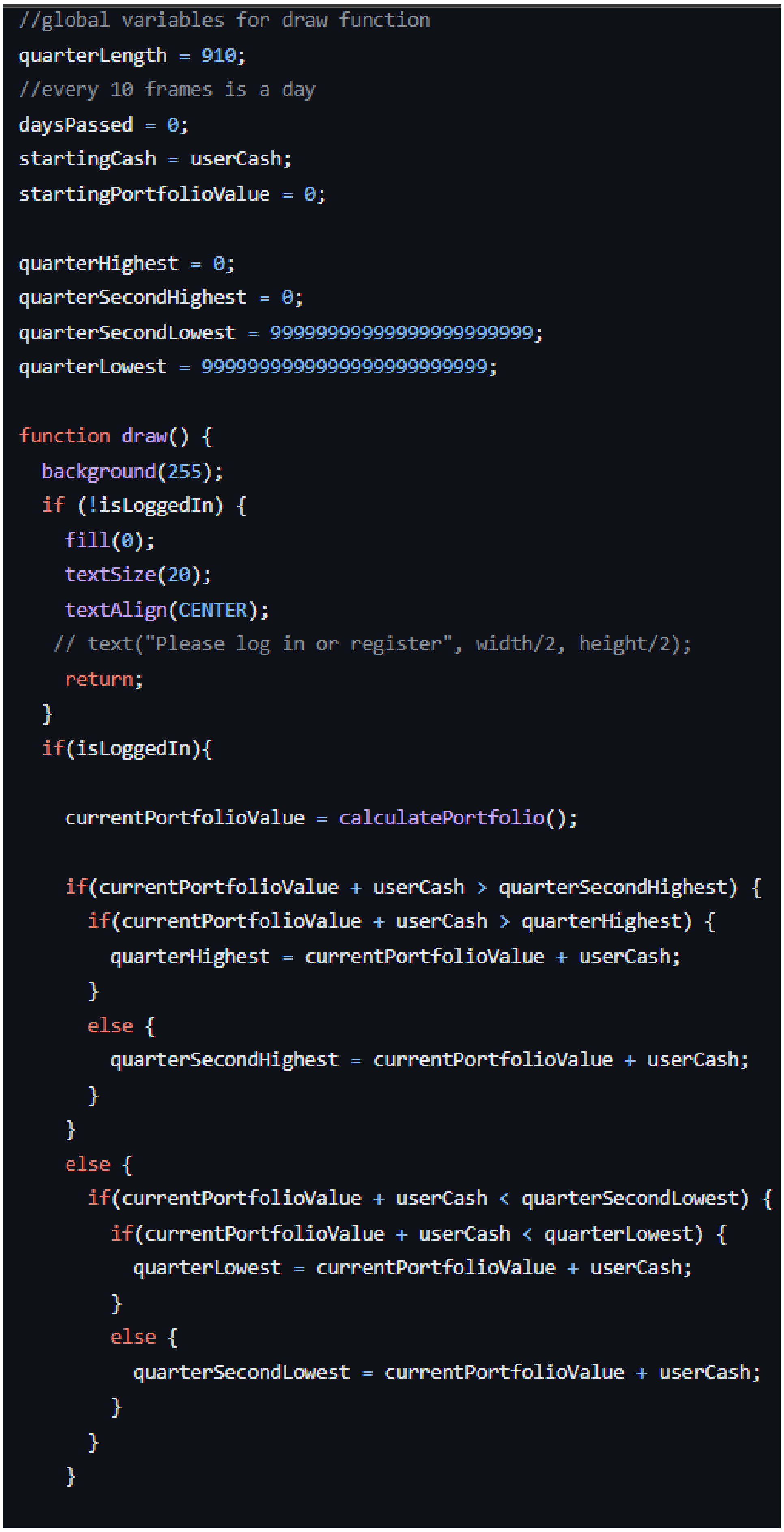
```
mysql> select * from users;
+----+-------------------+---------------------------+--------------+
| id | username          | password                  | cash_balance |
+----+-------------------+---------------------------+--------------+
|  1 | jomo              | password                  |         0.00 |
|  2 | please            | deez                      |         0.00 |
|  3 | zizz              | zzzz123                   |      1000.00 |
|  4 | miket1            | password                  |      1000.00 |
|  5 | ex1               | password                  |       899.46 |
|  6 | test2             | test2                     |         5.88 |
|  7 | test3             | test3                     |       800.75 |
|  8 | username1         | password                  |      1000.00 |
|  9 | billy             | password                  |      1000.56 |
| 10 | test4             | test4                     |       999.95 |
| 11 | test5             | test                      |      1000.00 |
| 12 | test6             | test6                     |      1000.00 |
| 13 | test7             | test7                     |      1000.00 |
| 14 | hey1              | hey2                      |       899.98 |
| 15 | he[redacted]      | hey[redacted]             |      1000.00 |
| 16 | [redacted]        | bor[redacted]             |       900.32 |
| 19 | [redacted]        | pa[redacted]              |      1000.00 |
| 23 | [redacted]        | yo[redacted]o             |      1000.00 |
| 24 | [redacted]tch     | twi[redacted]             |      1000.00 |
| 25 | [redacted]yFans   | only[redacted]o           |      1000.00 |
| 32 | [redacted]23      | WhyDo[redacted]cked?      |      1000.00 |
| 34 | Nat[redacted]d    | Nate1tig                  |      1000.00 |
| 44 | NateC             | 12345                     |      1000.00 |
| 45 | jimmydean         | sausage                   |      1000.00 |
| 46 | Ford              | Flex1010                  |       999.95 |
| 47 | Andrew            | Dodd                      |         4.75 |
+----+-------------------+---------------------------+--------------+
26 rows in set (0.04 sec)
```

Database schema includes stocks, user credentials and cash balance, and user portfolios.

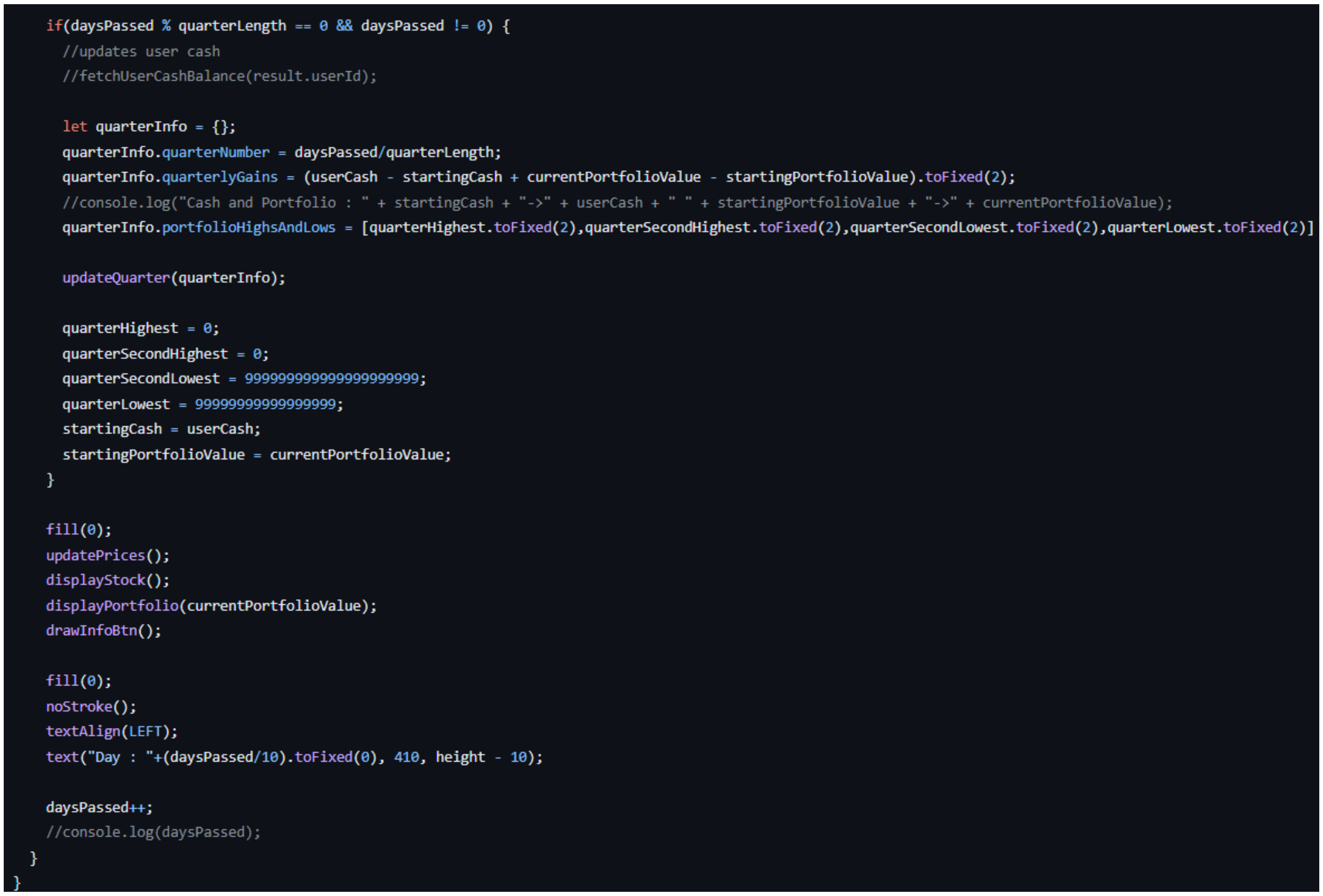


```javascript
function setup() {
  createCanvas(500, 500);
  initStocks();
  frameRate(15);
  const loginForm = document.getElementById('login-form');
  loginForm.addEventListener('submit', async (event) => {
    event.preventDefault();
    const username = document.getElementById('login-username').value;
    const password = document.getElementById('login-password').value;
    const result = await loginUser(username, password);
    if (result && result.message === 'Login successful') {
      userName = username;
      isLoggedIn = true;
      alert('Login successful');
      await fetchUserCashBalance(result.userId);
      document.getElementById('form-container').style.display = 'none';
    } else {
      alert('Login failed');
    }
  });

  const registerForm = document.getElementById('register-form');
  registerForm.addEventListener('submit', async (event) => {
  event.preventDefault();
  const username = document.getElementById('register-username').value;
  const password = document.getElementById('register-password').value;
  const result = await registerUser(username, password);
  if (result && result.message === 'User registered') {
    userName = username;
    isLoggedIn = true; // Set isLoggedIn to true
    alert('Registration successful');
    document.getElementById('form-container').style.display = 'none';
  } else {
    alert('Registration failed');
  }
});
}
```

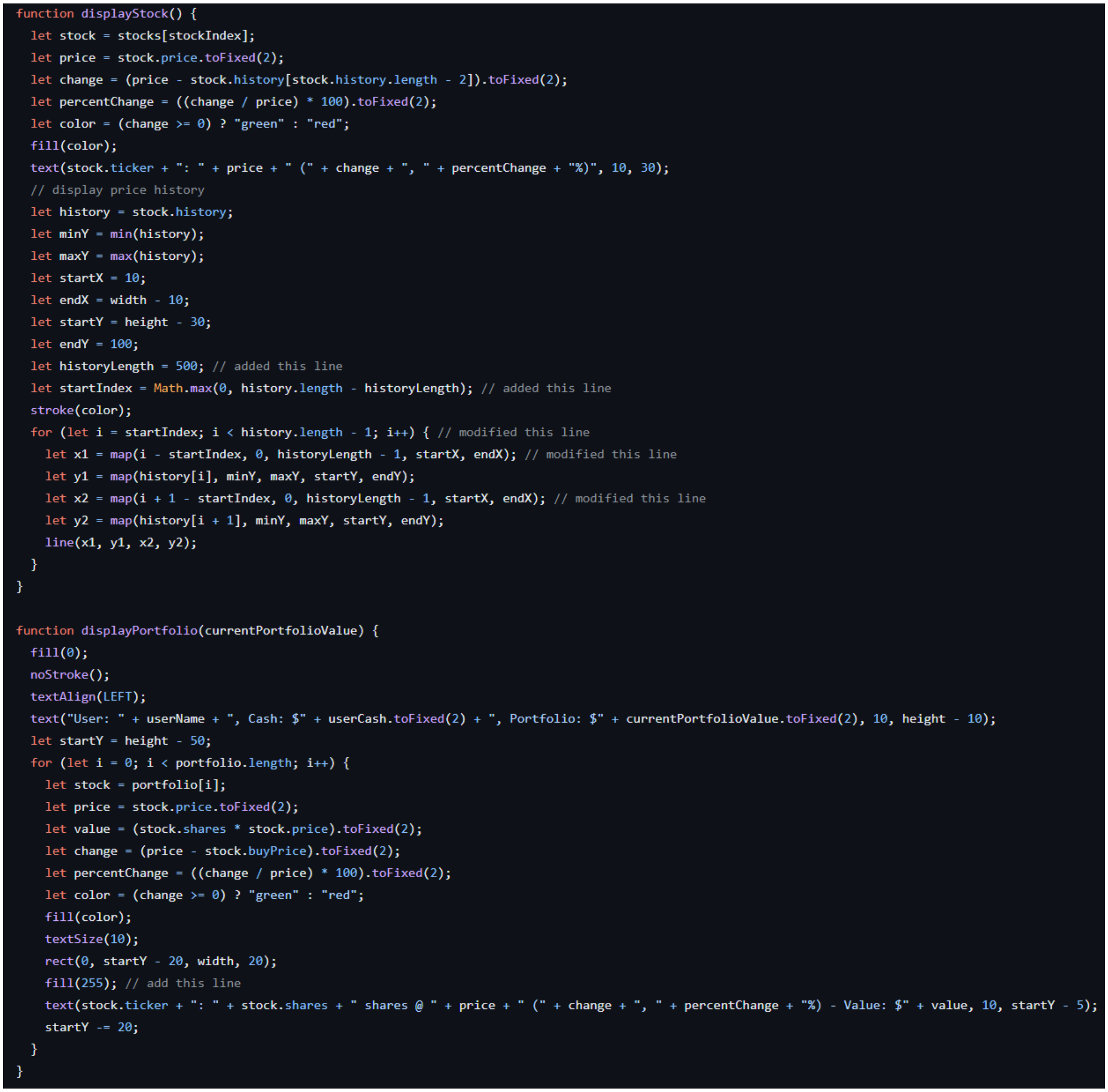
Frontend setup and login process (more on following pages)



```
//global variables for draw function
quarterLength = 910;
//every 10 frames is a day
daysPassed = 0;
startingCash = userCash;
startingPortfolioValue = 0;

quarterHighest = 0;
quarterSecondHighest = 0;
quarterSecondLowest = 99999999999999999999999;
quarterLowest = 9999999999999999999999999;

function draw() {
  background(255);
  if (!isLoggedIn) {
    fill(0);
    textSize(20);
    textAlign(CENTER);
   // text("Please log in or register", width/2, height/2);
    return;
  }
  if(isLoggedIn){

    currentPortfolioValue = calculatePortfolio();

    if(currentPortfolioValue + userCash > quarterSecondHighest) {
      if(currentPortfolioValue + userCash > quarterHighest) {
        quarterHighest = currentPortfolioValue + userCash;
      }
      else {
        quarterSecondHighest = currentPortfolioValue + userCash;
      }
    }
    else {
      if(currentPortfolioValue + userCash < quarterSecondLowest) {
        if(currentPortfolioValue + userCash < quarterLowest) {
          quarterLowest = currentPortfolioValue + userCash;
        }
        else {
          quarterSecondLowest = currentPortfolioValue + userCash;
        }
      }
    }
```

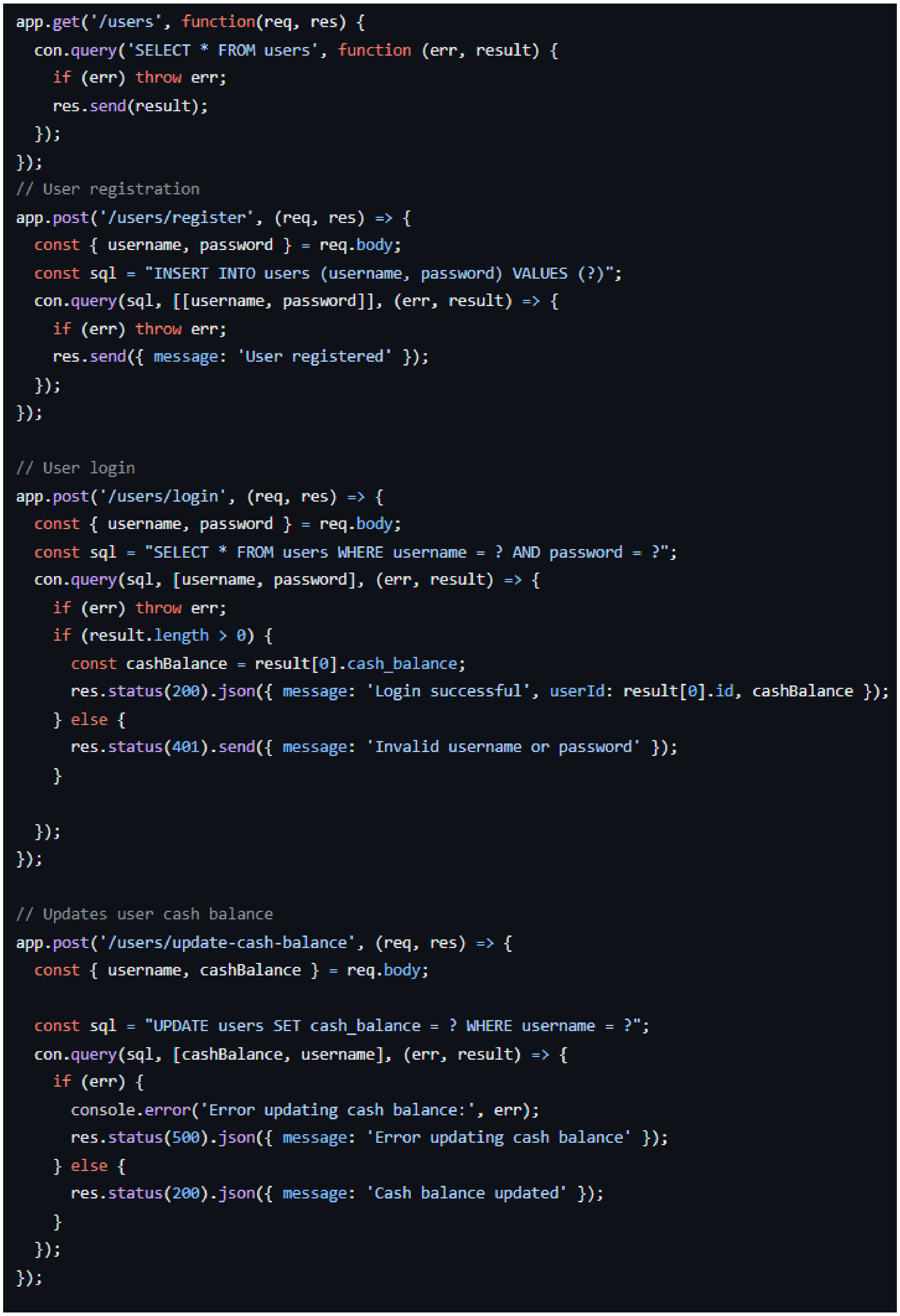


```
    if(daysPassed % quarterLength == 0 && daysPassed != 0) {
      //updates user cash
      //fetchUserCashBalance(result.userId);

      let quarterInfo = {};
      quarterInfo.quarterNumber = daysPassed/quarterLength;
      quarterInfo.quarterlyGains = (userCash - startingCash + currentPortfolioValue - startingPortfolioValue).toFixed(2);
      //console.log("Cash and Portfolio : " + startingCash + "->" + userCash + " " + startingPortfolioValue + "->" + currentPortfolioValue);
      quarterInfo.portfolioHighsAndLows = [quarterHighest.toFixed(2),quarterSecondHighest.toFixed(2),quarterSecondLowest.toFixed(2),quarterLowest.toFixed(2)]

      updateQuarter(quarterInfo);

      quarterHighest = 0;
      quarterSecondHighest = 0;
      quarterSecondLowest = 9999999999999999999;
      quarterLowest = 99999999999999999;
      startingCash = userCash;
      startingPortfolioValue = currentPortfolioValue;
    }

    fill(0);
    updatePrices();
    displayStock();
    displayPortfolio(currentPortfolioValue);
    drawInfoBtn();

    fill(0);
    noStroke();
    textAlign(LEFT);
    text("Day : "+(daysPassed/10).toFixed(0), 410, height - 10);

    daysPassed++;
    //console.log(daysPassed);
  }
}
```



```javascript
function displayStock() {
  let stock = stocks[stockIndex];
  let price = stock.price.toFixed(2);
  let change = (price - stock.history[stock.history.length - 2]).toFixed(2);
  let percentChange = ((change / price) * 100).toFixed(2);
  let color = (change >= 0) ? "green" : "red";
  fill(color);
  text(stock.ticker + ": " + price + " (" + change + ", " + percentChange + "%)", 10, 30);
  // display price history
  let history = stock.history;
  let minY = min(history);
  let maxY = max(history);
  let startX = 10;
  let endX = width - 10;
  let startY = height - 30;
  let endY = 100;
  let historyLength = 500; // added this line
  let startIndex = Math.max(0, history.length - historyLength); // added this line
  stroke(color);
  for (let i = startIndex; i < history.length - 1; i++) { // modified this line
    let x1 = map(i - startIndex, 0, historyLength - 1, startX, endX); // modified this line
    let y1 = map(history[i], minY, maxY, startY, endY);
    let x2 = map(i + 1 - startIndex, 0, historyLength - 1, startX, endX); // modified this line
    let y2 = map(history[i + 1], minY, maxY, startY, endY);
    line(x1, y1, x2, y2);
  }
}

function displayPortfolio(currentPortfolioValue) {
  fill(0);
  noStroke();
  textAlign(LEFT);
  text("User: " + userName + ", Cash: $" + userCash.toFixed(2) + ", Portfolio: $" + currentPortfolioValue.toFixed(2), 10, height - 10);
  let startY = height - 50;
  for (let i = 0; i < portfolio.length; i++) {
    let stock = portfolio[i];
    let price = stock.price.toFixed(2);
    let value = (stock.shares * stock.price).toFixed(2);
    let change = (price - stock.buyPrice).toFixed(2);
    let percentChange = ((change / price) * 100).toFixed(2);
    let color = (change >= 0) ? "green" : "red";
    fill(color);
    textSize(10);
    rect(0, startY - 20, width, 20);
    fill(255); // add this line
    text(stock.ticker + ": " + stock.shares + " shares @ " + price + " (" + change + ", " + percentChange + "%) - Value: $" + value, 10, startY - 5);
    startY -= 20;
  }
}
```

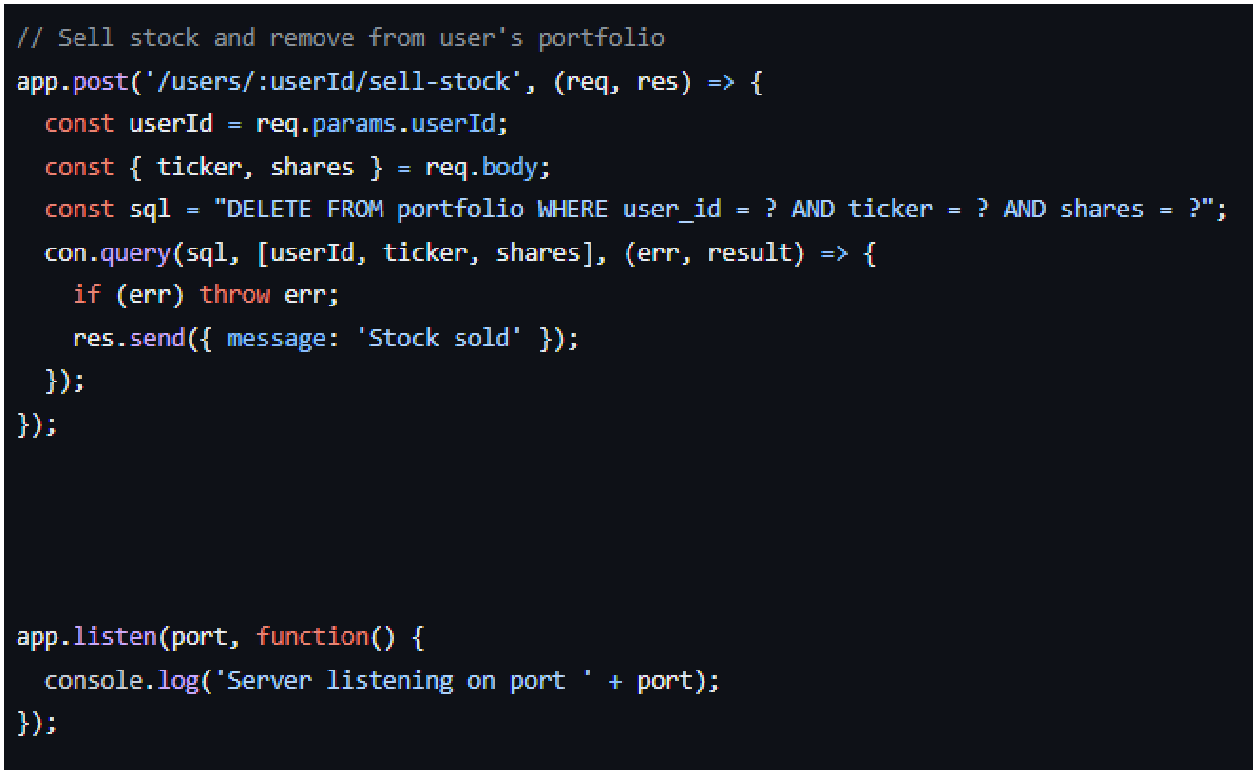
Functions for displaying stocks and portfolio.



```
app.get('/users', function(req, res) {
  con.query('SELECT * FROM users', function (err, result) {
    if (err) throw err;
    res.send(result);
  });
});
// User registration
app.post('/users/register', (req, res) => {
  const { username, password } = req.body;
  const sql = "INSERT INTO users (username, password) VALUES (?)";
  con.query(sql, [[username, password]], (err, result) => {
    if (err) throw err;
    res.send({ message: 'User registered' });
  });
});

// User login
app.post('/users/login', (req, res) => {
  const { username, password } = req.body;
  const sql = "SELECT * FROM users WHERE username = ? AND password = ?";
  con.query(sql, [username, password], (err, result) => {
    if (err) throw err;
    if (result.length > 0) {
      const cashBalance = result[0].cash_balance;
      res.status(200).json({ message: 'Login successful', userId: result[0].id, cashBalance });
    } else {
      res.status(401).send({ message: 'Invalid username or password' });
    }

  });
});

// Updates user cash balance
app.post('/users/update-cash-balance', (req, res) => {
  const { username, cashBalance } = req.body;

  const sql = "UPDATE users SET cash_balance = ? WHERE username = ?";
  con.query(sql, [cashBalance, username], (err, result) => {
    if (err) {
      console.error('Error updating cash balance:', err);
      res.status(500).json({ message: 'Error updating cash balance' });
    } else {
      res.status(200).json({ message: 'Cash balance updated' });
    }
  });
});
```

All queries being made to the database (includes the remaining images).

```
// Fetch user cash balance
app.get('/users/:userId/cash-balance', (req, res) => {
  const userId = req.params.userId;
  const sql = "SELECT cash_balance FROM users WHERE id = ?";
  con.query(sql, [userId], (err, result) => {
    if (err) throw err;
    res.send(result[0]);
  });
});


// Fetch user data
app.get('/users/:userId', (req, res) => {
  const userId = req.params.userId;
  const sql = "SELECT * FROM users WHERE id = ?";
  con.query(sql, [userId], (err, result) => {
    if (err) throw err;
    res.send(result[0]);
  });
});

// Fetch stock data
app.get('/stocks', (req, res) => {
  const sql = "SELECT * FROM stocks";
  con.query(sql, (err, result) => {
    if (err) throw err;
    res.send(result);
  });
});

// Fetch user's portfolio
app.get('/users/:userId/portfolio', (req, res) => {
  const userId = req.params.userId;
  const sql = "SELECT * FROM portfolio WHERE user_id = ?";
  con.query(sql, [userId], (err, result) => {
    if (err) throw err;
    res.send(result);
  });
});

// Buy stock and add to user's portfolio
app.post('/users/:userId/buy-stock', (req, res) => {
  const userId = req.params.userId;
  const { ticker, shares, buyPrice, stopLossPrice } = req.body;
  const sql = "INSERT INTO portfolio (user_id, ticker, shares, buy_price, stop_loss_price) VALUES (?, ?, ?, ?, ?)";
  con.query(sql, [userId, ticker, shares, buyPrice, stopLossPrice], (err, result) => {
    if (err) throw err;
    res.send({ message: 'Stock bought' });
  });
});
```



```
// Sell stock and remove from user's portfolio
app.post('/users/:userId/sell-stock', (req, res) => {
  const userId = req.params.userId;
  const { ticker, shares } = req.body;
  const sql = "DELETE FROM portfolio WHERE user_id = ? AND ticker = ? AND shares = ?";
  con.query(sql, [userId, ticker, shares], (err, result) => {
    if (err) throw err;
    res.send({ message: 'Stock sold' });
  });
});




app.listen(port, function() {
  console.log('Server listening on port ' + port);
});
```

Discussion

There were three main issues we faced during the development of our project. We have a group of 8, which is many more people in the group than necessary, and assigning tasks to each person in the group was difficult, let alone distributing work equally. It was also a difficult process to host our own database server, given that it was hosted on a different machine, and we did not initially have the knowledge and experience to easily get the DBMS running. Lastly, we had difficulties implementing a stretch feature in our project, which was the availability for our program to predict stocks based on past data. In the real world, there are many factors which influence the price of the stock. While there are less factors in play for our virtualized economy, it's more difficult in our case because the price of the stocks are determined arbitrarily (which is actually determined from a pseudorandom algorithm). Because of this, we decided to abandon this idea in its entirety.

Contributors

Aidan Eiler: Project Architect, Project Report & Presentation
Andrew Schacht: Frontend
Ford Morris: Sysadmin/Deployment
James Power: Backend
Jonathan Morse: Fullstack
Joseph Curtis: Frontend, Project Report
Lydia Parsa: Frontend
Nate Canterbury: Tester