Reverse Engineering

— PASSWORD AUTHENTICATION —

This application has the purpose of securely sign up and login users to an application. All user data is stored in a mysql database with the password being stored in encrypted format.

— USER STORY —

As a user I want to sign up to an app, so that I have a username and password.

As a user I want to be able to use my username and password, so that I can safely access the app.

— TECHNOLOGIES USED —

- EXPRESS
- EXPRESS-SESSION
- LOCAL
- BCRYPTJS
- PASSPORT
- MYSQL2
- SEQUELIZE

— GET STARTED —

To be able to utilize the app just clone the repository.

Once completed you need to follow these steps;

1. You need to create a local db called "passport_demo" in your local mysql workbench

2. Now you need to access the config.json file and insert your own password into the “development” section.

3. You now need to go into the root directory of your cloned repo and run “npm install” to install all required packages into your node_modules (HINT: To assure best practise when later pushing updates to your github it is recommended to add node_modules to your “.gitignore” file).

4. While in terminal, run "node server.js" to start your server.

5. You can now access the application via "http://localhost:8080" in your local browser

6. Enjoy using the app!

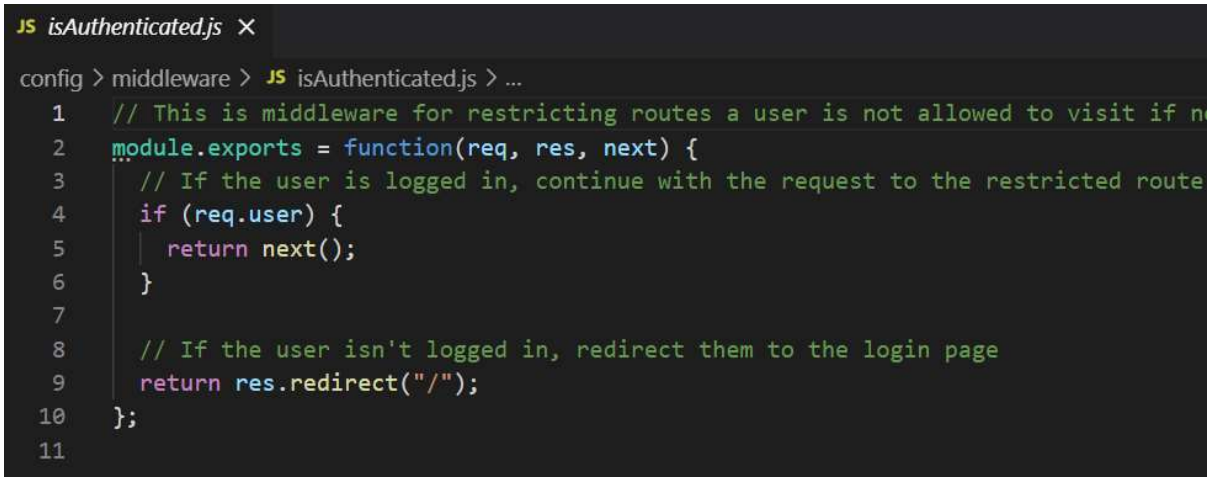
Reverse Engineering Code

— Codebase —

Config/middleware/isAuthenticated.js

The **isAuthenticated.js** is a function declaration be used as middleware for the application. The function checks whether the user credentials are correct or not. The user is then logged in or redirected.

The function gets called in HTML routes later and is only declared here.



Config/config.json

The **config.json** file is the configuration file for your connection to the database. The “development” and “test” keys are for the developers as they are creating, debugging and testing the app this will connect directly to the local mysql database (or one of developer’s preference if otherwise configured).

The “production” key is utilized when the application is deployed to a hosting platform such as HEROKU for instance. In this case the developer has to update the production section of their config.json file to reflect information which is usually given by the DSN of applications such as JAWS DB.

Config/passport.js

A LocalStrategy class which is given to us by “passport local” and imported into our **passport.js** file.

We also import “passport” into via require.

When the class is used by a passport method it is given an object via input and a callback. Within the object it is shown that the user will be using his email address as username. The callback then is given 2 inputs (email and password) and is then also given the done callback.

Within the done callback we look via sequelize methods in the database if the email exists. When this promise returns no matching email it returns “incorrect email”. Should the email exists it continues to check if the password matches and returns “Incorrect password” should the password not match. In any other case it returns the user.

It then utilizes serialize to keep authentication state across HTTP requests.

Models/index.js

The **index.js** creates a new instance of sequelize class. It then reads the current folder (models/) and adds the Sequelize models as defined.

The models are then used to create my database tables as this application is a code first application. Sequlize will also run its methods using the models as the templates for execution and validation.

The only defined Model for this application is the user.js which contains the table information for the table users and utilizes the hash methods of “bcryptjs”. These moduls are not the actual tables but a object oriented mapping of the tables.

The User.js will be exported as a module and required in the index.js file.

Public/js/login.js

The fetch is sent via front-end code which will ensure a form is filled out and its content sent to the server for user verification and login. The code uses jquery to add an event listener on the submit form which send the request via jquery/ajax.

Public/js/members.js

This is front end code that has a get request for the /members route.

Public/js/signup.js

This is all front-end code will ensure a form is sent to the server using fetch when a user signs up for the application.

Routes/api-routes.js

The api-routes.js is a file that defines the routes for the server.js file, the file has 2 post requested for when the member signs up and when the member logs-in, the signup post request uses sequelize to create a new User and the login request uses the passport module to authenticate the users details before allowing log-in. The get requests are for the log-out and to send the user data in json format.

Routes/html-routes.js

The html-routes.js uses get requests to assure the HTML files are getting served to the front end.

/server.js

The server.js file is where everything is brought together. It requires the necessary npm packages and modules. It uses express to set up the server/app and all the necessary middleware for needed for authentication. It then requires the routes and ensures that the server is listening on the specified server depending whether it is being hosted or on the local host.