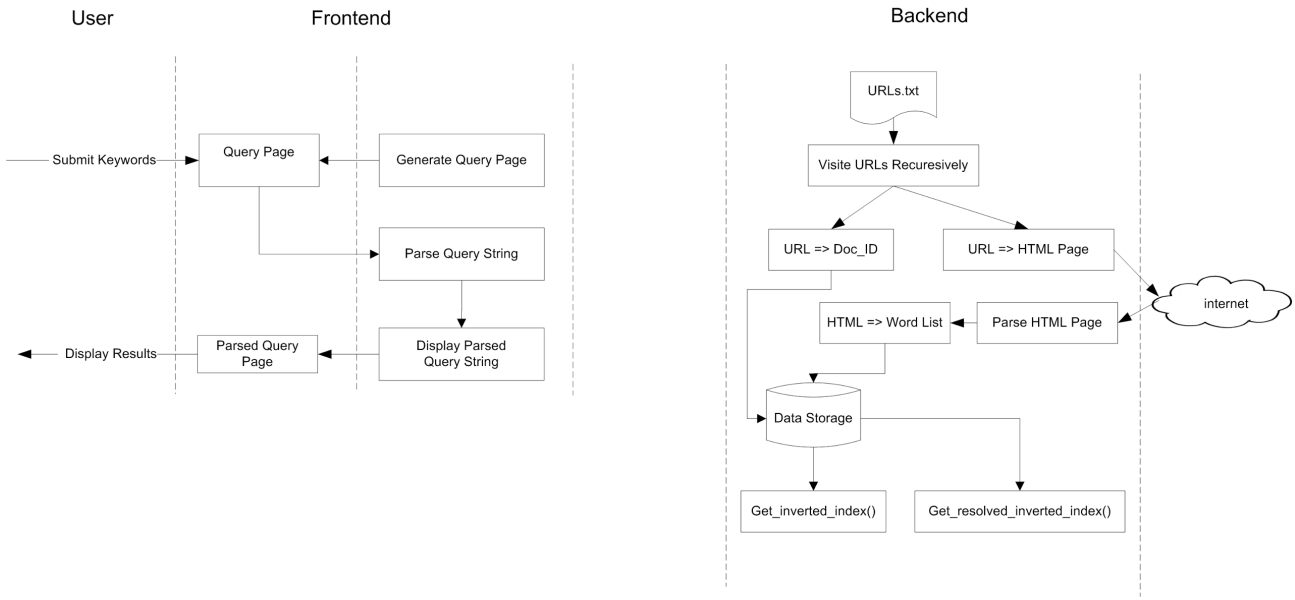
Lab 1 - Development Phase 1



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
- a button that submits the keyword
- the name and logo of your search engine (Try to be creative in this)

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 also 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 occurrence 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 "*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.

```
$ cd <YOUR-PROJECT-DIRECTORY>
$ wget https://pypi.python.org/packages/source/b/bottle/bottle-0.11.6.tar.gz
$ tar -zxvf bottle-0.11.6.tar.gz
$ cd bottle-0.11.6
$ python setup.py install --user
```

Create a file named HelloWorld.py with following lines.

```
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 keyword) is submitted in the query, your frontend should list the number of keywords in the phrase and the number of occurrence for each keyword in the phrase as specified in Part F2.
- Your frontend interface should have relatively consistent appearance across different browsers, e.g. Chrome, Firefox, Safari, IE, and etc.
- Your frontend should be user friendly. For example, when user resize the window of the browser, the elements on your page should react corresponding to the actions.

F5. Bonus

1. Store the history of the top 20 most popular **keywords** searched.
2. Display the top 20 keywords in on the query page, and the total number of times that these words have been searched.

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. Reference Implementation of a Crawler (crawler.py)

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

```
$ python crawler.py
```

B3. Requirements

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

- 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()***,that returns the inverted index in a dict(). 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 inverted index. get_resolved_inverted_index() should also return data in a dict().

Example

```
>>> crawler = new Crawler()
>>> inverted_index = crawler.get_inverted_index()
>>> print inverted_index
{ '1': set([1, 2]),
  '2': set([1, 3])}
>>> resolved_inverted_index = crawler.get_resovled_inverted_index()
>>> print resolved_inverted_index
{'google': set(['http://google.ca', 'http://google.com']),
  'search': set(['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.

B4. Bonus

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

Deliverables

Frontend

- Source code of the frontend

Backend

- Source code of the modified crawler
- Test cases for testing the correctness of the crawler

* All source code must be well commented, and explicit instructions must be provided to run your application. If your code is not simple to run, it will cost you.

Lab Report

- A report that describes the design of your search engine.
- Describe the design decisions made for the frontend.
- Describe the contribution for each member.
- Instructions to run your code.

Bonus Work

- To claim bonus marks, you must explicitly indicate the bonus features that you have implemented, and include it in your lab report.

Submission

Code

Compress all your files, including the source code and the report, and name it lab1_group_<group_number>.tar.gz with tar.

For example, for group 4, use the following command.

```
$ tar -zcvf lab1_group_4.tar.gz <frontend file> <backend file> <report file>
```

To submit your package, use the following command on EECG machine:

```
$ submitcsc326f 1 lab1_group_<group_number>.tar.gz
```

Note that it cost you if you don’t follow the above instruction exactly.

Progress Report

Create a Google Drive document for your lab report, and use it as a working document for all your labs, i.e. at the end of the project, this document should include all your lab work done from Lab 1 to Lab 4.

Think of the final report as the documentation for the whole project (with Architecture, Design, Implementation, Testing, Build and Deploy instructions. In addition to these sections, it should have a Table of Contents and Revision History table at the beginning of the report).

You have to share this document to zxuan@eecg.toronto.edu before the deadline of Lab 1.

Grades

Frontend	4
Backend	4
Report	2
Total grade for Lab 1	**10**
Bonus	2
Total possible grade for Lab 1	**12**

Note that the total grade for all labs will not exceed the total grade allocated for the course project.

Attendance and Grace Period

Attendance will be taken at the beginning of each lab. For those who have attended all the labs, they will have a total of 2 grace days for lab submission in the semester. For those who have missed only one lab in the semester, they will have 1 grace day. If you miss more than one lab, you will have no grace days.

All your lab submission will be graded. However, if you submit a lab late, and you have not acquire any grace period at the end of the semester. Your late submission will receive a penalized grade.