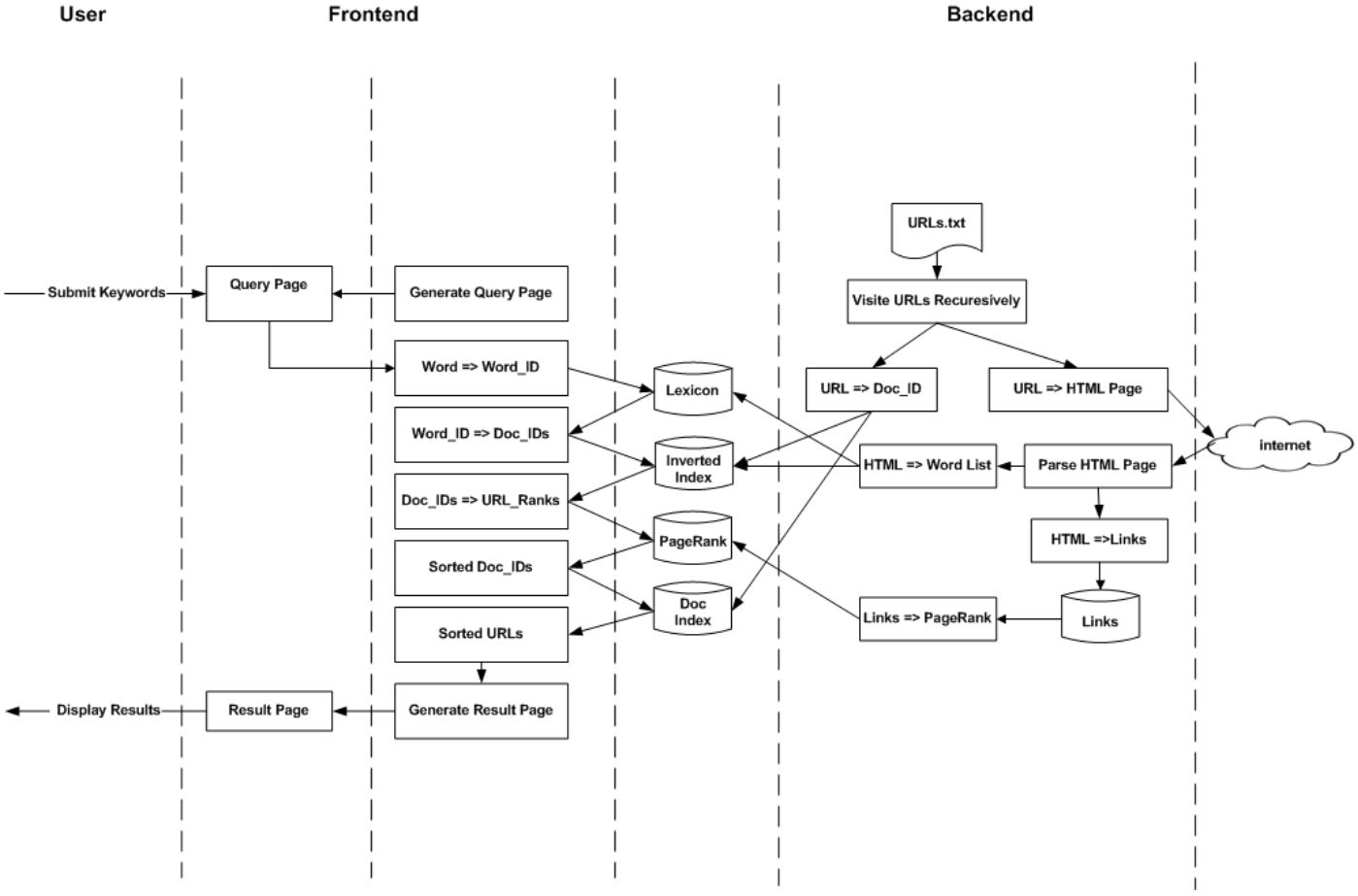
Lab 3 - Development Phase 2

In this lab, you will continue the development of your frontend by integrating the data 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.

By the end of Lab 3, your search engine should provide functionalities similar to those illustrated by the diagram below. Note that the following 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 persistent storage generated by the backend. The retrieved list of URLs should be displayed on the browser in an order sorted by PageRank scores.

To simplify the 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 lab3”, your search engine is only required to look up “CSC326” from the databases.

On the result page, your frontend should also provide a query interface, i.e. the input form, for user to submit a new query.

F2. Pagination

Your frontend should display a limited number of URLs per result page to avoid excessive processing time to display the first result page. For example, if 1000 URLs are found for a keyword, the first 10 URLs should to be returned for the first page of the result page. In the case when more than 10 URLs are returned from the persistent storage, your frontend should provide a navigation bar for user to switch between pages. Example: google.com

Alternatively, you may use Asynchronous Javascript and XML (AJAX) to load the page result dynamically as user scroll down the result page. This approached is not recommended for those without prior knowledge of javascript. Example: duckduckgo.com

F3. 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.

F4. Requirements

- When a keyword is submitted, your frontend should return a list of URLs, or a page indicating results for such keyword cannot be found.
- The returned list of URLs should be sorted by the PageRank score of each URL.
- Static pagination or dynamic page loading with AJAX should be used to limit the number of URLs returned by each request sent to the server.
- Error page should be returned when user is trying to access a page that does not existed, or using a HTTP method that is not supported. The error page should provide a link for user to visit the valid query 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, identified by anchor tags, 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 traversed and its relations are their relations are discovered, the PageRank function can be invoked to generate a score for each page or link, and the generated scores must be stored to persistent storage.

Reference implementation of PageRank can be found at pagerank.pys or http://www.petergoodman.me/courses/2011/csc326/project/pagerank.pys

B3. Persistent Storage

The data collected by the crawler and the PageRank scores must to be stored in a persistent storage, such that the frontend can retrieve the data when needed.

In this lab, you are free to choose any type of persistent storage, e.g. sqlite3, leveldb, MongoDB, Amazon RDB, and etc. If you do not have preference for a specific persistent storage, you may use sqlite3. 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()
```

B5. Requirement

- Compute the PageRank score for each page that is visited by the crawler, given a list of URLs specified in "urls.txt".
- Generate and store required data, i.e. lexicon, document index, inverted index, PageRank scores, in persistent storage.

B6. Bonus

- Implement a multithreaded crawler. The reference crawler provided in Lab 1 uses 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.
- Implementation of this part is not trivial, and you should do it only after the rest of this lab is completed and verified.
- In case you have an incomplete version of this implementation, you should submit the crawler with required features from B5 in a separate to avoid any instability introduced by this extra feature.

Deployment on AWS

Similar to Lab 2, the frontend should be deployed on AWS.

D1. Backend Data on Persistent Storage

For this lab, the data generated by the crawler should be stored to persistent storage before deploying the frontend of the search engine on AWS.

For example, if SQLite is being used for the persistent storage, you may run the crawler on EECG machine to generate an SQLite data file. To deploy your search engine, the frontend files and SQLite data file are copied to your AWS instance, and the crawler.py does not need to be uploaded to the AWS instance. When the frontend is being executed, the frontend reads database tables from the SQLite data file directly without invoking the backend crawler.

D2. Baseline Benchmarking

To evaluate the performance of your search engine, you should run the benchmark similar to what has been done in Lab 2. Collect the benchmark results and compare it with the results from Lab 2. In one paragraph, discuss briefly for the difference of the benchmark results between Lab 2 and Lab 3, and the cause of the differences.

D3. Bonus

The AWS free tier provides a limited amount of free usage for relational databases, Amazon RDB, and NoSQL databases, Amazon DynamoDB. You are encouraged to use the AWS RDB and DynamoDB to build a more scalable search engine. However, you are responsible for monitoring the actual usage of these services to avoid extra charges caused by your implementation.

Using RDB or DynamoDB is not required. Bonus marks may be rewarded to those who implement extra features beyond the basic requirements. Note that bonus mark may not be given to implementation without correct basic features.

Deliverables

Frontend

- Source code of your frontend

Backend

- Source code of the crawler with your implementation of PageRank algorithm.
- Unit test cases for testing your implementation of crawler and generated data for persistent storage.

AWS Deployment

- Search engine is active online for one week after the due date of this lab.
- Public DNS of the server should be included in a README file.
- Data generated by backend should have been already stored in persistent storage at the time when frontend is being deployed on AWS.
- Persistent storage should contain data crawled from www.eecg.toronto.edu with depth of one.
- Benchmark setup and results in the README file.

Bonus Work

- To claim bonus marks, you must explicitly indicate the bonus features that you have implemented whether it is completed or incomplete, and include it in a README file.
- Specify what is the gain of performance if you implement the bonus part for backend.

Submission

Compress and archive all your files, including the source code and the README file, and name it lab3_group_<group_number>.tar.gz with tar.

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

```
$ tar -zcvf lab3_group_4.tar.gz <files>
```

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

```
$ submitcsc326f-lab 3 lab3_group_<group_number>.tar.gz
```

Only one member of the group is required submit the package.