Course Topic Explorer

CS 410 Project (Free Topic)

Harita Reddy (haritar2), Eric Crawford (ecraw3)

OVERVIEW

The aim of this project was to build a comprehensive search system for students looking for the different courses that US universities are offering. We implemented a search engine with some recommendation features built based on the users' recent search history. The major component of this project is the search feature, which the students can use to get the courses and MOOCs relevant to the keywords they enter. An advanced feature is a simple recommendation system that will recommend suitable programs that the student can apply to get knowledge on the topics they have been searching for.

FUNCTIONALITY

The main functionalities provided by our code to the user are:

1. Search for courses from US universities. This is implemented through the functions in backend/search folder. get_relevant_courses is the function that needs to be called in order to get the relevant course documents for a given keyword query.
2. Get relevant MOOCs on Coursera offered by these universities for the searched keyword. This is implemented through the functions in backend/moocs folder. get_relevant_moocs is the function that needs to be called in order to get the relevant mooc documents for a given keyword query.
3. Recommend specific programs from different universities based on the history of keyword searches. The functions pertaining to this are implemented in backend/recommender. recommend_programs is the function that needs to be called to get the top 3 recommended programs based on the user's search history (last three searches).

We scraping was done once to get all the required courses. The functions are implemented in web_scraping folder.

VIDEO TUTORIAL

https://drive.google.com/file/d/1TIVTnXw6UoFsDHuBOFQaSI5DA8uokhr6/view?usp=sharing

If the above link is not accessible, try https://mediaspace.illinois.edu/media/t/1_kis3mddw

SOFTWARE IMPLEMENTATION DOCUMENTATION

Frontend

The frontend was developed as a single-page application using the Kotlin programming language. Kotlin is a statically typed language that can compile to many platforms like mobile apps, desktop apps, and the web. When targeting the web, the code is transpiled into plain old javascript which can be embedded into a webpage. We used Jetbrains Compose to lay out the UI elements. This library allows you to write modern reactive UI code that targets desktop and web applications, using a statically typed language. We also used the KTOR (https://ktor.io/docs/getting-started-ktor-client.html), web client, to make the necessary web requests to the backend and handle JSON serialization.

Backend Framework

The backend is run via Falcon Python server framework. This framework allows us to serve our data over a restful api that our frontend can call. We initially intended to implement the backend of this application using the Django framework. However, we ran into issues with using Django. Meta does not work with web servers like Django and Flask. There is a known issue with deadlocks in the pybind11 library that meta depends on. We remedied this issue by using the **Falcon** framework (https://falcon.readthedocs.io/en/stable/) instead after investigating similar issues reported by other users.

The backend code is located in the **backend** folder. The important files/folders in this directory are as follows:

moocs: This folder contains all the code that is used for indexing and searching for courses that are moocs

recommender: This folder contains all the code that is used for recommending/ranking the programs based on the search criteria.

search: This folder contains all the code that is used for indexing and searching for courses that are on campus

static: This folder contains all the html and javascript files

dockerfile: The file used to create the docker image

docker-compose.yml: The file used to spin up and run the application

falconapp.py: the main file that runs the backend server. (do to issues with metapy, we prefer running this inside a docker image, which the docker-compose.yml file does for us)

The server exposes several APIs that can be used to retrieve the data. These can be called from any frontend application or manually in a browser or via cURL.

Path	Parameters	Returns
/search	Key: q Type: String Value: The search Term	JSON Array of the on-campus courses
/moocs	Key: q Type: String Value: The search Term	JSON Array of the MOOC's
/recommend	Key(s): q1,q2,q3 Type: String, String, String Value: The last 3 search terms. All 3 terms are required	JSON Array of recommended programs
/main	No parameters	The HTML for the home page

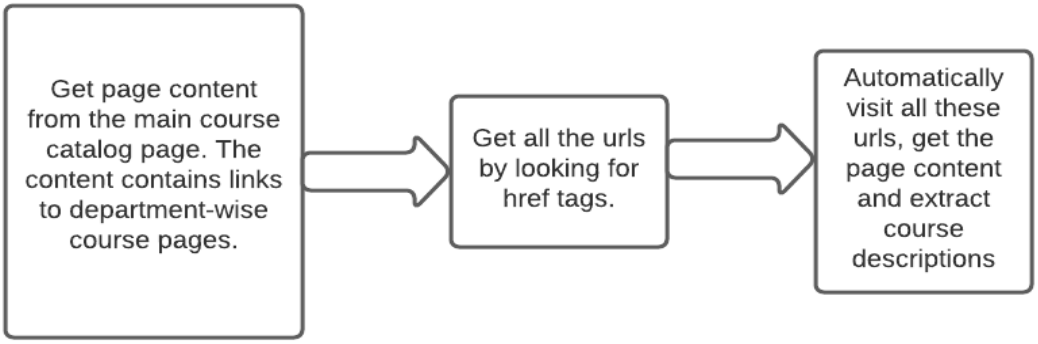
Web Scraping

Our implementation does not do web scraping live because of the constraints on making unnecessary requests to the university websites. We did a **one-time web scraping** to get all the course catalogs and store them into pickle files. The implementation is available in *web_scraping/* folder in the project code.

Course catalogs were scraped from the following university pages:

```
    'CMU': 'http://coursecatalog.web.cmu.edu/coursedescriptions/',

    'UIUC': 'http://catalog.illinois.edu/courses-of-instruction/',

    'GATECH': 'https://catalog.gatech.edu/coursesaz/',

    'UW': 'https://guide.wisc.edu/courses/',

    'UCB': 'http://guide.berkeley.edu/courses/',
    "MIT": 'http://catalog.mit.edu/subjects/',

    "UNC": 'https://catalog.unc.edu/courses/',

    "UCI": 'http://catalogue.uci.edu/allcourses/',

    "CUB": 'https://catalog.colorado.edu/courses-a-z/',

    "NE": 'http://catalog.northeastern.edu/course-descriptions/'
```

We have used Python’s BeautifulSoup and requests packages for scraping. Scraping is done in the following manner:


Get page content from the main course catalog page. The content contains links to department-wise course pages.
Get all the urls by looking for href tags.
Automatically visit all these urls, get the page content and extract course descriptions

class CourseScraper:

The university name to course catalog page mapping is present in web_scraping/config.py.

1. **scrapeAllPages(self):**

Returns: Dictionary with university id (such as ‘UIUC’, ‘GATECH’ etc as key and a list of courses as values. It iterates through the universities listed in the config.py file.

This function calls multiple helper functions in the same class. Some of the functions are listed below:

getSubLinks(self, page_content): Gets all the links (href) from the HTML page content

getPageContent(self, url): Gets the page content given the url

getPageURL(self, university_name): Given the university id, gets the corresponding main course catalog page url

getParentURL(self, url): Gets the parent URL for a given URL. This is useful for appending the href link to the parent URL in order to visit the required page, because the href URL is relative to the parent URL.

The main function runs the scrapeAllPages functions and writes the returned courses and their descriptions to pickle format. The pickle files were converted into dataset form suitable for search implementation.

class MOOCScraper has similar functions.

Search

There are two types of search in the backend, one being the course and search and the other being the MOOC search. Both of this are located in separated folders in the backend.

backend/

search/ Contains the code to search for relevant courses offered by different universities based on keywords.

moocs/ Contains the code to search for relevant MOOCs offered by some universities based on the entered keywords.

The ranker used for the course search and MOOC search implementation is **OkapiBM25** from Metapy.

class Search:

1. **get_relevant_courses(self, input_query, top_k):**

input_query: A string of keywords entered by the user. Eg: ‘spanish colonial rule’

top_k: Number of top relevant documents to be returned. Eg: 5

Returns: List of dictionaries of format {'course_content': ..., 'link': ..., 'university': ...}

For an input query and given top_k value that indicates how many relevant documents should be returned, the function returns the relevant documents in the form of list of dictionaries

class MOOCSearch:

1. **get_relevant_moocs(self, input_query, top_k):**

input_query: A string of keywords entered by the user. Eg: ‘spanish colonial rule’

top_k: Number of top relevant documents to be returned. Eg: 5

Returns: List of dictionaries of format {'course_content': ..., 'link': ..., 'university': ...}

For an input query and given top_k value that indicates how many relevant documents should be returned, the function returns the relevant documents in the form of list of dictionaries

Recommender

The recommender module implements the functions to recommend programs based on the user’s search history. It uses SpaCy, which is Python’s Natural Language Processing package. Specifically, it leverages the en_core_web_lg, which is a trained pipeline for English language and can be used to create a word embedding matrix. We used this trained pipeline because it is difficult to generate word vectors from large corpora. We also used Python’s scikit-learn for computing the cosine similarity between items.

SpaCy: https://spacy.io/

Scikit-Learn: https://scikit-learn.org/stable/

The file program_catalogs.tsv contains the list of programs that the recommender will use.

class Recommender:

Contains all the functions required to recommend programs based on three input strings given by the user. The input strings are either keywords searches that are entered by the user or the description of the top matching courses for the user’s query.

Initialisation

The initialisation step in the class reads the program descriptions from recommender/program_catalogs.tsv and loads SpaCy’s en_core_web_lg to create the required word embedding matrix.

1. **recommend_programs(self, query_sentences=['computer science', 'data mining', 'machine learning']):**

query_sentences: An array of three queries based on previous searches. Eg: ['computer science', 'data mining', 'machine learning'].

Returns: JSON list of the format [{'id': ..., 'program_name': ..., 'university': ..., 'description': ..., 'link': ...}, …] which are basically the top 3 recommended programs based on queries provided

The function calls the recommend(self, query_sentences, top_k) to get the top 3 recommended programs.

2. **recommend(self, query_sentences, top_k):**

query_sentences: An array of three queries based on previous searches. Eg: ['computer science', 'data mining', 'machine learning'].

top_k: Number of top recommended documents to be returned

Returns: A dataframe of top_k recommended documents (in this case, programs). The dataframe contains the columns 'name', 'university', 'description', 'link'.

This functions does the actual computation to get the top_k recommended documents. The computation is done using the concept of *item similarity* with the last three searches. The similarity score is computed using sklearn.metrics.pairwise.cosine_similarity.

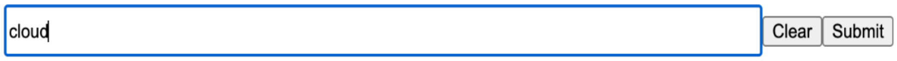
INSTALLATION AND USAGE

There were issues running the server with metapy on plain development machines but we were able to overcome this inside of docker containers, thus, this application will need to be run in docker. The instructions below will get the application server running locally on your computer.

1. Clone the project and enter the project in your terminal.
2. Install Docker if you don't already have it (Docker Installation)
3. Run Docker Desktop and wait for it to finish starting up.
4. Go to the backend folder of this project and run *docker-compose up*. It might take a few minutes for the image to build and the container to be started. Your terminal will prompt you when the server is ready.
5. Go to http://localhost/main and see the site.
6. When you are done using the application, type **ctrl c** on Windows or **control c** on Mac to terminate the docker container. You may also want to completely delete the container and image by following these instructions .

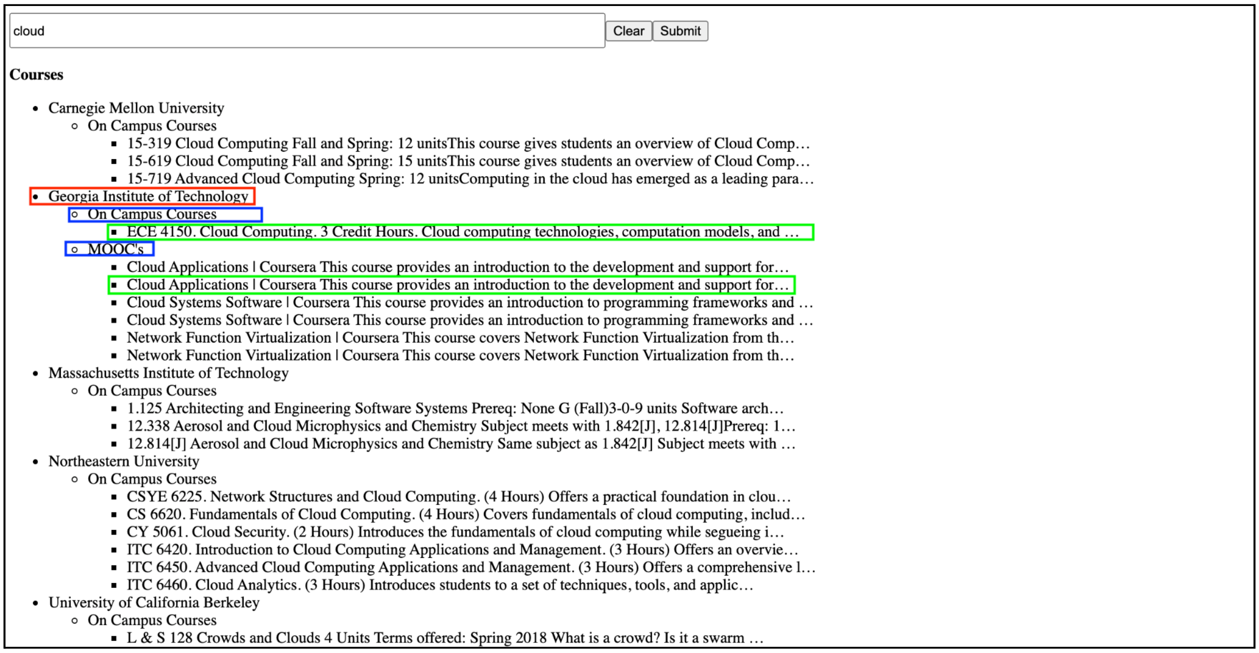
After Opening the Application

After opening http://localhost/main, you will be able to see a search bar where you can enter the search keywords:



After clicking ‘Submit’, you will be given the results of your search. The results will contain a list of universities and the course from that university that match your search criteria. We have highlighted the section break in the image below. It is ordered as follows:

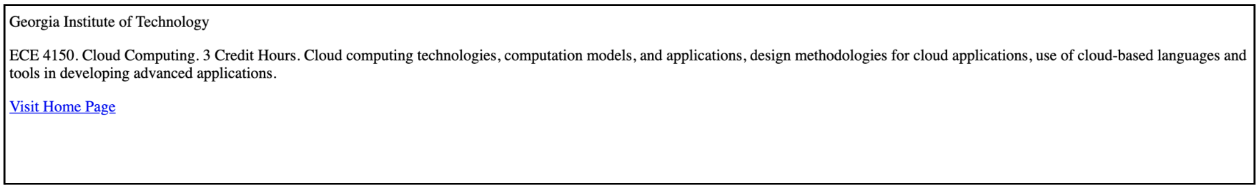
- University: Red
- Is this an on-campus course or MOOC: Blue
- The course description: Green



Courses

- Carnegie Mellon University
  - On Campus Courses
    - 15-319 Cloud Computing Fall and Spring: 12 unitsThis course gives students an overview of Cloud Comp…
    - 15-619 Cloud Computing Fall and Spring: 15 unitsThis course gives students an overview of Cloud Comp…
    - 15-719 Advanced Cloud Computing Spring: 12 unitsComputing in the cloud has emerged as a leading para…
- Georgia Institute of Technology
  - On Campus Courses
    - ECE 4150. Cloud Computing. 3 Credit Hours. Cloud computing technologies, computation models, and …
  - MOOC's
    - Cloud Applications | Coursera This course provides an introduction to the development and support for…
    - Cloud Applications | Coursera This course provides an introduction to the development and support for…
    - Cloud Systems Software | Coursera This course provides an introduction to programming frameworks and …
    - Cloud Systems Software | Coursera This course provides an introduction to programming frameworks and …
    - Network Function Virtualization | Coursera This course covers Network Function Virtualization from th…
    - Network Function Virtualization | Coursera This course covers Network Function Virtualization from th…
- Massachusetts Institute of Technology
  - On Campus Courses
    - 1.125 Architecting and Engineering Software Systems Prereq: None G (Fall)3-0-9 units Software arch…
    - 12.338 Aerosol and Cloud Microphysics and Chemistry Subject meets with 1.842[J], 12.814[J]Prereq: 1…
    - 12.814[J] Aerosol and Cloud Microphysics and Chemistry Same subject as 1.842[J] Subject meets with …
- Northeastern University
  - On Campus Courses
    - CSYE 6225. Network Structures and Cloud Computing. (4 Hours) Offers a practical foundation in clou…
    - CS 6620. Fundamentals of Cloud Computing. (4 Hours) Covers fundamentals of cloud computing, includ…
    - CY 5061. Cloud Security. (2 Hours) Introduces the fundamentals of cloud computing while segueing i…
    - ITC 6420. Introduction to Cloud Computing Applications and Management. (3 Hours) Offers an overvie…
    - ITC 6450. Advanced Cloud Computing Applications and Management. (3 Hours) Offers a comprehensive l…
    - ITC 6460. Cloud Analytics. (3 Hours) Introduces students to a set of techniques, tools, and applic…
- University of California Berkeley
  - On Campus Courses
    - L & S 128 Crowds and Clouds 4 Units Terms offered: Spring 2018 What is a crowd? Is it a swarm …

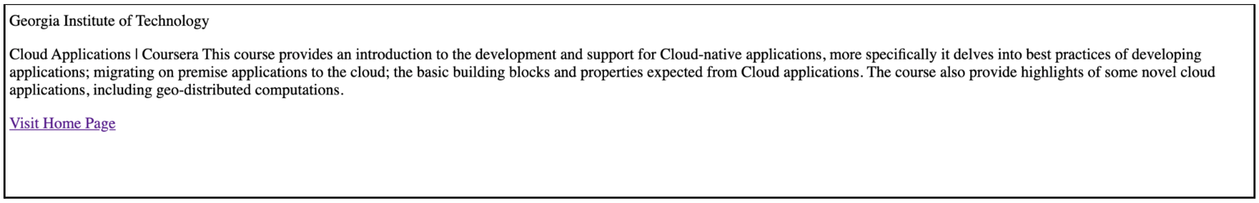
Clicking on any of the course descriptions will give you more details about that course and allow you to go directly to that department or MOOC home page:



Georgia Institute of Technology

ECE 4150. Cloud Computing. 3 Credit Hours. Cloud computing technologies, computation models, and applications, design methodologies for cloud applications, use of cloud-based languages and tools in developing advanced applications.

Visit Home Page

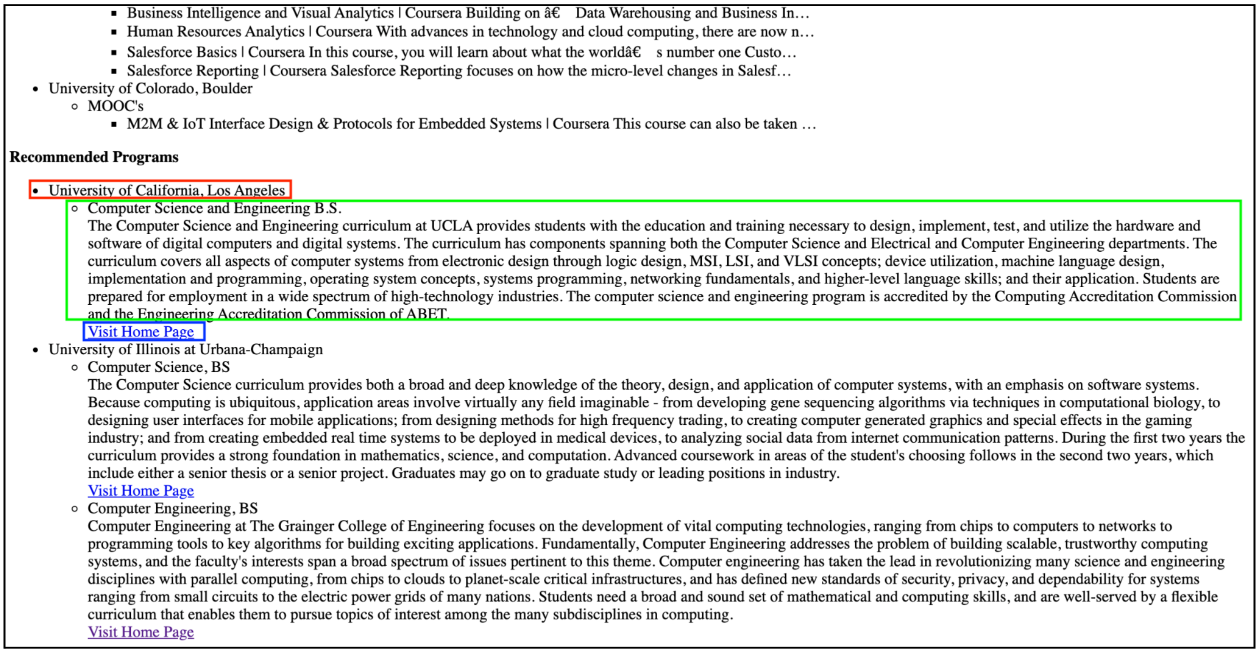


Georgia Institute of Technology

Cloud Applications | Coursera This course provides an introduction to the development and support for Cloud-native applications, more specifically it delves into best practices of developing applications; migrating on premise applications to the cloud; the basic building blocks and properties expected from Cloud applications. The course also provide highlights of some novel cloud applications, including geo-distributed computations.

Visit Home Page

Finally, under the list of courses, there is a list of recommended programs where your search criteria are most likely to be taught at:



- Business Intelligence and Visual Analytics | Coursera Building on â€ Data Warehousing and Business In…
- Human Resources Analytics | Coursera With advances in technology and cloud computing, there are now n…
- Salesforce Basics | Coursera In this course, you will learn about what the worldâ€ s number one Custo…
- Salesforce Reporting | Coursera Salesforce Reporting focuses on how the micro-level changes in Salesf…

- University of Colorado, Boulder
    - MOOC's
        - M2M & IoT Interface Design & Protocols for Embedded Systems | Coursera This course can also be taken …

Recommended Programs

- University of California, Los Angeles
    - Computer Science and Engineering B.S.
      The Computer Science and Engineering curriculum at UCLA provides students with the education and training necessary to design, implement, test, and utilize the hardware and software of digital computers and digital systems. The curriculum has components spanning both the Computer Science and Electrical and Computer Engineering departments. The curriculum covers all aspects of computer systems from electronic design through logic design, MSI, LSI, and VLSI concepts; device utilization, machine language design, implementation and programming, operating system concepts, systems programming, networking fundamentals, and higher-level language skills; and their application. Students are prepared for employment in a wide spectrum of high-technology industries. The computer science and engineering program is accredited by the Computing Accreditation Commission and the Engineering Accreditation Commission of ABET.
      Visit Home Page
- University of Illinois at Urbana-Champaign
    - Computer Science, BS
      The Computer Science curriculum provides both a broad and deep knowledge of the theory, design, and application of computer systems, with an emphasis on software systems. Because computing is ubiquitous, application areas involve virtually any field imaginable - from developing gene sequencing algorithms via techniques in computational biology, to designing user interfaces for mobile applications; from designing methods for high frequency trading, to creating computer generated graphics and special effects in the gaming industry; and from creating embedded real time systems to be deployed in medical devices, to analyzing social data from internet communication patterns. During the first two years the curriculum provides a strong foundation in mathematics, science, and computation. Advanced coursework in areas of the student's choosing follows in the second two years, which include either a senior thesis or a senior project. Graduates may go on to graduate study or leading positions in industry.
      Visit Home Page
    - Computer Engineering, BS
      Computer Engineering at The Grainger College of Engineering focuses on the development of vital computing technologies, ranging from chips to computers to networks to programming tools to key algorithms for building exciting applications. Fundamentally, Computer Engineering addresses the problem of building scalable, trustworthy computing systems, and the faculty's interests span a broad spectrum of issues pertinent to this theme. Computer engineering has taken the lead in revolutionizing many science and engineering disciplines with parallel computing, from chips to clouds to planet-scale critical infrastructures, and has defined new standards of security, privacy, and dependability for systems ranging from small circuits to the electric power grids of many nations. Students need a broad and sound set of mathematical and computing skills, and are well-served by a flexible curriculum that enables them to pursue topics of interest among the many subdisciplines in computing.
      Visit Home Page

MAIN RESULTS

The following were the outcomes of the project:

1. The user is able to go to the application, enter keywords and search for courses relevant to the keywords from different US universities.
2. If the university also offers related MOOCs, these MOOCs are listed as well.
3. Based on the user's search history, the application recommends certain programs to the user.

Search functionality is implemented using OkapiBM25 Ranker and the results of evaluation on the Cranfield Dataset for the chosen set of parameters is:

NDCG@10: 0.359

Our evaluation of the recommender was an empirical evaluation (since there is no gold standard to test against). We tried out our test cases and the recommender worked as expected.

Test Cases for Search Keywords:

- text analytics
- machine learning
- data mining
- english
- language courses
- literature

SELF EVALUATION

We have completed all the required features of this system and got the expected results. However, as always, there is definitely a scope for improvement. The areas in which we see scope for improvement and future extension are as follows:

1. **Improved Recommender System**: The recommendation algorithm has a significant scope of improvement because it does not always return what we expect. In the future, we can investigate more into having more accurate methods of recommendation.
2. **Dataset Expansion:** Due to the time constraints, we enabled to search and recommendation only for a few selected universities. In the future, we can extend the system to courses and programs from all US universities. However, one problem associated with this is the difficulty in scraping different university websites, which have different HTML class and id layouts.
3. **Optional Feature:** We can implement the feature we had listed as optional in the proposal, namely, to display the department (school within a university) that is offering the course and rank the departments based on the number of courses that the department is offering related to the topic the student is looking for.

INDIVIDUAL CONTRIBUTIONS

Member Name	Hours Spent	Contributions
Harita Reddy (haritar2)	~20-24	1. Scraping university websites for course catalogs and programs, storing them in the required

		format. 2. Creating a dataset of Coursera MOOCs offered by universities manually. 3. Course search and MOOC search functions using the above datasets and the Cranfield datasets 4. Recommender functions
Eric Crawford (ecraw3)	24	1. Setting up the backend server. The work involved experimentation with Django and other frameworks and finally arriving a the Falcon framework which was able to work with Metapy 2. Integrating search and recommendation features with the backend 3. Implementing the frontend interface