WEB322 Assignment 2

# Submission Deadline:

Tuesday, February 4th @ 11:59 PM

# Assessment Weight:

9% of your final course Grade (including week 3 and week 4 in-class assignments, 0.5% each)

# Objective:

Create and publish a web app that uses multiple routes which serve static content (text / json) as well as create a "data 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 "**NHSiteData.json**" and "**provinceAndTerritoryData.json**" files (available within a data.zip file on Blackboard)
* Within the "modules" folder, create a new file called: "**data-service.js**"

Now we now have the data for the assignment.

# Part 2: Writing data-service.js

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

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

const siteData = require("../data/NHSiteData");

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

This will automatically read both files and generate two arrays of objects: "siteData" and "provinceAndTerritoryData".

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

let sites = [];

This will be the completed array of Site objects, after processing the above "siteData" and "provinceAndTerritoryData" arrays.

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

* **initialize()**  
    
  The purpose of this function is to fill the "sites" array (declared above), by adding copies of all the **siteData** objects. The following is an example of an added object:

{

    "siteId": "BC010",

    "site": "Butchart Gardens",

    "description": "Internationally known gardens, including remarkable Sunken Garden in a former limestone quarry; unique combination of 3 aspects of Canadian gardening history: early 20th-century estate garden, early twentieth century beautification movement, and the Victorian bedding out system",

    "date": 1904,

    "dateType": "established",

    "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/d9/Sunken\_Garden.jpg/360px-Sunken\_Garden.jpg",

    "location": "Brentwood Bay",

    "latitude": "48.56528",

    "longitude": "-123.46944",

    "designated": 2004,

    "provinceOrTerritoryCode": "BC"

}

However, each of these objects must **also** include a new **"provinceOrTerritoryObj**" property. The value of the "provinceOrTerritoryObj" property should be the corresponding provinceOrTerritory object from the **provinceAndTerritoryData.json** file, whose "code" value matches the "**provinceOrTerritoryCode**" for the "siteData" object.

For example, in the above "siteData" object, we have a "provinceOrTerritoryCode" value of "BC". Therefore, we must add an additional "provinceOrTerritoryObj" property with the value of { "code": "BC", "name": "British Columbia", "type": "province", "region": "West Coast", "capital": "Victoria" }

since the "code" value for the object is also "BC" within the **provinceAndTerritoryData.json** file.

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

{

    "siteId": "BC010",

    "site": "Butchart Gardens",

    "description": "Internationally known gardens, including remarkable Sunken Garden in a former limestone quarry; unique combination of 3 aspects of Canadian gardening history: early 20th-century estate garden, early twentieth century beautification movement, and the Victorian bedding out system",

    "date": 1904,

    "dateType": "established",

    "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/d9/Sunken\_Garden.jpg/360px-Sunken\_Garden.jpg",

    "location": "Brentwood Bay",

    "latitude": "48.56528",

    "longitude": "-123.46944",

    "designated": 2004,

    "provinceOrTerritoryCode": "BC",

    "provinceOrTerritoryObj": {

      "code": "BC",

      "name": "British Columbia",

      "type": "province",

      "region": "West Coast",

      "capital": "Victoria"

    }

}

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

{

    "siteId": "QC001",

    "site": "Acton Vale Railway Station (Grand Trunk)",

    "description": "A small passenger terminal with dormer, turret and bellcast roof; symbolic of the expansion of the Grand Trunk Railway",

    "date": 1900,

    "dateType": "completed",

    "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/a/a5/Gare\_d%27Acton\_Vale\_2.JPG/320px-Gare\_d%27Acton\_Vale\_2.JPG",

    "location": "Acton Vale",

    "latitude": "45.6482917",

    "longitude": "-72.5640139",

    "designated": 1976,

    "provinceOrTerritoryCode": "QC"

}

This has a "provinceOrTerritoryCode" of "QC". From the "provinceAndTerritoryData.json" file, the object of the province or territory with code "QC" is { "code": "QC", "name": "Quebec", "type": "province", "region": "Central Canada", "capital": "Quebec City" }. Therefore, the new object to be added to the "sites" array should be:

{

    "siteId": "QC001",

    "site": "Acton Vale Railway Station (Grand Trunk)",

    "description": "A small passenger terminal with dormer, turret and bellcast roof; symbolic of the expansion of the Grand Trunk Railway",

    "date": 1900,

    "dateType": "completed",

    "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/a/a5/Gare\_d%27Acton\_Vale\_2.JPG/320px-Gare\_d%27Acton\_Vale\_2.JPG",

    "location": "Acton Vale",

    "latitude": "45.6482917",

    "longitude": "-72.5640139",

    "designated": 1976,

    "provinceOrTerritoryCode": "QC",

    "provinceOrTerritoryObj": {

      "code": "QC",

      "name": "Quebec",

      "type": "province",

      "region": "Central Canada",

      "capital": "Quebec City"

    }

}

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

* **getAllSites()**

This function simply returns the complete "sites" array

* **getSiteById(id)**

This function will return a specific "site" object from the "sites" array, whose "siteId" value matches the value of the "id" parameter, ie: if getSiteById("ON016") was invoked, the following site object would be returned:

{

    "siteId": "ON016",

    "site": "Public Grounds of the Parliament Buildings",

    "description": "The focal point for national celebrations in Ottawa; the grounds were originally designed by Calvert Vaux, and since supplemented by 18 monuments and memorials",

    "date": 1875,

    "dateType": "initial completion",

    "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/5/5f/East\_Block\_and\_Langevin\_Block.jpg/320px-East\_Block\_and\_Langevin\_Block.jpg",

    "location": "Ottawa",

    "latitude": "45.424807",

    "longitude": "-75.699234",

    "designated": 1976,

    "provinceOrTerritoryCode": "ON",

    "provinceOrTerritoryObj": {

      "code": "ON",

      "name": "Ontario",

      "type": "province",

      "region": "Central Canada",

      "capital": "Toronto"

    }

}

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

* **getSitesByProvinceOrTerritoryName(name)**

The purpose of this function is to return an array of objects from the "sites" array whose "name" value of the embedded "provinceOrTerritoryObj" object matches the "name" parameter. However, it is important to note that the "name" parameter may contain only part of the "name" value of the embedded "provinceOrTerritoryObj" object and must be case insensitive. For example:

getSitesByProvinceOrTerritoryName("Ontario");

would return all the sites from your "sites" array whose "name" value of the embedded "provinceOrTerritoryObj" object contains the string "Ontario" (ignoring case). For example, if your "sites" array contained the following objects, they would be returned (28 site objects):

[

    {

      "siteId": "ON001",

      "site": "Adelaide Hunter Hoodless Homestead",

      "description": "The childhood home of activist and organizer Adelaide Hunter Hoodless, educational reformer and co-founder of the Women's Institute, the National Council of Women of Canada and the Victorian Order of Nurses",

      "date": 1839,

      "dateType": "completed",

      "image": "https://upload.wikimedia.org/wikipedia/commons/5/5a/Hoodless\_Homestead\_NHS.jpg",

      "location": "Brant County",

      "latitude": "43.23694",

      "longitude": "-80.29667",

      "designated": 1995,

      "provinceOrTerritoryCode": "ON",

      "provinceOrTerritoryObj": {

        "code": "ON",

        "name": "Ontario",

        "type": "province",

        "region": "Central Canada",

        "capital": "Toronto"

      }

    },

    {

      "siteId": "ON002",

      "site": "Algoma Central Engine House",

      "description": "A well-preserved example of a brick engine house, and the first in Canada to have an internal turntable",

      "date": 1912,

      "dateType": "completed",

      "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/3/31/Algoma\_Central\_Engine\_House\_23.JPG/320px-Algoma\_Central\_Engine\_House\_23.JPG",

      "location": "Sault Ste. Marie",

      "latitude": "46.5281611",

      "longitude": "-84.3505389",

      "designated": 1992,

      "provinceOrTerritoryCode": "ON",

      "provinceOrTerritoryObj": {

        "code": "ON",

        "name": "Ontario",

        "type": "province",

        "region": "Central Canada",

        "capital": "Toronto"

      }

    },

    {

      "siteId": "ON003",

      "site": "Algonquin Provincial Park",

      "description": "The first provincial park in Canada, noted for its pioneering role in park management, visitor interpretation programs and the development of park buildings and structures, as well as its role in inspiring artists such as the Group of Seven",

      "date": 1893,

      "dateType": "established",

      "image": "https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/Algonquin\_Cache\_Lake\_Lookout.JPG/320px-Algonquin\_Cache\_Lake\_Lookout.JPG",

      "location": "Nipissing",

      "latitude": "45.58417",

      "longitude": "-78.35833",

      "designated": 1992,

      "provinceOrTerritoryCode": "ON",

      "provinceOrTerritoryObj": {

        "code": "ON",

        "name": "Ontario",

        "type": "province",

        "region": "Central Canada",

        "capital": "Toronto"

      }

    },

... ...

]

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

* **getSitesByRegion(region)**

The purpose of this function is to return an array of objects from the "sites" 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:

getSitesByProvinceOrTerritoryName("Prairie Provinces");

or

getSitesByProvinceOrTerritoryName("prairie provinces");

would return all the sites from your "sites" array whose "region" property contains the string "Prairie Provinces" or "prairie provinces" (ignoring case), i.e., all Prairie Provinces' sites from "sites" array would be returned.

# Part 3: Testing & Refactoring data-service.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 "sites" array to be filled with data).

**NOTE:** You should be able to run this file using the command "node modules/data-service.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 5 functions created (initialize(), getAllSites(), getSiteById(id), getSitesByProvinceOrTerritoryName(name), getSitesByRegion(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 "sites" array is filled with objects)
  + **getAllSites()** should resolve with the completed "sites" array
  + **getSiteById(id)** should resolve with the found "site" object, and reject with an appropriate message (ie: unable to find requested site) if the site was not found
  + **getSitesByProvinceOrTerritoryName(name)** should resolve with the found "site" objects, and reject with an appropriate message (ie: unable to find requested sites)
  + **getSitesByRegion(region)** should resolve with the found "site" objects, and reject with an appropriate message (ie: unable to find requested sites)

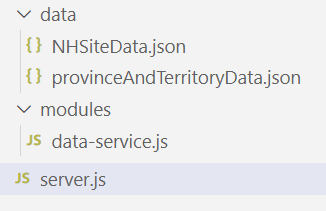
Finally, we must make sure this file functions as a "[module](https://webprogrammingtoolsandframeworks.sdds.ca/Web-Server-Introduction/modules-node-package-manager#modules)", either by manually adding all 5 functions to the "module.exports" object individually, or by assigning them at once using an object literal shorthand, ie:

module.exports = { initialize, getAllSites, getSiteById, getSitesByProvinceOrTerritoryName, getSitesByRegion }

# 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):  
    
  
* 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" must be 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 siteData = require("./modules/data-service");  
    
  This will ensure that the functions that we wrote in parts 2 & 3 will be available on the **siteData** object (ie: siteData.initialize(), siteData.getAllSites(), 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 "sites" array has been successfully built within our "data-service" module. Therefore, we must invoke the **siteData.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 "/sites"**  
    
  This route is responsible for responding with all of the sites (array) from our siteData module
* **GET "/sites/site-id-demo"**  
    
  In this route, you will demonstrate the **getSiteById** functionality, by invoking the function with a known id value, e.g., siteData.getSiteById("ON016") 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 **getSiteById** fails. If this is the case, simply respond with the error text.
* **GET "/sites/province-or-territory-demo"**  
  In this route, you will demonstrate the **getSitesByProvinceOrTerritoryName** functionality, by invoking the function with a string (of full or partial province or territory name, e.g "Ontario" or "ontar") as parameter. Once the function has been resolved successfully, respond with the returned array.  
    
  **NOTE:** Do not forget to include code to handle the situation where **getSitesByProvinceOrTerritoryName** fails. If this is the case, simply respond with the error text.
* **GET "/sites/region-demo"**  
  In this route, you will demonstrate the **getSitesByRegion** functionality, by invoking the function with a known region value from your data set (using a part of the region name in lower case, e.g., "Prairie Provinces" ). Once the function has resolved successfully, respond with the returned array.  
    
  **NOTE:** Do not forget to include code to handle the situation where **getSitesByRegion** 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://as2-322-sample.vercel.app/

# 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 (with all files/folders in the project) 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.