Access To Recreational Quality (ARQI) Website Documentation

Website URL: https://www.geos.ed.ac.uk/dev/ARQI

Project GitHub Repository: https://github.com/attilacodes/Capital-Greenspaces-Project

In this documentation, decisions around the design, the functionality of the webpages/web map and further bespoke resources will be discussed and provided. The document will be split as follows:

1. Flask
2. Homepage
3. Interactive Web Map
4. Query and Download
5. About Us

Flask

The website was developed using Flask, a Python-based micro-web framework. This was used due to its capability to provide structure to our rather small-scale webapp. The code is organised into a main folder and sub-directories as follows:

```
CapitalGreenSpaces
├── main.py
├── recreation
│    │    ├── __init__.py
│    │    ├── query.py
│    │    ├── views.py
│    │    └── webmap.py
│    ├── static
│    │    ├── .css
│    │    ├── .jpg/.png
│    │    ├── .js
│    │    ├── .geojson
│    │    ├── .csv
│    │    ├── .ico
│    │    └── .pdf
│    └── templates
│         ├── base.html
│         ├── homepage_ARQI2.html
│         ├── query_download.html
│         ├── about_us.html
│         └── webmap.html
├── README.md
└── Group_6_Documentation.pdf
```

`CapitalGreenSpaces` acts as the main folder storing all files and templates used to create the website. Within it, `main.py` imports all the app routes and instantiates the webapp. This can be run from the terminal when we want to start the website via `Python3 main.py`. In the development server, this returns a URL in development mode.

When deploying the website, we used *Gunicorn* – a Python HTTP server for WSGI applications – to serve the webapp. Gunicorn was used due to its ability to handle multiple requests from the webapp in place of the 'one-at-a-time' model as seen in the development server, and its relatively stable and easy-to-setup method.

`recreation` acts as the immediate sub-directory storing all the Python files used to create the website.

`__init__.py` makes the directory **`recreation`** a Python module package and stores a function that registers all extensions to the webapp via *Blueprint*.

`views.py` stores all of the main routes or the URL end points for the front-end aspects of the website, where users can navigate between pages e.g. The Homepage and *About Us.* The *Blueprints* of these are then imported and registered in the `__init__.py` file.

`query.py` stores the code used for the query and download application page.

`webmap.py` stores the code created for the interactive map.

The **`static`** folder contains all 'decorative' files such as CSS, JS and PNG/JPG files as well as datasets used for the web map.

In **`templates`** are files that contain all HTML files as well as the base template (`base.html`) to which all elements contained within extend to the other HTML templates. Flask uses the Jinja2 template library to render templates dynamically. This ensures every webpage in the application will have the same basic layout. Special delimiters are used to differentiate between Jinja2 syntax and the static objects in the template. Code between `{{ … }}` calls the variable in the Python files to be rendered in the HTML files. Meanwhile, `{% … %}` indicates a control flow statement. Where a call to a specific function in the Python files or CSS files are required, this is achieved using `{{ url_for () }}`.

Key Resources:

https://flask.palletsprojects.com/en/2.0.x/

https://docs.gunicorn.org/en/stable/

Homepage

This webpage showcases key information about the project including the topic itself, justification, and methodology. Both CSS and JavaScript were used to control web design and its interactive elements, the latter of which was particularly important here as the webpage was largely 'factual'. For instance, the interactive carousel provides a more visual and tactile engaging way of presenting text and images. The CSS file (`webpage_style.css`) comprised of a mixture of self-generated elements as well as inheritance from the W3.CSS template, and was chosen due to its responsive design. Below, I highlight further resources that point to bespoke plugins used:

The text-image carousel was created using Slick developed by Ken Wheeler which uses a jQuery plugin to power the responsive carousel. Further JavaScript was added to control the cycle speed in line with the client's browsing pace.

http://kenwheeler.github.io/slick/

To allow for a more succinct and visually pleasing way of displaying survey data, the AnyChart API was utilised to provide cross-browser HTML5/JavaScript charting. The data was first transformed into JSON format and hard-coded into the HTML file before JavaScript was employed to create and customise the chart.

https://docs.anychart.com/Quick_Start/Quick_Start

Interactive Web Map

The interactive web map uses a Python wrapper library called `Folium` to visualize spatial data on a Leaflet map. This was used due to our familiarity with the programming language and its well-written and relatively up-to-date documentation.

We used four built-in tile sets from OpenStreetMap, Stamen and CartoDB, giving the user a range of choices for a base map.

Several CSV and GEOJSON files are used as static data for the map layers. These are imported using the `os` module and read in with the `pandas` library; the latter also provided means of manipulating the data. Custom legends are created with `branca.colormap`, a utility module for dealing with colour maps.

In order to reduce client efforts e.g. through excessive clicking to retrieve information, the map layers are coded to enable a "hovering" effect, thereby showing key information once the cursor overlays a polygon. The user has several choices for visualizing the data including the ability to add, combine and minimise map layers depending on their needs.

To enhance user interactivity, the web map uses additional plugins to deliver the distance/area measure widget using `MeasureControl`; zoomable density maps using `MarkerCluster`; and finally, a heatmap using `Heatmap`.

Collectively, the web map comprises of the following layers:

1. ARQI choropleth map – Edinburgh layer displays the ARQI for each Datazone.
2. Greenspaces choropleth map – Edinburgh greenspaces layer displays only those used in our study.
3. Marker Cluster map – Edinburgh Greenspaces layer shows access points for each greenspace in the study.
4. Heatmap – Same layer as the access points, only displays the clustered access points.
5. SIMD – Scottish Index of Multiple Deprivation layer allows to compare results with our ARQI.

The web map is initially saved as a HTML file and stored one of the member's `public_html` folder on Linux. In the webpage itself, the HTML map file was added through an *IFrame*. Overall, the webpage is designed using the same CSS file as the homepage to maintain consistency.

Key Resources:

https://python-visualization.github.io/folium/

Interactive maps (autogis-site.readthedocs.io)

Query and Download

Python scripts were written to extend query functionality from the server-side to the HTML. In `query.py`, two functions were defined. The first establishes a connection to the DB using *cx_Oracle*, and a SQL query written to assist with the retrieval of all (recorded) greenspace names in alphabetical order. This therefore allowed the drop-down menu items rendered in the `query_download.html` file to be populated dynamically using Jinja2 templating. The second drop-down menu containing all three services were generated manually as these were not recorded in the DB and did not require a great deal of effort to hard code in the HTML file.

The HTML forms (drop-down menus) used the `POST` method to send data to the server side, and the `action` attribute (of the form) was defined to return retrieved data to another URL (*/submitted*). When inputs are submitted, the first element (in this case, first word of the greenspace name) is submitted (e.g. Inverleith, and not Inverleith Park). This triggers the second function in the Python script which uses an SQL bind variable to retrieve the correct data from the DB tables. Whilst the adage of 'never trust the user's input' is applicable here, the use of 'sanitised' data in the HTML forms avoids SQL injection attacks in general use-cases.

Finally, table headers are rendered onto HTML alongside the retrieved rows of data (in tuple format) using *flask.MarkUp*. Note: this only appears to work on Safari and Firefox.

Additionally, data download is offered as well to increase functionality for a range of users. The overall webpage is again designed using the same CSS file as the homepage to maintain consistency.

Key resources:

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

https://cx-oracle.readthedocs.io/en/latest/user_guide/bind.html

https://flask.palletsprojects.com/en/2.0.x/templating/

https://tedboy.github.io/flask/generated/generated/flask.Markup.html

About Us

This webpage seeks to bridge together the 'internal' environment – both the website and the team behind it – with the 'external' through the provision of additional information beyond the website. CSS was primarily used to control the design of the webpage. Alongside the W3.CSS template, another, different CSS file (`about_us.css`) was used to the homepage here due to the simultaneous web-design efforts from two members at the time of its development. The latter CSS elements were self-generated, albeit the overall look of the webpage is rendered the same as the other webpages. To aid visual exploration and communication, a Google map insert of the team's office location is embedded onto the webpage. Furthermore, a responsive HTML email form is included to extend efficient communication, and this is achieved using *FormSubmit*, a free form backend that handles HTML submissions. Additional security features include email address validation and reCAPTCHA (powered by Google) upon submit. FormSubmit was used due to its simplicity (does not require setting up a SMTP server and the use of PHP) and integration with HTML5.

Key Resource:

https://formsubmit.co/

References

AnyChart, 2022. *AnyChart Documentation*. Available at: https://docs.anychart.com/Quick_Start/Quick_Start. [Accessed: 10/12/2021].

Chesneau, B., 2021. *Gunicorn – WSGI Server*. Available at: https://docs.gunicorn.org/en/stable/. [Accessed: 07/01/2022].

DevroLabs, 2022. *FormSubmit*. Available from: https://formsubmit.co/. [Accessed: 15/01/2022].

Grinberg, M., 2017. *The Flask Mega-Tutorial Part I: Hello, World!.* Available at: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world. [Accessed: 12/12/2021].

Pallets, 2010. *Templates*. Available at: https://flask.palletsprojects.com/en/2.0.x/templating/. [Accessed: 12/12/2021].

Story, R., 2013. *Folium*. Available at: https://python-visualization.github.io/folium/. [Accessed: 21/12/2021].

tedboy, 2016. *flask.markUp Documentation*. Available at: https://tedboy.github.io/flask/generated/generated/flask.Markup.html. [Accessed: 23/12/2021].

Tenkanen et al., 2021. *Interactive Maps*. University of Helskinki. Available at: https://autogis-site.readthedocs.io/en/latest/notebooks/L5/02_interactive-map-folium.html. [Accessed: 22/12/2021].

Tuininga, A., 2016. *Using Bind Variables – cx_Oracle 8.3.0 Documentation*. Available at: https://cx-oracle.readthedocs.io/en/latest/user_guide/bind.html. [Accessed: 15/12/2021].

Wheeler, K., n.d. *Slick Documentation*. Available at: http://kenwheeler.github.io/slick/. [Accessed: 27/12/2021].