

# WEB322 Assignment 3

## Submission Deadline:

Friday, October 12<sup>th</sup>, 2018 @ 11:59 PM

## Assessment Weight:

9% of your final course Grade

## Objective:

Build upon the foundation established in Assignment 2 by providing new routes / views to support adding new employees and uploading images.

**NOTE:** If you are unable to start this assignment because Assignment 2 was incomplete - email your professor for a clean version of the Assignment 2 files to start from (effectively removing any custom CSS or text added to your solution).

## RESOURCES:

Web322.ca: Weeks 1,2,3,4,5 learning content as well as Blackboard; nodejs.org, expressjs.com, the web.

## Specification:

For this assignment, we will be enhancing the functionality of Assignment 2 to include new routes & logic to handle file uploads and add employees. We will also add new routes & functionality to execute more focused queries for data (i.e. fetch an employee by id, all employees by a department or manager number, etc.)

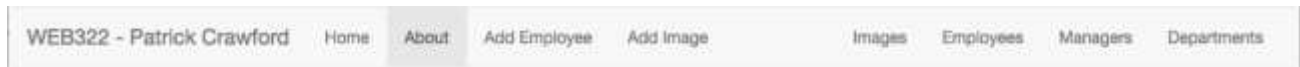
## Part 1: Adding / Updating Static (.html) Files & Directories

### Step 1: Modifying home.html & about.html

- Open the home.html file from within the "views" folder
- Add the following two entries to the `<ul class="nav navbar-nav">` element:
  - `<li><a href="/employees/add">Add Employee</a></li>`
  - `<li><a href="/images/add">Add Image</a></li>`
- Add the following entry as the **first child** element of the `<ul class="nav navbar-nav navbar-right">` element
  - `<li><a href="/images">Images</a></li>`
- Your "Home" page should now have a menu bar that looks like the following:



- Update your "About" page with the same changes. When complete, it should look like the following:



## Step 2: Adding new routes in server.js to support the new views

- Inside your server.js file add the following routes (HINT: do not forget `__dirname` & `path.join`):
  - GET `/employees/add`
    - This route simply sends the file `"/views/addEmployee.html"`
  - GET `/images/add`
    - This route simply sends the file `"/views/addImage.html"`

## Step 3: Adding new file 1: addEmployee.html

- Create a new file in your "views" directory called "addEmployee.html" and open it for editing
- Copy the contents of "home.html" and paste it in as a starting point.
- Ensure that the "Add Employee" item in the `<ul class="nav navbar-nav"> ...</ul>` element is the **only** `<li>` with the class "active" (this will make sure the correct navigation element is "highlighted")
- Remove all html code **inside** the `<div class="row"> ... </div>`
- Inside the (now empty) `<div class="row"> ... </div>` element, use the html from the sample solution ( <https://calm-atoll-83756.herokuapp.com/employees/add> ) to reconstruct the "Add Employee" form (HINT: You can right-click the page to "view source" - the html you want is within the `<div class="row"> ...</div>` element)

## Step 4: Adding new file 2: addImage.html

- Create a new file in your "views" directory called "addImage.html" and open it for editing
- Copy the contents of "home.html" and paste it in as a starting point.
- Ensure that the "Add Image" item in the `<ul class="nav navbar-nav"> ...</ul>` element is the **only** `<li>` with the class "active" (this will make sure the correct navigation element is "highlighted")
- Remove all html code **inside** the `<div class="row"> ... </div>`
- Inside the (now empty) `<div class="row"> ... </div>` element, use the html from the sample solution ( <https://calm-atoll-83756.herokuapp.com/images/add> ) to reconstruct the "Add Image" form (HINT: You can right-click the page to "view source" - the html you want is within the `<div class="row"> ...</div>` element)

## Step 5: Adding a home for the uploaded Images

- Create a new folder in your "public" folder called "images"
- Within the newly created "images" folder, create an "uploaded" folder

# Part 2: Adding Routes / Middleware to Support Image Uploads

## Step 1: Adding multer

- Use npm to install the "multer" module

- Inside your server.js file "require" the "multer" module as "multer"
- Define a "storage" variable using "multer.diskStorage" with the following options (HINT: see "Step 5: (server) Setup..." in the [week 5 course notes](#) for additional information)
  - **destination** `"/public/images/uploaded"`
  - **filename** `function (req, file, cb) {  
    cb(null, Date.now() + path.extname(file.originalname));  
}`
- Define an "upload" variable as **`multer({ storage: storage });`**

## Step 2: Adding the "Post" route

- Add the following route:
  - `POST /images/add`
    - This route uses the middleware: **`upload.single("imageFile")`**
    - When accessed, this route will redirect to `"/images"` (defined below)

## Step 3: Adding "Get" route / using the "fs" module

- Before we can add the below route, we must include the **"fs" module** in our **server.js** file (previously only in our data-service.js module)
- Next, Add the following route:
  - `GET /images`
    - This route will return a JSON formatted string (`res.json()`) consisting of a single "images" property, which contains the contents of the `"/public/images/uploaded"` directory as an array, ie `{ "images": ["1518109363742.jpg", "1518109363743.jpg"] }`. **HINT:** You can make use of the **`fs.readdir`** method, as outlined in [this example from code-maven.com](#)

## Step 4: Verify your Solution

At this point, you should now be able to upload images using the `"/images/add"` route and see the full file listing on the `"/images"` route in the format: `{ "images": ["1518109363742.jpg", "1518109363743.jpg"] }`.

# Part 3: Adding Routes / Middleware to Support Adding Employees

## Step 1: Adding body-parser

- Use npm to install the "body-parser" module
- Inside your server.js file "require" the "body-parser" module as "bodyParser"
- Add the `bodyParser.urlencoded({ extended: true })` middleware (using `app.use()`)

## Step 2: Adding "Post" route

- Add the following route:

- POST /employees/add
  - This route makes a call to the (promise-driven) addEmployee(employeeData) function from your data-service.js module (function to be defined below). It will provide **req.body** as the parameter, ie "data.addEmployee(req.body)".
  - When the addEmployee function resolves successfully, redirect to the "/employees" route. Here we can verify that the new employee was added

### Step 3: Adding "addEmployee" function within data-service.js

- Create the function "addEmployee(employeeData)" within data-service.js according to the following specification: (**HINT:** do not forget to add it to module.exports)
  - Like all functions within data-service.js, this function must return a Promise
  - If **employeeData.isManager** is undefined, explicitly set it to **false**, otherwise set it to **true** (this gets around the issue of the checkbox not sending "false" if it's unchecked)
  - Explicitly set the **employeeNum** property of **employeeData** to be the **length of the "employees" array plus one (1)**. This will have the effect of setting the first new employee number to 281, and so on.
  - **Push** the updated **employeeData** object onto the **"employees"** array and **resolve** the promise.

### Step 4: Verify your Solution

At this point, you should now be able to add new employees using the "/employees/add" route and see the full employee listing on the "/employees" route.

## Part 4: Adding New Routes to query "Employees"

### Step 1: Update the "/employees" route

- In addition to providing all of the employees, this route must now also support the following optional filters (via the query string)
  - /employees?status=**value**
    - return a JSON string consisting of all employees where **value** could be either "Full Time" or "Part Time" - this can be accomplished by calling the **getEmployeesByStatus(status)** function of your data-service (defined below)
  - /employees?department=**value**
    - return a JSON string consisting of all employees where **value** could be one of 1, 2, 3, ... 7 (there are currently 7 departments in the dataset) " - this can be accomplished by calling the **getEmployeesByDepartment(department)** function of your data-service (defined below)
  - /employees?manager=**value**
    - return a JSON string consisting of all employees where **value** could be one of 1, 2, 3, ... 30 (there are currently 30 managers in the dataset) " - this can be accomplished by calling the **getEmployeesByManager(manager)** function of your data-service (defined below)

- /employees
  - return a JSON string consisting of all employees without any filter (existing functionality)

### Step 2: Add the "/employee/value" route

- 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** - - this can be accomplished by calling the **getEmployeeByNum(num)** function of your data-service (defined below).

## Part 5: Updating "data-service.js" to support the new "Employee" routes

**Note:** All of the below functions must return a **promise** (continuing with the pattern from the rest of the data-service.js module)

### Step 1: Add the getEmployeesByStatus(status) Function

- 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".

### Step 2: Add the getEmployeesByDepartment(department) Function

- 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".

### Step 3: Add the getEmployeesByManager(manager) Function

- 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".

### Step 3: Add the getEmployeeByNum(num) Function

- 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".

## Part 6: Pushing to Heroku

Once you are satisfied with your application, deploy it to Heroku:

- Ensure that you have checked in your latest code using **git** (from within Visual Studio Code)
- Open the integrated terminal in Visual Studio Code
- Log in to your Heroku account using the command **heroku login**
- Create a new app on Heroku using the command **heroku create**
- Push your code to Heroku using the command **git push heroku master**
- **IMPORTANT NOTE:** Since we are using an "**unverified**" free account on Heroku, we are limited to only **5 apps**, so if you have been experimenting on Heroku and have created 5 apps already, you must delete one (or verify your account with a credit card). Once you have received a grade for Assignment 1, it is safe to delete this app (login to the Heroku website, click on your app and then click the **Delete app...** button under "**Settings**").

### Testing: Sample Solution

To see a completed version of this app running, visit: <https://calm-atoll-83756.herokuapp.com/>

**Please note:** This solution is **visible** to **ALL students** and **professors** at Seneca College. It is your responsibility as a student of the college not to post inappropriate content / images to the shared solution. It is meant purely as an exemplar and any misuse will not be tolerated.

### Assignment Submission:

- Before you submit, consider updating **site.css** to provide additional style to the pages in your app. Black, White and Gray is boring, so why not add some cool colors and fonts (maybe something from [Google Fonts](#))? This is your app for the semester, you should personalize it!
- Next, 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: _____  
*  
* Online (Heroku) Link: _____  
*  
*****/
```

- 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 (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, i.e.: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.
- Paste Heroku URL in the comments section of the submissions page (if missing there is a **-2.0 penalty**). Also, be sure to place the Heroku URL in the declarations section on top of server.js.