Homework 14,
Reverse Engineering Authentication

TABLE OF CONTENTS

I. ***DEVELOP FOLDER >***

1 ***CONFIG >***

1A MIDDLEWEAR> isAuthenticated.js: Passing restrictions on what user's can and cannot see based on if they are logged into the application.

DEPENDENT ON:
1. html-routes.js If the user is not logged in they will be redirected to the login homepage requesting their credentials.

1B CONFIG.JSON: Establishes global values within the config folder, loads environment specific configuration and overrides defaults. It is an npm package and can be installed npm i config.json

1C PASSPORT.JS: Passport is an npm package (install it using npm i passport). It is a software it is a package that authenticates a log-in/password and decides what happens when if it succeeds or fails. This file gives the package the parameters for it to work off of.

DEPENDENT ON:
1. models folder: it calls on functions and variables brought in from the folder. (line 7 -35)

2 ***MODELS FOLDER >***
The models folder is the way data is supposed to be structured, Sequelize allows us to structure the date using javascript. The models folder is where the developer turns Sequelize into a constructor.

2A > INDEX.JS: It gives the object 'db' parameters. If db is an existing user, it will allow for login or else, it will query the database for username, password to determine if the user is valid. Following this step; it 'reads' the config file and syncs it by using a conditional statement. using a function to check if the login credentials are valid. (20 - 43)

DEPENDENT ON:
1. DotEnv: an npm package (use npm i dotenv) that allows developers to manage configuration of applications separate from the code base. It protects passwords and API keys, as well as other valuable information public users should not acquire. While hiding sensitive information it allows the code to function properly. (line 16)
2. Config Folder: it relies on the config folder, in particular, the config.json
3. Sequelize an npm package that manages a user's database, as well as allowing for queries to be made to that database. It can be installed using npm i mysql

2B > USER.JS: Is a constructor that is a new instance of the Sequalize package. Following the constructor the developer ran 'sequelize.define' function. This function creates the database and names the columns along with the rules each column has. (line 7 - 20)

DEPENDENT ON:
1. Models>index.js: checks credentials of user.

I. *DEVELOP FOLDER >*

PUBLIC FOLDER >

3 The public folder is what the user sees when they interact with the website. Files that go into the public folder are front end files, like: CSS, html and javascript. The code included in the public folder is shared, so it should not include sensitive code information or passwords.

DEPENDENT ON: the whole public file is dependent on the installation of npm i express. Using express middlewear (see Server.js for more details) allows the developer to determine a folder in the directory that contains all the front end code.

3A PUBLIC > JS (javascript):

A separate folder with in the public folder that contains *only* front end javascript.

3B > LOGIN.JS: Establishes variables and parameters that tell the login function how to work.

DEPENDENT ON:

1. login.html: links the login page to this to login.js file via a script tag. Most importantly, it includes a CDN script tag for an Axios node package. The CDN link allows this file to run a request to the database for the user's login information (line 25)

3C > MEMBERS.JS: retrieves the logged in user's information (their login and password). Validating the login and password allows the user to stay logged in without having to re-do the login and password process each time. It is missing code that makes the 'logout' button functional on member.html.

DEPENDENT ON:

1. members.html: links members.html file via a script tag. Most importantly, it includes a CDN script tag for an Axios node package. The CDN link allows this file to run a request to the database for the user's login information (line 25)

3D > SIGNUP.JS: establishes variables, parameters and functions for new users to build login credentials.

DEPENDENT ON:

1. signup.html: links members.html file via a script tag. Most importantly, it includes a CDN script tag for an Axios node package. The CDN link allows this file to run a request to the database for the user's login information (line 25)

4 PUBLIC > STYLESHEETS > STYLE.CSS:

Style sheet allows for the separation style variables and the code. It keeps the code more streamlined. The style sheet contains the visual layout of the page. It can include items, such as: background color, font, border, image, content alignment and many more.

5 PUBLIC > HTML:

A folder within the public folder that only contains html files.

I. ***DEVELOP FOLDER >***

5A	> LOGIN.HTML: Builds the features for the login form. Features visible on this page are: the titles, user input boxes, text and buttons. DEPENDENT ON: 1. js>login.js: The login.js controls the functionality of the login.html page. Without the login.js the html page would have a non-functional button and would not validate a user's email or password. These functions are required to send the user into the member's page. 2. stylesheets>style.css: The style.css creates the margin at the top of the login page, meaning it creates separation from the top of the page and the user required information. It is acquired at the head of the html page (line 9) 3. Bootstrap: is a styling library that is acquired in the head tag of the page (line 8); it preformats many visual aspects of the html, limiting the need for a lot of code on stylesheet. 4. NPM Axios: At the bottom of the page there is a script tag, which links to an outside node package. In order for login.js to retrieve and validate login information on the backend.
5B	> MEMBERS.HTML: Is the 'homepage' for members once their login is validated. Include a navigation bar that allow users to logout. DEPENDENT ON: 1. js>members.js: Allows the user to stay logged in. Missing from this page is the functionality to make the logout button work. 2. stylesheets>style.css: The style.css creates the margin at the top of the login page. It is acquired at the head of the html page (lin 9) 3. Bootstrap: is a styling library that is acquired in the head tag of the page (line 8); it preformats many visual aspects of the html, limiting the need for a lot of code on stylesheet. 4. NPM Axios: At the bottom of the page there is a script tag, which links to an outside node package. In order for members.js to retrieve and validate a user's credentials.
5C	> SIGNUP.HTML: Allows for new user's to sign up for the site. It includes the user input boxes, titles and buttons. DEPENDENT ON: 1. js>signup.js: Allows a new user to enter their email and create a password. 2. stylesheets>style.css: The style.css creates the margin at the top of the login page. It is acquired at the head of the html page (link 9) 3. Bootstrap: is a styling library that is acquired in the head tag of the page (line 8); it preformats many visual aspects of the html, limiting the need for a lot of code on stylesheet. 4. NPM Axios: At the bottom of the page there is a script tag, which links to an outside node package. In order for signup.js to retrieve user's input and create a new account.
6	***ROUTES FOLDER >*** The routes folder contain files that take requests from the user (http requests) and forward the request to the backend.

I. ***DEVELOP FOLDER >***

6A	>API-ROUTES.JS: make the queries to the database, it will fetch the information and return it to the frontend DEPENDENT ON: 1. Models folder: brings in the models folder and set it to a variable db 2. Config folder>passport.js: will validate the user's login information. If the information is validated it will allow them to goto Members page, if it is not validated it will return the user an error. Set the folder to a variable passport
6B	> HTML-ROUTES.JS DEPENDENT ON: 1. Public folder:
7	***PACKAGE.JSON >*** Upon installing dependencies for the file 'package.json' will confirm if the installations were successful. If the installations were installed they will be visible in this file.
8	***README.md >*** Is a text file that includes an explanation of the project and the dependencies it requires. It is a step by step guide in how to download the application and what to install. It includes a short giph or image that displays the application's functionality.
9	***SERVER.JS >*** Is the backbone file to the entire application as it unites all the pieces of the application to run on a server. It gives the developer the ability to handle when a request is made and the ability to decide how to respond to that request. DEPENDENT ON: 1. Node Package Express (install using npm i express in terminal), express implies the web development process because it is built on top of node module. 2. PUBLIC folder (line 11-15) 3. CONFIG folder (line 4-5 & 17-19) 4. ROUTES Folder (lines 21 - 23)

II. ***DIRECTIONS TO ADD CHANGES >***	
	1. Create a new repository (directions vary on your repository platform)
	2. Clone the new repository to your local device - open the computer terminal and step into the directory (folder) where the project will live - step into directories by using the command 'cd' directory name ***it needs to be typed exactly as it appears in your computer,* Example: cd Desktop** - once in the correct directory use the command 'git clone: *paste link from repository website***
	3. Drag the project folder into VS code; this will ensure the entire folder and the directory are open and not just single files.
	4. Once the folder opens in VS code, open the terminal (Terminal, new window), to ensure all node packages the project requires have been installed, type the below command into terminal - npm I (return)
	5. Make changes in the html files - formatting, or any user changes can be made here (examples: adding text, images or more links) - you will have to edit each .html page separately - you can also add html pages here, be sure to add the name with .html - keep track of the id's you delete or change as they may correspond to js functions / commands. If you change them, be sure to update them in the .js file as well
	6. Make changes in public>js folder - change any js functions to match their new id (login.html changes will need to be updated on login.js, members.html corresponds to members.js, signup.html corresponds to signup.js) - if you create new .html files, be sure to create a corresponding .js file here that will include the routes and any additional functionality the .html page will need
	7. If you add new html, you will need to add a corresponding route, that will control the request and response of information to that page. - if a new route is created, be sure to 'require' it on the server.js file. Reference lines 22-23 for examples
	8. To change aesthetics goto public>stylesheets>css
	NOTES:
	If you wish to keep the functionality of the login, sign-up and out, focus on the public folder and routes, as models and config functionality is already established.