Flask Application Structure

Introduction

This website is built using the Flask 1.0.2. Flask is a Python web microframework which makes it relatively easy to build basic websites, but also offers extended features that enable development of more sophisticated sites. The documentation for the Flask python package library can be found at the Flask website.

From the outset, the project team decided that all the functional code within the application should be written in Python 3. Python was chosen as a widely used, powerful and relatively easy to read, programming language. In addition there are many packages available for Python and they are well supported.

Python version 3.6+ is necessary for functionality due to some of the functions used within the code having changed syntax from earlier versions of Python 3. Python 3 was used as Python 2 is being depreciated and will not be maintained past January 1st 2020.

Flask uses Jinga2 and the Werkzeug WSGI toolkit for web functionality (template rendering).

Additional Python Packages used within the webapp

`flask-WTF` and `WTForms` were used to create web forms for the search function and for the user data upload function. These packages give extra functionality in terms of validating form field and providing ready-formatted web display objects (form fields and buttons) specifically for Flask applications.

`flask-table` was used to create tables of browse and search results for display. This package creates table Class objects with user-definable fields including link-columns and passes html table class items for web display.

Webpage styles are based on `html5` with `Bootstrap CSS 4.2.1` with local tweaks in the `main.css` file, `jquery` is used for tabs on results page.

Further information on the following packages and how they are used in this application can be found in the accompanying documentation files.

- `SQLAlchemy` - used to create and populate the database.
- `pandas`, `statsmodels` and `numpy` - used to parse datasets, facilitate data import and for user data analysis.

- `matplotlib`, `wordcloud` and `plotly` - used to visualise analysed data

The Python modules' versions employed in this project are specified in the project's `README.md` file. They can easily be installed using `pip install -r requirements.txt` files.

Application structure

The PhosQuest application requires an SQLite3 database (`PhosQuest.db`) situated in the folder level outside of the `PhosQuest_app` folder. Also in this location, `application.py` is a script which is run to activates the Flask application `PhosQuest_app` . `python3 application.py` or a similar command should activate the application (when all requirements are installed).

App Folder structure

The structure of the software consists of a `PhosQuest_App` folder.This folder contains the `__init__.py` file that initiates the instance of the PhosQuest Flask application.

The `PhosQuest_App` folder contains subfolders corresponding to *Flask Blueprints* for routes `main`, `search`, `browse`, and `crunch`. Blueprints enable developers to simplify large applications and separate the code. For information about Flask Blueprints see link.

- `main` - contains `routes` for main pages on the website, *i.e.* the homepage, documentation pages and about us page.
- `search` - contains `routes` for search page and a `forms` script which initiates a form class object for display on search page
- `browse` - contains `routes` for browse pages
- `crunch` -contains `routes` for userdata analysis pages and a `forms` script which initiates a form class object for display on the upload page

Additional folders within the `PhosQuest_App folder are:

`static` - contains images used on the webpages and the following subfolders:

- Subfolder `styles`, containing the `main.css` local css file
- Subfolder `userdata_temp` for temporary storage of user data and output files.

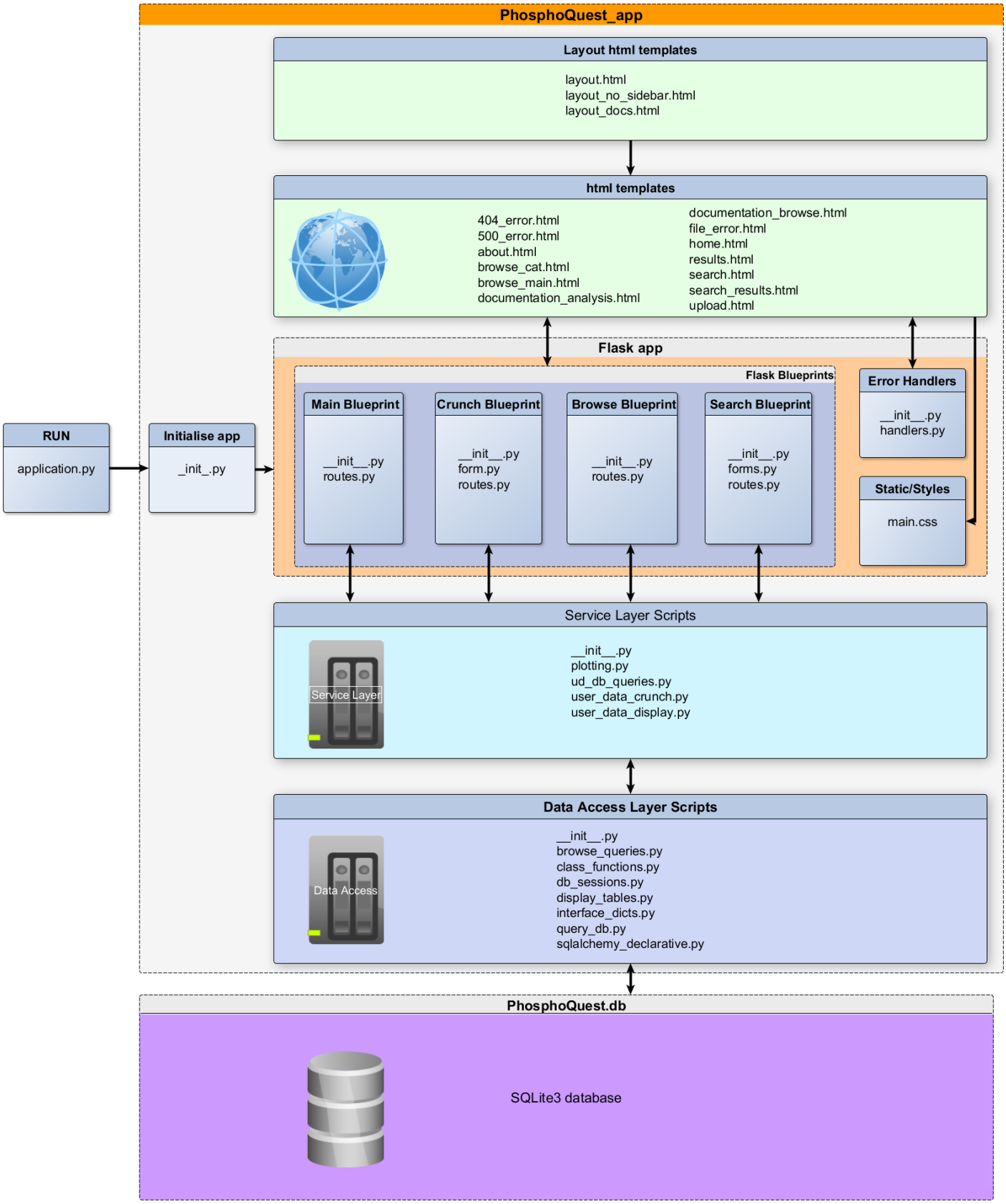
`templates` - contains all the website html template files.

`errors` - contains error handlers

`service_scripts` - containing scripts used in user data analysis

`data_access` - containing scripts for querying the PhosQuest database.

An overview diagram of the PhosQuest app structure is shown below.



Script interactions

The interactions between the various scripts and the methods used in the PhosQuest_app is shown in the detailed documentation.

- Browse and Search documentation
- User data analysis documentation
- Result plotting documentation
- Database documentation

Application limitations and potential issues

- Search function is basic and does not recognise logical functions or wild-cards in the text field.
- Text to say that your analysis is running displays on click of the button when no file has been chosen. An error message also appears, but this could be confusing.
- Data analysis is all done at once on successful upload of a file, and can take a minute or more to complete depending on input file and resources available.
- Due to the large amount of data and images rendered on the user-data results page it can take a few seconds after the page initially appears for the styles to render fully (eg. for the jquery tabs to show correctly).
- Once the results page has appeared, users cannot return to the result data if they navigate away from the page as all the variables are lost.
- Errors with data file upload can display in two ways depending on the error type, some are caught by the `user_data_check` function and display as flash messages on the upload page, and others cause an error which is caught by the `try-except` method and give the `file-error.html` page, the information on the page should, however, help to explain what the problem could be.
- Issues occasionally occur with the display of the html interactive pie-charts on the user-data results page. We do not currently know the reason for this, but re-running the analysis usually fixes the display issue