WEB322 Assignment 3

Assessment Weight:

9% of your final course Grade

Objective:

Practice creating a web server using Node.js & Express.js and client (browser) requests specific routes.

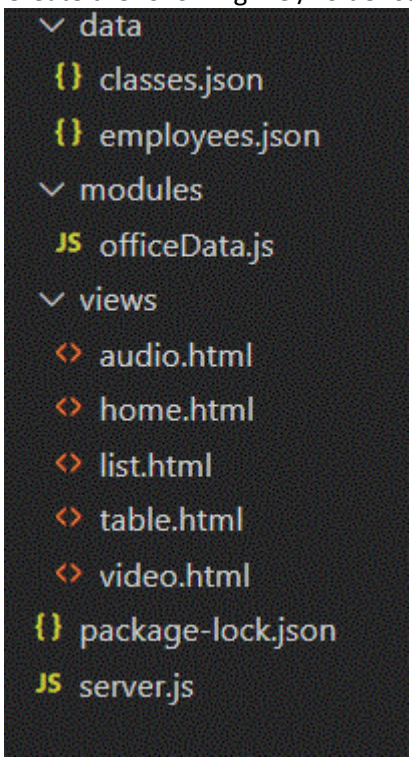
Specification:

This assignment will consist of our first, working web server created using the Express.js framework. We will continue to use our officeData module from assignment 2, as well as the "data" directory containing our employees and classes information.

Step 1: Project Setup

To begin this assignment, follow the following procedure:

- Create a new folder called web322-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!
- For the next step, you can fill in the following files with existing code:

- **classes.json** – this is the same file as your last assignment – supplied.
- **employees.json** – this is the same file as your last assignment – supplied.
- **officeData.js** – this is literally the same officeData.js from Assignment 2 – supplied.
- **home.html**– to be created by you. See description below.
- **audio.html -** to be created by you. See description below.
- **video.**html - to be created by you. See description below.
- **list.html** - to be created by you. See description below.
- **table.html** - to be created by you. See description below.

Adding Functionality to "officeData"

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 **officeData module.** In this case, it is a function (which **returns promise**) added to module.exports:

getEmployeeByNum(num)

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

Step 2: 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://web322.ca/getting-started-with-heroku (**NOTE**: We will not be pushing our code to Heroku just yet – we'll save that for Assignment 4 or later)

```
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 **officeData module**:

GET /PartTimer

- This route will return a JSON formatted string containing all of the employees resolved from the **getPartTimers** function (If the promise didn't resolve successfully, send the following object back as JSON {message:"no results"})

GET /

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

GET /employee/***num***

- This route will return a JSON formatted string containing a single employee whose **employeeNum** property matches the **num** parameter in the route, ie: http://localhost:8080/employee/5 returns employee 5 and http://localhots:8080/employee/6 returns employee 6 and so on. The newly created **getEmployeeByNum** function can help with this task.

GET /audio

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

GET /video

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

GET /table

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

GET /list

- This route simply returns the html code from the **list.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 **officeData** 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 "employees" and "classes" arrays. However, it also returns a **promise** that will only **resolve** successfully once the files were read correctly and the "employees" and "classes" 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.

Step 3: Updating the html files in Views folder

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.

NOTE: For each of the below pages, type your own Name in the title tag inside the head section so that your name is visible in the title/tab bar.

"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 reflective paragraph** of your learning experience so far with the course.
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.

"List" - list.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
4. Add a relevant **header** as a title for the next section (step 5)
5. Create any nested list with meaningful items using healthy food, calories and nutrition. • The nested list should contain at least one ordered list and at least one unordered list.

"Table" - table.html:

- 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.). The content could be something of your choice for ex. favorites books, authors, movies, actors etc.
- At the bottom of the section, write a short **paragraph** describing the content in the section

"Audio" - audio.html:

- 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 file if you wish (not mandatory if you have found something else that you like better)

  https://scs.senecac.on.ca/~tanvir.alam/shared/fall-2018/web222/Track03.mp3

  https://scs.senecac.on.ca/~tanvir.alam/shared/fall-2018/web222/Track03.ogg

  At the bottom of the section, write a short **paragraph** describing the content in the section

"Video" - video.html:

- 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://scs.senecac.on.ca/~tanvir.alam/shared/fall-2018/web222/movie.mp4

https://scs.senecac.on.ca/~tanvir.alam/shared/fall-2018/web222/movie.ogg

- 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:

```
/*********************************************************************************
* WEB322 – 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 web322-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 (11:30PM) 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.