CS410 - Text Information Systems

Final Project Documentation

Team: Women in STEM

Kathleen Barta	kbarta2@illinois.edu
Christine Zhou	xizhou4@illinois.edu
Sharon Lin	xinyiyl2@illinois.edu
Kris Kamaei	mkamaei2@illinois.edu
Payel Chakraborty	payelc2@illinois.edu

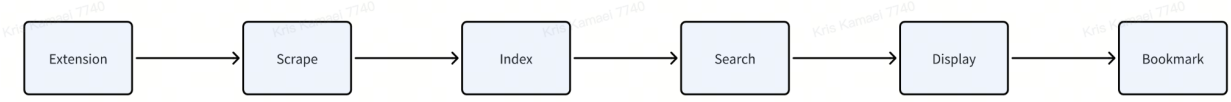
Overview

StudyFriend is a browser extension under the Intelligent Browsing theme intended to scrape and index course-related pages on sites such as Coursera and Campuswire. A user can search for a topic in the extension and be presented with links to related content on all available sites. Important links can be bookmarked for quick access. This will streamline the process for students searching for material by eliminating the need to search and research on each site.

There are 3 major parts to the tool: a web scraper, a search and index function, and a front-end web interface.

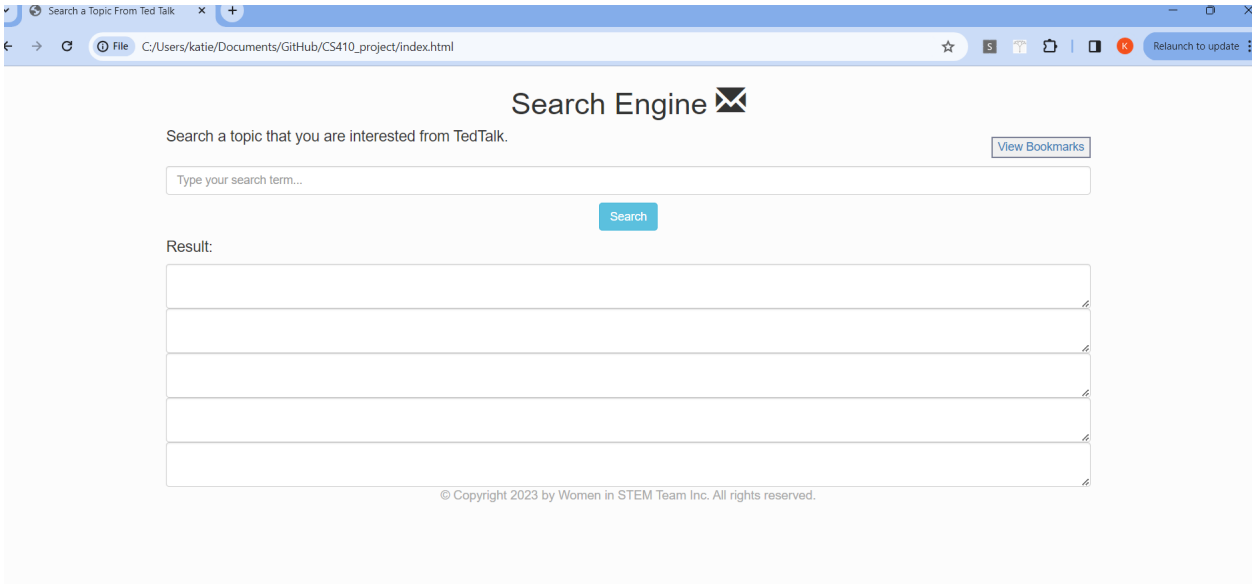
Implementation

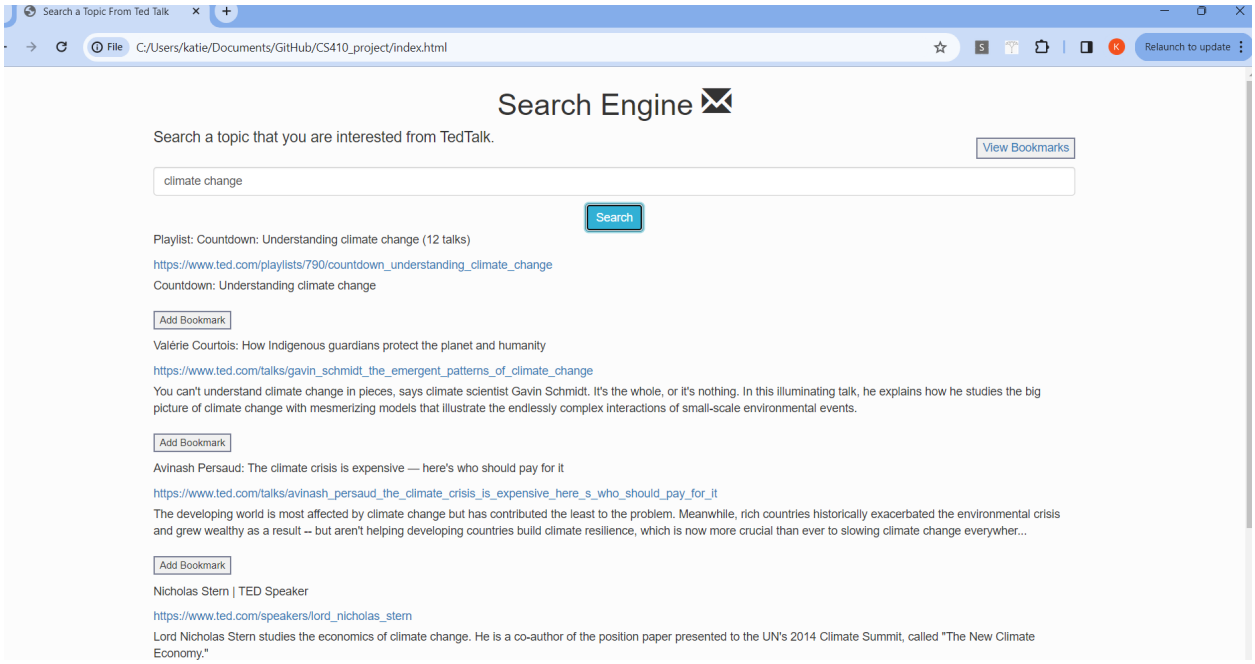
The process of building the extension takes multiple steps, both leveraging frontend and backend capabilities. The design is as follows

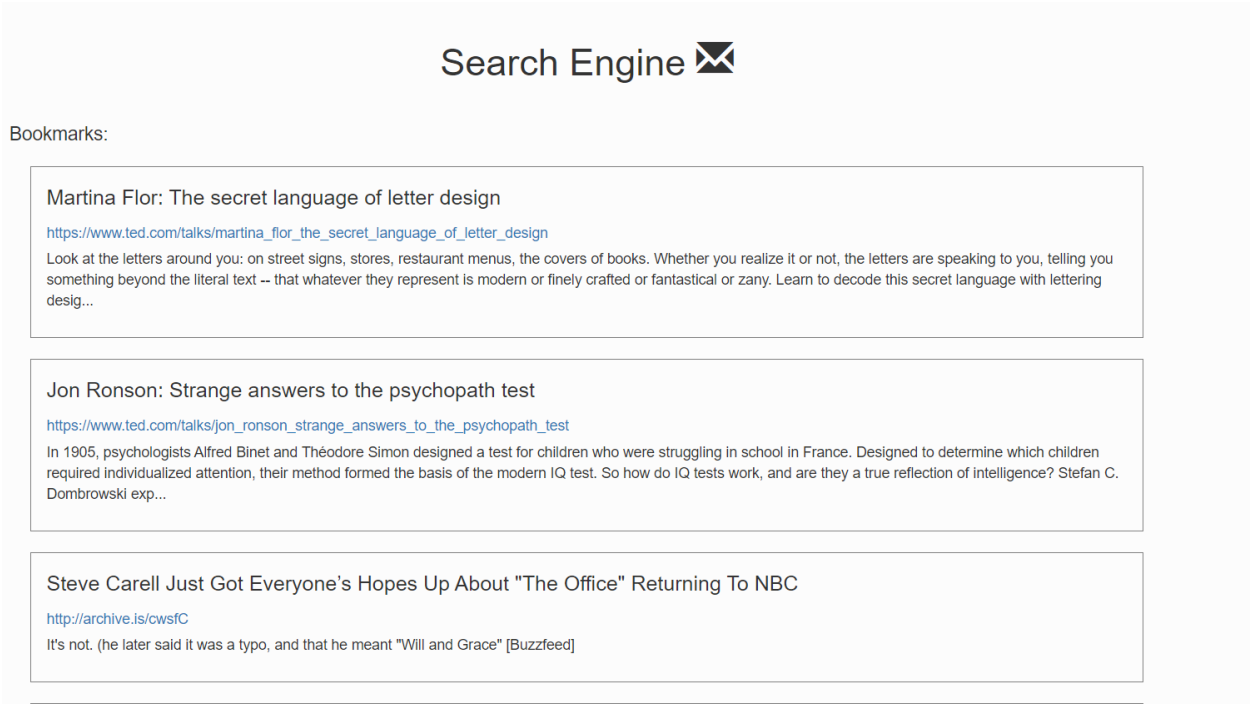


1. Extension

The front-end extension is written in HTML and Javascript. The extension can be viewed as a full webpage or enabled as an extension in Google Chrome and viewed as a popup. The extension uses a JavaScript function, performSearch(), which is triggered by the user entering a search query and pressing the search button. The performSearch function then makes an HTTP POST request to the local server based on a user's inputted search query. When the HTTP request returns, the function prints the output results of that query to the extension webpage or popup.







2. Scrape

Our scraper program returns relevant documents or URLs pertaining to a particular search string. We have focussed on searching topics on TED. It can search for web pages related to any topic as the person using our browser extension wants, but the default is set to the "climate change" phrase.
The first step is to create the full URL that we will search. That involves concatenating our search phrase with the base link "https://www.ted.com/search?q=". Our search phrase will possibly contain spaces or other characters in between as they are received as user input from the front-end. The urllib.parse will replace these characters and modify the search string in a fashion that is acceptable in the HTTPS world.

```
import urllib.parse as urlp

search_phrase_url_encode= urlp.quote(self.search_phrase)
url=
f"https://www.ted.com/search?q={search_phrase_url_encode}"
```

Next, we send the request for a web search. We use the "requests" module that allows us to send HTTP requests using Python. The HTTPS request returns a response object with all the response data. If it faces an error, it throws an error message "Issue calling url".

```
try:
```

```
            response= requests.get(url)
      except:
            print("Issue calling url")
```

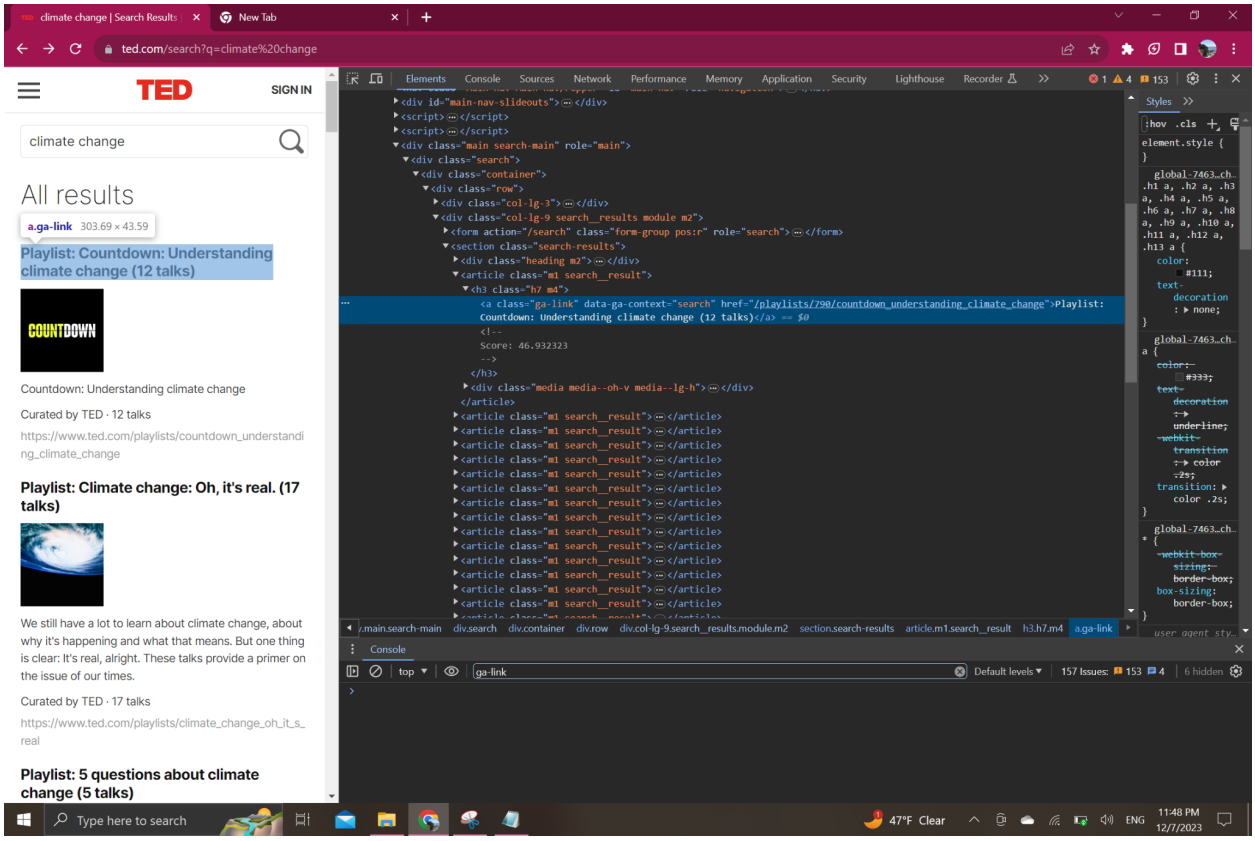
Thereafter, we use the Beautiful Soup Python package for parsing the Requests response (HTML documents). We parse the response content to extract the HTML content which represents the document in a nested data structure.

```
      soup_obj= BeautifulSoup(response.content, 'html.parser')

      return soup_obj
```

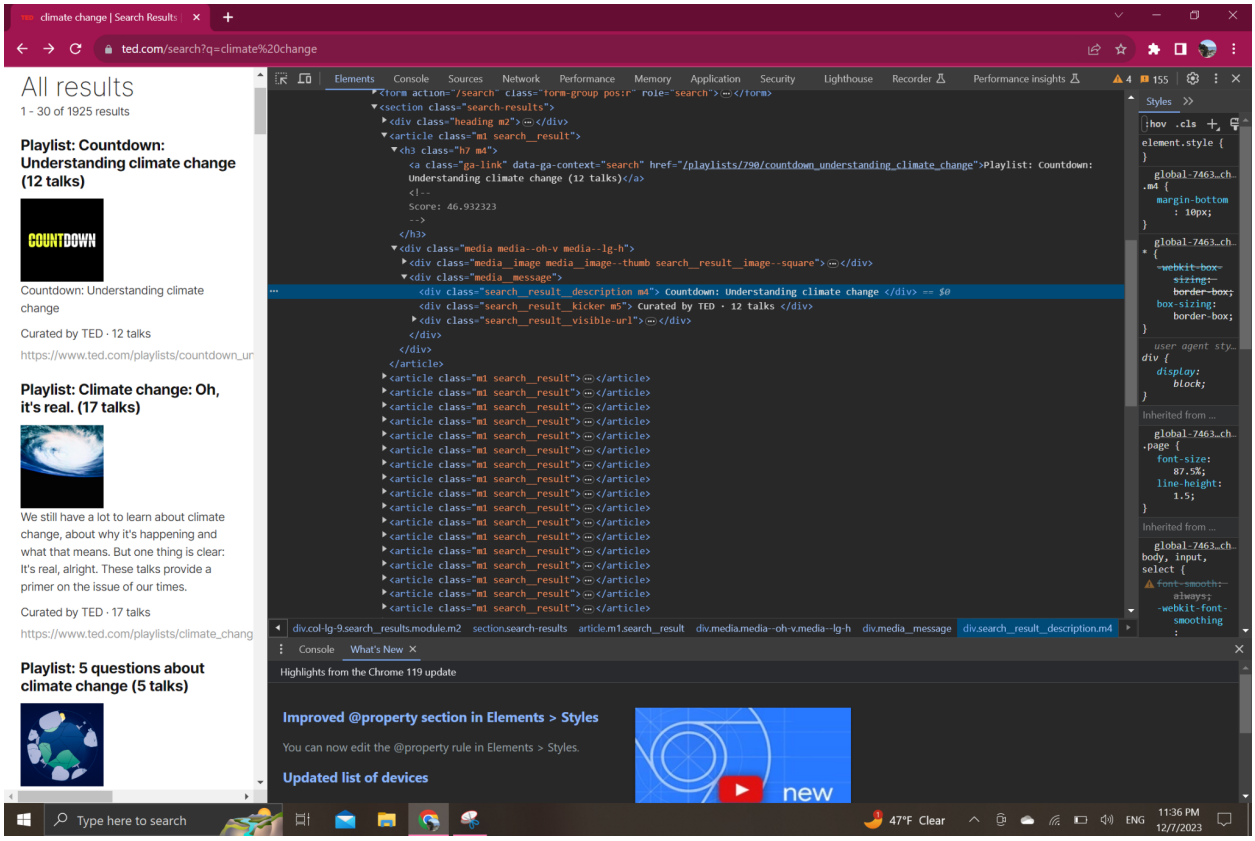
While the soup object is ready, we need to inspect the HTML page source as well as the soup object to find out the tag/level/path where the elements to be extracted are residing within the HTML tree. Our objective is to extract the Title, Description, and HTML link that appears as search results.
The title is found within the tag 'a' and class 'ga-link' by further filtering on 'data-ga-context' = search.

```
      name_list=[]
      lname= self._create_soup().find_all('a',
class_='ga-link')
      stg_name_list= [i.text for i in lname if
(i.get('data-ga-context')=='search' and i.text.find('https') <
0)]
```



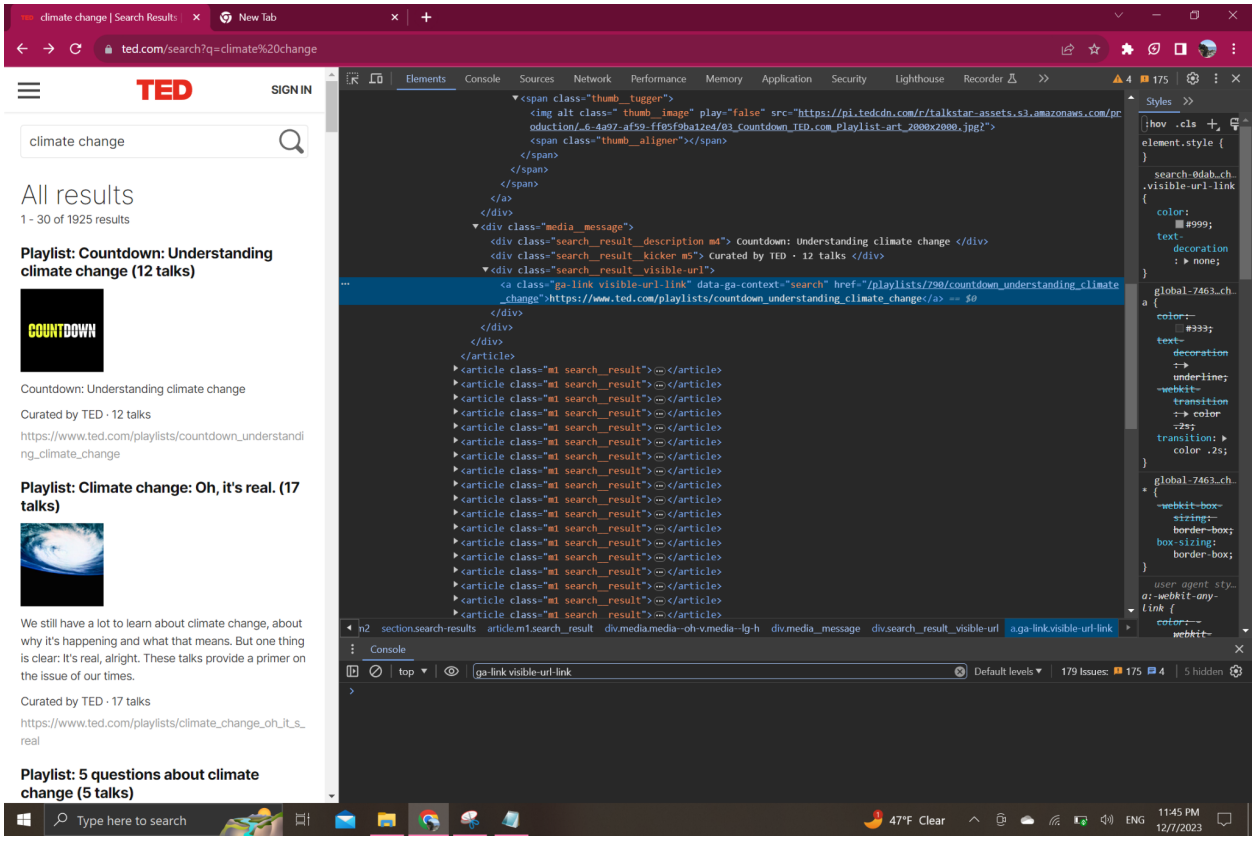
The descriptions are found within tag='div', class='search__result__description m4'.

```
        [ele.text.replace('\n','') for ele in
elf._create_soup().find_all('div',
class_='search__result__description m4')]
```



The URL links are embedded in tag='a', class= 'ga-link visible-url-link' and looping through 'href' within the results we get the list.

```
        url_list = [link.get('href') for link in
self._create_soup().find_all('a', class_= 'ga-link
visible-url-link' )]
        return [self.base_url+i for i in url_list]
```



Finally, we create a record set of Title, Description, and URL link.

<u>Challenges faced while developing the scraper:</u>

There were multiple hurdles that were encountered during the web scraper implementation. We first wanted to scrape the Coursera and Campuswire pages to search for our study materials. But on detailed inspection their nature was found to be of Dynamic Webpage. Dynamic Web pages are dynamically rendered by the client through embedded javascript instead of pulling the entire html from the server. So, when we tried scraping the dynamic webpage, only a partial html got returned through the 'requests' library. The most common library requests' works excellently with static webpage but falters with dynamic webpage. On further investigation it was found that the libraries requests-html and selenium are able to render the entire html by executing the javascript in the backend.

However, then we encountered the other hurdle which was the authentication problem. We tried various methods by using the username/password combination within the url and also creating a config file with the credentials to use them while calling the selenium webdriver. None of them worked, then I realized that selenium fails with 2FA authentication.

Finally, we went ahead with TED Talk.

3. Index and Search

The indexer and search receive input from the scraper in the format of a list data structure with key-value pairs of URL, title, and description extracted. Our approach to indexing and ranking the URLs is through context, by using the descriptions of the page to find the nearest neighbors by performing a similarity search.

Before indexing, we created embeddings out of the descriptions using Google Universal Sentence Encoder, which is available on Tensorflow Hub. Sentence embedding is a technique in NLP where sentences are translated into vector space representations. The primary purpose of sentence embedding is to capture the semantic meaning of a sentence in a way that is comprehensible to algorithms, rather than just analyzing individual words. It also reduces dimension and serves its purpose for semantic comparisons and quickly compares a query sentence against a large database of sentences to find the most similar ones. This is more efficient than comparing raw text.

```
    # load the encoder to generate embeddings
    embed =
hub.load("https://tfhub.dev/google/universal-sentence-encoder/4
")

    descriptions = [item['Description'] for item in
scrape_output]

    # Pass the sentence list to generate the embeddings
    embeddings = embed(descriptions)
```

For indexing and search, we used the Faiss FlatIndexL2 algorithm. The Faiss FlatIndexL2 algorithm is part of Faiss (Facebook AI Similarity Search), a library developed by Facebook for efficient similarity search and clustering of dense vectors. It is designed for efficient similarity search in a vector space. It's particularly used for situations where the distance between vectors is measured using L2 distance (Euclidean distance). The algorithm calculates the L2 distance between a query vector and all vectors in the dataset. FlatIndexL2 performs a brute-force search, comparing the query vector against every vector in the dataset to find the closest matches. This approach, while computationally intensive, is straightforward and doesn't involve any approximation, making it highly accurate.

```
    # Index the embeddings
    index = faiss.IndexFlatL2(embeddings.shape[1])
```

```
index.add(embeddings)

# Example query sentence
q = embed([query])


# Perform similarity search using Faiss
_, result = index.search(q, k)
```

We compared the top 5 results of TED's search and Faiss search

Query	**TED**	**Faiss**
Climate change	Countdown: Understanding climate change	Countdown: Understanding climate change
	We still have a lot to learn about climate change, about why it's happening and what that means. But one thing is clear: It's real, alright. These talks provide a primer on the issue of our times.	You can't understand climate change in pieces, says climate scientist Gavin Schmidt. It's the whole, or it's nothing. In this illuminating talk, he explains how he studies the big picture of climate change with mesmerizing models that illustrate the endlessly complex interactions of small-scale environmental events.
	In the scope of Countdown, TED's initiative to accelerate solutions to climate change, the TED team collaborated with scientists and the creative studio Giant Ant to prepare five short animations explaining concepts and answering important questions related to the climate. They are narrated by Kristen Bell. Learn more about Countdown at countdow...	The developing world is most affected by climate change but has contributed the least to the problem. Meanwhile, rich countries historically exacerbated the environmental crisis and grew wealthy as a result -- but aren't helping developing countries build climate resilience, which is now more crucial than ever to slowing climate change everywher…
	TED Studies, created in collaboration with Wiley,	Lord Nicholas Stern studies the economics of

	are curated video collections — supplemented by rich educational materials — for students, educators and self-guided learners. In Climate Change, speakers give talks that boldly illuminate the nature and scale of current-day climate science, policy and ethics. They explore the economics and psych...	climate change. He is a co-author of the position paper presented to the UN\'s 2014 Climate Summit, called "The New Climate Economy.”
	Learn about the ways climate change is deeply altering how we live, where we live and the foods we eat -- ultimately threatening some of our most basic human rights.	Top climate scientist James Hansen tells the story of his involvement in the science of and debate over global climate change. In doing so he outlines the overwhelming evidence that change is happening and why that makes him deeply worried about the future.

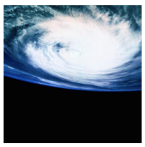
Playlist: Countdown: Understanding climate change (12 talks)

Countdown: Understanding climate change

Curated by TED · 12 talks

https://www.ted.com/playlists/countdown_understanding_climate_change

Playlist: Climate change: Oh, it's real. (17 talks)



We still have a lot to learn about climate change, about why it's happening and what that means. But one thing is clear: It's real, alright. These talks provide a primer on the issue of our times.

Curated by TED · 17 talks

https://www.ted.com/playlists/climate_change_oh_it_s_real

Playlist: 5 questions about climate change (5 talks)

In the scope of Countdown, TED's initiative to accelerate solutions to climate change, the TED team collaborated with scientists and the creative studio Giant Ant to prepare five short animations explaining concepts and answering important questions related to the climate. They are narrated by Kristen Bell. Learn more about Countdown at countdow...

Curated by TED · 5 talks

https://www.ted.com/playlists/5_questions_about_climate_change

Environmental studies: Climate Change

TED Studies, created in collaboration with Wiley, are curated video collections — supplemented by rich educational materials — for students, educators and self-guided learners. In Climate Change, speakers give talks that boldly illuminate the nature and scale of current-day climate science, policy and ethics. They explore the economics and psych...

https://www.ted.com/read/ted-studies/environmental-studies

Playlist: Why climate change is a human rights issue (11 talks)

Learn about the ways climate change is deeply altering how we live, where we live and the foods we eat -- ultimately threatening some of our most basic human rights.

Curated by TED · 11 talks

https://www.ted.com/playlists/why_climate_change_is_a_human

While Faiss results are based on TED's search results as outputted by the scraper and TED's results are highly relevant to the query, we noticed that TED-curated links are ranked higher than any other page. Another interesting observation is that the top 5 results of Faiss are included in TED's first page results.

As for the time to execute, Faiss took an average of of 5.868s to execute embedding generation, index and search

- 1st run: 6.0431 seconds
- 2nd: 5.833 seconds
- 3rd: 5.6734 seconds

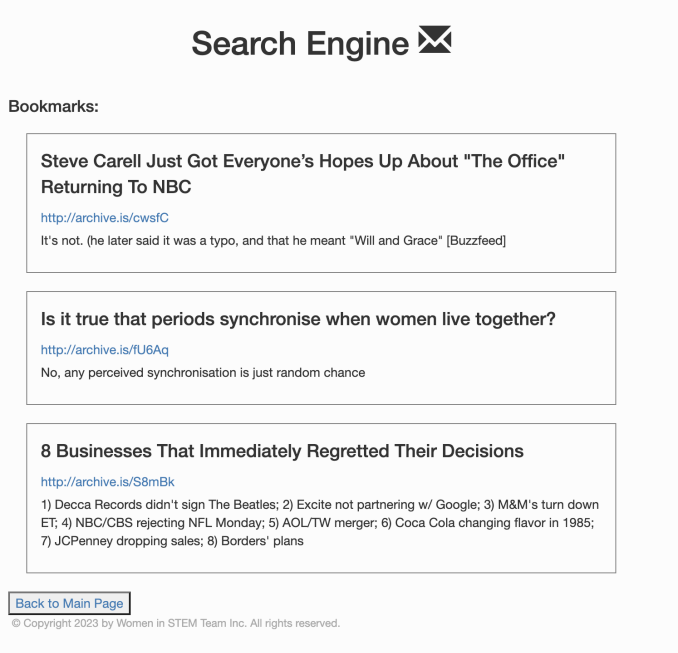
- 4th: 6.0975 seconds
- 5th: 5.7032 seconds

4. Display

The web interface, accessed by the index.html file, displays a search bar and slots for results to show. The user can enter a search term into the search bar and press the search button. The HTML and Javascript in index.html and script.js work together to call the performSearch() function, which accesses the back-end scraper and indexing functions. Then the search results from the TED website will show up in the results slots.

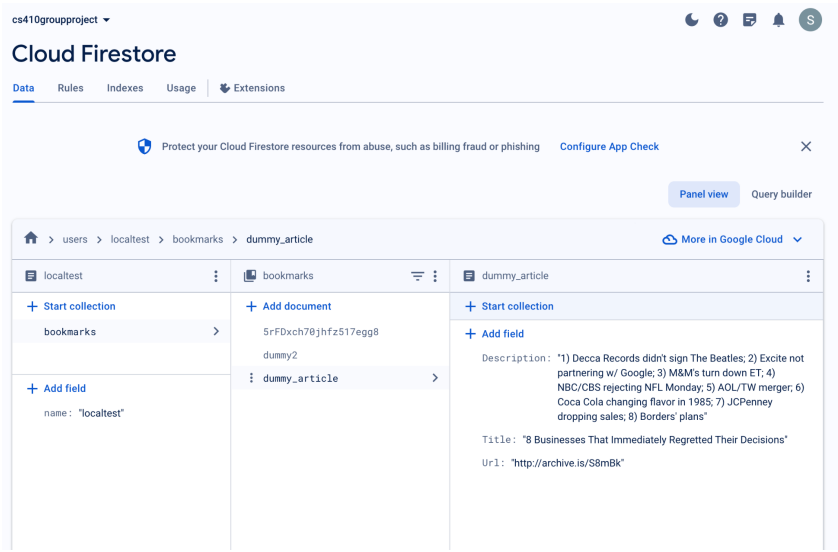
5. Bookmark

The extension comes with a bookmark functionality where users can bookmark TED search results that they find helpful and want to access again. Below each search result is an ‘Add Bookmark’ feature which users can bookmark individual search results after clicking the buttons. At the bottom of index.html, there is a ‘View Bookmarks’ button, which will direct the user to view_bookmarks.html which displays all the TED search results the user has bookmarked. Here’s a sample view of the View Bookmarks page:



These bookmarks are stored in Firestore (a NoSQL database hosted on Firebase). Whenever a user tries to bookmark a search result, the extension will connect to Firestore and run docRef.add(result) to store the specific result in the database. Similarly, whenever a user tries to load the ‘View Bookmarks’ page, the extension will run docRef.get() to fetch all the bookmarks stored in the database and display them on

the front end of the extension. These scenarios leverage Promise functions to handle asynchronous operations.



Installation and Use

1. Download and setup steps
    a. Install Google Chrome browser
    b. Git clone https://github.com/yejinzai/CS410_project
    c. Install Python
    d. Open a terminal window in the project folder and execute:
        i. pip install -r requirements.txt
        ii. Npm install firebase
        iii. npm install -g live-server
2. Run servers
    a. In a terminal window in the project folder:
        i. Flask run (side note: you may also be able to use: python app.py)
        ii. In a separate terminal execute: (for bookmarks)
            1. live-server .
                a. note: if this command causes an error about scripts being disabled, run a terminal as administrator and then execute: Set-ExecutionPolicy Unrestricted
3. Install the extension in your Google Chrome.
    a. In a Chrome browser, go to chrome://extensions/
    b. enable Developer Mode using the toggle switch

    c. click on "Load Unpacked" and select the directory where you downloaded this Github repo
4. Use the web interface
    a. Open index.html in your Google Chrome browser
    b. Enter a search term in the text box
5. Bookmark
    a. After performing a search, click the box next to each result to add it to bookmarks
    b. Follow the Bookmarks button on the bottom of the page to view bookmarks

Team Contribution

Kathleen Barta

Kathleen created the front end of the application, creating the html and javascript for the extension. Kathleen also connected the front and back end using Javascript, after Kris and Payel had completed the scraping and indexing, incorporating them into the application so that the application used those back-end functions and displayed the results to the webpage. Kathleen also put together the documentation and presentation for the video portion and recorded the video presentation. She finally made sure all parts of the project were available on the Github.

Christine Zhou

Christine created the display model for the front end of index.html with styling. And with the script.js within the performSearch() function to connect the client side and the server side. Each textbox will display pieces of information related to the topics (Title, URL, and general description). And also polished the web pages to make them look consistently tidy and organized.

Sharon Lin

Sharon coded the frontend and backend of the bookmark feature for the extension. Sharon also set up the database on Firebase and integrated it with the extension to ensure successful operation of the bookmark functionality.

Kris Kamaei

Kris took care of the backend codes for the index and search, as well as for the integration of all backend pieces. Kris created the initial draft of the documentation and ensured the team adhered to internal due dates for deliverables, helped team members when blocked, and provided overall support.

Payel Chakraborty

Payel did the analysis of Web scraping, researched all techniques to do static/dynamic scraping, talked to TA to get guidance as to how to do successful scraping execution, and then created the code for scraping TED.com.