WEB322 Assignment 2

# Submission Deadline:

Friday, February 2 @ 11:59pm

# Assessment Weight:

8% 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 an "UN countries 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/project.
* Within this folder, create a "data" folder and a "modules" folder
* Within the "data" folder, place the "**regionData.json**" and "**countryData.json**" files (available within a .zip file [located here](https://wsong18.github.io/web322/download/2241/a2-data.zip))
* Within the "modules" folder, create a new file called: "**unCountries.js**"

Now we now have the data for the assignment!

# Part 2: Writing unCountries.js

The purpose of the "unCountries.js" file is to provide easy access to the UN member states, i.e., UN countries data for other files within our assignment that require it.

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

const countryData = require("../data/countryData");

const regionData = require("../data/regionData");

This will automatically read both files and generate two arrays of objects: "countryData" and "regionData".

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

let countries = [];

This will be the completed array of Un "country" objects, after processing the above "countryData" and "regionData" arrays.

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

**Initialize()**  
  
The purpose of this function is to fill the "countries" array (declared above), by adding copies of all the **countryData** objects, eg:  
  
{

"name": "Canada",

"official": "Canada",

"nativeName": "Canada",

"a2code": "CA",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/Canada",

"capital": "Ottawa",

"regionId": "2",

"languages": "English,French",

"population": 38005238,

"flag": "https://flagcdn.com/w320/ca.png"

}

However, each of these objects must **also** include a new **"region**" property. The value of the "region" property should be the corresponding region **object** from the **regionData.json** file, whose "id" value matches the "regionId" for the "countryData" object.

For example, in the above "countryData" object, we have a "regionId" value of "2". Therefore, we must add an additional "region" property with the value of region object { "id": "2", "name": "Americas", "subs": "North America, South America, Central America, Caribbean" }, since the "id" value of the object is also "2" within the **regionData.json** file.

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

{

"name": "Canada",

"official": "Canada",

"nativeName": "Canada",

"a2code": "CA",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/Canada",

"capital": "Ottawa",

"regionId": "2",

"languages": "English,French",

"population": 38005238,

"flag": "https://flagcdn.com/w320/ca.png",

**"region":** {"id": "2", "name": "Americas", "subs": "North America, South America, Central America, Caribbean"}

}

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

{

"name": "China",

"official": "People's Republic of China",

"nativeName": "中国",

"a2code": "CN",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/China",

"capital": "Beijing",

"regionId": "3",

"languages": "Chinese",

"population": 1402112000,

"flag": "https://flagcdn.com/w320/cn.png"

}

This has a "regionId" of "3". From the "regionData.json" file, the region object with id "3" is added into the above object. Therefore, the new country object to be added to the "countries" array should be:

{

"name": "China",

"official": "People's Republic of China",

"nativeName": "中国",

"a2code": "CN",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/China",

"capital": "Beijing",

"regionId": "3",

"languages": "Chinese",

"population": 1402112000,

"flag": "https://flagcdn.com/w320/cn.png",

**"region": {"id": "3", "name": "Asia", "subs": "South Asia, Central Asia, Northeastern Asia"}**

}

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

* **getAllCountries()**

This function simply returns the complete "countries" array

* **getCountryByCode(countryCode)**

This function will return a specific "country" object from the "countries" array, whose "a2code" value matches the value of the "countryCode" parameter, ie: if getCountryByCode("ca") was invoked, the following country object would be returned:

{

"name": "Canada",

"official": "Canada",

"nativeName": "Canada",

"a2code": "CA",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/Canada",

"capital": "Ottawa",

"regionId": "2",

"languages": "English,French",

"population": 38005238,

"flag": "https://flagcdn.com/w320/ca.png",

**"region":** {"id": "2", "name": "Americas", "subs": "North America, South America, Central America, Caribbean"}

}

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

* **getCountriesByRegion(region)**

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

getCountriesByRegion(**"oceania"**);

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

[

{

"name": "Australia",

"official": "Commonwealth of Australia",

"nativeName": "Australia",

"a2code": "AU",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/Australia",

"capital": "Canberra",

"regionId": "5",

"languages": "English",

"population": 25687041,

"flag": "https://flagcdn.com/w320/au.png",

**"region": {**

**"id": "5",**

**"name": "Oceania",**

**"subs": "Australasia, Melanesia, Micronesia, and Polynesia"**

**}**

},

… …

… … ,

{

"name": "Vanuatu",

"official": "Republic of Vanuatu",

"nativeName": "Vanuatu",

"a2code": "VU",

"permanentUNSC": false,

"wikipediaURL": "https://en.wikipedia.org/wiki/Vanuatu",

"capital": "Port Vila",

"regionId": "5",

"languages": "Bislama,English,French",

"population": 307150,

"flag": "https://flagcdn.com/w320/vu.png",

**"region": {**

**"id": "5",**

**"name": "Oceania",**

**"subs": "Australasia, Melanesia, Micronesia, and Polynesia"**

**}**

}

]

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

# Part 3: Testing & Refactoring unCountries.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 "countries" array to be filled with data).

**NOTE:** You should be able to run this file using the command "node modules/unCountries.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](https://webprogrammingtoolsandframeworks.sdds.ca/Handling-Asynchronous-Code/promises-async-await)".

* Each of the 4 functions created (initialize(), getAllCountries(), getCountriesByCode(countryCode), getCountriesByRegion(region)) 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 "countries" array is filled with objects)
  + **getAllCountries()** should resolve with the completed "countries" array
  + **getCountriesByCode(countryCode)** should resolve with the found "country" object, and reject with an appropriate message (ie: unable to find requested country) if the country was not found)
  + **getCountriesByRegion(region)** should resolve with the found "country" objects, and reject with an appropriate message (ie: unable to find requested countries)

Finally, we must make sure this file functions as a "[module](https://webprogrammingtoolsandframeworks.sdds.ca/Web-Server-Introduction/modules-node-package-manager#modules)", 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, getAllCountries, getCountryByCode, getCountriesByRegion }

# Part 4: Creating the Web Server

The final part of this assignment is to create a "[simple web server](https://webprogrammingtoolsandframeworks.sdds.ca/Web-Server-Introduction/simple-web-server-using-expressjs)" 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):  
    
  A screenshot of a computer program

  Description automatically generated
* 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**, e.g.: "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 unCountryData = require("./modules/unCountries");  
    
  This will ensure that the functions that we wrote in parts 2 & 3 will be available on the **unCountryData** object (ie: unCountryData.initialize(), unCountryData.getAllCountries(), etc)

From this point on, we can begin creating a simple web server using [Express.js](https://webprogrammingtoolsandframeworks.sdds.ca/Web-Server-Introduction/simple-web-server-using-expressjs#expressjs) that features [simple GET routes](https://webprogrammingtoolsandframeworks.sdds.ca/Web-Server-Introduction/simple-web-server-using-expressjs#simple-get-routes).

However, before the server starts (ie: the **app.listen()** function is invoked), we must ensure that the "countries" array has been successfully built within our "unCountries" module. Therefore, we must invoke the **unCountryData.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 "/un/countries"**  
    
  This route is responsible for responding with all of the UN member states, i.e., countries (array) from our unCountryData module
* **GET "/un/countries/code-demo"**  
    
  In this route, you will demonstrate the **getCountryByCode** functionality, by invoking the function with a known countryCode ( i.e. a2code that means 'alpha 2 code', e.g., 'ca' for Canada) value from your country data. Once the function has resolved successfully, respond with the returned object.  
    
  **NOTE:** Do not forget to include code to handle the situation where **getCountryByCode** fails. If this is the case, simply respond with the error text.
* **GET "/un/countries/region-demo"**  
  In this route, you will demonstrate the **getCountriesByRegion** functionality, by invoking the function with a known region value from your country data (using a part of the region name in lower case, e.g. "oceania". Once the function has resolved successfully, respond with the returned array.  
    
  **NOTE:** Do not forget to include code to handle the situation where **getCountriesByRegion** fails. If this is the case, simply respond with the error text.

### **Testing**: Sample Solution

With your routes complete, you should be ready to hand in our assignment. To see a completed version of this app running, visit: <https://assignments-322-demo.cyclic.app/>

# Part 5: Pushing to Cyclic

* Once you are satisfied with your application, deploy it to Cyclic:
  + Ensure that you have checked in your latest code using **git** (from within Visual Studio Code)
  + Create a [GitHub](https://github.com/) (public, remote) repository.
  + Connect to local Git repository to GitHub.
  + Connect the GitHub repository to [Cyclic](https://www.cyclic.sh).
  + Change the GitHub repository to “Private”
* **IMPORTANT NOTE:** Since we are using an "**unverified" free** account on Cyclic, we are limited to only **3 apps**, so if you have been experimenting on Cyclic and have created 3 apps already, you must delete one (or verify your account with a credit card). Once you have received a grade for Assignment 1, it is safe to delete this app (login to the Cyclic website, click the A colorful flower with a black center

  Description automatically generated with medium confidence (Options & configs) button on your app and click **Delete app…** button under "**Advanced**").

# 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 Academic Policy. No part

\* of this assignment has been copied manually or electronically from any other source   
\* (including 3rd party web sites) or distributed to other students.  
\*   
\* Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\*  
\* Online (Cyclic) Link:   
\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* Compress (.zip) your assignment folder with all files/folders included 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.