COMP 2406 B - Fall 2022
Tutorial #6

Due Sunday, November 13, 23:59

Objectives

- Use Express to develop an HTTP web application
- Begin using RESTful design in web applications
- Practice providing functionality to create and retrieve resources from a server
- Use NPM to organize dependencies and allow easy installation

Expectations

You need to submit solutions to all the problems in this tutorial. Our TAs will mark your submission. Remember to use the available resources (w3schools, Node.js documentation, Eloquent JavaScript book, lecture materials, etc.) for more information if you are struggling to complete the problems.

Marking scheme. For each tutorial, you will receive:

- 2/2 for submitting high-quality solutions to all problems. "High-quality" means that your code works (solves the problem) and is also neat and concise (no overengineering, please).
- 1/2 for submitting solutions for all problems but some need improvement, or you are missing a problem.
- 0/2 if you are missing several problems or your solutions are poorly done; or you do not make a submission, or your submission cannot be executed.

Problem 1 (Creating an Express Application)

In this tutorial, you will be creating an Express-based application to store/serve recipes. This application will allow users to create new recipes, browse existing recipes, and view recipes. To start, download the **T6-Base-Code.zip** file from Brightspace. This contains some base HTML and JavaScript, along with an Express-based static server that you can use as a starting point. Before running the server code, you will have to install the required modules from NPM. First, execute the command `npm init` from the terminal within the directory containing your tutorial code. This will create a **package.json** file to store your project's dependency information. You will then have to install Express using the command `npm install express`. If this was successful, Express should be added into your **package.json** file as a dependency. Run the server code and ensure that the home page shows up properly when you access

http://localhost:3000/ and **http://localhost:3000/index.html**. Inspect the code and ensure you understand how it is working before moving on to the next problem.

Problem 2 (Creating New Recipes)

Initially, there will be only three recipes on your server, which are included in the `database` object within the server code. Open the **create.html** and **addrecipe.js** files in the public directory and inspect the code. The **create.html** page allows a user to enter recipe information. When the **Save Recipe** button is clicked, the **addrecipe.js** file sends the recipe data to the server using a **POST** request to the resource **/recipes**. Note that the code sets the `Content-Type` header so the server knows that it will be receiving **JSON** data in the request.

Within the server code, all recipes will be stored in a single object called `database`. The keys of this object will be unique IDs and the values will be the recipes associated with those IDs. Add a route within the server code to handle **POST** requests to the **/recipes** resource. The handler for this route should:

1. Extract the recipe object included in the **POST** request body. You can use the built-in `express.json()` middleware for this.
2. Generate a unique ID for the new recipe. An easy way to do this it to use a basic integer that increases every time a recipe is added.
3. Add a new entry into the recipes object with the key being the unique ID and the value being the recipe object.

The **create.html** file provides a button to randomly generate recipes. Test your code by using this functionality to add a few recipes to your server. For now, you can log the contents of the recipes object to see that it is storing the correct data. We will add recipe browsing functionality in the next problem.

Problem 3 (Browsing Recipes)

The **index.html** page also includes an HTML link to the resource **/recipes**. Create a **GET** route for the resource **/recipes** on the server. Add a template engine to your server and use it to generate an HTML response that contains a list of all recipes stored on the server. Each entry in this HTML list should have the text of the recipe name and a link to the address **/recipes/{recipeID}**, where **{recipeID}** is the unique ID assigned to that specific recipe. For example, a single entry may look something like:

```
<a href="/recipes/53">Hamburgers</a>
```

Problem 4 (Serving Recipes)

Now, add a route to the server that handles **GET** requests to the parameterized URL **/recipes/{recipeID}**. In this case, the **{recipeID}** parameter will indicate the unique ID of the recipe the user is trying to retrieve. If the specified ID indicates a recipe that exists on the server, then the handler should generate the HTML to represent the specified recipe and send it as the response to the user (like the previous problem, use the template engine to do the generation). As with the list, this HTML does not have to be anything fancy, but should include all the data about the recipe (name, prep/cook time, description, ingredients list). If the ID does not exist, then the server should respond with a **404 error**.

At this point, you should be able to add and retrieve recipes. One of the advantages of using NPM to manage your project dependencies is that it can simplify the install process. You should no longer include the **node_modules** directory with your submission. Copy your server code and resources, along with your **package.json** file to another directory (leave the **node_modules** directory out). Navigate to the new directory in the terminal and run the command `npm install`. This should install all the dependencies for your project, and you should then be able to run your server again without any additional steps. It will be required that this tutorial, any future tutorials, and any future assignments are submitted in this way.

You must include a README file with detailed instructions on how to run your server and test your application. If the TA cannot run your server, you will receive 0 marks for this tutorial. Make sure you download your zip file and check its contents after submitting. If your zip file is missing files or corrupt, you will lose marks.

Problem 5 - Optional (Further Challenges)

If you are looking for more interesting extensions to this code, consider the problems below.

1. Another function you may support is the editing of recipes. Modify the HTML representation that a user gets in response to recipe requests so that they can edit the recipe data and send the new data back to the server using a **PUT** request. Add a parameterized **PUT** handler for **/recipes/{recipeID}** so a client can **PUT** a new representation of the recipe containing modified data and have those changes remembered by the server.
2. If you have many recipes, it will be infeasible to list all of them when the user makes a **GET** request to **/recipes**. Instead, implement a pagination mechanism that will show, for example, the first 10 recipes for an initial request. Provide **next/previous** buttons to allow the user to navigate the complete list of recipes.
3. The RESTful design principles (that will be discussed in class) specify that the server should mark data as cacheable or not. It is logical to assume that the recipe data on your server will not change often. Add the appropriate headers to your response to enable clients to cache recipe pages locally. Request the same recipe a few times to verify that this mechanism is working.

Problem 6 (Save Your Work & Submit)

You are not required to solve/submit Problem 5. Once you have completed all the problems (except possibly Problem 5) for this tutorial, **zip** (compress) all your files for this tutorial into a single **.zip** file and submit it to the Tutorial submission on Brightspace. Your dependencies must be installable via the `npm install` command. You should not include your **node_modules** folder in your submission.

You must include a **README** file with detailed instructions on how to run your server and test your application. If the TA cannot run your server, you will receive 0 marks for this tutorial. Name your file **T6-YourName.zip**. Make sure you download your .zip file and check its contents after submitting. If your .zip file is missing files or corrupt, you will lose marks.