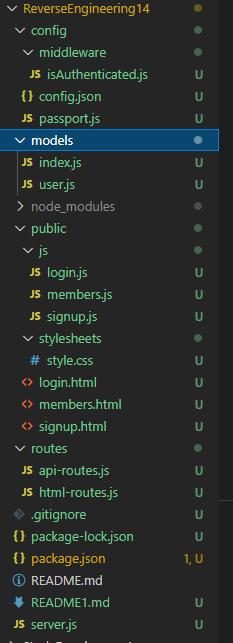
**Unit 14 – Sequelize Homework: Reverse Engineering Code**

**Directory Structure:**



**Main ReverseEngineering14 Folder:** The main folder has sub folders which I have inspected and will explain each of the following below.

**Server.js:** The server file shows a basic story of what happens on what application in what order. The application uses express, and it initializes passport, which it will use for login features, security, and authentication. The server file calls on it api-routes and html-routes and it does a console log to the current PORT.

**Package-lock-json:** package-lock. json is automatically generated for any operations where npm modifies either the node\_modules tree, or package. json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

Source: <https://docs.npmjs.com/cli/v6/configuring-npm/package-lock-json>

**Package-json:** A package. json is a JSON file that exists at the root of a Javascript/Node project. It holds metadata relevant to the project and it is used for managing the project's dependencies, scripts, version and a whole lot more.

Source: <https://blog.ezekielekunola.com/understanding-the-package.json-file>

**Config Subfolder:**

**isAuthenticated.js file:** This file is strictly here for restricting routes when a user is not logged in. There are methods in place in this file for what a user can see if they are a member vs non-member and logged in, vs. not logged in.

**config.json:** config. json is the main configuration file. Data from config. json is used to configure virtual machine. After editing file make sure that your JSON syntax is valid.

Source:[**https://www.drupal.org/node/2008800#:~:text=json%20Description-,config.,your%20JSON%20syntax%20is%20valid**](https://www.drupal.org/node/2008800#:~:text=json%20Description-,config.,your%20JSON%20syntax%20is%20valid)**.**

**passport.js:** Passport is authentication middleware for Node. It is designed to serve a singular purpose: authenticate requests. This separation of concerns keeps code clean and maintainable and makes Passport extremely easy to integrate into an application. Users will simply login by providing a username or password.

**Models Subfolder:**

**Index.js:** define’s the applications global variables, initializes sequelize

**User.js: -** This file is using bcrypt to hash a users password. When a password has been “hashed” it means it has been turned into a scrambled representation of itself. A user's password is taken and – using a key known to the site – the hash value is derived from the combination of both the password and the key, using a set algorithm. This file is doing that for the user to encrypt their password and enhance security.

**Public Subfolder:  
JS Subfolder:**

**login.js:** DOM handling, event handling, and executing API calls.

**members.js:** DOM handling, event handling, and executing API calls.

**Signup.js:** DOM handling, event handling, and executing API calls.

**Stylesheets Subfolder:**

**style.css:** where most of the page styling and esthetics goes on for the webpage. Fonts, colours, margins, etc.

**login.html:** login HTML.Basic HTML page. Calling on bootstrap, CSS.

**members.html:** Members HTML.Basic HTML page. Calling on bootstrap, CSS.

**signup.html:** Sign up HTMLBasic HTML page. Calling on bootstrap, CSS.

**Routes Subfolder:**

**Api-routes.js:** - Your routes and method handling for login, signup, logout, and user data. App GET and app POST methods are used.

**Html-routes.js:** - This file has basic html routes and landing pages to navigate the user. The routing navigates users to appropriate pages based on if they are logged in, logged out, a member, not a member.

**Future Improvements** – After reviewing the code and seeing one of the main features used is passport for login, security, and authentication, I recommend going a step further to improve this security and authentication in the future. OAuth could be used to further secure users and the help company authenticate the user as well. A single sign on using an OAuth provider like Facebook or Google could be good choices going forward.