NEW FEATURE REPORT: SEARCH

INTRODUCTION

For my new feature I chose to implement a search function, where one can enter a search query and the website would present you with results it could find which are relevant to your query.

DESIGN DECISIONS

When I first implemented the repository pattern, I added several search methods which would find books containing the given query in the title, authors, publishers, etc. At the time this seemed appropriate, and these were functions which I could likely expect any given backend to support. However, I quickly realised that this would be far to primitive of an approach, as it would only give exact matches, so if the user was not able to remember exact matching book titles or author names, they would not be able to find what they were looking for. This would essentially mean this form of search being pretty much useless.

I gave some thought to what would be appropriate to see presented on the search results page, and what I would want to see. I decided that searching all the details of each book might be a bit overkill and would likely result in too many results that were not what the user was looking for.
I decided that I would have 3 sections: Books, Authors and Publishers. Books would be searched by title, authors by their full name, and publishers by their name.

My next thought was to check the Hamming distance between the query and any potential results. I excited about this as it could potentially even find matches which contained typos, however I quickly realised that unfortunately, the search query and any potential results, would not be strings of the same length, and so a primitive implementation of Hamming distance would not be suitable, as it would only search the first portion of any potential results' strings.

Giving it some more thought, I realised I could split the query and any potential result up into separate words and then search for each of these separately. Unfortunately, even then, each word in the query would probably not be the same length as words in the results – though they would be more likely to be of a similar length. This would still give too many irrelevant results, so using Hamming distance felt inappropriate.

Instead, during search, I would store a list of "distance" integers initially set to the number of words in the search query. Then, for each potential result, for each query word that is also in the list of the potential result's words, I subtract 1 from the distance. Finally, if the distance is still equal to the number of query words, I do not return the result.

I can then sort the list of books by their “distance” value from the search query, giving me results most relevant to the search first.

Though in its current state, the search was working quite well, if only 1 word was entered as a query, I felt that maybe it should not split the potential result into separate words, but instead just check if the queried string appeared, at all, in each potential result. This works quite well to help if the user does not finish entering a word, for example if they aren’t exactly sure what they want to look for but know a little bit about what the title or name that they are looking for is.

To improve on the distance calculation, I switched the counting the number of occurrences of the query in each potential result. This would help to give the more relevant results first. For example, one might search “it”, then the results will be sorted by the number of times “it” appears in each result.

As a small improvement, I remove all characters from the initial query that are non-word characters or whitespace characters, so basically removing all punctation. I then strip the whitespace from the ends of the string.
Finally, if the query now has a length of 0, I return an empty list, as if you don’t enter any search query, you would expect not to find any results, but if I let the algorithm run, then technically all potential results would contain “”, and so you would just get all book, authors and publishers (sorted by their lengths, as that would be how many times “” appears).

Initially, I had these search methods as part of the repository, however I realised they were more suited as services, since I don’t want to rely on the repository implementation to have this search algorithm implemented. Having them as services instead also adheres better to the Single Responsibility Principle.

APPEARANCE

When the user enters a query, they are likely to only need the first few results, so I only show a maximum of the first 4, and if there are more than 4, a link to show all results of the given category (Books, Authors, or Publishers). This also means that if the query has a LOT of results, they won’t all be thrown at the user at once – And since Books are shown first, if too many were shown initially, the user would have to do a lot of scrolling to find the Author or Publisher results.

If the user enters a number as the query, and said numbers appears in the list of release years, a link to all books from the given year is shown near the top. This is beneficial as the results’ release years are not used as part of the search algorithm; however, the user might still want results from a given year.