2020

CAB230 Stocks API – Server Side

Declan Barrett

N10219358

5/10/2020

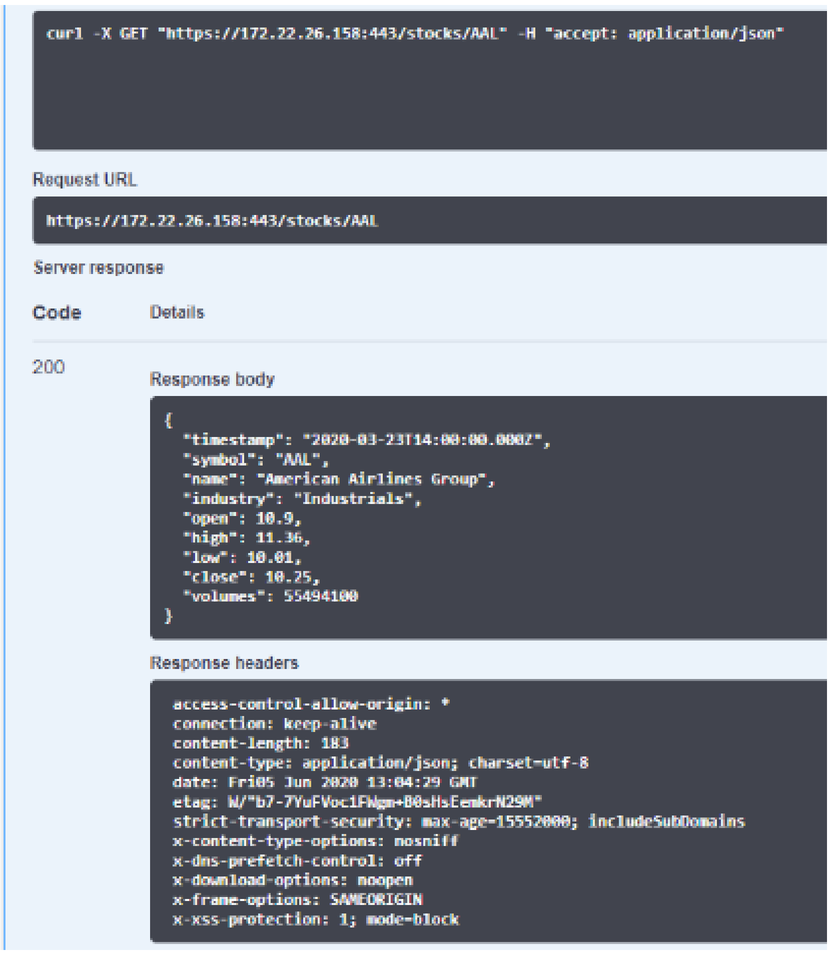
Contents

Introduction

Purpose & description

The Stocks REST API is designed to allow client-side apps to GET stock market statistics and POST user information, to and from a database. Through all this the API is meant to be clean, easy to query, secure and enhance the functionality of any client-side app that uses the API. Apps can get an array of information from the API, such as dated stock data, and have this data filtered by specific filters. The API can also handle the storage and security of user's login information and have the server check whether their information is valid or not.

The API's ease of use is enhanced by its interactive documentation which allows developers to view sample GETs and POSTs, and input search parameters into these examples and actively query the database from within the documentation. Behind the scenes, the API also has logging functionality to help with its management and security, is invulnerable to SQL injection attacks through the use of a SQL query builder and is a TLS server using self-signed certificates. The API is also reliable since it runs all day and all night, seven days a week, and can automatically restart itself if it crashes for any unforeseen reasons.



```
curl -X GET "https://172.22.26.158:443/stocks/AAL" -H "accept: application/json"
```

Request URL

```
https://172.22.26.158:443/stocks/AAL
```

Server response

Code Details

200 Response body

```
{
  "timestamp": "2020-03-23T14:00:00.000Z",
  "symbol": "AAL",
  "name": "American Airlines Group",
  "industry": "Industrials",
  "open": 10.9,
  "high": 11.36,
  "low": 10.01,
  "close": 10.25,
  "volumes": 55494100
}
```

Response headers

```
access-control-allow-origin: *
connection: keep-alive
content-length: 183
content-type: application/json; charset=utf-8
date: Fri05 Jun 2020 13:04:29 GMT
etag: W/"b7-7YuFVoc1FWgm+B0sHsEenkrN29M"
strict-transport-security: max-age=15552000; includeSubDomains
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-download-options: noopen
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
```

Completeness and Limitations

The Stocks REST API is a successful deployment of an Express API which supports all of the endpoints and interacts successfully with the database. The Swagger docs has been successfully deployed and can query the API and the site is secured using Helmet and Knex. All filtering, including time and industry, work and the authenticated route requires authentication to respond with data. Registration and login and JWT token handling have been successfully completed. All error responses have the correct messages and statuses. The site uses Morgan for database logging and uses self-signed certificates to achieve TLS making it a HTTPS site. The server is deployed on Linux and is running as a daemon using pm2.

End Points

/stocks/symbols

This is fully functional. When hit the endpoint returns all symbols with their name, industry and symbol, unless a query filter has been sent. The query filter is checked to see if it is called industry and if it is, the database query is changed to only retrieve symbols which have the query as part of their industry. If none match, an error is sent to the user, else the API responds with the requested information

/stocks/{symbol}
This is fully functional. When requested the endpoint returns the symbol and its current date data. If no symbol is provided, or the symbol does not exist on the database, an error is returned. If a date is supplied, an error returns specifying that they should use the authed endpoint.

/stocks/authed/{symbol}
This is fully functional. When the API is hit with a valid authorization token and symbol it returns the current symbol data. If it is not a valid token, it will respond with an access denied error. If a date is supplied, it will search with the date (either from or to), and If the date filtering provides nothing it will supply a queried date not found. If the endpoint is queried with rubbish query parameters then it will ask for the from and to parameters.

/user/register
The register endpoint is fully functional. When the API is posted an email and password it checks to see if the email already exists, and if it does not then it creates the new user. If the user already exists, it sends this error back. If there is incomplete post information, then it will also error and send it back.

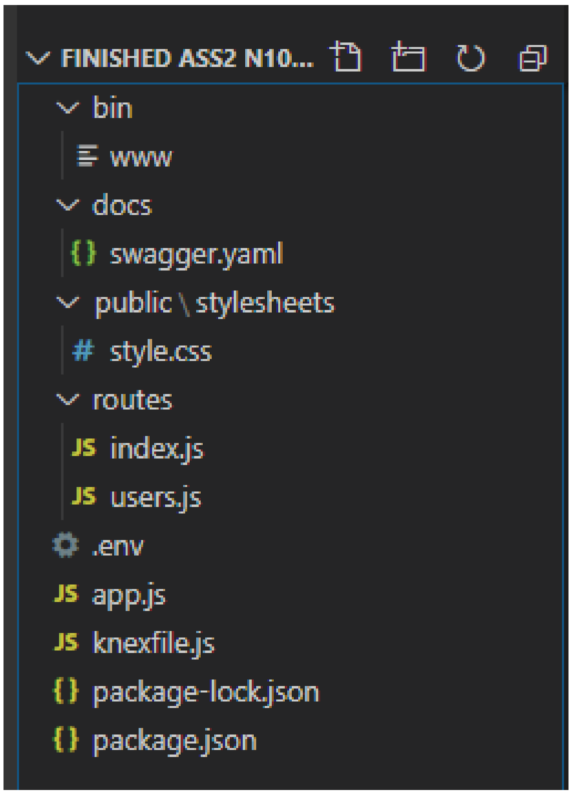
/user/login
This endpoint is fully functional. When the API is posted an email and password it checks to see if the email already exists, and if it does then it checks the password. If the hashes and emails line up, then it generates a JWT token and sends it back. If there is an error, such as the passwords do not match, or there is missing information, then the appropriate error is returned.

Overall, based on the assignment's specification and the current state of the Stocks REST API the web application completely fulfills the requirements of a grade of 7.

Modules used
No additional modules used

Technical Description

Architecture



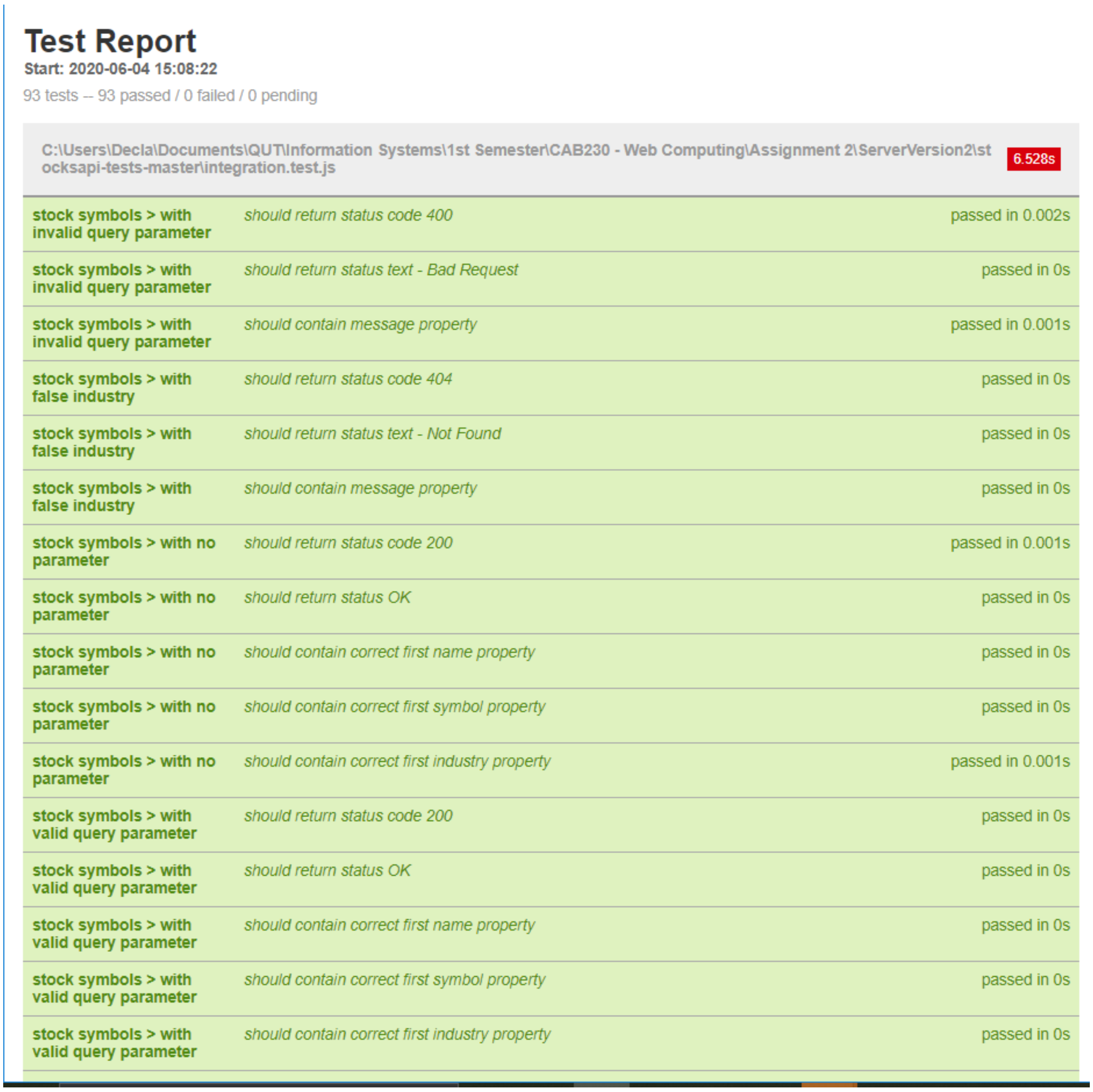
```
FINISHED ASS2 N10...
  bin
    www
  docs
    swagger.yaml
  public \ stylesheets
    style.css
  routes
    index.js
    users.js
  .env
  app.js
  knexfile.js
  package-lock.json
  package.json
```

The architecture of the API consists of a folder layout with separated route, docs and application layout. The routes are in the route folder, consisting of index and user and used in the app.js file. The index file is used for the stocks routes and the user file used for the user routes – which more specific routes placed first so that the more general routes do not match first and display. The www file is the bin and is used to start the server as https. The swagger docs is in the docs folder. The public folder has some CSS in it and the app.js, knexfile and env are at the top level of the application. The ENV file isn't used in the current version but is kept there in case it needs to be quickly changed to a different port alongside. The knexfile is used to set the details for connecting to the database, which is done in the app.js file, and used in both the index and user files. The www file initiates the server that uses app.js. In the app.js file, the required modules are linked and then the index and user routers are run through and appropriate responses returned. The layout could be modified and improved if the incoming API requests grew and a reverse-proxy was needed and thus nginx could be implemented (NGINX, 2020). Any routes not handled by the index or users have the swagger documents returned. The login route located in users sends a JWT token, which is authenticated in the index folder when it is being requested for the authed route.

Security

Knex was used as the query builder for this assignment. Helmet was also used with default settings enabled; no settings were changed or added, being used to set HTTP response headers to improve security. Cors also adds HTTP headers telling the browser to access selected resources from different origins. Morgan was used with dev logger mode selected, with additional header information displayed as well. Bcrypt was used to hash passwords to be stored in the database and JWTs were created when password hashes had been checked. JWT was used to check the JWT tokens when they were sent to the API for the authed route.

Test plan



Test Report

Start: 2020-06-04 15:08:22

93 tests – 93 passed / 0 failed / 0 pending

C:\Users\Decla\Documents\QUT\Information Systems\1st Semester\CAB230 - Web Computing\Assignment 2\ServerVersion2\stocksapi-tests-master\integration.test.js 6.528s

stock symbols > with invalid query parameter	should return status code 400	passed in 0.002s
stock symbols > with invalid query parameter	should return status text - Bad Request	passed in 0s
stock symbols > with invalid query parameter	should contain message property	passed in 0.001s
stock symbols > with false industry	should return status code 404	passed in 0s
stock symbols > with false industry	should return status text - Not Found	passed in 0s
stock symbols > with false industry	should contain message property	passed in 0s
stock symbols > with no parameter	should return status code 200	passed in 0.001s
stock symbols > with no parameter	should return status OK	passed in 0s
stock symbols > with no parameter	should contain correct first name property	passed in 0s
stock symbols > with no parameter	should contain correct first symbol property	passed in 0s
stock symbols > with no parameter	should contain correct first industry property	passed in 0.001s
stock symbols > with valid query parameter	should return status code 200	passed in 0s
stock symbols > with valid query parameter	should return status OK	passed in 0s
stock symbols > with valid query parameter	should contain correct first name property	passed in 0s
stock symbols > with valid query parameter	should contain correct first symbol property	passed in 0s
stock symbols > with valid query parameter	should contain correct first industry property	passed in 0s

Difficulties / Exclusions / unresolved & persistent errors /

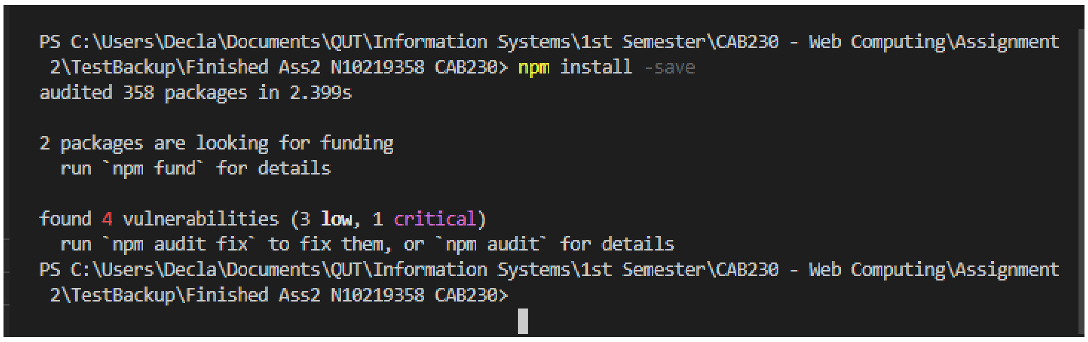
The major roadblocks that I had were dealing with the differences between the client side and server side programming styles. Express was new to me and having everything branch off of it was a learning experience. The flow and making sure the routes were handled as intended with no route stealing the intended request to another route tripped me up a couple of times. The database used in the practical made me stumble for a while since the database was returning that the query was successful but nothing was appearing the users table. I figured it out that the email varchar(40) was not long enough after managing to get a short email through. The logic surrounding the dates also made my brain melt, but after a lot of thinking I managed to get it working as intended. I also started the swagger docs before the yaml version was released and learning how swagger works with its layout of objects was quite confusing, but I managed to get the first query working properly. A large problem later on in the project was copying the server version (after implementing https) to my external hard drive. Linux refused to copy random files every time I tried. I had to copy the project and then copy the routes separately for some reason. All functionality was implemented, and no known bugs exist. The only thing I wanted to get implemented that wasn't included was to stop serving the swagger docs on all non-handled routes.

Extensions

Future extensions for the application would involve expanding the database dataset to include more recent data, adding the functionality to trade stocks and making the database update live with the stock market. To trade stocks that database would need to store a lot more information about currently owned stocks and positions for each person, quantities of money in their accounts and would need an increased level of verification inline with Australian and International laws.

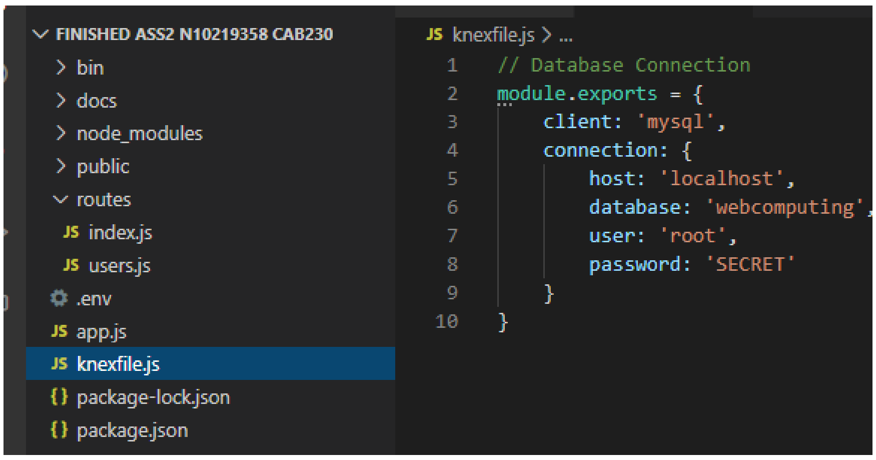
Installation guide

To install the API, first you need to download the API from whereever it is being stored. Then place it where you want to be stored. The API is currently configured to run on a Linux system thus placing it on a Linux system is reccommended. To run it, first make sure you have Node and MySQL installed. A copy of visual studio code is also helpful for editing files. Then install the dependencies using the command 'npm install --save' while inside the folder.



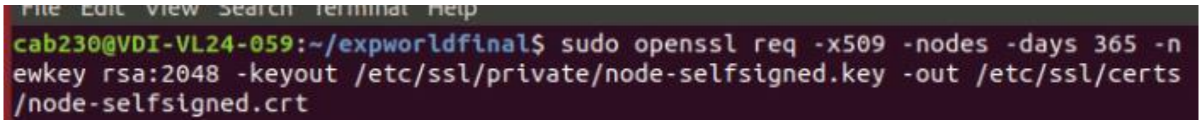
```
PS C:\Users\Decla\Documents\QUT\Information Systems\1st Semester\CAB230 - Web Computing\Assignment
 2\TestBackup\Finished Ass2 N10219358 CAB230> npm install -save
audited 358 packages in 2.399s

2 packages are looking for funding
  run `npm fund` for details

found 4 vulnerabilities (3 low, 1 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
PS C:\Users\Decla\Documents\QUT\Information Systems\1st Semester\CAB230 - Web Computing\Assignment
 2\TestBackup\Finished Ass2 N10219358 CAB230>
```

Next a MySQL Database using the stocks.sql and user.sql files will need to be created. These files create the stocks table and users table. The password used for the MySQL database then needs to be placed in the knexfile, so the API can query the database.



```
// Database Connection
module.exports = {
    client: 'mysql',
    connection: {
        host: 'localhost',
        database: 'webcomputing',
        user: 'root',
        password: 'SECRET'
    }
}
```

The API is also setup to offer https. A new private key and certificate will need to be generated and placed at '/etc/ssl/private/node-selfsigned.key' and '/etc/ssl/certs/node-selfsigned.crt' respectively. This is done using the openssl command and inputting the required details, including the ip address of the current machine (example provided by QUT).



```
cab230@VDI-VL24-059:~/expworldfinal$ sudo openssl req -x509 -nodes -days 365 -n
ewkey rsa:2048 -keyout /etc/ssl/private/node-selfsigned.key -out /etc/ssl/certs
/node-selfsigned.crt
```

Run ‘npm start’ while in the API folder or start the API using pm2 from the /bin/www file, and the API should be up and running. The API can then be queried from postman, using the array of tests at https://github.com/frankisawesome/stocksapi-tests or simply visited via typing in the desired URL and path. To use the API, follow the swagger docs provide on the / path.

References

NGINX. (2020). Welcome to NGINX Wiki! | NGINX. Retrieved 4 June 2020, from https://www.nginx.com/resources/wiki/