Assignment 3: Server-Side Scripting with Node.js & Express

The purpose of this assignment is to create a simple web application that runs on a server and responds to requests from clients. We'll use the Express framework in the Node.js environment to make an application that allows users to access information about music bands and albums.

Throughout the lesson, I'll be giving examples of URLs from my specific Codespace (*fluffy-space-enigma)* but you should substitute these with the unique domain URL of your own Codespace. Copying and pasting the URLs I give here won't work.

In the starter files, I've included a video of how the site should work when complete.

Task 1: Setup

Create a new project in **Codespaces** and initialize a blank **Node.js** project. Install **Express** and create a new Express application that runs on port **3000**. Configure the project to use the ECMAScript import syntax we learned in week 2.

Create a new folder in the root of the project called **our-modules**. This will be used for modules that we create to organize our code. In the starter files, I've given you a file called **data.js,** which contains an array of data about various bands; place this in the **our-modules** folder. Study the content of **data.js** carefully to see what's in there and how the data is structured.

Create another folder in the root called **public**. This is where we'll put files that will, eventually, be accessible to users, although we'll have to manually set up that functionality in the next task.

Task 2: Static Files

Create a **route** to handle GET requests to URL endpoints of the following form:

/pages/name-of-file

where **name-of-file** could correspond to any file name, like **index.html** or **style.css**. Use **route parameters** for this.

The route handler should look for a file in the **public** folder with that name and send it to the client. For example, if the user requests the following URL

*https://fluffy-space-enigma-639pr5x9p492476q.github.dev****/pages/home.html***

the file **/public/home.html** should be sent. In the starter files, I've given you two files that you can use to test this: **test-page.html** and **style.css**. Place these two files in the **public** folder to make sure your **route handler** is working.

It's important to know that your route handler does NOT have to handle requests for files in subfolders within the **public** folder.

Task 3: Redirects

Create a new file in **public** called **index.html** and copy the content from **test-file.html**, but change the heading text to **"Our Music Website"** and the paragraph text to **"A great place to get information about music!"** Change the **<title>** text to **"Music - Welcome**".

Create a single route for BOTH endpoints **"/"** and **"/pages"**. You can do this by passing an Array of Strings to **app.get,** with each String representing an endpoint, instead of a single String. The route handler should **redirect** the request to the route you created in **task 2,** and use it to send the **index.html** file to the client; you can do this by using

res.redirect("target endpoint goes here").

Task 4: Bands

Create a **route** for endpoints of the form:

> ***/api/music/id-of-a-band***

where **id-of-a-band** is expected to correspond to the **id** property of a **band object** in **our-modules/data.js.** Your application should find the band with that **id** and send it to the client as a **JSON** object. For example, the following URL...

> *https://fluffy-space-enigma-639pr5x9p492476q.github.dev****/api/music/hakushi-hasegawa***

Should result in the following JSON object being sent to the client:

To complete this task, use JavaScript's Array **find** method.

Note: the reason we have "/api/" in the URL is to indicate that this resource is meant to be used by other web applications, NOT directly by users. Generally speaking, JSON responses are NOT meant to be seen by users. More on this in a future lesson.

Task 5: Music Pages

Create a new file in **our-modules** called **renderAlbumPage.js**; this JavaScript file should declare a single function called **renderAlbumPage** and export it as the **default** export. Import this function into your main application script.

The function **renderAlbumPage** should do the following:

- Take the following **two parameters**:
    - The **id** of a band;
    - The **id** of an album.
- Find the correct **band** and **album** in **data.js** (you can use the **Array find** method for this);
- **Return** the HTML markup for a webpage with the following:
    - An **h1** heading with the name of the album;
    - A **paragraph tag** with the name of the band (preceded by the text "Artist: ")
    - A **paragraph tag** with the year of release (preceded by the text "Released in: ")
    - The name of the album in the **<title>** tag.

      To create the HTML elements, simply place all HTML code for the full page into a **template literal** and inject the correct data where needed. You won't be able to use methods like **document.createElement** because the **document** object is not available in the Node.js server environment. (Next week we'll learn how to generate pages from templates using some special built-in tools with **Express**.)
    - Have the above page linked with a <link> tag to the **style.css** file you created early, using the proper path format that you configured.

Use the **renderAlbumPage** function to implement a **route** for endpoints of the following form:

/music-pages/id-of-band/id-of-album

where **id-of-band** and **id-of-album** are expected to be band and album IDs that match those in **data.js.** The response should contain the webpage generated by **renderAlbumPage.** Here's an example: when accessing the following URL

*https://....github.dev****/music-pages/hakushi-hasegawa/mahōgakkō***

one should see a page in the browser that looks similar to the following:

What we've done here is called **server-side rendering,** in which an application running on a server renders an HTML page and sends it to the client in its final form. Last week, we did **client-side** rendering, in which the HTML was rendered by JavaScript code running in the browser. Both these methods are often used in web development depending on the situation, and sometimes a hybrid approach is used.

Task 6: Update Home Page

Now open **data.js** and add at least **one band or music artist** that you like and at least **one album** by that artist**.** Use the same object format that I used for the bands/albums I added. The **ids** should have values that are URL-friendly (meaning they are lower-case and don't contain any spaces or special characters other than dash and underscore).

Update **index.html** by adding an unordered list with **two** hyperlinks to album pages. You can hard-code these as static HTML, you don't have to write any JavaScript code to dynamically fetch them.

Hand In

Create a **script** in **package.json** called **"dev"** that runs the application script with the **--watch** flag, like this:

```
...
"scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "dev": "node --watch name-of-your-app-script.js"
 },
...
```

You can run this script in the terminal with the command:
npm run dev

Download your project to your local computer; make sure the **node_modules** folder is NOT included. Rename the project folder to **a3-firstname-lastname** with your own first and last name. Make sure no unnecessary or unused files are included. Zip the folder and hand it in to Brightspace. Complete the above steps in this order, as it makes a difference to us when we grade your project.

Grading

- **[1 mark]** Node.js Express server running on port 3000
- **[1 mark]** route for accessing files from public folder with endpoints "/pages/filename" using route parameters
- **[1 mark]** redirect of "/" and "/pages" to "/pages/index.html"
- route for "/music/id of band"
    - **[1 mark]** set up properly with JSON response
    - **[1 mark]** uses find method
- renderAlbumPage function
    - **[1 mark]** function with correct parameters imported from file in our-modules
    - **[1 mark]** HTML created with template values and link to style.css, using template literal
    - **[1 mark]** function used in route for "/music-pages/id of band/id of album"
- **[1 mark]** index.html updated with hyperlinks
- **[1 mark]** your own music info added to data.js

Total: 10

As usual, up to -25% will be deducted for improper hand in, poorly-organized project, poorly-formatted code, etc.