# Config Folder:

This folder contains a middleware folder that holds 3 files inside:

* isAuthenticated.js: it is a middleware that restricts routes a user can access if they are not logged in. Once the user is confirmed to be logged in, the middleware will send him/her to the app page, and if not, a "redirect" will send them to the login page.
* config.json - inside this file, there are 3 databases: "passport-demo", "database-test", "database-production". The first is used for development, second for tests, and the last is for the production, when the user can see the page.
* passport.js -contains the passport authentication middleware for node.js to authenticate requests - for user login. It has an object named "LocalStrategy '' that uses the functions to work with "strategy" from inside of "passport-local" library. The variable "db" reads the models that exist and that control my database. It uses the "validPassword'' method to verify if the password is the same in the database. An object is created (new) to set how the user will sign-in, by email.This email search will be done through sequelize "findOne". If the user doesn't type the email, an error message will show: "Incorrect email". The same with the password, a message will show: "Incorrect Password" if the input isn't correct.

# Models Folder:

This folder contains index.js and user.js files. The first thing index.js does is finding out if we are running in Production or Development, and based on that it will configure Sequelize accordingly to the settings from config.json. After that it will use the FS library to read through the files found in the “models” folder that end with JS and is not itself (line 20), and then load them as models inside Sequelize. This is important because in the future we can add more models without changing the code, and just simply by adding more JS files into the Models folder.

This user model is in user.js that is responsible for setting up the sequelize table with email and password.The "addHook" is an automatic method and it runs before an user is created and "will automatically hash their password." When hashing the password, the purpose is to not allow someone to read the DB record and use the password to gain access later. It makes the password more secure.

# Public Folder:

This folder contains all the htmls: login, members, and signup.It also contains the same files names in js, which contain the code needed for the html files. The stylesheet folder contains the style.css and it only has the margin for the forms, everything else in there is commented. The JS files will wire up the events to the proper buttons using jQuery after the DOM is loaded. In the events, we will have jQuery used to create GET and POST calls to the proper API endpoints in the server.

# Routes Folder:

This folder contains two js. files. One is for the api-routes and the other for html-routes. The first, contains POST and GET routes to sign in and out options for the user. It will use the passport.authenticate middleware and it will send the user to members's page if he or she already has valid login credentials. The html routes render the html pages: the authenticated ones for users that are logged in, or back to the login page otherwise.

# Package.json:

The project name is "1-Passport-Example" and the main file is server.js. which is the starting file for node.js. A nodemon package was installed so there is no need to restart the server at every change in the code. There are other npm packages that were installed to be used to have access to the DB, to secure the password, to use express and sequelize (bcryptjs, express, express-session, mysql12, passport, passport-local, and sequelize).

# Server.js:

In this file added the required npm packages and it was created the express app setting it up on port 8080. The html and api routes were added as a links and the connection to the database server happens through sequelize.

# Instructions of how we could add changes in this project:

We could create a password generator for this user. The generator would prompt users and give options to choose from, such as : capital letters, lower case letters, number of characters that could be chosen and type of characters, like numbers and letters.