

**University of Toronto**  
**Faculty of Applied Science and Engineering**  
**CSC326**  
***Final Report***

Date	2017.12.01
Team Number	5
Project Title	MouZhiA
Tutorial Section	TUT01
Prepared By (Name and Student #)	Feiran Hu    1002160056 Xin Li        1002371408 Yunru Pan    1002159353

## 1 Introduction

MouZhiA is a lightweight search engine that aims to provide a simple, user-friendly and aesthetic interface which allows users to retrieve urls by entering either keywords or image. It provides users options to crawler urls in certain domain with a specified depth.

The websites strives to provide the simplest and most intuitive user interface, while achieving complex queries. It does do by deploying complex back end system, both in data-structure and algorithm, yet hiding all the computational heavy layers under the front-end.

## 2 Detailed Design Clarification

In order to maximize utilization of current web development and Python knowledge, we took both algorithm and feature route to enhance our search engine and a total number of eleven features are categorized as shown in Table 2. We will clarify detailed design within frontend and backend in 2.1 and 2.2 respectively.

Table 2 Algorithm and Features

Algorithm Route	Feature Route
1.Multi-word Searching	1.Spell Correction
2.Complex Ranking System	2.Auto Completion
3.Search engine data structure Optimization	3.Query Phrase Interpretation
N/A	4.Minimize Number of Clicks for Each Search
N/A	5.Animations
N/A	6.Search with Image
N/A	7.Preview
N/A	8.Multi-language Environment

## 2.1 Frontend

With the help of Bottle framework, the frontend script first manages to retrieve all necessary files including templates, javascripts, CSS, images etc. by routing so as to compose the interface. The frontend interface takes in keywords or image from user input and query the database for optimal search results and then paginate the results and display it to the user.

Features implemented in frontend includes:

- Spell Correction
- Auto Completion
- Query Phrase Interpretation
- Minimized Number of Clicks for Each Search
- Animations
- Search with Image
- Preview
- Multi-Language Environment

### 2.1.1 Spell Correction

The implementation of spell correction was inspired by Peter Norvig's essay regarding realization of a spell corrector. Due to the limited depth and amount of data in database, we made specific changes upon the spell corrector that it first fetches words from the crawler database instead of from English dictionaries so that we are able to ensure the searchability of corrected keywords. The spell corrector queries all possible spelling corrections by checking the number of edits required and finally return top candidates. For misspells, the interface automatically corrects spelling and display search results of corrected keywords [Figure 2.1.1(a)]. And if the user insist on the wrong spelling, the interface will give another wrong spelling warning [Figure 2.1.1(b)].

*Figure 2.1.1(a) Spell Correction*

*Figure 2.1.1(b) Spell Correction*



About 3 results

**Showing results for *engine***

Search Instead for *engin*



About 3 results

**Did you mean: *engine***

### 2.1.2 Auto Completion

Auto completion requires coordination of both javascript and Python files. The search bar traces “key up” event and send post request via Ajax to ‘srv.py’. The server file first uses the technique of spelling correction to obtain the capital suggestion and then check searching history for other similars. In other words, auto completion is a self-learning process that the system initially only provides similar expression to the user [Figure 2.1.2(a)] but then is capable of learning from habits and preferences of the user to create a more reliable word database [Figure 2.1.2(b)].

Figure 2.1.2(a) Auto Completion with No History

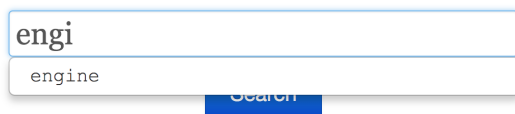
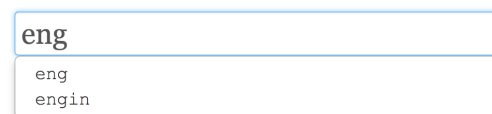



Figure 2.1.2(b) Auto Completion with History of 'engin'



### 2.1.3 Query Phrase Interpretation - Simple Maths Equation Computation

The search engine is able to compute simple maths equations including addition, subtraction, multiplication, true number division, power, trigonometric functions, exponential, absolute value, truncation and rounding. The realization of the feature requires not only parsing but also python maths library.

Figure 2.1.3(a) Multiplication



About 39 results

$$3 * 2 = 6.0$$

Figure 2.1.3(b) sin function



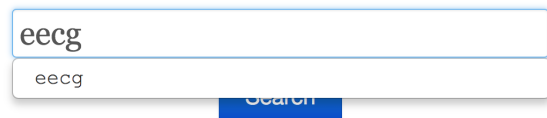
About 0 results

$$\sin(\pi/6) = 0.5$$

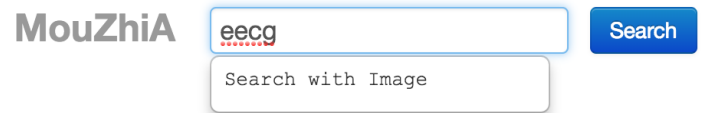
### 2.1.4 Minimized Number of Clicks for Each Search

Both index and result page both contains a search bar. Thus, after initial search at the index page, the user is able to start the next search by inputting keywords or image into the search bar at result page. Specifically, it only requires one click onto the search bar for each search because the form can be submitted by ‘Return’ key. To conclude, minimum number of clicks required for each search equals to 1.

*Figure 2.1.4(a) Search Bar at Index Page*


 A search bar on the index page with the text 'eecg' entered. Below the input field is a blue 'Search' button.

*Figure 2.1.4(b) Search Bar at Result Page*


 A search bar on the result page with the text 'eecg' entered. To the left of the search bar is the 'MouZhiA' logo. Below the input field is a button labeled 'Search with Image'. To the right of the search bar is a blue 'Search' button.

=

### 2.1.5 Animations

The search engine adopts three animations including an animated logo, a loading animation and a “no result found” emoji animation. Our team created originally two of the three animations with After Effects. And with the means of Bodymovin and Lottie library, the interface loads and plays all animations from “.json” files via javascript.

*Figure 2.1.5(a) Animated Logo*



*Figure 2.1.5(b) Loading Animation*



*Figure 2.1.5(b) No Result Animation*



### 2.1.6 Search with Image

The search engine supports a "searching with image" option, in which users can upload an image to the server. The server processes the image into bit-file, creates a graph from the image, and then computes it with a pre-trained image recognition model. The neural network will try to identify the best description of the image and provide that description as search keywords

### 2.1.7 Preview

Preview is triggered by clicking the green arrow on the left of every url and is substantially temporary screenshot of the selected website. The server captures and saves screenshot via Selenium library and PhantomJS driver and finally the system pop ups the screenshot to the user.

Figure 2.1.7(a) Preview Trigger

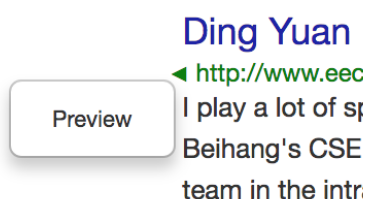
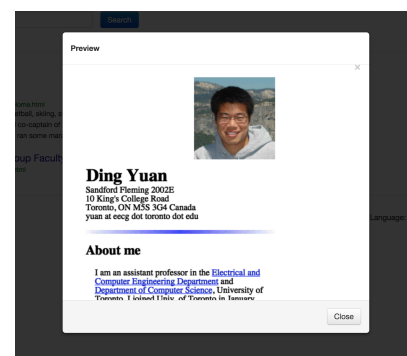


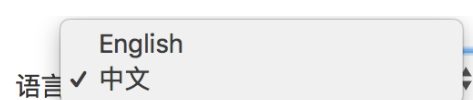
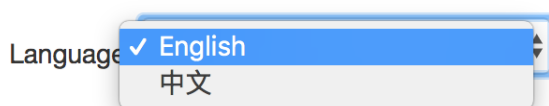
Figure 2.1.7(b) Preview Pop Up



### 2.1.7 Multi-Language Environment

The search engine provides users with two language options, English and Simplified Chinese. Webpages switch language by reading cookies of language and therefore replace texts of html elements with javascript.

Figure 2.1.7 Language Options



## 2.2 Backend

The backend consists of algorithms for crawler, pagerank, and multi word search. As well as persistent database to store page information.

### 2.2.1 Crawler

The backend starts with a crawler, in which the code skeleton is provided by the course. The group did some simple modifications to the crawler by supplying more “enter[]” functions to explore other types of tags.

### 2.2.2 SQL

SQL is used to store all the information from the crawler. A simple relational schema design is performed to represent websites, relationships between websites, words, relationship between words and websites. SQL provided very fast search and update, ensuring search delivery speed.

### 2.2.3 Pagerank

Pagerank algorithm is based on the recommended code skeleton provided by the course. It takes all the graph information provided by the SQL database and calculates the rank based on “referrals”. The “/a” tags are considered to be the sinks of the graph are considered higher in rank.

### 2.2.4 Multi Word search

Words are stored with the following information:

- word content
- in which url
- how many times
- average appearance location, computed based on the insertion sequence of each instance of the word. The idea is, if two words are always adjacent, then their average appearance location is close too.

Then, given string is split and the ranking is based on

- the total number word hits in a given url - T
- Rank - R
- and the sum of (average word position differences of adjacent words in the input string) - P

The pages that contain at least one word in the string are sorted by:

$$(1+R) * T^2/P$$

This is an empirical formula tuned by the group member

The result is somewhat working, where the user intended websites can show up in the first page.

### 3 Code Documentation

#### 3.1 Code References of Features

The following list provides the location of implementation of each feature:

- *Spell Correction* in “*spellingCorrection.py*”
- *Auto Completion* in “*srv.py*”
- *Query Phrase Interpretation* in “*severHelper.py*”
- *Minimized Number of Clicks for Each Search* in “*searchResultAnonymous.tpl*”
- *Animations* in “*base.tpl*”, “*searchResultAnonymous.tpl*” and “*engineAnimation.js*”
- *Search with Image* in “*image\_recognition.py*”
- *Preview* in “*srv.py*” and “*serverHelper.py*”
- *Multi-Language Environment* in “*languageHandler.js*”
- *Multi-word Searching* in “*serverHelper.py*”
- *Complex Ranking System* in “*crawler.py*”
- *Search engine data structure Optimization* in “*crawler.py*”

#### 3.2 External Dependencies

The following list provides information regarding external dependencies:

- Bottle
- Beautifusoup
- Numpy
- Google-api-python-client
- Oauth2client
- Beaker
- Pyparsing
- Filedepot
- Selenium
- Phantomjs
- Tensorflow

It's important to note that Google API is not necessary and also intentionally not to work in the final project but the system still requires dependencies of Google-api-python-client,

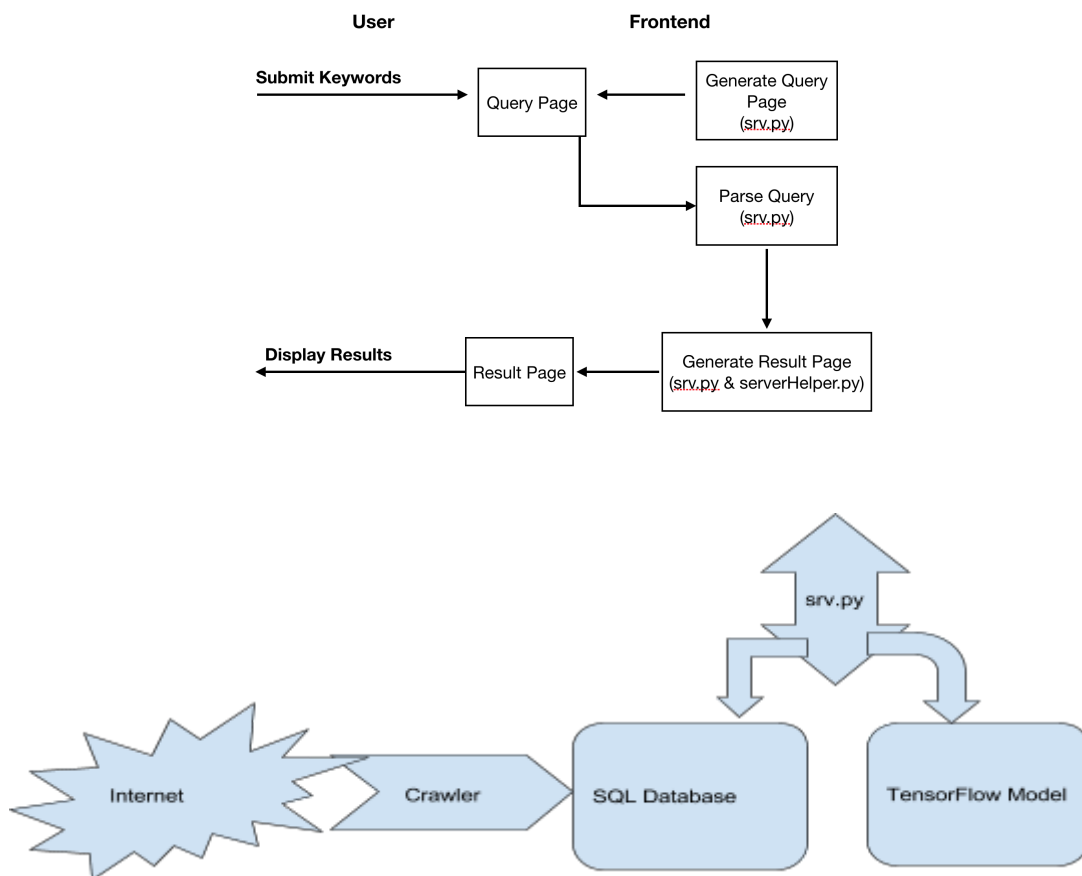


Oauth2client and Beaker.

### 3.3 High-level UML Diagram

The following diagrams illustrates the relations among project files.

*Figure 3.3.1 High-level UMI Diagrams*



## 4 Proposed Design Reflection

The proposed design contains one feature - Search Suggestion which the completed design lacks. The initial design was to display related searches at the bottom of the search result page. We chose to abandon this feature due to two reasons. First, we have already implemented as many as eleven features and are also lack of time due to other labs, quizzes and upcoming final exams. Moreover, after reading documents and closely observing how “related searches” works in Google, we realize that a perfect search suggestion function does

require a large database, otherwise it will have bad performance. To conclude, our team's lacking time and strive for perfection lead to failing the realization of Search Suggestion.

## **5 Testing Strategy**

Backend is tested with testing code such as "test\_backend.py" to check the validity of the result from the crawler. We also used SQL viewer to perform standalone SQL query to validate the result.

Frontend is tested with apache benchmarking, primarily on keyword searches. For the frontend user interface, all group members participated in real-life use case testing.

## **6 Lessons Learned**

In large scale projects, it is extremely important to have hierarchical design and layers of abstraction. Each abstraction layer is program with a different set of requirements using different set of tools. In the case of this project, the split of frontend, backend, and very-backend crawler and aws provides easy collaboration, debugging and development.

"Not re-inventing the wheel" is very necessary in web-development, because of the excellent web frameworks. The group used bootstrap, jQuery and aJax, which vastly improved the frontend quality, development speed and code reliability.

## **7 What Could be Done Differently**

If we have more time, we would do more reliability testing and performance improvement. The code may contain bugs that we are not aware of, but testing the web interface takes a lot of time.

## **8 - 11 Course Feedback**

### **How the material from the course helped you with the project?**

There is a disconnection from the course and the lab, student often do not find the lecture helpful when coding the lab.

### **How much time it takes for you to complete each lab outside the lab sections?**

Days. - At least 10 hour per lab, 30 hours for last lab

### **Which part of the project you think is useful and you believe the labs should spend more**

**time on it?**

Pagerank. Did not spend much time because the course provided sample code and it works with small tweaking. Many student still do not understand the algorithm even though is

**Which part of the project you think is useless and you think it should be removed from the labs when this course is being offered in the future?**

The very first part of the lab is a bit unnecessary, we did not end up using any of the code written in that section.

## **12 Other feedback or recommendations for the course.**

The course gives student good opportunities to tackle open-ended problems, which is a good preparation for real life software development cycles. However, the metrics on what the course grader is looking for is sometimes unclear, and student often times spend a lot of time pondering what exactly he/she needs to implement.

While the solutions can open-ended, it is better for the rubrics to be specific

Another common issue that open arises in the lab is the lack of help sessions or resources provided by the course. This is not an problem for experienced web programmers, but for students who are new to html/css/javascript, AWS, and Database systems, it is particularly time consuming for them to read through API references and figure out which tools they should use. Maybe next term you could offer a help session for at least AWS (so students do not get overcharged also).

This is the feedback gathered from the people in the lab room, our group is satisfied with the practice and knowledge we are getting from the course.

## **13 Distribution**

Overall, all the work is equally distributed to our three team members. Detailed work distribution is as follows:

- Feiran Hu: frontend
- Xin Li: backend
- Yunru Pan: half frontend, half backend