WEB322 Assignment 2

Submission Deadline:

Friday, September 29th, 2023 @ 11:59 PM

Assessment Weight:

9% of your final course Grade

Objective:

Create and publish a web app that uses multiple routes which serve static content (text / json) as well as create a "lego service" module for accessing data.

Part 1: Getting Started (Files & Directories)

As with the previous assignment, we will first be creating the files and directories for our solution. To begin:

- Create a new folder somewhere on your system to store the assignment.
- Within this folder, create a "data" folder and a "modules" folder
- Within the "data" folder, place the "**themeData.json**" file (available within a .zip file located here)
- Within the "modules" folder, create a new file called: "**legoSets.js**"

Now that our overall directory structure is in place, we must download the "sets" dataset in a .csv format (available from Rebrickable.com). You can find the specific "sets.csv" file (available within a .zip file) located here. If you open this file in a spreadsheet editor (ie: Excel), you will notice that it contains 5 columns of data and over 20,000 rows.

For this assignment, we will select a subset of this data for our web application and include it in our "data" folder alongside the associated "themeData.json":

- First, select the rows of lego sets that you would like to use in your assignment (select between 30 and 50 rows - these do not have to be sequential), as well as the first "header" row.
- With the rows selected hit Edit > Copy (or Cmd / Ctrl + C) to copy the rows
- Navigate to: https://tableconvert.com/csv-to-json and paste your rows within the "Data Source". You will see that the data in the "Table Editor" is automatically populated, as well as the "Table Generator", where you will find your data, converted to a "JSON" format.
- Copy the newly generated "JSON" formatted data, and place it in a new file within the "data" folder of your solution, called "**setData.json**"
- Congratulations – we now have the data for the assignment!

  NOTE: Since you can choose your own data for this assignment, the below instructions may not use examples from your particular data set. They are there simply to illustrate the functionality of the assignment.

Part 2: Writing legoSets.js

The purpose of the "legoSets.js" file is to provide easy access to the Lego data for other files within our assignment that require it.

To start, add the following two lines to the beginning of the "legoSets.js" file:

```
const setData = require("../data/setData");
const themeData = require("../data/themeData");
```

This will automatically read both files and generate two arrays of objects: "setData" and "themeData".

Next, create a variable called "sets", initialized to an empty array, ie:

```
let sets = [];
```

This will be the completed array of Lego "set" objects, after processing the above "setData" and "themeData" arrays.

The remainder of this file will contain functions, according to the following specification:

- **Initialize()**

  The purpose of this function is to fill the "sets" array (declared above), by adding copies of all the **setData** objects, ie:

  ```
  {
    "set_num": "001-1",
    "name": "Gears",
    "year": "1965",
    "theme_id": "1",
    "num_parts": "43",
    "img_url": "https://cdn.rebrickable.com/media/sets/001-1.jpg"
  }
  ```

  However, each of these objects must **also** include a new **"theme"** property. The value of the "theme" property should be the corresponding theme **name** from the **themeData.json** file, whose "id" value matches the "theme_id" for the "setData" object.

  For example, in the above "setData" object, we have a "theme_id" value of "1". Therefore, we must add an additional "theme" property with the value of "Technic", since the "id" value for "Technic" is also "1" within the **themeData.json** file.

  This will result in the following object being added (pushed) to the "sets" array:

  ```
  {
    "set_num": "001-1",
    "name": "Gears",
    "year": "1965",
    "theme_id": "1",
    "num_parts": "43",
  ```

```
  "img_url": "https://cdn.rebrickable.com/media/sets/001-1.jpg"
  "theme": "Technic"
}
```

As an additional example, consider the following object from a "setData" array:

```
{
  "set_num": "0011-2",
  "name": "Town Mini-Figures",
  "year": "1979",
  "theme_id": "67",
  "num_parts": "12",
  "img_url": "https://cdn.rebrickable.com/media/sets/0011-2.jpg"
}
```

This has a "theme_id" of "67". From the "themeData.json" file, the name of the theme with id "67" is "Classic Town". Therefore, the new object to be added to the "sets" array should be:

```
{
  "set_num": "0011-2",
  "name": "Town Mini-Figures",
  "year": "1979",
  "theme_id": "67",
  "num_parts": "12",
  "img_url": "https://cdn.rebrickable.com/media/sets/0011-2.jpg"
  "theme": "Classic Town"
}
```

HINT: Consider using the **.find()** and **.forEach()** Array methods for your solution

- **getAllSets()**

  This function simply returns the complete "sets" array

- **getSetByNum(setNum)**

  This function will return a specific "set" object from the "sets" array, whose "set_num" value matches the value of the "setNum" parameter, ie: if getSetByNum("001-1") was invoked, the following set object would be returned:

```
{
  "set_num": "001-1",
  "name": "Gears",
  "year": "1965",
  "theme_id": "1",
  "num_parts": "43",
  "img_url": "https://cdn.rebrickable.com/media/sets/001-1.jpg"
  "theme": "Technic"
}
```

HINT: Consider using the **.find()** Array method for your solution

- **getSetsByTheme(theme)**

  The purpose of this function is to return an array of objects from the "sets" array whose "theme" value matches the "theme" parameter. However, it is important to note that the "theme" parameter may contain only part of the "theme" string, and case is ignored. For example:

  getSetsByTheme(**"tech"**);

  would return all the sets from your "sets" array whose "theme" property contains the string "tech" (ignoring case). For example, if your "sets" array contained the following objects, they would be returned:

  [{
  "set_num": "02-Jan",
  "name": "Bulldozer Chain Links",
  "year": "1982",
  "theme_id": "453",
  "num_parts": "50",
  "img_url": "https://cdn.rebrickable.com/media/sets/01-2.jpg",
  "theme": "Technic"
  },
  {
  "set_num": "01-Feb",
  "name": "Extra Large Tires & Hubs",
  "year": "1982",
  "theme_id": "453",
  "num_parts": "4",
  "img_url": "https://cdn.rebrickable.com/media/sets/02-1.jpg",
  "theme": "Technic"
  }]

  HINT: Consider using the **.filter()** Array method as well as the **.toUpperCase() / .toLowerCase()** and **.includes()** String methods for your solution

Part 3: Testing & Refactoring legoSets.js

Once you have completed the above four functions, test them at the bottom of your file by first invoking the "initialize()" function, then the others (since they all need the "sets" array to be filled with data).

NOTE: You should be able to run this file using the command "node modules/legoSets.js"

If the correct data is returned and you're satisfied that everything is working correctly, you can delete your testing code.

Now, we can begin to refactor our functions to use "Promises".

- Each of the 4 functions created (initialize(), getAllSets(), getSetsByNum(setNum), getSetsByTheme(theme)) must return a new Promise object that "resolves" either with data (if the function returns data) or "rejects" with an error, if the function encounters an error, for example:
    - **Initialize()** should resolve with no data, once the operation is complete (ie: the "sets" array is filled with objects)
    - **getAllSets()** should resolve with the completed "sets" array
    - **getSetsByNum(setNum)** should resolve with the found "set" object, and reject with an appropriate message (ie: unable to find requested set) if the set was not found
    - **getSetsByTheme(theme)** should resolve with the found "set" objects, and reject with an appropriate message (ie: unable to find requested sets)

Finally, we must make sure this file functions as a "module", either by manually adding all 4 functions to the "module.exports" object individually, or by assigning them at once using an object literal shorthand, ie:

```
module.exports = { initialize, getAllSets, getSetByNum, getSetsByTheme }
```

Part 4: Creating the Web Server

The final part of this assignment is to create a "simple web server" that makes our data available. For now, this will simply involve returning the data – we will focus on "rendering" actual HTML pages in future assignments.

To begin:

- Create a new file called "server.js" in the root of your assignment folder (making sure not to accidentally place it within the "data" or "modules" folders):

- Next, run the command "npm init" in the integrated terminal, ensuring that your "entry point" is **server.js** (this should be the default), and "author" is your full name, ie: "John Smith".
- Once this is complete, run the command "npm install express" to generate the "node_modules" folder and update your newly-created "package.json" with the "express" dependency
- Finally, open your server.js file and add the line:

    ```
    const legoData = require("./modules/legoSets");
    ```

    This will ensure that the functions that we wrote in parts 2 & 3 will be available on the **legoData** object (ie: legoData.initialize(), legoData.getAllSets(), etc)

From this point on, we can begin creating a simple web server using Express.js that features simple GET routes.

However, before the server starts (ie: the **app.listen()** function is invoked), we must ensure that the "sets" array has been successfully built within our "legoSets" module. Therefore, we must invoke the **legoData.initialize()** function and make sure it has completed successfully (ie "resolves") before we execute our app.listen() function.

NOTE: Even though we do not have any routes configured, we should be able to start the server using the command "node server.js"

Finally, ensure that the following routes are configured in your server.js file:

- **GET "/"**

  This route simply sends back the text: "Assignment 2: Student Name - Student Id" where "Student Name" and "Student Id" are your Name & Seneca Student Id

- **GET "/lego/sets"**

  This route is responsible for responding with all of the Lego sets (array) from our legoData module

- **GET "/lego/sets/num-demo"**

  In this route, you will demonstrate the **getSetByNum** functionality, by invoking the function with a known setNum value from your data set. Once the function has resolved successfully, respond with the returned object.

  NOTE: Do not forget to include code to handle the situation where **getSetByNum** fails. If this is the case, simply respond with the error text.

- **GET "/lego/sets/theme-demo"**

  In this route, you will demonstrate the **getSetsByTheme** functionality, by invoking the function with a known theme value from your data set (using a part of the theme name in lower case, ie "tech" for "Technic", or "town" for "Classic Town"). Once the function has resolved successfully, respond with the returned array.

  NOTE: Do not forget to include code to handle the situation where **getSetsByTheme** fails. If this is the case, simply respond with the error text.

With your routes complete, you should be ready to hand in our assignment. To see a completed version of this app running, visit: https://wptf-a2-sampl e.cyclic.cloud

Assignment Submission:

- Add the following declaration at the top of your **server.js** file:

```
/*********************************************************************************
*  WEB322 – Assignment 02
*
*  I declare that this assignment is my own work in accordance with Seneca's
*  Academic Integrity Policy:
*
*  https://www.senecacollege.ca/about/policies/academic-integrity-policy.html
*
*  Name: _____ Student ID: _____ Date: _____
*
*********************************************************************************/
```

- Compress (.zip) your assignment folder and submit the .zip file to My.Seneca under **Assignments** -> **Assignment** 2

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.