2020

CAB230 Stocks API – Server Side

CAB230

Stocks API – Server Side Application

Shaun Mccasker

N9965271

4/23/2020

Contents

This template is adapted from one created for a more elaborate application. The original author spends most of his professional life talking to clients and producing architecture and services reports. You may find this a bit more elaborate than you are used to, but it is there to help you get a better mark

This report will probably be around 5 pages or so including screenshots

Introduction

Purpose & description

The purpose of this application is to deploy an express server to serve information from a database using a set of endpoints hosted on an API. The database holds information about stock companies and their relevant stocks. This information is accessible to all users however, certain endpoints require authentication.

Completeness and Limitations

The current implementation fulfills all requirements, the following are functional;

- An express server is hosted and deployed on https.
- Users can register for an account
- Users can login to their account
- Upon login a session token is generated
- Password input is hashed and salted
- Users can access all not authenticated endpoints
- Authenticated users can access authenticated endpoints using a session token
- Endpoints allow for filtering when appropriate
- Error handling for incorrect parameters and/or endpoints
- Middleware is implemented to enhance security
- Secure information is stored within an external file .env

The application fulfills all requirements hence, there are no limitations

Endpoints

Swagger is hosted at “/”.

/stocks/symbols

FULLY FUNCTIONAL

/stocks/{symbol}

FULLY FUNCTIONAL

/stocks/authed/{symbol}

FULLY FUNCTIONAL

/user/register

FULLY FUNCTIONAL

/user/login

FULLY FUNCTIONAL

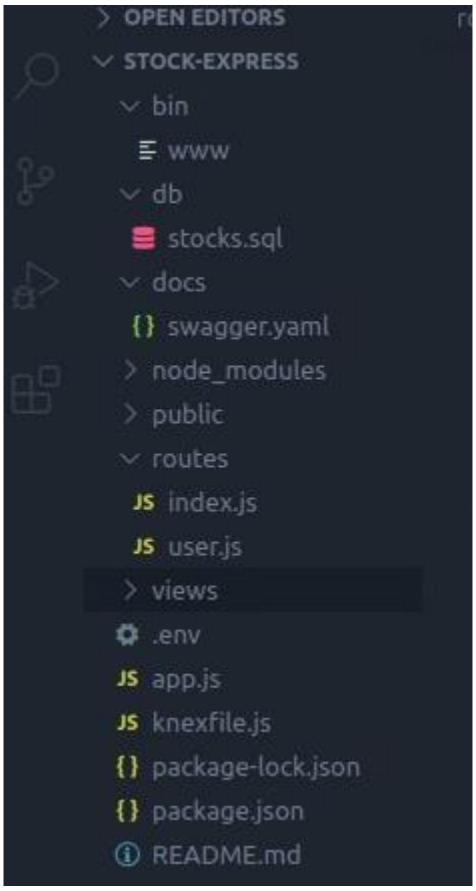
Modules used

No additional modules used

Technical Description

Architecture

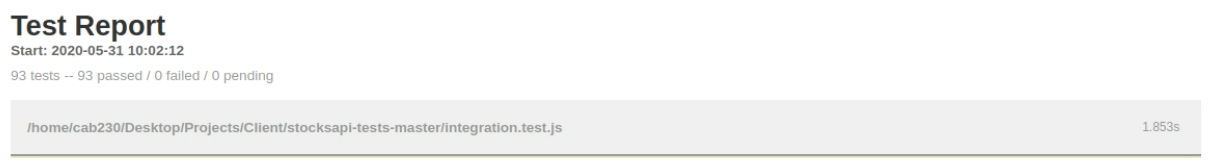
The architecture follows the model imposed by the Express-generator exactly. The server setup is initiated within ‘bin/www’. The database model is stored within ‘db/stocks.sql’. The swagger docs is stored within ‘docs/swagger.yaml’. The routes are handled within the route folder. As the routing is quite simple for this application, they are split into user and index, user handles user registration and login/logout and index handles all other endpoints. Public and Views contain styling, js and html elements however this was not needed to implement a simple API, but left within the structure for scalability. ‘.env’ holds local variables such as the port, local-ip and secret key. This file is made to store variables which need to be used within the application but should not be known by others. This file should not be uploaded or given to others. ‘knexile.js’ exports the required parameters to connect to the database. ‘app.js’ is the main top-level application, this handles all imports and calls all other function required to run the application.



```
> OPEN EDITORS
v STOCK-EXPRESS
  v bin
      www
  v db
      stocks.sql
  v docs
      swagger.yaml
  > node_modules
  > public
  v routes
      index.js
      user.js
  > views
    .env
    app.js
    knexfile.js
    package-lock.json
    package.json
    README.md
```

Testing

Tests.html contains the full report.



Test Report

Start: 2020-05-31 10:02:12

93 tests -- 93 passed / 0 failed / 0 pending

/home/cab230/Desktop/Projects/Client/stocksapi-tests-master/integration.test.js 1.853s

Difficulties / Exclusions / unresolved & persistent errors /

Difficulties

The process of developing this application was very straightforward, there were no difficulties in terms of implementing functional code.

Everything was implemented so there are no Exclusions or errors

Extensions (Optional)

This project could be extended by including a larger database to allow for increased functionality within the API. This could include;

- a larger stocks history to more accurately represent stock prices
- More stock companies
- The ability to view companies from foreign security exchanges, filterable by country

This would result in more endpoints and more functionality for the user.

Installation guide

Install all node modules as per the worksheets (including https)

Pm2 was occasionally crashing when trying to connect, I don't know what this happened, but as up this submission it seems to be working.

If the ip is incorrect:

- Check the ip of the local machine and append the '.env' file to contain the correct variable for ip
- Change ip in swagger docs to the new ip

Use 'sudo npm start' to run the server

Note: within the database the user table was set up as per the worksheets aswell.

The tests passed all the same both via 'sudo npm start' and the pm2 server.