Web-Based Content Search and Ranking Engine

Tyler Anatole

DSE I2400 Data Engineering

Professor Kumud Majumder

5/20/2025

Overview:

This project implements a search engine pipeline that allows users to enter a keyword or phrase and retrieve relevant results across multiple search engines. The results are ranked by how often the term or phrase appears in each page and is displayed like Google with a card interface.

Technologies Used:

|  |  |
| --- | --- |
|  | Tools: |
| Frontend | Streamlit |
| Web Scraping | Requests, BeautifulSoup |
| Browser Automation | Selenium |
| OCR | Tesseract (Pytesseract), PDF2Image |
| Search APIs | Google Search, Bing, Yahoo, DuckDuckGo |
| Database | MySQL |
| Programming Language | Python |

Challenges:

Throughout this project, I faced a lot of challenges. For one, I am not proficient in python and had to reference a lot of forums like StackOverflow or read articles on sites like Medium.com. Once I got the project up and running using streamlit, I found that my pipeline took over 40 minutes. The main cause of the long runtime was the large result sets, unoptimized OCR loops and lack of caching. I fixed this by adding a cap on the amount of OCR screenshots (you can always increase this amount, but be ready for a hefty runtime), streamlined MySQL inserts, and moved slow operations out of the UI thread.

Another issue I faced was duplicate and redundant logic. Some of my processes like urls\_to\_results\_table() were called multiple times. By auditing the pipeline and centralized responsibilities in main.py, I was able to reduce duplication and improve clarity.

The final challenge was the switch that had to be made from Azure OCR (cloud) to pdf2image (local). After hitting the limit for Azure’s Cognitive Services for OCR free tier multiple times, I decided to rewrite the OCR pipeline to use pdf2image + pytesseract locally which turned out to be much better. This switch allow me to have offline processing, unlimited document support (which was best as I had to scrape multiple pages) , and faster iteration without relying on the cloud.

In summary, the biggest challenge was balancing performance and completeness. Every component from scraping to OCR to UI – had trade-offs. I learned that well-scoped logic, schema flexibility, and progressive optimization are key to shipping real-world data pipelines.

ER Diagram:

A screenshot of a computer

AI-generated content may be incorrect.

Architecture:

A diagram of a computer program

AI-generated content may be incorrect.

Process:

1. Built a fully functional Streamlit web app for running real-time search and text analysis

2. Fetched live search results using Google, Bing, Yahoo, and DuckDuckGo

3. Captured full-page screenshots of search results using Selenium

4. Performed OCR using both Tesseract (local) and ~~Microsoft Azure Computer Vision (cloud)~~ Pdf2Image (local)

5. Parsed and scraped HTML content from fetched URLs

6. Ran term frequency analysis to score how often a search term/phrase appears per result.

7. Inserted all content (text, OCR, media) into a structured MySQL database.

8. Created a modular data pipeline in Python with clean .py files for scraping, OCR,

and database logic

9. Identified and labeled media file types (PDF, images, etc.)

10. Displayed ranked results in the app using clickable links and match counts

Screenshots:

Search UI + Final Results

A screenshot of a computer

AI-generated content may be incorrect.

Database Tables:

For the following tables, we will look at search\_id 31 as it is the latest search.

**Search Table**

**A screenshot of a computer

AI-generated content may be incorrect.**

**OCR Screenshots**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Results Table**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Content Info Table**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Term Frequency**

**A screenshot of a computer

AI-generated content may be incorrect.**

Code Snippets:

* How Sponsored/AD links were identified:

A screenshot of a computer code

AI-generated content may be incorrect.

* How text was extracted with Tesseract (pytesseract):

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer code

AI-generated content may be incorrect.

And with PDF2Image which replaced Microsoft Azure’s OCR once we hit the free tier limit several times.

A screenshot of a computer code

AI-generated content may be incorrect.

Some sources used to help with this project:

* Google search engine with streamlit:

“Create a Search Engine with Streamlit and Google Sheets.” *Streamlit*, 14 Mar. 2023, <https://blog.streamlit.io/create-a-search-engine-with-streamlit-and-google-sheets/>.

* Joining URLs:

Glasshost. “Join a Base URL with Another URL in Python.” *Medium*, 4 May 2023, <https://medium.com/@glasshost/join-a-base-url-with-another-url-in-python-549b6506e414>.

* PDF2Img:

Doshi, Nisarg. “From PDF to Images: A Handy, Robust Python Script for Efficient Conversion!” *Medium*, 3 July 2023, https://nisargdoshi.medium.com/from-pdf-to-images-a-handy-robust-python-script-for-efficient-conversion-469c7d7ac687.