**Assignment 14 – Sequelize Homework: Reverse Engineering Code**

**Instructions:**

* Inside the “**Develop**” folder, the folder structure of this project follows the MVC layout (model–view–controller), which is a software design pattern.
* The **model** manages fundamental behaviors and data of the application. It can respond to requests for information, respond to instructions to change the state of its information, and even to notify observers in event-driven systems when information changes. This could be a database, or any number of data structures or storage systems. In short, it is the data and data-management of the application.
* The **view** effectively provides the user interface element of the application. It will render data from the model into a form that is suitable for the user interface.
* The **controller** receives user input and makes calls to model objects and the view to perform appropriate actions.
* The “**Develop**” folder contains a “models” folder – as well as a “config,” “public,” and “routes,” folder, although in this case there are no “controllers” or “view” folder, as there are in turn no .sql files for database and seeds files that would be contained inside a “view” folder.
* **“config” Folder:** where the configuration files are stored.
* **passport.js:** A file linking the website to local storage.
  + At the top of the “**passport.js**” file inside the “config” folder, are variables which indicated the libraries that we require in order for the application to run.
  + In the variable “passport,” we require “passport,” which is authentication middleware for Node.js. Middleware is software that acts as as bridge between an operating system or database and applications, especially on a network. The variable LocalStrategy instead requires “passport-local,” which is a passport strategy for authenticating with a username and password. In the variable “db,” we require the contents inside the “../models” folder and file-path.
  + Line 7 tells passport that we want to use a Local Strategy, and login with a username/email and password.
  + Line 10 states that users will sign in through the log-in form with an email rather than a username.
  + Lines 12-16 are a callback function that requires an email and a password, and which runs once the user has signed in.
  + Lines 18-24 are another callback function with an if statement, which states states that if there is no user with the given email, then to return the message “incorrect email.”
  + Lines 25-30 are an else if statement that states that if the user has give a valid email, but an incorrect password, then the message “Incorrect password.” will be returned.
  + Line 32 states that if none of the previous scenarios occur, then to return the user.
  + Lines 40-46 state that in order to help keep authentication state across HTTP requests, Sequelize needs to serialize and deserialize the user, which is a sort of boilerplate code needed for this to work.
  + Line 49 exports our configured passport for other files to use.
* The “**config.json**” file inside the “config folder” loads the default configuration file, with configuration settings, loading an environment specific configuration file, which overrides defaults. It then uses environment variables and command-line arguments to override data from configuration files. It also contains information on the database, including the username and password.
* **middleware:** Inside the “config” folder, there is also a “**middleware**” folder; middleware is software that acts as a bridge between an operating system or database and applications.
  + **isAuthenticated.js:** The “**middleware**” folder contains an “**isAuthenticated.js**” file, with middleware for restricting routes a user is not allowed to visit if not logged in. If they are logged in, then the request will be continued; however, if the user is not logged in, they will be redirected to the log-in page. Thus, the file checks whether the user is logged in or not.
* **“models” Folder:** The “**models**” folder stores the models, which manage the fundamental behaviors and data of the application
  + **index.js:** The “**index.js**” fileserves as the router; it sets up the database through ORM and links it to the server. It requires the ‘fs’, ‘path’, sequelize’ modules, and the “/../config/config.json” file.
  + **user.js:** The “**user.js**” file requires bcrypt for password hashing. It is where the user model for login requirements is defined; the e-mail must be a STRING, and the field cannot be null, while the e-mail must also be unique and validated, while the password must be a string and its field also cannot be null.
* **“public” Folder:** The **public** folder contains the front-end portion application, namely the three html pages, “login.html”, “members.html,” and “signup.html,” and a “js” and “stylesheets folder.” The html files create the basic structure of the application.
* “**login.html**” is a file that allows already registered users to log-in. It contains a link to the “style.css” file in the head and a link to the “login.js” file at the bottom of the body. The page contains a form with user input boxes for their email and password, and a login button, as well as a link that redirects users to the “signup” page if they do not have an account.
* “**signup.html**” is a file that allows users to sign up; it contains a link to the “style.css” file in the head and a link to the “signup.js” file at the bottom of the body. The page also contains a form with user input boxes for their email and password, and a login button, as well as a link that redirects users to the “login” page if they already have an account.
* “**members.html**” is a welcome page for members who have logged in. It contains a link to the “style.css” file in the head and a link to the “member.js” file at the bottom of the body. It contains a heading which says “welcome,” and a “logout” button, which redirects users to the “login” page.
* “**public/stylesheets**”: A folder for the css file(s), which styles the html file(s).
  + “**style.css**”: A css file which styles the html files; in this case, the css file is very basic, and merely adds a margin on the top of the “signup” and “login” forms.
* **public/js:** Inside the “**public/js**” folder are three front-end javascript files that relate and are linked to their respective html pages, and which are responsible for the functionality of the html files: “login.js”, “members.js,” and “signup.js.” The data from these files is also used to populate the databases in the “users.js” and “index.js” files
* The “**login.js**” file ensures that when the submit button is clicked, that we validate whether an email email and password entered, and the form is subsequently cleared. If the login data is correct, then a successful post request is made to the “api/login” and the user is redirected to the “/members” page.
* The “**signup.js**” file ensures that when the signup button is clicked, that we validate whether an email and password have been entered, and the form is subsequently cleared. If the login data is correct, then a successful post request is made to the “api/signup” and the user is redirected to the “/members” page; otherwise an error alert is shown..
* The “**members.js**” page makes a get request to understand which user is log in and updates the “members.html” page
* **“routes” Folder:** This folder contains all of the files with the route definitions for the application. It also links the front-end with the back-end
  + **api-routes.js:** The **“html-routes.js**” file organises the routes for storing user login-data. It requires the “../models” folder and the “..config/passport.js” file. The file uses a POST method to store the user’s email and password when signing them up; it also uses a GET method to log the user out and redirect them to the login page, and to retrieve and return information on the user (e-mail and id).
  + **html-routes.js:** The **“html-routes.js**” file organises the routes for the various html pages. It requires the path module, which provides utilities for working with file and directory paths. Based on the information of the “../config/middleware/isAuthenticated” file, which informs us whether a user is logged in or not, the paths will send the user to the “members.html” file/page if they already have an account (“../public/members.html”) or to the “signup.html” file/page if they do not have an account (“../public/signup.html”); if no information is entered, then the user will be redirected to the “login.html” file/page.
* **“node modules” Folder:** contains all the node modules that have been installed, which are needed to run the application.
* **package.json:** The “**package.json**” 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. Thus, when we run “npm install” inside the develop folder in the terminal window, all the necessary node modules and libraries need to run the application are installed inside the “mode modules” folder
* **server.js:** In the back-end portion of the project, we have the “**server.js**” file, which builds the server, which receives requests from and returns information to the client. The file also connects the code to a specific server.
* At the top of the file (lines 1-23), we require the necessary npm packages (lines 1-5), as well as the “./config/passport” file, and the “./models” folder (line 9), and the “./routes/html-routes.js” and “./routes/api-routes.js” files (lines 21-23). We also set up the port on line 8 (8080),
* In lines 11-30 we build and set-up the server.
* In lines 11-15, we create an express app and configure the middleware needed for authentication.
* In lines 16-19 we use sessions to keep track of a user’s login status.
* Finally, in lines 25-30, we synch our database and log a message to the user upon success once the connection to the server is established.

**Adding Changes to This Project:**

There are a number of ways in which this project could be expanded. As the files are interconnected, one must pay attention when modifying certain files, to ensure that the changes have been applied to all applicable files, and that they have been tested to ensure that the code is still functional and not broken.

One of the most obvious manners in which this starter code could be developed would be by adding additional pages to the application as .html files in the “public” folder, or front, and redirecting users to these pages through the “html-routes.js” file.

Another possibility would be to add a delete function which would allow users to delete their account. This would require a delete button in the “members.html” file page, and an onClick event handler in the “members.js” file, which would trigger an alert, and a DELETE method in the “api-routes.js” file upon confirmation, and which would subsequently remove the user’s information from the Sequelize database.

If we wanted to use handlebars to generate the .html pages, they would be contained in a “layouts” folder, inside a “views” folder, which is in turn inside the level directly below the “Develop” folder.