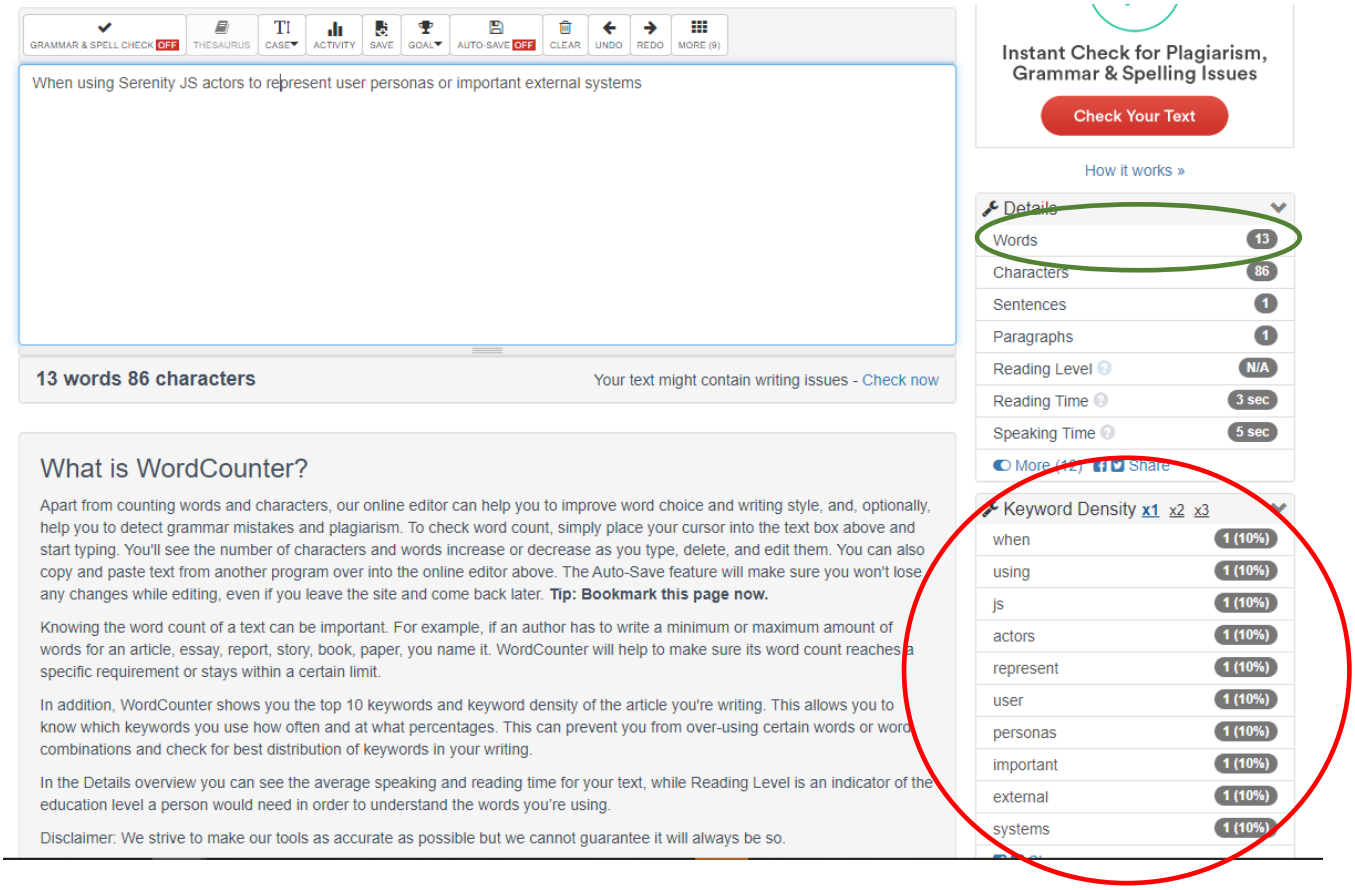
WordCount Web Page Failure Report and UX Suggestions

I. WordCounter Web Page Defect Finding

Description: There is a strange cause of error in the word frequency (Keyword Density) report: The service omits words in its results.

Replication Procedure:

1. Enter this text in the text input box: “When using Serenity JS actors to represent user personas or important external systems”.



2. As you can see the sentence have 13 words, and that is correct in the highlighted word count report (green eclipse), but a little below, into the Keyword Density frame are showed only 10 words. The words: “Serenity”, “to” and “or”, were omitted.

Additional Details:

1. If the last letter of “Serenity” is removed or replaced i.e. “Serenit”, “Serenitx” or “Serenitz” is written, the word will be recognized and will be shown in “Keyword Density” frame. Similarly occurs for the words: “to” and “or”.

II. WordCounter Web Page UX Suggestions

1. **Reorganizing the website.** Currently, the website looks messy. Navigation on the website can be improved. Users might find it hard to explore some of the available features. Assigning a prominent position to main features of the website would be an improvement, particularly for mobile devices

   Although the website provides a lot of useful features, having all the features and resources on one page can be overwhelming to users. Consider reorganizing the website and putting some features on separate pages and making the user experience more focused.

   The “What is WordCounter?” section requires its own page or tab.

2. **Visual Enhancements:** Enhancing the visuals of the website would make it more inviting. Use more visual aids to guide users, improve the display of selected results, and implement a better color scheme to ensure that the website looks modern. Buttons and its titles can be a little bigger and colored.

III. Computational Complexity of the Algorithm to Implement the Word Frequency Counter

The computational complexity of an algorithm it is the sum of the computational complexity of their steps. The steps of the algorithm along its computational complexity are detailed below:

Step Description	Computational Complexity	Location in the Project
1. **Reading the text file:**	O(n)	ReadFile class
2. **Counting word frequency:**	O(n)	getWordFrequencyOf method
3. **Sorting the word frequency map:**	O(n log n)	getWordFrequencyOf method
4. **Printing word frequency and histogram building:**	O(n)	getWordFrequencyOf method
5. **Calculating total words and characters:**	O(n)	getWordFrequencyOf method

In the first step ‘n’ is the number of characters in the text file, in al others steps ‘n’ is the number of unique words. So, the computational complexity of this procedure is mostly generated by the sorting step which is made up by these lines of code:

```
List<Map.Entry<String, Integer>> sortedList = new LinkedList<>(wordFrequency.entrySet());
            sortedList.sort((o1, o2) -> (o2.getValue()).compareTo(o1.getValue()));
```

“sort()” method used here, internally implements a variation of merge sort for sorting the list of entries by their values (frequencies). The merge sort algorithm is on of the most efficient sorting algorithms, so we can trust this is a highly efficient implementation.