

DCU School of Computing Assignment Submission

Student Name(s): Hannah O Connor, Catherine Mooney

Student Number(s): 16382283, 16416052

Student Email(s): hannah.oconnor26@mail.dcu.ie,
catherine.mooney33@mail.dcu.ie

Programme: BSc in Computer Applications

Project Title: CA4009 Research and Development Project

Module code: CA4009

Lecturer:

University Search System

Table of Contents

Declaration	2
Abstract	3
Introduction	3
User Analysis	4
Functional Description	4
OCR	4
Crawler	5
Google Cloud Speech-to-Text API	5
Google Translation API	5
Corpus	6
Video & Image	6
Audio	7
Electronic Documents	7
Online Resources	8
Implementation	8
Current System	8
High Level Architecture Diagram	10
Overall Architecture Diagram	11
Tokenization	11
Preprocessing	11
Dynamic Inverted Index	12
Information Retrieval	13
Request Parser	13
Document Retrieval	13
Document Ranking	14
Document Classification	15
Result Filtering	15
Recommender System	16
User Interface	16
Evaluation	17
Conclusion	18
References	19

Declaration

I understand that the University regards breaches of academic integrity and plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach this policy. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the sources cited are identified in the assignment references. I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work. By signing this form or by submitting this material online I confirm that this assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. By signing this form or by submitting material for assessment online I confirm that I have read and understood DCU Academic Integrity and Plagiarism Policy.

Signed: Hannah O'Connor, Catherine Mooney

Date: 13/12/2019

Abstract

The purpose of this report is to show our analysis and developed technical specification with architectural diagrams, implementation and evaluation plans for a proposed new search application system.

We believe that the notes on loops and library material should be located in the same datastore available to be queried at any given time. Traditionally, when a student would work upon a given assignment they would either seek out the course notes on Loop or any related Library books. Yet, these processes can take time and essentially close the student off to the abundance of other online resources. Many lecturers and peers often suggest online material for modules, but it is up to the students to research and retrieve those additional notes

Our proposed project idea is to enhance the standardized and typical University search system to accommodate a wider range of course related content, by transforming the preexisting simple engine to one central tool for students alike. Essentially, it will be built on the integration of Loop and the current online library catalogue. Along with extending that further to include some additional external sources.

In this report we detail how we would establish this central student search engine which could be utilized to search relevant books, lecture's notes, screencasts, udemy courses and Youtube videos, etc.

Introduction

Our project is based upon an online University search system where all electronic archives and module content are stored within a central location. This online University search system will primarily be used to handle search requests that are a lot less specific than a simple document title or published year. On loop, many of the notes are labeled by the week number or 100 pages of notes are categorised under 1 single heading. And this can make it extremely difficult for students to obtain answers to questions, as more often than not, students would sit scrolling through those pages or watching lengthy videos. This is not only time consuming but also causes a drop in focus due to low attention spans. *See Section "Current System" for more detail.*

Our proposed system would require the user to log on using their DCU credentials. From this we could yield their degree's module notes, retrieved from the DCU student database. The system will then be searchable by the user, to query keywords or phrases, which will render relevant documents based on previous searches and the degree itself.

User Analysis

Our target user type will be predominantly DCU students and alumni members, therefore the UI will be designed with simplicity in mind. With many possible means and use cases for this integrated learning search engine, including but not limited to: Researching, Assignments, Studying. For instance, if a student is studying for exams using past papers and comes across a specific question they cannot answer, they could query the key words in the question. The system would then return an index to the specific part of their notes where it was found, along with any other articles, videos, sites from our collection. As students can come from various technology backgrounds & courses, we want to assume that no one would be intimidated by complexity.

The inspiration for this idea came from ourselves, the effectiveness of having one single and accessible platform would aid us greatly in our own course work. Being students we can understand what the requirements for our user type would be, and how to incorporate facilities to support learning.

Functional Description

The following tools allow content in any form in our search system to be made both indexable and searchable.

Optical Character Recognition (OCR)

OCR will convert the system's graphical text into digital form. It can identify words from an image whether it's typed, printed or handwritten text. A good usage would be for a lecturer to scan their written notes. Here, OCR would then translate it to text, making it searchable. It does this by recognising characters on an image, such as letters, numbers and symbols, by using feature extraction *Zoning* on the segmented characters. Initially, we planned on implementing matrix matching, however, in cases where handwritten notes are used for screencasts, feature extraction prevails. This too accommodates different fonts, with the only precondition being that image must be high resolution. [1]

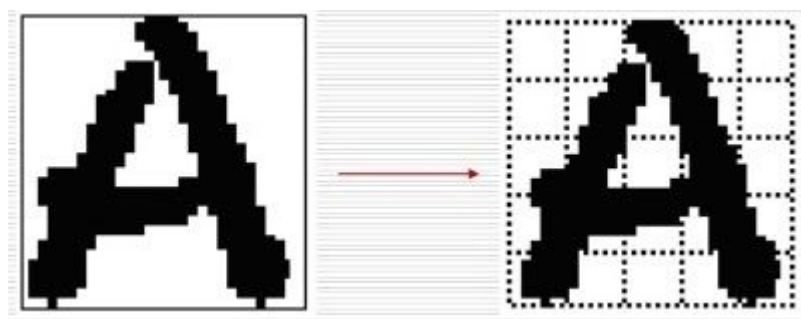


Image: Feature Extraction Zoning [2]

Crawler

Here, the system will create metadata of our online services, to be indexed by a “spider crawler”. That spider can then visit the coded list of URLs, retrieving their content and processing, indexing and storing that data within our central database. Our system will integrate Scrapy, an open source, highly scalable and dynamic web crawler written in Python. [3]

Each spider will have 2 main methods: start and parse. *Start* will work its way through our queue of URLs and parsing the HTTP response. But it not just as straightforward as that, we need to also developed the spider to look for pattern within the HTML source code of sites, such as Videos, Images and Hyperlinks. We do this through using XPATH. Multimedia content can be downloaded using the imported Python Requests library. The data is first selected from html pages using XPATH and then retrieved by the request function: `request.get(multimedia_url).content` [4]

The crawler could pattern match hyperlinks using: `response.xpath("//*[@href or @src]")` and then recursively call on those embedded links. [5]

Google Cloud Speech-to-Text API

By using neural network models, the Google Cloud Speech-to-Text API would allow us to translate any audio to digitalised text. This would allow for screen casts, webinars, videos, podcasts, etc. to be made indexable and searchable as the voice over often contains a lot of valuable information that previously could have been missed, had it just been searched by image or text. [6]

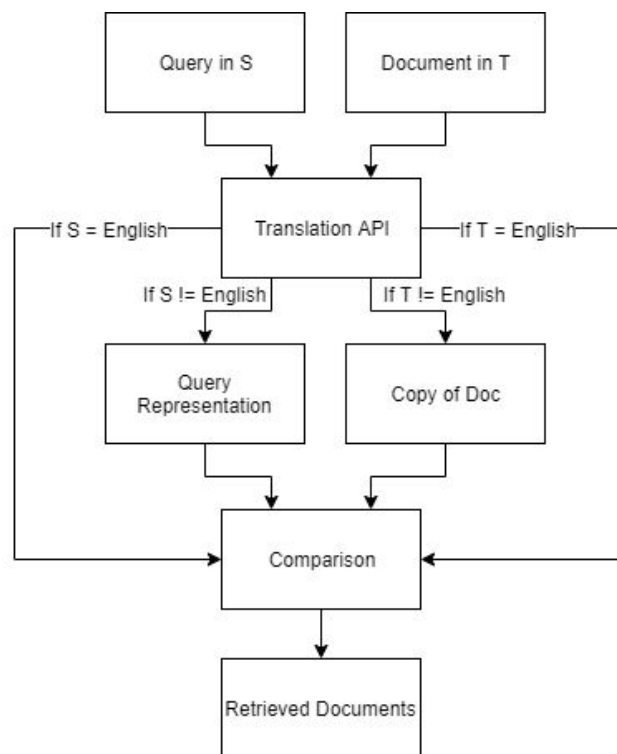
There are some disadvantages to this model, given the difficulty of obtaining a perfect text translation. We need to consider the case where it may derive incorrect matchings or miss important terms, as this aspect could be catastrophic for our system’s overall precision.

Google Translation API

By adopting the Google Translation API into our architecture, it transforms our system into a “*Cross-language information retrieval system*”. Here, the system will return a list of documents based on the query, regardless of the language, ultimately creating a multilingual environment.

By creating English copies of each foreign document, it removes the complexity of having to integrate multiple tokenizers and inverted indexes stores for each accommodate language. For this aspect of our stack, we had investigated various research findings for the best approaches, and simple query translation was the most flexible and accessible route to take. [7]

For more details, see Section “Preprocessing”.



Corpus

Our corpus consists of various multimedia content which we will discuss below. Given that our collection comprises of content sources from various different locations, it introduces a much higher complexity than its original separate counterparts. Hence, why we wish to perform the preprocessing and indexing steps offline.

Video & Image

Our system will accommodate various multimedia types that match the user query, including videos and images. This relies on crowdsource-based labelling for screencasts and online multimedia content. The system will also request lecturers to give a brief description of their documents.

Our searchable video types would be those which contain text and audio like screencasts. Videos can be searched in multiple forms. These include: metadata, audio or broken down sequential frames and images. This means that OCR will search a video's frames and images to translate both into digital text. [8]

Audio

The audio of video and audio documents, we will be treated by Google's Audio-to-Text API, to retrieve their digital text format. Often podcasts,

screencasts, webinars, etc. contain long spoken content which are multi-topical. Consequently, this can make it difficult for users to concentrate on finding exact information due to a low attention span.

We are aware that Audio-to-Text translation can introduce complications. Words can be merged together and others are extremely ambiguous to categorised. After various research and discussion, we concluded that the Google Audio-to-Text API, based on its extensive and continuous testing by Google Developers, is the best option for our system. [4]

To use this API on an audio document, it will need to be uploaded to Google's Storage bucket. This API call and uploaded to the bucket will be completed within our preprocessing steps. The following is a Python example to make an API call to Cloud Speech-to-Text.

```
def transcribe_model_selection(speech_file, model):
    """Transcribe the given audio file synchronously with
    the selected model."""
    from google.cloud import speech
    client = speech.SpeechClient()

    with open(speech_file, 'rb') as audio_file:
        content = audio_file.read()

    audio = speech.types.RecognitionAudio(content=content)

    config = speech.types.RecognitionConfig(
        encoding=speech.enums.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=16000,
        language_code='en-US',
        model=model)

    response = client.recognize(config, audio)

    for i, result in enumerate(response.results):
        alternative = result.alternatives[0]
        print('-' * 20)
        print('First alternative of result {}'.format(i))
        print(u'Transcript: {}'.format(alternative.transcript))
```

Electronic Documents

We assume that the engine will be in contact with various electronic documents formats, so we need to ensure that each can be supported and made searchable. This includes PDFs, Word Documents, PPT files, etc. Yet, this can increase complexity and so, to resolve any potential issues, each will be converted into an indexable format, such as HTML.

Online Resources

Below shows how our scrapy crawler would be created, and transverse our predefined URLs for the Massive open online learning courses (MOOCs):

```
import scrapy
import requests

class moocs_spider(scrapy.Spider):
    # name of job
    name = 'MOOCs'

    allowed_domains = ['www.udemy.com', 'www.coursera.org']

    start_urls = ['https://www.udemy.com/courses/search/',
                  'https://www.coursera.org/courses/']

    # Parse the JSON response below
    def parse(self, response):
        pass
```

Code Snippet: *How to* [9]

In addition, we will have another spider to crawl the URLs that were scraped from our document collection, those being lecturer's recommendations or reading lists, extracted through the use of regular expressions.

Our system will retrieve more candidates for crawling through gathering the URLs from loop itself in the case of a lecturer using their own external websites for note uploads.

Implementation

Current System

As previously discussed, the pre-existing systems are separated into their own components rather than one combined engine. As seen in the image below, the current online DCU library catalogue is a Boolean Retrieval System. However, this isn't the best option for novice users and requires some IR knowledge in order for it to reach its potential. For instance, grammatical errors aren't always considered.

Search for:

Search

✓ AND
OR
NOT

with

Author

Try surname only as we may only store initials

Image: *Online DCU Library Catalogue Search* [10]

For loop, you can see the below snippet of the tree folder structure of module content. There is no easy way to quickly navigate and search the resources. In contrast, we believe that our system will allow for at least half of the user's time to be saved when seeking out requested information.

Course Overview	URLs: 4
25 September - 1 October	Files: 4 URL: 1
2 October - 8 October	File: 1

◀ Course Overview

25 September - 1 October

2 October - 8 October ▶






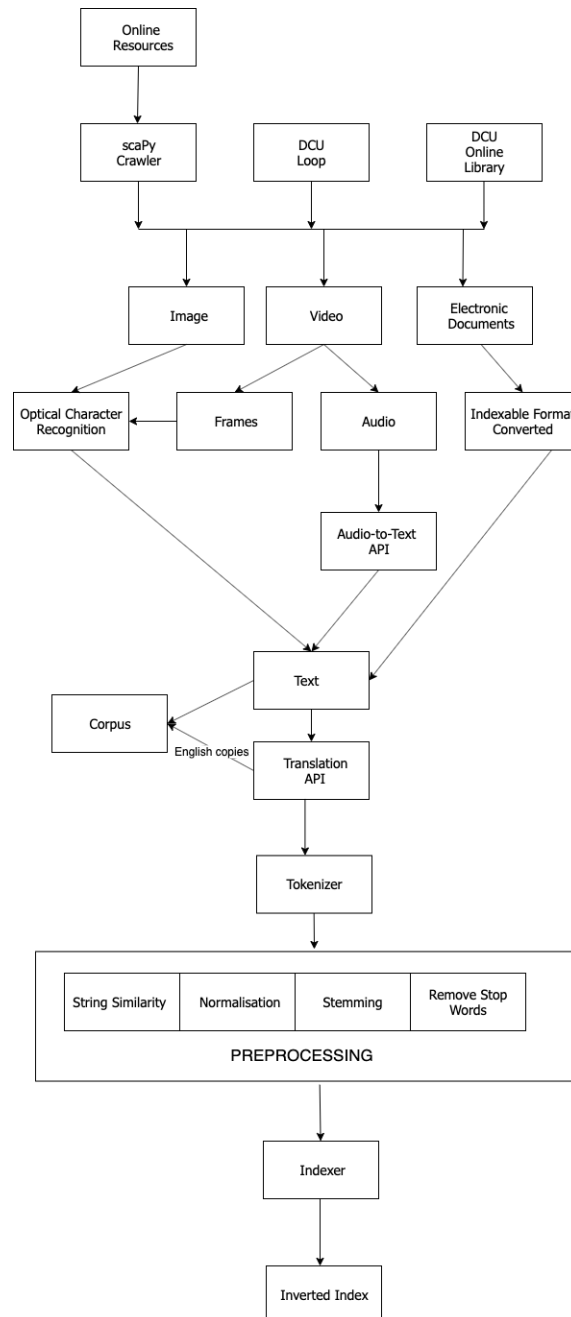
-  Sub-SuperNetting 2016
-  Subnetting questions with answers
-  Subnetting a class C IPV4 address ▼
-  IP Forwarding, subnetting, VLSM and CIDR exercises 70KB Powerpoint presentation
-  Questions with some answers

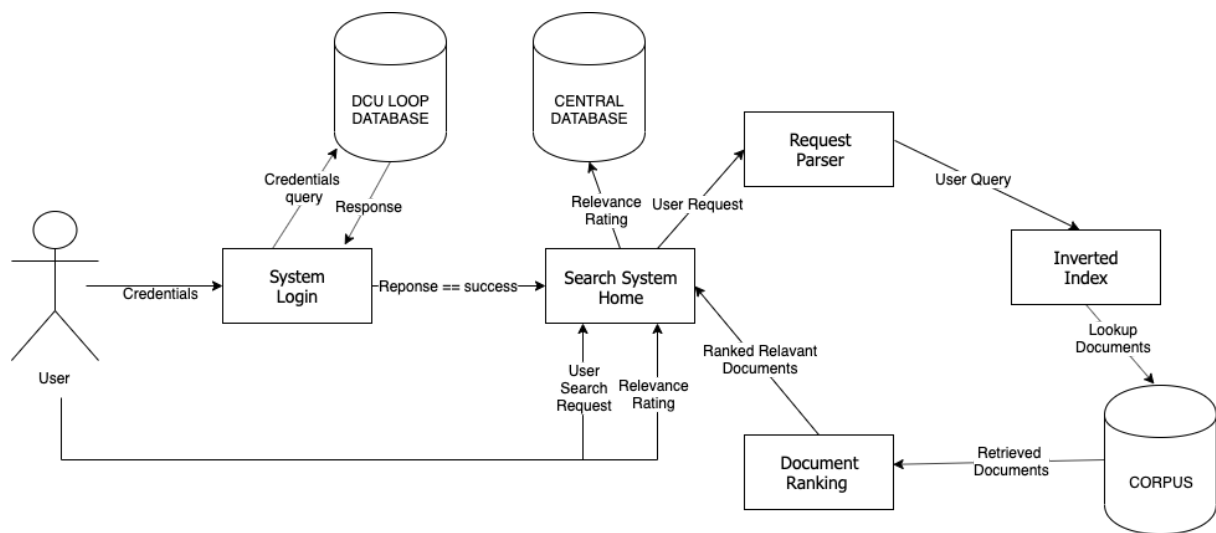
Image: *DCU Loop* [11]

High Level Architecture Diagram



Above shows the process in creating our Inverted Index. Given that this system has mixed media types, each will be treated as its own component irrespectively of the others, regardless of its source.

Overall Architecture Diagram



The above highlights the central flow of our system. The returned ranked documents will be from highest to lowest while still retaining a high precision. That way, we can avoid the user opting to click “More Results”.

Tokenization

Once all forms of media are converted to digital text, we perform tokenization on the documents. The same tokenizer is used for documents as for queries. It ensures that no matter what sequence of characters are in the query, it'll always match the text in the document. This helps in the case of a student searching for a specific IP address or the name of a programming language like C++, etc. [12]

Gather the tokens from each text document in one simple line of code:

```
tokens = [document.split(' ') for document in documents]
```

Preprocessing

Given that English will be the system's primary language, the translation API will be applied to all documents if another language is detected, creating English copies which can then be processed. Next, It looks to remove stop words like 'or' 'and' 'but', as they don't contain any value and prevent us seeking a sentence's semantic meaning. Also, processing won't be case sensitive. Found research, we gathered that porter-stemming would be the best stemming method and that Levenshtein's algorithm for string similarity would be best practice. Next, is the normalising of the document length and of the tokens, using equivalence classes. [14]

Dynamic Inverted Index

Our system opts for a dynamic Inverted Index, a mapping of each unique term to the list of documents it's contained within.

We had originally assessed whether Forward or Inverted indexing would be the most appropriate for our system type. Ultimately we decided on the latter due to forward's high search cost. Although Inverted indexing can increase latency, there is a simple work around. The indexing process will be ran at a pre-scheduled time while the system is not actively in use.

Given a user query, we can use this data structure to return the list of relevant documents based on it. As we don't wish to include every found word from our collection in the index, we will perform the preprocessing of the text beforehand. This way we reduce the indexing computational time drastically.

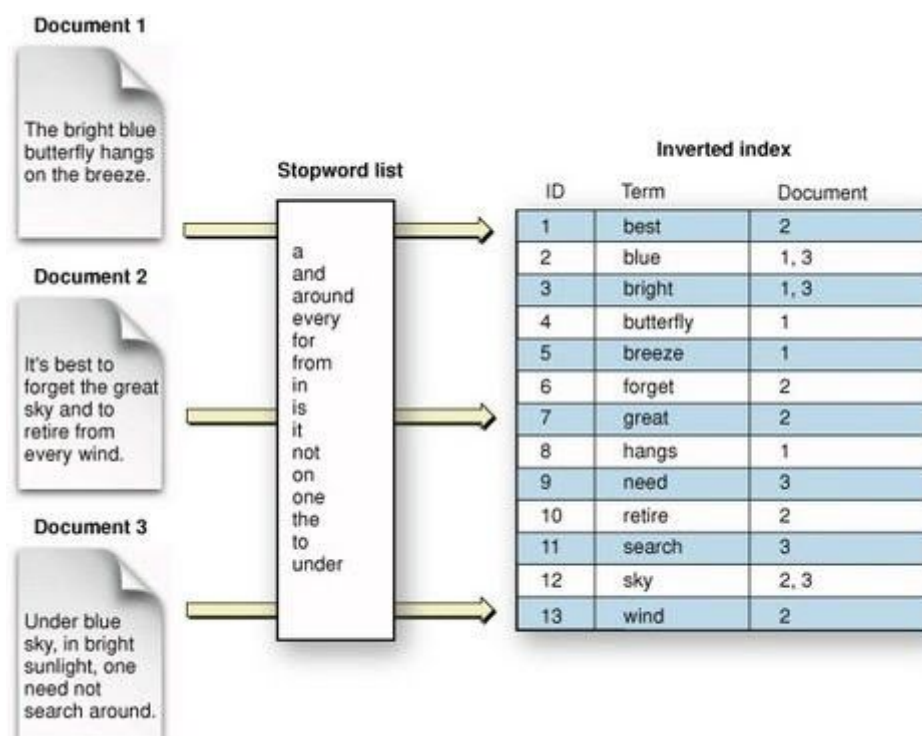


Image: *Inverted index* [13]

This will be implemented using Python's dictionary data structure, where every token would act as a unique key. Rather than the dictionary being a simple term to list of document mapping, we will map each token to a list of lists. Within each dictionary index, contains a document and the token's list of positions within that single document as one single entry. By doing this, it allow for the feature of returning the term's position. [14]

Example:

```
'''
Key represents the term,
Value represents a list of lists where within each list, the 0 index
contains the document_id & the 1 index represents the position of that
term from the given document.
'''
{ 'word' : [1,[4]], [2,[6,19]] }
```

Through storing our inverted index locally to disk, it'll ultimately optimise the system's speed when responding to a user request.

The system will make use of it's timer, both to rerun the spiders and seek out any new documents. This means reindexing at regular intervals to accommodate for new additions or changes.

Information Retrieval

Request Parser

To begin Information retrieval, we needed for the users request to be translated into an effective query. First, we parse the natural language from the documents and the user request itself, to identify potential indexing terms. The parser drops ambiguous terms which don't aid the system's attempt in retrieving relevant documents and then computes the finalised search query used to retrieve documents.

Document Retrieval

Prerequisite: multimedia type files have been processed to digital text by either OCR or Audio-to-text API.

Our search system has two user query types, **standard** and **phrase**. Standard queries consists of a sentence of terms, each separated by white space. This is the best case for the system for lookup and retrieval time. The resulting list would be the concatenation of each of the terms in the query, each treated independently. Therefore, the retrieved documents contains at least one of those words found in the original request. Essentially, it doesn't rely on a direct match. For phrase queries, the retrieved documents would contain all of the individual tokens from the user query in the correct original position. This utilises each term's positioning values in our index to determine this and by doing so, can in turn inform the user of the exact positioning of their request in a document. [15]

Before any query is acted on, it will be run through the *request parser* before looked up by the index.

Document Ranking

In continuation of the previous section, we'll investigate document ranking based on a given query's terms. And how to return an ordered list of documents back to user.

Thoroughly investigating led us to believe that **tf-idf** and **Vector Space Modeling** would be the best approach for our system.

First step in this ranking process is to calculate term frequency for each individual term in each individual document. This metric informs us of how common it is, however, it doesn't always highlight how important. Hence why we introduce our second measure, Index document frequency (idf).

Term frequency:

```
tf(i,j) # the number of occurrences of term t(i) in the document d(j)
```

The purpose of this idf measurement is to evaluate how relevant a term is by accessing how much it appears in an entire corpus, rather than just one document.

Index document frequency:

```
idf(t) = log N/n(i)
...
```

```
Where N is number of documents in collection & n(i) is number of
documents t appears in
...
```

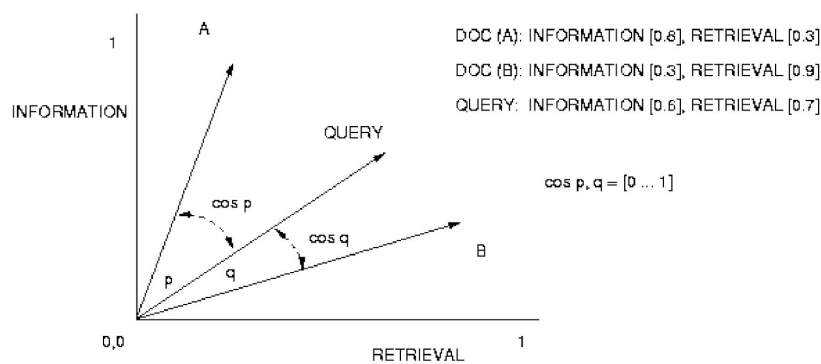
By gathering both, we can access the overall tf-idf scoring of a single term in a document. This calculation is quite simplistic, as all that needs doing to retrieve the term's weight is multiplying it's tf by its corresponding idf.

Next, is the evaluation of the **cosine similarity** using Vector Space Modeling. With this approach, each document is a vector representation of its tf-idf score and the algorithm will examine which documents' vectors are closest to the original search query vector. That being, the closer the distance, the higher the chance of a document being relevant to the query. It returns a indices list of those documents ids where we continue to retrieve the positional indexes. As a result, the user not only retrieves the relevant documents but also which point in those documents that their queried information was sourced. [12]

Cosine Similarity:

$$Sim(q, d(j)) = \frac{\sum_{i=0}^{I-1} w_q(i) \cdot w_d(i, j)}{\sqrt{\sum_{i=0}^{I-1} w_q(i)^2} \sqrt{\sum_{i=0}^{I-1} w_d(i, j)^2}}$$

Vector Space Modelling:



$\cos(q) > \cos(p)$ so B is more similar to the QUERY than A, hence should be ranked higher in a ranked retrieval list.

Document Classification

To make our documents more manageable, we identified the best way to classify them. Uploaded loop documents will automatically be classified by its course and module. Additional online resources can be made available to all students, hence require classification. First, we examine the document for words related to certain topics to predict the classification of the document in question. For longer documents, certain sections are often ambiguous and hard to classify. We therefore allow for each documents to have multiple category types and multiple labels. Documents retrieved can be returned to students in different courses, as one course's document may be helpful to a student in another.

Result Filtering

Our system includes the filtering of the returned resources based on their metadata such as module codes or original languages, allowing the user to

re-categorise the results. This increases the probability of highly relevant returned documents for the user.

Recommender System

We'll introduce Machine Learning for our system to allow it to keep track of the user's past searches and relevance ratings to use for future recommendations, this could be implemented using *Content-Based Filtering*. As it optimises user's time even further. [16]

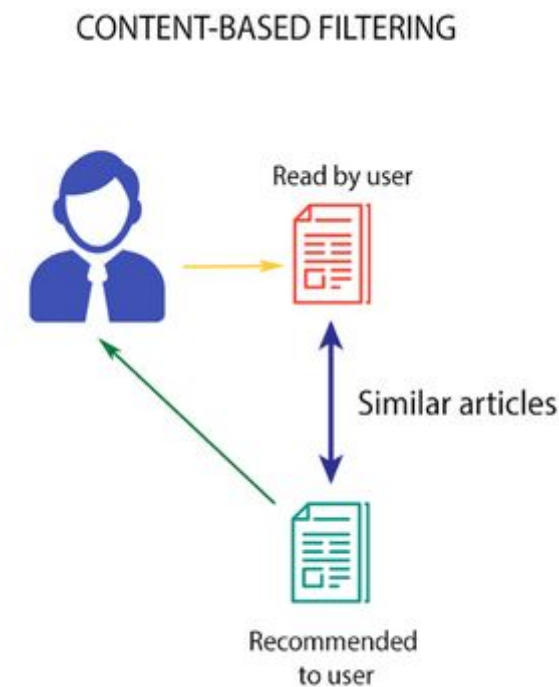
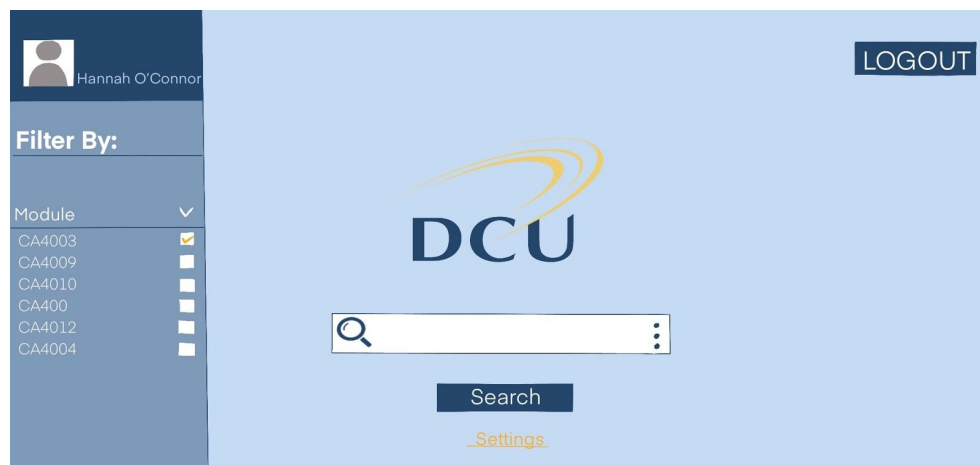


Image: Content-Based Filtering [17]

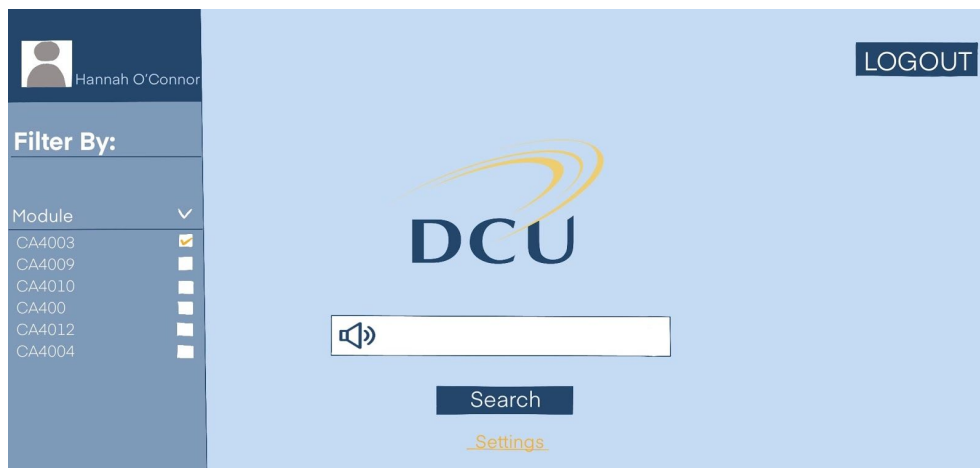
User Interface

Below shows the UI plans for this system. As you can see we had no bias when it came to the design and have adopted techniques from our previous module '*Human Computer Interaction*'.

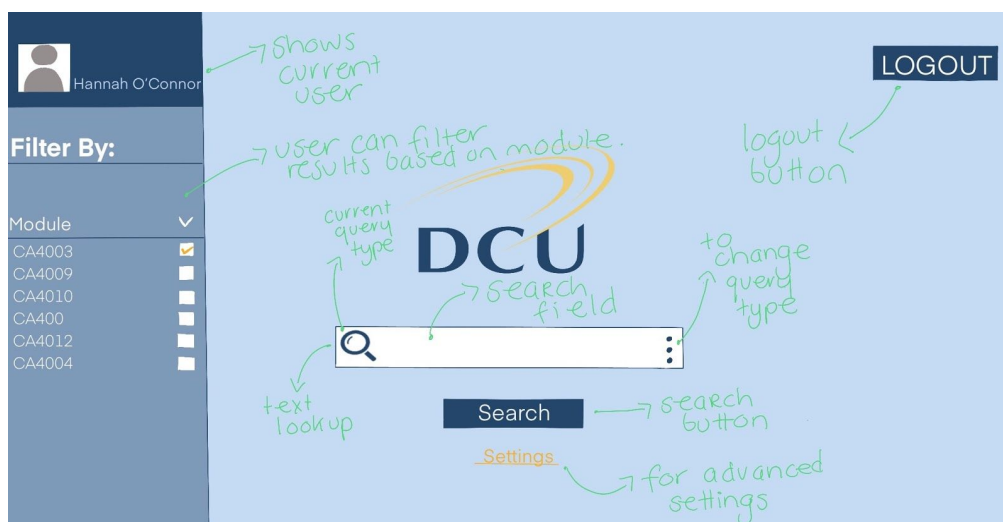
Home Screen:



Changeable search option:



Explaining features:



For settings, the user can modify how many documents are returned by the system.

Evaluation

Our biggest priority is meeting the needs of our users. As are aware of the various methods for evaluation, we wanted to incorporate a way to receive feedback primarily from the user, where they state whether returned documents were relevant or not. This makes the system learn and become more efficient over time.

Additionally, we have multiple ways to evaluate the relevance of returned documents, which the user does not see.

Including:

- User activity
- The number of clicks
- What is searched

We perform an analysis based on these results, calculating the weighted average of similarity by those in the ranked list and those chosen manually.

The above methods encourages us to consistently test and improve our algorithms efficiently to meet user needs. As user testing/evaluation is not possible in this given time frame, we can assess the time consuming activities when manually searching online or on Loop. We strongly believe that we've covered as many possible aspects to ensure high system performance and compatibility.

Conclusion

Our report has detailed the methods which our system would use to sufficiently search and retrieve documents from the mentioned sources, and explaining how we'd implement them and too, evaluate the system. For future reference, we'd introduce a higher involvement of machine learning algorithms to maximise efficiency. The system could be transformed to open source, making it adaptable and modifiable for other universities and learning institutions, not solely DCU.

References

- [1] "How does OCR Work? A Short Explanation" Victoria May 18, 2016
<https://www.scan2cad.com/tips/how-does-ocr-work/>
- [2] Unknown [slide]
<https://image.slidesharecdn.com/off-linehandwrittenocr-140612010204-phpapp02/95/off-line-handwritten-optical-character-regonisation-ocr-14-638.jpg?cb=1418822033>
- [3] ScaPY Documentation: <https://scrapy.org/>
- [4] "How to Scrape the Web using Python with ScraPy Spiders" Luciano Strika, 28/08/2019
<https://towardsdatascience.com/how-to-scrape-the-web-using-python-with-scrapy-spiders-e2328ac4526>
- [5] "Fetching all href links from the page source using Selenium WebDriver with Java", Stackoverflow,
<https://stackoverflow.com/questions/28163618/fetching-all-href-links-from-the-page-source-using-selenium-webdriver-with-java>
- [6] "Transcribing videos" 2019 Google Cloud
<https://cloud.google.com/speech-to-text/docs/video-model#speech-select-model-python>
- [7] "Cross-Language Information Retrieval" Graeme Hirst, University of Toronto [online] <http://disi.unitn.it/~bernardi/Courses/DL/clir.pdf>
- [8] "Image and Video Searching on the World Wide Web" Michael J. Swain, Michael J. Swain, Unknown
https://www.bcs.org/upload/pdf/ewic_im99_paper11.pdf
- [9] "How To Develop Your First Web Crawler Using Python Scrapy" Adnan Siddiqi, Feb 8, 2017
<https://medium.com/better-programming/develop-your-first-web-crawler-in-python-scrapy-6b2ee4baf954>
- [10] DCU library Catalogue: <https://capitadiscovery.co.uk/dcu/home>
- [11] DCU Loop: <https://loop.dcu.ie/>
- [12] "CA4009 Search Technologies - Text Retrieval", Gareth Jones, Oct 2019

https://loop.dcu.ie/pluginfile.php/2865816/mod_resource/content/15/ir.pdf

[13] Inverted Indexing Diagram:

https://mentormate.com/wp-content/uploads/2017/01/inverted_index.jpg

[14] "How to Implement a Search Engine Part 1: Create Index", Arden Dertat, May 20 2011

<http://www.ardendertat.com/2011/05/30/how-to-implement-a-search-engine-part-1-create-index/>

[15] "How to Implement a Search Engine Part 2: Query Index", Arden Dertat, May 20 2011

<http://www.ardendertat.com/2011/05/31/how-to-implement-a-search-engine-part-2-query-index/>

[16] "Comprehensive Guide to build a Recommendation Engine from scratch (in Python)" Pulkit Sharma, June 21, 2018

<https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/>

[17] "Content Based Filtering" - Image Source:

<https://www.yash.com/blog/recommendation-system/>

[18] "Introduction to Information Retrieval" Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Cambridge University Press. 2008.

<https://nlp.stanford.edu/IR-book/information-retrieval-book.html>