Unit 14: Sequelize – Homework – Reverse – Engineer Tutorial
Julie Ann Schaub
University of Kansas Coding Bootcamp

1. User will need to create database passport_demo in MySQL Workbench
2. Install the required dependencies in terminal
    - i. npm i
        1. This process creates the package.json and package-lock.json files when run
    - ii. npm i express (required in the server.js file)
    - iii. npm i express-session (required in the server.js file)
    - iv. npm i passport (required in the passport.js folder in the Config folder)
    - v. npm i fs (required in Models index.js file)
    - vi. npm i path (required in Models index.js file)
    - vii. npm i bcrypt (required in Models index.js file)
    - viii. npm i sequelize if not installed already globally (required in Models index.js file)
3. server.js File
    - a. This page requires the passport page, which is part of the config folder, and links to it here – var passport = require("./config/passport"); (see 4.c. for functionality of this file)
    - b. PORTs are configured to use 8080
        - i. To run the app the user will first type node server.js in terminal and once the database is confirmed to be running (see 1.g.) type http://localhost:8080 in the url in a web browser – this allows the user to monitor changes and make sure the app is running appropriately before deploying
        - ii. The models folder is required here and links to it, which is the database for the application – var db = require("./models");
    - c. Middleware is installed, which allows for communication and data management of the application
    - d. Sessions and passport are required next, which help keep track of the user's login status and their activity
    - e. Routes are required next and linked to the two files in the routes folder
    - f. The final step is to sync with the database (i.e.: sequelize.db.sync()) and send a message to the user via console.log that the app is functioning.
        - i. force: true is not passed into the function as a parameter in this file so when node server is run it will not wipe the database each time
4. Config Folder
    - a. Middleware folder – isAuthenticated.js file
        - i. This file's purpose is to ensure that the user is logged in before they are able to access anything in the app.
        - ii. If they are not logged in it will redirect them to the login page.
    - b. config.json
        - i. This file sets up the requirements and information for the development, test and production platforms
        - ii. User will need to make sure that the development portion contains their own password information to actually run node server and connect to the database.
    - c. passport.js
        - i. requires passport – var passport = require("passport");
        - ii. requires passport-local – var LocalStrategy = require("passport-local").Strategy;
        - iii. requires the database from the models folder – var db = require("./models");

- iv. Local Strategy in this module allows authentication using a username and password in the application. By requiring passport, local authentication can be easily and unobtrusively integrated into the application.
- v. User will login with an email address instead of a username (line 10) then queries the database for the user information
  1. If there is no user with the email a message is displayed "Incorrect email" (lines 20-23)
  2. If there is an email but the password is incorrect a message is displayed "Incorrect Password" (lines 26-29)
  3. If none of those instances occur, the user is logged in (line 32)
- vi. Passport will serialize and deserialize the user's information (lines 40-46).
  1. passport.serializeUser and passport.deserializeUser are used to set the user id as a cookie in the user's browser and then get the id from the cookie to get the user info in a callback.
- vii. Passport is exported (line 49)
  1. This file is required in the server.js file (see 2.iv. above)

5. Models folder
   - a. index.js file
     - i. This file requires the following dependencies:
       1. var fs = require("fs");
       2. var path = require("path");
       3. var Sequelize = require("sequelize");
       4. var basename = path.basename(module.filename)
       5. var env = process.env.NODE_ENV || 'development';
       6. var config = require(__dirname + '/../config/config.json)[env];
       7. var db = {};
     - ii. This file initializes the sequelize process (lines 11-15)
     - iii. The fs section sets up the access to the physical file (lines 17-25)
     - iv. object.Keys returns an array of the database (lines 27-30)
     - v. exports.module = db (line 36)
       1. This module is used in the Routes folder (see 8,a.i.1)
   - b. user.js file
     - i. This file requires the following dependency:
       1. var bcrypt = require("bcryptjs")
          - a. This is the password hashing function
     - ii. This file creates the User model that will run when node server is initiated and create the table Users in the MySQL Workbench database
       1. Sets requirements for email (can't be null and must be of a proper email string) (lines 7-13)
       2. Sets requirements for password (can't be null) (lines 16-19)
     - iii. Creates a custom method for the User model.
       1. This will check to see if an unhashed password matches the hashed password in the database (lines 22-24)
          - a. Hashing guards against the possibility that someone who gains unauthorized access to the database can retrieve the passwords of every user in the system. It performs a one-way transformation on a password, turning the password into another String, called the hashed password. "One-way" means that it is practically

impossible to go the other way - to turn the hashed password back into the original password.

    iv. The hook runs, automatically hashing the user’s password before the user is created in the database (lines 27-29)
    v. Once those processes run, the User is returned (line 30)

6. Node Modules
    a. This file is automatically created when npm i is run
7. Public Folder
    a. Contains all of the html files needed for styling of the front-end site
        i. login.html
            1. login page for site
                a. User will enter email address or signup to access site
        ii. members.html
            1. Welcome page for site
        iii. signup.html
            1. Page for a new user to signup for access to the site
    b. stylesheets folder
        i. styles.css file
            1. Will contain the css styling for the site, currently only a top-margin is defined (lines 1-4)
    c. js folder
        i. login.js
            1. Retrieves references to the forms and inputs
                a. var loginForm = $(“form.login”);
                    i. pulled from html file login.html
                b. var emailInput = $(“input#email-input);
                    i. pulled from html files login.html and signup.html
                c. var passwordInput = $(“input#password-input”);
                    i. pulled from html files login.html and signup.html
            2. When the submit button is clicked, verifies that there is an email address and password entered (lines 8-17)
            3. If there is an email address and password that match, it runs the loginUser function and clears the form (lines 20-23)
            4. loginUser does a post to the api/login route and redirects the user to the Member page. (lines 26-34)
                a. The $.post loads data from the server using an HTTP Post request
            5. If there is an error, it logs the error (lines 35-37)
        ii. members.js
            1. This file does a GET request to figure out who just logged in to the site and updates the HTML on the page (lines 1-7)
        iii. signup.js
            1. Retrieves references to the forms and inputs
                a. var signUpForm = $(“form.signup”);
                    i. pulled from html file signup.html
                b. var emailInput = $(“input#email-input);
                    i. pulled from html files login.html and signup.html
                c. var passwordInput = $(“input#password-input”);
                    i. pulled from html files login.html and signup.html

            2. When the signup button is clicked, verifies that there is an email address and password entered (lines 8-17)
            3. If there is an email address and password, run the signupUser function (lines 19-22)
            4. The function signupUser does a POST to the signup route then directs the user to the members page (lines 26-36)
            5. If there is an err it throws up an alert (lines 35-41)
                a. The alert message comes from the signup.html page, and hasn't been filled in yet

8. Routes folder
    a. api-routes.js
        i. Declares required dependencies
            1. var db = require("../models) (see 5. for functionality of this folder)
            2. var passport = require("../config/passport) (see 4.c. for functionality of this folder)
        ii. Runs module.exports function and does a post using the passport middleware with the local strategy (see 4.c.iv. for functionality of this file)
            1. If the user provides good login credentials they are redirected to the members page, otherwise they are thrown an error (lines 5-11)
        iii. The next app.post runs the functionality of the signup page and adds the information to the sequel database and automatically logs the user in (lines 16-23)
            1. The user's password is automatically hashed and stored securely (see 3.b.ii.1.a. for an explanation of this process)
        iv. If there is an error it returns a status 401 (lines 24-27)
            1. A HTTP 401 Unauthorized client error status response code indicates that the request has not been applied because it lacks valid authentication credentials for the target resource.
        v. app.get("/logout) is the route for logging the user out and redirecting them back to the home page (lines 30-33)
        vi. The next app.get("/api/user_data/") pulls data from the database to be used on the client side (lines 36-46)
            1. If the user is not logged in it returns an empty object (lines 39-40)
            2. It will only send back the user's id and email address (lines 43-46)
                a. It's never a good idea to send back passwords, whether hashed or not
    b. html-routes.js
        i. Declares required dependencies
            1. var path = require("path");
                a. This allows us to use relative routes to the HTML files
            2. var isAuthenticated = require("../config/middleware/isAuthenticated)
                a. This allows us to check if a user is logged in or not (see 4.a. for explanation of functionality)
        ii. app.get("/") redirects the user to the login page if they already have an account set up (lines 11-15)
        iii. app.get("/login") redirects the users to the members page if they already have an account (lines 17-23)
        iv. app.get("/members) incorporates the isAuthenticated middleware and redirects the users to the signup page if they have not previously logged in (lines 27-29)

9. To run app, run node server in terminal
    a. This creates the table users in the database passport_demo (see 5.b.ii for description and functionality)
    b. Allows the user to view the site on http://localhost:8080 in the browser
10. Additional improvements
    a. CSS styling on all pages
    b. Detailed instructions for app use or purpose on homepage
    c. Hide password in config file using the .env process – do this before deployment especially
    d. Alerts on login page if login is not successful
    e. Alert at logout to know if logout was successful
    f. Add .gitignore file for node modules and .env file (suggested in 10.c.)