WEB322 Assignment 4

Submission Deadline:

Friday, June 23rd, 2017 @ 11:59 PM

Assessment Weight:

5% of your final course Grade

Objective:

Practice working with Handlebars.js as a templating engine & handling POST routes / processing POST data from our server.

Specification:

Extend your web322-app to listen on a number of additional routes, update our existing routes to return rendered HTML pages using the "express-handlebars" module, create web forms using HTML & the Bootstrap Forms classes, and lastly, extend our data-service.js module to accommodate requests to add or update employee data.

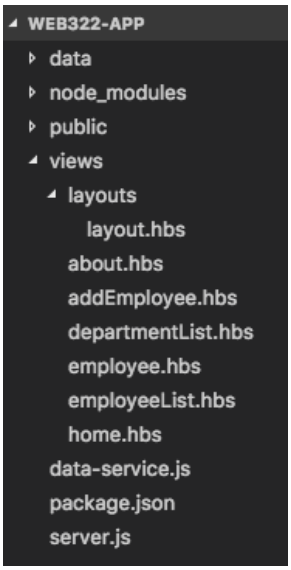
NOTE: If you are unable to start this assignment because Assignment 3 was incomplete - email your professor for a clean version of the Assignment 3 files to start from (effectively removing any custom CSS or text added to your solution). Remember, you must successfully complete ALL assignments to pass this course.

Getting Started:

Before we start updating our code, we first must add / rename some files in our web322-app folder:

- Under the **views** folder, add a folder called **layouts**
- Inside the **layouts** folder, add a new **layout.hbs** file
- Inside the **views** folder, add the new files: **addEmployee.hbs**, **departmentList.hbs**, **employee.hbs** and **employeeList.hbs**
- Rename the **about.html** and **home.html** files to use the new **.hbs** extension

If you followed the instructions correctly, your folder should look like the image to the right:



Next, we must add 2 additional modules: body-parser and express-handlebars:

- Open the Integrated Terminal
- Run the commands to install **body-parser** and **express-handlebars** using the **--save** flag

Once this is complete, we must ensure that these modules are correctly loaded in our **server.js** file:

- Open **server.js**
- Beside the other "require" statements, add the lines:
  const exphbs = require('express-handlebars');
  const bodyParser = require('body-parser');
- Under your **app.use(express.static('public'));** line, add the code:

```
app.use(bodyParser.urlencoded({ extended: true }));

app.engine(".hbs", exphbs({
  extname: ".hbs",
  defaultLayout: 'layout',
  helpers: {
    equal: function (lvalue, rvalue, options) {
      if (arguments.length < 3)
        throw new Error("Handlebars Helper equal needs 2 parameters");
      if (lvalue != rvalue) {
        return options.inverse(this);
      } else {
        return options.fn(this);
      }
    }
  }
}));
app.set("view engine", ".hbs");
```

- This will ensure that the bodyParser middleware will work correctly. Also, this will allow the .hbs extensions to be properly handled, add the custom Handlebars helper: "equal" and set the global default layout to our layout.hbs file

Creating a common Layout :

Now that we have all of the correct modules loaded and working properly, we need to update our existing "home" and "about" routes to leverage an important feature in handlebars: Layouts.

Since our application is generally always going to have the same header, navigation and footer, it would make sense for us to place these chunks of HTML in a common layout, used by every page:

- Open the **home.hbs** file
- Copy the entire contents of the file into the **layout/layout.hbs** file
- Inside the **layout.hbs** file, remove the chunk of code of HTML that is going to change from page to page, ie: the … element **inside** the (but not the <div class="container" tags), beneath the <nav> element.
- Place the text {{{body}}} inside the container, ie: {{{body}}} This will be where we will inject all of our other "views"!
- Change the <title></title> attribute to read "WEB322 App"
- **Remove** the CSS reference element:
  <link rel="stylesheet" href="https://cdn.rawgit.com/WEB222-Samples/Utilities/master/bootstrap-md-grid/bootstrap-md-grid.css" type="text/css" />
- **Replace** it with:
  <link rel="stylesheet"
  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
  type="text/css" />
- The above CSS file links to the full bootstrap 3.3.7 css file - this will give us access to all of the CSS that bootstrap 3 has to offer (not just the "md" grid classes)
- Update the link to your own custom "site.css" file to make sure that there is a leading "/" - ie, change **href="css/site.css"** to **href="/css/site.css"**
- Finally, update the <nav> element so that we have access to the new views. Here, we will be making use of the bootstrap classes "pull-left" and "pull-right":

```
<ul class="pull-left">
    <li> <a href="/"> Home </a> </li>
    <li> <a href="/about"> About </a> </li>
</ul>

<ul class="pull-right">
    <li> <a href="/employees"> All Employees </a> </li>
    <li> <a href="/managers"> Managers </a> </li>
    <li> <a href="/departments"> Departments </a> </li>
</ul>
```

Updating home.hbs & about.hbs to use the new layout

Now that we have a custom layout that we can use for every page, let's update our existing home.hbs & about.hbs files to use this new layout:

- Open **home.hbs**
- Remove all the code that is now inside **layout.hbs**, ie: all the code **above and below** the main ... element. This is the element that contains the <aside> and <section> elements that contain all of our information specific to home.
- This should leave you with a single … element with your "home" content inside
- Similarly, open **about.hbs** and repeat the process (remove everything above and below the main <div class="row" element)
- Now that our views are updated, we must make a small change to how we actually "serve" the views from within our server.js file:
- Instead of using code to send the .html file, ie: res.sendFile(path.join(__dirname + "/views/home.html")); simply use the line: **res.render("home");**
- Similarly, for the /about route, use **res.render("about");** instead of res.sendFile(path.join(__dirname + "/views/about.html"));
- Test your server - you should be able to see your home page on the "/" route, and your about page on the "/about" route, as well as all of the JSON data routes for Employees, Managers & Departments. Additionally, you should see the style change *slightly*, since the full bootstrap.css also includes normalize functionality.

Creating new .hbs views to render the Employee JSON data

Now that we have the full handlebars view engine functionality at our disposal, we can update all of our views that return JSON, to actually render the data:

- In your server.js file, update the GET route: **/employees** to use **res.render("employeeList", { data: data, title: "Employees" });** instead of **res.json(data);** inside the **.then()** callback. Also, use **res.render("employeeList", { data: {}, title: "Employees" });** inside the **.catch()** callback (if there is an error, send an empty object for "data")
- Update the GET route: **/managers** to use **res.render("employeeList", { data: data, title: "Employees (Managers)" });** inside the **.then()** callback and use **res.render("employeeList", { data: {}, title: "Employees (Managers)" });** inside the **.catch()** callback
- Update the GET route: **/departments** to use **res.render("departmentList", { data: data, title: "Departments" });** inside the **.then()** callback and use **res.render("departmentList", { data: {}, title: "Departments" });** inside the **.catch()** callback

- Inside the file: **employeeList.hbs**, add the code:

```
<div class="row">
  <div class="col-md-12">
    <h1>{{title}}<a href="/employees/add" class="btn btn-success pull-right" style="margin-top:5px;">Add New Employee</a><h1>
    <hr />
  </div>
</div>
<div class="row">
  <div class="col-md-12">
    {{#each data}}
      {{firstName}} {{last_name}}<br />
    {{/each}}
  </div>
</div>
```

- Run your server and access the **/employees** route - you should see the following page:

- Update this page to render all of the data in a table, using the bootstrap classes: "table-responsive" (for the containing the table) and "table" (for the table itself) - Refer to the completed sample here.
- The table must consist of 8 columns with the headings: **Employee Num**, **Full Name**, **Email**, **Address**, **Manager ID**, **Status**, **Department** and **Hired On**
- Additionally, the Name in the Full Name column must link to /employees/**empNum** where **empNum** is the employee number for that row
- The "Email" column must be a "mailto" link to the user's email address for that row
- The "Manager Id" link must link to /employees?manager=**employeeManagerNum** where **employeeManagerNum** is the manager number for the employee for that row
- The "Status" link must link to /employees?status="Full Time" if "Full Time" is clicked, and /employees?status="Part Time" if "Part Time is clicked
- The "Department" link must link to /employees?department=**department** where **department** is the department number for the employee for that row

Creating new .hbs view to render the Departments JSON data

Now that we have the "Employee" data rendering correctly in the browser, we can use the same pattern to render the "Departments" data in a table:

- Inside the file**: departmentList.hbs**, add the code:

```
<div class="row">
  <div class="col-md-12">
    <h1>{{title}}<h1>
    <hr />
  </div>
</div>
<div class="row">
  <div class="col-md-12">
    {{#each data}}
      {{departmentId}} {{departmentName}}<br />
    {{/each}}
  </div>
</div>
```

- Run your server and access the **/departments** route - you should see the following page:

- Update this page to render all of the data in a table, using the bootstrap classes: "table-responsive" (for the containing the table) and "table" (for the table itself).
- The table must consist of 2 columns with the headings: **Department Number** and **Department Name**
- Refer to the example online at: https://polar-forest-65855.herokuapp.com/departments for the final result

Creating new .hbs file / route to Add Employees:

Currently, if you click on the "Add New Employee" button from **/employees**, you should see a "Page Not Found" error. This is because we have not configured the route from our server.js file yet:

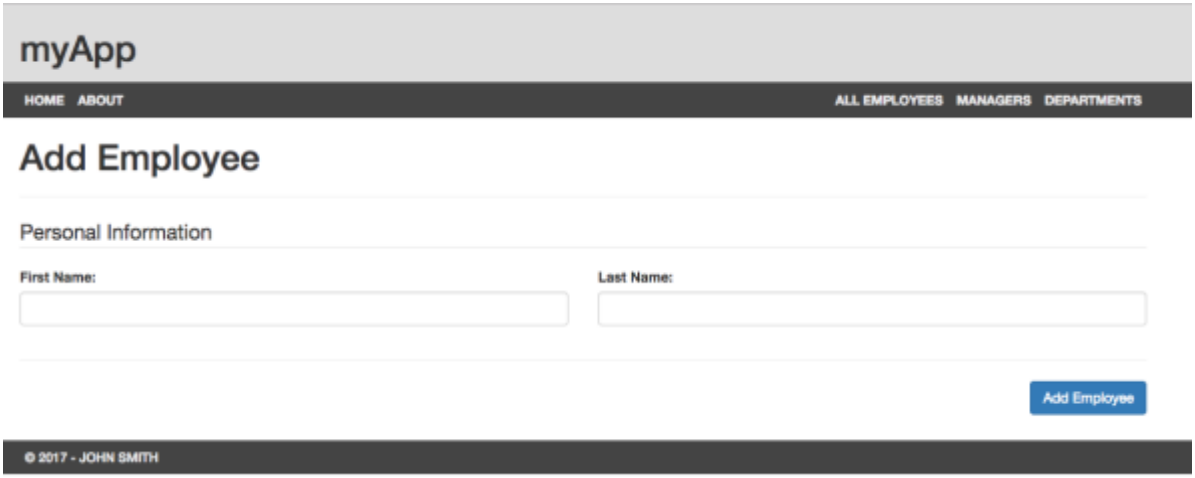
- Open the file **server.js**
- Add the "GET" route for "**/employees/add**" which renders the view "addEmployee":

```
app.get("/employees/add", (req,res) => {
  res.render("addEmployee");
});
```

- Inside the "**addEmployee.hbs**" file, add the code:

```
<div class="row">
  <div class="col-md-12">
    <h1>Add Employee</h2>
    <hr />
    <form method="post" action="/employees/add">
      <fieldset>
        <legend>Personal Information</legend>
        <div class="row">
          <div class="col-md-6">
            <div class="form-group">
              <label for="firstName">First Name:</label>
              <input class="form-control" id="firstName" name="firstName" type="text" />
            </div>
          </div>
          <div class="col-md-6">
            <div class="form-group">
              <label for="last_name">Last Name:</label>
              <input class="form-control" id="last_name" name="last_name" type="text"  />
            </div>
          </div>
        </div>
      </fieldset>
      <hr />
      <input type="submit" class="btn btn-primary pull-right" value="Add Employee" /><br /><br /><br />
    </form>
  </div>
</div>
```

- Test the server **(/employees/add**) - this will get you started on creating the form using the bootstrap 3 classes:



- Continue this pattern to develop the full form to match the [completed sample here] - you may use the code in the sample to help guide your solution

    - **Email**: type: "email", name: "email"
    - **Social Security Number:** type: "text", name: "SSN"
    - **Address (Street):** type: "text", name: "addressStreet"
    - **Address (City):** type: "text", name: "addresCity"
    - **Address (State):** type: "text", name: "addressState"
    - **Address (Zip Code):** type: "text", name: "addressPostal"
    - **Manager:** type: "checkbox", name: "isManager"
    - **Employee's Manager Number:** type: "text", name: "employeeManagerNum"
    - **Status:** type: "radio" name: "status", values: "Full Time" or "Part Time"
    - **Department** type: "select", name: "department", values: 1 - 7 inclusive
    - **Hire Date** type: "text", name: "hireDate"

- No validation (client or server-side) is required on any of the form elements at this time

- Once the form is complete, we must add the **POST** route: **/employees/add** in our server.js file:

```
app.post("/employees/add", (req, res) => {
  console.log(req.body);
  res.redirect("/employees");
});
```

- This will show you all of the data from your form in the console, once the user clicks "add employee. However, in order to take that data and add it to our "employees" array in memory, we must add some new functionality to the **data-service.js** module:

- Before we add any methods, we must add a new property called **empCount** underneath **employees** and **departments** and initialize it to **0**. This is simply a counter that increases whenever a new employee is added - this will help us to assign employee ID's to new employees

- Inside the **initialize** method, set **empCount** to **employees.length** before resolving the promise.

- Next, add the new method: **addEmployee(employeeData)** that also **returns a promise**. This method will:
    - Increment **empCount**
    - Take the JavaScript object (parameter employeeData) and assign it's **employeeNum** property to the current value of **empCount** (this should be 281 the first time addEmployee is called)
    - Add the new employeeData object to the **employees** array
    - Once this has completed successfully, invoke the **resolve()** method without any data.
- Now that we have a new **addEmployee()** method, we can invoke this function from our newly created **app.post("/employees/add", (req, res) => { … });** route. Simply invoke the **addEmployee**() method with the **req.body** as the parameter. Once the promise is resolved use the **then()** callback to execute the **res.redirect("/employees");** code.
- Test your server in the browser by adding a simple "John Smith" user. Once you have clicked "Add Employee" and are redirected back to the employee list, "John Smith" should appear at the end of the list, with Employee Number 281!

Creating new .hbs file / route to Update Employees:

Lastly, if you click on a user's name in the employee list (or by accessing the employee directly, ie: **http://localhost:8080/employee/21**), you should see a JSON formatted string representing the corresponding employee (ie: employee 21).

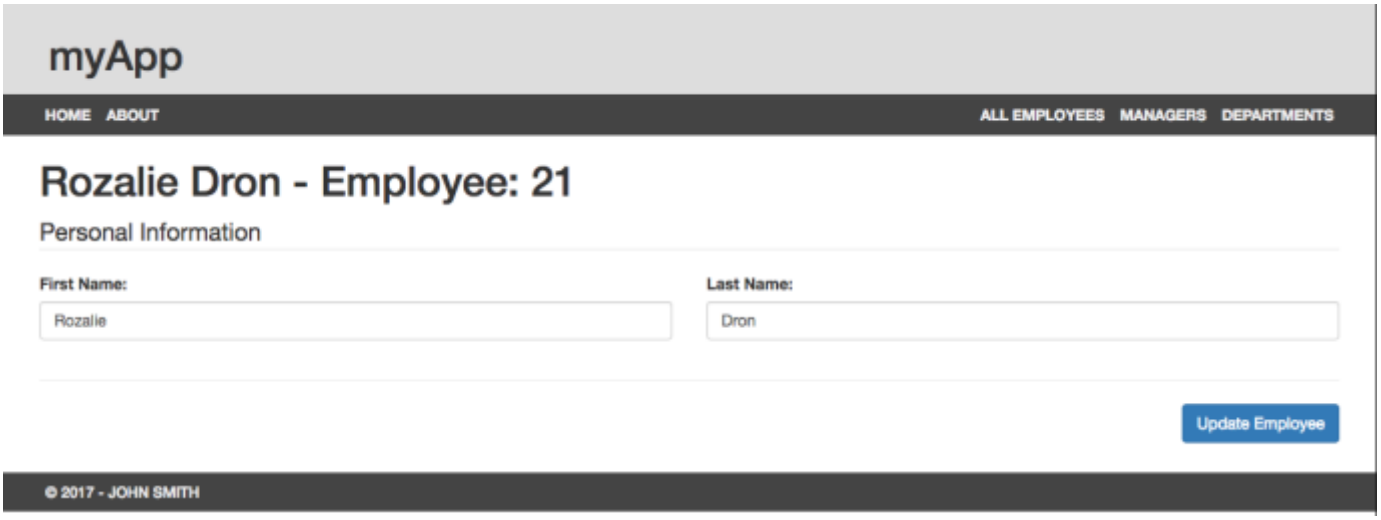
Now that we know how we can use this data to render a meaningful page (view), we must update the last file: **employee.hbs** and "render" the view from within **app.get("/employee/:empNum", (req, res) => { … });**

- Open the file **server.js**
- Update the "GET" route for **"/employee/:empNum** " to use **res.render("employee", { data: data });** inside the **.then()** callback and use **res.status(404).send("Employee Not Found");** inside the **.catch()** callback
- Inside the "**employee.hbs**" file, add the code:

```
<div class="row">
  <div class="col-md-12">

    <h1>{{data.firstName}} {{data.last_name}} - Employee: {{data.employeeNum}}</h2>

    <form method="post" action="/employee/update">

      <fieldset>
        <legend>Personal Information</legend>
        <div class="row">
          <div class="col-md-6">
            <div class="form-group">
              <label for="firstName">First Name:</label>
```

```
                <input class="form-control" id="firstName" name="firstName" type="text"
value="{{data.firstName}}" />
              </div>
            </div>
            <div class="col-md-6">
              <div class="form-group">
                <label for="last_name">Last Name:</label>
                <input class="form-control" id="last_name" name="last_name" type="text"
value="{{data.last_name}}" />
              </div>
            </div>
          </div>
        </fieldset>
        <hr />
        <input type="submit" class="btn btn-primary pull-right" value="Update Employee" /><br /><br /><br
/>
      </form>
    </div>
</div>
```

- Test the server **(/employee/21**) - this will get you started on creating / populating the form with user data:



- Continue this pattern to develop the full form to match the [completed sample here](#) - you may use the code in the sample to help guide your solution
    - **employeeNum**: type: "hidden", name: "employeeNum"
    - **Email**: type: "email", name: "email"
    - **Social Security Number:** type: "text", name: "SSN", readonly
    - **Address (Street):** type: "text", name: "addressStreet"
    - **Address (City):** type: "text", name: "addresCity"
    - **Address (State):** type: "text", name: "addressState"
    - **Address (Zip Code):** type: "text", name: "addressPostal"
    - **Manager:** type: "checkbox", name: "isManager", **(HINT**: **use the #if helper - {{#if data.isManager}} … {{/if}} to see if the checkbox should be checked or not)**

    - **Employee's Manager Number:** type: "text", name: "employeeManagerNum"
    - **Status:** type: "radio" name: "status", values: "Full Time" or "Part Time" **(HINT, use the #equals helper - {{#equal data.status "Full Time" }} … {{/equal}} to see if Full Time or Part Time is checked)**
    - **Department** type: "select", name: "department", values: 1 - 7 inclusive **(HINT, use the #equals helper - {{#equal data.department "1" }} … {{/equal}} to determine which option should be selected)**
    - **Hire Date** type: "text", name: "hireDate", readonly

- No validation (client or server-side) is required on any of the form elements at this time
- Once the form is complete, we must add the **POST** route: **/employee/update** in our server.js file:

```
app.post("/employee/update", (req, res) => {
  console.log(req.body);
  res.redirect("/employees");
});
```

- This will show you all of the data from your form in the console, once the user clicks "Update Employee". However, in order to take that data and update our "employees" array in memory, we must add some new functionality to the **data-service.js** module:
- Add the new method: **updateEmployee(employeeData)** that **returns a promise**. This method will:
    - Search through the "employees" array for an employee with an employeeNum that matches take the JavaScript object (parameter employeeData).
    - When the matching employee is found, overwrite it with the new employee passed in to the function (parameter employeeData)
    - Once this has completed successfully, invoke the **resolve()** method without any data.
- Now that we have a new **updateEmployee()** method, we can invoke this function from our newly created **app.post("/employee/update", (req, res) => { … });** route. Simply invoke the **updateEmployee**() method with the **req.body** as the parameter. Once the promise is resolved use the **then()** callback to execute the **res.redirect("/employees");** code.
- Test your server in the browser by updating Employee 21 (Rozalie Dron). Once you have clicked "Update Employee" and are redirected back to the employee list, Employee 21 should show your changes!

Sample Solution

To see a completed version of this app running, visit: https://polar-forest-65855.herokuapp.com/

Assignment Submission:

- Add the following declaration at the top of your server.js file:

  /***
  * WEB322 – Assignment 04
  * 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** 4

Important Note:

- All HTML rendered in the browser **must not** contain any **errors** (see: W3C Validation)
- **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.