Refactorisation

Introduction

As with every project or program in general, there is always room for improvement. This document describes a few of the ways in which I have returned to completed, functioning areas of my actual application and refactored them to make them better.

Contents

Transferral of Data Around the Application

Service Worker API

As requested by my project supervisor, after I was finished with version 0.4, I began looking into the Service Worker API so that I could develop a version of the application (or prototype) that uses it instead of the Flask backend. However, it was not long before I encountered issues. The first issue was that the Service Worker (SW) API refused to work with web applications that were not launched via HTTP or HTTPS, so I was unable to just open the HTML file for my test application in my browser. This meant that I would still have to find a way to host my application locally, and, while there are several ways to do this, Flask was the easiest and cleanest way I found of doing this, (which is why I was using it in the first place,) so, immediately, I knew that the SW API would not be able to replace my backend entirely.

The next set of issues I ran into are harder to explain. It would seem that, when running the test application on different browsers, each browser would act slightly differently and throw different errors. I once received an error code from Firefox that implied that the SW would refuse to operate if I was not using HTTPS, after it had previously informed me that HTTP was fine. However, when I tested the application on Google Chrome, no such issue was apparent. The issue also just vanished when I went back to Firefox to test it again, and I was not a fan of the unpredictability. Then, when I did get the SW API to work, the service worker itself wouldn't: It would be registered, it would be active, but it wouldn't actually be doing anything. That was, unless I duplicated the tab that I was running the test application on, and moved to that duplicated tab, then it would work fine.

To top it all off, after experimenting with easier ways of locally hosting a web application and working through the bizarre and unpredictable behaviour of the SW API, I couldn't actually see a way in which it could improve the actual application. The service workers themselves didn't seem to do much, and while there was a lot of indication that they could act as a proxy server and intercept HTTP requests from the frontend, I couldn't get them to do much with the example requests I sent, which showed to me that I would not only need to rehaul the backend of the actual application in order to implement this API, but to also change the way in which the frontend performs requests.

All in all, it seemed like a lot of work would need to be done for an uncertain amount of improvement, and, with the amount I still have yet to add to the application, I decided that I simply did not have the time to invest in getting the SW API to work with my application. That being said, I was willing to make a compromise.

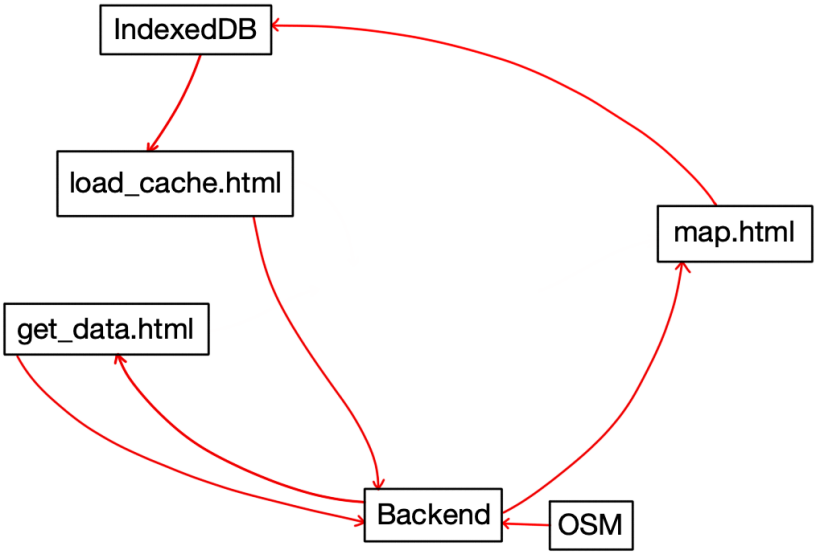
sessionStorage

The main complaint my supervisor had about my implementation was that it was “wonky”, and I do understand what he meant by that. In order to transfer map data from one page of the application directly to another, I was taking that data and storing it in variables in the backend, and then making it inject those variables into the HTML of map.html when rendering the web page. I see now that this is quite an odd way of doing this, but, at the time, I didn’t have enough experience to realise this, whereas I do now. So, I looked into a better solution for this issue.

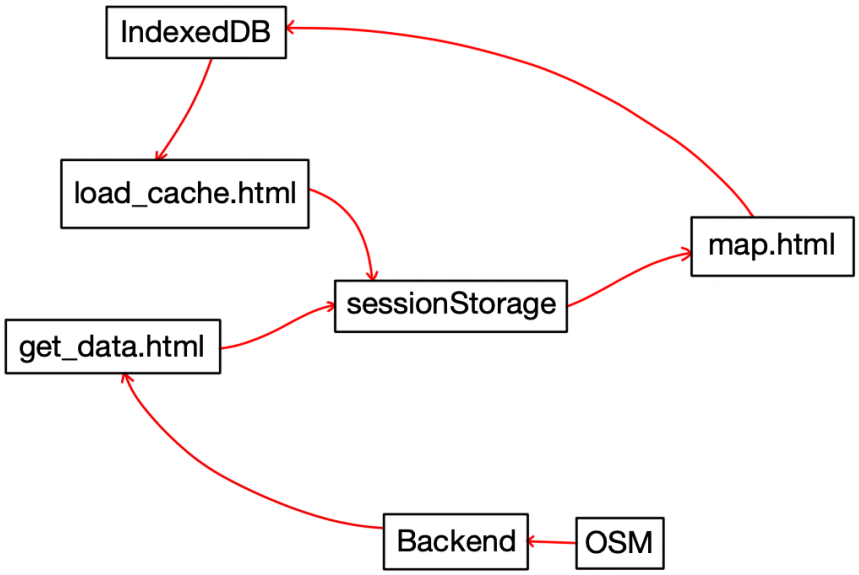
I realised that transferring data directly between web pages via their URLs would still not be acceptable, (due to the great volume of data that needed to be transferred,) so I would still need some kind of middle-man. Ideally, this middle-man would operate on the browser, (like IndexedDB,) and would only store the map data necessary for loading a single map for the duration that that data is needed (unlike IndexedDB). In other words, I was looking for small, temporary, client-side storage that would be easier and quicker for the frontend to access than whatever I was doing with the backend. In essence, I was looking for sessionStorage.

Although sessionStorage does have the annoying limitation of only being able to store strings, (which is not ideal for storing booleans and JSON objects, but can be worked around,) it was perfect for my desired use. I created a test application to get to grips with how it worked, and then easily integrated it into the actual application. It was shockingly easy to use and the performance improvements it caused were immediately apparent. I didn’t have to do any proper tests to realise that the application was noticeably smoother, and, even if the performance increase was just a figment of my imagination, the considerably cleaner state of the code was enough to justify the change (in my opinion).

To demonstrate this, here is a diagram showing the original flow of map data through the application:



And here is a diagram of the current flow of map data:

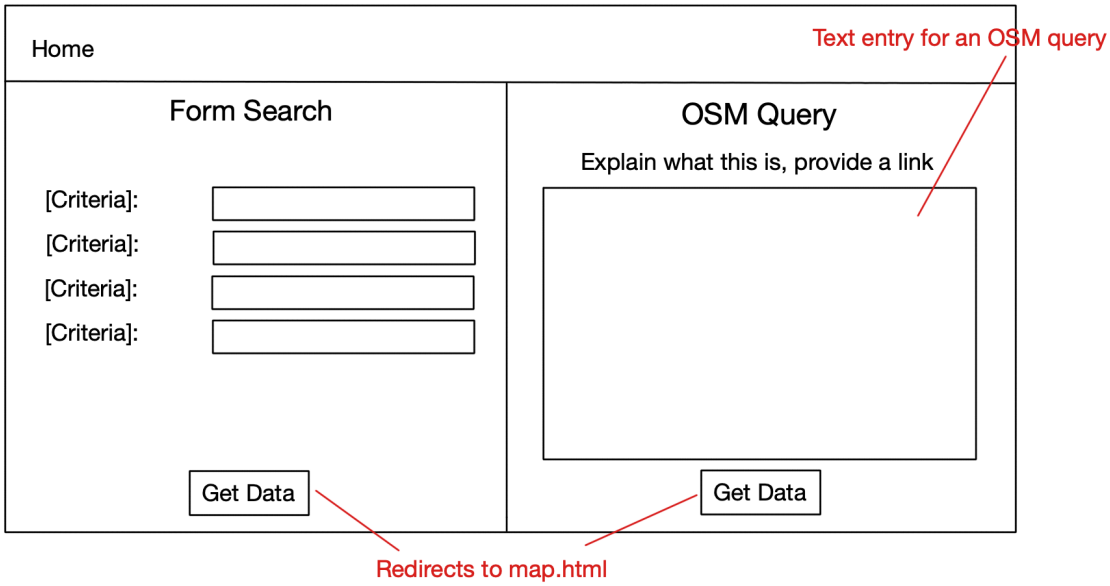


Ideally, the backend would be cut out of this diagram completely so that get_data.html interacts with the OSM database directly, but I do not currently have the skills or knowledge to do this. This is, however, something I can look into near the end of my project if I still have time. Until then, I have other features to work on, such as reworking the user interface to become more accessible and easier to use.

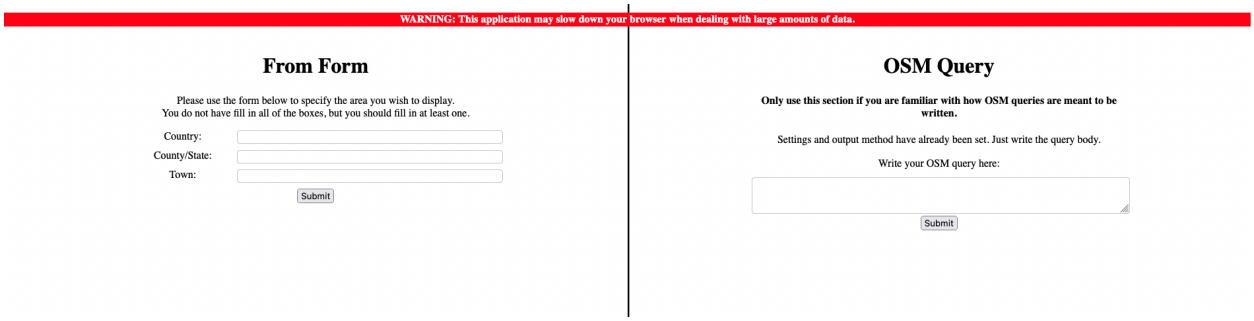
Rehauling the User Interface

The “Get Data” Page

Also referred to as “get_data.html” in some of my other documentation. This is the page that allows a user with an active internet connection to query for and retrieve map data, which can then be rendered into a map on the “Map” page, or “map.html”. It was the second page I created for the actual application, and has been a part of every release since version 0.1. In my earliest designs for this page, I outlined that it should have two main features: The ability to retrieve map data via a form, and the ability to retrieve map data via an Overpass query. Version 0.1 implements the page and its features faithfully to this design, and it remains mostly unchanged until version 0.35.



[My first design for the “Get Data” page.]



[The “Get Data” page as it appears in version 0.1]

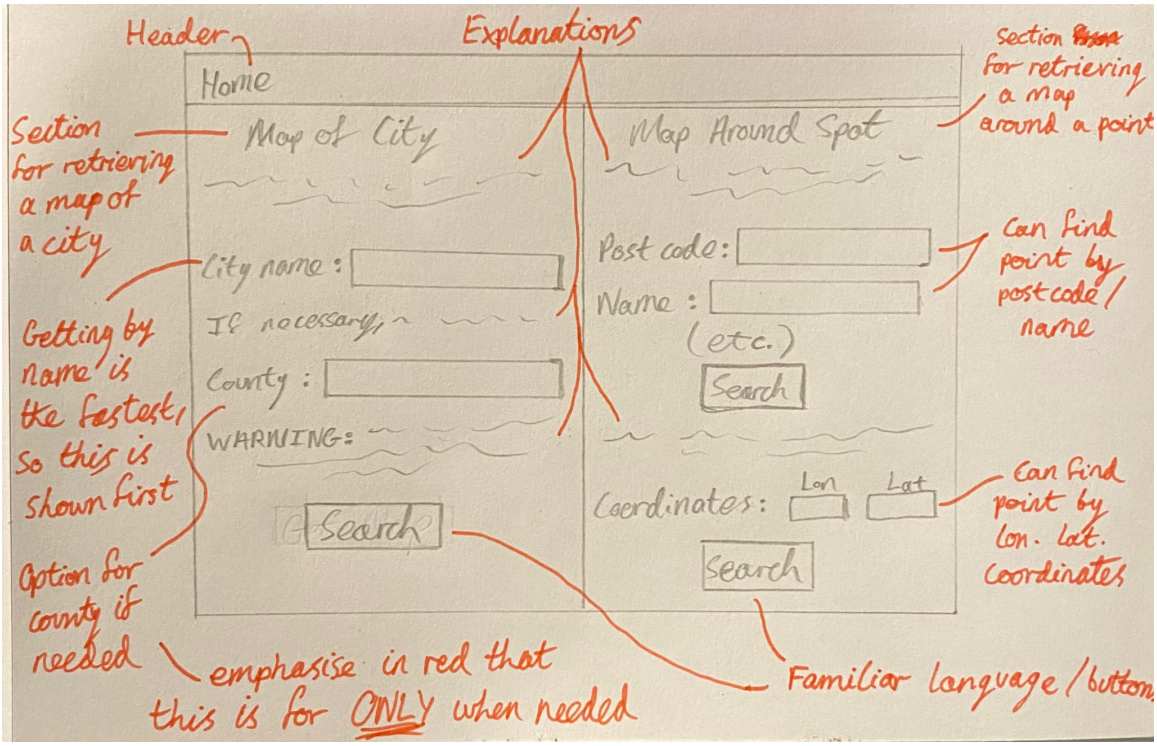
Whilst working on version 0.3, I was on my winter break and staying at home with my family. This meant that I had a ready supply of [something] testers and the opportunity to receive feedback on the current status of my application - as I had planned to overhaul the app's UI in version 0.5, so it would be good to know where the average person thought my application was lacking. As a result of this testing, I was made aware of several areas of improvement throughout my application, including the fact that the language I had used throughout the application was not particularly friendly to a layman. Most of these issues were easy enough to address that I could fix them then and there, but there was one that really stood out against the rest: The blatant inadequacy of the "Get Data" page.

While the page itself was fully functional, only half of it served any use to the average user. Most users would not have even heard of OpenStreetMap, let alone would they know how to write a query to retrieve map data from it using the Overpass Query Language, so the query-based search would be completely useless to the majority of my potential user base - as was the case with my [something] testers. The form-based search also had a glaring issue not in the way that it worked, but in the way that it was laid out. Although I state on the "Get Data" page that filling in each part of the form is unnecessary, some users were still inclined to fill in each field. With each field provided, the time it takes for the Overpass API to perform the query increases, so the user would be left waiting quite a while for data that could have (possibly) been retrieved in seconds.

Another noticeable issue with the page was that it lacked a feature that most digital maps would have: The ability to search for a location by its postcode. I had considered the inclusion of such a feature myself during development, but it was when one of my [something] testers brought the idea up to me independently that I decided that it should indeed be a feature of the application. To summarise:

- The query-based search is useless.
- A postcode-based search would be useful.
- The form for the form-based search needs to be restructured.

From this, I created a new design for the "Get Data" page, in which I addressed each of those points:

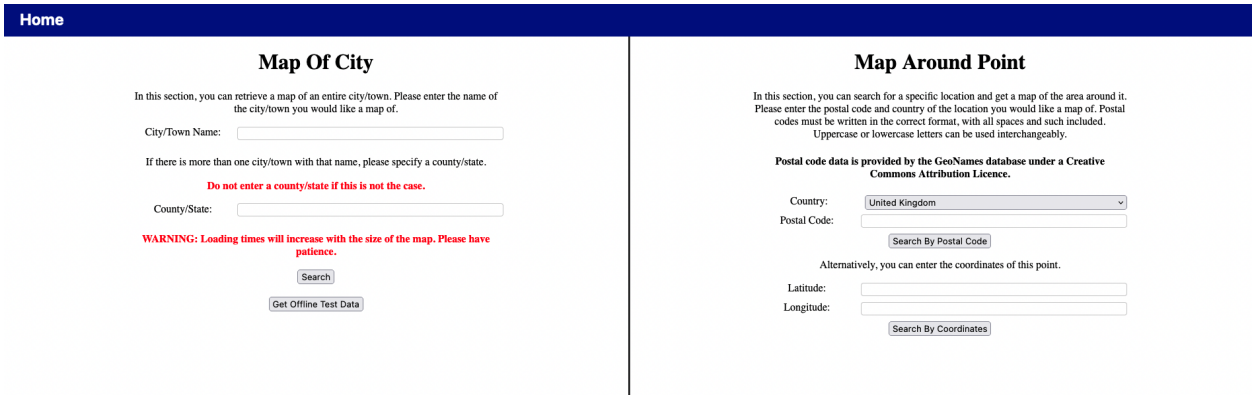


[My second design for the “Get Data” page.]

I also added the ability to search for a location by its geo-coordinates because I felt like that would be a nice addition. I was also too lazy to digitise this design, so I apologise if it is less clear than the first.

Given by how prominent of an issue the old design of this page was during the [something] testing, I decided that the rehauling of this page was not something that could wait until version 0.5, so I planned a new, intermediary version - version 0.35 - dedicated to fixing this issue, and allotted some time to do so.

This is the version of the “Get Data” page released with version 0.35:



[The “Get Data” page as it appears in version 0.35.]

Version 0.5

Whilst the improvements made in version 0.35 were good, they were not intended to fix every issue with the user interface, or even just with the interface of get_data.html. They were mainly focused on adding and enhancing functionality, whereas the usability of the application is just as important, and had yet to be addressed. That is where version 0.5 comes in.

[Add the rest of the report here once you're done with version 0.5.]