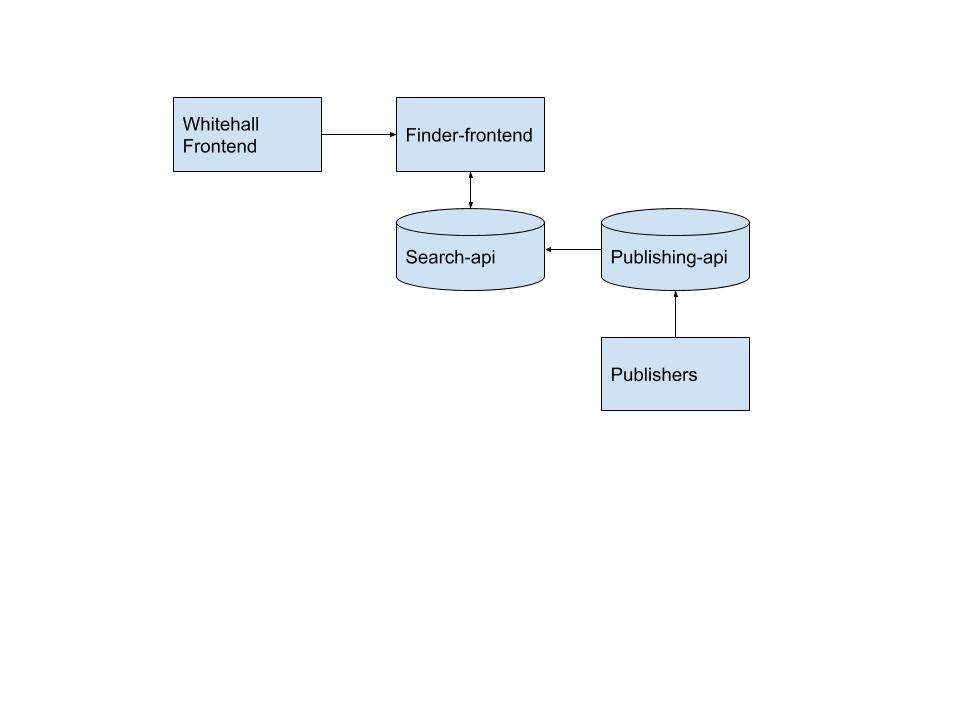
# Adding a Redirect to a New Webpage

The main project I was working for this ticket was part of a phasing-in/out process. My team creates tools that are used to search across a government website, and within that, there are various specialised tools that can be used to search for very specific things. For example, there is a tool that is specifically for finding official publications from the government.

## The Applications

Currently, my team is shifting from an old set of tools to a newer set. This involves decommissioning the old tools, and making sure that users are gracefully introduced to the new tools when trying to access them. This work was brought about because the older tools weren’t fit for purpose - they were designed when the site was much smaller, and therefore their scope was much more limited. As time went on, the site grew in size and in nature, so the search tools started to fall behind in terms of functionality.

I originally created this diagram to help myself understand the structure of the system. The arrows indicate where api calls are made, although the relationship between the Finder-frontend and Search-api component ends up being more complicated than that.



Citizen users will interact with the site through the Whitehall Frontend application. This acts as a framework for the entire site, and it calls on other applications to perform specific tasks. For searches, this application talks to the Finder-frontend application, which provides the frontend code for the search tools. This is where the bulk of my work took place, although not all of it.

The Finder-frontend application will make requests to the Search-api, which is an elasticsearch database. This will efficiently respond to search queries with the appropriate search results, and is designed to be as fast as possible.

On the other end of the process, content is published to the site through a variety of Publisher applications. These tend to be government departments that are introducing new content to the site, and this is all done on a relatively ad-hoc basis. All of this content is passed into the Publishing-api, which organises the content in a MongoDB as JSON files.

There is an information exchange between the Publishing-api and the Search-api which updates the search database with any new content - therefore it will show up in search results, and has a direct connection back to the actual content item itself in the MongoDB.

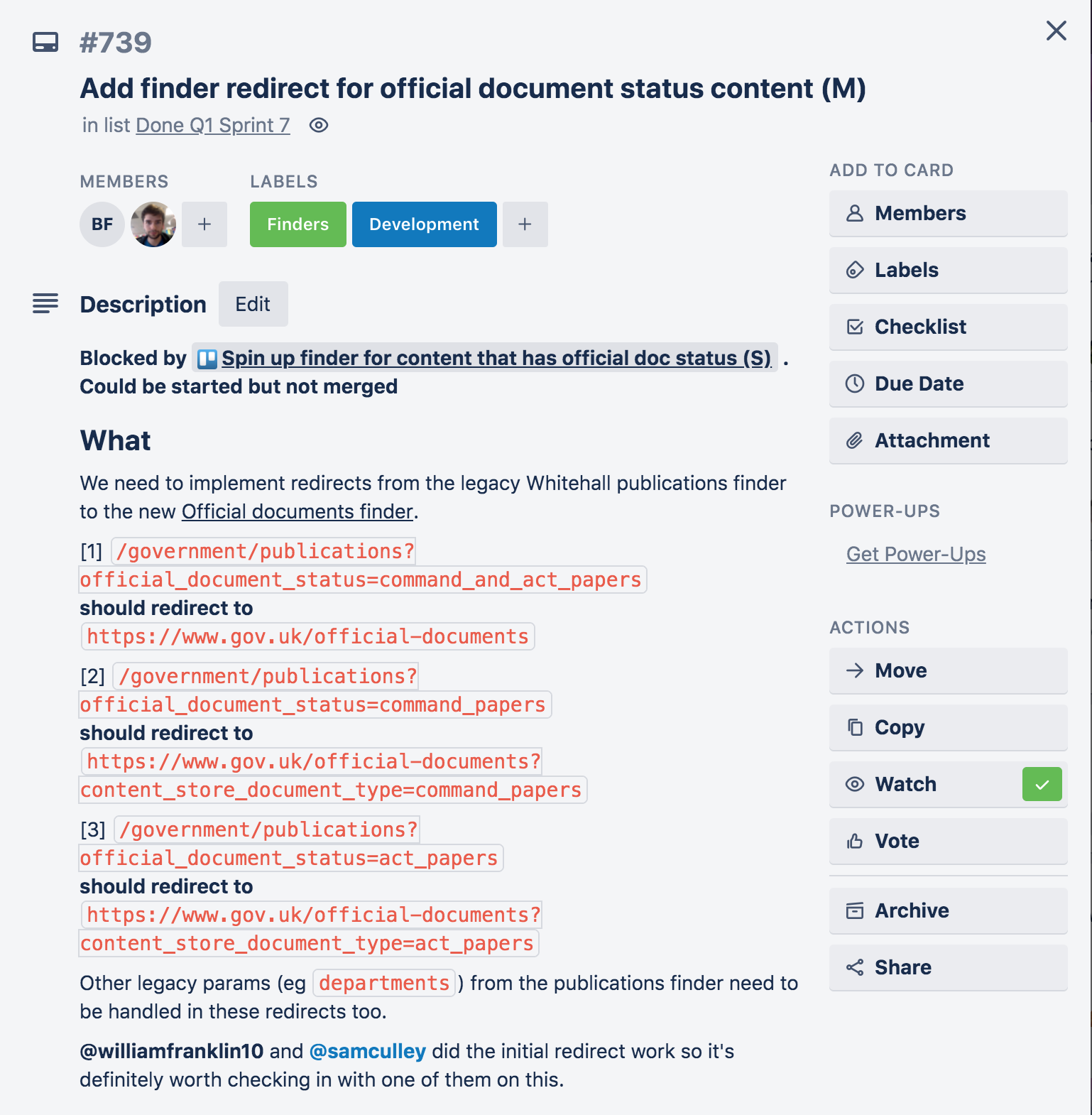
## The Work

My team works in weekly sprints, which involves planning the work for the sprint, assigning tickets to different people in different roles, and then reviewing how things are progressing during daily standups. Tickets can be created by anyone, and I have contributed to some of the work backlog by writing tickets for things that I have found that need to be actioned. An example would be where I was investigating a bug that had been flagged by our 2nd line support team, and once the cause of the bug had been identified, I wrote a ticket that covered the work required to fix it and put it in the backlog.

Tickets follow a template that the team has agreed on, and they’re designed to be as informative as necessary, as well as allowing people to feed into the work simultaneously. The main structure for a ticket involves a description of the work, the reason behind why it’s being done, and acceptance criteria to work out when it has been completed.

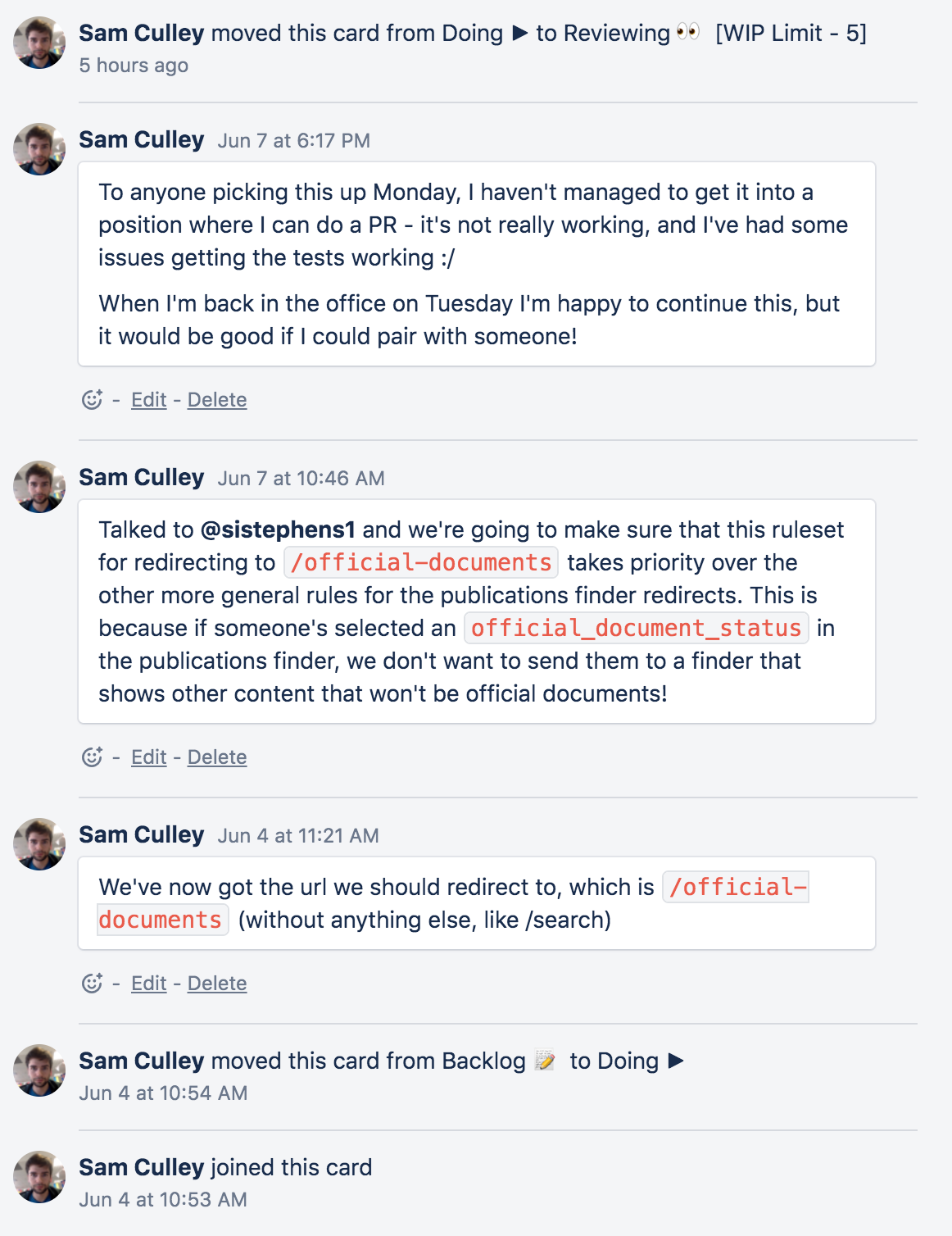
People can leave messages on tickets to add their further comments, and update progress on the work being done. Each ticket is reviewed during the daily standup, and the key person assigned to a ticket will give an update on their progress, and raise any issues they encounter.

I picked up this ticket as part of the work for shifting to the new search tool. This ticket involved setting up redirection rules that ensure users who use links to the old page are routed through to the most appropriate new version. I had to read through the ticket to understand what was required, and who would be the best person to speak to.



For this ticket, it looked like there would need to be some redirects introduced to the old tool, that pointed towards the newer one. There were a series of before and after path suggestions, but I wanted to get some clarification on how this would work. I spoke to the product manager, who had written the ticket, to find out how this should fit in with the other redirects that were already in place. It turned out that these redirects should take priority over any pre-existing ones.

I updated the ticket with the new information, and then moved on to the technical work.



I went into the codebase to learn more about the current setup. There were already some redirects that had been established, and the ones I needed to add had to override those if they were triggered. I contacted the person who had made the most recent changes to this code, and he helped me understand how I would go about making my changes take priority in the logic.

In order to introduce higher priority redirects, I split the work into two parts. Firstly, I added the redirection logic to the general pool of redirect rules. This meant that when provided with the appropriate information, the redirection logic would know where to send the user. The screenshot below shows the code that I added to the hash of PUBLICATIONS\_ROUTES. This is a large object that contains all of the possible redirect rules.



I then needed to make sure that the logic I’d added to the routing would be appropriately prioritised over the pre-existing information. I located the section of the code that extracted the relevant parameter for determining where the redirect was. This line, shown below, attempts to extract a parameter from the original path, which is then checked against the PUBLICATIONS\_ROUTES as a key. I wanted to make sure that the parameters I was creating this for would have the highest priority, so I added an additional parameter check, with an ‘or’ operator, for an ‘official\_document\_status’. If this was found, then it would mean that one of my new redirect rules should be followed, instead of any based on subsequent parameters.

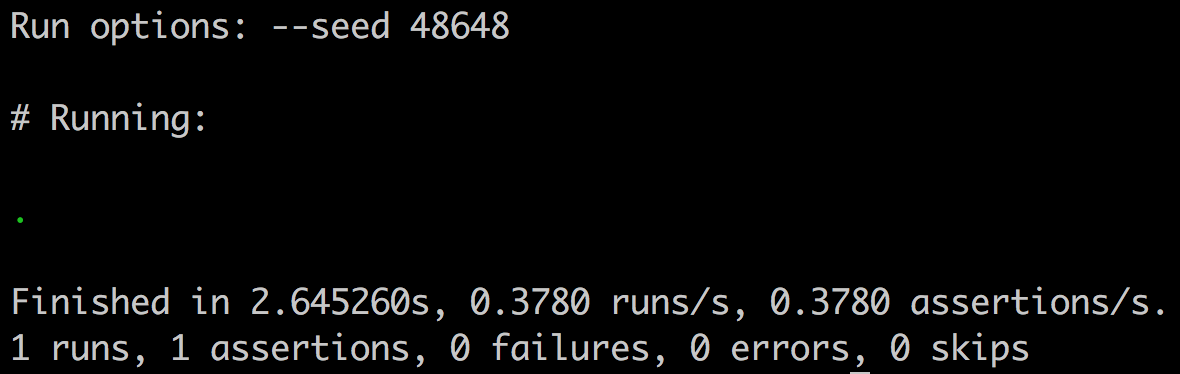


I then created a test to ensure that my code was running appropriately. I first looked through the test suite to see how the original code had been tested. This gave me a good framework when I created my own test for the new functionality I’d introduced. I set up a test that had a descriptive name, and that passed in a parameter that I knew would need to trigger the redirects that I’d introduced. This then checked to see what the response was, which should be a redirect to a specific path. I also wanted to make sure that any other parameters that had been passed in were also handled properly, and that it wasn’t just losing them in the redirect. I decided to use a ‘default\_params’ object that had already been set up in the test suite for this purpose.



In order to run the test, I used the command line to pick out the specific line in the test file where my test was, and only ran that. This let me quickly assess whether the test was functional, without having to run the entire test suite.

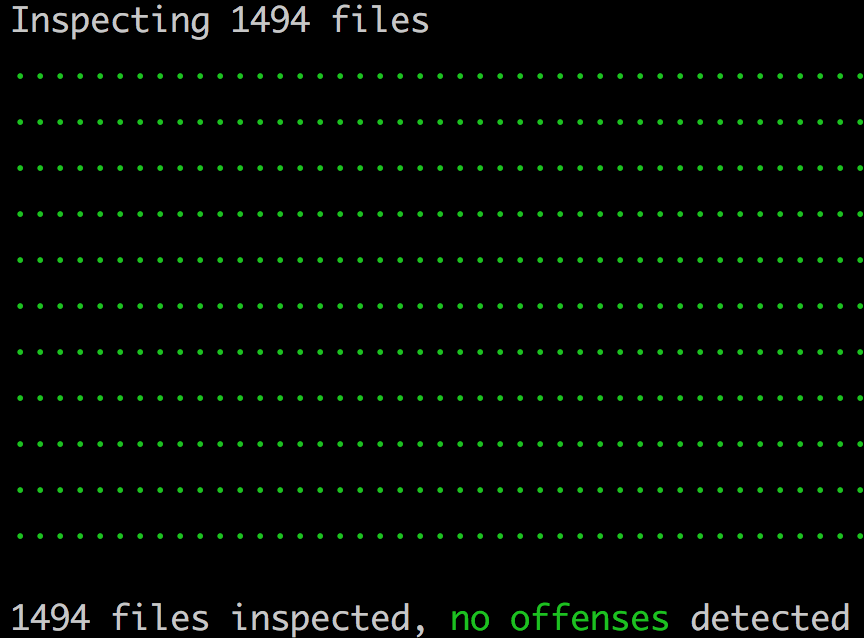




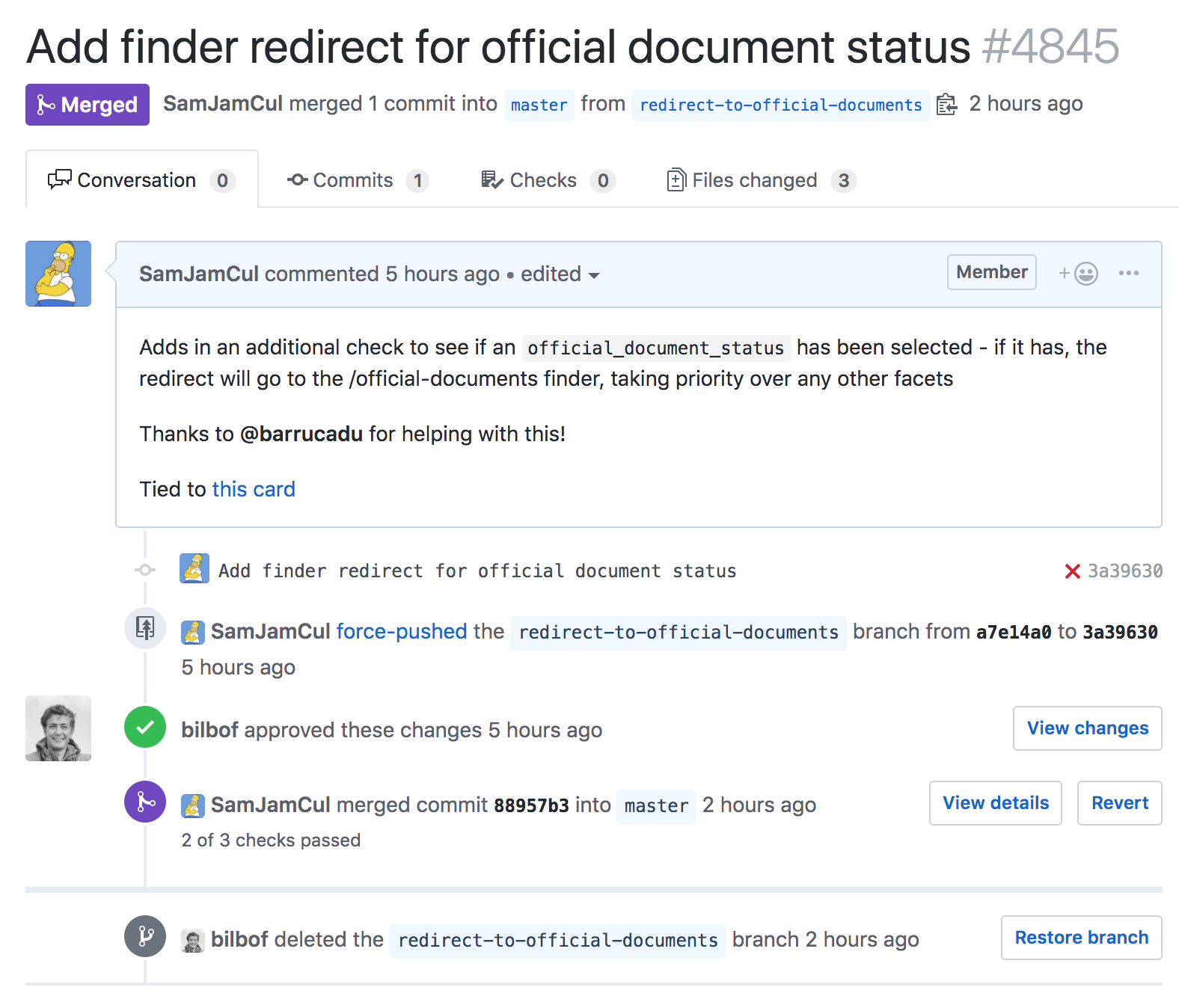
Once I was happy that this test was passing, I also wanted to run the other tests to see if I had inadvertently broken anything. I ran the entire test suite (which took a while), but it didn’t look like any of the tests had stopped passing.

I also ran the linter, to check and see if there had been any formatting errors in the code I’d introduced. I used the line shown below, and corrected the mistakes that turned up. There were two issues that turned up, and both were to do with additional line breaks being used in the test I’d written.





Using Git version control, I committed my changes and pushed them to the origin repository. From here, I made a pull request, which automatically started a continuous integration check. Additionally, I made a post on our team channel requesting reviews for the PR. I received an approval from a team member after they’d checked through the pull request. I also made sure that the continuous integration checks that were run using Jenkins didn’t have any issues.



Finally, once the pull request had been merged, I needed to wrap up the ticket. As part of the ticket’s criteria, I need to speak to a performance analyst to see if there would need to be any additional monitoring for these redirects once it went live. It didn’t look like it, so I added a comment to the card explaining this (shown in the image below, speaking with **@harrietwailling**).



I then checked off each of the requirements, and moved the card into the done column to indicate that I’d finished work on it. My last action was explaining at the next daily standup what had happened, and how the ticket had progressed until it was completed.

## Summary

This ticket was relatively straightforward, and it built on some previous knowledge that I’d had from shadowing others on my team. However, it was a good introduction to working on my own more independently, and I felt that I got to do things in a way that suited me. It was interesting working out how the data that was handled in the URL could be passed around and separated out into key elements, and I feel like I got a good grip on how that worked.

The end result was that the old web page now redirects to a more appropriate new one. Any parameters that were included in the old page request are included in the redirect, so ideally no information is lost. I also got to see first hand how the deployment process fit together, and how other members of the team would chip in at the right times to ensure things moved smoothly.