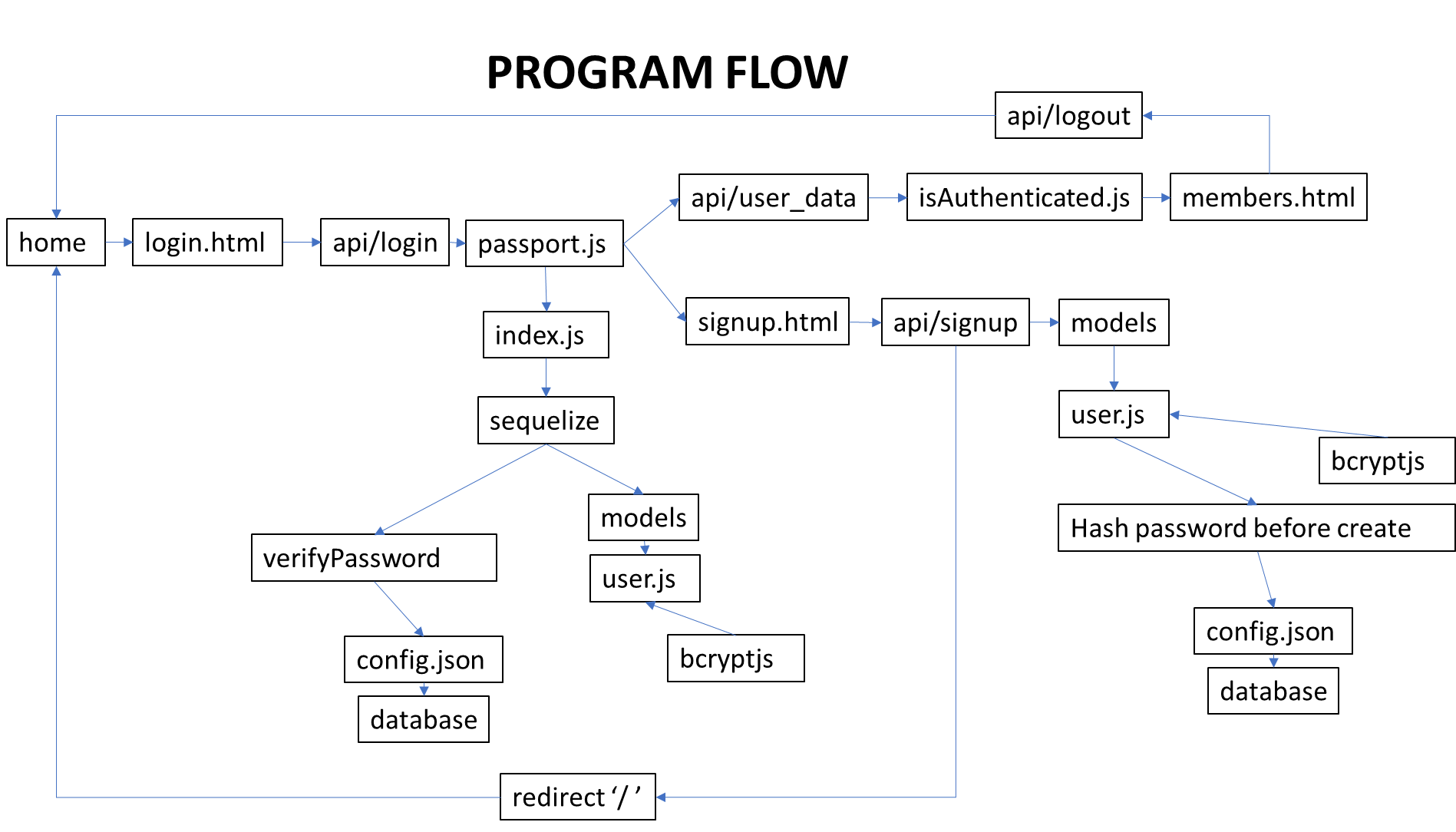
REVERSE ENGINEER

SEQUELIZE PASSPORT AUTHENTICATION



**How to use this guide**

This guide is a walkthrough tutorial on explaining how the sequelize passport works. This guide should be read in conjunction with the comments in the source code. The notes herein serve as supplementary to the comments in the code and do not attempt to teach programming principles. For further information, please refer to the official framework documentation.

**Application**

The application demonstrates the use of passport and sequelize frameworks in using a node express server.

A passport-demo SQL database has been created in order to understand and demonstrate how the program works. The application essentially has three pages; login page, sign up page and members only page to demonstrate passport authentication.

This authentication is achieved when registering a new user to be saved in the database and also when login into the application using user/member credentials.

**Technologies**

The following frameworks are used for authentication.

**Front End**

The application used basic html, css and javascript for the front end with 3 pages:

1. login page
2. sign up page
3. members page

The following front end frameworks used:

1. Boostrap
2. Ajax from the jquery library

**API End Points**

The front end pages have trigger events that invoke functions in the corresponding javascript files. These functions include ajax calls with GET and POST methods to hit the key api endpoints for our pages:

1. POST: api/login in login.html
2. POST: api/signup in signup.html
3. GET: api/user\_data members.html

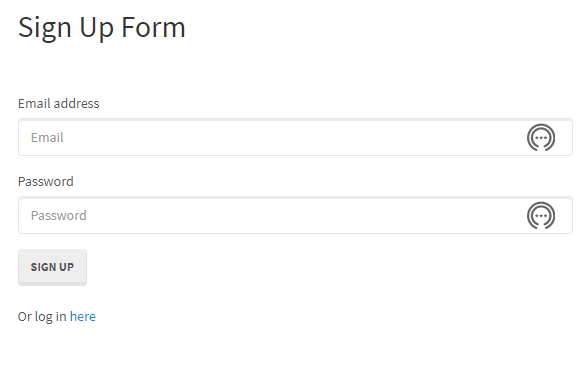
**Back End**

1. node expres.js
2. mySQL
3. Passport
4. Sequelize
5. Express
6. bcryptjs

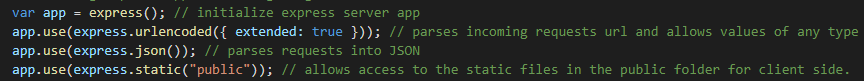
**Program Flow**

The program kicks off at the default page which is the sign up form assuming the user has not registered. Otherwise, the user is redirected to the members page. If the session has expired, then the registered user is redirected to the sign up page.

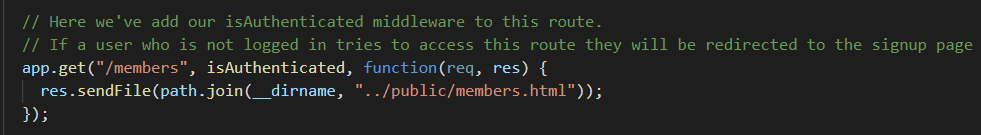
**Sign Up Page**



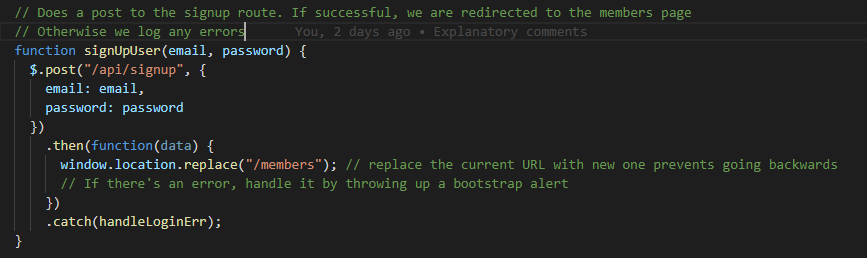
This page is served as the default home page and is part of static pages renders from the public folder. Using express middleware.



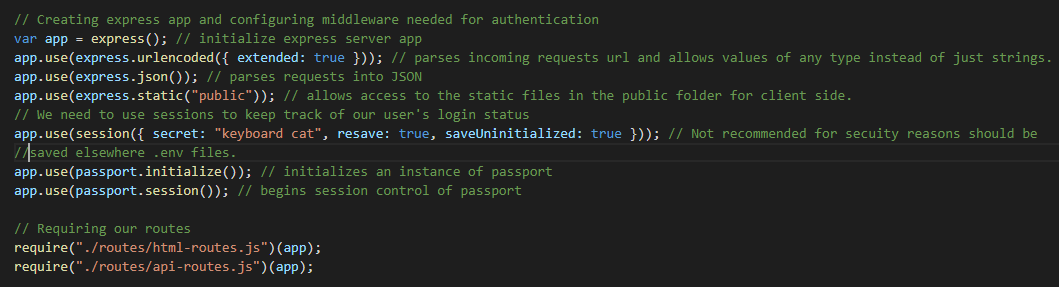
A user is always authenticated before displaying the members only page using the isAuthenitcated.js middleware is used for static html page routes to the members only page in the html-routes. This prevents an unauthenticated user with a valid ‘/members’ url from accessing members only page.



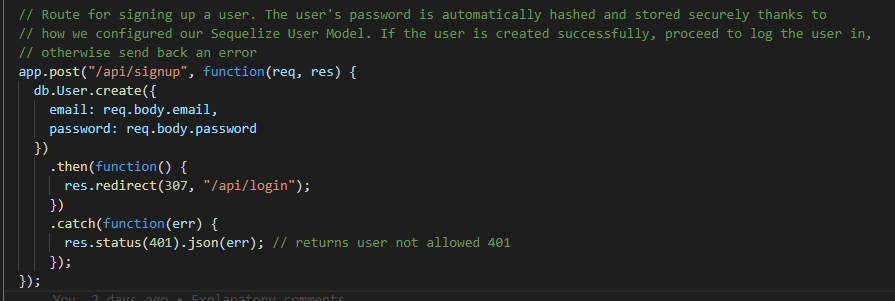
When the user signs up the details are email and passwords are captured and posted to the back end api end point ‘api/signup’ and redirects the new user to the members page. This is performed by the signUpUser function in signup.js.



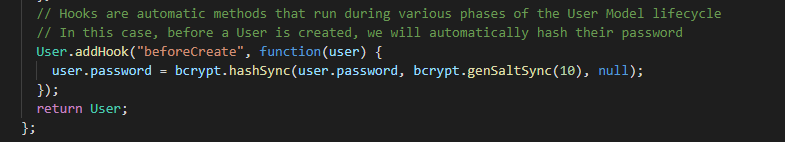
When the request hits the api/signup on the server, the server uses middleware defined in express server.js on the post request before passing the request data to the api router.

****

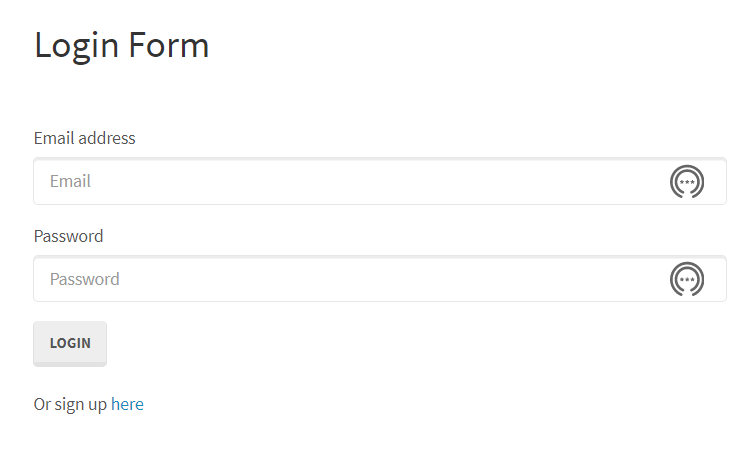
Thereafter, api-routes.js handles the endpoint and creates a new User using the new credentials in the request body.



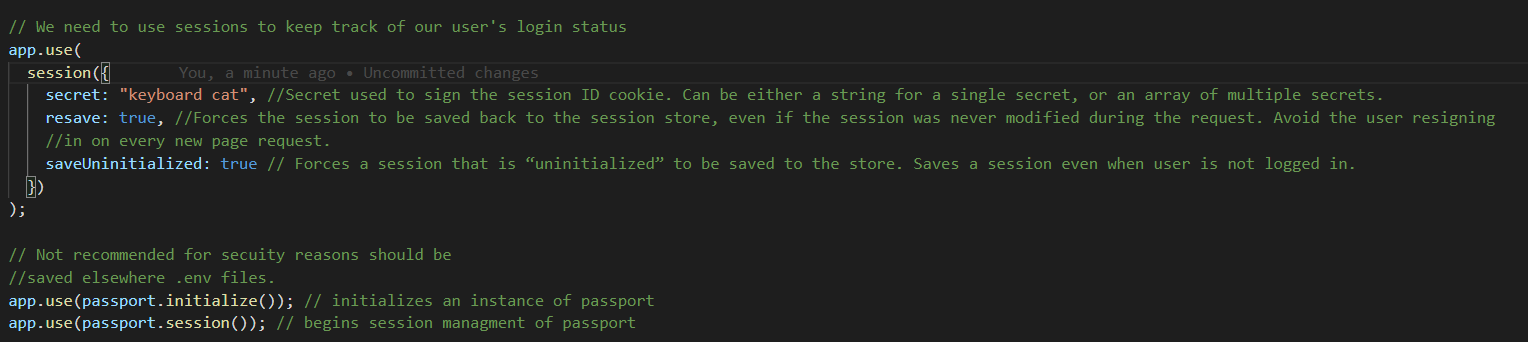
It is important to note that the password is never stored in its raw format but is ‘hashed’ or encrypted using the bcrypt library envoked in the ‘db.User.create’ method. The User class along with sequelize is imported into the api-routes.js via ‘db’. This is where the brcyptjs comes in to ensure that the password is hashed before User is created and stored in the mySQL database. The ‘beforecreate’ lifecycle hook is made available as function via the user.js model to execute this.

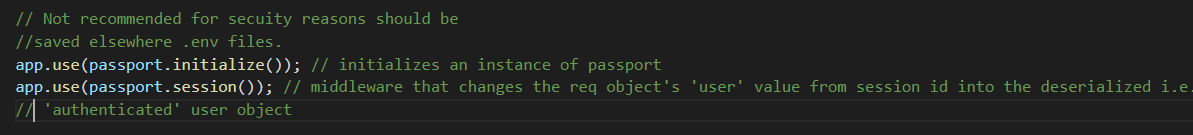


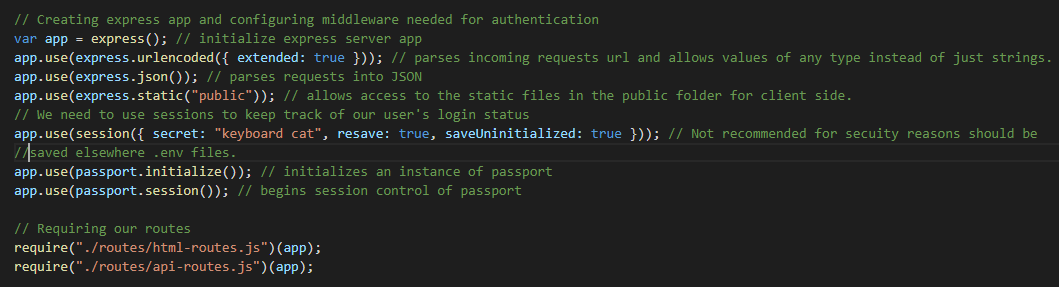
**Login Page**

****

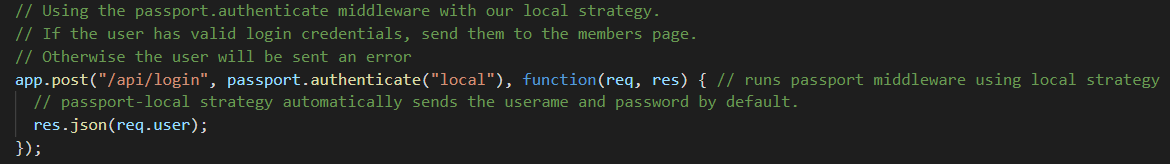
The user is prompted for credentials email and password before the data posted to the server. On the server, the request uses several middleware including the express-session and passport.



****

Thereafter, api-routes.js handles the /api/login endpoint and creates a new User using the new credentials in the request body. ****

In the api-routes.js, the passport.authenticate(“local”) middleware using the local strategy is invoked using the email and password from the request body as default parameters.



**Passport Middleware**

Thereafter passport middleware authenticates the user using the local strategy with username field and password field.

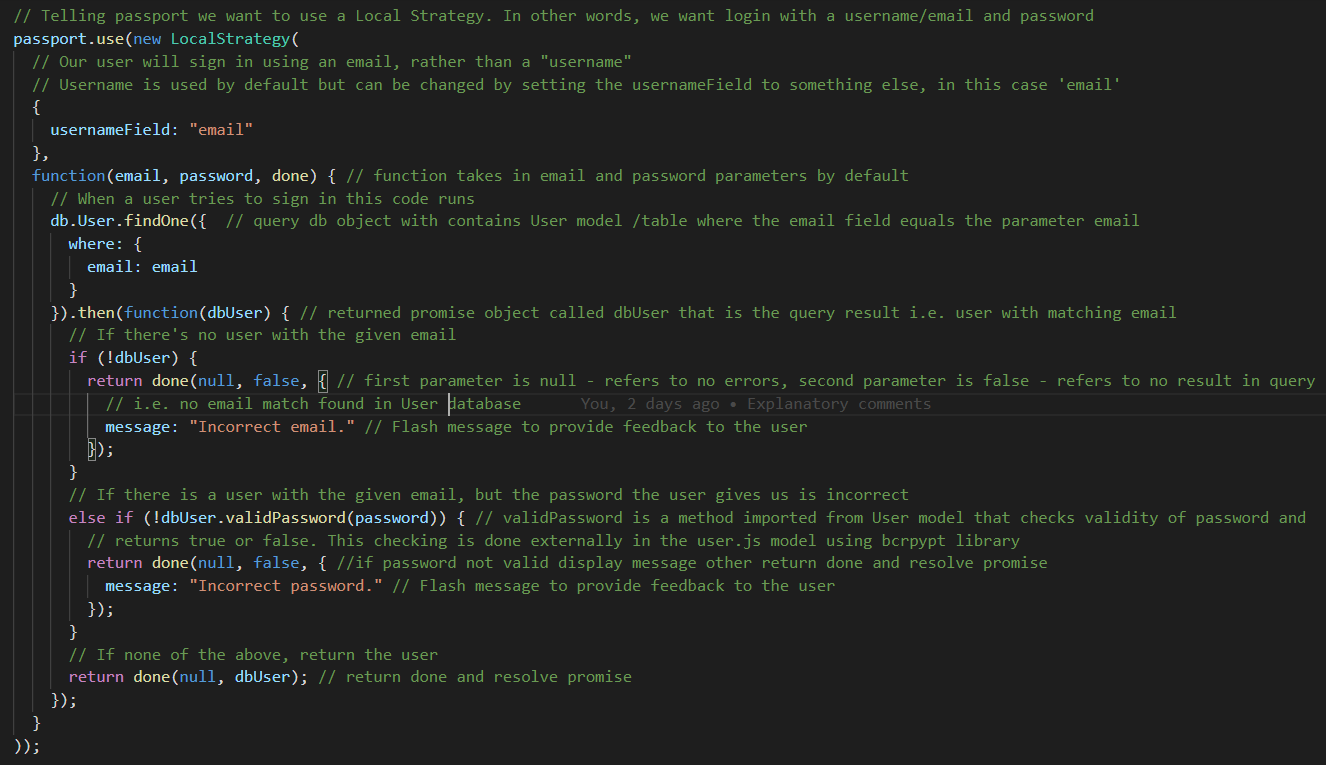
**Username validation**

By default, passport local strategy sets the usernameField to ‘username’ however, in this case it is set to ‘email’. This is validated by running a query in the database using the User model and sequelize method db.User.findOne().

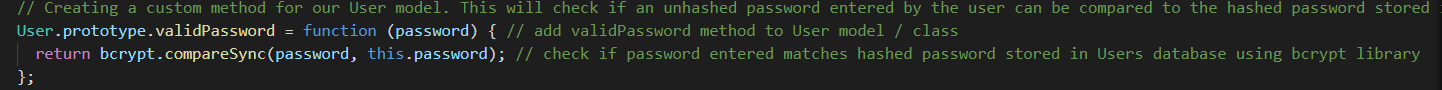
**Password validation**

At the same time the User password is validated using a validPassword() method that again is contained in the User model.

***Passport.js***



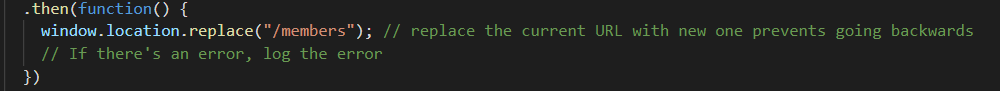
***User.js***



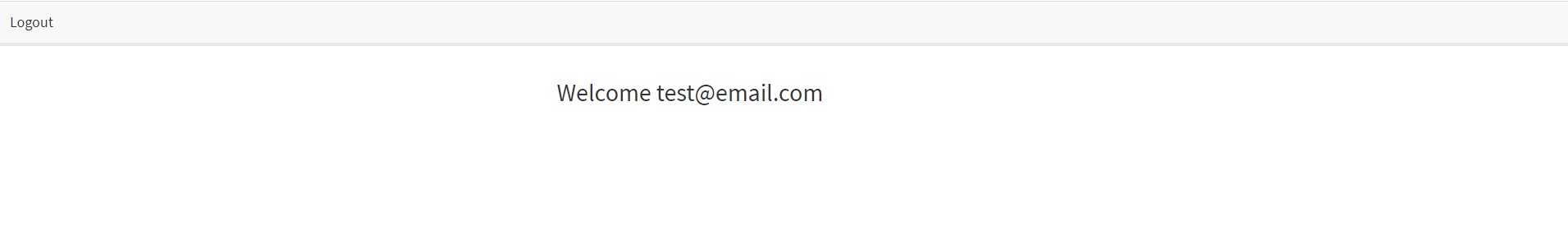
After passport middleware authentication control is passed back to the callback function in api/login endpoint returning/resolving the res.json with valid user object back to the front end.



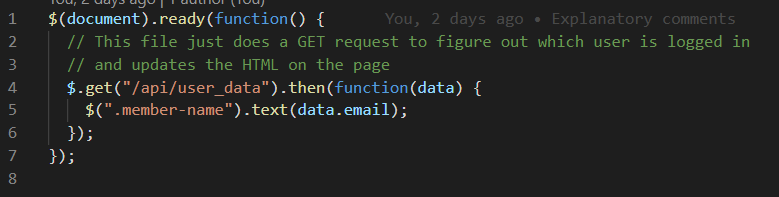
Once the authenticated user is verified, in the ‘login.js’ of frontend, the user is directed to the /members page.



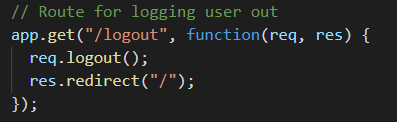
**Members Page**



Once logged in, the members.js uses the response user object email data to render the email on the page.



Also note that the page has a logout link that hits the ‘/logout’ endpoint in the api-routes.js. This executes the logout function and redirects the user to the home page.



When client requests the /members from the server, the server applies middleware including the isAutheniticated.js which verified the user before using the html-route end points. Thereafter, te server sends the static members. html file contained in the public folder.

