**Task:** to dissect the app, file by file, to convey how code within each file functions as well as how certain files/folders interconnect with one another

**Outcome:** a clear understanding of the website and its functionalities.

**High-level picture**

This app is consisting of three HTML pages in total (login.html, members.html, and signup.html). It illustrates login and signup feature works in applications. It requiring the user to input an email address and password to authenticate. If user doesn’t have an account, system will allow to create an account. This information of email and password are saved into MySQL database along with a surrogate primary key and datetime stamps for create and update (generated by sequelize).

Express-session will create a section and save the section id in browser as a cookie. The password is hashed with salt by bcryptjs and saved into database. While user login, if the account exists, system will allow to login and show home page with user name. The authentication part is handled using “passport”, and “passport-local” using local strategy.

Following is the directory structure of the application



**Homepage and session**

Upon connecting to website, express will use "express-session" to generate session and encrypt the session using secret.



Passport uses following functions:



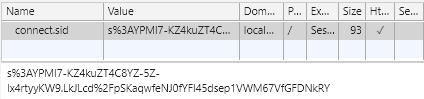
The user id (you provide as the second argument of the done function) is saved in the session and is later used to retrieve the whole object via the deserializeUser function.

serializeUser determines which data of the user object should be stored in the session. The result of the serializeUser method is attached to the session as req.session.passport.user = {}. Here for instance, it would be (as we provide the user id as the key) req.session.passport.user = {id: 'xyz'}

The first argument of deserializeUser corresponds to the key of the user object that was given to the done function (see 1.). So your whole object is retrieved with help of that key. That key here is the user id (key can be any key of the user object i.e. name,email etc). In deserializeUser that key is matched with the in memory array / database or any data resource. The fetched object is attached to the request object as req.user

At the end, Passport will push the session key into browser and it is kept in the cookie session as shown below. Each time user makes a request, this information is also send along with request and Passport will validate and authenticate the user.





**Signup**

First system ill check for user login and if user is not logged in, it will re-direct to signup page.



During signup, system will ask for email address (this is used as user name) and password. It will call an api.



As shown above, the API will call User model. In user model, there is a hook "beforeCreate", and this will convert the password into hashed before saving to system.

Hooks (also known as lifecycle events), are functions which are called before and after calls in sequelize are executed.



It uses “bcryptjs” to hash the password along with salt and it goes through hash cycle for 10 rounds. This value is saved in database.



**Okay, so its overarching purpose is to execute user login/registration with authentication?**

Precisely. This app is essentially a skeleton that can be repurposed time and time again when building more complex or involved apps. That, and it illustrates how sequelize can be used in conjunction with mysql2. Why? To show how user information can be stored in a SQL database securely while simultaneously reducing the amount of boiler-plate code required. For example, if someone were to hack into this database they wouldn’t have access to user password information since it is stored in its hashed form. Considering security implications is of utmost importance throughout any app development process.

**Config.json**

This file is for settingup db connection and environment.



The config.json file is where several instances are declared. These instances pertain to development, test, and production in this case. The development instance containing the passport\_demo database is used throughout the development process. The testing instance containing the database\_test database is used thereafter. The production instance is only used once one is certain of functionality and UI implications. The production instance is the live instance, or publicly accessible one, that users directly interact with. The dialect chosen for this program is MySQL but there are alternative dialects including but not limited to MSSQL, PostgreSQL, and SQLite. So what exactly does this file do? It creates instances for development, test, and production. These instances are used in conjunction with an ORM such as Sequelize.

**passport.js**



The passport.js file is located in the config folder. This file requires passport and passport-local. Passport is authentication middleware for node.js. Passport has one purpose and one purpose alone; authenticating requests. Passport achieves this via the use of plugins known as strategies. In this app, the passport-local is also declared as a dependency. This implies that this app utilizes a local strategy, which authenticates using username and password key-value pairs only. This local strategy is used in conjunction with express-session which keeps track of a users login status. This strategy requires a verify callback to accept user-input credentials.

  In the application, it is defined as usernameField: "email. Sequelize acts to serialize and deserialize the user across http requests to maintain authentication state.

**Middleware**

**isAuthenticated.js**

 This file serves as middleware that restricts routes (html routes) a user can visit if not logged in. If the user is not logged in, attempting to navigate to a restricted route redirects them to the login page.

**Models**



**index.js**

The index.js file is generated via running the sequelize CLI. This file is essentially boiler-plate code for Sequelize.

 The above code defines the db connection (which set of configuration need to be used).



The above code will read all the model files and save to db object.

**user.js**

user.js utilizes the bcryptjs npm. Bcrypt is used for password hashing/unhashing which is achieved via salt. Salt is similar to the concept of nonce. Bcrypt uses a “beforeCreate” addHook method to automatically hash the user input password prior to account creation. This ensures that a user password is only stored in its hashed form in the database which is imperative in the context of security implications. This file also utilizes sequelize to create the user model which is simply username and password key-value pairs (local strategy). This file also contains logic to ensure that a password entered matches the password associated with the corresponding username in the database.

**Routes**

**api-routes.js**

This file utilizes express (request, response) for post and get calls. These calls correspond to user login, user signup (registration), user logout, and acquiring user data. This file requires the use of passport as authentication middleware. If the user has valid login credentials they are able to access the members page. If the user fails to provide valid login credentials they will be sent an error. The response for acquiring user data contains the users email and corresponding id but not the password even in its hashed form.

**html-routes.js**

This file requires the use of the path npm and relative paths may be used to access HTML files (located in the public folder). The config/middleware.isAuthenticated.js file is required in this file to ensure that the user is logged in. Using the isAuthenticated middleware, if a user inputs values that do not correspond with an existing account they are redirected to the registration page.

Public



**signup.html and signup.js**

signup.html is for user to signup to the application and signup.js is for controlling actions in signup.html.

**login.html and login.js**

Login.html is for user to login to the application and login.js is for controlling actions in login.html.

**members.html and members.js**

members.html is for user to login to the application and members.js is for controlling actions in members.html.

**Server**

**server.js**

The server is where the PORT is defined, the express app is declared, and session is called. It also gets all the models and sync the models with db. It has all the routes and controls the api and html routes as per the requests it gets from the listener.

**npm dependencies**

