14: Reverse Engineering Authentication

FILE STRUCTURE

Config
- Middleware
  - isAuthenticated.js
- Config.json
- Passport.js

Models
- Index.js
- User.js

Public
- Js
  - Login.js
  - Members.js
  - signup.js
- Stylesheets
  - style.css
- Login.html
- Members.html
- signup.html

Routes
- Api-routes.js
- Html-routes.js

Schema.sql

Package.json
server.js

Dependencies used:

- Express
- Express-session
- Mysql2
- Sequelize
- Passport
- Passport-local
- BCRYPTJS

How to use the application:

1. Create a mysql database named “passport_demo”
2. In the config folder, open config.json and update your mysql DB password
3. Using the terminal in the current repo use “npm i” to install the provided dependencies
4. After everything is installed, run the server by typing “node server.js” into the terminal
5. Using the browser open localhost:8080

Files Explained:

--Config--

- **Middleware**
    - **isAuthenticated.js**
        - This file consists of middleware for restricting routes the user is not allowed to visit if not logged into the application. If the user isn’t logged in, it will redirect to the home page.

- **Config.json** allows developer to insert their database password to sync the database with the application

- **Passport.js** requires passport, passport-local and the models folder. Within this file contains logic that finds the user email in the database “dbUser” by using findOne(). This file will also validate password and if password is incorrect return a message “Incorrect password”.

--Models--

- **Index.js** provides a connection to the database and imports the user login data. This file requires sequelize and config/config.json along with the standard fs.

- **User.js** requires bcrypt and sequelize. This will create our user table data in our mysql workbench. Also, validates the password to not be null.

--Public--

- **Js**

  - **Login.js** creates the logic that retrieves the user input for the login form and posts to the "api/login" route when successful, will redirect the user to the members page. This js logic ties the attached form the the login-html page and submits it to the backend.

  - **Members.js** will "get" the data from the database and updates the html page.

  - **Signup.js** does the same as the login.js page but this time it will register the new user into the database

- **Stylesheets**

  - **Style.css** is the stylesheet. For this app, it adjusts the margin top to 50px for the signup and login form.

Login.html contains the html page which will allow the user to login to the application.

Members.html contains the members html page for returning the database to the browser.

Signup.html contains a html page which will allow the user to sign up for the application.

--Routes--

Api-routes.js requires models and config/passport. The api routes provides routes for logging in, signing up, and displaying users information from the database on the application.

Html-routes.js requires config/middleware/isAuthendicated. The html routes unfold the url options and performs a "get" from the CRUD operators that will go and retrieve the specific html page for the user. In this case, the user has three options:

1) “/” which is the home default route
2) “/members” which is the sign up page
3) “/login” which is the login page

Server.js requires express, express-session, passport, html-routes,and api-routes. Establishes a connection to the PORT and syncs our database into our localhost:8080.

Schema.sql creates the database “passport_demo”.

Package.json contains all dependencies information, node modules, versions of dependencies used.