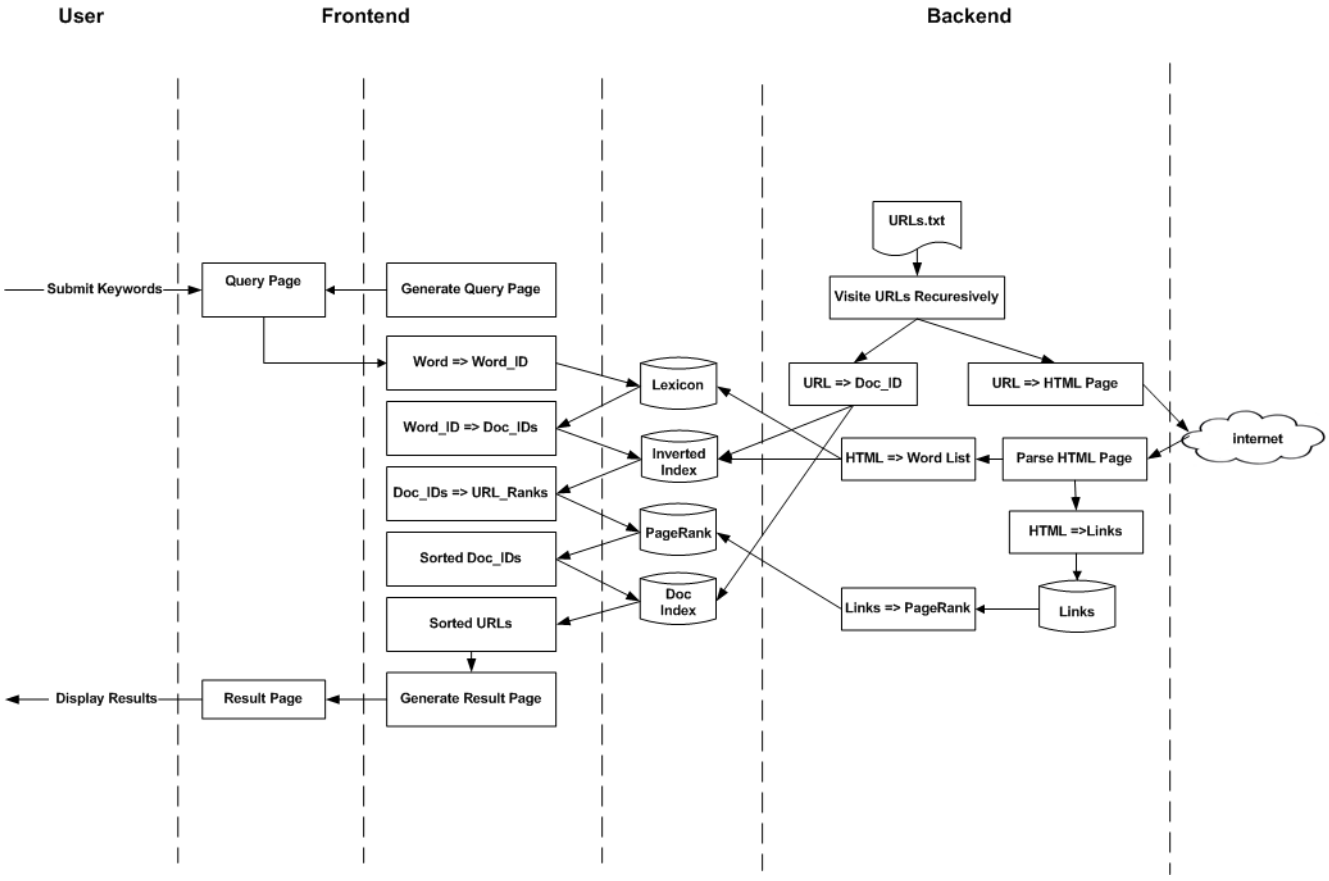
Lab 2 - Development Phase 2

In this lab, you will continue the development of your frontend by integrating the inverted index generated by the backend. For the backend, you will compute and store the PageRank scores for URLs in the document index. In this lab, you should store all the collected data, including the inverted index, lexicon, document index, PageRank scores to persistent storage, i.e. databases.

By the end of Lab 2, your search engine should provide functionalities similar to those illustrated by the following diagram. Note that the below architecture is not optimized, and is only for reference. You are free to implement any architecture for your search engine, as long as it preserves three main components: the frontend, the backend, and the persistent storage.



Frontend

F1. Search Engine Result Page

In response to the user query from the web browser, your frontend should search the keywords against the databases generated by the backend. The retrieved list of URLs should be displayed on the browser in a sorted order.

To simplify your search algorithm, your search engine is only required to search against the first word in the query string. For example, if an user searches “CSC326 lab2”, your search engine is only required look up “CSC326” from the databases.

On the result page, your frontend should also provide a query interface for user to search a new keyword.

F2. Error Page Handling

When user is trying to access a web page or a file that does not exist on your website, your frontend should return an error page indicating that such page or file does not exist, and provide the user a link to the home page of your website.

F3. Requirements

- When a keyword is submitted, your frontend should return a list of URLs.
- The returned list of URLs should be sorted by the PageRank score of each page.
- Your frontend should be user friendly, such that it only returns a limited number of URLs per result page. For example, if 1000 URLs are found in the database when a keyword is searched, you do not want to display the entire list of URLs at the same time.

F4. Bonus

- Provide additional information for the search results, including the title and short description of each URL.
- Load and display the search results asynchronously with AJAX on one page when user scroll down the browser. For example, if your frontend displays 100 URLs per page, you should display first 20 URLs when the page is loaded, and display the rest of the page gradually when user scroll down the page.

Backend

B1. PageRank Algorithm

PageRank is a link analysis algorithm that rank a list of documents based on the number of citations for each document. PageRank algorithm gives each document a score, which can be considered as the relative importance of the document. When a page has high PageRank score, it may be pointed by many other pages, or it may be pointed by some pages that have high PageRank scores.

For details about Google's original PageRank Algorithm, see the paper at http://infolab.stanford.edu/~backrub/google.html

B2. Compute PageRank Scores

To compute the PageRank score, the link between pages must be discovered.

Your crawler should implement a method to capture the link relations between pages, and you need to design a data structure to store these links. Make sure your data structure is compatible with the interface for the PageRank function if you use the reference implementation.

Once the links are discovered, the PageRank function can be called to generate a score for each page, and the generated scores must be stored.

B3. Persistent Storage

The data collected by the crawler and the PageRank scores need to be stored in a persistent storage, such that the frontend can retrieve the data. In this lab, you are free to choose any type of persistent storage you prefer. Below is an example for using sqlite3 in python.

```
>>> import sqlite3 as lite
>>> con = lite.connect("dbFile.db")
>>> cur = con.cursor()
>>> cur.execute('CREATE TABLE IF NOT EXISTS dummyTable(id INTEGER PRIMARY KEY, value
TEXT);')
>>> cur.execute('INSERT INTO dummyTable(value) VALUES("foo");')
>>> cur.execute('SELECT id FROM dummyTable WHERE value="foo"')
```

```
>>> id = cur.fetchone()
>>> print id
(1,)
>>> con.commit()
>>> con.close()
```

B4. Resource

- Reference Implementation of PageRank algorithm, pagerank.py

B5. Requirement

- Compute the PageRank score for each page that is visited by the crawler, given a list of URLs specified in "urls.txt".
- Your backend should generate and store required data in persistent storage for the frontend to retrieve.

B6. Bonus

- Implement a multithreaded crawler. The reference crawler provided in Lab 1 is implemented using a single thread. Since the list of URLs is not required to be visited sequentially, performance of the crawler may be improved significantly if it is implemented with multi-threads.

Deliverables

Frontend

- Source code of your frontend

Backend

- Source code of the modified crawler with your implementation of PageRank algorithm.
- Test cases for testing your implementation

* All source code must be well commented, and explicit instructions must be provided on how 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 both the frontend and backend.
- 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.
- Specify what is the gain of performance if you implement the bonus part for backend.

Submission

Code

Compress all your files, including the source code and the report, and name it lab2_group_<group_number>.tar.gz with tar.

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

```
$ tar -zcvf lab2_group_4.tar.gz <frontend file> <backend file> <report file>
```

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

```
$ submitcsc326f 2 lab2_group_<group_number>.tar.gz
```

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

Progress Report

Add the Lab 2 report to the document you created for Lab 1.

This document should be used 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).

Grades

Frontend	4
Backend	4

Report	2
Total grade for Lab 2	**10**
Bonus	2
Total possible grade for Lab 2	**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.