WEB700 Assignment 5

Assessment Weight:

9% of your final course Grade

Objective:

Build upon the code created in Assignment 3 by incorporating the Handlebars view engine to render our JSON data visually in the browser using .hbs views and layouts. Additionally, update our officeData, server.js module to allow for employees to be viewed and added individually using web form. The assignment requires you to do research about bootstrap framework classes as well – however you may use your own css styling if you wish.

Specification:

Part 1: Getting Express Handlebars & Updating your views

Step 1: Install & configure express-handlebars

- Use npm to install the "express-handlebars" module
- Wire up your server.js file to use the new "express-handlebars" module, ie:
    - "require" it as exphbs
    - add the app.engine() code using exphbs.engine({ … }) and the "extname" property as ".hbs" and the "defaultLayout" property as "main" (See the Week 9 Notes)
    - call app.set() to specify the 'view engine' (See the Week 9 Notes)
- Inside the "views" folder, create a "layouts" folder - done
- In the "layouts" directory, create a "main.hbs" file (this is our "default layout") -done

Part 2: Updating the Employees Route & Adding a View

Rather than simply outputting a list of employees using res.json, it would be much better to actually render the data in a table that allows us to access individual employees and filter the list using our existing req.params code.

Step 1: Creating a simple "Employees" list & updating server.js

- First, add a file "employees.hbs" " in the "views" directory
- Inside the newly created "employees.hbs" view, add the html:

```
<div class="row">
```

```
<div class="col-md-12">
    <br>
    <h2>Employees</h2>
    <hr />

    <p>TODO: render a list of all employee first and last names here</p>

</div>
</div>
```

- Replace the <p> element (containing the TODO message) with code to iterate over **each employee** and simply render their first and last names followed by a
 element (you may assume that there will be a "employees" array (see below).
- Once this is done, create/update your GET "/employees" route in server.js according to the following specification
    - You are using getAllEmployees() from officeData
    - Every time you would have used res.json(data), modify it to instead use res.render("employees", {employees: data});
    - Every time you would have used res.json({message: "no results"}) - ie: when the promise has an error (ie in .catch()), modify instead to use res.render("employees", {message: "no results"});
- Test the Server - you should see the following page for the "/employees" route:

Step 2: Building the Employees Table & Displaying the error "message"

- Update the employees.hbs file 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 6 columns with the headings: **Employee Num**, **Full Name**, **Email**, **Address**, **Status** and **Course ID**
    - The "Email" column must be a "mailto" link to the user's email address for that row
- Beneath … element, add the following code that will conditionally display the "message" only if there are no students (**HINT**: #unless students)

```
<div class="col-md-12 text-center">
        <strong>{{message}}</strong>
</div>
```

This will allow us to correctly show the error message from the .catch() in our route

Step 3: Adding body-parser in server.js

- Add the express.urlencoded({ extended: true }) middleware (using app.use())
- Test your /employees route

Part 3 : Adding a View, Route to Support Adding Employees

Step 1: Adding new file in the views directory: addEmployee.html – write the function below

- Complete the checkLastName() function in the addEmployee.html so that it alerts if user inserts any characters other than letters. If non-alphabetic characters are inserted by the user, then alert an appropriate error message, set the focus to the last name input box and return false. If no error i.e., user inserts only alphabets, then return true. You may check for other errors for this field such as last name cannot be empty etc.

Step 2: Adding "Get" route "/employees/add" in server.js

- Inside your server.js file, add the route "/employees/add", which will simply send the newly supplied "addEmployee.html" file

Step 3: Adding "Post" route "/employees/add" in server.js

- This route makes a call to the (promise-driven) addEmployee(employeeData) function from your officeData.js module (function to be defined below in step 4). It will provide **req.body** as the parameter, ie: "addEmployee(req.body)".
- When the addEmployee function resolves successfully, redirect (res.redirect) to the "/employees" route. Here we can verify that the new employee was added

Step 4: Adding "addEmployee" function within officeData.js

- Create the function "addEmployee(employeeData)" within the officeData.js module according to the following specification: (**HINT**: do not forget to add it to module.exports)
    - Like all functions within officeData.js, this function must return a Promise
    - If **employeeData.EA** 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 "dataCollection.employees"** array **plus one (1)**. This will have the effect of setting the first new student number to 261, and so on.
    - **Push** the updated **employeeData** object onto the **"dataCollection.employees"** array and **resolve** the promise.

Step 5: 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 description view & updating server.js

Step 1:

- First, add a file "description.hbs" in the "views" directory
- Inside the newly created "description.hbs" view, add text content to answer the following (be creative with your design to display the content):
    1. Explain the code in the main.hbs
    2. Explain the input elements in the addEmployee.html
    3. Explain what is returned to the form from checkLastName() function and why?
- Once this is done, add/update your GET "/description" in server.js – this will just render the description.hbs
- Test the Server

Part 5: Publishing your site (optional)

Finally, once you have tested your site locally and are happy with it, it's time to publish it online. Make sure you do not have error in your app before pushing to cloud.

Check the "Cyclic Guide" for more information.

Assignment Submission:

- Add the following declaration at the top of your **server.js** file **(failure to do so will result in zero marks)**:

```
/*********************************************************************************
* WEB700 – Assignment 05
* 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 (Cyclic) Link: _____
*
********************************************************************************/
```

- Compress (.zip) your assignment folder and submit the .zip file to My.Seneca under **Assignments** -> **Assignment 5**

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 (11:30PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.