# ExpertSearch v2.0

## Table of contents



## Video Presentation

**Software usage tutorial presentation**

MacOS To open in a new tab: Cmd ⌘ + Click

Windows To open in a new tab: Cmd ^ + Click

## Using the ExpertSearch v2.0 Webapp

---

⚠️

The team put all emphasis on the NLP, text retrieval, text mining techniques and aspects for faculty search. The webapp is the platform to display the outcomes of the implementation while putting everything together for demonstration purposes. However, making the webapp perfect in terms of end-to-end best user experience was not part of the goal for this project. The webapp still has got many improvement areas in terms of UX, UI display, communication, request response, industry standards and completeness which could be a separate project by itself. While developing the ExpertSearch v2.0 we considered developing the prototype and leaving the opportunity to build things on top of this.

---

**Using Search Feature**

1. Launch the app in your Chrome browser using the Demo URL: http://34.231.70.1:8095/

2. The ExpertSearch v2.0 Home Page should be displayed.
3. Now try searching a faculty by entering queries like:
    - Names can be provided such as Matt Caesar, Cheng Zhai, John Hart etc.
    - Location can be provided such as Illinois, Utah etc
    - Search queries can be provided such as deep learning, text information, visualization etc
4. ExpertSearch v2.0 should display search results along with the faculty attributes such as Name, Department Name, University Name, Areas of Interest, Phone number, Email and Location.
5. Few of the faculty attributes will be displayed as search results. A user can perform action on them such as clicking the email to open the local outlook composer, or clicking location to opt in for details from Google Maps etc.

**Using Admin Feature**

1. Go to the Admin interface using the settings buttons. Demo URL: http://34.231.70.1:8095/
2. Admin interface is a way for admin users (currently open for all users) to add universities, department and faculty urls for the system to be able to append and store additional faculty data to its database. This will result in broadening the search results and data availability. This is also a continuous process to enrich the system with much more data as they are explored.
3. User can provide either university name or university url or department url. User can also specify University Name and Department name together in the search area.
4. The ExpertSearch v2.0 system would asynchronously start crawling and scraping data on the backend system and would also prompt the user that the data will be added through a background process eventually.
5. As this is background update, user may come back after a while and search data related to newly entered university and the search results should get displayed given the background process has completed.
6. If a faculty is already present in the system, the system would not insert a duplicate record of the same faculty and will silently ignore the faculty in the process.

Back to top

# Overview

The ExpertSearch v2.0 application is a faculty search web application that uses NLP and Text Processing (Retrieval and Mining) abilities leveraging the in-built python NLP libraries. The ExpertSearch v2.0 application was build on top the existing ExpertSearch web application [1] that has features such as faculty search, filtering based search, displaying search results (with options to open faculty bio pages, emailing, location info), pagination etc.
As a team, we did a deep analysis of the current ExpertSearch capabilities and found several deficiencies that can be addressed to make it a better search system. The deficiencies include lack of accuracy in the search results, lack of relevant search results and inconsistencies in the search results, older and out of support python library usages, undocumented code and repo etc.
These deficiencies can be addressed using the right text retrieval and text mining techniques along with some good practices that will improve the overall search experience in the ExpertSearch system.
Our team was involved in implementing following features as we forked the existing ExpertSearch System and added/improved core-functionalities that we built on top of it. Below are the core functionalities that we added/improved on the existing ExpertSearch thereby calling it **ExpertSearch v2.0**:

▼ Converting unstructured dataset to structured dataset. Click to learn more

The new ExpertSearch v2.0 system can scrap faulty pages. By using text retrieval techniques, it extracts structured data such as Name, email, department name, university name, phone number, email, location, areas of interests etc. and saves in the database along with the scraped bio-data. Whenever a user enters a search query, along with the optional filter used with the search query, the system grabs the bio-data from the database and ranks them using ElasticSearch Indexing ranking algorithm. After the ranking process is run, the system grabs the ranked data and gets corresponding faculty attributes from the structured database entry for the faculties. This makes the new system much more organized with data while maintaining consistency in displaying search results with uniformity in data. The existing ExpertSearch system doesn't have the structured data implementation and search results displays and very inconsistent.

**Unstructured dataset in existing ExpertSearch**


**Structured dataset generated from unstructured data in new ExpertSearchv2.0**


▼ Topics extraction (e.g., areas of interest) for faculties and display in search results. Click to learn more.

The new ExpertSearch v2.0 implemented the innovative feature of displaying areas-of-interest for each faculty in the search results. This was achieved by performing topic mining techniques on the faculty bio-data to extract most relevant topics. The areas of interests field, which is being displayed in the search results of faculty, is the outcome of this innovative approach. The existing ExpertSearch application doesn't have this feature.

**No topic Modeling in existing ExpertSearch**

**Topic Modeling in ExpertSearchv2.0**

```
# Create Corpus - # Term Document Frequency
corpus = [id2word.doc2bow(tokens)]

# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=1,
                                            random_state=20,
                                            update_every=1,
                                            chunksize=1,
                                            passes=50,
                                            alpha='auto',
                                            per_word_topics=True)

dirname = os.path.dirname(__file__)
lda_dataset = os.path.join(dirname, "../../../lib/lda/lda-trained-dataset")

try:
    lda_model.load(lda_dataset)

except:
    lda_model.save(lda_dataset)
```

▼ Architected admin interface for auto crawling and scraping faculty pages asynchronously. Click to learn more.

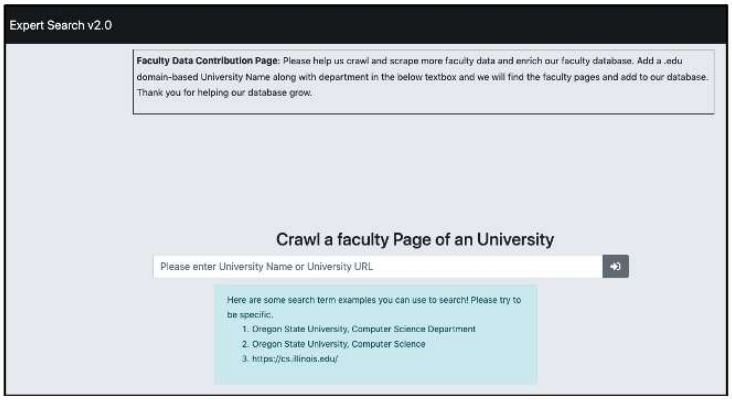
In the new ExpertSearch v2.0 system the admin interface has been reengineered with many improvements and feature upgrades. Admin interface is primarily responsible to receive university input from user to parse, crawl, scrape the data and insert structured data to database. Below are few major upgrades in the Admin interface:

- User can now provide either university name or university url or department url. User can also specify University Name and Department name together.
- The Admin interface receives input from user and asynchronously responds to user that the database will be updated eventually. This is `eventual consistency` model. If user closes browser, the backend server will still be processing the crawling and scraping threads.
- User can keep entering multiple requests one after another and system will eventually process one after another in the background. There is no dependency with browser session. Although the system overall performance and security loopholes are out of scope for this project.
- As soon system extracts bio-data for the faculty url, it then extracts structured data using text retrieval and topic mining techniques and saves in database as structured data.

The existing ExpertSearch system doesn't have these robust, user-friendly functionalities. The existing system doesn't also have mechanism to extract structured data from unstructured dataset.

**No Crawling feature in the existing ExpertSearch**

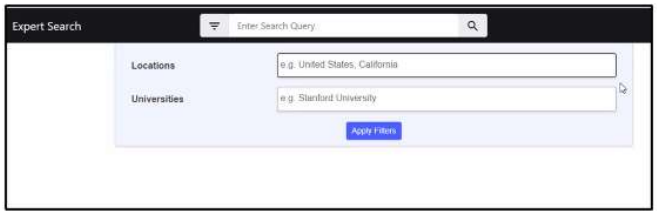
**Interactive Crawling feature in the ExpertSearchv2.0**




▼ Added more filter criteria. Click to learn more.

ExpertSearch v2.0 maintains structured data set in database. Based on search query with filters we can retrieve all saved bio-data matching the filters and apply elasticsearch ranking to only the filtered bio-data set for a specific query. This is how the new system able to offer more filter criterion as compared to the existing ExpertSearch system

Existing ExpertSearch with lesser filter options

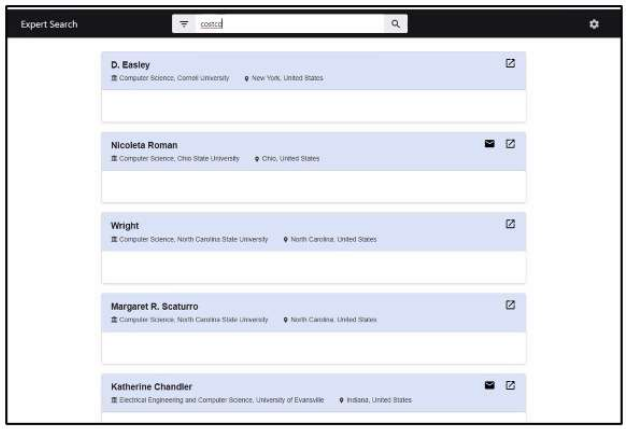



New ExpertSearchv2.0 with more filter options

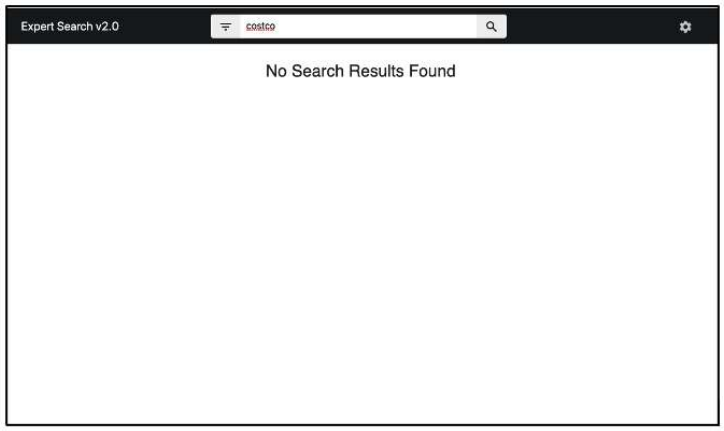

▼ Improved search results with consistent display of the faculty attributes. Click to learn more.

New ExpertSearch v2.0 leverages the structured data to display faculty attributes in the search results display page. The old ExpertSearch did runtime operations which A. makes the system slower for heavy text retrieval techniques and B. missed few modern text retrieval techniques for extracting fields such as department name, phone number etc. We leveraged key browser provided information in html elements such as "title" which mostly provides unique info. We also improved the regex package for extracting accurate phone numbers, email etc. Since these operations are done during crawling and scraping and saved into database as structured data, hence during actual query based on the elasticsearch ranking results, the data is fetched from database. The overall improved process resulted in improved search results and faculty attributes display with no missing information thus improving consistency.

Existing ExpertSearch returns irrelevant search result on no query term match such as “costco”




New ExpertSearchv2.0 doesn’t show irrelevant search result on no query term match such as “costco”

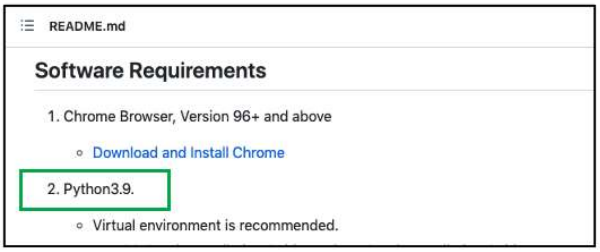



▼ Major System upgrade from Python2.7 to Python3.9. Click to learn more.

New ExpertSearch v2.0 is build on latest Python3.9 and dependent packages compared to the existing ExpertSearch that is build on old and out of support Python2.7. With that said the team went through many research and exploration phases as few of the NLP / Text Processing libraries from old system aren't supported in Python3.9 version. New and modern standard libraries were tested and adopted (such as nltk, gensim etc.) and then engineered to fit the logic of indexing, ranking, scoring, topic mining and text retrieval techniques in the new ExpertSearch v2.0 system. In summary, all capabilities of existing ExpertSearch system have been covered by ExpertSearch v2.0 along with additional features like new set of Python3.9 libraries which by itself is a great achievement.

**Existing ExpertSearch runs on Python2.7**


**New ExpertSearch v2.0 runs on Python3.9**




▼ Documentation and Artifacts for future development. Click to learn more

The new ExpertSearch v2.0 comes with well documented modules, functions and modularized codes that overall improves the readability. There are also design artifacts in terms of workflow diagrams that were generated for users to understand the features and functionality of the code. This will encourage better participation for future development. When compared to the existing ExpertSearch application, the existing web application has almost zero documentation and artifacts which made it very difficult for code analysis, code re-usability and increased the overall development window for enhancements. Below snapshot demonstrates a good comparison between the existing and new ExpertSearch.

**Existing ExpertSearch with almost no documentation**

README.md

## ExpertSearch

To run the code, run the following command from `ExpertSearch` (tested with Python2.7 on MacOS and Linux):

```
gunicorn server:app -b 127.0.0.1:8095
```

The site should be available at http://localhost:8095/

**New ExpertSearch2.0 with comprehensive documentation along with improved functionality**

README.md

## ExpertSearch v2.0

### Video Presentation

Software usage tutorial presentation

MacOS To open in a new tab: Cmd ⌘ + Click

Windows To open in a new tab: Cmd ^ + Click

### Overview

The existing ExpertSearch web application (https://github.com/CS410Assignments/ExpertSearch) has features such as faculty search, filtering based search, displaying search results (with options to open faculty bio pages, emailing, location info), pagination etc. As a team, we did a deep analysis of the current ExpertSearch capabilities and found several deficiencies that can be addressed to make it a better search system. The deficiencies include lack of accuracy in the search results, lack of relevant search results and inconsistencies in the search results. These deficiencies can be addressed using the right text retrieval and text mining techniques that will improve the overall search experience in the ExpertSearch system. Our team involved in implementing following features as we fork the existing ExpertSearch System and added/improved core-functionalities as we build on top of it. Below are the core functionalities that we added on the existing ExpertSearch and we called it ExpertSearchv2.0:

- ▶ Converting unstructured dataset to structured dataset. Click to learn more.
- ▶ Topics extraction (e.g., areas of interest) for faculties and display in search results. Click to learn more.
- ▶ Rearchitected admin interface for auto crawling amd scraping faculty pages async. Click to learn more.
- ▶ Added more filter criteria. Click to learn more.
- ▶ Improved search results with consistant display of the faculty attributes. Click to learn more.
- ▶ Major System upgrade from Python2.7 to Python3.9. Click to learn more.
- ▶ Documentation and Artifacts for future development. Click to learn more

### Hardware Requirements

1. Modern Operating System [Minimum]
   Linux or MacOS **[Recommended]**
2. x86 64-bit CPU [Minimum]

```python
class KLDivergenceRanker(metapy.index.RankingFunction):
    def __init__(self, cfg_file):
        self.forward_idx = metapy.index.make_forward_index(cfg_file)
        self.ranker = metapy.index.JelinekMercer(0.667)
        super(KLDivergenceRanker, self).__init__()

    def score(self, idx, query, nd):
        ql = len(query.content().split())

        k = max(50 - 3 * ql, 3)
        k = min(k, 50)

        kl = metapy.index.KLDivergencePRF(self.forward_idx, self.ranker, 0.78239, 0.64786911, int(k))
        return kl.score(idx, query, nd)
```

```python
class Ranker:

    def __init__(self, corpus: list):
        """
        Ranker class to do ranking of docs on Corpus
        :param corpus:
        """
        self.corpus = corpus
        self.logger = logging.getLogger('my_module_name').setLevel(logging.WARNING)

    def score(self,  query: str, n: int = 10):
        """
        Ranks list of string (docs) based on the query string using BM25 algorithm.
        :param query: the query string aginst which the scoring will be done
        :param n: no of documents to return based on the ranking
        :return: Returns a list of ids that are ranked as per the bm25 logic.
        """
```

Back to top

# Technologies

Below are the main technologies that were used to build ExpertSearch v2.0

- Python3.9
- Modern text Retrieval and Text Mining techniques
- Webpages crawling and scraping
- NLP Libraries - NLTK, Gensim, Spacy
- Redis Cluster
- Elastic Search
- Sqlite3 Database
- Web technologies like HTML, CSS, JQuery
- Flask based web server

# Hardware Requirements

1. Modern Operating System [Minimum]
   Linux or MacOS **[Recommended]**

2. x86 64-bit CPU [Minimum]
   x86 64-bit CPU Multi Core **[Recommended]**

3. 16 GB RAM [Minimum]
   32 GB RAM **[Recommended]**

4. 4 GB free disk space [Minimum]
   8 GB free disk space **[Recommended]**

Back to top

## Access and Permission Requirements

1. Access to the terminal window

2. A user with admin-level privileges

Back to top

## Software Requirements

1. Chrome Browser, Version 96+ and above

   - Download and Install Chrome on MacOS
   - Download and Install Chrome on Linux

2. Python3.9.

   - Virtual environment is recommended.
   - MacOS Conda Installation Guide or Linux Conda Installation Guide
   - Managing Conda - Install python virtual environment using Conda

   Run the below command in virtual environment and Make sure you have installed Python3.9.X.

   ```
   python --version
   ```

3. Java Runtime Environment (OpenJDK JRE)

   - Linux/MacOS OpenJDK Installation Guide

   Run the below command in virtual environment and make sure you have installed Python3.9.X.

   ```
   python --version
   ```

4. `pip` package installed in python3.9 virtual environment - *should be defaulted with Python3.9*
    - To install pip on your virtual environment run below command

```
conda install pip
```

5. git cli tool
    - Install git
6. `make` tool

Back to top

## Setup

1. ElasticSearch Setup

   Installing ElasticSearch on MacOS or Linux

2. Redis Setup

   Installing Redis on MacOS or Linux

```
# get the stable version of redis
wget http://download.redis.io/redis-stable.tar.gz

# extract
tar xvzf redis-stable.tar.gz
cd redis-stable

# install
make
make install
```

3. Using git clone the repository to a path

```
cd <desired path where you want to download the project>
git clone https://github.com/sudiptobilu/CourseProject.git
cd CourseProject
```

4. Set `GOOGLE_API_KEY` Environment Variable
   Click Here to get the Google API Key from the CMT Abstract section (Make sure you are in reviewer role.

Credentials required). You can also use your own Google API Key if you have any.
Add the below line in your `~/.basrc` or `~/.bash_profile`

```
export GOOGLE_API_KEY=<Add Google api Key from CMT Abstract section>
```

Also create a `.env` file in project root directory of project and add the GOOGLE_API_KEY. Run the below commands

```
cd <path to CourseProject repo>

echo "GOOGLE_API_KEY='<Add Google api Key from CMT Abstract section>'" > .env
```

5. Set `PYTHONPATH` Environment Variable
   Add the below line in your `~/.basrc` or `~/.bash_profile`

```
export PYTHONPATH="<path to CourseProject repo>"
```

   Also create a `.env` file in project root directory of project and add the PYTHONPATH. Run the below commands

```
cd <path to CourseProject repo>

echo "PYTHONPATH='<path to CourseProject repo>'" > .env
```

6. Source the `~/.basrc` or `~/.bash_profile` on your terminal

```
source ~/.basrc

# or

source ~/.bash_profile
```

7. Switch to the Python3.9 virtual environment
   If using Conda,

```
# TIP: Show all conda environment
conda env list
```

```
# pick the Python3.9 environment name from the above output. Then run
conda activate <python3.9 virtual environment name>
```

8. Install the project requirements file on Python3.9 virtual environment

```
pip install -r requirements.txt
```

Back to top

## Deploy

1. On a separate terminal, Start ElasticSearch Service (preferably on a screen session)

   For MacOS or Linux

```
cd <path to elasticsearch>
./bin/elasticsearch
```

2. On a separate terminal, Start Redis Server (preferably on a screen session)

   For MacOS or Linux

```
cd <path to redis-stable>
redis-server
```

3. Make sure you are on the Python3.9 environment.

```
python --version
```

4. On a separate terminal, from the project directory, Run the redis `crawler-worker` (preferably on a screen session)

```
# go to project directory root level
cd <path to CourseProject repo>

rq worker crawler-worker
```

5. On a separate terminal, from the project directory, Launch the ExpertSearch v2.0 server (preferably on a screen session)

```
# go to project directory root level
cd <path to CourseProject repo>

python apps/frontend/server.py
```

6. Open Chrome browser and browse the below url

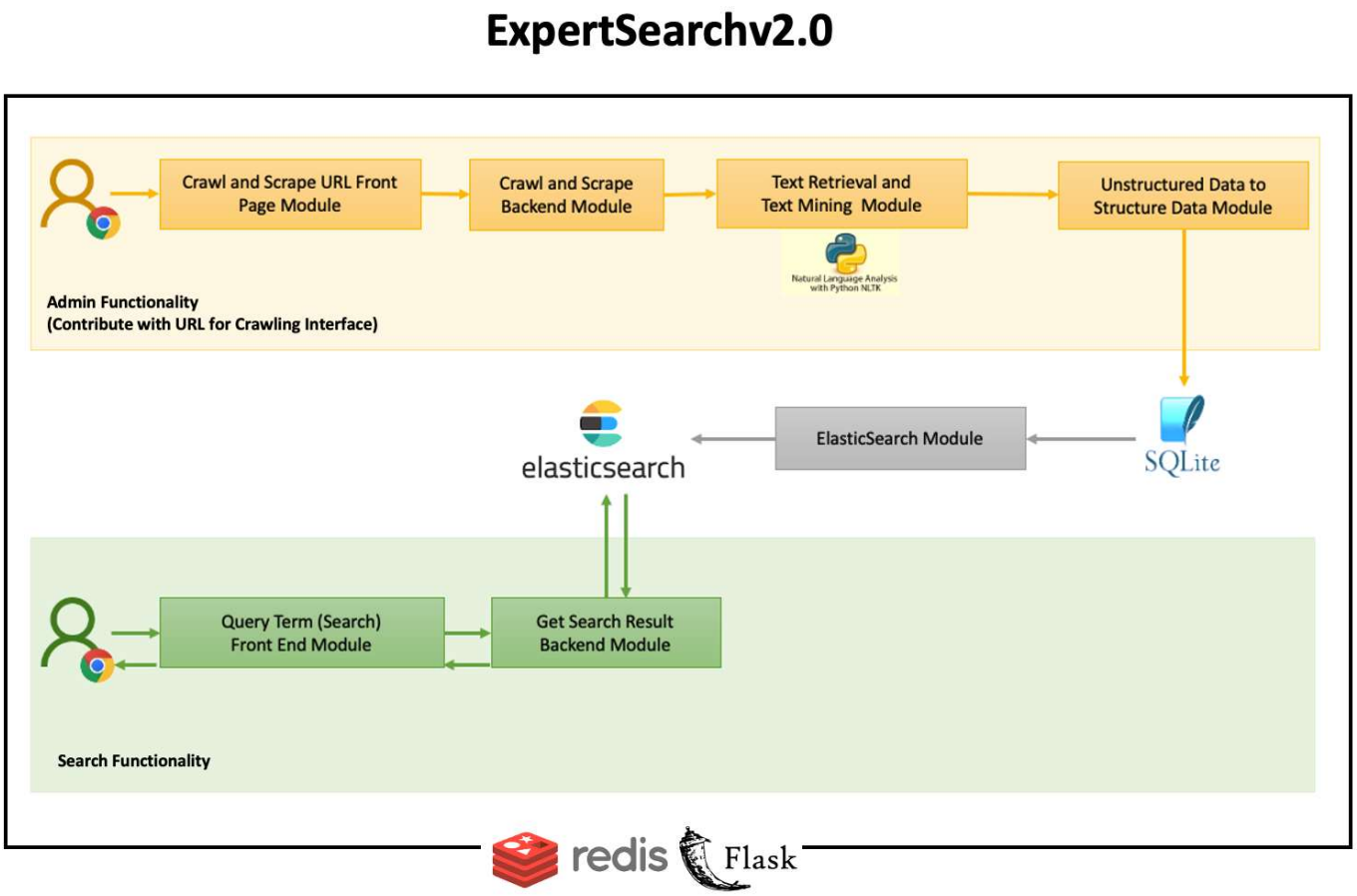
```
http://localhost:8095

http://<localhost_ip>:8095
```

7. The browser should display ExpertSearch v2.0 search application

! A comprehensive software usage video presentation is also available. Click Here

Back to top

## Implementation Details



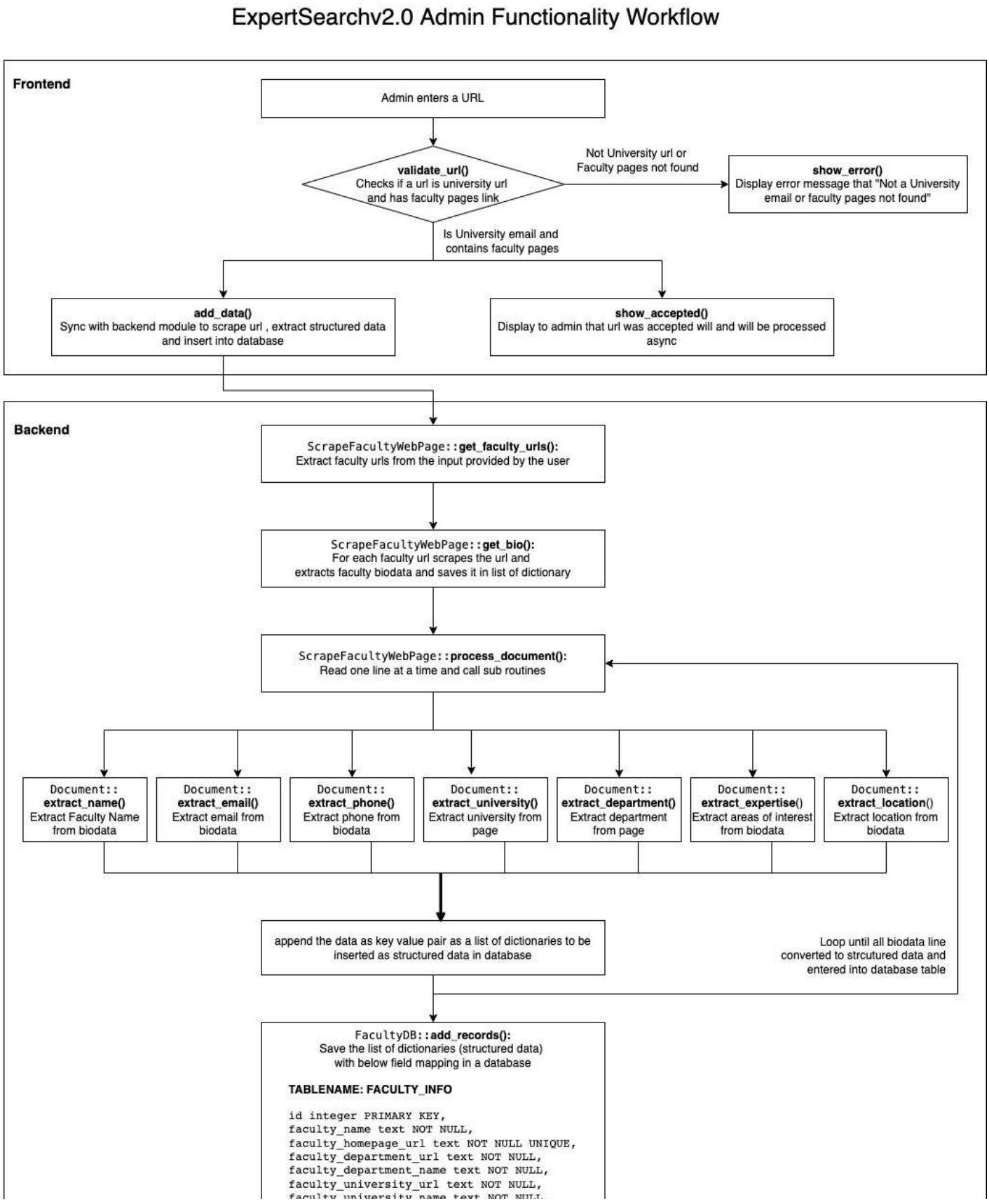
📌📌

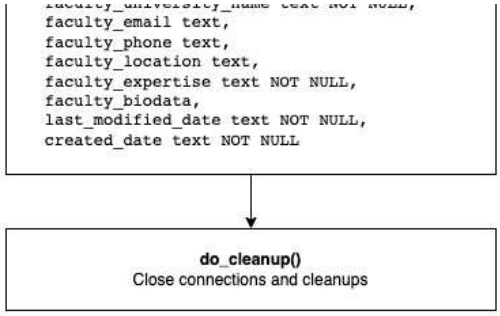
**ExpertSearch v2.0 Admin Functionality (Contribute with URL for Crawling Interface)**

- Admin interface is primarily responsible for receiving university/department/faculty name or urls as input from user to parse, crawl, scrape data, retrieve fields using various text mining approaches and insert structured data to database.
- The purpose of Admin is to add more data from admin provided urls to broaden the search results and add more data availability. This is also a continuous process to enrich the system with more and more data as they are available or explored.
- For admin functionality the front end file is located at web/templates/admin.html
- User provides either university name or university url or department url. User can also specify University Name and Department name together.
- The Admin interface receives input from user and asynchronously responds to user that the database will be updated eventually. This is `eventual consistancy` model. If user closes browser, the backend server will still be processing the crawling and scraping.
- For this we have two instances of server running at the front end.
    1. apps/frontend/server.py::crawl() that accepts university crawl/scrape requests from the user and lets the user know that it will eventually process the data.
    2. A redis instance runs that queues the actual crawl requests and passes the requests to TODO.
- If user passed a name and not an url, then we are using Google API service apps/backend/api/googleapi.py to extract the url from university and department name.
- Once the URL is extracted, the crawler apps/frontend/crawler/crawler.py crawls and finds all the faculty home page urls.
- The faculty homepage urls are then provided to apps/frontend/crawler/faculty_url_scrapper.py for scraping the data and saving the data into database as structured data. The `faculty_url_scrapper` is an orchestration module that performs multiple steps for extracting structured data from the unstructured data and saving it in database. Below are few major tasks it performs:
    - Once the faculty data is scraped, it is then passed to apps/backend/utils/document.py. The `Document` class has then calls the text retrieval and text mining extraction functions to extract Faculty Name, Faculty Contact Number, email, department name, university name, location, areas of interests etc.
        - Phone number, Email is extracted using regex technique
        - Department name, University Name and Faculty Name are extracted from html `title` element from their corresponding urls page source code
        - Location data is extracted using the Google API Service
        - Faculty area of interests are extracted using the Gensim LDA Topic mining method (Text Mining technique)
    - At this stage we have both A. the structured data and also B. Scraped data (as unstructured data) for each faculty
    - Both structured data along with the raw bio-data are then inserted to data/sqlite3/faculty.db database using the class apps/backend/utils/facultydb.py::add_records()

‼ The entire workflow and code discussed above is all new work in the ExpertSearch v2.0 that has been done. Tasks involved adapting new libraries for Python3.9, explorations, PoCs, and then designing an effective workflow and implementing it.

▼ Click to See the Workflow Diagram of Admin Functionality






---

📌📌

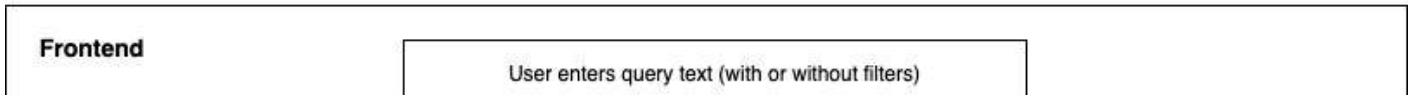
**ExpertSearch v2.0 Search Functionality**

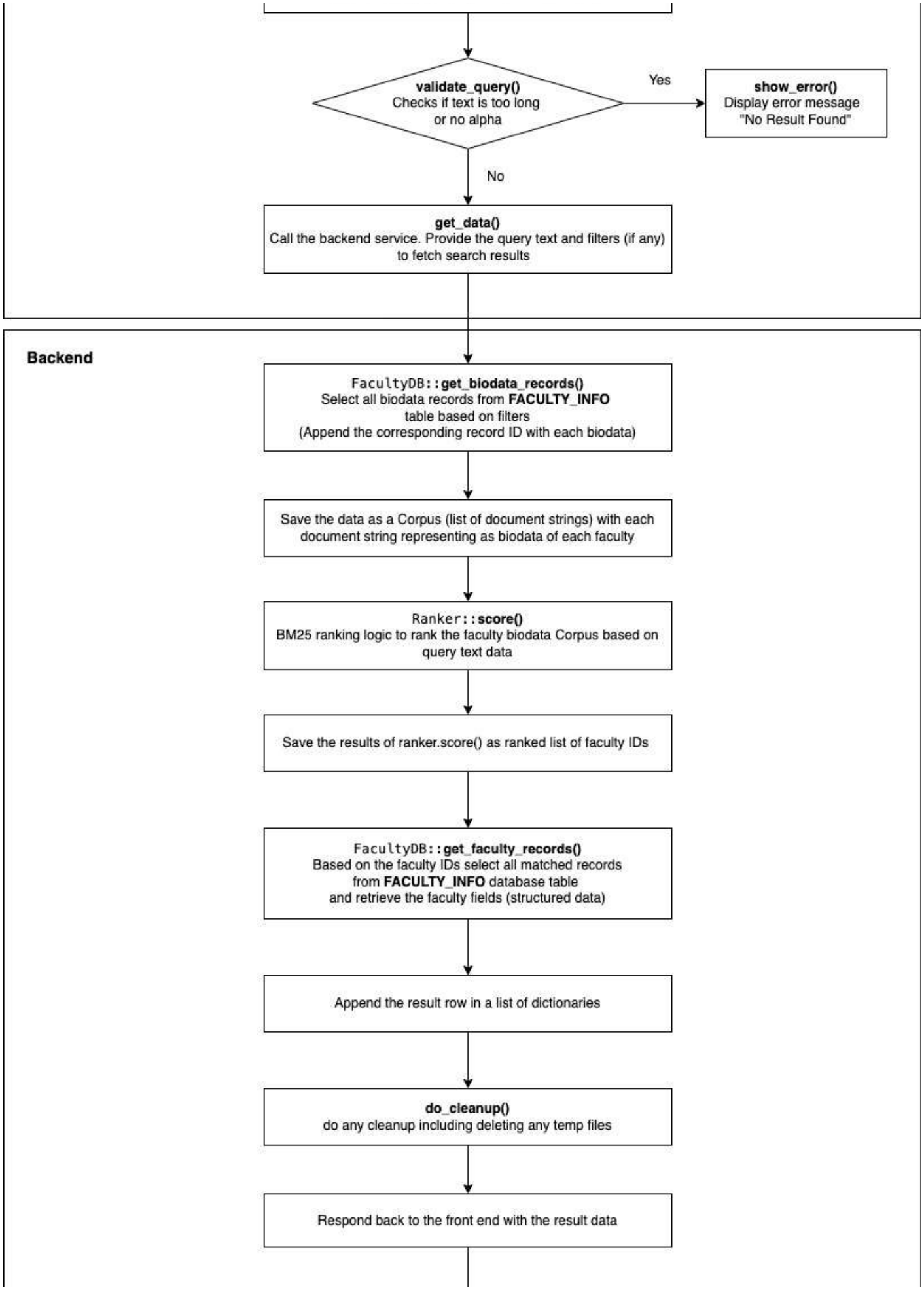
- For search functionality the front end file is located at web/templates/index.html
- User enters query to the html file (user can provide filters too) which is passed to the backend Flask server apps/frontend/server.py::search() as a http request
- The server receives the query string and calls apps/frontend/server.py::search()
- The `search` in turn calls an orchestration function apps/backend/api/search.py::get_search_results()
- The `get_search_results` is an orchestration function and calls different backend systems to retrieve the data
    - The call first goes to apps/backend/utils/facultydb.py::get_biodata_records() and grabs all scraped bio-data stored in a database table column along with corresponding structured data id.
    - Then the corpus data is passed to apps/backend/utils/ranker.py::score() to score the corpus bio-data dataset based on search query. The function used elasticsearch ranking abilities to rank corpus documents.
    - Once ranking is done the corresponding structured data ids were returned as a ranked list of faculty ids
    - The ranked ids were taken and passed to apps/backend/utils/facultydb.py::get_faculty_records() to get the structured data from database
- The result's dataset is now a structured data with key pair values being displayed in the front end accordingly
- The benefit of displaying structured data is consistency in displaying results and all the attributes and allowing certain user actions on them. (for e.g. send email, explore location etc.)

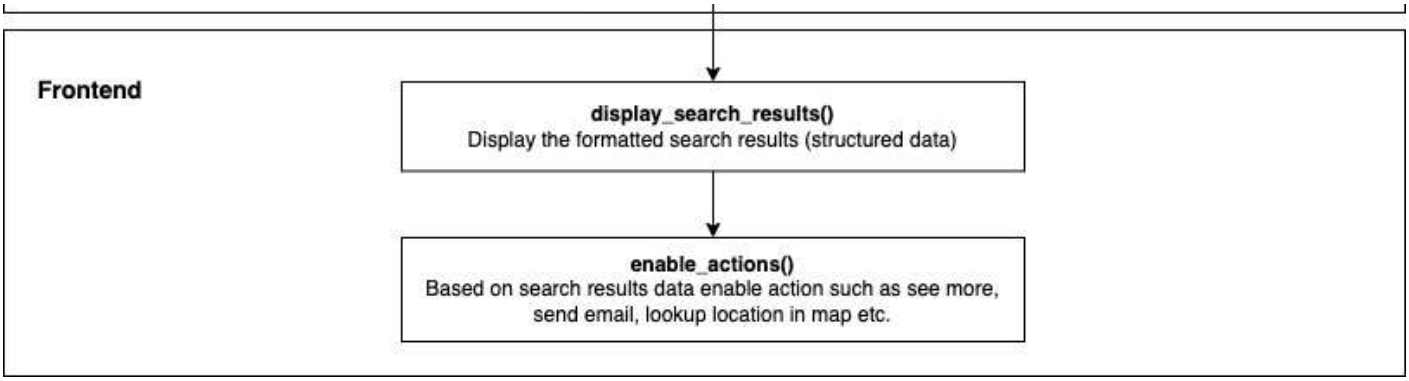
‼️ The entire workflow and code discussed above is all new work in the ExpertSearch v2.0 that has been done. Tasks involved adapting new libraries for Python3.9, explorations, PoCs, and then designing an effective workflow and implementing it.

▼ Click to See the Workflow Diagram of Search Functionality

ExpertSearchv2.0 Search Functionality Workflow




validate_query()
Checks if text is too long
or no alpha
Yes
show_error()
Display error message
"No Result Found"
No
get_data()
Call the backend service. Provide the query text and filters (if any)
to fetch search results
Backend
FacultyDB::get_biodata_records()
Select all biodata records from FACULTY_INFO
table based on filters
(Append the corresponding record ID with each biodata)
Save the data as a Corpus (list of document strings) with each
document string representing as biodata of each faculty
Ranker::score()
BM25 ranking logic to rank the faculty biodata Corpus based on
query text data
Save the results of ranker.score() as ranked list of faculty IDs
FacultyDB::get_faculty_records()
Based on the faculty IDs select all matched records
from FACULTY_INFO database table
and retrieve the faculty fields (structured data)
Append the result row in a list of dictionaries
do_cleanup()
do any cleanup including deleting any temp files
Respond back to the front end with the result data




Back to top

## Project Team Members

| Name | NetID | Email |
|---|---|---|
| Ujjal Saha | ujjals2 | ujjals2@illinois.edu |
| Arnab KarSarkar | arnabk2 | arnabk2@illinois.edu |
| Sudipto Sarkar | sudipto2 | sudipto2@illinois.edu |

Back to top

## User Stories and Contributions

1. **Epic: Crawling and Scraping** *(Contributions 👏 : Arnab KarSarkar, Ujjal Saha)*

   ▼ **User Story:** Crawler Implementation for a given webpage url. Click for Story Details

   In the admin interface when admin inputs an url, this story takes the url as input and scrapes the page and extracts the faculty bio-data. We also implemented intelligent logic in scraper to find right faculty page if the base url has links that led to multiple faculty related pages.

   ▼ **User Story:** Adding admin interface for web page indexing. Click for Story Details

   As our project scope does not include auto crawler features for the entire web, the admin interface we are implementing in ExpertSearch system allows the admin to enter base url of the university and based on valid/invalid university email (different story), the admin interface will fetch the url to the crawler module to scrape faculty data.

   ▼ **User Story:** Displaying accepted/rejected web page based on url. Click for Story Details

   When admin enters the base url, this module will check if the url is a valid university url. If yes, the module forwards the url for crawling and scraping the faculty pages. If not, the module lets the admin know that base url does not belong to a university or no faculty page found.

2. **Epic: Search Experience Enhancement using Text Retrieval Techniques** *(Contributions 👏 : Ujjal Saha, Sudipto Sarkar)*

▼ **User Story:** Build a structured dataset from Unstructured datasets. Click for Story Details

In current ExpertSearch system the faculty data are stored as unstructured data as a file. We are implementing a functionality that will convert the unstructured data to structured data. For e.g., using text mining and text retrieval techniques we are planning to extract fields like Faculty Name, Department, University, Area of Interests, email, phone, etc. and store them in a structured form (either in csv, or database etc.). This will enhance the overall search experience.

▼ **User Story:** Enhance the search experience with relevant search results. Click for Story Details

Based on search input, we will look up all bio-data from structured dataset and implement a elasticsearch ranking. Based on elasticsearch ranking results we will extract corresponding fields from the structured data and display as search results. We will enhance filter based searching feature too, where user can get better accuracy because of structured dataset.

▼ **User Story:** Better consistency in displaying attributes by leveraging structured data. Click for Story Details

The current expert search system does not show contact info (email, etc.) consistently across the search results even if the faculty page does have the data. Our improved scraping and structured data along with improved data display logic will increase the consistency in displaying the fields in search results.

---

3. **Epic: Topic Mining** *(Contributions 👏 : Sudipto Sarkar, Arnab KarSarkar)*

▼ **User Story:** Using text mining methodologies extract interest areas of a faculty. Click for Story Details

Using text mining methods we are planning to generate “Areas of Interests” data from the faculty bio. We are using Trained LDA model, NER taggers and Gensim/NLTK libraries along with its parameter set variations to generate relevant topics.

▼ **User Story:** Display Areas of Interest in the faculty search result. Click for Story Details

We are enhancing the front end of ExpertSearch to display faculty search results along with additional relevant fields such as faculty areas of interest and few more.

---

4. **Epic: Deployment** *(Contributions 👏 : Arnab KarSarkar, Ujjal Saha, Sudipto Sarkar)*

▼ **User Story:** Understand and Install current ExpertSearch System. Click for Story Details

Install and explore the ExpertSearch system and understand the features and functionalities (both frontend and backend). Experiment with code changes etc.

▼ **User Story:** Deploy code into AWS. Click for Story Details

Being web-based framework, we will do our deployments in AWS Cloud and make it public. We will also do a git PR on the existing original ExpertSearch repo. But launching as an improved system and others to validate, we will separately host ExpertSearch v2.0 in AWS.

▼ **User Story:** Validation Exercises. Click for Story Details

As we do development and deployment, we are doing multiple rounds of verification and validation and some will require integrated end-to-end validation steps.

---

5. **Epic: Documentation and Presentation** *(Contributions* 👏 *: Ujjal Saha, Sudipto Sarkar, Arnab KarSarkar)*

   **User Story**: Proposal Documentation

   **User Story**: project Progress Documentation

   **User Story**: Final Project Report Documentation

   **User Story**: Final Project Video Presentation

---

Back to top

## Improvements Areas

- Crawling and Scraping activities can be tracked if implemented as a publisher and subscriber. However, we did not pursue it as it would not add much value to our goal and focus on Text Retrieval and Mining techniques.
- There is huge opportunity to improve the crawling and scraping feature. Each university has their own styling in website url names and page contents. So more we explore more logical scenarios we can add so more pages can be crawled with a generic implementation.
- Many times a faculty has multiple homepage urls and the system. In the main homepage url there could be multiple links and requires more intelligent to identify which page is the main homepage url.
- We have used trained LDA Model which generated better output of topical data. However, future research can be done to check if using a trained model alongside a specialized topic mining with seeded datasets could result is more relevant topic words for a faculty. We also explored GuidedLDA and did not find any improvements either.
- Admin interface repeated entry can be malicious and hence there is a need to implement some sort of restrictions

- Making the webapp perfect in terms of end-to-end best user experience was not part of the goal for this project. The webapp still has got many improvement areas in terms of UX, UI display, communication, request response structure, industry standards, completeness which could be a separate project by itself.
- We were only able to crawl and scrape HTML from US universities. Any international university may or may not work with the current implementation.

## Licensing

The ExpertSearch v2.0 was build upon existing ExpertSearch web application [1] and thus will inherit the original licensing terms and conditions of the original ExpertSearch system.

## Acknowledgements


- Our special thanks to Prof. Cheng Zhai and all the TAs in CS410 Text Information Systems Course for making the course engaging and help with all the queries.
- Many thanks to original creators of existing ExpertSearch web application [1] and letting others build on top of it.
- Also thanks to our open source community contributors for so many contributions in NLP, Text Retrieval and Text Mining based python packages which are effective, efficient and free to use.
- Thanks to our project team members and project reviewers too for wonderful collaboration and feedback and making the project successful.
- Special thanks for University of Illinois - Urbana Champaign for providing students with endless software tools, collaboration mediums and resources such as box, sharepoint, Google Drive, library, zoom, slack, CampusWire and many more. The availability of these tools help online students a lot.


## References


[1] Existing ExpertSearch web application
[2] Coursera - CS410 Text information Systems - Course Project Overview
[3] NLTK API Reference
[4] Gensim API Reference
[5] Topic Modelling in Python with NLTK and Gensim
[6] Beginners Guide to Topic Modeling in Python
[7] BM25 Ranker
[8] Flask + Bootstrap. HTML interface for effortless Python projects
[9] How to Use Redis With Python
[10] Python Elasticsearch Getting Started Guide
[11] AWS Cloud Hosting Service - EC2
[12] Github ReadMe - Basic writing and formatting syntax


Back to top