Frederick Nguyen
US Digital Corps
2024 Software Engineering Project-Based Assessment

Documentation of Project-Based Assessment

Overall Process

- Problem Description
    - The task at hand is to create a JavaScript function, findSearchTermInBooks, that searches for a given search term within a JSON object representing scanned text. The function should return search results indicating where the term was found.

- Solution Approach
    - In creating the findSearchTermInBooks function, the goal was to make searching for terms in a digital library easy and efficient. Imagine needing to find a needle in a haystack; this function is like a magnet to find that needle faster. I started by understanding the needs: quick, accurate searches in a large collection of books. I chose tools that are known for handling such tasks well. The key was to make the function work like a helpful librarian, who knows exactly where to find what you're looking for.

Thought Process and Decision Making

1. Inputs: It takes two main inputs: the word you're looking for ('searchTerm') and a collection of books ('scannedTextObj'), which includes details like ISBN, page, line, and text.
2. Setting Up a Detector: The function sets up a 'regular expression' (regex) - think of it as a smart filter that can recognize the word you're looking for in different variations (like 'Book' vs 'book').
3. Avoiding a Blank Search: If no word is entered, it skips the search and returns no results.
4. Scanning Each Book: It then goes through each book, checking every line on every page.
5. Detecting the Word: If it finds your word in a line, it verifies with the regex to ensure it's the exact word (not part of another word).
6. Gathering Results: Every time it finds the word, it notes down where it found it (ISBN, page number, line number) and adds this to the results.
7. For test cases, we checked for different words that were included and not included inside the object we are testing, which is the book. In this case we also tested for edge cases like empty inputs, and substring of words. We also tested edge cases like hyphenated word breaks.

Frederick Nguyen
US Digital Corps
2024 Software Engineering Project-Based Assessment

Testing and Iteration Testing Strategy

General Overview

For testing, I devised a comprehensive set of test cases to validate the functionality of the ‘findSearchTermInBooks’ function. These test cases cover various scenarios, including positive tests, negative tests, and case sensitivity tests. The function performed well in positive tests, correctly identifying the search term and returning the expected results.

Technical Approach

- To solve this problem, I followed these steps:
- Initialize an empty result object to store the search results.
- Create a regular expression to perform a whole-word search with case insensitivity (to match the exact search term).
- Loop through each book in the scanned text object.
- For each book, loop through its content items (pages, lines, text).
- Check if the search term is an empty string. If it is, return an empty result object.
- For each content item, check if the search term exists in the text, and if it matches the whole word using the regular expression.
- If a match is found, add the book's ISBN, page number, and line number to the results object.
- Return the results object.

Unit Testing

We started with unit tests, focusing on individual components of the function. This included testing the basic functionality - whether the function could accurately identify a given search term in a text snippet.

Positive and Negative

Testing Our tests were designed to cover both positive and negative scenarios. Positive tests checked the function's ability to correctly identify and return instances of the search term. Negative tests ensured the function properly handled cases where the search term was absent, avoiding false positives.

Case Sensitivity Tests

Given the importance of accurate search in a digital library, we rigorously tested for case sensitivity. The function needed to reliably identify search terms regardless of their case (e.g., 'Book', 'book', 'BOOK').

Frederick Nguyen
US Digital Corps

Considerations

1. I decided to use a regular expression for whole-word matching to ensure that we only find exact matches of the search term.
2. To handle case sensitivity, I made the regular expression case-sensitive.
3. An empty search term should result in an empty result object, as there is nothing to search for.

Proudest Achievement & Most Challenging Aspect

The part of the solution I am proudest of is the regular expression-based whole-word search. It ensures that the function finds exact matches of the search term and doesn't mistakenly match substrings. I was most proud of the part because it was difficult to deal with substrings otherwise, coming with this solution was a good way to make sure that the word we are searching for is exactly the same instead of fragments of words.

On the other hand, the most challenging part was managing case sensitivity. Developing a method that consistently recognizes different case variations of the search term (like 'Book', 'book', or 'BOOK'), while still ensuring that only whole-word matches are returned, required intricate coding and testing. I had to ensure that the function could handle mixed-case variations of the search term while performing whole-word matching. Also some edge cases like searching for an empty string (“”). Were not considered at first until running and going over the program again.

Project Enhancements/Additional Notes on Edge Cases

Given more time, I would address additional edge cases to enhance the function's robustness. This could include:
Hyphenated and Compound Words: Handling cases where the search term may be part of a hyphenated or compound word.
Punctuation Sensitivity: Refining the function to recognize search terms adjacent to punctuation without misidentifying them.
Contextual Understanding: Implementing a basic level of semantic understanding to differentiate between the search terms used in different contexts.
Language Variations: Adapting the function to handle variations in language, such as British and American English spellings.
Special Characters and Accents: Ensuring the function accurately recognizes and searches for terms with special characters or accents.