WEB700 Assignment 4

Submission Deadline:

Thursday, July 18th, 2024 @ 11:59 PM

Assessment Weight:

9% of your final course Grade

Objective:

Build upon the foundation established in Assignment 3 by providing new routes / views to support adding new students as well as use modern CSS techniques to create a "theme" for your application.

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. Please note: the "home", "about" and "htmlDemo" html files will not be included in the clean version of Assignment 3

Specification:

For this assignment, we will be enhancing the functionality of Assignment 3 to include new routes & logic to add new students to the system. This will include building a form from a template and wiring up new "GET and "POST" routes. We will also use our knowledge of CSS to customize the appearance of our app by adding new colours, fonts and design options.

Part 1: Creating a "theme.css" file within a "public" folder.

Step 1: Creating a "theme.css" file and linking to it from our HTML files.

- Within the root of your application, create a new folder called "public"
- Within the "public" folder, create a "css" folder (this is where we will place our CSS file(s)
- Finally, within the "css" folder, create a "theme.css" file.
- Now that we actually have a "theme.css" file (albeit an empty one), we must include it in all of our "views" (.html files), including:
    - home.html
    - about.html
    - htmlDemo.html

  This will involve using the appropriate <link> element in the <head> of your files (**after** the bootstrap CSS).
- With this complete, we must use the built in express "static" middleware within our server.js file to identify our newly created "public" folder as a source for static files.

Step 2: Updating "theme.css" to provide a unique look / feel for your application

Now that we have a "theme.css" file and its correctly linked in our html files, we can start to think about personalizing our web app by adding some CSS. There are plenty of resources online to help you pick colours and find (as well as generate) interesting styles to apply to selected elements. Some quality resources to get your started include:

- This "Colour Wheel", used to pick complimentary colours and get their "hex" values: https://www.canva.com/colors/color-wheel/
- A "Box Shadow" Generator, used to provide the complete CSS for adding a "box shadow" to an element: https://www.cssmatic.com/box-shadow
- Sample CSS for generating a "full page gradient" (**Note:** for this example to work for us, we must change the selector from "html" to "body, html" and instead of "#red" and "#blue", simply use "red" and "blue" – or whatever other colours you like): https://coderwall.com/p/ape0jg/full-page-gradient-background

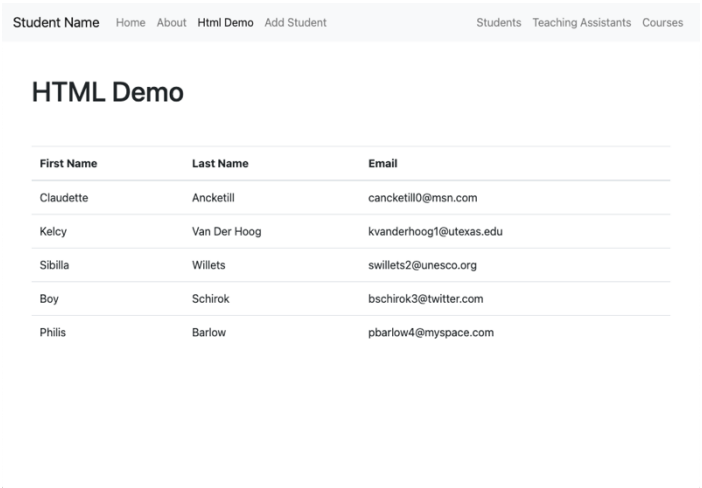
  Additionally, to ensure that your gradient covers the entire page, regardless of how far you scroll, you must add "**overflow: auto;**" as well.
- Google Fonts, used to generate an @import statement to add to our "theme.css" file that will enable us to use 3rd party "web fonts" (**Note:** Add the @import statement at the top of your theme.css file and add the "font-family" style to whatever element you wish to style the font): https://fonts.google.com/

Regarding our own app, we must style the following elements to have an impact on our layout (see the table in Step 3: Elements and Styles). **Note:** This is the **minimum** amount of styling required. Please feel free to add **additional styles and html** if you have a specific vision for your app that goes beyond the requirements.

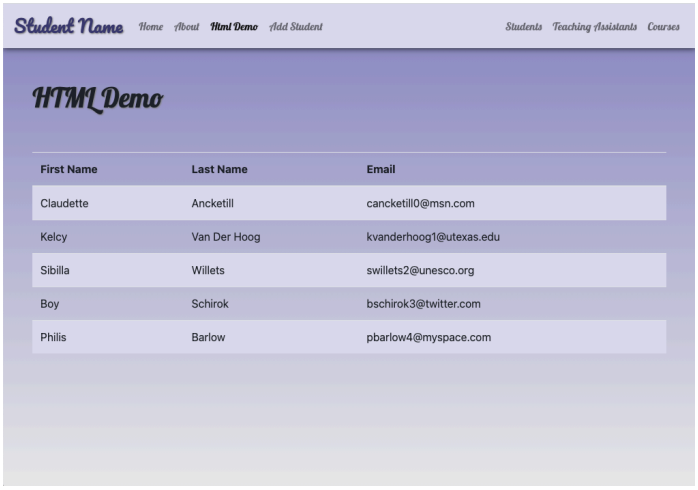
Step 3: Elements and Styles

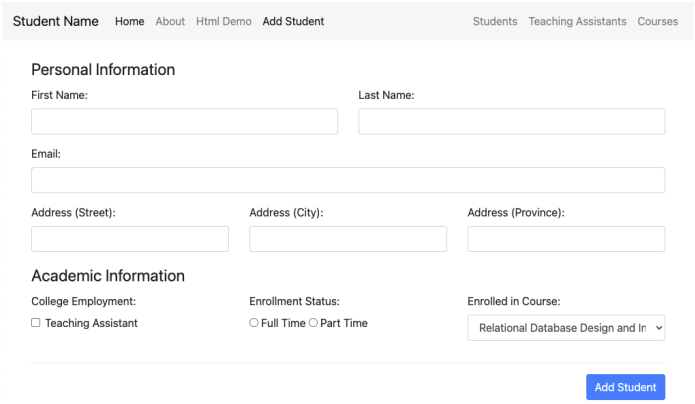
The below table can be used as a guide to help you style your application. To get a sense of what your application could look like after applying you own styles to the following elements, please see the following "before" and "after" images (**NOTE**: Consider adding class="table" to your <table> tags – this was done in the sample):

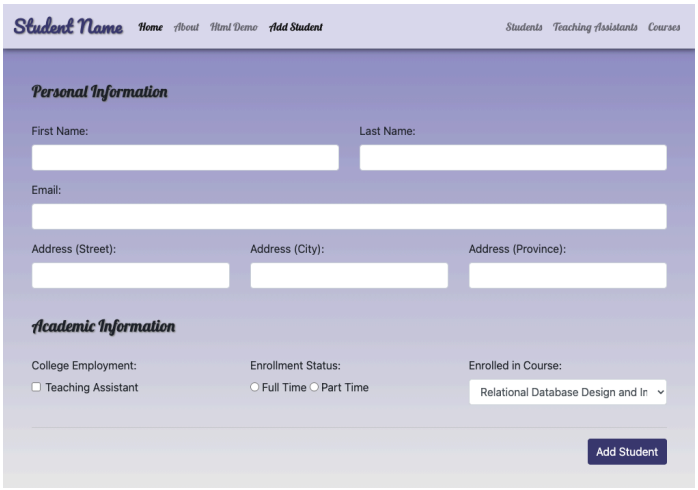
Before



After







Element	Style Suggestions
Navigation Bar - Element with class "bg-light"	- Background colour or gradient - Box shadow (outline or inset)
Navigation "Brand", ie: "Student Name" - Element with class "navbar-brand" inside an element with class "navbar-light"	- Cool font from "Google Fonts" (do not forget the @import statement) - New Text Colour - Text shadow
Navigation "Link", ie "Home", "About", etc. etc. - Anchor element with class "nav-link" - Anchor element with class "nav-link" when "hovered"	- Cool font from "Google Fonts" (do not forget the @import statement) - New Text Colour / hover colour, etc.
App Background - Both body and html elements	- Full page gradient - Solid background colour - Image (consider using "cover" - https://www.w3schools.com/cssref/css3_pr_background-size.asp)
"Primary" Button (used in Add StudentForm) - Element with class "btn-primary" - Element with class "btn-primary" when "hovered"	- Solid Background Colour - Gradient Background - Box shadow (outline or inset) - Different colour for hover

Table cells inside "odd" numbered table rows - "td" elements, inside all (odd) "tr" elements (see :nth-child(odd) pseudo-class), inside a "table" element	- Solid Background Colour - Gradient Background
All "Heading" elements - All elements from "h1" to "h6"	- Cool font from "Google Fonts" (do not forget the @import statement) - New Text Colour - Text shadow
All "Anchor" elements (HTML Links) - "a" elements - "a" elements when hovered	- New Text Colour
Fieldset "Legends" (used in Add Student Form) - All "legend" elements	- Padding Top / Bottom - Cool font from "Google Fonts" (do not forget the @import statement) - New Text Colour - Text shadow

Once the above elements have been styled, there should be a drastic change in the appearance of the web app. However, If you're enjoying tinkering with the look and feel, please feel free to style additional elements or update the existing html structure of the layouts.

Part 2 : Adding a View, Routes & Middleware to Support Adding Students

Step 1: Adding body-parser

- Add the express.urlencoded({ extended: true }) middleware (using app.use())

Step 2: Adding new file: addStudent.html

- Create a new file in your "views" directory called "addStudent.html" and open it for editing
- Copy the contents of "home.html" and paste it in as a starting point.
- Create a new "Add Student" list item in the **<ul class="nav navbar-nav mr-auto"> …
** element that links to **"/students/add"** (to be created in step 4) and ensure that this is the **only** with the class "active", ie: <li class="nav-item active">.
- Remove all html code **inside** the **<!-- Start your code here --> <!-- End your code here -->** Comment block. This will essentially leave the <main class="col-md-12">…</main> element empty

- Inside the (now empty) **<main class="col-md-12">** … ** ** element add the html from here: addStudentForm.txt
- You will notice that the <form> element is missing the **method** and **action** attributes. Add them in according to the following specification:
    - Method: POST
    - Action: /students/add
- Additionally, you will notice that all the **name** attributes are missing on the form elements. These must be created such that the correct form element matches the corresponding **property** in our **student** objects, for example, the code:

```
<div class="form-group">
  <label>First Name:</label>
  <input class="form-control"  type="text" />
</div>
```

  should be updated to use "firstName" as the "name" property, since this field's label and input type matches the type of data we wish to collect for an student's "first name", ie:

```
<div class="form-group">
  <label>First Name:</label>
  <input class="form-control" name="firstName" type="text" />
</div>
```

- **NOTE:** You can check all of the properties for student objects by looking at any given student within the "students.json" file inside the "data" folder. We will **not** allow users to add a "studentNum" – this will be done automatically (see Step 6).

Step 3: Update "Add Student" link in other views

- To ensure that users can access the "Add Student" link from anywhere within the application, add the "Add Student" link to all other remaining views (.html files). This should be the same list item from Step 2, only without the class "active", ie: <li class="nav-item">.

Step 4: Adding "Get" route "/students/add" in server.js

- Inside your server.js file, add the route "/students/add", which will simply send the newly created "addStudent.html" file

Step 5: Adding "Post" route "/students/add" in server.js

- This route makes a call to the (promise-driven) addStudent(studentData) function from your collegeData.js module (function to be defined below in step 6). It will provide **req.body** as the parameter, ie: "addStudent(req.body)".
- When the addStudent function resolves successfully, redirect (res.redirect) to the "/students" route. Here we can verify that the new student was added

Step 6: Adding "addStudent" function within collegeData.js

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

Step 7: Verify your Solution

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

Part 3: Pushing to Heroku

Once you are satisfied with your application, deploy it to Heroku (This will require having an account on Heroku as well as having the Heroku CLI Installed):

- Issue the following command from the integrated terminal: **git init** - this will initialize a local git repository in your working folder. You will notice that an icon in the left bar now has a blue icon showing a number. This represents all of the files that must be added to our local git repo. Click the button and type “first commit” for the message in the “Message” box. Once this is done, click the checkmark above the message box to commit your changes.
- **NOTE:** If, at this point, you receive the error: “Git: Failed to execute git”, try executing the following commands in the integrated terminal:
    - git config --global user.email "you@example.com"
    - git config --global user.name "Your Name"
- Once this is complete, attempt to commit your code again.
- Now that our local git repo is ready to go, we have to create an application within Heroku to send our code to. This is done by issuing the following commands in the integrated terminal:
- **heroku login** - this command will prompt you to enter your Email and Password. Once you have done this successfully (by providing the correct email and password for your Heroku account) you will see a message “Logged in as …” where … is your email address.
- **heroku create** - this command will create our new app within Heroku! The name that it gives our app is random (you can change it later). You will know the name it has given your app by looking at the text next to creating app… done, [*app name here*]. You will also see a url in the form **https://[*app name here*].herokuapp.com**
- We’re getting close, but not quite done yet - we need to issue one more command: **git push heroku master** - this command pushes the content of our local git repo to our new app on Heroku

- **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). To delete an app, login to the Heroku website, click on your app and then click the **Delete app…** button under "**Settings**".

Assignment Submission:

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

```
/*********************************************************************************
* WEB700 – 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 3rd party web sites) or distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
* Online (Heroku) Link: _____
*
*********************************************************************************/
```

- Compress (.zip) your app folder and submit the .zip file to My.Seneca under **Assignments** -> **Assignment** 4

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, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.