WEB700 Assignment 3

# Submission Deadline:

Saturday, June 17th @ 11:59 PM

# Assessment Weight:

9% of your final course Grade

# Objective:

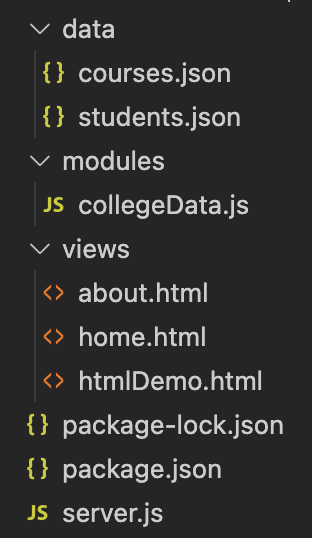
Practice creating a web server using Node.js & Express.js. Additionally, we will practice writing HTML Containers and Media Elements to be displayed when the client (browser) requests specific routes.

# Specification:

This assignment will consist of our first, working web server created using the Express.js framework. We will continue to use our collegeData module from assignment 2, as well as the "data" directory containing our student and course information. Additionally, we will leverage existing HTML code using the [Bootstrap 4 framework](https://getbootstrap.com/docs/4.3/getting-started/introduction/) to kickstart our own HTML for the assignment. Lastly, we will practice creating tables, lists, images and media from scratch in an "HTML Demo" page.

### **Step 1:** Project Setup

To begin this assignment, follow the following procedure:

* Create a new folder called web700-app somewhere on your system (this will be the folder containing all of our code) and open it in Visual Studio Code
* Create the following file / folder structure for our assignment:   
    
  
* You will notice that some of this looks very similar to what we created in Assignment 2. That is because we are actually starting this assignment with some of the files from the last assignment! **NOTE:** If your last assignment was incomplete and your collegeData.js did not work properly, please **email your professor for a working copy of collegeData.js** so that you can complete this assignment.
* For the next step, you can fill in the following files with existing code:
  + **courses.json** – this is the same file as your last assignment. You can find a fresh copy here: <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A2/courses.json>
  + **students.json** – this is the same file as your last assignment. You can find a fresh copy here: <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A2/students.json>
  + **collegeData.js** – this is literally the same collegeData.js from Assignment 2 (email your professor for a working copy if required)
  + **about.html** – the "starter html" can be found here:   
    <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/about.html.txt>
  + **home.html**– the "starter html" can be found here:   
    <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/home.html.txt>
  + **htmlDemo.html**– the "starter html" can be found here:   
    <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/htmlDemo.html.txt>

### **Step 2:** Adding Functionality to "collegeData"

Before we create a package.json file, install express and start writing our code in server.js, we must first add some new functionality to our **collegeData module.** In this case, it is two functions (both of which **return promises**) added to module.exports:

### getStudentsByCourse(course)

* This function will provide an array of "student" objects whose **course** property matches the ***course*** parameter (ie: if ***course*** is 5 then the array will consist of only students who belong to course 5 ) 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".

### getStudentByNum(num)

* This function will provide a single "student" object whose **studentNum** property matches the ***num*** parameter (ie: if ***num*** is 22 then the "student" object returned will be "Ellette Emby" ) using the **resolve** method of the returned promise.
* If for no student was found, this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

### **Step 3:** Writing server.js

Once you're confident that everything is working properly up to this point, we can start to create and test our web server:

* As a first step, open the integrated terminal in Visual Studio Code and enter the command **npm init** and follow along with the text-based wizard in the terminal (you can accept all of the default values). Once this is completed, you should see a "package.json" materialize in the root of your project folder
* Next, execute the command: **npm install express** to grab the **express** module and store it in a newly created "node\_modules" folder (you should also see a package.lock.json appear in the root of your project folder.
* Now, we can finally start writing code within our server.js file. As a starting point, you can use the following code (copied almost verbatim from the "getting started with Heroku" guide on the course website, here: <https://web700.ca/getting-started-with-heroku> ( **NOTE**: We will not be pushing our code to Heroku just yet – we'll save that for Assignment 4)

var HTTP\_PORT = process.env.PORT || 8080;

var express = require("express");

var app = express();

// setup a 'route' to listen on the default url path

app.get("/", (req, res) => {

    res.send("Hello World!");

});

// setup http server to listen on HTTP\_PORT

app.listen(HTTP\_PORT, ()=>{console.log("server listening on port: " + HTTP\_PORT)});

* Before we move on, try running the server locally by executing the command "**node server.js**" in the integrated terminal. Once you see the text "server listening on port 8080", proceed to your favourite web browser and navigate to <http://localhost:8080> to see the text "Hello World!" – this signifies that our server code is working.
* With our server in good working order, we can delete the "/" route that returns "Hello World" add the following routes (NOTE: For some of these routes to work, you will need to **require** certain modules including **path** and your **collegeData module**:  
    
  GET /students
* This route will return a JSON formatted string containing all of the students resolved from the **getAllStudents** function ( If the promise didn't resolve successfully, send the following object back as JSON {message:"no results"} ), ie: <https://arcane-fortress-97059.herokuapp.com/students>
* **Additionally**, this route will be able to handle the following optional query parameter:
  + /students?course=***value***
    - Return a JSON string consisting of all students where ***value*** could be one of 1, 2, 3, … 7 (there are currently 7 courses in the dataset) " - this can be accomplished by calling the **getStudentsByCourse(course)** function of your collegeData, ie: <https://arcane-fortress-97059.herokuapp.com/students?course=2>

GET /tas

* This route will return a JSON formatted string containing all of the managers resolved from the **getTAs** function ( If the promise didn't resolve successfully, send the following object back as JSON {message:"no results"} ), ie: <https://arcane-fortress-97059.herokuapp.com/tas>

GET /courses

* This route will return a JSON formatted string containing all of the courses resolved from the **getCourses** function ( If the promise didn't resolve successfully, send the following object back as JSON {message:"no results"} ), ie: <https://arcane-fortress-97059.herokuapp.com/courses>

GET /student/***num***

* This route will return a JSON formatted string containing a single student whose **studentNum** property matches the **num** parameter in the route, ie: <https://arcane-fortress-97059.herokuapp.com/student/5> returns student 5 and <https://arcane-fortress-97059.herokuapp.com/student/6> returns student 6 and so on. The newly created **getStudentByNum** function can help with this task.

GET /

* This route simply returns the html code from the **home.html** file located within the **views** directory

GET /about

* This route simply returns the html code from the **about.html** file located within the **views** directory

GET /htmlDemo

* This route simply returns the html code from the **htmlDemo.html** file located within the **views** directory

[ no matching route ]

* If the user enters a route that is not matched with anything in your app (ie: http://localhost:8080/asdf) then you must return the custom message "**Page Not Found**" with an HTTP status code of **404**.
* **Note:** at this point, you may wish to send a custom 404 page back to the user instead of a message (completely optional, but everyone loves a good 404 page). Here’s some ***inspiration*** for your own designs: <https://medium.com/@CollectUI/404-page-design-inspiration-march-2017-f6d9f7efd054>
* As a final step Before we can test our server, we must make a small update to the code ***surrounding*** the app.listen() call at the bottom of the server.js file. This is where the **initialize()** method from our **collegeData** module comes into play.

Fundamentally, initialize() is responsible for reading the .json files from the "data" folder and parsing the results to create the "global" (to the module) dataCollection object, containing the "students" and "courses" arrays. However, it also returns a **promise** that will only **resolve** successfully once the files were read correctly and the "students" and "courses" arrays were correctly loaded with the data.

Similarly, the promise will **reject** if any error occurred during the process. Therefore, we must **only call app.listen()** if our call to the **initialize()** method is successful, ie: .**then(() => { //start the server })**.

If the initialize() method invoked **reject**, then we should not start the server (since there will be no data to fetch) and instead a meaningful error message should be sent to the console, ie: **.catch((err)=>{ /\*output the error to the console \*/})**

* Now that your routes are complete, you can test your server running on localhost against the sample online here: [https://arcane-fortress-97059.herokuapp.com](https://arcane-fortress-97059.herokuapp.com/)

### **Step 4:** Updating the Views "home.html", "about.html" and "htmlDemo.html"

With our server in good working order, we can now proceed to update the three **static** resources, ie the html pages in our **views** directory: home.html, about.html and htmlDemo.html.

**NOTE:** For each of the below pages, please locate the "Student Name" in the navbar and change it to match your own name.

### **"Home" - home.html:**

1. Add a **professional greeting** to the visitor, ie: "Welcome to my website, I will be demonstrating HTML5 principles and techniques, Routing in Express"… and so on.
2. Add a relevant **header** as a title for the next section (step 3)
3. Add a **short paragraph** that introduces this website, for example: "This site consists of 3 static views (Home, About, HTML Demo) as well as 5 routes that supply data , including: All Students, Students by Course, A Single Student By Student Number, All Managers and All Courses.  **NOTE:** This paragraph **must include** all relevant links to the **8 pages** – for the "Student by Student Number link, just use a working link to a random student, ie /student/2 and for the "Students by Course" link, you can use a working link to a random course, ie /students?course=3
4. Add a relevant **header** as a title for the next section (step 5)
5. Add a **short paragraph** introducing HTML 5, ie: "This site utilizes HTML5: a markup language used for structuring and presenting content on the World Wide Web"… and so on.

### **"About" - about.html:**

1. Add a **professional greeting** that welcomes the visitor to the current page.
2. Add a relevant **header** as a title for the next section (step 3)
3. Add a **short paragraph** that states some information about yourself including a minimum of your student name, id and program

### **"HTML Demo" - htmlDemo.html:**

The HTML Demo page will be the busiest page and will consist of a playground for us to practice creating HTML elements. The page will be divided into sections, with each section starting with a relevant **header** ie, "Tables", "Lists", etc.

"Tables" Section

* This section must have a relevant **header** to distinguish it from the rest of the page
* Next, it must consist of **two** HTML tables that must contain a **minimum** of **3 rows** and **3 columns** and must make use of the following elements / properties:
  + caption
  + thead,tbody,tfoot
  + th, td, tr
  + the "rowspan" property and the "colspan" property
* The cells in the tables **cannot be empty**, but can contain any content that you wish (images, links, text, etc)
* At the bottom of the section, write a short **paragraph** describing the content in the section

"Lists" Section

* This section must have a relevant **header** to distinguish it from the rest of the page
* Next, it must consist of a single list (either unordered or ordered) with a minimum of **5** **items** – at least one of these items should contain a **list** that is a **different type** than the parent list (ie: if the parent is an ordered list, then one of the items should contain an unordered list, and vice versa.) with at least **3 items.**
* The items in the list **cannot be empty**, but can contain any content that you wish (images, links, text, etc).
* At the bottom of the section, write a short **paragraph** describing the content in the section

"Images" Section

* This section must have a relevant **header** to distinguish it from the rest of the page
* Next, it must consist of at least 3 images that include "alt" text. You're welcome to use any images you like (as long as they can be reused for "noncommercial" purposes). **NOTE:** the images **do not** have to exist on your sever – you may use complete paths to the images, ie:  
    
  <https://upload.wikimedia.org/wikipedia/commons/6/69/Snowboarder_in_halfpipe.jpg>
* At the bottom of the section, write a short **paragraph** describing the content in the section

"Audio" Section

* This section must have a relevant **header** to distinguish it from the rest of the page
* This section will consist of an HTML 5 audio player. For the audio source, you may use the following files if you wish (not mandatory if you have found something else that you like better)
* <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/sample-audio.mp3>
* <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/sample-audio.ogg>
* At the bottom of the section, write a short **paragraph** describing the content in the section

"Video" Section

* This section must have a relevant **header** to distinguish it from the rest of the page
* This section will consist of an HTML 5 video player. For the video source, you may use the following files if you wish (not mandatory if you have found something else that you like better)
  + <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/sample-video.mp4>
  + <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/sample-video.ogg>
  + <https://pat-crawford-sdds.netlify.app/shared/fall-2021/web700/A3/sample-video.webm>
* At the bottom of the section, write a short **paragraph** describing the content in the section.

# Assignment Submission:

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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* WEB700 – Assignment 03  
\* 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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* Compress (.zip) your web700-app folder and submit the .zip file to My.Seneca under   
  **Assignments** -> **Assignment** 3

# Important Note:

* **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
* After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
* 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.