CAB230 Server side report

Shannon Stephens, N10205144

# Introduction

What doesn’t quite work:

* When you have more than one term for a search criteria
* Swagger search (the swagger helper functions work fine)
* Knex raw

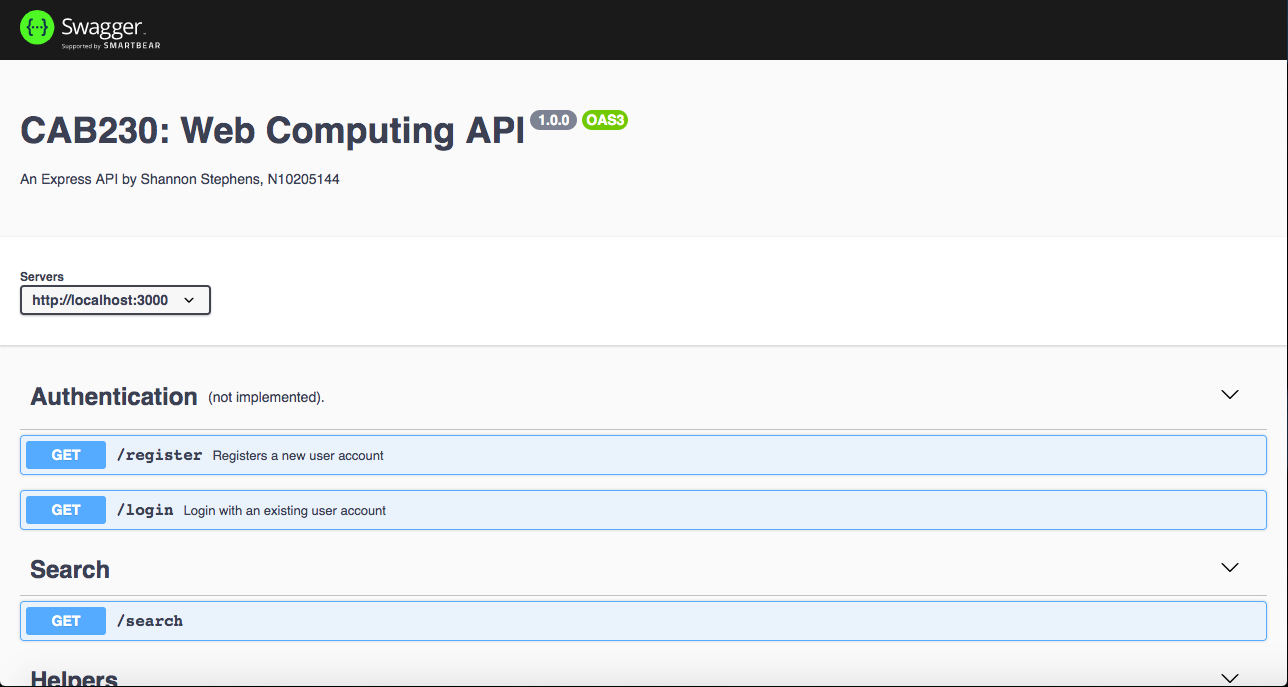
The server side portion of this assignment required an Express API that can handle all the routes as demonstrated with the given hackhouse API. This assignment does not handle any of the authentication routes and therefore does not require authentication for the search route.

This assignment does support all of the ‘helper’ endpoints: offences, areas, ages, genders and years. It also handles a search endpoint that allows for more than one search parameter (can search for an offence for a particular area in a year, ect.). It has a Swagger doc on root. The API gets the information requested from a database using Knex. Middleware is also used, though there is not much chaining and there is also basic logging.

There are three bits of functionality that leave something to be desired: when a search has many terms for the same criteria (e.g. gender=male&female) the search only responds with results for the first term, you cannot search for more than one offence at a time, and the Swagger doc does not return a working search (though the helper endpoints all work fine).

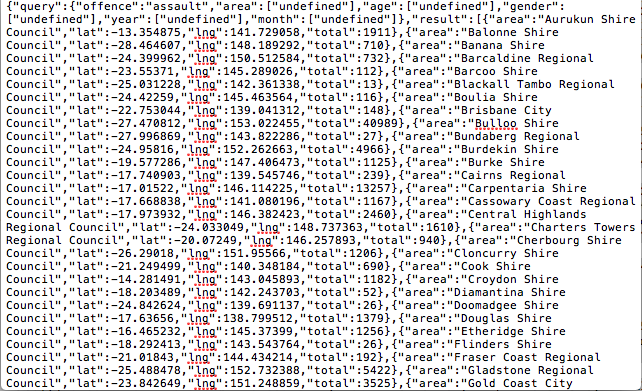
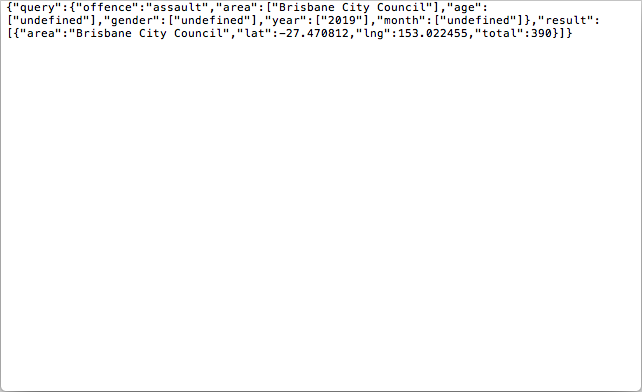
Similarly to the client side portion of this assignment, much of the more advanced functionality was not implemented due to timing issues. I started on this portion of the assignment after I felt I had a passing grade for the client side and that was rather late in the semester.

Below you can see the Swagger doc, the offence helper route, a basic search for an offence and a search with multiple criteria. Note that the screenshots containing json result data have been modified only for visual aid.

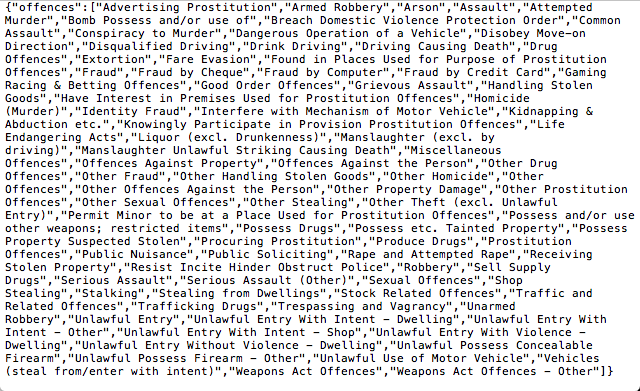


*Figure 1 – the Swagger doc on the API root*

*Figure 4 – the result of /search/offence=assault&area=Brisbane%20City%20Council&year=2019*



*Figure 3 – the result of /search/offence=assault*



*Figure 2 – the result of /offences*

# Technical description of the application

The application is made up of several files split (in my mind) logically. First off there is app.js which is where everything is held together. This is where all of the app.use() calls are and where all of the routes come together. Now the routes are split between three files, helpers.js, search.js and authentication.js where each file holds the route/s that pertain to that category. Nothing in authentication.js is of a working standard but was left in anyway.

App.js is also where the Swagger documentation is called, though the JSON for that is in a folder called docs.

As mentioned in the introduction, the use of middleware is quite basic in this assignment. There aren’t any chaining calls.

As again mentioned in the introduction, there were three problems in the implemented functionality, the first being that searches with many terms of the same criteria yield results only concerning the first term. This did not break the API and results were still given, though they were inaccurate for the complete search. This is rather similar to an issue I had with the first portion of the assignment. Strangely this issue does not seem to interfere where the search criteria contains numbers (e.g. year). I suspect it has something to do with the way the database is being queried.

The next issue with the API is that you cannot search for more than one offence at a time. Again I think this issue is to do with how the database is being queried however time did not permit for this issue to be resolved.

Lastly, the Swagger doc does not have a functioning search. The look of the Swagger doc is fairly the same as the one seen on hackhouse and the helper routes function just fine, though the search does not function on the Swagger doc page (and the authentication routes because they were not implemented, though they are present in the Swagger doc).

# Security

There was little security implemented as there is no authentication routes. In saying this, Knex was used for querying the database to prevent against SQL injections and the app is also using Helmet.

# Testing and limitations

|  |  |  |  |
| --- | --- | --- | --- |
| **Functionality tested** | **Test setup** | **Expected result** | **Actual result** |
| Root shows Swagger doc | Go to the root of the API | Swagger doc | As expected |
| Swagger doc displays all routes correctly | Visually look at the Swagger doc | Can see all routes that are meant to be in this assignment (including ones that are not implemented) | As expected |
| Swagger doc has functioning endpoint tests for implemented functionality | ‘Try it out’ on all implemented routes | Each implemented route returns the correct data | All of the helper functions return what they should in the Swagger doc, though the search endpoint returns a HTML file in the response body which it should not do. |
| All helper routes return their data | Go to the helper routes in the API and check | The helper routes return their expected data | As expected |
| Simple search | Try a simple search with only the offence | The API returns the expected data | As expected |
| Advanced search | Try a search with one term for each search criteria | The API returns the expected data | As expected |
| Multiple criteria search | Try a search with more than one term for each search criteria | The API returns the expected data | The returned data is accurate only for the first term for each search criteria |
| Multiple offences search | Try and do a simple search but with multiple offences | The API returns the expected data | The API returns “Error executing MySQL query” |
| Bad search | Try a search with rubbish terms for any search criteria | The API returns “Error executing MySQL query” | As expected |

# References

The Knex documentation was relied upon very heavily and there is code in this assignment which may bear a close resemblance to that in the Knexjs.org documentation.

# Appendix (installation guide)

This guide assumes you are working on a campus computer.

First things first you will need to open the VMWare Horizon Client and login with your student number and password. From there you will need to select Student – CAB230 Ubuntu and login to that with the password ‘Cab230!’. Open up a terminal in the virtual machine and check to see that node, npm and mySql are installed.

Copy over your server side code via your student drive or external usb to the home directory.

In the folder you just copied across to the home directory install the node modules by running ‘npm install –save’ in a terminal.

Edit the knexfile.js to use the database ‘web\_computing’, username ‘root’ and password ‘Cab230!’.

Use ‘/etc/ssl/private’ for keys and ‘/etc/ssl/certs’ for certificates.

Run the command ‘$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 - keyout /etc/ssl/private/node selfsigned.key -out /etc/ssl/certs/node-selfsigned.crt‘ for the openssl command

Now we need to edut the path for the key and the certificate files. The code should be in the ‘bin’ directory in the main application directory. Use an editor to open up the www file to put in the paths for the key and the certificate. This should look like the following:  
 const fs = require(‘fs’);

Const privateKey = fs.readFileSynch(‘/etc/ssl/private/node-selfsigned.key’ ‘utf8’);

Const certificate = fs.readFileSynch(‘/etc/ssl/certs/node-selfsigned.crt’ ‘utf8’);

Const credentials = {

Key: privateKey,

Cert: certificate

};