Version Planning

Developments in the Project

Since the creation of the Project Plan, many developments and discoveries have been made in the project. Initially, I thought it would be best to use Java for the application's backend, along with a number of Maven plugins that would allow the backend to both serve as a mini web server, and interact with Overpass API to provide the frontend with OSM data. However, when I looked into this further, I found that Python was more suited for the job.

Python not only has the ability to host websites using no external libraries at all, but it also has a wide range of web development and web hosting modules made for it, such as Django and Flask. These modules are used in the industry, and having knowledge and experience with them is vital for some job positions. They are also a lot easier (in my opinion) to set up and learn than their Java equivalents.

Because of this, the project now uses a pure Python backend with only two external dependencies. These are:

- The "requests" library, which is used to get and return data from the Overpass API.
- The "Flask" library, which will allow the backend to act as a web server, and to control the flow of data throughout the application.

The "requests" module allows you to send HTTP requests from a Python program, and has very simple syntax. It is used in the application to send GET requests to one of Overpass API's many endpoints, and return the response it gives. One may argue that this is a redundant feature for the backend to have - as the JavaScript frontend is perfectly capable of sending HTTP requests using jQuery. However, in order to pass data from one web page to another using JS, one must place it into the URL of the web page. With the large amounts of (OSM) data being circulated in the application, this would result in incredibly long URLs that would be very easy for the user to modify. This would be neither visually-appealing nor secure, but, fortunately, Flask has a better way of transferring data to a web page, in the form of "templates".

Flask templates allow the backend to pass data into a web page while it is serving the page to the user. It does this by putting placeholders in the HTML file where the data would be stored, (i.e. in a JavaScript variable,) and then replacing those placeholders with given values when rendering the page. Here is an example of placeholders being used in the "head" tag of map.html:

```
<script src="{{url_for('static', filename='js/jquery-3.6.1.min.js')}}"></script>
<script>
  // The name of the data store to use when loading stored data
  var data_store = {{ data_store|tojson }};
  // Arrays containing the OSM data to render, whether the data had been stored or not
  var nodes = {{ node_array|tojson }};
  var ways = {{ way_array|tojson }};
  var rels = {{ rel_array|tojson }};
</script>
<script src="{{url_for('static', filename='js/map.js')}}"></script>
```

and the corresponding code from actual_app.py that fills in these placeholders:

```
# - Variables and Constants -
# Url of the Overpass API endpoint to be used
OVERPASS_URL = "http://overpass-api.de/api/interpreter"
# Three arrays for storing the OSM data that will be displayed by map.html
nodes, ways, rels = [], [], []
# Holds the name of the local data store that the OSM data is stored in
data_store = ""

# Route to the map page
@app.route("/map", methods = ["GET"])
def map():
    # If the request is a GET request,
    if request.method == "GET":
        # Use the values of nodes, ways, and rels from the global space
        global nodes, ways, rels
        # Renders the map.html web page and provides it with formatted OSM data
        return render_template("map.html", data_store=data_store, node_array=nodes, way_array=ways, rel_array=rels)
```

As you can see, these placeholders can also be used to provide a relative path to static files, so the page can make use of JS files and CSS stylesheets without having to store all these files together. This is another advantage of using Flask - as it allows one to organise each type of file into its own directory, making everything easier to find and more aesthetically-pleasing.

A change in language for the backend is not the only change that has been made from the initial project plan. The order in which tasks have been completed and milestones have been met has not lined up with the timeline that was outlined in the plan. While it is still useful to provide a rough idea of what to work on next, I found that a lot of the goals were being achieved a lot earlier than they were supposed to be. Thus, I have decided to draft up a new “version plan” that will detail the next steps in the project and their deadlines.

The New Plan

Version Number	Main Goal	Success Criteria	Rough Deadline
0.1	This version focuses on basic functionality. It will allow a user to retrieve map data via some inputs, and display that map data to the user in basic form.	● Map data can be displayed. ● Can get map data from query input. ● Can get map data from form input.	23/10/22
0.2	This version shall give offline functionality to the app, by allowing the user to store data locally on their browser.	● Map data can be stored. ● Stored map data can be deleted. ● A map can be rendered using stored map data.	13/11/22
0.3	This version aims to improve the rendering of the map in order to produce a more useful map.	● Map can be panned. ● Map is colour-coded. ● Map is labelled to show element names. ● Relations are represented in some way.	04/12/22
0.4	This version enters the “extensions” territory. It will allow users to highlight certain features of the map based on their inputs.	● Can highlight elements by their name. ● Can highlight elements by their tags.	29/01/22
0.5	This version is solely concerned with making the application more visually/aesthetically pleasing. This can include other design improvements for more efficient HCI as well.	● Application looks better.	19/02/22
0.6	This version also falls under the “extensions” territory. Allow the application to handle not just JSON data from queries, but all kinds of different formats that OSM queries can return.	● Can use XML OSM data. ● Can use CVS OSM data.	12/03/22

This plan uses the fact that it took roughly 3 weeks to complete Version 0.1 as a basis for each successive deadline. It does not allocate time within the Christmas holidays for working on the project (unlike the previous plan). Non-programming deliverables and optimisation efforts are not included in this plan, as they will be worked on throughout. The extensions are still optional, but are highly desirable.