WEB700 Assignment 1

# Assessment Weight:

5% of your final course Grade

# Objective:

This first assignment will focus primarily on applying JavaScript fundamental concepts including: defining / using variables, arrays, functions, looping / conditional control structures and commenting code.

The main objective is to create a very simple "Web Server Simulator" function that responds to requests for data using a specific HTTP Verb (ie: "GET", "POST", etc) and path (ie: "/home") combination. Once this is complete, a second "tester" function will be created to invoke the "Web Server Simulator" function with a random request from a predefined list.

# Specification:

This assignment will consist of a single JavaScript file (".js"), developed using [Visual Studio Code](https://code.visualstudio.com/). To execute our code, we will be using the [Node.js](https://nodejs.org/en/) JavaScript runtime from within the [Integrated Terminal](https://code.visualstudio.com/docs/editor/integrated-terminal) included with Visual Studio Code.

### **Step 1:** Download & Install the Required Software

Download and install the following software for your operating system:

* [Visual Studio Code](https://code.visualstudio.com/)
* [Node.js](https://nodejs.org/en/)
* [Git](https://git-scm.com/downloads) Command Line Tool (Used later in the course)

### **Step 2:** "Hello World"

With all our software successfully installed, it's time to create the file and make sure that we can execute JavaScript code using Node.js

* Somewhere on your computer, create a folder for your assignment 1 code
* Open Visual Studio Code and Choose **File > Open** to open your newly created folder
* Using Visual Studio Code, create a new file within the folder called "a1.js"
* In the code editor, type the JavaScript code: **console.log("Hello World");**
* Save the file and open the “Integrated Terminal” using either the “View” menu and choosing “Terminal” or using the key combination ctrl + `
* In the "Integrated Terminal" type the command: **node a1.js** to execute your code.

### **Step 3:** Creating the "Server Paths"

Now that we know that we can execute JavaScript code, we can remove the "Hello World" code and create some predefined paths (later referred to as "routes") for our web server simulator to "listen" for.

Use the following table to create 3 variables (serverVerbs, serverPaths, and serverResponses) at the top of the file that each store an array with the following data (strings):

**Note**: Fill in your own Student Name / Email in the spaces indicated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** | **[5]** |
| serverVerbs | "GET" | "GET" | "GET" | "POST" | "GET" | "POST" |
| serverPaths | "/" | "/about" | "/contact" | "/login" | "/panel" | "/logout" |
| serverResponses | "Welcome to WEB700 Assignment 1" | "This course name is WEB700. This assignment was prepared by ***Student Name***" | "***Student Email***  ***Student Name***” | “Hello, User Logged In" | "Main Panel" | "Logout Complete. Goodbye” |

By defining the arrays in the above order, we can associate a specific "server Response" with the matching "verb" + "path" combination. For example, a "POST" request on the "/login" path should cause the "web server simulator" function (defined below) to return a message containing the text "User Logged In". Similarly, a "GET" request on the default path ("/") should return a string containing the text "Welcome to WEB700 Assignment 1"

### **Step 4:** Creating the "web server simulator" Function - "httpRequest"

The next step is to implement the actual function that will do the work of our simple web server (see: [JavaScript Functions in Week 2](https://sictweb.github.io/web700/notes/week02)). We'll call this function "httpRequest" and it will accept two parameters: **httpVerb** and **path**. This function will behave according to the following specification:

* If the **httpVerb** parameter and **path** parameter can **both** be found at the **same index** in the global "serverVerbs" and "serverPaths" arrays (see step 3), the matching "serverResponses" value will be returned including the status code "200".  
    
  For example, if **httpRequest("GET", "/")** is invoked, then the function should return "200: Welcome to WEB700 Assignment 1", since the serverVerbs[0] == "GET" and "serverPaths[0] == "/". Similarly, if **httpRequest("POST", "/logout")** is invoked, then the function should return "200: Logout Complete", since serverVerbs[5] == "POST" && serverPaths[5] == "/logout".   
    
  The idea here is that the **httpVerb** parameter must be found in the **serverVerbs** array at the same **index** as the **path** parameter in the **serverPaths** array. If this is the case, the **serverResponses** array at the same **index** will be returned along with the 200 status code.  
    
  **HINT:** You can use a "for loop" to loop through all the server paths (using serverPaths.length) and make sure that the current index of the loop (ie 0) returns the requested value from the **serverVerbs** array AND the **serverPaths** array.
* If a mismatch occurs, ie: the **httpVerb** value of **"GET"** is found at index 0 and the **path** value is found at index 3 (ie: **httpRequest("GET", "/login")**), then the server has encountered an error with the client request (404). If this is the case, the function should instead return the string: "404: Unable to process ***httpVerb*** request for ***path***" where **httpVerb** and **path** are the values passed to the function (ie: "404: Unable to process **GET** request for **/login**"

### **Step 5:** Manually Testing the "httpRequest" Function

Before we create our next function, manually test the "httpRequest" function by calling the it with each of the possible combinations including a non-matching combination. For example:

* console.log(httpRequest("GET", "/")); // shows "200: Welcome to WEB700 Assignment 1"
* console.log(httpRequest("GET", "/about")); // shows "200: This Assignment was prepared by ***Student Name***"

and so on, including a "404" error:

* console.log(httpRequest("PUT", "/")); // shows "404: Unable to process PUT request for /"

### **Step 6:** Automating the Tests by creating a "automateTests" Function

Now that we're confident that our "httpRequest" function is behaving correctly, let's introduce another function that will continuously invoke it with a random request that consists of a "GET" or "POST" verb along with one of our predefined paths (including some "notFound" paths). It will consist of 0 parameters, since the testing logic and parameters will be defined entirely within the function.

However, before we begin to create the function, we will need to first introduce a new "utility" function near the top of our code that will help us generate a random integer value between 0 and a specified maximum.

You can find this function here: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random>

It's called "getRandomInt(max)" and you are free to copy/paste the function for use within your code.

With this function in place, we can now define our new "automateTests" function to use the following code / logic:

* As a first step, define the following arrays:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **[0]** | **[1]** | **[2]** | **[3]** | **[4]** | **[5]** | **[6]** | **[7]** |
| testVerbs | "GET" | "POST" |  |  |  |  |  |  |
| testPaths | "/" | "/about" | "/contact" | "/login" | "/panel" | "/logout" | "/randomPath1" | "/randomPath2" |

* Next, within this function, define another function called **randomRequest** (accepts 0 parameters) - we will use the [setInterval](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval) function to repeat this function over and over again every 1 second
* Inside the new randomRequest function, declare the following variables:
  + **randVerb** - stores a random value from the **testVerbs** array (ie: testVerbs[*some random index between 0 and 1 inclusive*])
  + **randPath** - stores a random value from the **testPaths** array (ie: testPaths[*some random index between 0 and 7 inclusive]*
* Now that these variables are declared, we can use them to invoke our "httpRequest" function with the random values! (**NOTE:** be sure to **console.log()** the result, so that we can see the result of the request in the console.
* Underneath the **randomRequest** function definition (but still inside the **automateTests** function), use the [setInterval](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval) function to execute the **randomRequest** function over and over again every 1 second (1000 milliseconds)

### **Step 7:** Invoke the "automateTests" function

At the bottom of your code, beneath all function definitions and variable declarations, add the code to invoke the "automateTests" function. When we run our a1.js file, this function, this function will be responsible for kicking off the testing process.

### **Step 8:** Running the Code

As a final test to ensure that the code is working properly, open the **Integrated Terminal** and execute the command: **node a1.js**. At this point, you should see your program continuously outputting values every second to the console, ie:

404: Unable to process GET request for /logout  
404: Unable to process GET request for /randomPath1  
200: User Logged In  
200: Main Panel  
200: User Logged In

…

etc, etc.   
  
What we are essentially doing here is testing a very high-level approximation of a web server that responds to random requests from clients.

**NOTE:** This code will continuously generate random requests once every second. However, there is nothing in our code to put a stop to it. The only way to stop the program at this point is to use the command **ctrl + c** in the **Integrated Terminal** to stop the code execution.

## Assignment Submission:

1. Add the following declaration at the top of your a1.js file:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* WEB700 – Assignment 1  
\* 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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Submit your a1.js file online using My.Seneca along with GitHub link, screenshots and small video recording with appropriate name for all files. Please submit each file as separate submission.

## 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 (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.