Intermediary Versions

Identified Issues

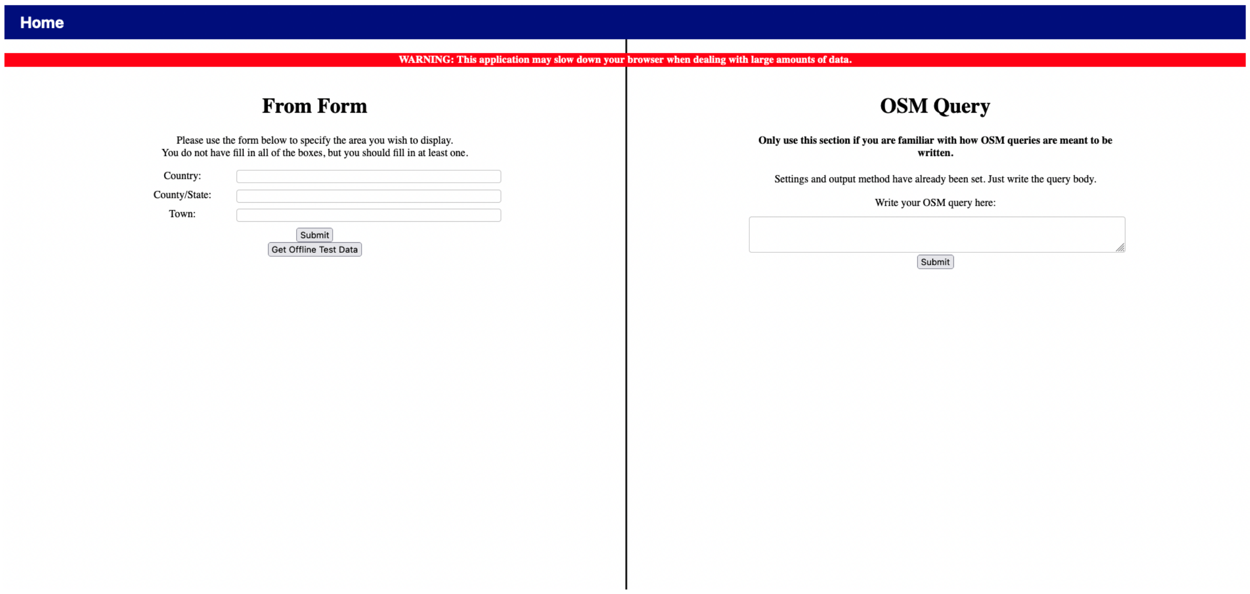
During the development of version 0.3, I asked a number of family members to test my application and offer feedback on its ease of use. Given that the version was all about improving the usability of the map itself, I thought that it was essential to get some outside input before declaring the version as complete - as, if the map I released with the version was unusable, then I would have failed to reach the goal of the version (i.e. its reason for existence in the first place). However, I naturally wanted to take the opportunity to acquire feedback for other areas of the application as well, and I made a lot of great improvements as a result, such as:

- Adding the ability to view information about buildings and amenities.
- Adding a map key.
- Making certain features more obvious. (E.g. adding a note to say that you can zoom and pan the map with your mouse.)
- Making the map zoom feel more consistent.
- Using more layman-friendly language.

However, that was not the end of the suggestions, and it also became abundantly clear throughout testing that there was one area of the application that required serious improvement: The get data page.

Currently, the get data page consists of two sections: One section for getting data “from [a] form” and another for getting data “from [a] query”. The query section requires an understanding of OSM and the Overpass API that the vast majority of users would not have, and it lacks any kind of input validation because checking whether a user’s input could be considered a valid Overpass query is not as straightforward as other kinds of input validation. Because of this, I do not see much use in keeping this section, and it would probably be more trouble than it’s worth to secure, so it would be wise to remove this.

The “from form” section of the get data page also has some issues. First of all, the name does not do a good job at informing the user what it’s for. (I.e. Specifically getting a map of a city/town.) Second of all, and most glaringly, the way in which the form is structured encourages users to fill in all three inputs (country, county, and city) when only one is needed in most cases. This causes the process of data retrieval to be a lot slower - as the more criteria you provide, the longer the search, and this is especially bad when the country is specified - and this is not ideal. So, how do I fix this? Well, I have a number of ideas.



WARNING: This application may slow down your browser when dealing with large amounts of data.

From Form

Please use the form below to specify the area you wish to display.
You do not have fill in all of the boxes, but you should fill in at least one.

Country:
County/State:
Town:

OSM Query

Only use this section if you are familiar with how OSM queries are meant to be written.

Settings and output method have already been set. Just write the query body.

Write your OSM query here:

(Current get data page.)

First of all, I will restructure the “from form” section of the page to have the input box for the city name first, and then the input box for the county afterwards. I will place warnings and guidance in red between these input boxes to make it very obvious that “county” should only be used when necessary (i.e. when two places share a name) and the input box for “country” will be removed entirely - as there should be no location that shares the same city name and county name with another. I will also rename the section to “Map of City” to make it more layman-friendly.

After that, I will replace the “query” section with a section for retrieving a map of the area around a specific point, called “Map around Spot”. This will allow the user to search for a place/point using its postcode (and perhaps other attributes) and generate a map based on the area surrounding that point. It will also have a subsection that allows the user to do the same for a set of longitude and latitude coordinates. I’m not certain what the best radius around the point to use would be, but I can deduce that via testing. This feature was something I’d thought of and considered during the development of the application, but it was when my father suggested the same idea to me that I realised that I should probably implement it. This was not the only suggestion he had for that page, however, and I found his second suggestion to be particularly interesting because it was something that hadn’t crossed my mind at all.

What3Words API

During his feedback, my father noted that a service known as “What3Words” had become quite popular these days, and that certain individuals favoured using it over postcodes and other traditional methods of identification. Thus, he suggested that it could be good to use alongside an option to search for locations via postcode, and I agreed. Not only would it probably get me more marks in the “Technical Expertise” section of the mark scheme, but it would also broaden my user base and help to modernise the application as a whole. So, I’ve decided to look into its API to see if it would be a suitable (and doable) addition.

The What3Words API is free-to-use for up to 1000 “convert-to-coordinates” requests per month[1]. That works out to a maximum of about 33 requests per day per API key. For a fully-fledged application (that would have more than one user), this would not be sufficient. At the moment, I can think of two ways of circumventing this:

1. Treat the application in its current state as a sort of “proof of concept”, with the idea that an actual release of it would use a paid API key.
2. Ask the user to provide their own API key in order to be able to access the What3Words functionality.

There are potential issues with both of these approaches. The first would create a financial barrier to actually being able to release the application and, even if I did not intend to ever do so, this just feels wrong - as one of the main reasons for using OSM is to promote free, accessible data for all, as opposed to monetizing information about the world around us. I also don’t intend to ever pay a subscription fee and wouldn’t expect anyone else to just to access my work.

The second approach’s main issue is that, by asking the user to provide their own What3Words API key, I am, in a way, endorsing What3Words as a company and the way in which it uses its users’ data - as I am asking my users to give W3W their personal data in order to use a function of my application. For the sake of integrity, I would have to do a thorough investigation into their terms of service and any allegations against them of improper data-handling before making any attempt to use their services in my application.

Service Worker API

Aside from my family, I have also received feedback on my application from my project supervisor. During my second meeting with him, he made me aware of a technology called the Service Worker API and suggested that I use it as an alternative to my current Flask backend.

The Service Worker API[2] is a type of web worker (essentially, a program that runs alongside your application) that is used in both online and offline applications to manage requests and process them based on the status of the application, network and web server. It is a proxy that sits between the frontend and the backend, and thus does not replace the backend, but instead tries to improve how the two interact with each other. This means that they won't be able to replace the Flask backend in its entirety, but they could speed up the rate at which data is transferred around the application.

In other words, the service worker could take the responsibility of transferring data from load_cache.html and get_data.html to map.html, and the Flask backend just has to organise and fetch resources. I, however, have no experience with using this technology, so I would like to experiment with it before declaring that it would be suitable to use in my application. I could do this by creating a series of test applications, including those such as:

- Using the Service Worker API to perform any kind of action.
- Using the SW API with a Flask backend.
- Using the SW API to transfer data from one page to another.
- Using the SW API to transfer data from IndexedDB between pages.

If the development of all of these applications goes well, and if each application functions to the same standard as a non-SW application, (or better,) then I would consider using the API in my application.

The Intermediary Versions

From what I have discussed above, you can see that there are a lot of potential changes and improvements that could be made to my application that were not outlined in the initial scope or the subsequent “Version Planning” document. Ideally, I would want to avoid “feature creep”, and not interfere with the deadlines of the versions that have already been planned, but I also want to strive to make a good, usable end product. So, I will be separating the desired additions into their own intermediary versions, based on how much effort/time I think they will take and the importance of the version they come before. I have not assigned deadlines for these versions because it is quite hard to judge how long each will take.

Version Number	Main Goal	Success Criteria
0.35	To rework get_data.html to be (generally) faster to use and have the ability to generate maps from the area surrounding a given point.	● Query search has been replaced with the ability to search for a node/single location. ● Form search has been renamed. ● “Country” has been removed from form search. ● From search has been restructured to emphasise that only the city name is needed in most instances.
0.45	Evaluates external APIs to see if they would make suitable additions to the application, and weaves these APIs into the application if so.	● Evaluates using the What3Words API. 　○ OPTIONAL: Adds an option to get_data.html to search for a point using W3W’s 3 words. ● Evaluates the Service Worker API. 　○ OPTIONAL: Uses the Service Worker API to control the flow of data around the application (instead of the Flask backend doing this).

The changes to get_data.html are somewhat urgent - as its issues hinder user testing - and shouldn’t be particularly difficult to implement, so I think they should be done before anything else. The implementation of those APIs, however, will probably take quite a bit of work, and would thus delay the version of the application they come before. Version 0.4 adds an important bit of functionality, so I feel that it should take priority here, whereas version 0.5 is all about improving the UI, which is less important - as the UI isn’t that terrible (judging from the results of the user testing). So, I think I should leave all experimentation until after version 0.4 has been completed.

References

1. https://what3words.com/select-plan?referrer=%2Fpublic-api&currency=GBP
2. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API