**CAB 230 Server-Side Report**

**Adam Conway n9819801**

Introduction

This report will cover the Server-side functionality for the cab 230 project. This is the api that serves data to the client side. For my client side the implemented feature include:

Get requests:

Ages, Areas, Genders, Offences, Search, Years

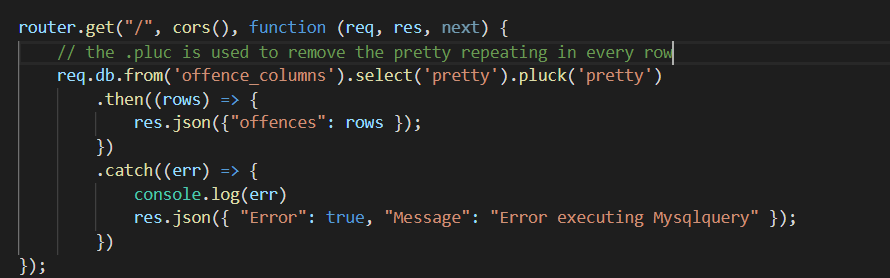
Post requests

Login, Register

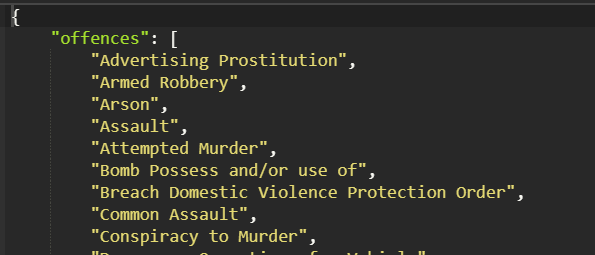
The Search Route is an authenticated route so only authenticated users may interact with this route. For searching Filtering by Age, Area, Gender year is available but no multiple term filtering is available. Filtering by month is not available either. The application is built on express with Middleware such as Knex, Helmet and CORS for database connection and security. Swagger documents have been implemented and are available on the /docs route.

Server URL https://172.22.24.177/

Technical Description

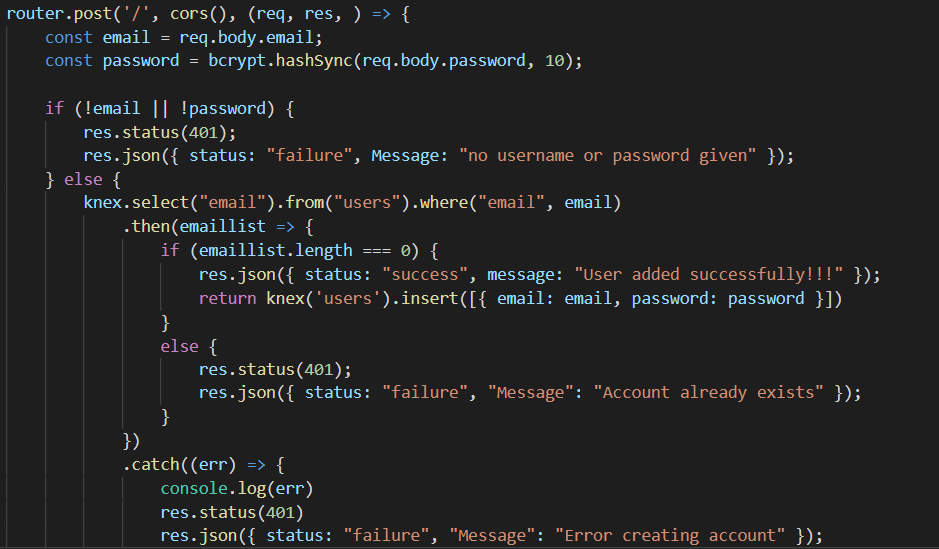
For this API each route is separated into its own file. The helper functions were simple to make as they require no parameters to get the information the user requires. All the helper functions share the same process and only vary in the information retrieved from the data base. The /offences route is a get request that uses Knex to interact with the database. Once the request is made, a Knex query is created selecting the ‘pretty’ values from the offence\_colums table then plucking the pretty from the returned results. 

The results are then formatted into a response with json and sent. The other helpers like areas and ages will select distinct values from the offences tables.

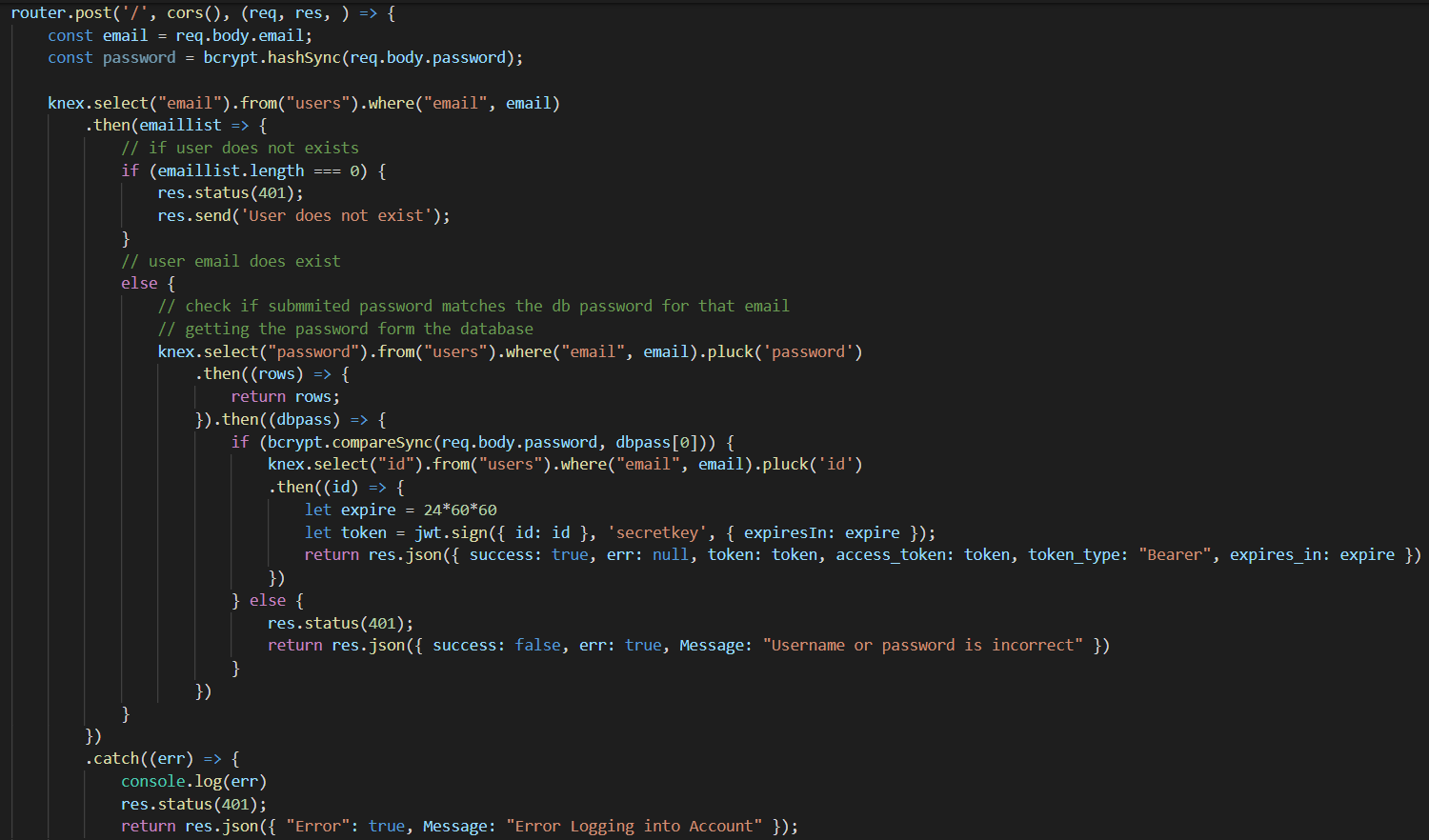


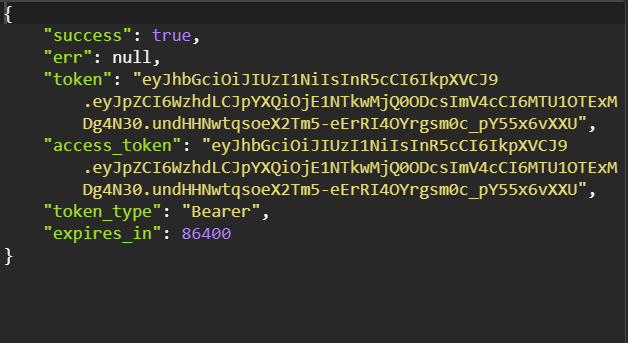
The Authentication routes are post request and require a different logic as the user wants to add information to the database or authenticate themselves to use the search endpoint.

The first Authentication route is registration. This route uses the standard cors, helmet and knex middleware but also bcrypt. Bcrypt is a hashing middleware that is used to conceal the password in the database so people with access to the database do not have the users passwords given to them. The logic for registration is more complex than the helper functions. Once a post request is sent the email and password are retrieved from the request, the email is store in the scope of the request function but the password is first hashed then stored. The function then checks if there email are password are not null, if either the email or password is null meaning the user did not send in an email or password in the request the server responds with a corresponding message. If that check passes the next check is to see if a user with that email is already in the data base if not the user is added to the database and client is sent a message saying registration successful. If the email already exists the client is sent a 401 status and told that the account already exists.



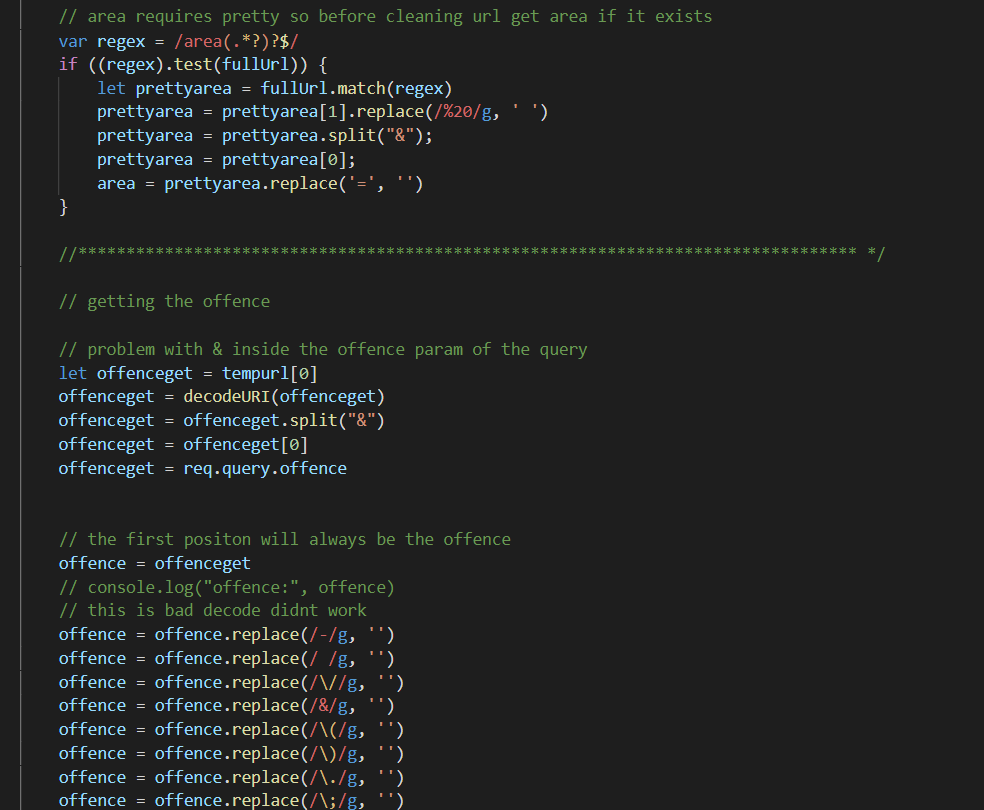
The second Authentication route is login. Login requires an extra middelware function of jsonwebtoken. This is used to create a token to authenticate the user for the search route. The starting of the login route is the same as registration. Get the email hash the password check if the email exists. Once up to the this instead of adding a user more checks are done. The password is retrived from the database in its hashed form and is campared to the client given password using bcrypts compare function. If the compare returns true it means the the password the client gives is the same as the stored password in the database linked to that email. Since the user has given the correct details the id is retrived from the database as the users do not know their id number and this is more secure then using the email. The id is then used with jsonwebtoken with the secretkey to sign a token to the users id. The token is then sent to the cleint along with the type and expire time of the token.



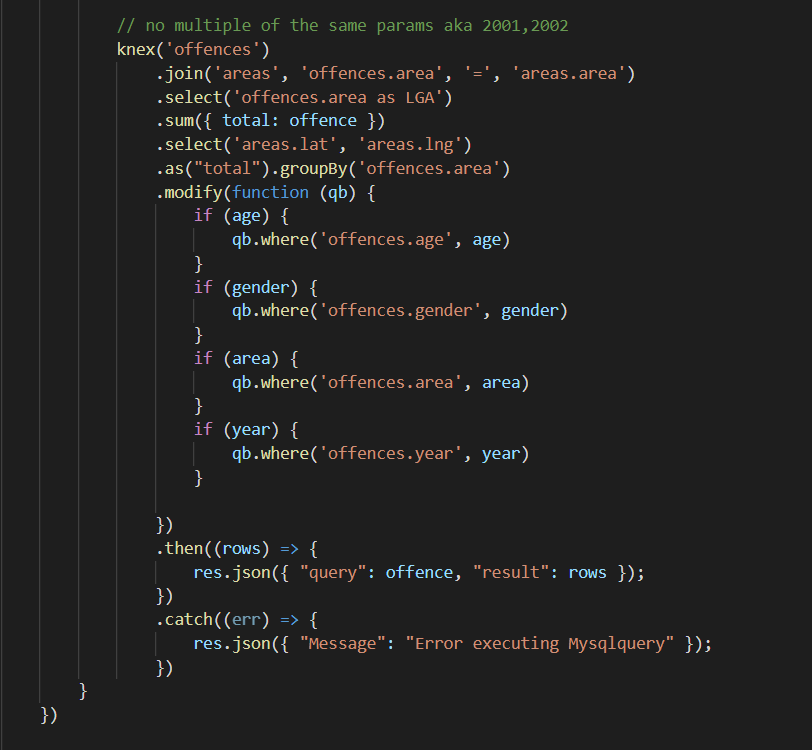
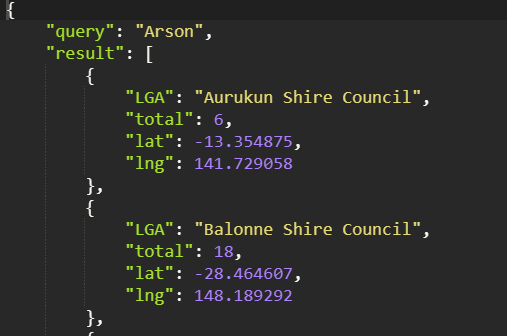


The final route is the search endpoint and this endpoint is the most tricky from all the endpoints. The Search endpoint was originally built without authentication to tests if the results were correct once I was happy with the functionality authentication was added on top of the search so only users with a valid token could search. The report will discuss search endpoint with authentication in place. The search route starts by getting the URL which contains all the parameters the client side has requested and the token from the authorization header. The token comes with the bearer tag and the tag is removed as the function only requires the actual token. If this fails it means no token was sent so the server responds with a 401 error. Once the token is retrieved it is then verified with the secret key using jsonwebtoken. If an error occurs in this step it means the token was not authenticated and the server responds with another 401 error. If the token is validated the query to the database starts being checked. The URL is then cut to find the offence query. If the url does not contain the offence query the server responds with a 400 error an message saying that search requires offence. Once this is passed another test is run to ensure the offence query has an offence param and if not, the server responds with another 400 error but message saying the offence query was empty. Once all these checks pass the query to the database begins building.

This is the part I struggled with as during creation I tried using request.params but I could not figure out how to use them properly and errors and bugs came though that I didn’t know how to fix. A more brute force attempt is used with regex but this is bad practice and should be replaced given time. The area is cleaned up first as it can cause trouble with DecodeURI. So using regex the area is taken from params if it exists and is cleaned up as the query I use to return the results requires the pretty version of the area name. The offence was the next problem as some offences use symbols which are not used in the database. As the offence is required for the query it can be retrieved from the url as its presence in the query has already been tested. Once the offence is retrieved and any offences with & in then have gotten the work around another bad coding practice is used and all the symbols are removed using .replace. The offence and area are now ready to be used in the database and stored in a local variable. The gender age and year params were much simpler to retrieve and required no special handling other than using regex to find them in the query *(the param was not developed in this serverside)*. If they exists in the query the are handled and stored in local variables to be used in the database functon.



Now that all the parameters have been retrieved a knex query is built joining the offences and areas tables and selecting the area the sum of offences grouped by area and the area longitude and latitude. If the filtering params are given the query is modified to include the filter to change the results depending on the filtering parameters. Once the knex query has finished and the data is returned the sever then responds with the query given to it along with the data it got from the knex query. If any invalid params were used the server responds by sending the client a error message. This helps deal with sql injection by denying any weird queries that are not made from the knex query.



Security

This application is served with HTTPS using a self signed certificate. Knex is used to prevent SQL Injection attacks. Helmet is used to secure the HTTPS headers to stop information leaking along with routing all http request to HTTPS. CORS is used to control what can be accessed and allows other webservers to access the information safely. Bcrypt keep the users password confidential and jsonwebtoken is used to authenticate/check authentication of the users.

Testing and Limitations

The only major limitation in this assessment was The & symbol not being encoded properly on my machine. I would expect it when retrieved from the client that the & symbol would become %26 but it was not the case. A work around was put in place to deal with this problem.

For testing the sql server was modified to allow the search request to work. To modify access the sql server and run the command *sql\_mode='STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_AUTO\_CREATE\_USER,NO\_ENGINE\_SUBSTITUTION';*

All Tests were run using Google Chrome or postman.

This is not the result of all test but is an example of the manual testing

Url for testing [https://172.22.24.177](https://172.22.24.177/)

**Helpers**

|  |  |  |
| --- | --- | --- |
| test | Expected output | Test result |
| /offences | Offences :[  Advertising Prostitution,  Armed Robbery,  … ] | Pass |
| /ages | ages :[  Adult,  Juvenile  ] | pass |
| /areas | areas :[  Aurukun Shire Council,  Balonne Shire Council,  … ] | Pass |
| /genders | genders :[  Male,  Female,  Not stated  ] | pass |
| /Years | years :[  2001,  2002,  …  ] | pass |

**Authorization**

|  |  |  |
| --- | --- | --- |
| test | Expected output | Test result |
| register with valid details | 200 | Pass |
| register with wrong details | 401 | pass |
| register with no email | 401 | Pass |
| register with no password | 401 | pass |
| login with valid details | 200 | pass |
| login with invalid details | 401 | pass |
| Login no username | 401 | pass |
| Login no password | 401 | pass |

**Search**

|  |  |  |
| --- | --- | --- |
| test | Expected output | Test result |
| Search with valid offence | 200  Results {} | Pass |
| Search with filtering parameters | 200  Results{} | pass |
| Search with invalid offence | 500  Error message | Pass |
| Search with invalid filters | 200  Empty Results | pass |
| Search with incorrect param names | 400  No results | pass |

References

Sql Server: <https://stackoverflow.com/questions/23921117/disable-only-full-group-by>

Knex: <https://devhints.io/knex>

Helmet: <https://www.npmjs.com/package/helmet>

CORS: <https://www.npmjs.com/package/cors>

Swagger: <https://editor.swagger.io/>

BCrypt: <https://www.npmjs.com/package/bcrypt>

JsonWebToken: https://www.npmjs.com/package/jsonwebtoken

Appendix

