**Reverse Engineer**

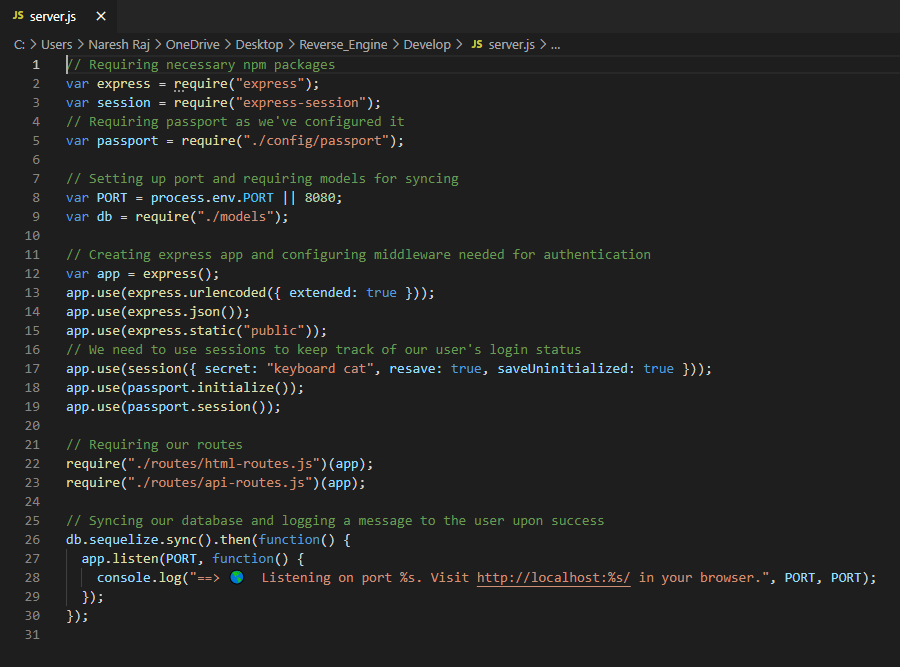
When we look inside the developer folder there are many files and folder, but the first thing to look for is the package.json file because it is a vital file always located in the root of a Node project and it holds the metadata related to the project and manages dependencies, scripts, versions etc.

So when we open this file , we will see a list of packages under dependencies which are the npm packages needed to run the app. One can visit npmjs.com website to know more about the packages and the commands to install them.



Here we can see that to run the app we need to run the server.js file ( “node server.js”)

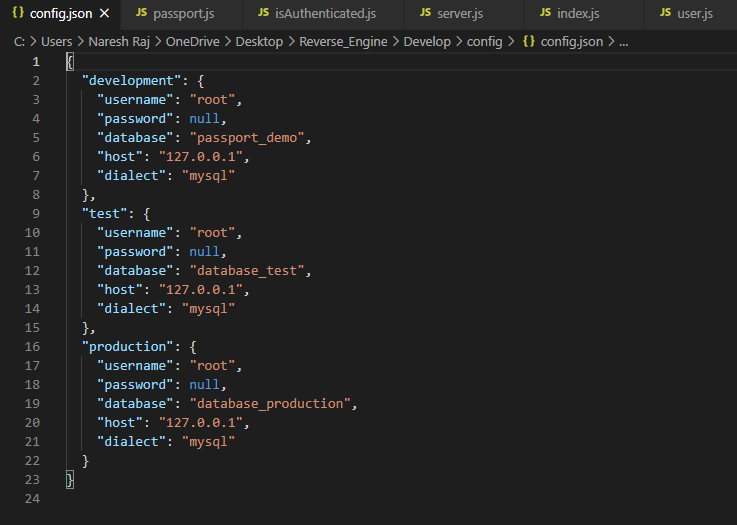
***server.js***



As we can see in the screenshot above, this file requires several npm packages : express, express-session and also requires passport.

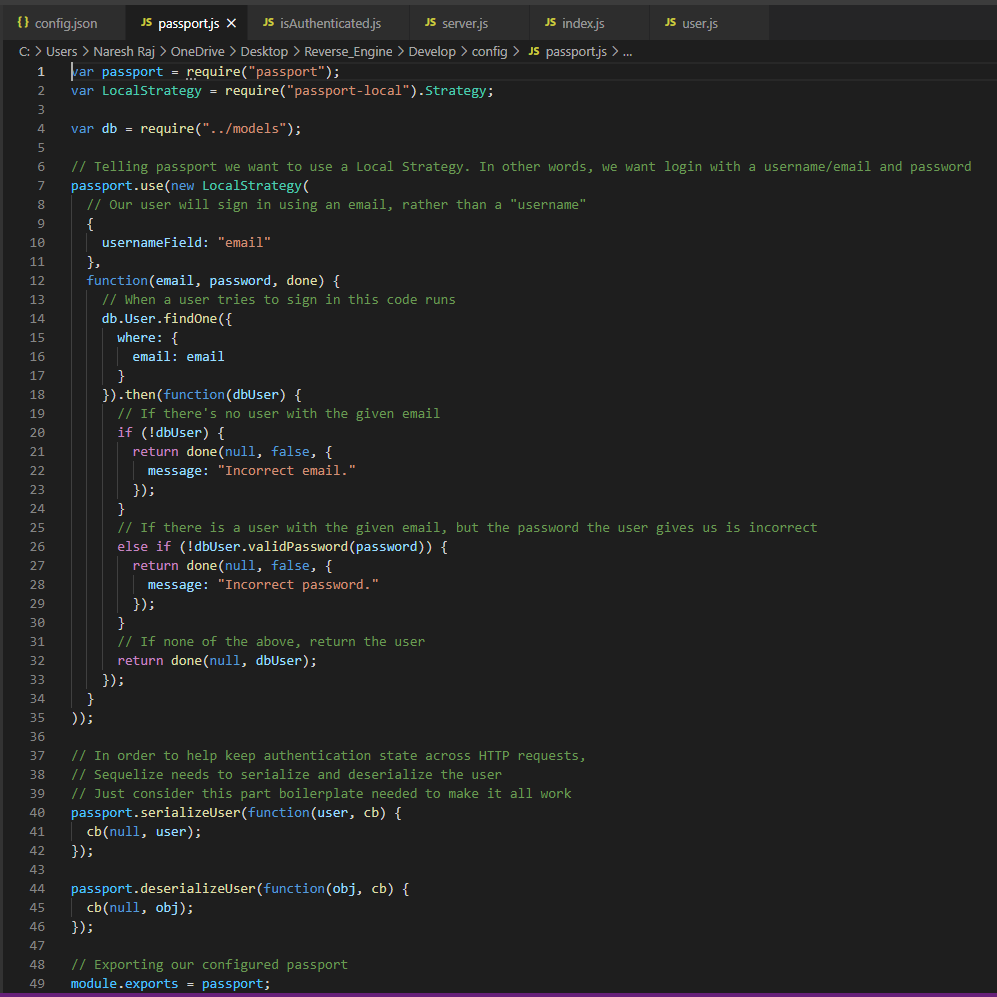
* + It first sets a PORT connection to a database.
  + In lines 12-19 we can see that a variable app is created and uses the express library. Using the app function from the express library it configures the middleware needed for authentication. The last three lines track the user’s log in status by creating a session from the moment they log in.
  + Lines 22-23 require the API routes files in the directory and use the app function to fun.
  + Lines 26-30 – this function syncs the databases, it then console a message when the connection is successful and how to open the file in browser.

***Config/ config.json***



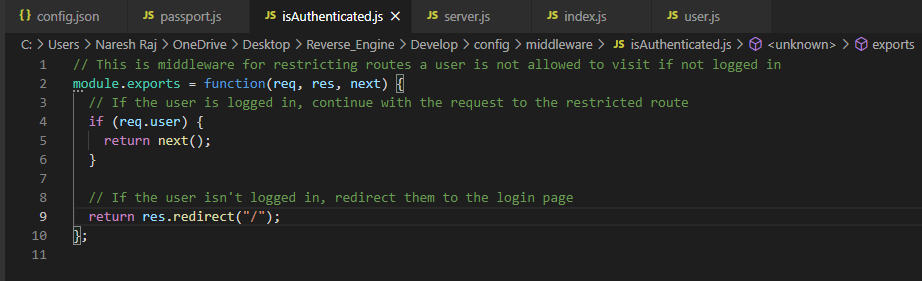
* + In this file, update your password for your local host. This page allows you to have multiple DB connections to run the app without having to change files every time you connect to your local DB or your production DB.

***Config/ passport.js***



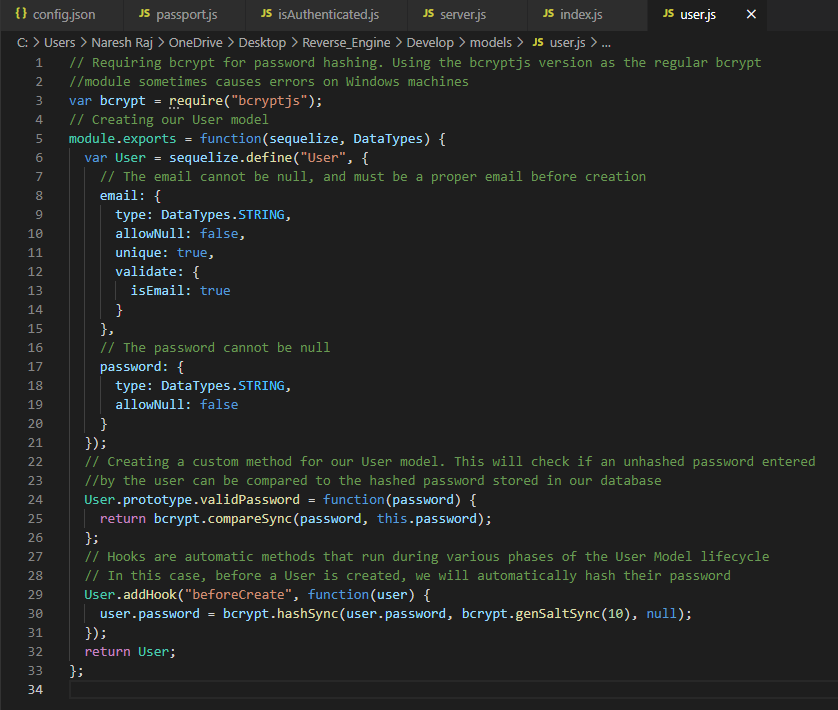
* + This page uses the passportjs middleware, by requiring the passport library and passport-local middleware. Passportjs authenticates requests from the browser to the server.
  + The page creates an object to import the models folder.
  + It uses the LocalStrategy callback function that accepts username and password arguments which are submitted through the app. The user signs in using their password, the code searches for a user to match the given email, if it finds it proceeds to the next page. If the email is not found then it will return a message indicating “incorrect email”. If the email is found but the password is incorrect it will return a “incorrect password” message.
  + The code at the bottom of the page between lines 40-49 is used to authenticate the HTTP requests, it ‘serializes’ or deconstructs the requests and once it reaches its endpoint it will reconstruct the request and run the code.

***Config/Middleware/isAuthenticated.js***



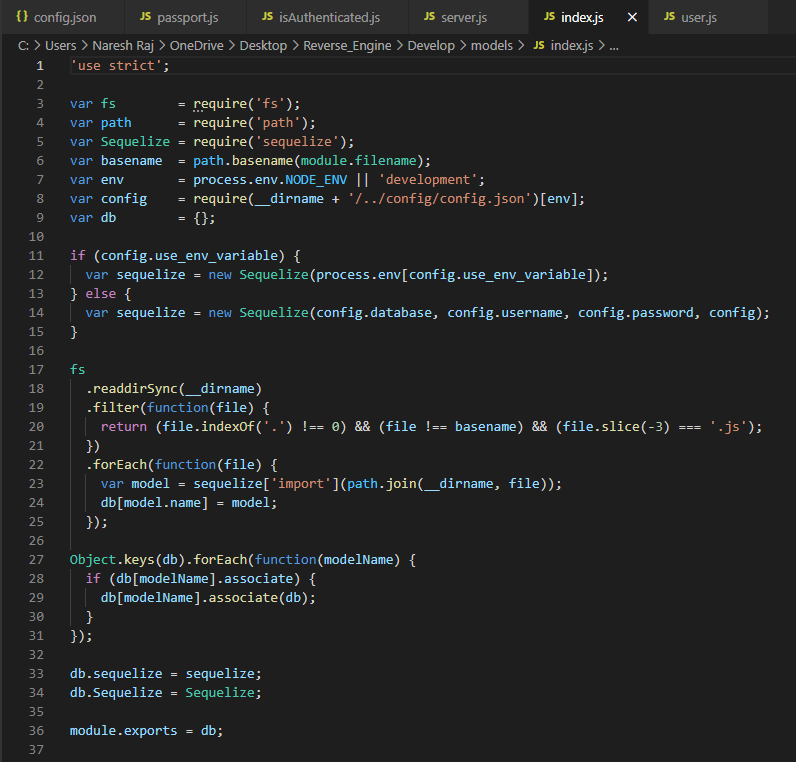
* + This file is the route to middleware in Express. This code restricts routes if the user is not logged in. It consists of a call back function that takes arguments: request, response and the next. This callback function, if the user is logged in, they will be able to continue with the request to the restricted route. If the user is not logged in it will redirect them to the log in page.
  + Middleware is also used for authenticating user connection.This will log the user out or end the connection with if there is no user activity for a predetermined amount of time. It will redirect the user to a login page to be able to regain access to that route (webpage).

***Model/user.js***



* + This file requires an npm package called bcryptjs.
  + It creates the User model using sequelize. The table has an email column with several attributes, it must be string, cannot be blank (null), must be unique, and must be validated. The second column is the password column and it accepts string data types.
  + The last lines in this file are functions that focus on reading hashed/masked passwords on DB and comparing it to the password the user entered, the final lines focus on password security and immediately hashing the password in the DB.
  + In lines 24 and 25, this function will read the password entered by the user and compares it to the hashed password stored in the database. The function consists of the user entry to match the function that will take the password as an argument and return a bcrypt function which is configured to read the password and compare to the password stored in the database.
  + The final lines (from 29) use the addHook function from bcrypt that will first hash the user password even before it creates the user.

***Model/index.js***

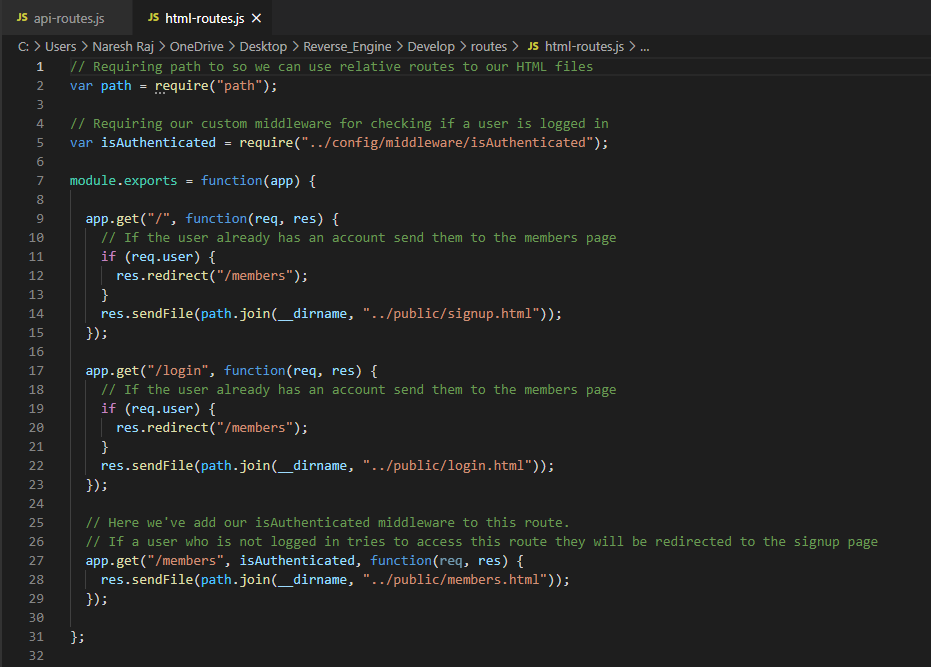


* + This file requires the following npm packages: fs, sequelize, path, and dot env(.env). It also requires the config.json file from the config folder.
  + A variable is declared as DB and is an empty object. It then uses a conditional statement and uses the “.env” function. The .env functionality is an ‘environmental variable’ created outside the program (typically through the operating system), and put it inside a new variable, if it is not an environment variable then it will configure the variable, pass it through the database - username and password tables.
  + Lines 17-25 use the file system or ‘fs’ npm module. The FS functions enable a synchronous connection to the DB directory to the index.js file. The Filter function checks that in that connection call there is something to read, cannot be this file and cannot end in “.js”. Each model file is then put an object to export.
  + Lines 27-31 with the model files retrieved will be useed the associate function to the model if it matches.

***Router/api-routes.js***

* + This file requires the models folder. It also requires the passport file and puts it inside the passport variable.
  + Lines 5-11 imports the function from the user.js file in the model folder. It uses the function passport.authenticate middleware to validate user login credentials, if successful it will send them to the main page, if unsuccessful the user will get an error message.
  + Lines 16-27 creates an API rout to for new users that sign up. It creates a post into the DB, using the User table, it creates a new user with the email and password attributes. If the new user is created successfully it will log the user in, if unsuccessful it will send back a 401-error message.
  + Lines 30-33 create a route for users to log out, redirecting them to the home page.
  + Lines 36-49 create a GET route to send some user data (email and user id) to the end user (browser) if the user is logged in. If the user is not logged it tit will send an empty object.

***Router/html-routes.js***



* This file is used for routing the user to concerned pages:
* If the user is logged in then send them to the related page (member page)
* And by using middleware to check that one user is logged in or not? and if not , then the user will be redirect to the signup page