Web Scraper App

Author: Michael Jee

Created: January 28th 2023

**Approach:** After reading the prompt, I made a mental checklist of local environment items I would need and outlined a design on how I would accomplish the task.

I designed the application flow would be as follows:

1. The frontend presents the user with a form that allows them to enter a url and search term

2. The user enters a url and search term (since strings, not much validation)

3. The frontend takes this input and makes an API call to the backend

4. The backend accepts this request with its single API endpoint (I expected CORS issues here). It would make an http request to a webpage, get the body of the page, count the number of times that the term shows up, and returns this number (and perhaps echos back inputs) to the user. If there are any exceptions, return an error to the frontend

5. The frontend then takes the response data and presents the result to the user

I decided to use javascript for the backend to compliment the ReactJS frontend. I slowly scaffolded my **two** applications and added features to complete the task as I went

**PROS:**

- I am most familiar with javascript of the three backend options, which would make setting-up a server the quickest and easiest which is good since my previous API experience uses Spring (Java)

- on further inspection, ReactJS has a good offering of libraries I could use the for the backend, making standing up the project much faster

- two applications makes the code cleanier and is better practice from what I've seen

- developing in increments helped me debug faster and also learn how items interact better

**CONS:**

- a bit less incidental learning from using python or ruby as the backend server

- two applications can be a little clunkier to host, test, and develop

- developing in increments is slower when I know what I'm doing, but prevent careless mistakes from piling up and making debugging nightmareish

**What gave me trouble:**

**Backend items:**

- CORS made my frontend unable to make calls to the backed. I was able to resolve quickly since I suspected it would be an issue and have fixed it before in Spring

- I struggled a lot getting my fetch() request to successfully get the webpage data. I eventually figured out that the issue was I needed to make the requests ASYNC and add 'await' in a couple areas. I suspect some parts of the methods ran before the page request went through and stalled out the code.

**Frontend items:**

- I had to take some time to get used to using ReactJS syntax, but tutorials & googling got me there

- It took me some time to get the syntax and logic for custom form submission correct

- most other issues had to do with integrating with the backend

**Time spent:**

**Set-up:** I about an hour to set-up for ReactJS. I had Git Bash, npm, and node on my personal system, but then had to go over a couple tutorials to get set-up for ReactJS specifially, read over general documentation on React, and how it differs from Angular, which I use for work. The remainder of this time I spent scaffolding my frontend and backend to get them to work with minimum functionality.

**Developing:** I spent about 3 hours of actual development then actually building the frontend and backend syntax, logic, and integration.

I used some additional time to clean-up my code and shore-up documentation such as my README and this EXPLAIN.

With more time, I would work on the presentation of the UI and the general UX, but the required fucntionality is there. In terms of backend, I could use time to refine requirements, such as if the task is only supposed to count the search term in the actual content of the page as opposed to the entire body. I made it the entire body however due to time constraints and since I figured the metadata has value. More features could also be added such as separating the count by if it comes from metadata or just the content, adding in the ability to search for two+ terms with customizable logic (AND, OR, XOR), and a bulk search feature.