Project 2: Search Engine Analysis

Data Engineering DSE I2400

Wild-Hummingbirds Team:

Cody Bisram, Rabiul Hossain, Ivan Miller, Satesh Ramnath, Tyron Samaroo

Spring 2022

Project Description

Goal

Build a web based contents search engine to perform aggregated web search across several search engines that displays result set ordered by frequency match for relevant search terms.

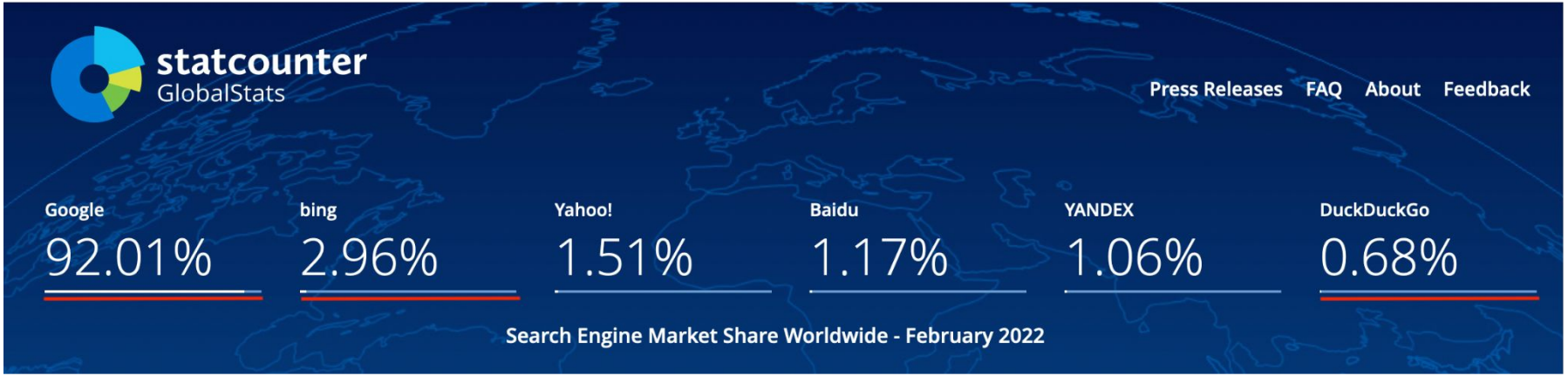
Features

- Search across Web & PDF
- “On the fly” content extraction and frequency scoring of the results from each search engine
- Real time bidirectional sync with MySQL database
- Front end Flask Web Application

See details on GitHub

Search Engine Market Share Worldwide

Google + Bing + DuckDuckGo



Selenium WebDriver + BeautifulSoup

- Headless automation to navigate through Google
- Utilize BeautifulSoup to parse through search meta-data
- Generated a maximum of 180 results per query

Bing Search API v7

50 search results to return per response, up to 200 in total

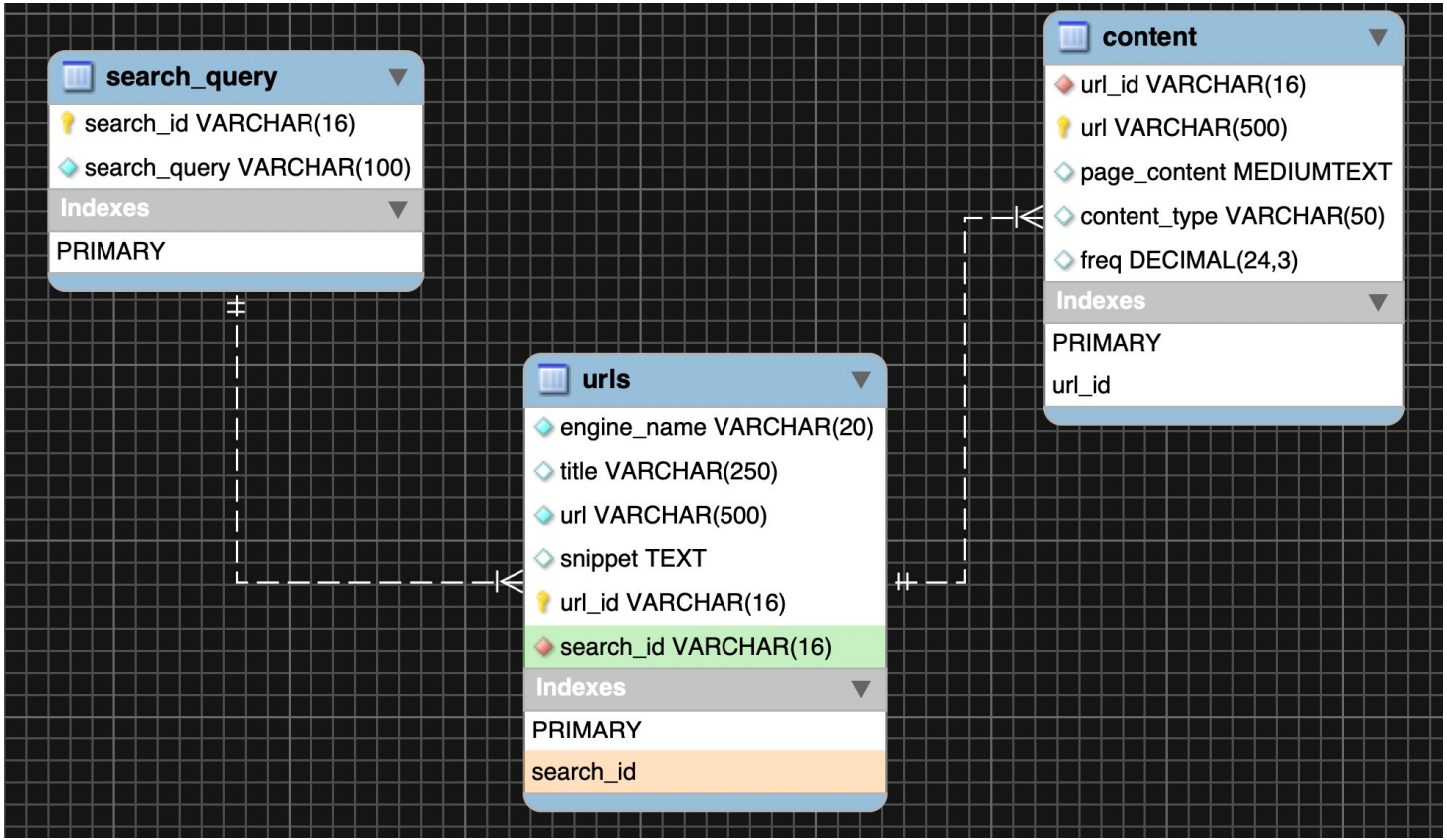
Deduplication of results before loading the data into MySQL database

DuckDuckGo

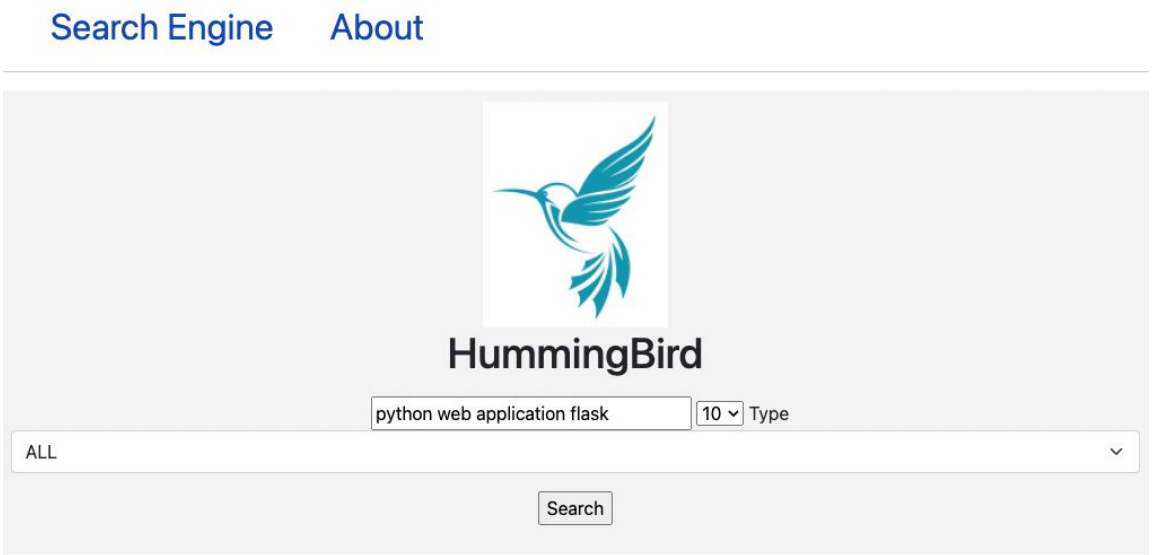
duckduckgo-search 1.8 python package

28 search results to return per response, up to 200 in total

MySQL Database Design



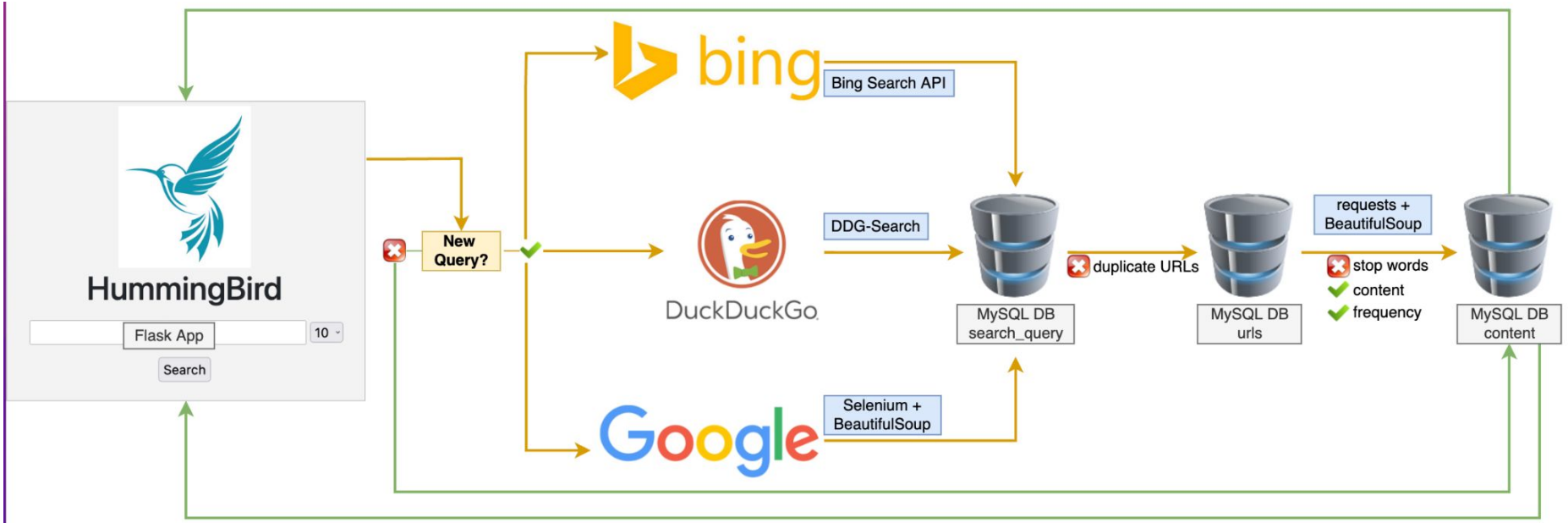
Flask Web App



HummingBird - Search Indicator

A cute animated hummingbird was added to the UI to indicate the idle state of the app

Flow Diagram



Technology APIs

Bing:

1. Create a free Azure subscription

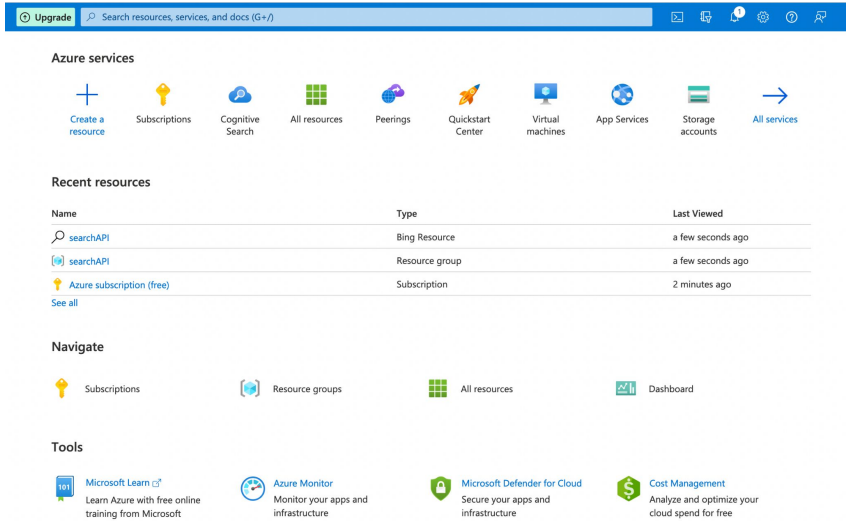
2. Login to https://portal.azure.com/#home

3. Create a resource >> "Bing Search v7"

4. Go to Keys and Endpoint tab and copy the key to use a subscription_key

5. Save the key in a .env file under BING_SEARCH_V7_SUBSCRIPTION_KEY=YOUR_KEY_FROM_#4

Google & DucDuckGo:



Technology APIs

Web Pages

Use search engine specific API or

PDF content

PDF documents are being handled by a combination of packages:

BytesIO - allows for string and byte data manipulation in memory. The usefulness in this is in extracting text from a PDF in memory, as opposed to downloading to our server.

PoolManager - allows for arbitrary requests while keeping track of necessary connection pools.

PDFPlumber - extracts text content using the

```
if res_type == 'HTML':
try:
    page_content = getWebpageText(url)
    print('Webpage text extracted!')
except:
    page_content = None

elif res_type == 'PDF':
try:
    print('Starting PDF extraction')
    print(url)
    if url.endswith('.pdf'):
        page_content = extract_pdf_by_url(url)
    else:
        page_content = None
    print('PDF text extracted!')
except:
    page_content = None

cursor.execute(check_url_exists, (url,))
```

Constraints

- Runtime
  - In order to collect roughly over 500 searches per query, it takes around 2 minutes just to collect the data from all search engines
    - Excludes processing each individual URL and getting word frequency
  - DuckDuckGo Search Engine HTML properties changed and broke our Project 1 application.
    - Switch towards using API and getting smaller set of searches
  - Not all URLs give access to their content
    - Pass null content but keep Search in table compromising of all search results

Images: Optical Character Recognition

Tested Solutions

Google Cloud Vision API

- The API takes ML models centered around image recognition and formats it in a simple REST API interface
    - The image is processed remotely on Google Cloud and produces the corresponding JSON format with respect to the function
    - The JSON file is returned as the output, along with the text from the given image

Python Tesseract

Images: Optical Character Recognition

Implementation

Due to a consistently low frequency score achieved during testing of both Google Cloud Vision API and Tesseract on images, that part of the project was not implemented in the final version.

You could find more details about the implementation on GitHub here.

Demo