Week 14 Homework Develop Folder

**Package.json**

Package files show dependency and terminal commands for to run and watch the app server.js. The Package files designated server.js as “main” file. The project has express session with “bcryptjs”, mysql2, passport, passport-local, and sequelize as dependencies.

**Server.js**

Server.js requires npm packages for express, express-session. It also requires passport for configuration from a different config folder in the project.

It sets up port 8080 and then requires the database model from the models folder.

It sets up the express app and configures the middleware so the app can use the asset from the public folder and also for authentication.

It creates a season to keep track of user’s login status.

It requires routes from other files for html and api.

It syncs database and log message to the user when the database successfully connect.

**Config folder**

1. **Config.json**

There are 3 environments within the Config.json file: Development, Test, and Production.

They all have same username, password, host, and dialect; however, the databases are different based on environment.

1. **passport.js**

This file uses passport and local strategy passport packages as well as the database models.

It tells passport to use the local strategy to allow login with username and/or email with password. When a user tries to sign in, it looks in the database for user’s email; an incorrect email message shows up when it cannot find the email. Next, it looks for correct password; an incorrect password message shows up if password is not valid. Otherwise, it returns the user associate with that email address from the database. Then passport serialize user database and then deserialize the user to help keep authentication state across HTTP. The last command is to export configured passport.

**Middleware folder**

1. **isAuthenticated.js**

This is middleware for restricting routes a user is not allowed to visit if not logged in. If the user is logged in, continue with the request to the restricted route. If the user isn’t logged in, redirect them to the login page.

**Models folder**

1. **Index.js**

This index file uses strict mode. It requires fs, path, sequelize, basename,config, database, and it is in the development environment.

If config tells it to use the environment variable, it initializes a new sequelize with the process environment. Otherwise, the new sequelize will use config database, config username and config password.

The file stream looks for file inside the directory from \_dirname. The “for each” command uses the import file information to creates a sequelized model for the database.

The Object keys database goes through each of the database models to find association with model name.

At the end, it exports the database module.

1. **user.js**

This file requires bcrypt for password hashing. It creates user model with sequelize. The email cannot be null and must be a proper email address. The password cannot be null also. It also has a custom method for validating hashed password from the user model.   
The hooks are automatic methods that run during various phases of the user model lifecycle. Before the user is created, the password is hashed.

**Public folder**

1. **Javascript (js) folder**

**I. login.js**

When the user submit the loginform, it validates email and password exist. If the mail and password are there, it will run the login function and clear the email and password inputs. The login function makes a post in the api login route using email and password. The next function is to log error if there’s an error with catch function; otherwise it goes to members page.

**II. members.js**

This file uses the GET command to api user data and update html on the page with member name and email.

**III. signup.js**

When the user submit the signup form, it looks for email and password inputs and validate them. It tries to make an user object. If the email and password inputs are blank, then it goes back. If it is populated, then run the signup user function. Then it clears the information from the inputs. The signup user function make the post request with user data to api sign up route and redirects to the window’s member page. If there’s an error, the catch command will make an alert with error show up.

1. **Style.css**

It is a simple style for signup and login with margin top of 50px.

1. **login.html**

This html specified as english “en”. It has the title of Passport Authentication with character sets “UTF-8”. It is a boiler plate from the beginning with bootstrap link and also attach to stylesheet. It has a navbar and a login in form inside the container. The login form has 2 inputs, one email, and one for password, with a submit button. It also offers a link to the sign-up page.

1. **Members.html**

This html specified as english “en”. It has the title of Passport Authentication with character sets “UTF-8”. It is a boiler plate from the beginning with bootstrap link and also attach to stylesheet. It has a navbar and a logout button inside the navbar. There is a welcome message that gets filled in by the user name.

1. **signup.html**

This html specified as english “en”. It has the title of Passport Authentication with character sets “UTF-8”. It is a boiler plate from the beginning with bootstrap link and also attach to stylesheet. It has a navbar. There’s a Sign Up Form with input for email address and password with submit button. It also has a place for error message to display and a login link.

**Routes folder**

1. **Api-routes.js**

It requires models and passport as they configure. The api login route uses post to authenticate the user data with the local strategy to validate login credentials. Otherwise, the user will have error message sent. The api signup route also uses post to create a user in the database and either redirect to the login in or 401 error page. The logout redirect to the main after logging the user out. If the api user data route sends the user and id if the user is logged in; otherwise, it send back empty object.

1. **html-routes.js**

It requires path to the related routes to HTML files. isAuthenticate is a custom middleware for checking a user login. It uses get function to redirect member if they are login; otherwise it displays the signup.html. On the login route, if the user already has an account, it send them to the member page; otherwise it displays the login page. The members route uses isAuthenticated middleware to prohibit non login users.