CAB230 Web Computing Assignment
Assignment 2: Server Side

Release Date: May 9 2020
Submission Date: June 5 11:59PM
[Friday of Week 13, the week after lectures finish]
Weighting: 45% of Unit Assessment
Task: Individual Project

Updated Wednesday May 20 2020 at 10:35AM to make some security and architecture requirements more explicit. These comments point you to information in the pracs and the JWT Server Side worksheet. Updated June 2 to give port 3005 below.

Introduction:

This assignment is a complement to assignment 1. In assignment 1, you 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/

Your task here is to write and deploy an Express application that replicates the services provided by this API as documented in the Swagger at the URL above. We will not at any stage present an API specification in this document. You should rely solely on the Swagger.

The basic technologies will be very similar to those you have seen in the server side pracs:

- Node
- Express
- MySQL
- Swagger (YAML source supplied)
- Knex, helmet, morgan, JWT and others

No alternatives to these core technologies are permitted, but as before you may use a range of additional packages to assist you, with axios perhaps being the most obvious example.

Almost everything you need for this assignment can be adapted from the code you have written (or will write) for the server side pracs and the JWT server-side worksheet. It is a requirement that you deploy your application to the Linux VMs we have provided within the QUT firewall, and so you must ultimately use the MySQL DB on your instance, though you are free to develop your code elsewhere and deploy later if you wish. You should initially develop your application to serve HTTP, and make the switch to (self-signed) HTTPS just before submission.

The Data:

The stocks data will be provided to you as a SQL dump and we will give instructions on loading this into MySQL. Once these data are in the database, you should not touch them at all. You will need to create an additional table to handle the user accounts, but this is a separate issue.

The REST API:

The REST API endpoints are as documented in the Swagger docs referenced above, and you are required to implement them as they appear there. We will not rehash them here in any detail, and in any conflicts between what is written in this document and the Swagger API docs, the Swagger docs have absolute precedence.

Staged Development:

The requirements for the server side of this assignment are ultimately pretty straightforward, but we strongly recommend that you follow the path laid out below. Note that you should follow the conventions used in the pracs and the worksheets in respect of the application architecture and the application security, including middleware and handling of private information such as hashing of passwords. These details are in those pracs and worksheets.

Step 1: Create an Express App using Express Generator and establish the routes needed for the application without spending too much time on the application functionality. You can see this all on localhost at this point. You may organise these routes as you see fit, but we expect to see sensible application structure and use of routers to handle related tasks. The basic `indexRouter` vs `userRouter` split imposed by Express is a good approach. Applications in which all of the routes are handled from app.js will not be viewed favourably.

This assignment is (intentionally) very similar to the World DB REST API in the pracs. Follow the same approach, and use simple logging to ensure that the routes operate successfully.

Step 2: Having created an application and ensured that the routing is handled successfully, you should now replace these temporary logging statements with the requirements for each of the API endpoints. Most of the query endpoints correspond to simple `SELECT *` queries in SQL and these are a good point to begin. It is sensible to proceed as follows:

- Implement the *Query* endpoints in their basic form, ignoring authentication.
- Implement the *Users* endpoints, ignoring JWT
- Implement 'authentication' using a single dummy token to get the logic right
- Then add proper JWT token generation and handling and protect the authenticated route.
- Most of the filtering is simple – with the possible exception of the time ranges - but you might also want to add this in stages.

Step 3: Go through now and ensure that your endpoints have all of the responses covered. The Swagger docs show all of the appropriate codes that come back, though we do not

necessarily show all of the appropriate error messages – note that the docs say 'example message'. If you have the occasional incorrect error message, that is ok, but your response codes ***must*** match ours.

Step 4: At this stage, you should test your application using the tests that we will make available via a git repository (if you don't know git you will be able to download a zip archive). You will also have the option to submit additional tests via a pull request which will be made available to other students. Note that these tests will all be based on the outward facing endpoints of your system. They will not be local or unit tests. The testing section of your final report will be based on this work and may consist solely of the automated output.

Step 4: Having created a working application deployed via HTTP, you should then migrate the application to serve HTTPS on localhost using a self-signed certificate. This is based on the prac from week 9. At the same time or subsequently you will need to deploy your application to the Linux VM we have provided for you, and this will be the basis for subsequent demonstrations and marking.

Security and Logging:

We will require that you undertake an appropriate review of your application to limit its vulnerability… which is a very overblown way of saying that we expect some basic level of security as covered in the lectures and the pracs. This has to be a checklist at this level, and the requirements are:

- Use of `knex` or other query builder without raw SQL
- Use of `helmet` with at least the default settings enabled
- Use of `morgan` with a logging level similar to that used in the pracs.
- Appropriate handling of user passwords as described in the JWT Server-Side worksheet.

As noted above, migration to HTTPS should be undertaken just before submission.

Grade Standards:

Broadly speaking, grade standards for the server side of the assignment will correspond to the feature levels laid out below, and there is a reasonable alignment with the development steps outlined above. Mostly the lower grades correspond to the easy endpoints. The higher grades require that you successfully implement filtering, and that you follow a professional approach in the architecture and construction of the server. In particular, you should demonstrate that:

- The routes and the overall architecture are professional – logical and uncluttered, with appropriate use of specific routers mounted on the application
- There is appropriate use of middleware for managing components such as database connectivity and security

- There is appropriate error handling and responses match those in the API spec
- The application is successfully deployed using HTTPS
- There is an appropriate attention to application security
- The application successfully serves accurate Swagger docs on the home page route

All of these requirements must be met in order to achieve the highest grade levels.

As with the client-side assignment, you should understand that the grade levels discussed below assume that the feature levels required have been implemented competently. As always, if the implementation is substandard, then the marks will go down, sometimes substantially. It is more than possible to get a 4 level mark by doing a bad job of a 5 level requirement. Similarly, if you do a bad job of the 4 level requirement, then you are likely to get a 3 level mark. The precise mark necessarily depends on the mix of features successfully implemented, and the grade levels below are intended as a guide.

After taking note of all of these warnings and exceptions, the passing levels are as follows:

- **[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.

Submission

We will provide clear instructions on deployment and submission closer to deadline. We will certainly be connecting to your machine on the QUT local network and so it will be necessary to ensure that your server remains running during the marking period.

As with the client-side assignment, we will expect a short report, generally running to 5 (or at most 10) pages or so, including screenshots. We will again provide a template and guide, but your report must include the following sections:

1. Introduction – telling us what was implemented and what wasn't.
2. Technical description of the application. This section is to allow you to talk about the application architecture and middleware choices, and to discuss technical issues that caused you problems. This is especially important if something doesn't actually work.
3. Security – a brief discussion of the security features of your application.
4. Testing and limitations – test results as discussed above.
5. References
6. Appendix: a brief installation guide.

The CRA rubric will be released as a separate file.