**COMP4321 GP15 Web-Based Search Engine Project Report**

**1. Overall System Design**

This project implements a web-based search engine with 4 core modules:

**Crawler**, **Indexer**, **Retrieval Engine**, and **Web Interface**. The system is designed to recursively fetch web pages, build inverted indexes for efficient search, and provide a user-friendly interface for querying and displaying results.

* 1. **Crawler Key Features**:
* Asynchronous fetching with aiohttp and asyncio for high concurrency.
* Checks Last-Modified headers to avoid re-fetching unchanged pages.
* Uses a visited\_links set to prevent revisiting URLs.
* Records URL relationships in the child\_links table for structural analysis.

**1.2 Indexer Key Features**:

* Stopword removal using predefined stopwords.txt (NLTK stopwords + course-specific list).
* Porter’s stemming algorithm (via nltk) to normalize words (e.g., "running" → "run").
* Uses SQLite to store indexed data in tables

**1.3 Retrieval Engine**

* TF-IDF (Term Frequency-Inverse Document Frequency)
* Cosine Similarity
* Title Boosting
* Phrase Search

**1.4 Web Interface**

* Query input box handles single terms, quoted phrases.
* Paginated results (up to 50 documents) to display scores and page details.

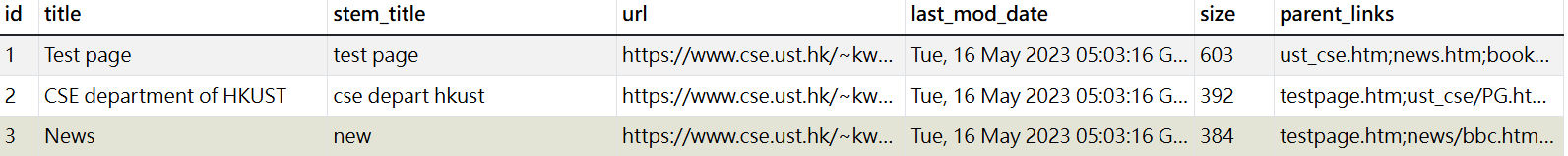
**2. Table Design**

The schema consists of 5 primary tables and uses an SQLite database (scraper.db) for storing crawled data and indexes.

***A. links***

**Purpose: Stores metadata for each crawled page.**

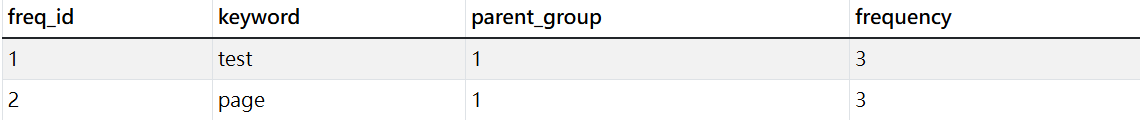
1. **id**: Unique identifier for the page.
2. **title**: The title of the page.
3. **stem\_title**: The stemmed version of the title.
4. **url**: the URL of the page.
5. **last\_mod\_date**: The last modification date of the page.
6. **size**: The size of the page content in bytes.
7. **parent\_links**: A semicolon-separated list of stemmed parent URLs.



***B. keywords\_freq***

**Purpose: Stores keywords and their frequency for each page.**

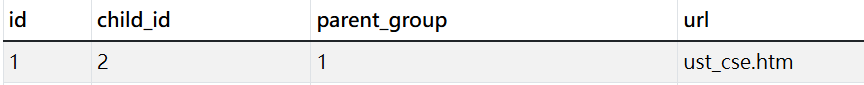
1. **freq\_id:** Primary key.
2. **keyword:** Indexed keyword.
3. **parent\_group:** Reference to the page ID.
4. **frequency:** Frequency of the keyword in the page.



***C. child\_links***

**Purpose: Stores child links of each page.**

1. **id:** Primary key.
2. **child\_id:** Reference to the child page ID.
3. **parent\_group:** Reference to the parent page ID.
4. **url:** stemmed URL of the child link.



***D. body\_positions***

**Purpose: Stores positions of keywords in the page body.**

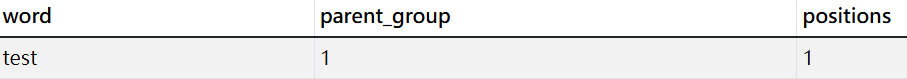
1. **word:** Indexed word.
2. **parent\_group:** Reference to the page ID.
3. **positions:** Comma-separated list of positions.



***E. title\_positions***

**Purpose: Stores positions of keywords in the page title.**

1. **word:** Indexed word.
2. **parent\_group:** Reference to the page ID.
3. **positions:** Comma-separated list of positions.

**(F. sqlite\_sequence)**

**Purpose: number of row of child\_links and keywords\_freq**

一張含有 文字, 螢幕擷取畫面, 行, 字型 的圖片

AI 產生的內容可能不正確。

Forward Index

* Implemented via the `keywords\_freq` table.
* Maps each page (identified by `parent\_group`) to its keywords and their frequencies.
* Structure: For a given page ID, retrieve all associated keywords (`keyword` column) and their occurrence counts (`frequency` column).

Inverted Index

* Implemented via the `body\_positions` and `title\_positions` tables.
* Maps keywords (`word` column) to the pages (`parent\_group`) where they appear, along with their positional offsets (`positions` column).
* Enables efficient keyword-based searches by linking words to their locations in specific pages.

Mapping Tables

* Handled by the `links` and 'child\_links' table.
* Columns `links.id` and ` child\_links.parent\_group` provide bidirectional mapping.

**3. Algorithms and Mechanisms**

**3.1 Crawler: Breadth-First Search (BFS)**

**Strategy**: Fetches pages level-by-level from the starting URL (e.g., https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm), ensuring uniform coverage of the site.

**Pseudocode**:

文本

AI 生成的内容可能不正确。

**3.2 Indexer: Stemming and Stopword Removal**

* **Porter’s Stemming Algorithm**: Reduces words to their root form (e.g., "computing" → "comput", "matches" → "match") to normalize search terms.
* **Stopword Filtering**: Removes common words (e.g., "the", "and") from stopwords.txt to focus on meaningful keywords.

**3.3 Retrieval: TF-IDF and Cosine Similarity**

***TF-IDF Calculation***

文本

AI 生成的内容可能不正确。***Title Boosting Mechanism***

* Keywords in page titles are weighted 3 times higher than those in the body to improve relevance. For example, if keyword “t” appears in a title, its TF-IDF score is multiplied by 3:

***Cosine Similarity***

* Converts queries and documents into vector space and computes similarity as:

文本

AI 生成的内容可能不正确。

* Queries with phrases (e.g., "hong kong") are treated as sub-vectors, requiring consecutive positions in the index for exact matches.

**4. Installation procedure**

Prerequisites

* Python: **Version 3.10 or higher.**
* Flask: Install Flask using **pip install flask.**
* NLTK: Install NLTK and its dependencies using **pip install nltk.**
* SQLite: Pre-installed with Python.

Setup Instructions

1. Clone the repository and navigate to the project directory.
2. Run the database setup script (**main.py**) to initialize the database and index pages.
3. Start the Flask server: python **Flask.py**
4. Access the web interface at **http://127.0.0.1:5000/index.html.**

**5. Implementation Details**

**5.1 Crawler and Indexer**

The crawler and indexer (scraper.py file): uses the aiohttp and asyncio libraries to perform asynchronous crawling. The main steps are as follows:

1. Initialize a queue with starting URL and a set to keep track of visited URLs.
2. While the queue is not empty and the number of indexed pages is less than the maximum number of pages:

|  |  |
| --- | --- |
| **Crawler** | **Indexer** |
| 1. Pop a URL from the queue. 2. Check if the URL has been visited. If not, mark it as visited. 3. Fetch the page content and its headers. 4. Check the Last-Modified header. If the page has been modified, process it; otherwise, skip it. 5. Extract the title, keywords, and links from the page content. 6. Update the inverted index with the keywords. 7. Add the new links to the queue. | 1. Tokenize the page content and title. 2. Remove stopwords from the tokens. 3. Apply the Porter’s stemming algorithm to the tokens. 4. Update the inverted index with the stemmed tokens and their positions.   (support double quote{“”}) |

**5.2 Retrieval Engine**

The retrieval engine is implemented in the Flask.py file. It uses the TF-IDF and cosine similarity algorithms to rank the documents. The main steps are as follows:

1. Calculate the TF-IDF scores for each keyword in the query and each document.
2. Apply the title boosting mechanism to the keywords in the title.
3. Convert the query and documents into vector space.
4. Compute the cosine similarity between the query vector and each document vector.
5. Rank the documents based on their cosine similarity scores.

**5.3 Web Interface**

The web interface is implemented on index.html. It provides a simple form for users to input queries and displays the search results in a structured way. The main steps are as follows:

1. Receive the user query from the form.
2. Call the retrieval engine to get the ranked documents.
3. Format the search results and display them on the web page.

Functionality

* Search Bar: Allows users to input queries (with or without phrases in quotes).
* Spelling Correction: Automatically suggests corrections for misspelled queries by using TextBlob.
* Query history: Users have access to their previous search queries.
* Bookmark: Allows users to save specific search pages.
* Similar page: Base keywords on specific page to make new query
* Jump to rank: Users can quickly navigate to a specific rank in the search results, with options to jump to the top or bottom.

一張含有 文字, 螢幕擷取畫面, 軟體, 網頁 的圖片

AI 產生的內容可能不正確。

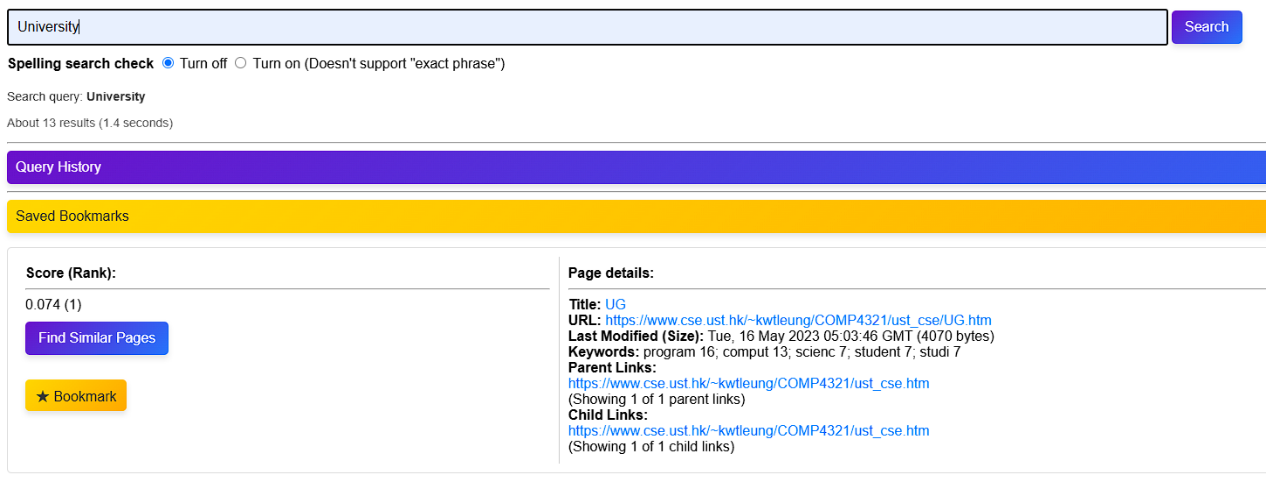
**6. Testing**

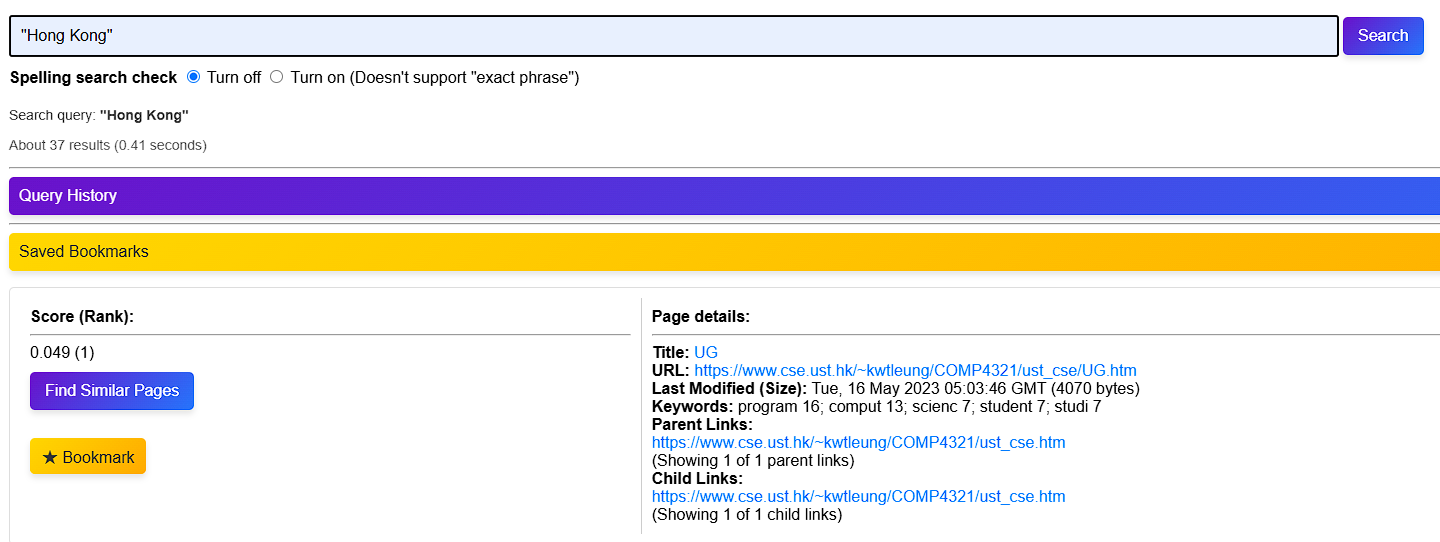
**Spider Functionality:**

**URL:** [**https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm**](https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm) **(297).**

|  |
| --- |
|  |

**Search Engine:**

**Term: (University)**

**Phrases: ("hong kong")**

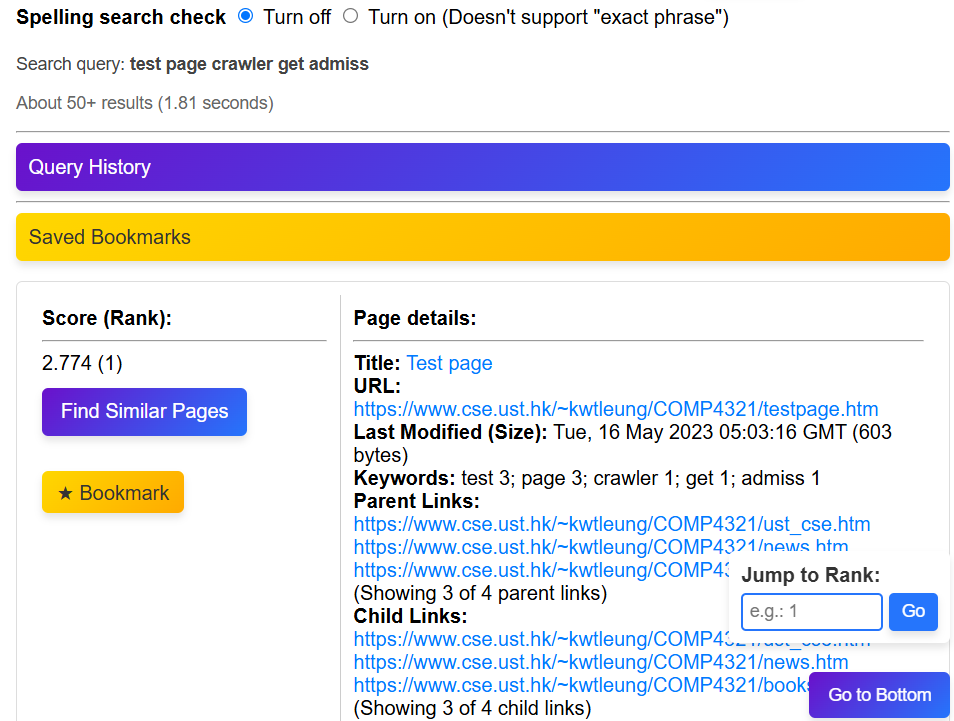
**Mixed query: ("hong kong" university)**

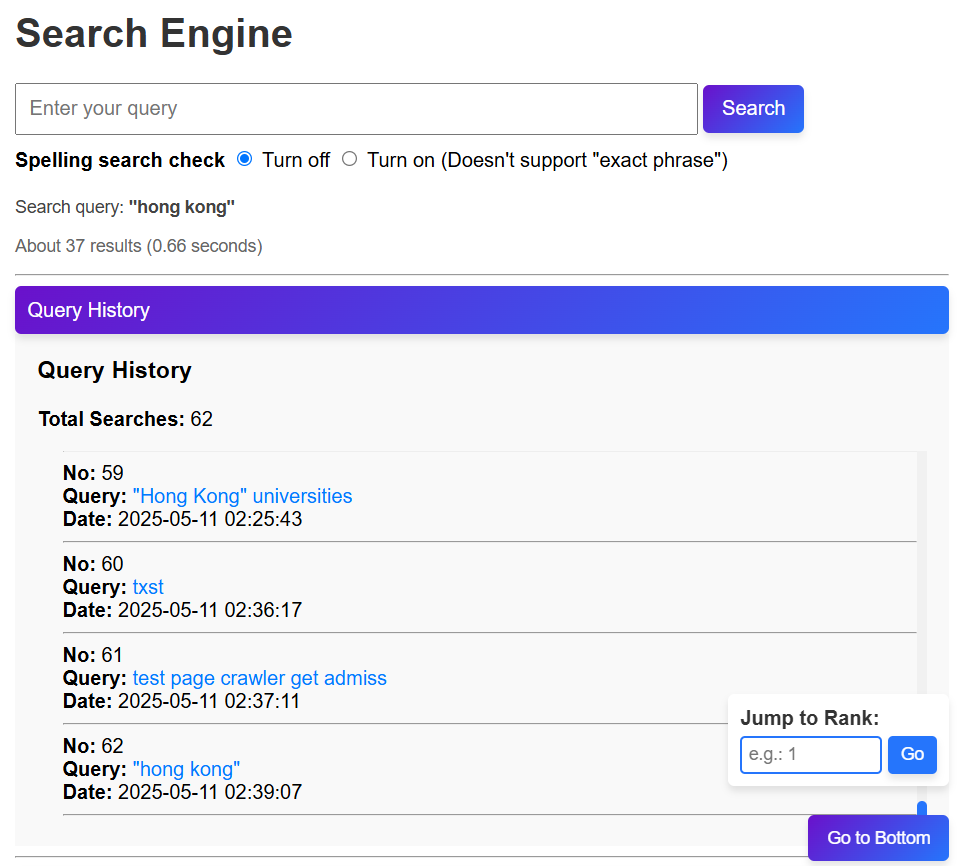
**Spelling Correction**

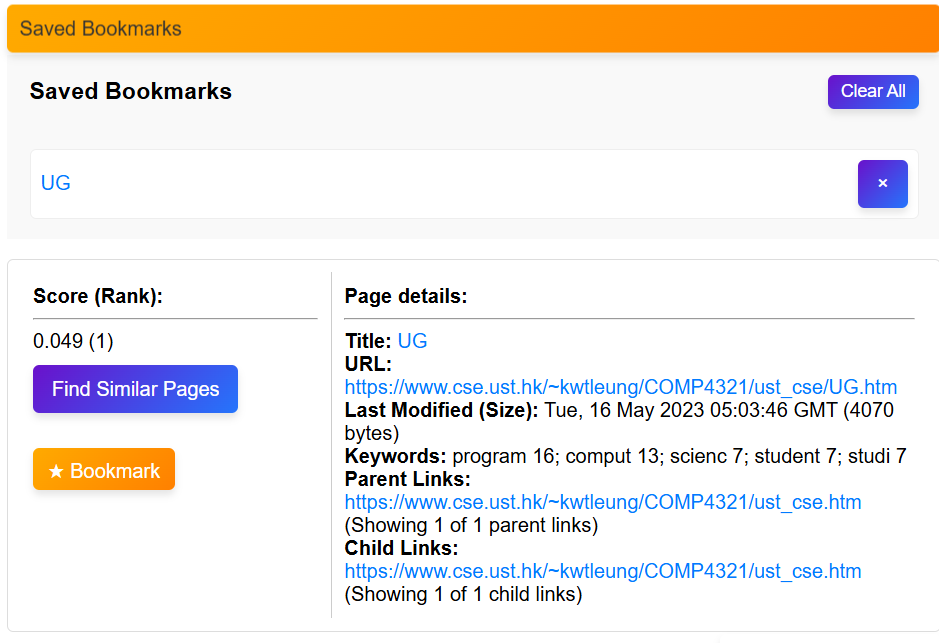
一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

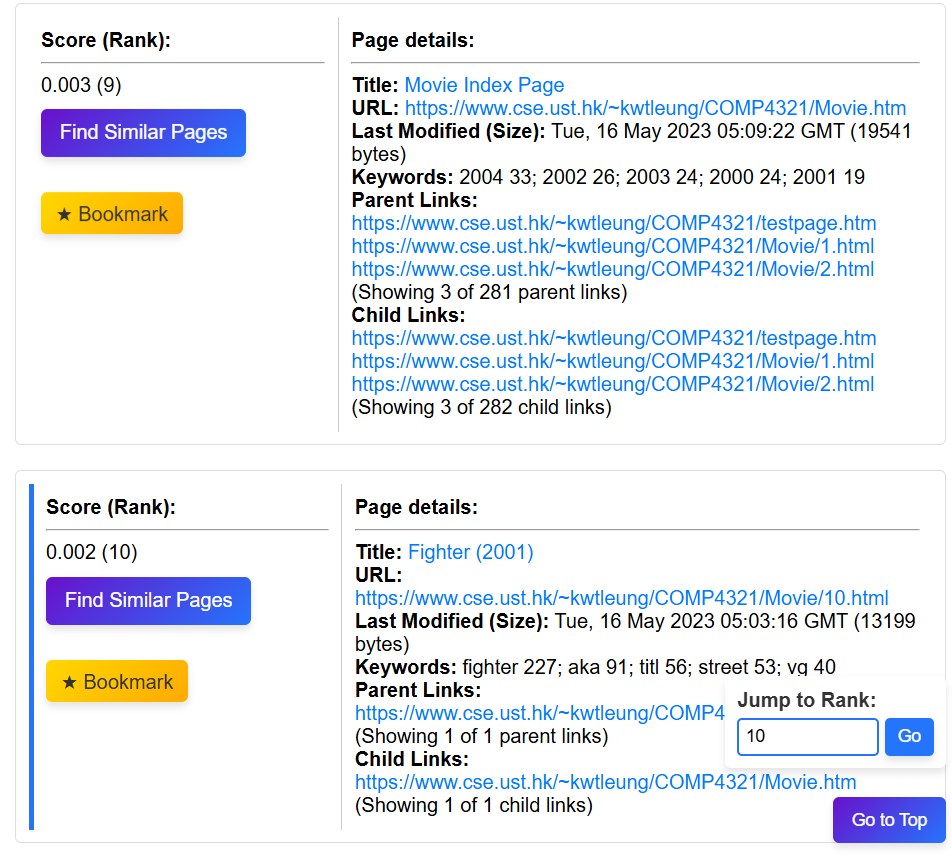
AI 產生的內容可能不正確。

**Similar page (base on up to 5 keywords for that page)**



**Query history**

**Bookmark**

**Jump to rank**

**7. Conclusion**

**7.1 Strengths**

* High Concurrency: The use of asynchronous crawling improved the crawling speed significantly.
* Incremental Update: The incremental update mechanism reduced the redundant crawling and improved efficiency.
* Title Boosting: The title boosting mechanism improved the relevance of the search results.
* Phrase Search: The support for phrase search provided more precise search results.

**7.2 Weaknesses**

* Simple Ranking Algorithm: The ranking algorithm based on TF-IDF and cosine similarity may not be able to handle complex queries effectively.

**7.3 Future Improvements**

* Integrate a PageRank algorithm to improve ranking.
* Develop a mobile-friendly interface using AJAX and DHTML.
* Improve Ranking Algorithm: Explore more advanced ranking algorithms, such as PageRank, to improve the search results.

**8. Contribution**

**William Chen: 40% (crawler, indexer, build up the database)**

**Ho Po Wa: 40% (retrieval engine, enhance the indexer, web interface, flask, html builds up, testing, report writing)**

SZE, Fung Ming: 20% (query input handler, report writing).

[**https://github.com/Skyturtl/Web-Scraper**](https://github.com/Skyturtl/Web-Scraper)