Euan Mendoza Year 12 Major Project Logbook

# 15/02/2019

Tested different web frameworks for the potential use

# 22/02/2019

Created a material ui react project and testing different configurations

# 01/03/2019

Created a rust rocket rest api for potential use in project

# 08/03/2019

Created a go project and moved repository to it, discovered a web technology called graphql which removes a redux state management dependency but is not available in rust, so moved to a go web server

# 25/03/2019

Moved repositories again to conform to the go development standards of using go modules, keeping the same packages

# 01/04/2019

Started developing web server. Setup Git Submodules, created branch feature/setup\_webserver in repo. Created a static webserver route for deploying the react website with Go.

# 05/04/2019

Rewrote algorithms and logic within requirements report and planning to better suit the project and improve simplicity. Planned crud features for database managed through an admin panel similar to the Django framework.

# 09/04/2019

Started building the actual graphql api using go-graphql. Some issues I have found is that go-graphql is not documented well, while other graphql libraries such as graphql-java and python graphene are documented well, made me reconsider using go. The other issue is code examples all have one to three queries and responses, my plan has a lot more than three graphql queries. Three quries in a graphql application is small enough to be built in one file, but my project is a lot larger, and personally I prefer to have a lot of code split up into multiple files than a large file with lots of functions. I considered moving to java to make the project object oriented so I could easily organise the project, but java involves a lot of setup with the build systems and dependency management. I also considered switching to a different go graphql implementation that had better support for separating graphql responders into separate files. The solution I ended up using was to create a file called models where I declare the ‘models’ for how the data is requested and stored based off the Django framework. It contains all the structs also known as objects or non primitive data types depending on which language your using, which are used in the application to serialize and deserialize into json which can be used by the client. The structs are than serialised in the same file into a graphql object, which can be used for commands like query users id which returns a list of all users id’s. The graphql object is than mapped to a graphql query request within the schema file, this file handles create, read, update, and delete requests. If a read request is called, a callback function is called which exists within separate files that end in handler, these functions return the requests from the graphql query. This way there will still be a large amount of functions per file, but it is neater than any of the examples provided.

# 10/04/2019

In order to validate the json web tokens for authorization, I needed to pass the json web token in the http request header, the only problem is that I was using a pre written http request and response handler. I decided that because go has inbuilt support for a lot of the server features, and the original intention of the go graphql library is that you roll your own http handler, I decided on creating a http handler function. This function takes the json web token and passes it into the context of the graphql request.

# 11/04/2019

I refactored the code and added comments and extra information

# 12/04/2019

I created the functions and modules that are used to store the constants, for example the secret key used for tagging the json web tokens. The functions are stored locally in a json file and parsed using the native libraries included with golang.