DUNWOODY
COLLEGE OF TECHNOLOGY

Lab 3

Node application utilizing RESTful API service

Due: Friday May 11th, 2018 @ 11:59pm

10 Points

Resources needed for this lab:

- Text Editor
- RESTful API data source that you can connect (API Must have good documentation)
- Github Repository
- Postman desktop app
- Language – NodeJs
- NPM modules
    - Nodemon - https://nodemon.io/
    - Hapijs - https://hapijs.com/
    - handlebars.js - https://handlebarsjs.com/
    - vision.js - https://github.com/hapijs/vision

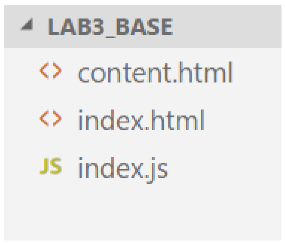
Overview:

For this lab, you will continue to build on your knowledge gained from labs 1 and 2. This lab will require you to select a RESTful API data source and utilize the data to bring about meaningful content to the end-user of your application. You will build an application that consist of three pages. You will have a home page (index.html) that introduces the end-user to your application and how you are using the data, and a content page (content.html) that utilizes the data requested via API calls. Your application will need to be responsive, aesthetically pleasing and presents the end-user with meaningful information. You are free to add you own content to this application, but data from the API call must also be present. Please place a comment above the code in your html page where data is displaying from the API.

Requirements:

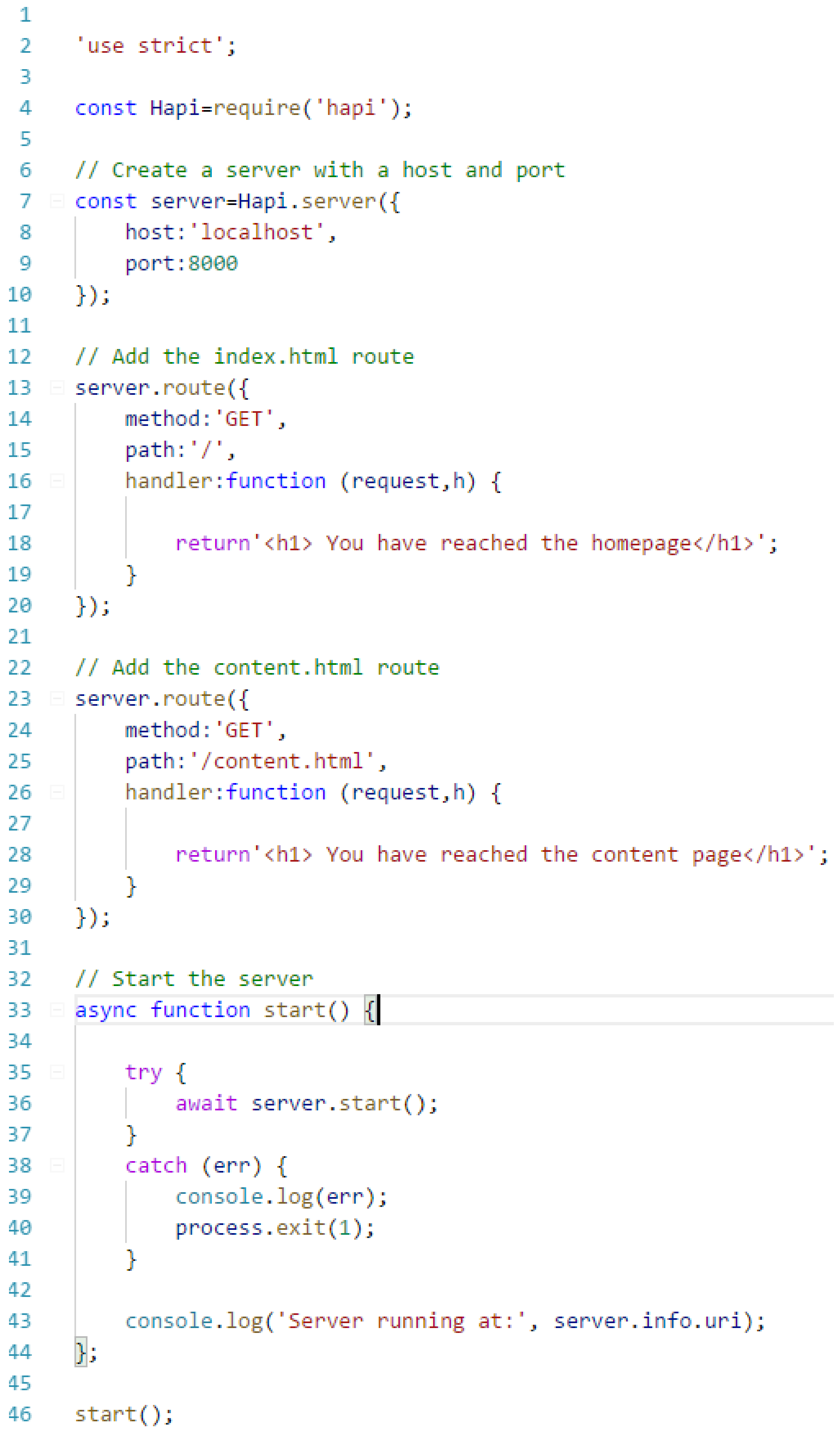
Part A

1) Download the project folder titled **"lab3"** from the Learning Management system (Canvas).
2) Version control project and initialize Git by running the command **git init**
3) Open the project folder in your preferred editor. The folder should look similar to the image below:



4) Run the **npm init** command to create package.json and the package-lock.json file
5) Install nodemon third-party module – **npm install nodemon --save**
6) Install hapi.js third-party module – **npm install hapi –save**

7) Open the index.js file and type in the code listed below
   a. Type in the code listed in the picture below(the code listed below is the code from https://hapijs.com/ feel free to copy code but make sure to make modifications based on what is depicted below:



```

'use strict';

const Hapi=require('hapi');

// Create a server with a host and port
const server=Hapi.server({
    host:'localhost',
    port:8000
});

// Add the index.html route
server.route({
    method:'GET',
    path:'/',
    handler:function (request,h) {

        return'<h1> You have reached the homepage</h1>';
    }
});

// Add the content.html route
server.route({
    method:'GET',
    path:'/content.html',
    handler:function (request,h) {

        return'<h1> You have reached the content page</h1>';
    }
});

// Start the server
async function start() {

    try {
        await server.start();
    }
    catch (err) {
        console.log(err);
        process.exit(1);
    }

    console.log('Server running at:', server.info.uri);
};

start();
```

8) Save the code listed in index.js and run the command **nodemon index.js or node index.js**
***Note: Make sure that your package.json file has index.js listed as the value for the “main” key value pair.**

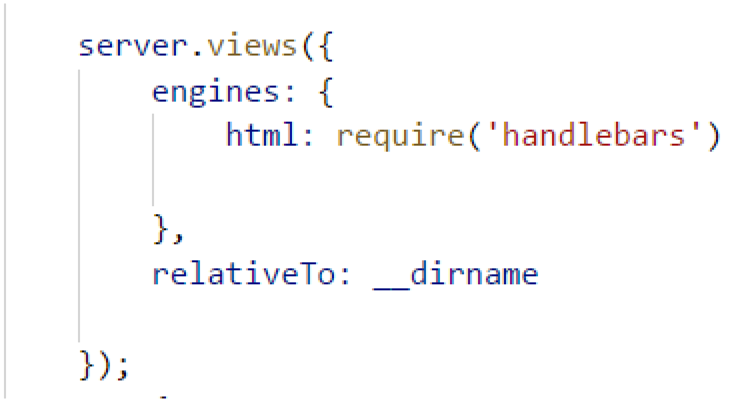
9) Open your browser and navigate to pages listed below:
   a. Localhost:8000/ → You have reached the homepage
   b. Localhost:8000/content.html → You have reached the content page
***Note: to kill server press the control key + c key**

At this point, your program should properly route and provide relevant content based on intended route. Prior to starting step 10, make sure to you select an API that will allow you to request data. The example below will walk you through New York Cities OpenData for open jobs:

https://data.cityofnewyork.us/City-Government/NYC-Jobs/kpav-sd4t
https://dev.socrata.com/foundry/data.cityofnewyork.us/swhp-yxa4

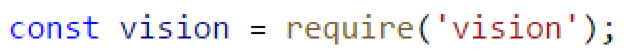
Part B – Listed below is a base example of routing and connecting to an external API. Feel free to use steps 10 – 20 as an example when connecting to an API data source and routing/rendering data to the interface.

10) This application will utilize handlebars.js to for binding data and rendering views to the user interface. Run the command **npm install –save handlebars** and add the code snippet below within the start function prior to the try/catch block. The code snippet below is registering hapi as the templating engine.



```
server.views({
    engines: {
        html: require('handlebars')

    },
    relativeTo: __dirname

});
```

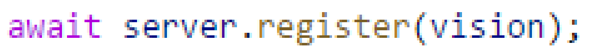
11) We will also need to import a module called vision that adds template-rendering support to hapi. Run the command **npm install vision --save** and add the statement below toward the top of your index.js file.



```
const vision = require('vision');
```

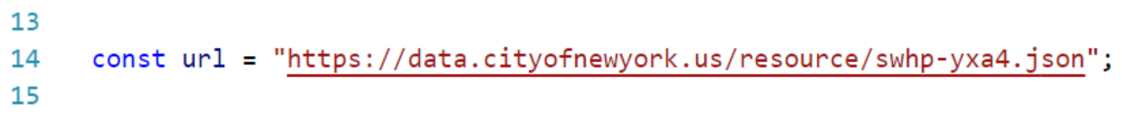
More Info: https://hapijs.com/tutorials/views if you want to render static files, consider importing a module called inert. You can find more info at https://github.com/hapijs/inert

12) Add the statement below within the start function on the index.js file



```
await server.register(vision);
```

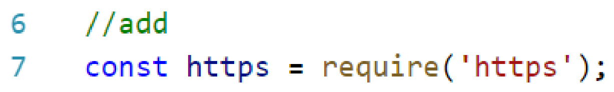
13) Once you have selected an API, make sure to study the structure of the data so that you can successfully call the necessary data needed for your application. Determine the URL you will use to retrieve data. Declare and initialize a variable to store the URL. Listed below is my sample connection string that will be placed after the require statements.



```

const url = "https://data.cityofnewyork.us/resource/swhp-yxa4.json";

```

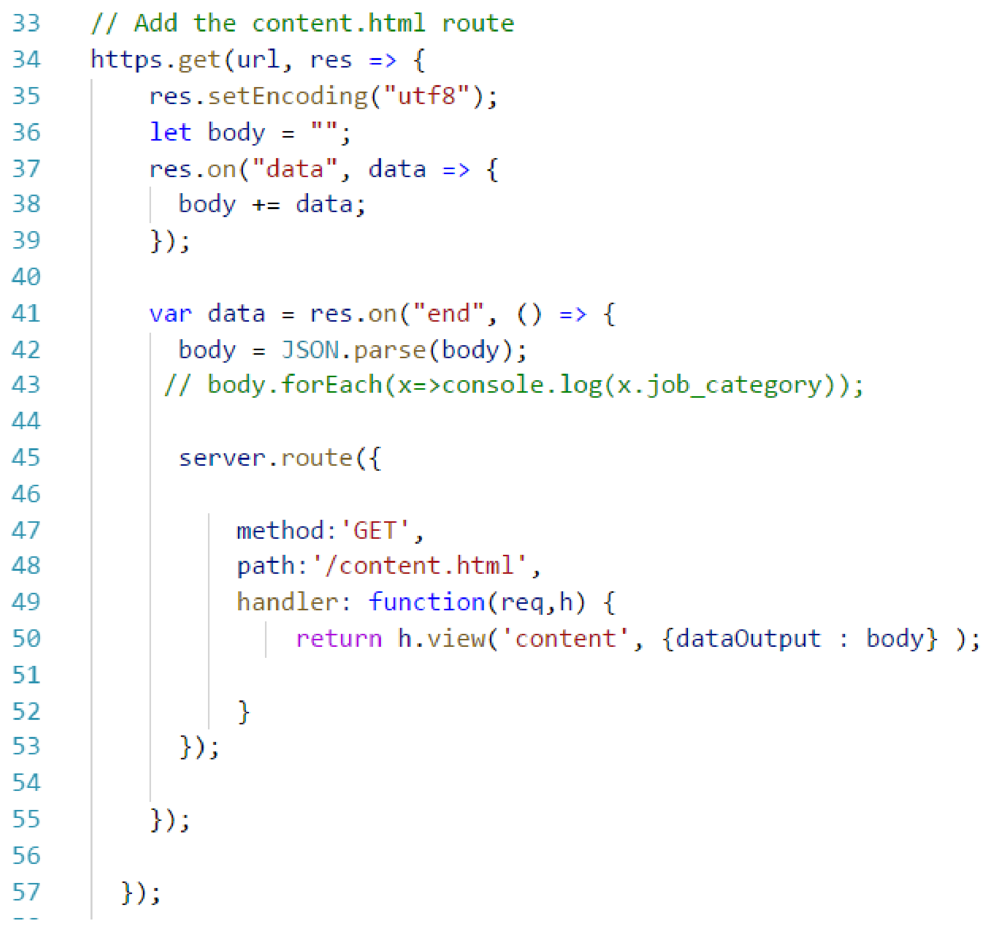
***Open the Postman app and past in the URL above to observe the JSON structure. The postman app will allow you to make request and modify the URL to observe what will be returned.**

14) Add the "https" core module to your application by placing the below statement toward the top of your index.js file.



```
//add
const https = require('https');
```

15) Add the logic to make the https request over the network and nest the content route within the GET request. Alter your code to reflect what is below:

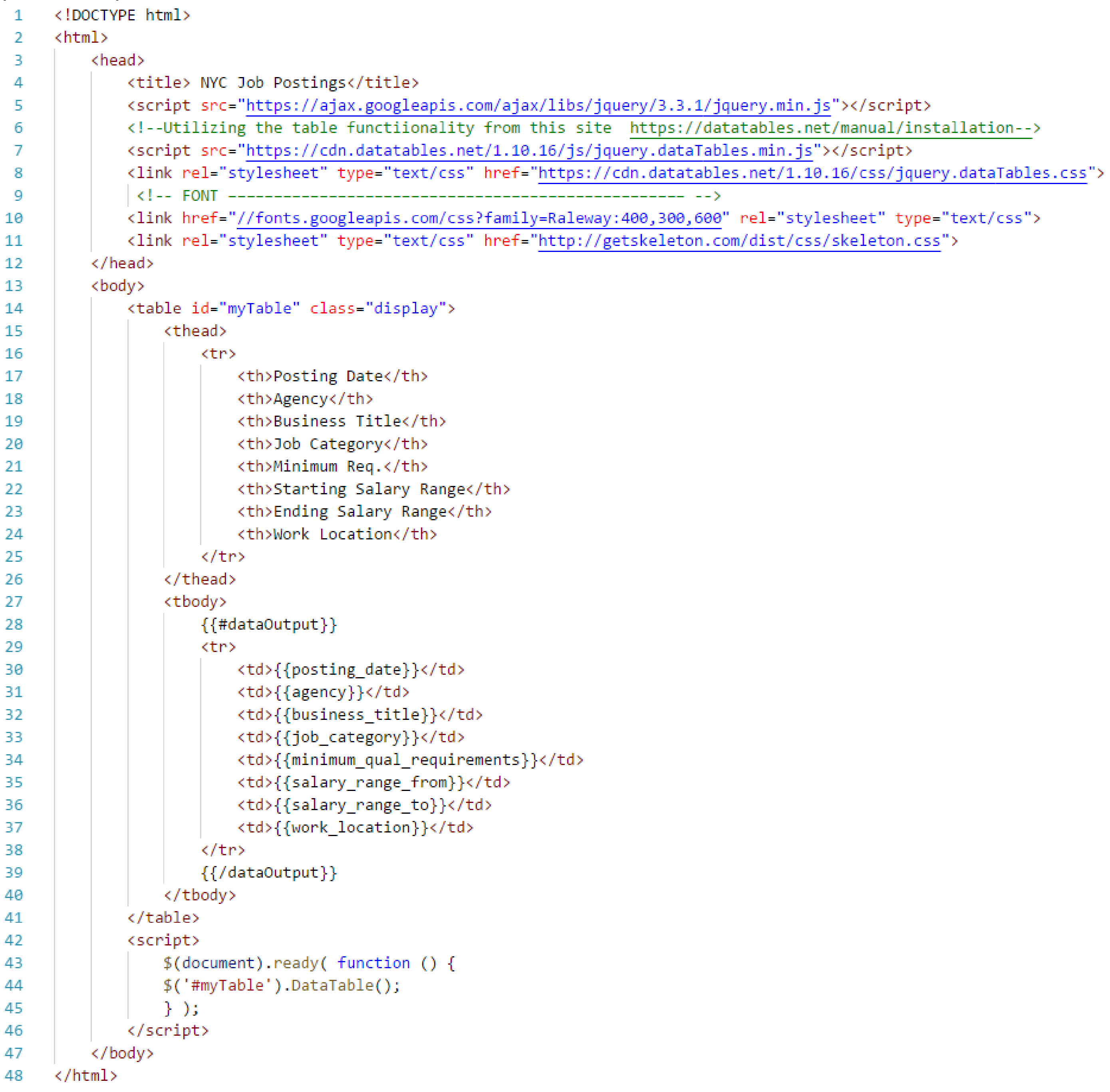


```
// Add the content.html route
https.get(url, res => {
    res.setEncoding("utf8");
    let body = "";
    res.on("data", data => {
      body += data;
    });

    var data = res.on("end", () => {
      body = JSON.parse(body);
     // body.forEach(x=>console.log(x.job_category));

      server.route({

          method:'GET',
          path:'/content.html',
          handler: function(req,h) {
              return h.view('content', {dataOutput : body} );

          }
      });

    });

  });
```

*Line 50 from the code snippet above assumes that the content.html file is located within the root folder of your project folder. If your content.html file is located in a directory, provide the relative file path for the first argument in the view method.

The second parameter of the view method is an object with a dataOutput property and body value. The dataOutput is used on the content.html page to bind the body data to the front-end.

16) Open the content.html page and type the code below. This example is using a resource from https://datatables.net/ to display the data within a table. The example also is using an external style sheet CDN from getSkelton.com to provide presentation.



```html
<!DOCTYPE html>
<html>
    <head>
        <title> NYC Job Postings</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
        <!--Utilizing the table functiionality from this site  https://datatables.net/manual/installation-->
        <script src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
        <link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.css">
        <!-- FONT ––––––––––––––––––––––––––––––––––––––––––––––––––––––– -->
        <link href="//fonts.googleapis.com/css?family=Raleway:400,300,600" rel="stylesheet" type="text/css">
        <link rel="stylesheet" type="text/css" href="http://getskeleton.com/dist/css/skeleton.css">
    </head>
    <body>
        <table id="myTable" class="display">
            <thead>
                <tr>
                    <th>Posting Date</th>
                    <th>Agency</th>
                    <th>Business Title</th>
                    <th>Job Category</th>
                    <th>Minimum Req.</th>
                    <th>Starting Salary Range</th>
                    <th>Ending Salary Range</th>
                    <th>Work Location</th>
                </tr>
            </thead>
            <tbody>
                {{#dataOutput}}
                <tr>
                    <td>{{posting_date}}</td>
                    <td>{{agency}}</td>
                    <td>{{business_title}}</td>
                    <td>{{job_category}}</td>
                    <td>{{minimum_qual_requirements}}</td>
                    <td>{{salary_range_from}}</td>
                    <td>{{salary_range_to}}</td>
                    <td>{{work_location}}</td>
                </tr>
                {{/dataOutput}}
            </tbody>
        </table>
        <script>
            $(document).ready( function () {
            $('#myTable').DataTable();
            } );
        </script>
    </body>
</html>
```

17) Make sure to save the content.html file and start your sever by running the command **nodemon index.js or node index.js**
    a. Navigate to the localhost:8000/content.html and the content from the API should render.
18) Kill the server by pressing CTRL + C
19) Delete the node_modules folder from your directory and commit the changes to your repository.
20) Create a Github repository and push your local repository to the Github repository titled “lab3_[your_first_and_last_name].

When applicable, use programming logic covered in Chapter 4 – 10 to discover information about the data. Create two algorithms that will quantify your data.

Submission of Lab

You will submit this lab via Github. The url to your public repository will be due on the date listed above.