2020

CAB230 Stocks API – Server Side

CAB230

Stocks API – Server Side Application

<John Nguyen s>

<N10132767//s>

4/23/2020

Contents

Table of Contents

Introduction

Purpose & description

The server assignment is a complement to assignment 1. In assignment 1, we built a React app to hit the API hosted on the server at http://131.181.190.87:3000/. The updated version of this API for this assignment may be found at http://131.181.190.87:3005/.

The task is to implement and deploy an Express application that replicates the services provided by this API as documented in the Swagger at the aforementioned URL.

The principal technologies used in this express application are:
• Node
• Express
• MySQL
• Swagger (YAML source supplied)
• Knex, helmet, morgan and JWT

Completeness and Limitations

Here we want you to tell us in a couple of sentences what works and what doesn't. ***Make a claim against the standards we laid out in the assignment specification (see below) and briefly justify that claim.*** Don't give us deep details of the bugs here. Putting a positive spin on what you have achieved is fine – by all means focus on the stuff that works. But be realistic in your claim. As for the client side, the text below is stolen straight from the assignment specification:

- **[Grade of 4 level]:** Successful deployment of an Express based API which supports ***some*** of the endpoints and interacts successfully with the database. Most likely people will complete the basic *Queries* routes, but may not have proper filtering or completed *Users* routes, or have managed the authenticated query route. Swagger docs may not be deployed, and there may be significant gaps in the security requirements.
- **[Grade of 5 level]:** Successful implementation of ***all*** of the query endpoints at a basic level. Time-based filtering should basically work, even if the route is not properly authorised. Registration and login and JWT token handling must be attempted, though there may be issues with the authentication. At the grade of 4 or 5 level there may be a number of incorrect responses and codes even if the basic application works.
- **[Grade of 6 or 7 level]:** The grade of 6 and 7 levels require successful completion of ***all of the routes***. The distinction between 6 and 7 level grades for functionality then relates to problems in the valid and error responses on the routes, and in the successful use of middleware security, database connectivity and deployment of the Swagger docs. There is no 'killer' requirement here that makes the difference between a 6 and a 7. These requirements are best seen as a set that together, and done very well, give you a 7 standard mark. If you miss some of them, but still do a good job of the others, then you are likely to get a 6 standard mark.

Claim: standards of this react application has met the criteria of Grade 6 and 7 level

The express application successfully executes every route without failure. Positive and negative cases regarding POST and GET responses have been successfully tested on POSTMAN application with success. Furthermore, the deployed express application (on HTTPS) successfully passes all tests on all routes using cab230 stocksapi-test from GitHub. The application successfully implements use of middleware security, database connectivity and the deployment of swagger docs.

You may find it helpful to work with the list of endpoints below, telling us of any limitations. If the endpoint is fully functional, just say 'fully functional' after the route:

/stocks/symbol

- Fully functional

/stocks/{symbol}

- Fully functional

/stocks/authed/{symbol}

- Fully functional

/user/register

- Fully functional

/user/login

- Fully functional

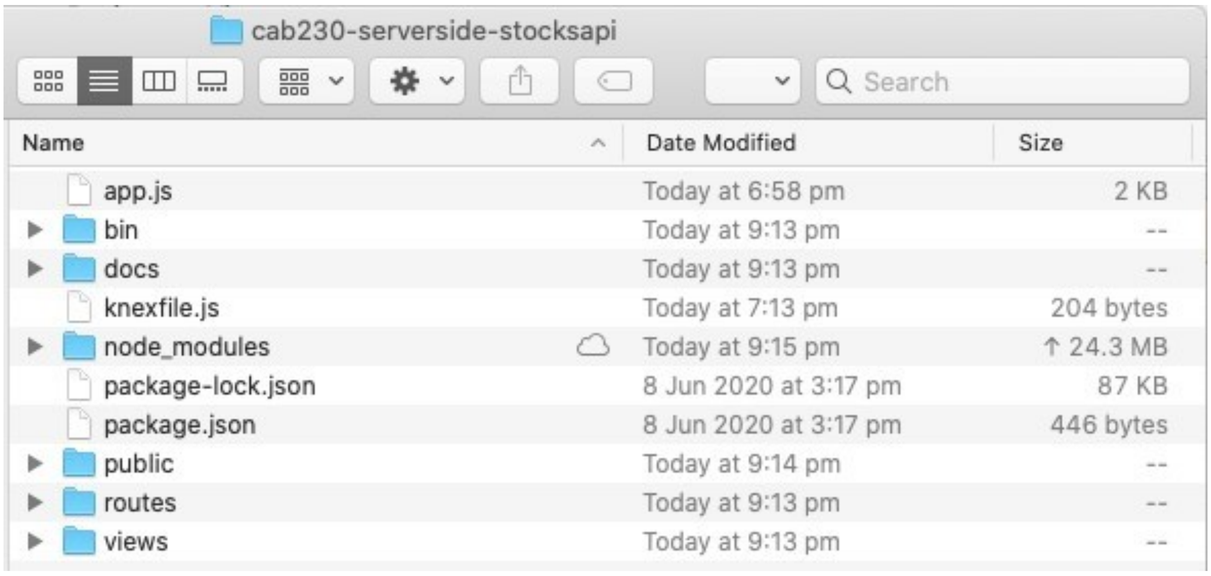
Modules used

No additional modules used

Technical Description

Architecture

The architecture of this application at source code level includes a main folder (named cab230serverside-stocksapi) which is the express application for this assignment. Inside that folder includes a bunch of important subdirectories and files which allows for the implementation for the express application. The first file is app.js, which is the main configuration file which contains most of the modules and middleware imported from the node modules folder and the knexfile for the database connectivity into MySQL. Furthermore, it contains modules imported from subdirectories that are from the bin, docs and routes.


cab230-serverside-stocksapi

Name	Date Modified	Size
app.js	Today at 6:58 pm	2 KB
bin	Today at 9:13 pm	--
docs	Today at 9:13 pm	--
knexfile.js	Today at 7:13 pm	204 bytes
node_modules	Today at 9:15 pm	↑ 24.3 MB
package-lock.json	8 Jun 2020 at 3:17 pm	87 KB
package.json	8 Jun 2020 at 3:17 pm	446 bytes
public	Today at 9:14 pm	--
routes	Today at 9:13 pm	--
views	Today at 9:13 pm	--

The /bin directory serves as a subdirectory where you can define various startup scripts. www is the file to start the express app as the webserver on QUT's VM. Specifically, it used for deployment in the assignment when using pm2. To deploy the express app, the command is: sudo pm2 start ./bin www –name="express_app_folder_name" (in this case is cab230-serverside-stocksapi)

The /docs directory contains the swagger documentation in the form of a YAML file. The swagger documentations will serve as the main UI at the home page, which will generate and maintain a visual solution for the API documentation.

The /node_modules directory (will be discarded for submission) contains all the dependency modules required to run the express application, which are relevant within the app.js and package.json files.

Finally, the /routes directory contains routes for the main endpoints for the server API. Specifically, routes contains with GET and POST methods that are linked to the MYSQL database – using knex middleware. The endpoints include:

stocks/{symbol}   */stocks/authed/{symbol}*

/user/register

/user/login

Security

There are a few security features used in this application.

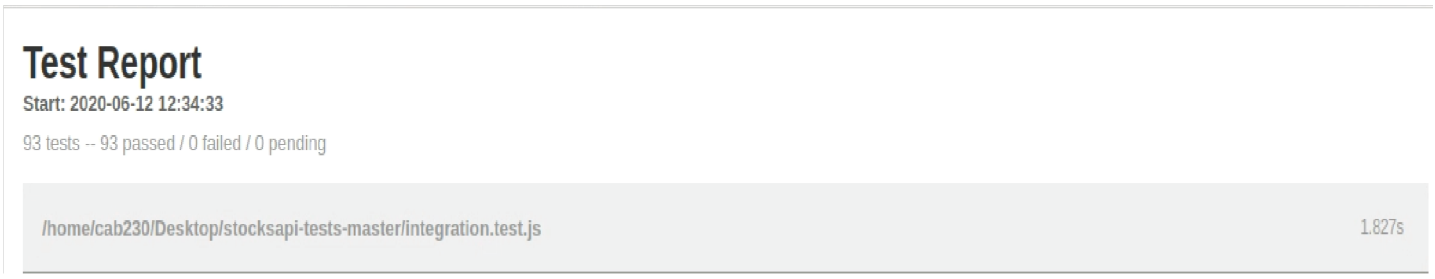
The use of knex was used as the query builder for the index.js file for efficient queries on the MySQL database. It manages the connection to the webcomputing database on the stocks table, and users table which are all relevant to all the of endpoints for the API data.

The application of helmet is a middleware to set headers and avoid vulnerabilities when authenticating tokens for login etc. For example, error handling is implemented for such cases where the token is invalid – perhaps the signature is invalid because a secret key value isn't defined, or the token has expired after 24 hours of logging in.

The use of morgan logging feature is to keep track of mainly the tokens within each request that is happening within the express application. By logging the tokens and keeping track of them, morgan allows the tokens to be stringified as the req headers (via a middleware) – which is useful when we're wanting to retrieve the token through the authorization header by writing: req.headers.authorization.

The appropriate handling of user passwords as described in the JWT server-side worksheet has been incorporated and adapted in the express application. Namely, bcyrpt and jwt-webtoken modules are included to create a hash for the password & compare the hashes of passwords for authentication and signing to create signature for the token, respectively.

Testing



Test Report

Start: 2020-06-12 12:34:33

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

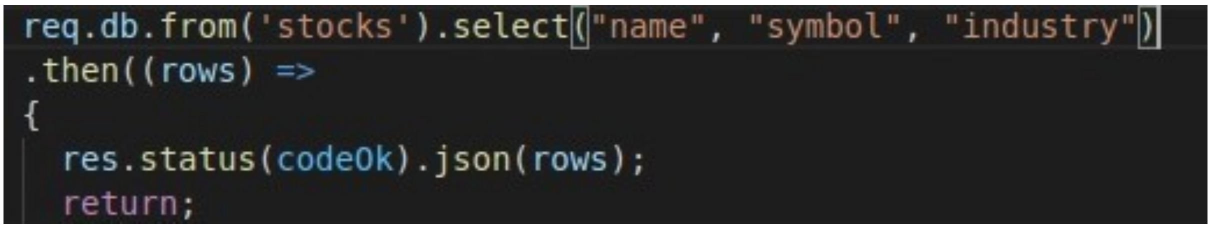
/home/cab230/Desktop/stocksapi-tests-master/integration.test.js 1.827s

In this section we expect you to screenshot the html test report:

Test report: 93/93 passed

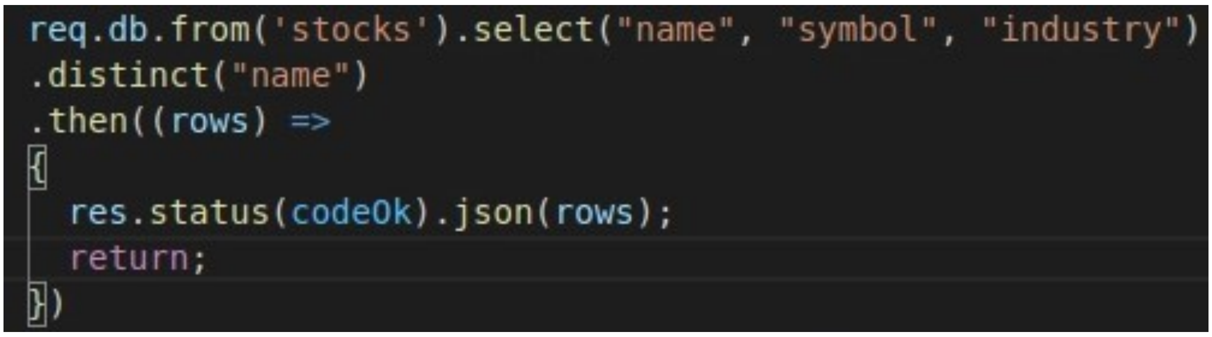
Difficulties / Exclusions / unresolved & persistent errors /

One roadblock discovered was display the /stocks/symbols endpoint and having all the data with nonduplicate values. Even when the endpoint displayed duplicate values on the Swagger UI, the testing still passed. Using this knex query in the screenshot below, it returns duplicate values in the swagger UI documentation under /stocks/symbols route.



```
req.db.from('stocks').select("name", "symbol", "industry")
.then((rows) =>
{
  res.status(codeOk).json(rows);
  return;
```

The fix was to include an extra step after the select clause and make the name of the stock unique by using the distinct operator. The fixed query is as follows:



```
req.db.from('stocks').select("name", "symbol", "industry")
.distinct("name")
.then((rows) =>
{
  res.status(codeOk).json(rows);
  return;
})
```

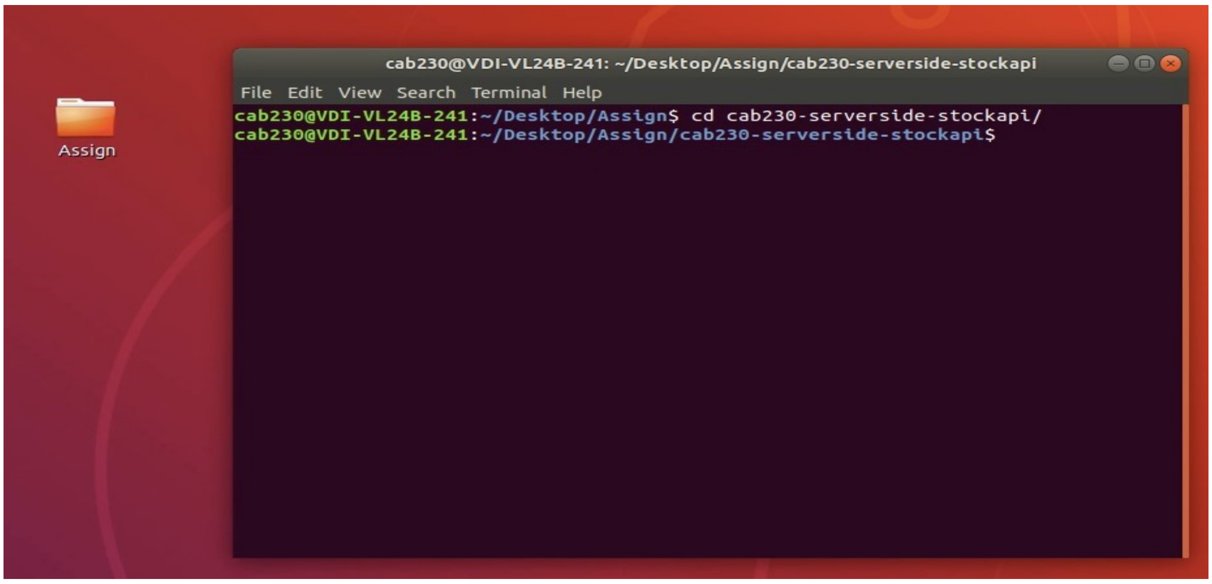
Another roadblock was that the email and password length for the user table was short, thus the registration and authentication testing for the express application failed. I followed the world.sql for the JWT prac sheet and used VARCHAR(45) and VARCHAR(60) for the password and email lengths, and it could not work for the webcomputing database. A quick fix was to change the varchar value to 80 as solved in the cab230 assignment 2 FAQ.

Installation guide

Tell us how to install the application – this should be at the same sort of level as most github sites.

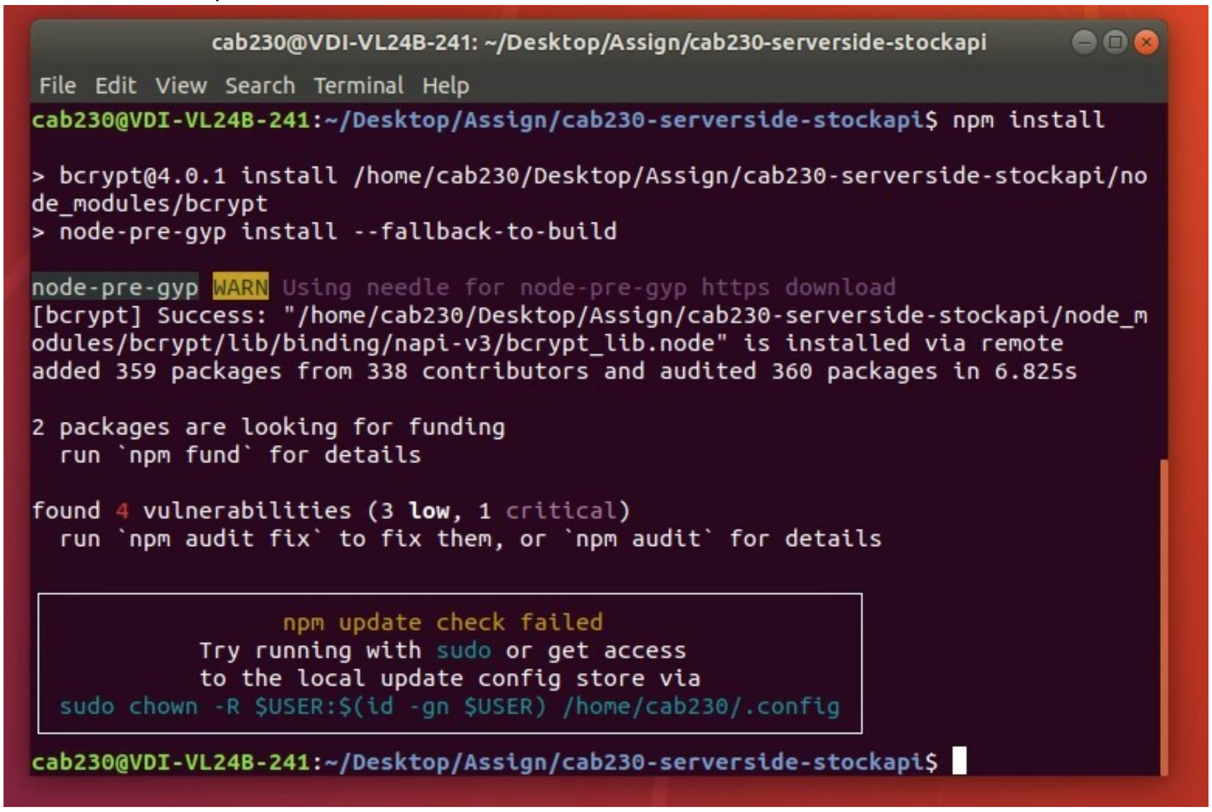
Use screenshots as needed.

On the QUT's Linux VM, download the assign.zip file, extract it and right click the assign folder and open terminal. Thereafter, head into the express application, by the following command: Cd cab230-serverside-stocksapi


Assign
cab230@VDI-VL24B-241: ~/Desktop/Assign/cab230-serverside-stockapi
File Edit View Search Terminal Help

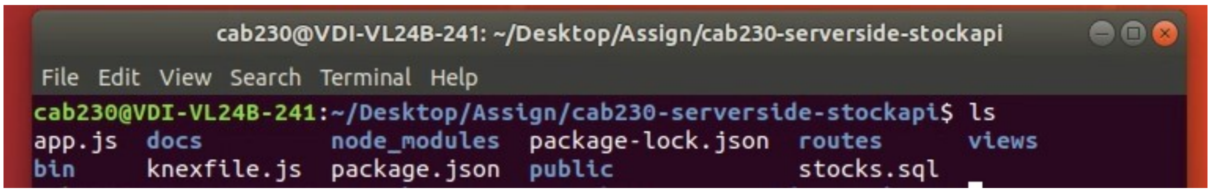
```
cab230@VDI-VL24B-241:~/Desktop/Assign$ cd cab230-serverside-stockapi/
cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$
```

When we list the files, we can see that the node modules aren't installed so we will install it by using the command: npm-install


cab230@VDI-VL24B-241: ~/Desktop/Assign/cab230-serverside-stockapi
File Edit View Search Terminal Help

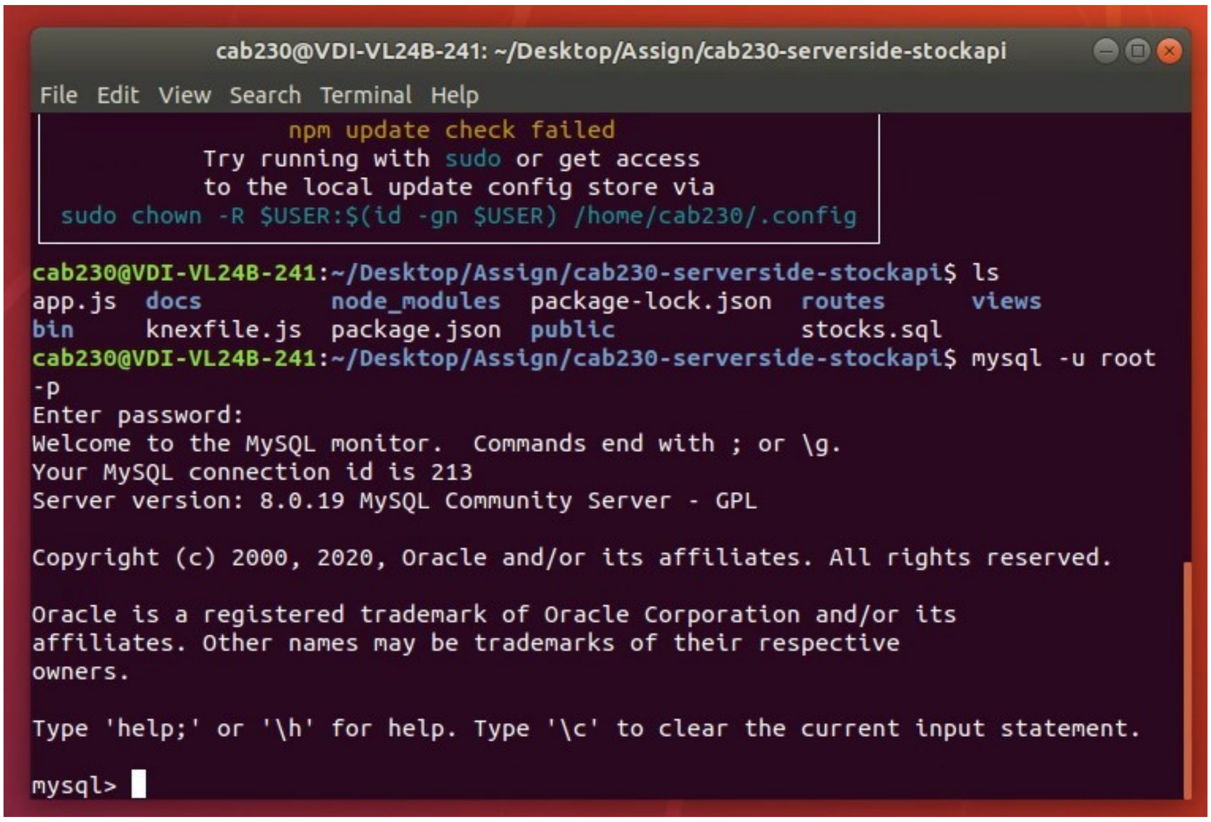
```
cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$ npm install

> bcrypt@4.0.1 install /home/cab230/Desktop/Assign/cab230-serverside-stockapi/no
de_modules/bcrypt
> node-pre-gyp install --fallback-to-build

node-pre-gyp WARN Using needle for node-pre-gyp https download
[bcrypt] Success: "/home/cab230/Desktop/Assign/cab230-serverside-stockapi/node_m
odules/bcrypt/lib/binding/napi-v3/bcrypt_lib.node" is installed via remote
added 359 packages from 338 contributors and audited 360 packages in 6.825s

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

               npm update check failed
         Try running with sudo or get access
         to the local update config store via
  sudo chown -R $USER:$(id -gn $USER) /home/cab230/.config

cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$
```

Now, we want to set up the database and load it with data for the express application. When we list the files in main directory, we see a stocks.sql dump file on the bottom right corner.


cab230@VDI-VL24B-241: ~/Desktop/Assign/cab230-serverside-stockapi
File Edit View Search Terminal Help

```
cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$ ls
app.js  docs         node_modules  package-lock.json  routes      views
bin     knexfile.js  package.json  public             stocks.sql
```

Next, we want to access mysql as the root user using the following command: mysql -u root -p


cab230@VDI-VL24B-241: ~/Desktop/Assign/cab230-serverside-stockapi
File Edit View Search Terminal Help

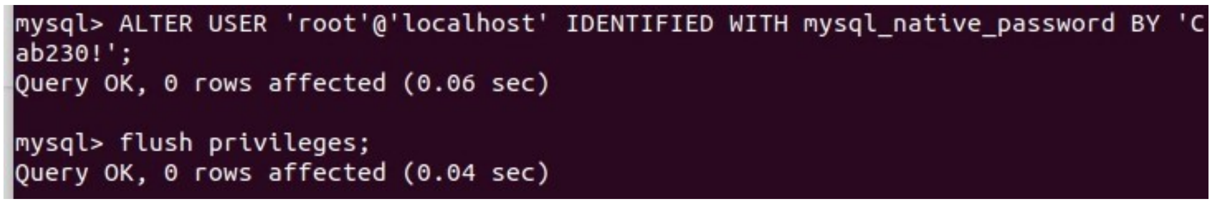
```
             npm update check failed
        Try running with sudo or get access
        to the local update config store via
  sudo chown -R $USER:$(id -gn $USER) /home/cab230/.config

cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$ ls
app.js  docs         node_modules  package-lock.json  routes      views
bin     knexfile.js  package.json  public             stocks.sql
cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$ mysql -u root
-p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 213
Server version: 8.0.19 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Now, we need to fix authentication protocol error in order to run the express application as the current SQL version is too high.

We type: ALTER USER ‘root’@’localhost’ IDENTIFIED WITH mysql_native_password BY ‘Cab230!’;

Where ‘root’ is the user for localhost for the database USER and ‘Cab230!’ is the database password. These values are important connecting the database to the express app through the knex middleware.

Then after the query execution, we want to flush privileges with the command: Flush privileges;



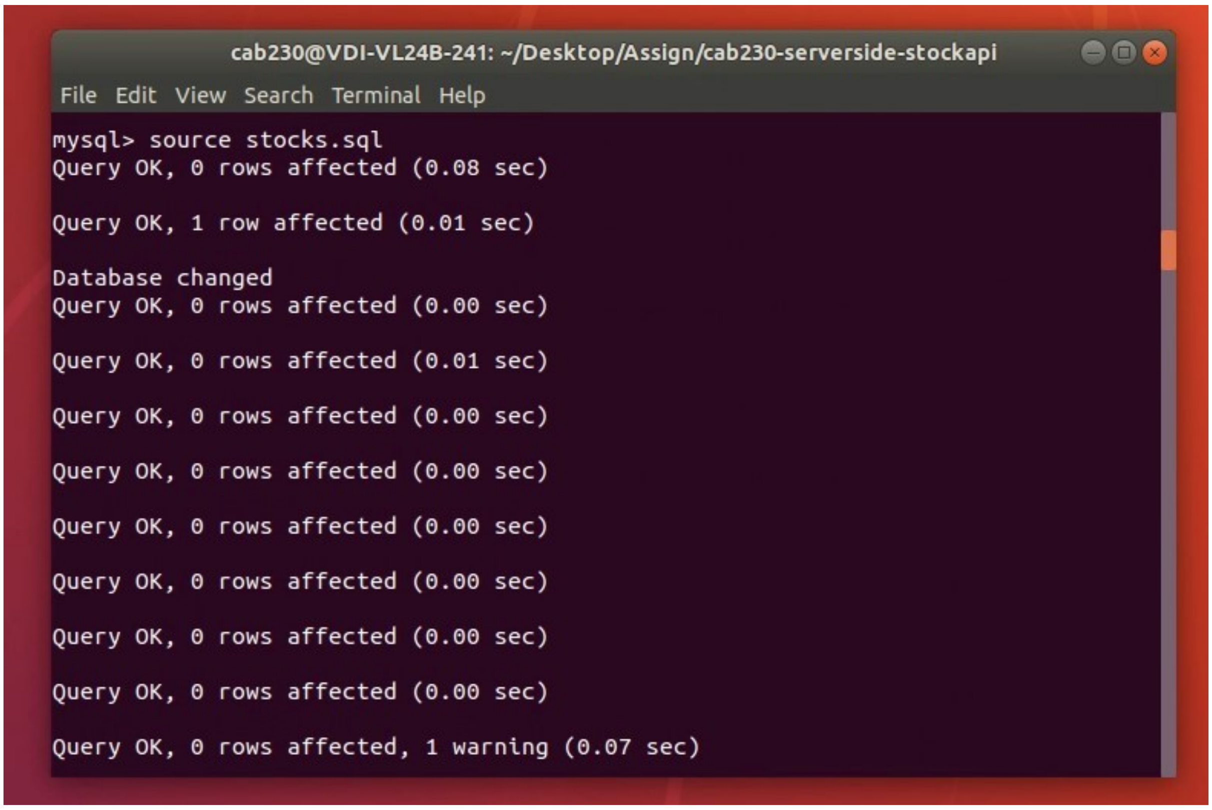
```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'C
ab230!';
Query OK, 0 rows affected (0.06 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.04 sec)
```

Next we want to create a database named webcomputing and use it with the following commands:

CREATE DATABASE webcomputing;

USE webcomputing;

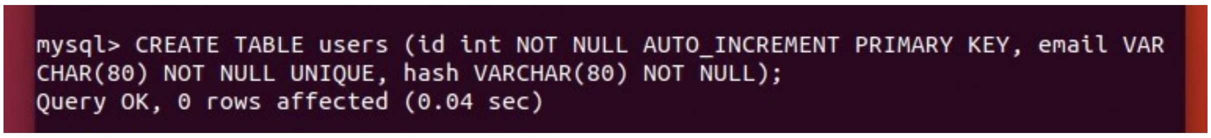
We can now load files into the database by using this source command: Source stocks.sql


cab230@VDI-VL24B-241: ~/Desktop/Assign/cab230-serverside-stockapi
File Edit View Search Terminal Help

```
mysql> source stocks.sql
Query OK, 0 rows affected (0.08 sec)

Query OK, 1 row affected (0.01 sec)

Database changed
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.07 sec)
```

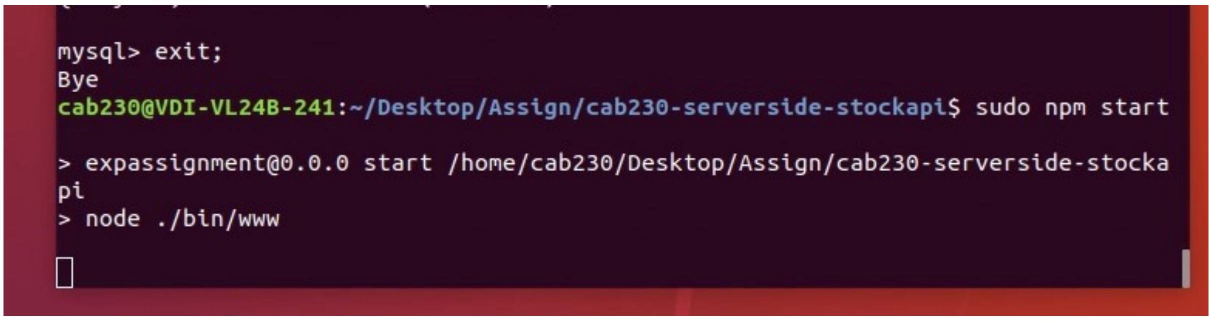
Then, we have to create the users table for user registration. The table will have 3 files, an id, email and hash (password).

Use the following command: CREATE TABLE users (id int NOT NULL AUTO_INCREMENT PRIMARY KEY, email VARCHAR(80) NOT NULL UNIQUE, hash VARCHAR(80) NOT NULL);



```
mysql> CREATE TABLE users (id int NOT NULL AUTO_INCREMENT PRIMARY KEY, email VAR
CHAR(80) NOT NULL UNIQUE, hash VARCHAR(80) NOT NULL);
Query OK, 0 rows affected (0.04 sec)
```

Now the express application is ready to be launched. Simply exit out of the database with the command “exit;” then launch the app by the command: sudo npm-start



```
mysql> exit;
Bye
cab230@VDI-VL24B-241:~/Desktop/Assign/cab230-serverside-stockapi$ sudo npm start

> expassignment@0.0.0 start /home/cab230/Desktop/Assign/cab230-serverside-stocka
pi
> node ./bin/www
```