HW- Reverse engineering the passport code

The following packages are dependencies for the passport program

1. **Bcrypt** - For safely storing the password by encryption- it utilizes hash algorithm
2. **Express** - For handling requests to and from the server
3. **Express-session** - For storing confidential, temporary user session data on the server by assigning a unique id and functioning as a middleware
4. **Mysql2-** For handling connection requests to and from mysql databases
5. **Passport**- For handling authentication of requests. It provides hooks to control what occurs when authentication fails or succeeds. It maintains persistent login sessions by serializing authenticated user to the session.
6. **Sequelize**- Facilitates execution of queries for CRUD operations on MYSQL database

**File structure:**

There is a main server javascript file to run the application and four folders:

1) **Routes:**

Contains files that handle server routes for handling requests for the database (api-routes.js) for handling requests directly from the user via the HTML (html-routes.js) via the routes /, /login and /members that displays the respective html pages of signup (default), login and member.

2) **Public:** It has two sub-folders that separates HTML and Javascript code

Contains html files that capture user input in the form of e-mail and password at the time of sign up (signup.html), alternatively a login page for a pre-existing user (login.html), displays a welcome page after login, with the option to log-out by clicking a link (members.html)

* HTML code features - There are forms that allow a user to input their e-mail and password and a login ‘submit’ button in the login page and a signup button in the sign-up page. The the input fields are specified e-mail and password types and are marked by ids: email-input and password-input respectively
* Javascript code for the chain of events

1. signup.js : On clicking the submit button (sign up). The values in the e-mail and password forms are captured and stored into an object called userdata with the properties specified as ‘email’ and ‘password’. Then it checks whether the username and password already exist- if not then it starts the *signupUser* function which takes up the email and password from the userdata object as parameters and then posts the typed in email and password to the server via the signup route (handled by api-routes.js via /api/sign-up) and then stores the username and password in a table. At the same time it redirects to the user to the login page. Once this step is complete, the email and password field values are turned to blank
2. Login.js: The code here is similar to signup.js in the sequence of obtaining email and pw values. The main difference is that there is a *loginUser* function instead of the *signupUser* function. This function sends a request to the server using the /api/login route specified in the api-routes.js. On this route the passport is authenticated by utilizing passport.authenticate and sends back user info in JSON format. Then another get request is performed via the route api/user\_data when sends an empty object if the user info does not match and e-mail and user id if the user info matches.
3. Members.js: It utilizes the route /api/user\_data and then displays the email

3) Models: This folder contains two javascript files user.js and index.js files

1. The user.js file handles 3 tasks. The first is performing validation checks on user input for the email and password- ensuring that the password is not blank and is typed as string; and in addition for e-mail to ensure that it is unique. The second using bycript for comparing hashed password with what the user has typed. The third task is using hooks to automatically hash the password.
2. The index.js file is auto generated during the configuration process, specifies the required components including fs, sequelize, config.

4) Config: It contains a file isAuthenticated.js inside a subfolder (middleware) that contains a function that takes request, response and next as parameters. If a request matches a user, then it returns the next function which continues with the request to the restricted route and if the user is not logged in then it redirects to the signup page. There is also a passport.js file that utilizes the passport package to verify whether an e-mail matches the stored user information and gives an error message if not. Same for password.

5) Server.js: This is configuring the server by specifying the configurations for express, express-session and passport packages. It specified the files to the routes or html and api. It syncs the MYSQL database and specifies the listening function and displays a message in the terminal upon success.