WEB322 Assignment 2

Assessment Weight:

9% of your final course Grade

Objective:

The second assignment will focus on more advanced JavaScript skills including: modules, promises, using / defining objects using the "class" keyword and passing functions as parameters (callback functions).

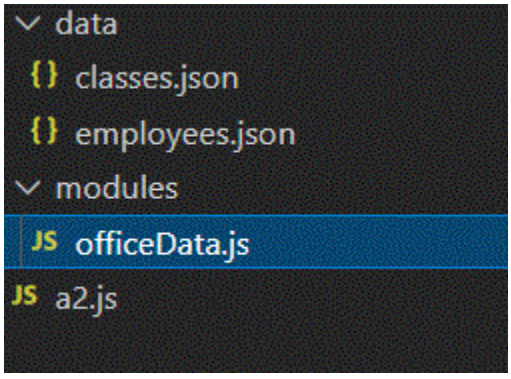
The main objective is to create and test a module that is responsible for pulling data from multiple files as well as to provide multiple promise-based functions that "resolve" with the data.

Specification:

This assignment will consist of multiple files, including: a main "a2.js" file, a "modules" folder containing one module ("officeData.js") and a "data" folder containing two files ("classes.json" & "employeess.json").

Step 1: Create the Files & Directories

Whenever we start a new project or example for this course, we will always create a new folder to contain the code. For this assignment, start by creating a new folder somewhere on your local machine to house all of your code. Once you have done this, open it up in Visual Studio Code and create the following file / folder structure:



You must follow the above naming including case. Failure to do so will result in Zero mark for the assignment.

Step 2: Obtaining the Data (classes.json & employees.json)

The data for this assignment will exist in two separate files: classes.json and employees.json. Follow the steps below to obtain the data. **Recall:** JSON stands for **JavaScript Object Notation** and is a way of representing JavaScript Objects in a plain-text format.

- The data for classes is supplied with the assignment requirements. Copy the contents of the JSON file to your own classes.json file (within the "data" folder).
- The data for employees is supplied with the assignment requirements. Copy the entire contents of the JSON file to your own employees.json file (within the "data" folder) - this should be an array of 260 "employee" objects

Step 3: Writing the "officeData" Module

The promise driven officeData.js module will be responsible for reading the employees.json and classes.json files from within the "data" directory, parsing the data into arrays of objects and returning elements (ie: "employee" objects) from those arrays to match queries on the data.

Essentially the officeData.js module will encapsulate all the logic to work with the data and only expose accessor methods to fetch data/subsets of the data.

Module Data

To help us manage the data in this module, we must create a **class** called "Data" according to the following specification:

- This class only contains a single constructor which takes two parameters: **employees** and **classes**.
    - This constructor will take the value from the **employees** parameter and use it as the value for a property called "employees" within the class
    - Similarly, the constructor will take the value from the **classes** parameter and use it as the value for a property called "classes" within the class

Once the class definition is complete, proceed to declare the variable "dataCollection" beneath the class and assign it to ***null***. This will be the variable that holds an *instance* of the "Data" class once its created (within the "initialize" function defined below).

Exported Functions

Each of the below functions are designed to work with the employees and classes datasets. Since we have no way of knowing how long each function will take (we cannot assume that they will be instantaneous, ie: what if we move from .json files to a remote database, or introduce hundreds of thousands of objects into our .json dataset? - this would increase lag time), **every one of the below functions must return a promise** that **passes the data** via it's "**resolve**" method (or - if **an error occurred**, passes an **error message** via it's "**reject**" method).

When we access these methods from the a2.js file, we will be assuming that they return a promise and we will respond appropriately with **.then()** and .**catch()**.

initialize()

- This function will read the contents of the "./data/employees.json" file

  hint: see "Reading files with Node.js" (from the documentation), ie:

```
fs.readFile('somefile.json', 'utf8', function(err, dataFromSomeFile){
    if (err){
        console.log(err); // or reject the promise (if used in a promise)
        return; // exit the function
    }

    let data = JSON.parse(dataFromSomeFile); // convert the JSON from the file into an array of objects
    console.log(data);
});
```

- Only once the read operation for "./data/employees.json" has completed successfully (not before), repeat the process for the "./data/classes.json"
- Once these two operations have finished successfully, create a new *instance* of the **Data** class (defined above) and assign it to the variable dataCollection using the data returned from your fs.readFile operations, ie:

  dataCollection = new Data(***employeeDataFromFile***, ***classDataFromFile***);

  Going forward, you can access the full array of **employees** and **classes** in your other functions using the **dataCollection** object, ie: **dataCollection.employees** or **dataCollection.classes**
- Finally, invoke the **resolve** method for the promise to communicate back to a2.js that the operation was a success.
- **NOTE:** If there was an error at any time during this process, instead of outputting an error to the console, invoke the **reject** method for the promise and pass an appropriate message, ie: reject("unable to read employees.json").

getAllEmployees()

- This function will provide the full array of "employee" objects using the **resolve** method of the returned promise.
  - If for some reason, the length of the array is 0 (no results returned), this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

getEAs()

- This function will provide an array of "employee" objects whose **EA** property is **true** using the **resolve** method of the returned promise.

- If for some reason, the length of the array is 0 (no results returned), this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

getClasses()

- This function will provide the full array of "class" objects using the **resolve** method of the returned promise.
    - If for some reason, the length of the array is 0 (no results returned), this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

getPartTimers()

- This function will provide an array of "employee" objects whose **status** property is **"Part Time"** using the **resolve** method of the returned promise.
    - If for some reason, the length of the array is 0 (no results returned), this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

Step 4: Writing the "a2.js"

The final development step in the assignment is to actually write the a2.js file that will test the functionality of our module:

- First, we must "require" the officeData module at the top of a2.js
- Next, we must invoke the "initialize" function from our officeData and output a success message to the console of it was successful and an error message if it failed (using then and catch).
- Once this is complete, try running your code. If you see the success message, then you were able to successfully read both files and you can proceed with the rest of the assignment. If you see the error message, go back and look at that "initialize" function and make sure there aren't any errors.
- Now that you're confident that your initialize function works properly, you can delete your success message and in its place, write code to test (invoke) all 3 of your other functions from officeData:

getAllEmployees()

- To ensure that this function is working properly, output the number of employees to the console in the format: "Successfully retrieved **x** employees", where **x** is the number of employees (**HINT**: You can use the array.length property to get the number of elements in an array)

getClasses()

- Testing this function is almost identical to what was done for "getAllEmployees()", however it will display the number of "classes" returned in the format: "Successfully retrieved **x** classes", where **x** is the number of classes.

getEAs()

- This is another function to test. To ensure it works properly, display the number of "EAs" returned in the format: "Successfully retrieved **x** EAs", where **x** is the number of EAs

getPartTimers()

- This is another function to test. To ensure it works properly, display the number of "Part Time" returned in the format: "Successfully retrieved **x** Part Timers", where **x** is the number of employees with Part Time status

NOTE: Do not forget to handle the possibility that a promise returned from officeData may get **reject**ed in the future, so do not forget to include "catch" functions where appropriate.

Step 5: Confirming the Output

If everything is working properly once the program starts, you should see the following in the console:

Successfully retrieved 260 Employees
Successfully retrieved 11 classes
Successfully retrieved 30 EAs
Successfully retrieved 106 Part Timers

Assignment Submission:

1. Add the following declaration at the top of your a2.js file:

```
/*********************************************************************************
* WEB322 – Assignment 2
* I declare that this assignment is my own work in accordance with Seneca  Academic Policy.
* No part of this assignment has been copied manually or electronically from any other source
* (including web sites) or distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
*********************************************************************************/
```

2. Compress your working folder (containing your a2.js file as well as your "data" and "modules" folders) and submit it on My.Seneca

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.
- After the end (11:30PM) of the due date, the assignment submission link on My.Seneca will no longer be available.