WEB322 Assignment 3

Submission Deadline:

Friday, June 9th, 2017 @ 11:59 PM

Assessment Weight:

5% of your final course Grade

Objective:

Practice routing using Express as well as module design & promises in an application using a mock (JSON) dataset.

Specification:

Extend your web322-app to listen on a number of additional routes (listed below). We will also introduce a custom (promise driven) module that will handle requests for data on our mock datasets.

Getting Started:

As a first step, open your web322-app folder from Assignment 2 in Visual Studio Code. if you have not completed Assignment 2, you can still complete this assignment, however you will have to catch up and get a basic server running on Heroku first (see the Getting Started with Heroku Guide) and use that as your starting point.

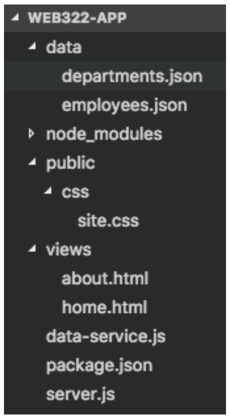
Step 1: Obtaining the Data

- Before we fetch the data, we need to create a new folder at the root of our application called "**data**"
- Once this is complete, create 2 new files inside the "data" folder: **departments.json** and **employees.json**
- Open your web browser and navigate to: this link (departments.json) and copy the contents of the JSON file to your own departments.json file (within the "data" folder).
- Next, navigate to: this link (employees.json) and copy the entire contents of the JSON file to your own employees.json file (within the "data" folder) - this should be an array of 280 "employee" objects

Step 2: Adding a custom data-service.js module

- Create a new file at the root of your application called "**data-service.js**"
- "**require**" this module at the top of your **server.js** file so that we can use it to interact with the data

If you have followed Step 1 and Step 2 correctly, your folder structure should look like the image to the right:



```
WEB322-APP
  data
    departments.json
    employees.json
  node_modules
  public
    css
      site.css
  views
    about.html
    home.html
  data-service.js
  package.json
  server.js
```

Adding additional Routes:

We will be making use of this employee data from a different location from our "/" and "/about" routes. These routes will serve as the public-facing pieces of our application, whereas we will be dealing with employee management in a private area (later protected by a login page / user authentication, etc).

Adding New Routes for server.js

Inside your server.js add routes to respond to the following "get" requests for the application. Once you have written the routes, test that they work properly by returning a confirmation string using **res.send()** and testing the server using localhost:8080. For example, **localhost:8080/employees?department=1** could be set up to return something like **"TODO: get all employees for department 1"**. This will help to confirm that your routes are set up properly and you are correctly capturing the parameter values.

Important Note: Any response sending JSON from the server must include the correct content-type header - see res.json([body])

/employees

- This route will return a JSON formatted string containing all of the employees within the employees.json file
- It also includes the following optional filters (via the query string)
  - /employees?status=***value***
    where ***value*** could be either "Full Time" or "Part Time"
  - /employees?department=***value***
    where ***value*** could be one of 1, 2, 3, … 7 (there are currently 7 departments in the dataset)
  - /employees?manager=***value***
    where ***value*** could be one of 1, 2, 3, … 30 (there are currently 30 managers in the dataset)

/employee/**value**

- This route will return a JSON formatted string containing the employee whose **employeeNum** matches the ***value***. For example, once the assignment is complete, **localhost:8080/employee/6** would return the manager: **Cassy Tremain**.

/managers

- This route will return a JSON formatted string containing all the employees whose **isManager** property is set to **true**.

/departments

- This route will return a JSON formatted string containing all of the departments within the departments.json file

[no matching route]

- If the user enters a route that is not matched with anything in your app (ie: http://localhost:8080/app) then you must return the custom message "**Page Not Found**" with an HTTP status code of **404**

Creating the data-service.js module:

The promise driven data-service.js module will be responsible for reading the employees.json and departments.json files from within the "views" directory on the server, parsing the data into arrays of objects and returning elements ("employee" objects) from those arrays to match queries on the data.

Essentially the data-service.js module will encapsulate all the logic to work with the data and only expose accessor methods to fetch subsets of the data.

Module Data

The following two arrays should be declared "globally" within your module:

- **employees** - type: **array**
- **departments** - type: **array**

Exported Functions

Each of the below functions are designed to work with the employees and departments datasets. Since we have no way of knowing how long each function will take (we cannot assume that they will be instantaneous, ie: what if we move from .json files to a remote database, or introduce hundreds of thousands of objects into our .json dataset? - this would increase lag time).

Because of this, **every one of the below functions must return a promise** that **passes the data** via it's "**resolve**" method (or - if **no data was returned**, passes an **error message** via it's "**reject**" method). When we access these methods from the server.js file, we will be assuming that they return a promise and will respond appropriately with **.then()** and **.catch()** (see "Updating the new routes…" below).

initialize()

- This function will read the contents of the "./data/employees.json" file (**hint**: see the fs module & the fs.readFile method), convert the file's contents into an array of objects (**hint**: see JSON.parse) , and assign that array to the **employees array** (from above).

- Only once the read operation for "./data/employees.json" has completed successfully (not before), repeat the process for the "./data/departments.json" and assign the parsed object array to the **departments array** from above.

- Once these two operations have finished successfully, invoke the **resolve** method for the promise to communicate back to server.js that the operation was a success.

- If there was an error at any time during this process, invoke the **reject** method for the promise and pass an appropriate message, ie: reject("unable to read file").

getAllEmployees()

- This function will provide the full array of "employee" objects 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".

getEmployeesByStatus(***status***)

- This function will provide an array of "employee" objects whose **status** property matches the ***status*** parameter (ie: if ***status*** is "Full Time" then the array will consist of only "Full Time" employees) 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".

getEmployeesByDepartment(***department***)

- This function will provide an array of "employee" objects whose **department** property matches the ***department*** parameter (ie: if ***department*** is 5 then the array will consist of only employees who belong to department 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".

getEmployeesByManager(***manager***)

- This function will provide an array of "employee" objects whose **employeeManagerNum** property matches the ***department*** parameter (ie: if ***manager*** is 14 then the array will consist of only employees who are managed by employee 14) 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".

getEmployeeByNum(***num***)

- This function will provide a single of "employee" object whose **employeeNum** property matches the ***num*** parameter (ie: if ***num*** is 261 then the "employee" object returned will be "Glenine Focke") 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".

getManagers()

- This function will provide an array of "employee" objects whose **isManager** property is **true** 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".

getDepartments()

- This function will provide the full array of "department" objects 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".

Updating the code surrounding app.listen()

Before we start updating the routes in server.js we must make a small update to the code ***surrounding*** the app.listen() call at the bottom of the file. This is where the **initialize()** method from our data-service.js 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) arrays, "employees" and "departments" that are used by the other functions. However, it also returns a **promise** that will only **resolve** successfully once the files were read correctly and the "employees" and "departments" 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(()=>{ /*output the error to the console */})**

Updating the new routes to use data-service.js

Now that the data-service.js module is complete, we must update our new routes (see above) to make calls to the service and fetch data to be returned to the client. Recall: Any response sending JSON from the server must include the correct content-type header - see res.json([body]).

Since our data-service.js file exposes functions that are guaranteed to return a **promise** that (if resolved successfully), will contain the requested data, we must make use of the **.then()** method when accessing the data from within our routes.

For example, the **/departments** route must make a call to the **getDepartments()** method of the data-service.js module to fetch the correct data. If **getDepartments()** was successful, we can use **.then((data) => { /*return JSON data*/ })** to access the data from the function and send the response back to the client.

If any of the methods were unsuccessful however, ie: **getEmployeeByNum(-4)** the **.then()** method will not be called - the **catch()** method will be called instead. If this is the case, the server **must** return a simple JSON object with 1 property: "message" containing the message supplied in the **.catch()** method, ie: **.catch((err) => { /* return err message in the JSON format:** {message: ***err***}***/ })**.

By **only** calling **res.json()** from within **.then()** or **.catch()** we can ensure that the data will be in place (no matter how long it took to retrieve) before the server sends anything back to the client.

Sample Solution

To see a completed version of this app running, visit: https://boiling-brushlands-45900.herokuapp.com

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 web sites) or
*  distributed to other students.
*
*  Name: ______________________ Student ID: ______________ Date: ________________
*
*  Online (Heroku) Link: ________________________________________________________
*
********************************************************************************/
```

- Publish your application on Heroku & test to ensure correctness
- Compress your web322-app folder and Submit your 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.