Reverse Engineering Code, homework 14

Description of all the files in the Develop folder:

1. **Config:**
2. **Middleware:**
3. **isAuthenticated.js: This is middleware for restricting routes a user is not allowed to visit if not logged in.**

* **Line 2:** If the user is logged in, continue with the request to the restricted route.
* **Line 8:** If the user isn’t logged in, redirect them to the login page.

1. **Config.json: This file is to identify which database we want to connect to and to setup the username, database name, password and the host IP.**
2. **Passport.js: Telling passport we want login with a username/email and password.**

* **Line 7:** Telling passport we want to use a Local Strategy. In other words, we want to login with a username/email and password.
* **Line 8:** Our user will login in using an email, rather than a “username”.
* **Line 14:** When a user tries to sign in, this code runs
* **Line 20:** If there’s no user with the given email, it will send a warning
* **Line 26:** If there’s a user with the given email but the password is incorrect, it will send another warning
* **Line 32:** If none of the above, return the user
* **Line 40:** SQL needs to serialize and deserialize the user, in order to help keep authentication state across HTTP requests. This part is the boilerplate needed to make it all work
* **Line 49:** Exporting out configured passport

1. **Models:**
2. **Index.js:**

* **Line 3-5:** requiring (fs), (path) and (sequelize)

1. **User.js: Requiring a password hashing. Using the bcryptjs version as the regular bcrypt module sometimes causes errors on Windows machines**

* **Line 4:** Creating our user model
* **Line 7:** The email can’t be null and must be a proper email before creation
* **Line 16:** The password cannot be null
* **Line 22:** Creating a custom method for our User model. This will check if an unhashed password entered by the user can be compared to the hashed password stored in our database

1. **Public:**
2. **Js:**
3. **Login.js:**

* **Line 3-5:** Getting references to our form and inputs
* **Line 8:** When the form is submitted, we validate there’s an email and password entered
* **Line 20:** If we have an email and password we run the login User function and clear the form
* **Line 26:** login User does a post to our “API/login” route and if successful, redirects us the the members page
* **Line 35:** If there is an error, log the error

1. **Members.js:  This file just does a GET request to figure out which user is logged in and updates the HTML on the page**
2. **Signup.js: Line 3-5: Getting references to our form and inputs**

* **Line 8:** When the signup button is clicked, we validate the email and password are not blank
* **Line 19:** If we have an email and password, run the sign up user function
* **Line 26:** Does a post to the signup route. If successful, we are redirected to the members page. Otherwise, we log any errors
* **Line 35:** If there’s an error, handle it by throwing up a bootstrap alert

1. **Stylesheets:**
2. **Style.css: Setting a top margin to the sign-up and login forms**
3. **Login.html: This is the form of the login main page**
4. **Members.html: This is the form of the members main page**
5. **Signup.html: This is the form of the sign-up main page**
6. **Routes:**
7. **Api-routes.js:**

* **Line 2-3:** Requiring our models and passport as we’ve configured it
* **Line 9:** the passport.authenticate middleware with our local strategy. If the user has valid login credentials, send them to the members page. Otherwise, the user will be sent an error
* **Line 16:** Route for signing up a user. The user’s password is automatically hashed and stored securely thanks to how we configured our SQL User Model. If the user is created successfully, proceed to log the user in. Otherwise, send back an error
* **Line 30:** Route for logging user out
* **Line 36:** Route for getting data about our user to be used client side
* **Line 39:** The user is not logged in, send back an empty object
* **Line 43:** If the user is logged in, send back the user’s email and id

1. **Html-routes.js:**

* **Line 2: Requiring path to so we can use relative routes to our HTML files**
* **Line 5:** Requiring our custom middleware for checking if a user is logged in
* **Line 11:** If the user already has an account send them to the members page
* **Line 19:** If the user already has an account send them to the members page
* **Line 27:** This line is to add our is Authenticated middleware to this route. If a user who is not logged in tries to access this route they will be redirected to the signup page

1. **Package.json: this file holds various metadata relevant to the project. This file is used to give information to npm that allows it to identify the project as well as handle the project’s dependencies**
2. **Server.js:**

* **Line 2-3: Requiring necessary NPM packages**
* **Line 5:** Requiring passport as we have configure it
* **Line 8-9:** setting up ports and requiring models for syncing
* **Line 12:** creating express app and configuring middleware needed for authentication
* **Line 17:** we need to use sessions to keep track of our user’s login status
* **Line 22-23:** requiring our routes
* **Line 26:** Syncing our database and logging a message to the user upon success.