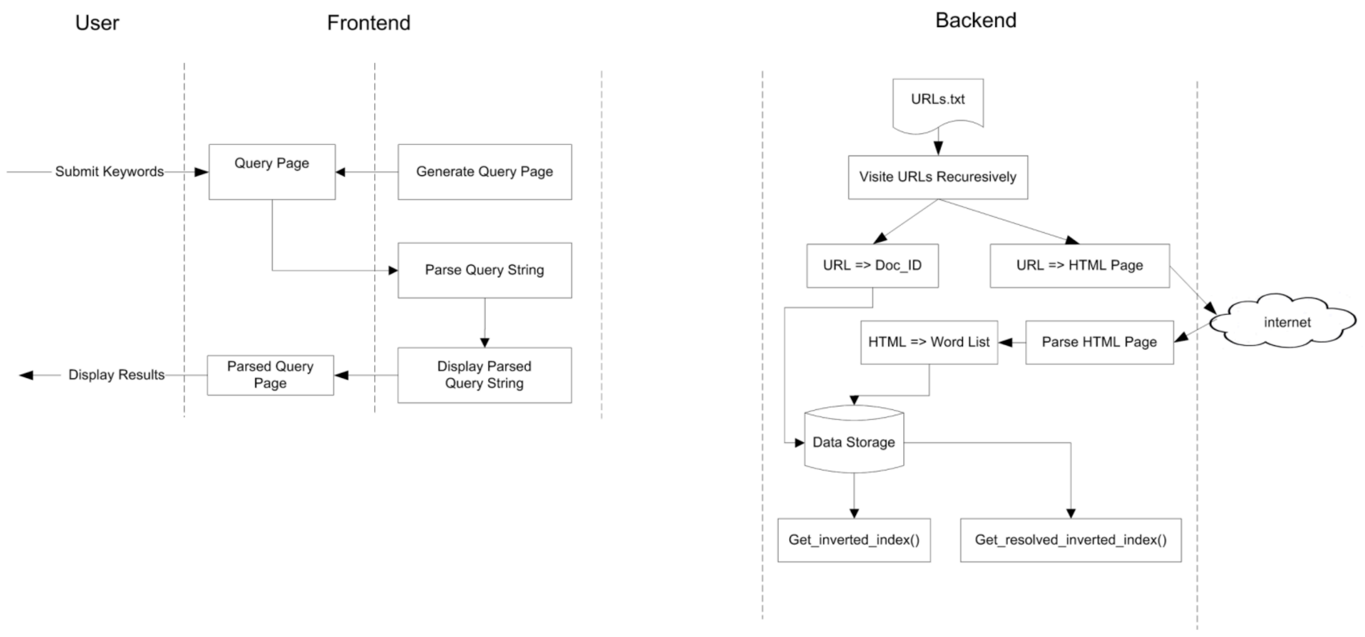
Lab 1 - Development Phase 1

In this project, you will be building a web search engine that resembles the Google search engine. The project will be decomposed into four phases. This is the first phase of the project.

Below is a reference architecture of the web search engine with a frontend component and a backend component. The frontend is in charge of serving the search request; while the backend is in charge of indexing a subset of the internet, such that the search requests can be served quickly, and more importantly, with relevant search results.



Example of search engine architecture for Lab 1

Frontend

F1. Simple Query Interface

In this lab, you will implement a simple query interface that asks for a query string (can be a single keyword or more) from the web users. Your interface should include at least the following components:

- an input box for typing a keyword string
- a button that submits the keyword
- the name and logo of your search engine (Try to be creative)
  Using CSS and Javascript is optional for this lab.

F2. Query Data Processing

Your frontend should be able to recognize the number of words in the query submitted by the user, and should ignore white spaces in the query.

In this lab, your frontend should display the number of words that have been submitted in a phrase, and display the number of appearances for each word.

Example:
User submits the following string in the text box:

The lab 1 of CSC326 lab is the first lab

You frontend should display following information in the result page:

Search for "The lab 1 of CSC326 lab is the first lab"

Word Count

the 2
lab 3
1 1
of 1
csc326 1
is 1
first 1

F3. The Bottle web framework

To host your web service, you should use the Bottle web framework, which is a single file module, and has no dependencies other than the Python Standard Library. The documentation of Bottle can be found on http://bottlepy.org. Below is a simple example to install and setup the server.

```
pip3 install bottle
```

Create a file named `HelloWorld.py` with following lines.

```python
from bottle import route, run
@route('/')
def hello():
    return "Hello World"
run(host='localhost', port=8080, debug=True)
```

When the server is running, you can access your web page in the browser with address `http://localhost:8080/`

F4. Requirements

- Your frontend should present a simple interface for user to submit a keyword or a phrase
- In response, your frontend should show the keyword on a result web page. In the case when a phrase (i.e. more than one word) is submitted in the query, your frontend should list the number of keywords in the phrase and the number of appearances for each keyword in the phrase as specified in Part F2.
- Store the history of the top 20 most popular keywords searched since the server is launched. At the time when the server is launched, it should have no record for any keyword.
- Display the top 20 keywords on the query page, and the total number of times that these words have been searched

F5. HTML Format

- Query submission must be placed in an HTML form using HTTP GET method
- Query submission form and result page must be processed at the root path of the website, i.e. http://hostname:port/
- The keyword string must be submitted using an HTML form input with name **"keywords"** and type **"text"**
- Results for the word counts must be placed in an HTML table with name **"results"**
- Results for the top 20 keywords must be placed in an HTML table with name **"history"**

Example of an expected HTML table for the results:

```
<table id=”results”>
    <tr>
        <td>the</td>
        <td>2</td>
    </tr>
    <tr>
        <td>lab</td>
        <td>3</td>
    </tr>
</table>
```

Note that failure to follow the exact requirements for the HTML formats for input and output may result a grade of zero for the correctness of the implementation for the frontend.

Backend

B1. Inverted Index

An inverted index stores a mapping from content to its locations. For example, it maps a word to a list of documents that contain the word. In a search engine, the inverted index is used to look
up a list of URLs given a keyword.

For search engine, a crawler is used to visit a repository of web pages, and generate the document index, lexicon, hit lists, forward index, and inverted index.

B2. Crawler

You can find a reference implementation of the crawler on Quercus

The crawler recursively traverses a list of URLs specified in the `urls.txt`.

- A document id is created for each new URL visited, and the URL is cached such that a new document id will not be generated for next visit.
- If the URL has been visited, then move to the next URL on the list.
- For URLs that have not been visited, `BeautifulSoup` APIs are called to load the content of the page specified by the URLs.
- Once a page is loaded, the crawler traverses the document in depth-first order and processes the text between tags.
- Text statements will be parsed by the crawler, and each new word is added to the index with a unique word id. Also, html tags that are not interested will be ignored by the crawler, e.g. `<meta>`, `<script>`, etc.
- After a list of words is created for a page, all the words should map to the document id of the page.

To run the crawler, execute the following command in terminal

```
$ python3 crawler.py
```

B3. Requirements

After the crawler finishes processing a list of URLs, your data structure should maintain the following information in memory:

- Document Index, that keeps information about each document. This document should be ordered by document id.
- Lexicon, that keeps a list of words.

- Inverted Index, that returns a list of document Ids given a word id
  In the crawler class, you should provide a function, `crawler.get_inverted_index()`, which
  returns the inverted index in a **dictionary**. You should use the word id as the key, and the list of
  document ids as the value. The list of document ids should be stored in a **set**.

Also, you should provide a function, `crawler.get_resolved_inverted_index()`, where word ids are replaced by the word strings, and the document Ids are replaced by URL strings in the `invertedindex. get_resolved_inverted_index()` should also return data in a **dictionary**.

Example

```
>>> from crawler import crawler
>>> crawler = crawler(None, ‘urls.txt’)
>>> inverted_index = crawler.get_inverted_index()
>>> print(inverted_index)
{'1':{1, 2},
 '2':{1, 3}
}
>>> resolved_inverted_index = crawler.get_resovled_inverted_index()
>>> print(resolved_inverted_index)
{'google': {'http://google.ca', 'http://google.com'},
 'search': {'http://google.ca', 'http://bing.com'}}
```

In the above example, ***word id*** of `google` and `search` are **1**, and **2** respectively. The ***document id*** for `http://google.ca`, `http://google.com`, and `http://bing.com` are **1**, **2**, and **3** respectively.

Hints
You may find the following features useful for your future development of the labs, however these implementations will not be graded for this lab:

1. Store the title for each web page and a short description of each web page, i.e. first 3 lines on the page, in the document index
2. Provide an API that returns the title, and short description for each web page given a document id

Deliverables

Frontend(50%)

- Source code of the frontend
- UI design must follow requirements listed in the **Frontend** section

Backend(50%)

- Source code of the backend
- Valid test cases for verifying the correctness of the crawler (using any test framework, e.g., unittest, pytest, etc.)

Group Number

You should be working in a group of two for the course project. To receive your group number, one member of your group should send your names and UTORIDs to the lab TA.

To work alone or in a group of more than two members, you must obtain the approval from the instructor.

Submission

Code

Place all your files in a directory named **lab1_group_<group_number>**, and compress it to **lab1_group_<group_number>.tar.gz**

For example, for group 4, use the following command
`tar -zcvf lab1_group_4.tar.gz lab_1_group_4/`

Try to minimize the submission size by including only mandatory files and directories needed to build your project (do not include directories such as **.idea**, **.vscode**, **.git**, **venv**, **.DS_Store**, etc.).

You may attach a **README** file if you have specific instructions for building the project, as well as a **requirements.txt** file for additional dependencies.