CAS CS412 MEAN PS4: External Data in Node

Time to start working with external data.

Requirements

0. Create a new branch off of main, PS4. Be sure that your .gitignore file has an entry to exclude node_modules/ from being pushed to github. You also should include a .gitignore entry for the directory that contains configuration information, such as API keys and endpoints.

1. Build a Node/Express application. The app must have:
   a. An instance of express.Router in a separate route file that is mounted on the path /ps4 (i.e. app.use('/ps4'…)).
   b. A route on a **POST** method that retrieves data from an external API. For example, you might hit the Weather Channel API to retrieve the current weather conditions. ProgrammableWeb.com is a good spot to find APIs offering data you are interested in. The route must use **Promises** to manage the async http call. Use the 'request' package as your HTTP client (with the understanding that it is deprecated). Wrap the request in a Promise.
   c. A second route, similar to the one in b. that uses **async/await** syntax rather than Promises. You can hit the same API endpoint as b. Use 'node-fetch' for this one.
   d. A third route, similar to b. and c., that uses a **callback** to handle the async API call. For this one, use the 'request' package. It's fine to hit the same endpoint.
   e. A back-end rendered template to display the results of your call. This can be EJS, Pug, or any other templating language that you'd like to use. You can use the same Pug template for all of your routes.
   f. An HTML form to send a search string into your route. For example, if you are hitting a weather API, you might have a form that asks for a city. You can use Pug for this page, or EJS, or write straight HTML. Your form will use a POST method to submit.

g. A file that contains **configuration** information, and that is not pushed to github. This would be URL endpoints, API keys, and the like.

2. Submit the following files on Gradescope:

   - app.js
   - router file ps4.js
   - HTML / Pug file containing your form

You should not submit your configuration files.

A note about choice of API
You will be using this API in future assignments, connecting a Node backend to an external API, driven from an Angular frontend. It's helpful if the API you choose returns a data set that has different levels of information — one upcoming assignment has you 'drilling down' to get detail information, for example. It also is useful to choose an API that returns an iterable set of data — another assignment will ask you to display iterated data in Angular, and if your data set is already structured that way, you won't need to fake anything.

Also, some APIs come with rate limits (ie 5 calls per day), cost, or take significant time to apply for and receive a key. Try to find a data set that satisfies everything above but also is free and offers instant access. You won't have time to wait for a key. For example, I applied for a last.fm dev key last semester, and still haven't received it.

It's tempting to use an API like Twitter, but be aware that most of the social media services will require an OAuth login to get user data, and we won't be talking about OAuth for another week or so.

If for some reason you pick an API and it stops working, or doesn't seem to be what you want, it's fine to pivot to a different API later. I just wanted you to know that a little planning ahead of time would help later in the semester.

Overview of data flow
Your app should present a form with a text box and a 'submit' button. The simplest approach is to use a route on '/' that renders your form, so that hitting http://localhost:3000/ will display it.

You are going to hit three different back-end routes off of form submissions, one at a time. The simplest way to handle this would be to duplicate the form so that you have three submit buttons on your initial page, each hitting one of the three back-end routes.

The form will send a piece of text to the backend. If you are using a weather API, you might submit the name of a city to get the weather for. The back-end route will take the information from the form data and use it to construct a call to your third-party API. Once you get the data, render it in a Pug or EJS template. Since the data should be the same for each of the three routes, you can render the same template.