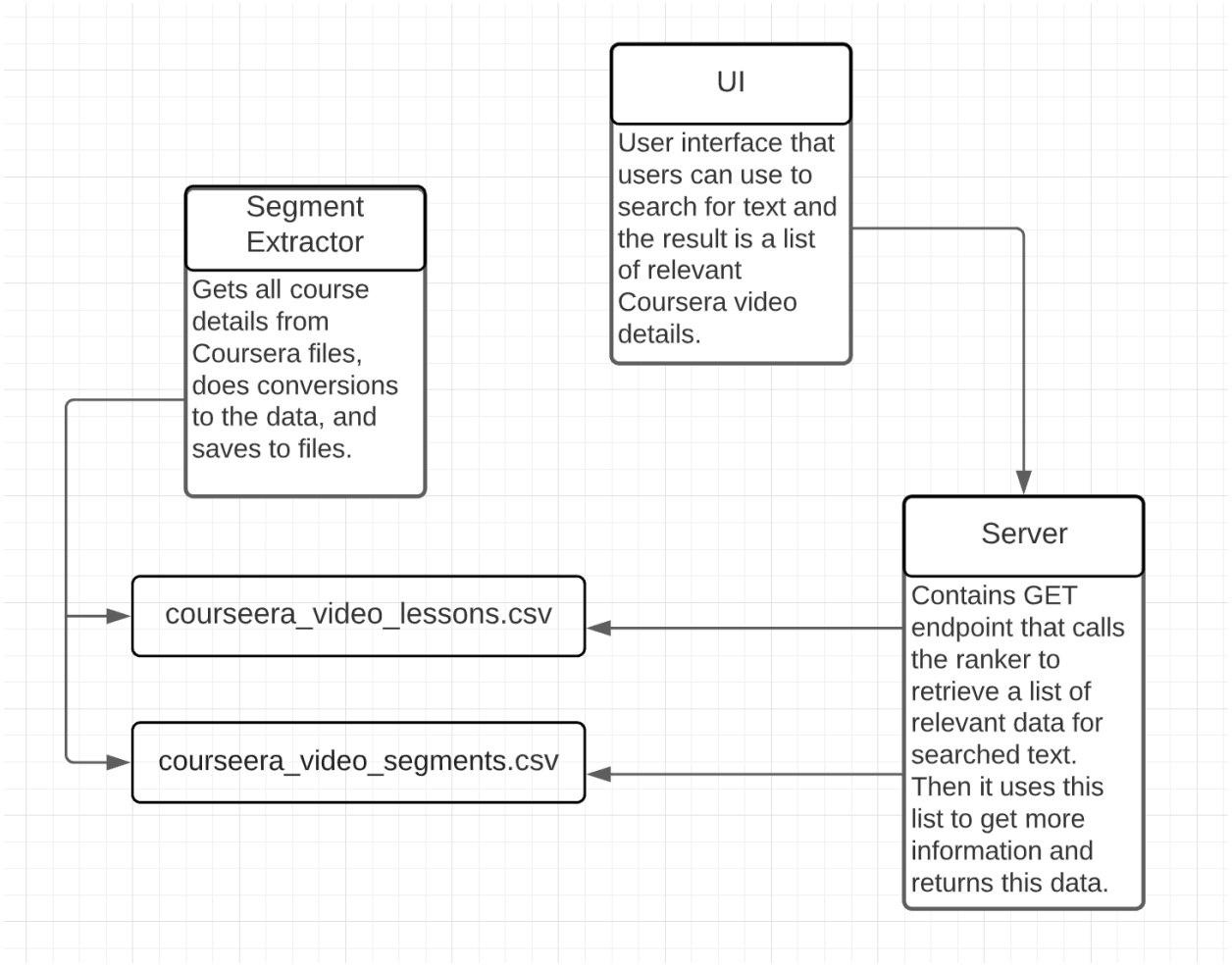
Course Project Documentation

1) An overview of the function of the code (i.e., what it does and what it can be used for).

We developed a “smart video search” application that allows users to search for particular video segments in Coursera across courses. The result of the search is a list containing details of the top x number video segments that contain the searched text, where x is a configured number in the code. This software is useful, because it lets users easily find useful videos to help with studying.

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

Overview:



Detailed Implementation:

- Scrapping of Data
  - We used the coursera-dl (python based utility) to download the videos and its associated text files.

    Example of how to scrap the videos for course cs-410 :

```
Selvaganapathys-MacBook-Pro:courseera-dl selvaganapathythirugnanam$ coursera-dl cs-410 --debug
```

- Segment Extractor
  - “Segment Extractor” is a utility written by our team that works on top of the scrapped videos and associated metadata. The Segment Extractor takes these downloaded videos and associated metadata as input. It then extracts the following attributes from parsing these video files and srt files :

Derived Fields	Description
Course ID	Actual Course (Eg: cs-410)
Week ID	Associated week of that video (Eg: Week 1)
Lesson ID	Lesson Number for a respective week
Video Title	Title of the video/lesson
Video Content	Video content as text (per lesson)
Video Segment Text	Segments within a video. A video segment consists of a specific time frame in a video along with the associated text content for that timeframe
Link To video segment	Link to precise location of the video segment
Video Thumbnail	An image representing the video subsegment.

The Segment Extractor utility is a generic utility which accepts an array of courses that can be parsed. In our project, we parsed 3 courses(cs-410, cs-416-dv, stat-420) to build a strong index. The output of the Extractor are 2 csv files :

- **Coursera_video_lessons.csv** - This csv file is the main output and contains a collection of raw text content of each video. Each video text

content is saved as one line. The idea is to treat each video text content as a document for the Ranker.

- **Coursera_video_segments.csv -** This csv file contains details regarding video segments. The Server code refers to this file based on the recommendations received from the Ranker. In short, this file is used for finding the precise location of the video when applicable.

- Search User Interface
  - The Search UI is a simple interface which accepts a query from the user and displays the results as video links with appropriate video thumbnails.

    The UI is built on top of ReactJS and Bootstrap. The UI itself runs on top of the NodeJS server on port 3000. When running locally, the app can be accessed by hitting the following URL in the browser: http://localhost:3000 Once the page loads, the user will have the option to perform a search , clear the previous results etc. When the search button is clicked, HTTP GET based AJAX calls are made to the server (running in python) . The response payload is in JSON. The UI component then parses the response JSON and renders the results as video links. The user when clicking on the video thumbnail link automatically takes the user to the respective lesson video in the courseera website and also the video will start playing based on the timeframe available in that link.

    For Eg: The query parameter(i.e ?t=41) in the below video link states that the video from the **41st** second.
    https://www.coursera.org/learn/cs-410/lecture/MsDCs?**t=41**

- Server
  - Runs on http://localhost:5000
  - When the server is run it initializes the ranker. This process starts with building a corpus using the text content that is created in the segment extraction process, this is found in the file courseera_video_lessons.csv. This corpus is saved in the file, course.dat. Next, the other components of the ranker are initialized: inverted index, config, etc. OkapiBM25 is used for ranking.
  - The server has a GET endpoint that accepts two parameters. The first is the searched text and the second is an optional integer that will decide how many results to return. The endpoint goes through several processes...

- It first calls the query_result function in the ranker. This function returns the top x number of results from the ranker that are the most relevant to the searched text, where x is either the second endpoint parameter or the default value. Each result contains a course ID, week number, and lesson ID. The ranker is built using five steps. First it uses the coursera_video_lessons.csv to build a corpus. Then for faster access, it builds an inverted index on the corpus. The next step is to build a ranking function. For the project, OkapiBM25 is used with set parameters. The final step is to use this ranking function to score the query. This is done using a metapy method called score. This method returns top 5 documents matching the query. The query_result uses this result to return course ID, week number, and lesson ID for each document.
- After this, the endpoint loops over each result and calls the get_video_details function. This function searches the courseera_video_segments.csv file for all of the lines that contain the result's course ID, week number, and lesson ID. Let's call these lines resultLines. Out of the resultLines, the function looks for the first line that contains the searched text. If not found, then use the first line. Let's call this line correctLine. Next, we need to make sure that the image displayed is not blank. The segment extractor only saves images from the course videos in 60 second intervals, because it would take up too much data to store all of the images. This means that the text in the videos that occur before 60 seconds has a dark image from the 0 second. To fix this, we check to see if correctLine's time_start has a 0 minute. If it does then we look for the first line from resultLines that has a time_start with 1 minute and use that image name for correctLine. Finally, the function puts together the array that contains correctLine's data and returns it. The endpoint returns a JSON array of arrays containing each results' correctLine data.
    - Example of a results correctLine
        - {

"course": "stat-420",
"week": "7",
"video_title": "Introduction",
"text_preview": "jhuioaufaui",
"segment_link":
"https://www.coursera.org/learn/stat-420/lecture/sViGf?t=0",

"img": decoded utf-8 image path
}

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

API cURL examples:

curl --location --request GET 'http://127.0.0.1:5000/search?q=slope'

curl --location --request GET 'http://127.0.0.1:5000/search?q=slope&number_results=6'

How to run UI (Windows):

1. Make sure node and npm are installed
2. Open a new command line
3. Navigate to the smart-video-search folder
    a. Example: cd C:\CourseProject\search-ui\smart-video-search
4. Run npm install
5. Run npm start
6. If the browser doesn't open automatically, go to Chrome and search http://localhost:3000/

How to run Server (Windows):

1. Open a new command line
2. Navigate to the CourseProject folder
    a. Example: cd C:\CourseProject
3. Run server.py using python
    a. Example: py -3.7 server.py

*** You may need to install modules

4) Brief description of contribution of each team member.

Kartik - Worked on Ranker and presentation

Selva - Worked on UI and Segment Extractor and documentation

Diana - Worked on Server and documentation

Matthew - Worked on Segment Extractor and documentation