Unit 5 Assessment

Introduction

It's time to practice and solidify the skills you learned in this unit.

Independent work
This is an individual assignment. Please do not collaborate with your peers or share your work until the project is reviewed as a class.

Setup

. Click here to download the code files you need to for this project.

. Navigate to the files on the command line.

Part 1: Data Modeling

Imagine you are opening a pet adoption agency where you will rescue and care for animals and try to find them owners who are a good match for them.

Design a database with at least 4 tables for your pet adoption agency. Include any relationships between tables where you feel they are needed.

For example, you'll need an animals table. Perhaps you have an animal species table as well. The relationship between animal species and animals is one-to-many. For every one species in the species table, you could, at most, have many animals of that species in the animals table.

Submit a diagram of your database for this project.

Drawing Tools
You can use ***https://www.dbdesigner.net/*** (or ***https://draw.io*** or ***https://drawings.google.com/***) to create your diagram. Make sure you specify your relationships between tables in your diagram.

When you're finished with your drawing, either take a screenshot or download a PDF of you diagram, add it to this repository, and push it to Github.

Part 2: Sequelize

For this portion, you'll be connecting a front end to a database through a server and making queries using Sequelize.

The front end of the app is simple – there is a form where you can add the name of a city, the country it's in, and your rating of that city. All of the cities that you add will show up on the page as well.

Setup

. Ensure you're in this project's folder in your terminal.

. Run npm install to get the project's dependencies installed.

Step 1: Dotenv

For our server files to work, we're going to need ***dotenv***.

. Install 'dotenv' using npm install dotenv

. Create a file in the root of the travel-journal directory called .env

. In that file, create a SERVER_PORT variable and set it to **4004** (it needs to be this number or the front end won't work)

. Additionally, create a CONNECTION_STRING variable and set it to your ***URI*** from bit.io (you can use the database from your lab earlier this unit or create another)

. At the top of controller.js, require in the dotenv package and call its config method (you can ignore the other code in there until later steps)

. Optional: destructure CONNECTION_STRING from process.env in controller.js

Step 2: Sequelize

In order to make queries to our database, we'll need Sequelize.

. Install Sequelize and its dependencies using npm install sequelize pg pg-hstore

. In controller.js, require the sequelize package and save it to a variable called ***Sequelize*** (we'll be working with the seed function in the next step, don't edit it yet)

. Initialize a new Sequelize instance, passing in your connection string and an object that looks like this:

```
{
  dialect: 'postgres',
  dialectOptions: {
      ssl: {
          rejectUnauthorized: false
```

```
    }
  }
}
```

Step 3: Seeding the Database

Let's take a look at the seed function. Near the top of the query, you'll see a line that says *****YOUR CODE HERE*****.

- Delete *****YOUR CODE HERE***** and replace it with a SQL query.
- Your query should create a table called cities that has 4 columns (**Make sure you use these exact names so the front end works**): city_id: serial, primary key; name: varchar; rating: integer; country_id: integer that should match a country_id from the countries table.
- Start up your backend with nodemon (make sure you're in the right directory, also you can keep it running for the duration of the assessment)
- Open Postman and make a POST request to ***http://localhost:4004/seed***.
- Your database should now be seeded!

Step 4: Getting Countries

In the seed function, a list of countries was added into your database. The front end needs that list so it can populate a select element with options for the user.

- In controller.js's export object, write a new function called getCountries (make sure it accepts req & res). In the function:
  - Use sequelize.query to query your database for **all** the columns from the countries table.
  - Pass a callback to the .then() that looks something like the code below (you can also add a .catch if you'd like):

```
(dbRes) => {
  res.status(200).send(dbRes[0])
}
```

- In index.js, comment line 15 back in (this line: app.get('/countries', getCountries))
- Open index.html (from the public folder) in your browser
- You should be seeing a list of countries in the dropdown selector in the form! (If

that's working, there will still be 2 errors in the console – GET and a 404 – they will get fixed shortly)

Step 5: Adding Entries

Now you'll write the function that will let you add cities to the database.

. In controller.js, write a new function called createCity. In the function:
  - Use sequelize.query to query your database to insert some data into your table. A name, a rating, and a countryId will be sent on the req.body. Write an insert statement that adds those into the database. (Remember to use a template string for this, and feel free to destructure the values from req.body if you'd like).

  - Pass a callback to the .then() that looks something like the code below (you can also add a .catch if you'd like):

```
. (dbRes) => {
.   res.status(200).send(dbRes[0])
. }
.
```

. In index.js, comment line 18 back in (this line: app.post('/cities', createCity))

. You should now be able to add cities in the browser! You can confirm this by using the form and then selecting from your DB on bit.io's site. However, they won't be showing up in your app yet, which is where the next step comes in.

Step 6: Getting Cities

You're doing great!

. In controller.js, write a new function called getCities. In the function:
  - Use sequelize.query to query your database for columns from both the cities and countries tables. **cities**: city_id, name (alias 'city'), rating. **countries**: country_id, name (alias 'country'). Make sure to spellcheck the aliases as well as the column names. JOIN the tables where country_id is equal.
    - ***Hint:*** *you can give a column an alias using the AS keyword. Refer to this handout for a refresher on aliases.*

  -

- Pass a callback to the .then() that looks something like the code below (you can also add a .catch if you'd like):

```
(dbRes) => {
  res.status(200).send(dbRes[0])
}
```

3. In index.js, comment line 19 back in (this line: app.get('/cities', getCities))
4. You should see your entries now in the browser!

Step 7: Deleting Cities

Last, let's add the ability to delete a city from our list.

1. In controller.js, write a new function called deleteCity. In the function:
   - Using sequelize.query query your database to delete a city. An id will be sent on req.params. That will be the city_id of the city that you will delete from the cities table. (Remember to use a template string for this, and feel free to destructure the values from req.params if you'd like).

   - Pass a callback to the .then() that looks something like the code below (you can also add a .catch if you'd like):

```
(dbRes) => {
  res.status(200).send(dbRes[0])
}
```

2. In index.js, comment line 20 back in (this line: app.delete('/cities/:id', deleteCity))
3. Now you can delete cities!

Extra Credit

Travel Journal

1. Update the getCities function to get the cities back with the highest rated cities at the top down to lowest rated cities at the bottom.

. Update your seed function to initially insert at least 3 entries to the city table.

Be sure to push your code to GitHub for this assignment and turn your link in on Frodo!

To pass this assessment you must score at least 15/21.