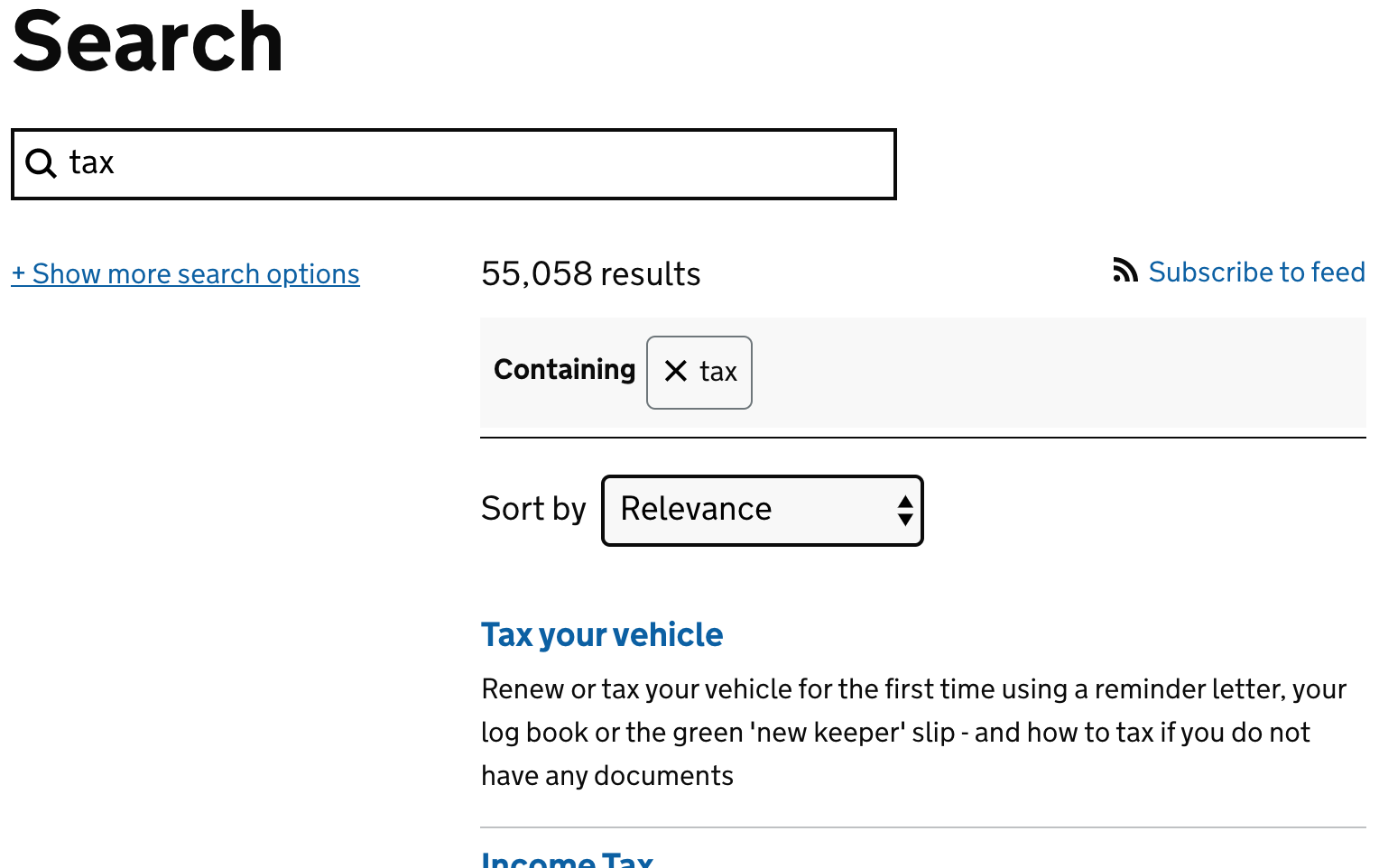
# Adding Highlights to Relevant Search Terms in Their Results

The languages that I used during this ticket were Ruby and HTML/CSS, and there were also some command line functions.

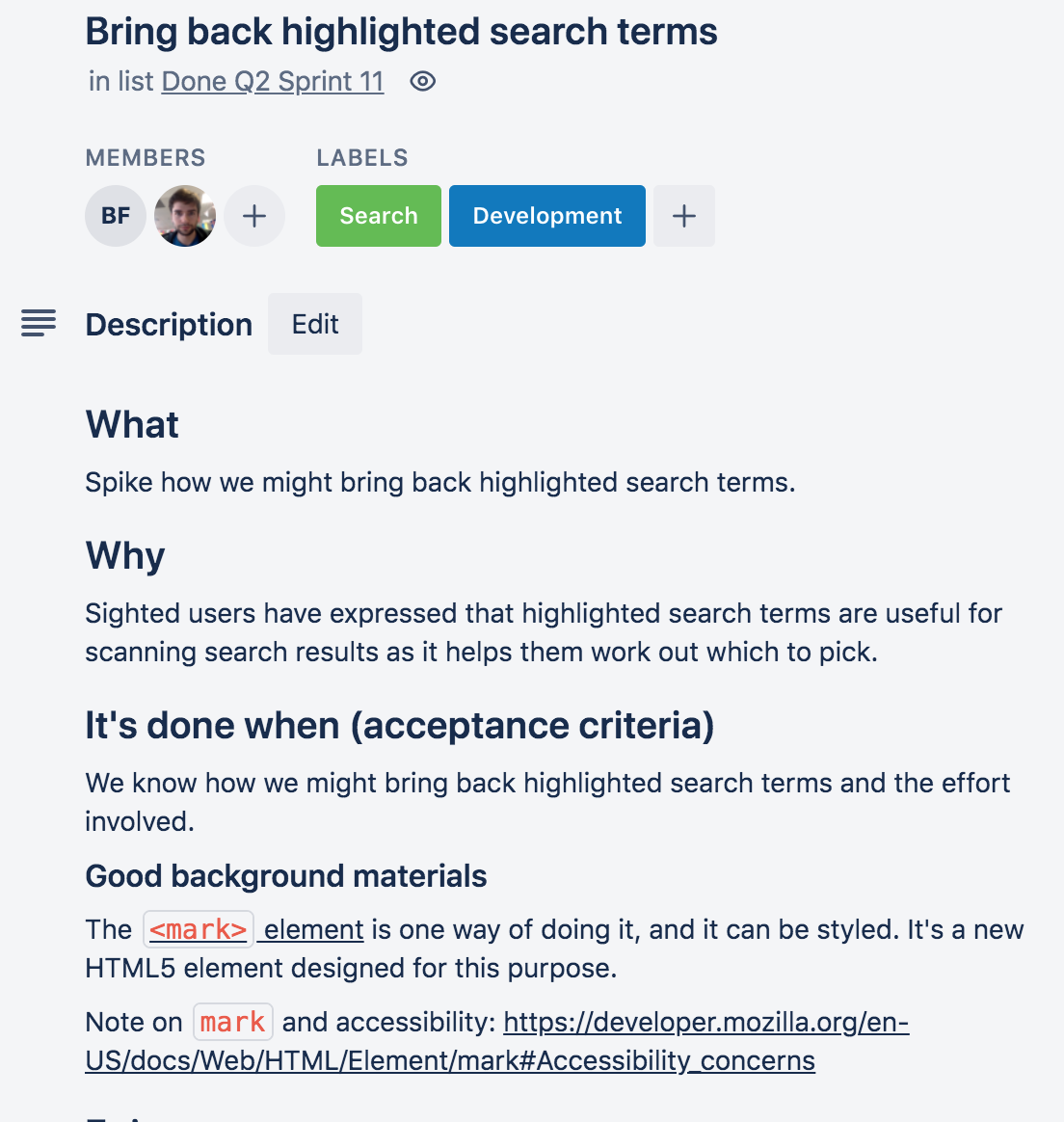
I picked up this ticket along with another developer on my team. This work was fairly straightforward, and involved making a frontend change that would help our users with our search tool.

The basic idea behind the change was to somehow highlight any instances of a search term in the results for that search. In our tool, we present results as a title, and then the first sentence for that piece of content. An example of this is shown below.



The proposed change would make it so that any instances of the word ‘tax’ in the result descriptions would be in bold text instead, giving users more information as to how their search term fits in with the results being presented.

The ticket itself set out the requirements for the work, but also a suggestion on how to style the highlighted terms, using the <mark> html tag.

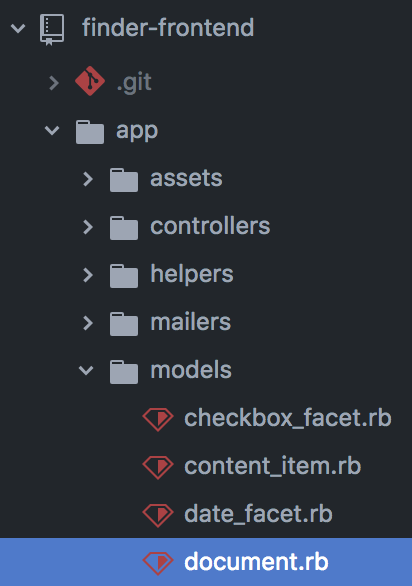


The interesting thing about the <mark> tag is that it’s specifically designed for this sort of thing, and it’s also well understood by screen reading software. This means that anyone using a screen-reader would ideally benefit from this change, making our site more accessible overall.

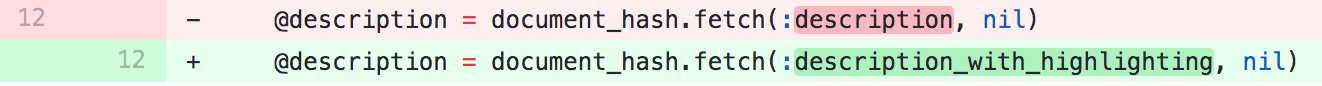
The key issue here was that the default behaviour for the <mark> element is to change the background colour for anything within the tag. I instead wanted text inside the tag to be bold, which meant that I’d need to override the default behaviour and introduce custom behaviour instead.

The first step for this change was to add the tag to the relevant search term. There were a few possible ways to do this, so I sought out some help from another developer on my team to see what they would suggest. I thought that it might make sense to have a Javascript function that would check over the results, and insert a <mark> tag whenever the term showed up. However, my colleague knew about a specific elasticsearch parameter that I could use, which would simply return the results with the markup already in place.

The key to making this change was to modify the parameters that were being requested from the elasticsearch query. I found where the elasticsearch query was constructed, which was in the app/models/document file. This section of the codebase dealt with how data structures were created and information was supplied to them, so it was a suitable place to look. Whenever a search is made, a request is sent to a search api that my organisation is responsible for - and the api in turn sends back a document object in a JSON format. This file handles the parsing and construction of an object that can be used by this application.



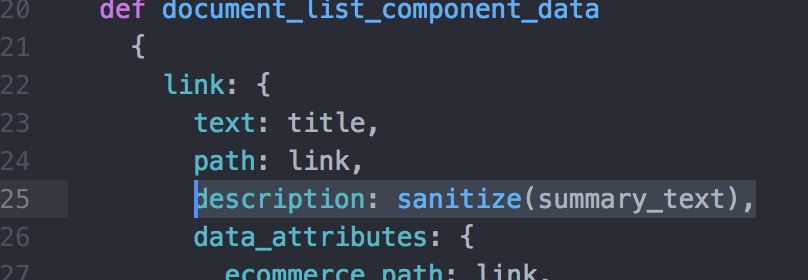
Inside the file, I looked for the section of the code that assigned elements of the returned document. This included assigning the description for the results, and this was where I would need to make a change in order to get back the descriptions with additional markup. The description\_with\_highlighting key would return descriptions where the relevant search term would have the <mark> tags encapsulating them. This was perfect for what I needed, so I modified the @description instance variable so that it would contain the highlighted variant.



I wanted to see what effect this had on the page, so I loaded a local version of the page, and put in a search term. However, there wasn’t any highlighting present - I was at least expecting the search terms to be highlighted, but it looked like the <mark> tag hadn’t been applied.

I wanted to work out at which stage this change hadn’t propagated through, so I went back to the code to check.

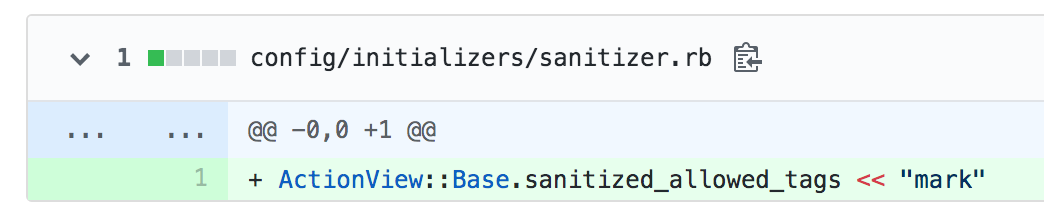
After some digging, I was able to find a function that the description was being run through before it was displayed - this was in the ‘presenter’ section of the app, specifically the search\_result\_presenter file. In here, the ‘sanitize’ function was being used on the description. I originally wasn’t sure what this did, so after some online searching, I found that it was primarily a security feature, used to remove any unwanted markup from search results. This would remove things like inline HTML styling as a security feature, so that the results displayed in the search window wouldn’t behave too unpredictably. Unfortunately, this also meant that the <mark> tags that were being added by the highlighting were then being removed by this command.



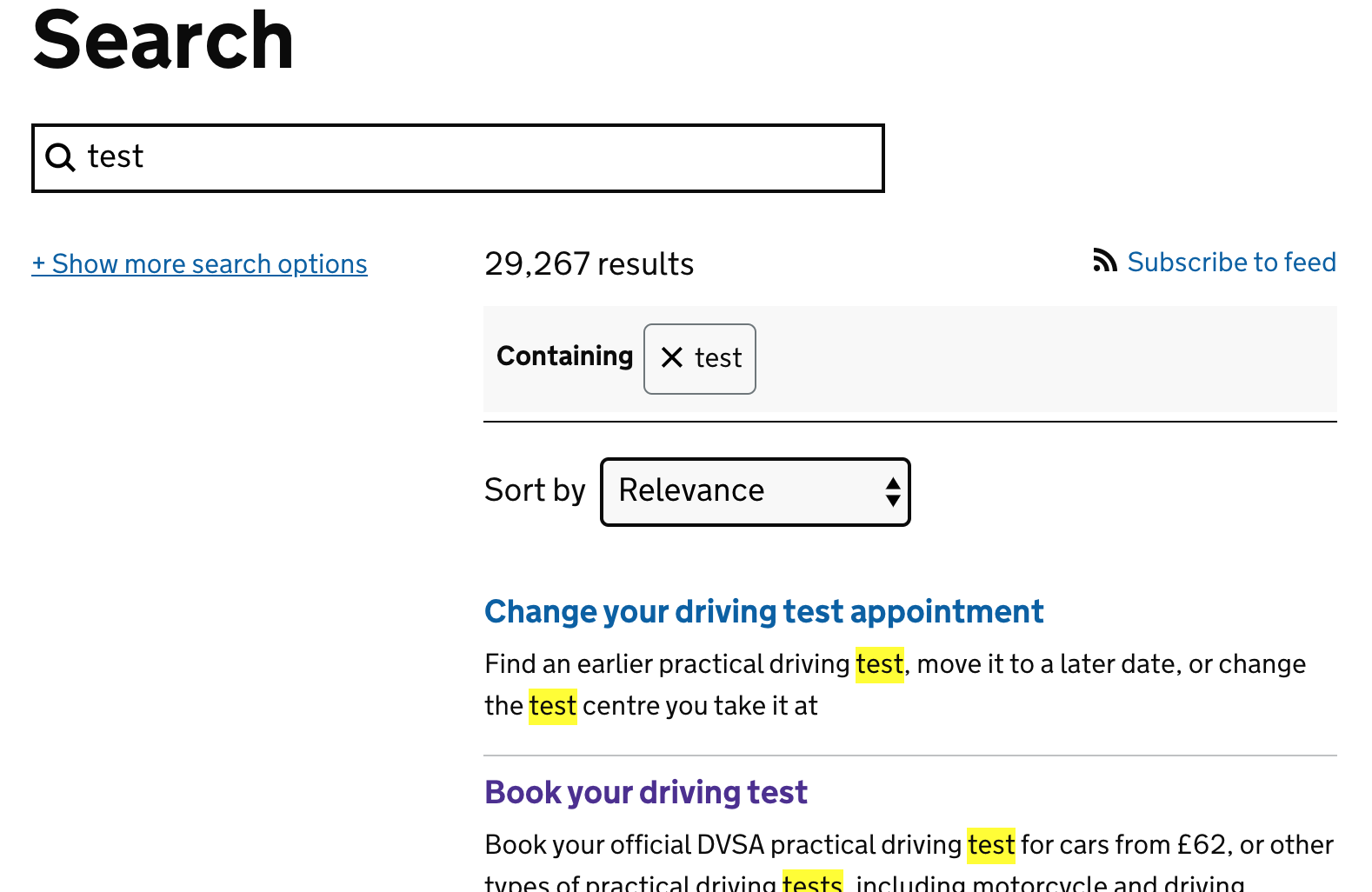
I knew that I needed to ensure that the markup wasn’t removed, but I didn’t just want to remove the sanitizer - this could have unintended consequences, and would compromise the security for the app.

Instead, I investigated further into how the sanitize function worked. It turned out that the function was provided by a gem, and there was documentation on how the gem worked. The gem had a function which would allow me to add a tag to its whitelist, changing the behaviour from the default, and allowing the <mark> tag to run. Because the gem was being loaded in when the app started running, I would need to make this change outside of the presenter layer.

I added a new file in the config/initializers folder, which would execute whenever the application was initialized. In this file, I added the custom command that would add the mark tag to the sanitizer’s whitelist.



After doing this, I ran the application again. It looked like it had been successful, and the search terms were now highlighted in the results. However, they still had the default styling for the mark tag, which wasn’t suitable. I wanted the results to instead be in **bold**, so I would need to make a change to the CSS for the page.

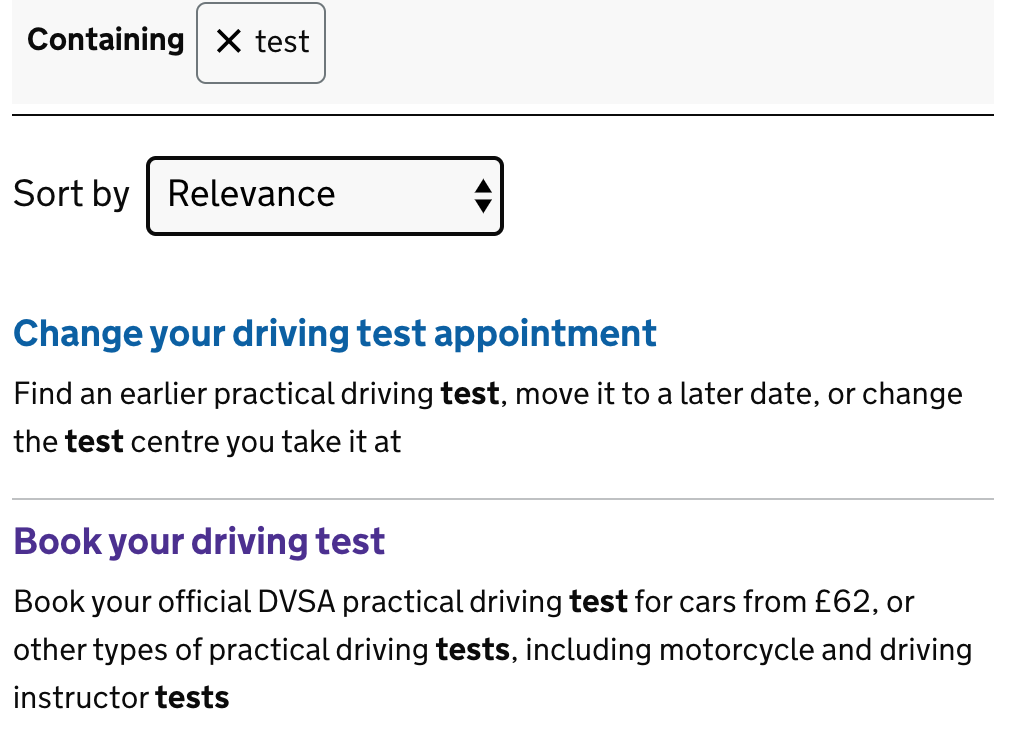


I went to the stylesheets area of the codebase, and looked for the stylesheet that corresponded to the search results page. In here, I added a section that would determine the function that a mark tag would have. I wanted to do two things - remove the background colour change, and make the text bold. The screenshot below shows the changes I made.



This would mean that the background colour would be set to whatever its parent’s background was, and the font would now be styled using my organisation’s custom bold styling.

After doing this, I ran the local page, and it looked like it was working appropriately.



At this point, the feature was complete, but I wanted to make sure there were appropriate tests for it. I ran the suite of tests for the application, and a large number of tests ended up failing. Looking through the stack trace, it turned out this was because I had changed the document hash check from ‘description’ to ‘description\_with\_highlighting’, and there was a factory that was using the old term. The stack trace showed the relevant code, so I went to that document and changed the term.



Additionally, a number of tests were trying to access the ‘description’ key, so I modified all the ones that had flagged this, putting the new term instead. There were quite a few of these, but I was able to use a find-and-replace tool, and match the changes up with the test results as I continually reran them after each change. This let me see which tests were still failing, and whether or not my changes had enabled the tests to pass.



Once all the tests were passing, the ticket’s requirements were set. I created a commit that summarised the changes, and pushed it up to the remote repository. I made a request on the team Slack channel looking for a pull review, and managed to obtain one fairly quickly. At this point, my team member handled the deployment for the branch, merging it into the master branch, and running through the rest of the deployment pipeline. After it had gone live, he posted a message on the team channel, and tagged me in to ensure I knew it had happened.



## Summary

This ticket was fairly straightforward, and I had quite a lot of fun debugging the issues that arose. It was quite challenging to work out where in the chain of events things had gone wrong, as it wasn’t an area of the application I was too familiar with. Learning more about how this application speaks to and parses information from the API was interesting, and I felt like my understanding grew as a result.

Doing it again, I would want to seek help earlier with how to handle the ticket. I spent some time trying to come up with possible solutions, but it turned out there was already an inbuilt solution that I was able to use quite easily. It’s hard to know beforehand whether there’s already a pre-existing function that can do what you need, but I should probably check more thoroughly just in case.

The end result of this change was that search terms are now highlighted in results for our search tool. This makes those terms bold, and lets users see how their search matches the results. Additionally, the <mark> tag can be used by screen-readers to give users a better understanding of how the page is structured, and what parts are important to them.