Documentation for Book Search Assessment

By: Jesse Woo, USDC 2024 Candidate

Overall process and decision making

My solution is quite straightforward, it iterates through all data in the scannedTextObj (which is to say, all the nested JSON Book and Scan objects, if any), and looks for a simple match of the substring from the searchTerm parameter within the “Text” field of the Scan. I initially considered using regular expressions to search for exact matches over discrete words, but I felt that using substrings would give the user more precision in their searches. For example, a user who searches for “run” but does not want to return a match for “running” could specify this by searching for “run “, with a space. However, if they did want to search for “run” and return possible matches where the term is a substring (e.g. “run” and “running”), they can do so by searching with “run”. Using a regex with exact word matching would not allow for this easily. I felt that possibly returning false positives like “running” along with “run” was better than missing matches because the regex pattern was too inflexible.

The other reason to use substring matching is just that it is simpler. I find the code easier to write, read, and debug than a regular expression, which can get quite complicated. I use the built-in javascipt includes() function to accomplish the search in one line of code. Of course, using stemming or other NLP techniques would allow for more flexible and sophisticated searching, but given the limited time to implement this solution and the inability to use external libraries, I opted for something simple. This does put more burden on the user to be precise with their search terms, but I find that iterative development works best when I start with the most obvious solution and build up from there. Beyond the string matching there are basic checks for input type and empty objects, mainly to avoid runtime errors in the event of badly formed inputs.

Testing and iteration

My strategy for writing tests is to first test the most obvious things, namely that the code produces the expected output on easy inputs. Then I test other behaviors that I want to ensure are there, namely that searching for substrings that are not in the data will not return a match, and searching for a substring of a word *will* match. I also wanted to be sure to test for mal-formed data such as empty objects or objects with null values where they might not be expected. I wanted to ensure that data cleanliness issues would not cause runtime errors because I know that the only reliable expectation when working with large amounts of data from diverse stakeholders (such as in the government), is that it will be messy. I also wanted to confirm that my solution worked when there are multiple matches, or even multiple matches in the same page and line. My solution is quite simple so it handles this easily, but as code becomes more complex these kinds of corner cases can cause unexpected problems.

I don’t know that there is anything that is particularly hard or difficult about my code, but I was pleased with myself when I figured out that I could test equality of the expected and test output with JSON.stringify. Testing equality between each field of the result would not be difficult but it would be tedious and verbose, especially if I am expecting multiple matches in a book. Given more time, I would expand my testing to more edge cases with a focus on unexpected data types or hygiene issues. Currently, I assume that if a Book or Scan actually has data in it, it’s fields will follow the specified structure. With more time I might include more error checking to confirm this. The best way to test code is to have someone else use it, so ideally with more time I would also give it to others to beta test.