Project Instructions: REST API

Github

- Have a GitHub account and create a new repo for this project.
- Create a README.md file for your repo that explains what the project is and anything your fellow developers might need to know to use the project.

Download the project files:

- The README.md file lists the folders and files you'll be starting with.

Ensure that you have Node installed

- Make sure you have a recent version of Node: 8.0.0 or later.
- Installing Node.js and NPM on windows: https://treehouse.github.io/installation-guides/windows/node-windows.html
- Installing Node.js and NPM on MAC: https://treehouse.github.io/installation-guides/mac/node-mac.html

Install Node modules and get the database setup

- Open a Command Prompt (on Windows) or Terminal (on macOS and Linux) instance and browse to the root project folder.
- Run the command npm install to install the required dependencies.
- Run the command npm run seed to create your application's database and populate it with data.
- After the command completes, you'll find in the project's root folder a SQLite database file named fsjstd-restapi.db. To view the data inside the database, you can use DB Browser for SQLite. See https://sqlitebrowser.org/ for more information.
- Run the command npm start to run the Node.js Express application.
- You can press Ctrl-C to stop the Node.js REST API.

Working on the project

- The app.js file located in the root of the project folder configures Express to serve a simple REST API. You'll update this file to add your REST API routes.
- You'll build your application by adding .js files to the project. Use folders as you see fit to organize your application's files.

Install and Configure Sequelize

- Use npm to install Sequelize (the module is named sequelize.)
    - Note: The sqlite3 package that Sequelize depends upon to communicate with SQLite databases has already been installed.
- Instantiate an instance of the Sequelize class and configure the instance to use the fsjstd-restapi.db SQLite database that you generated when setting up the project.
- Use the authenticate() method to test the connection to the database.
    - Log a message to the console indicating if the connection was successfully made or failed.

<u>Define your Sequelize models</u>

- Define two Sequelize models: one for the Users table and another for the Courses table.`. Define the models following these requirements:
- ***User***
    - id (Integer, primary key, auto-generated)
    - firstName (String)
    - lastName (String)
    - emailAddress (String)
    - password (String)
- ***Course***
    - id (Integer, primary key, auto-generated)
    - userId (id from the Users table)
    - title (String)
    - description (Text)
    - estimatedTime (String, nullable)
    - materialsNeeded (String, nullable)
- When defining models for an existing database…
    - Be careful when naming your models and model properties! Model names and model properties need to match the above provided names exactly. Otherwise, your database access code won't work as expected.
    - If Sequelize throws an error related to a mismatch between the model and the associated table, the error message should tell you the cause of the problem.

<u>Define associations between your models</u>

- Within your User model, define a HasMany association between your User and Course models (i.e. a "User" has many "Courses").
- Within your Course model, define a BelongsTo association between your Course and User models (i.e. a "Course" belongs to a single "User").

<u>Create the user routes</u>

- Set up the following routes (listed in the format HTTP METHOD Route HTTP Status Code):
    - GET /api/users 200 - Returns the currently authenticated user

    - POST /api/users 201 - Creates a user, sets the Location header to "/", and returns no content

Create the course routes

- Set up the following routes (listed in the format HTTP METHOD Route HTTP Status Code):
    - *GET /api/courses 200* - Returns a list of courses (including the user that owns each course)
    - *GET /api/courses/:id 200* - Returns a the course (including the user that owns the course) for the provided course ID
    - *POST /api/courses 201* - Creates a course, sets the Location header to the URI for the course, and returns no content
    - *PUT /api/courses/:id 204* - Updates a course and returns no content
    - *DELETE /api/courses/:id 204* - Deletes a course and returns no content

Update User and Course routes

- Update the User and Course POST and PUT routes to validate that the request body contains the following required values. Return validation errors when necessary.
- ***User***
    - firstName
    - lastName
    - emailAddress
    - password
- ***Course***
    - Title
    - description
- Implement validations within your route handlers or your Sequelize models.
    - Sequelize model validation gives you a rich set of tools to validate user data. See Sequelize docs for more information.
    - Use the Express next() function in each route handler to pass any Sequelize validation errors to the global error handler.
- Send validation error(s) with a400 status code to the user.

Hashing the password

- Update the POST /api/users route to hash the user's password before persisting the user to the database.
- For security reasons, we don't want to store user passwords in the database as clear text.
- Use the bcryptjs npm package to hash the user's password.
- See https://github.com/dcodeIO/bcrypt.js for more information.

Set up permissions to require users to be signed in

- Add a middleware function that attempts to get the user credentials from the Authorization header set on the request.
- You can use the basic-auth npm package to parse the Authorization header into the user's credentials.
    - The user's credentials will contain two values: a name value—the user's email address—and a pass value—the user's password (in clear text).
- Use the user's email address to attempt to retrieve the user from the database.
- If a user was found for the provided email address, then check that user's stored hashed password against the clear text password given using bcryptjs.
- If the password comparison succeeds, then set the user on the request so that each following middleware function has access to it.
- If the password comparison fails, then return a 401 status code to the user.
- Use this middleware in the following routes:
    - GET /api/users
    - POST /api/courses
    - PUT /api/courses/:id
    - DELETE /api/courses/:id

Test the routes

- Postman is an application that you will use to explore and test REST APIs. We've provided you with a collection of Postman requests as part of the project files. Here's how to load the provided collection into Postman:
- If you haven't already, install Postman. Links and instructions are available on their website at https://www.getpostman.com/
- Once you have Postman installed and open, click on the "Import" button in the top left hand corner of the application's window.
- In the opened dialog, click the "Choose Files" button and browse to the folder that contains your project files.
- Select the RESTAPI.postman_collection.json file.
- You should now see the FSJS Techdegree: REST API Project collection in the left hand pane of the main Postman window.
- Be sure that your REST API is currently running (see the previous project step for details).
- Click on one of the available requests to load it into a tab. Click on the Send button to issue the request to the local server.
- When testing routes that require authentication, make sure to set the Authorization Type in postman to Basic Auth to enter the user's username (their email address) and password.

Debugging help

- As you build out your REST API, you'll naturally encounter errors and unexpected behavior. Here are some reminders and suggestions on how to debug your REST API.
- You can edit the nodemon.json file to enable additional logging options for your application.

    - Under the env section in the JSON configuration, set the DB_ENABLE_LOGGING environment variable to enable logging of all database queries and set the ENABLE_GLOBAL_ERROR_LOGGING to enable logging of all errors handled by the global error handler.
    - If you change the nodemon configuration while the application is curr ently running, you'll need to press Ctrl-C to stop the application and re-run the npm start command.
- If Node.js crashed as a result of the error, you can look in the Command Prompt (on Windows) or Terminal (on macOS and Linux) window and see the exception information.
- Sometimes errors don't result in exceptions, but instead are returned as 400 or 500 HTTP status codes. Errors returned from your REST API will be logged in Postman.
- For a deeper, more detailed analysis of the state of your application, you can use Google Chrome to debug your Node.js application.

Add code comments!

EXTRA CREDIT

Add additional user email address validations to the POST/api/users **route**

- Validate that the provided email address value is in fact a valid email address.
- Validate that the provided email address isn't already associated with an existing user record.

Ensure that a user can only edit and delete their own courses

- Update the PUT /api/courses/:id and DELETE /api/courses/:id routes to check if the course for the provided :id route parameter value is owned by the currently authenticated user.
- Return a 403 status code if the current user doesn't own the requested course.

Update the Sequelize model queries for the Courses endpoint GET **routes to filter out the following properties.**

- ***User***
    - password
    - createdAt
    - updatedAt
- ***Course***
    - createdAt
    - updatedAt