Finding The Right Education Using A Recommender System

Second Semester Project Report

Group SER2.1
Hans Hansen
Mikkel Skovsmose Tewes
Nicolai Foldager
Roman Burenko

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg

Department of Electronic Systems
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø
http://es.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Finding The Right Education Using A Recommender System

Theme:
Application Development

Project Period:
Spring Semester 2017

Project Group:
ICTE SER2.1

Participants:
Hans Hansen
Mikkel Skovsmose Tewes
Nicolai Foldager
Roman Burenko

Supervisor:
Per Lynggaard

Copies: 1

Page Numbers: 107

Date of Completion:
January 28, 2018

Abstract:

The aim of this project was to build an application for potential students to find educations, where they can could get recommendations based on their preferences. Therefore, the following problem statement was formed:

How can a recommender system based application support people to find an education that aligns with their preferences?

In order to answer the problem statement a few sub-problems were formed, which lead to the investigation of the following subjects: recommender system techniques, available education information, extracting user preferences, handling user preferences privately and securely, and how such an application could be designed and implemented.

The outcome was that a hybrid of content- and constraint-based recommender systems can be used. Some of the content-based data used was not perfect, and the user interface should be optimized, but the prototype did prove it to be possible to build such a solution.

Contents

Abbreviations

AAU	Aalborg University
API	Application Programming Interfaces
FR	Functional Requirement
GDPR	General Data Protection Regulation
HTTPS	Hypertext Transfer Protocol Secure
IDF	Inverse Document Frequency
KOT	Den Koordinerede Tilmelding
NFR	Non-functional Requirement
PII	Personally Identifiable Information
RS	Recommender System
SSL	Secure Socket Layer
TF	Term Frequency
TLS	Transport Layer Security
UG	UddannelsesGuiden
UML	Unified Modelling Language
VCS	Version Control System
VSM	Vector Space Model
uZOOM	UddannelsesZoom

1 Introduction

The advancements in ICT the recent time have resulted in an information explosion. Today you have access to data and information about anything imaginable. With a few clicks, one can get to know the weather situation in Nicaragua while being located in Denmark, get a personalized book recommendation and access pictures of planets millions of kilometers away. The ease of access within the information society allows us to always make the most educated decision, whether it being what to eat for dinner or how to shape our future. Combine this with the rise of Web 2.0, where the Internet move from being a static entity to a dynamic entity, largely shaped by the users. Not only do you have access to reading about other's experiences and opinions, you also have the opportunity to ask both regular users and domain experts in-depth questions. Further, you can engage in dialog with multiple people through forums who can comment on posts written by others. This enables us to peer-review texts on the Internet by harnessing the intelligence of the masses [1].

While the amount of information available can be huge, it is not worth much if you do not know what you are looking for. In addition, sometimes the amount of information can be intimidating and possibly cause an overload of information to the point where the users will stop researching. In these cases, individualized recommender systems can filter out irrelevant information for each user based on their preferences. Similarly, recommender systems can be used to give the users recommendations based on their prior behaviour or by comparing them to users with similar preferences or behaviour. In short, recommender systems support users in decision-making processes and has been widely adopted; Netflix uses them to recommend movies and TV shows that fit your taste based on what you have watched in the past. Amazon uses recommender systems to show products you might be interested in and to show similar products to the one you are currently looking at. The field of recommender systems is a multi-disciplinary subject. It spans over artificial intelligence, human-computer interaction, data mining, statistics and many more [2, p. 21].

1.1 Motivation

Every year, at the start of July, up to 100.000 Danes each apply to up to eight different post-secondary educations [3]. In the time up to the deadline on July 5th, most spend days if not weeks, on researching countless of possible educations to ensure they apply and prioritize

the correct ones. The time commitment to find the correct education is understandable as in 2015 there were over 840 post-secondary education options. A number that grew by 57 over only four years [4]. Not only does this make it difficult to grasp the possibilities, but it also makes it difficult to distinguish between two similar educations.

There are a vast amount of sources that can help people make the decision about which educations to apply to, many of which are only possible due to the advancements in ICT. In Denmark, most services are websites that should be treated as a reference for finding information about educations that you already know about and not as a way to discover new educations. Today, if you want to discover new educations you have to know the industry that you want to work in. This approach may be useful for people who does know exactly what industry they would like to work in. However, for the people who does not know, they need to sort through various education names, descriptions, admission requirements, etc. to find a candidate education.

Several scholars have already discussed how recommender systems can be used within the classroom to improve learning, for example [5, 6, 7]. However, maybe recommender systems can be used before the students enter the classroom and help them decide which class room to enter. The assumption is that a recommender system can help the search for educations by filtering out irrelevant results for the user and only show educations they only have an interests in.

1.2 Problem Statement

The following problem statement was formed based on the issue that searching for educations to apply for can be overwhelming, and on the fact that recommender systems have proved to be useful in decision-making processes in the past.

How can a recommender system based application support people to find an education that aligns with their preferences?

Additionally, the following sub-questions were formed to support the problem statement.

- What recommender system techniques can be utilized?
- What education information is needed to support the RS and how can it be obtained?
- How can user preferences be extracted and used?
- How can the user preferences be handled privately and securely?
- How can the architecture be designed and implemented?

1.3 Delimitation

Delimitations have been set in order to narrow down the scope of the project. First, only educations at Aalborg University will be used for the prototype development. This narrows the amount of possible suggestions down to 184 from over 840 educations. Further, this narrows down the possible data collection to a single source rather than one for each educational institution. In addition, the implementation was limited to ten select educations. The educations selected and the reasoning behind will be elaborated in chapter 6 Implementation.

Second, domain experts are used in many recommender systems in order to get a perfect filtering and classification of data used by the system. As we do not have access to a domain expert for every single education we will do the filtering and classification ourselves based on our knowledge of domain.

Third, the prototype will be developed using the MEAN stack as it is a major part of the 2^{nd} semester course Development of ICT and Media Services. Furthermore, this means that the prototype will be a web application and not a native mobile application.

1.4 Report Structure

This report follows a conventional structure. First, the methodology will be described which includes development methodology. Next, the recommender systems will be presented and discussed in chapter 3 State Of The Art and 4 Analysis. These will end out in requirement specifications for the prototype. Afterwards, the architecture and development of the prototype will be presented in chapter 5 Design and 6 Implementation. Finally, the prototype will be evaluated in chapter 8 Discussion. The report and development of the prototype will be concluded in chapter 9 and any continued work in chapter 10 Future Works.

2 Methodology

This project consists of two major parts; a report and a prototype. Due to this, time management was emphasized in order to ensure that every task had ample time to be completed and was completed in the end. The primary tool used to gain an overview of the time available and follow up on the progress was Gantt Charts. One for the report (see B.1 in Appendix B) and one for the prototype development (Appendix B.2 in Appendix B).

Every meeting started out with creating an agenda where tasks were discussed and defined. This agenda was used until the next meeting. A picture of what an agenda looked like can be seen in Appendix C. Furthermore, we had smaller meetings throughout the days, where each of us gave a status update on how we were doing with the different tasks. As the the project's deadline approached this was emphasized more, where we specifically added deadlines to the whiteboard in the top right corner. The aforementioned can seen in Appendix C

This project was heavily influenced by desktop research as "recommender systems" is a multi-disciplinary subject. It consisted primarily of research papers, books, and technical documentation.

2.1 Development Method

A prototype was developed during this project and a version control system (VCS) was used. This means that all the changes made to files were saved in the VCS and this allowed us to go back to the previous state of files [8]. A distributed VCS Git was chosen for the project. This has the benefit that all developers have a local copy of the code residing in the VCS, and it is therefore also a backup in itself. Git also provides feature branching, which allowed each developer to work on different features for the prototype while having a working version running on what is referred to as the *master* branch.

The three different methods presented in table 2.1 were considered when choosing an appropriate development methodology. The different methods with their advantages and disadvantages are also presented in the table.

Table 2.1: Advantages and disadvantages of three different software development methods

Method	**Advantage**	**Disadvantage**
Joint Application Development [9]	Involves the customer in the design and implementation phase. This is through workshops called JAD sessions	The customer might end up defining unrealistic requirements which could lead to over or under-developed functionality
Rapid Application Development [9]	The business owner participates in the prototyping, provides use case scenarios and participates in unit testing	This method depends on strong teams and individual commitment. Decision making is not done by the development team
Lean development method [10]	Delivers a minimalistic solution fast in with less functionality, e.g. to test a new technology	The developed product might loose its competitive edge due to the lesser functionality

The Lean development method was chosen, due to the need of delivering a minimalistic solution with limited functionality, in order to answer our problem statement. As this project's aim was to uncover a new way of helping people chose an education, the disadvantage of the Lean method seemed feasible and affordable. A more in-depth description of what the Lean method's principles are, can be seen in Appendix A.

2.1.1 UML

Figures and diagrams were used throughout the project in order to make it easier to explain ideas and designs, and the Unified Modelling Language (UML) was therefore used. Using UML made sure that a standardized way was followed and it ensured that structure, behaviour and interaction diagrams would be understandable for people other than ourselves. But it mostly helped communicate the ideas within our project group.

3 State of the Art

3.1 Recommender System Techniques

Many available services today are using recommender systems (RS) to provide recommendations for each individual user who uses the service. [11] mentions that many years ago researchers would define recommender systems as systems where users provide recommendation inputs, and then the system uses these inputs for aiding other users. However, [11] also states that recommender systems have a much broader meaning today, and he describes recommender systems as any system that provides a user with an individualized recommendation or if the system has any effect on guiding the users to an item, which the user might be interested in. Examples of such services are Amazon and Netflix, where Amazon for example suggests other products related to the one you are currently looking at, and Netflix recommends users with movies they might like based on watch history and ratings. There are many different techniques that can be used for recommendations that can be either personalized or non-personalized [12]. Non-personalized recommendation is for example when a system can recommend the most popular song among a collection of songs, which means that the same recommendation is given for all users. It is a generic recommendation on what users might like compared to a song being picked randomly. Personalized recommendations is when recommendations are tailored for each individual user.

There are many types of RSs, some generate a profile of a user, which is needed in order to be able to produce a good recommendation. The more information available on the user the more accurate the recommendation can be, because the system needs enough information about the past to be able to predict what the user will like in the future. Items, which e.g. can be a book or a car, can also require a type of profile before they can be recommended. For example, a system that recommends books might require users to rate other books before they can be used for recommendation. This means that for some RSs new users can be problematic since there is not enough information about the user, and the same can be said about new items that are added to a system. This is referred to as the 'cold start' problem [13].

3.1.1 Collaborative Filtering

In *collaborative filtering* the RS uses the ratings of other users in order to provide a recommendation [14, 11, 15, 12]. An example in a music application is if Jack gives the same rating of a song as Anna, then it might be likely that Jack would like a song that Anna has rated highly. This means that users who rate items in a similar way are somewhat interested in the same items. This is based on the idea that people who have agreed in the past tend to agree in the future. Collaborative filtering is one of the most used techniques, and there are several different categories of them, which each uses different inputs [11, 12].

Memory-based algorithms use an entire user-item database in order to generate a prediction. Furthermore, statistical techniques are used to generate a set of similar users, which are known as neighbors. These neighbors have a history in agreeing with the target user [16]. Once a neighborhood (database of users) is generated, the memory-based systems can use different algorithms to define a prediction or give a recommendation, based on the neighbors combined preferences [16]. This technique is also known as user-based collaborative filtering or nearest-neighbor.

Model based algorithms on the other hand provide recommendations based on a model built around the users rating. The model is built by performing different machine learning algorithms such as clustering and rule-based [16]. For instance, the clustering model works by treating it as a classification problem and clusters similar users together, and then estimates the probability of which a user belongs to a specific group [16].

3.1.1.1 User-based Collaborative Filtering

User-based collaborative filtering collects user profiles which should represent the preferences of many different users,and this is used to locate users with similar profiles [17]. Once the similar users have been computed, the most similar ones are selected and used as a recommendation source. The concept is that the user likes the same as what the similar users have liked in the past.

3.1.1.2 Item-based Collaborative Filtering

Item-based collaborative filtering works by exploring the relationship between the items rather than exploring the relationship between the users [16]. Since the relationship between items are relatively static, item-based algorithms might be able to provide the same quality as the user-based one with less computational power [16].

In order to calculate a recommendation, it is necessary to check two parameters: 1) similarity between a target item and user's rated items, and 2) how high the user has rated the most similar items. For example, if the similarity is significant and the user gave a high rating to the item being compared, then the target item can be recommended [16].

Advantages & Disadvantages

An advantage of using collaborative filtering is that it can offer subjective data that can for example be based on user ratings, which means that features of items or users are not required. Collaborative filtering systems can also be adaptive, which results in better recommendations over time [11]. However, this technique suffers from the sparsity problem, which is when only a few users have rated an item [11]. The quality of the system also depends on the amount of users [11]. The cold-start problem is also an issue, since a system could consists of many items which no user has rated.

3.1.2 Content-Cased Filtering

Another technique is called *content-based filtering*, which is where a RS tries to recommend items that are similar to items the user have shown an interest in at an earlier point [14, 11, 15, 18, 12]. A content-based filtering RS works by calculating the similarity between items based on the chosen features [11, 12]. The similarity can be calculated in various ways such as keyword matching or by representing the content in a vector space model (VSM). A content-based RS can be broken down into three main components; a *content analyzer*, a *profile learner*, and a *filtering component* [18]. Figure 3.1 shows how these components work together.

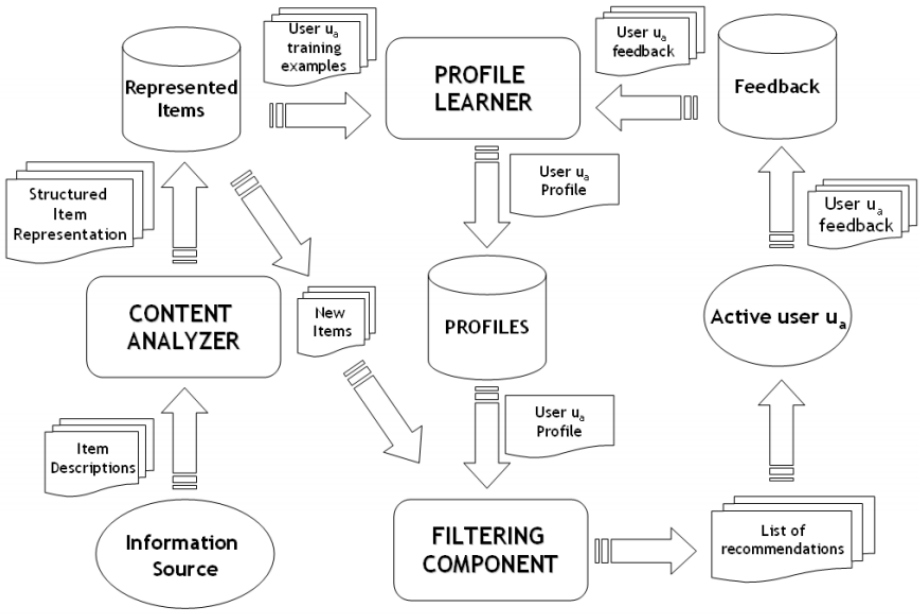

Figure 3.1: Illustration of a content-based architecture [18]

The content analyzer receives the necessary information needed for the system and processes it in order to create structural information that describes the content. This could e.g. be extraction of keywords, structuring them, and then saving them as part of the *Represented items*. They are needed to create a *user profile* before providing any kind of recommendation. This user profile is generated based on the user's opinion on the represented items, which is referred to as *feedback*. Feedback can be either explicit or implicit. Explicit feedback requires the user to describe his or her opinion regarding an item, where implicit feedback is where the opinion of the user is gathered without involving the user's interaction. The filtering component thereafter compares the represented items to the user profile, which it does for each item, and is then able to recommend a set of items to the user.

Each of the represented items usually consist of a collection of features that are used. If for example a RS was based on blog posts from a website, then each blog post would be considered an item while title and content could be considered features. Represented items and a user profile are often compared using VSM [18]. The elements in the VSM are usually filled with weights based on the frequency of words in each document and over all documents. The most common method used to weight terms is using term frequency and inverse-document frequency.

Advantages & Disadvantages

Content-based RSs have advantages and disadvantages, just as any other RS. Some of the benefits are that the technique does not have the cold-start problem [19], and the technique is efficient in providing recommendations for new items [20]. Another advantage is also that there is no bias towards popularity, which can occur in e.g. a collaborative system where the most popular items are more likely to be recommended compared to those with less popularity. However, content-based RSs can provide very obvious results since keywords are used [20], and the recommended items can often be very alike [19]. Another disadvantages, referred to as serendipity [12], is that content-based systems can have a hard time recommending novel content, because the system will recommend items which are similar to the user profile. If for example a user only showed interests in drama movies, then the system would most likely recommend other drama movies instead of a movie in another genre.

3.1.3 Knowledge-Based Filtering

The two more popular RSs, collaborative and content-based, are not always appropriate to use, and an alternative is to use a knowledge-based RS. Knowledge-based systems rely more on explicit feedback, which means that users could be asked to answer questions or fill out rating scales that can be used for the recommendation [13, 21]. There are two types of knowledge-based recommender systems - case-based and constraint-based. [2, p. 162].

Case-based RSs work by looking into the past each time a new recommendation has to be made, as case-based RSs use previous cases which are similar to the new recommendation request [22]. The user provides explicit feedback and if the user's 'case' is similar to an old case, then it can just recommend the old case to the user. There are of course many old cases, and the system will most often choose a set of the most relevant cases, and if the user does not like the recommendations received, then he can often adjust his feedback to get more accurate recommendations [22].

A constraint-based RS is used when there are requirements that have to be met before users get a recommendation [13]. An example could be that a user wants to buy a car, and the RS asks him a few questions about what car he needs. He explicitly specifies that the car has to be big enough for six people, since that is the size of his family. As with case-based it is also possible for the user to adjust his input if the recommendations are not satisfactory.

Advantages & Disadvantages

No matter which type of knowledge-based RS is used, there will always be an interaction with the user where some input is required, this means that knowledge-based RSs do not suffer from the cold-start problem [21]. A disadvantages is that domain expert knowledge is often required and it can take a lot of time to transform available data into a format which the system can use [11].

3.1.4 Demographic Filtering

Another type of RS is based on demographics, where items are recommended based on the user's demographic [12]. Age or gender are examples of characteristics that a demographics RS can use. This type of recommendation is quite easy to add to a system and it is seen on many websites today. It is used where it can be assumed that users with the same demographics are interested in the same items.

3.1.5 Hybrid-Based filtering

All RSs have some advantages and disadvantages which can make it hard to choose the best type for a system. However, hybrid recommender systems is where two or more types of RSs are combined in order to get better results. It can often help decrease the cold-start problem in some situations [11]. There are seven types of hybridization methods [11]:

- **Weighted** - Where the numerical score of different RS techniques are put together in order to end up with only one numerical score.
- **Switching** - A switching method is where a system can switch from one RS technique to another in some situations. An example can be that no recommendation could be provided with collaborative filtering, then it would switch to content-based filtering.
- **Mixed** - Mixing is used when a system should show multiple RSs results simultaneously.
- **Feature Combination** - Where features from different RSs are used in one algorithm. An example can be that the results of a collaborative RS can be used as input for a content-based RS.
- **Cascade** - This method consists of two RSs, where the first one gives a coarse ranking of candidates, and then the other is used to refine the set of candidates.
- **Feature Augmentation** - This technique uses a RS to rate or classify an item, and this information is incorporated into another RS.

- **Meta-level** - This method resembles feature augmentation, but instead of only using the output of the first technique, it uses the whole system as input.

3.2 Education Information

Whichever type of RS will be chosen for this project it might end up relying heavily on initial input data in order to provide proper recommendations to the users. In addition, the data might be used for the presentation of the recommendations.

3.2.1 Collection

Depending on the availability of the data, there are a couple of different ways of gathering it. Apart from asking for a copy of the data, there are a couple ways to move the data from a remote server to a local one.

3.2.1.1 Application Programming Interfaces

Application Programming Interfaces or APIs is a set of well defined and well documented endpoints. They allow others to access data by abstracting the underlying system and only expose specific elements through APIs that developers can implement into their own service. APIs can be used in many contexts, e.g. it can be used in a web-service, a software library or an operating system [23].

APIs as a web service is usually a transfer of data from a company to an application that utilizes it. Today, APIs usually make use of a RESTful architecture and the lightweight data interchange format JSON [24]. As the RESTful architecture utilizes the HTTP protocol it can run on most devices without having to be setup. Web APIs often provide high availability, stability and backwards compatibility as companies are often actively seeking a high usage of their APIs, as the data gathered in return can improve the core service.

For this project APIs can prove useful, as it would make updating the input data easier. For example, a yearly update of grades gathered from KOT might be needed. Using an API could automate everything from gathering to processing and presenting it. However, this requires the APIs not only to be developed, but also to be updated regularly as new educations are introduced include endpoints with relevant and useful data available.

3.2.1.2 Web Scrapers

A web scraper can be used if the data is available online on websites, but there is no API provided to access the data. Web scraping is the act of having a web crawler to analyze the HTML file of a website and look for a specific element, e.g. a class, id or object type. This allows the crawler to extract information from the website, which can be used to give structure to unstructured data so it can be stored and analyzed [25].

Web scraping can prove to be more manual labor than using APIs as they require meticulous setup to ensure that the data they scrape is correct. In case the data sources does not provide an API the data sources should be analyzed carefully in order to see if they support web scraping. If not the data might have to be gathered fully manually, which could prove to take a long time.

3.2.2 Sources

In order to provide data for not only the RS but also to the user, information about each individual education have to be sourced. Fortunately, there are several online sources that provide various kinds of information about educations.

3.2.2.1 UddannelsesGuiden.dk

UddannelsesGuiden (The Education Guide) is a website provided by the Danish Ministry of Education. It contains information of over 2.000 educations and jobs. The list of educations includes primary, secondary and post-secondary educations. The primary function is to guide people interested in attending an education in Denmark as well as study counsellors to seek information [26]. The descriptions of the educations contain various kinds of information that can be integrated into a RS. Most notably are the compact but information rich descriptions and career possibilities that can be beneficial in a content-based RS. However, it also includes educations form (group work, self study, etc.), admission requirements, required grade from secondary education to be admitted, location, and similar educations that can be used in a constraint based RS. Schools and universities are required by law to report this information to UddannelsesGuiden, this ensures the data is always up to date [27]. UddannelsesGuiden does not have an API for retrieving information, which means a web scraper is needed to extract information from the website.

3.2.2.2 Aalborg University's Website

Aalborg University's website (AAU.dk) offers descriptions of each education and any specialization if the education has one. Generally, the data is very unstructured and can contain anything from a few bullet points and videos to long and elaborate descriptions. Similar to UddannelsesGuiden, this website also describes admission requirements, location and career possibilities. However, here it is also possible to find relevant masters educations and information about internships and studying abroad, which could be beneficial in a constraint-based RS. A web scraper would also be needed for AAU.dk since no API for it exists.

3.2.2.3 UddannelsesZOOM

UddannelsesZoom (uZoom) is a new education information system by the Ministry of Higher Education and Science that is made to help choosing the correct education. It is possible to compare up to three educations at once. The data available includes the following [28].

- Completion time
- Dropout rate
- How current students like various aspects of the education (Teachers, social environment, etc.)
- Education form
- Income, employment rate, career possibilities, etc.

All data from uZoom is publicly available in a Microsoft Excel file [29], which means importing it into the prototype should be fairly easy. However, as the service is rather new, a lot of educations are still lacking data. New educations are discovered through a search query on the education names, which is why it should be treated as a data source and not a method of discovering new educations.

3.2.2.4 Den Koordinerede Tilmelding

Den Koordinerede Tilmelding (The Coordinated Enrolment), or KOT for short, was developed to coordinate enrolment into post-secondary educations in Denmark to ensure that students can apply for up to eight educations, but may only be enrolled in the one with the highest possible priority. Every year the Ministry of Higher Education and Science release the numbers for the latest round of enrolment with publicly available data dating back to 1977 [30]. These numbers describe the following [31].

- Amount of accepted students

- Amount of students accepted on standby
- Total number of applicants
- Applications with the education as first priority
- Admission grade for accepted students
- Admission grade for students on standby

As the grades are available many years back it is possible to get an idea of the average admission grade. Using the average grade could possibly enable constraint-based filter features, such as filtering educations based on the users' grades. No API was found on KOT data and accessing the data would therefore require a web scraper.

3.3 Existing Solutions

3.3.1 Studievælgeren

Studievælgeren (The Study Picker) is a web service designed to help people who are looking for post-secondary educations to apply to [32]. The services excels in helping students enlighten their future expectations and considerations for the education such as career possibilities, salary and study abroad. In addition, the service implements some capabilities of constraint-based recommender systems, as it can filter geographical study locations based on the admission grade from last intake.

3.3.2 Adgangskortet

Adgangskortet (The Admission Map) is a service designed to show students which post-secondary educations they can enter based on the courses they completed in their secondary educations. After the users input their courses and difficulty the 12 major industry categories will update to show how many educations are available within the respective industry [33]. If needed, the industry categories can prove useful if the educations need to be classified together in groups.

3.4 User Preferences Management

In recent years it has been common to hear about companies of all sizes having their databases compromised and thus leaking personal identifiable information (PII) on the Internet for the public to access. Examples for PII are name, address information, personal characteristics, etc. [34]. To avoid a similar scenario happening for this service, there has been put emphasis

on what information and data to collect, store, and use. Even more focus has been put on the topic with the new General Data Protection Regulation (GDPR) from the EU Parliament. The GDPR focuses on strengthening data protection across EU countries.

3.4.1 General Data Protection Regulation

The General Data Protection Regulation (GDPR) is a legislation that puts more emphasis on protecting the citizens data. The legislation is affecting companies within EU, but it also affects companies outside EU, which are collecting data about their users. This means that companies outside EU also have to take the GDPR into consideration. Companies who do not abide to the legislation can be fined up to 20 million euros or 4% of their annual global turnover (whichever is the highest) [35].

GDPR also requires companies to fulfill some specific requirements, such as implementing a concept called Privacy by Design (PbD). PbD has seven founding principles, which all place responsibility for the correct collection, handling and usage of user data on the companies. This means that, when the GDPR becomes active, companies who does not abide by it can receive a penalty, so implementing the seven principles or patterns from e.g. [36], would help avoid not being able to abide the legislation Hence assuring that the PII or any other type of data collected will be handled, used and collected in the correct manner. Furthermore, GDPR also emphasizes that the consent forms should be readable and understandable for everyone, and that the data subjects at anytime have the rights to ask the company to delete any information collected about them. One of the only instances where this might not be applicable is if the state deems the information collected helpful for the greater good.

3.4.2 Security

When HTTP packets are sent across a network, a danger exists that those HTTP packets can be captured by a third party. It is important to make sure that sensitive information transferred over the Internet, e.g. from a user to the server, is protected from "man in the middle" attacks, to avoid a third party accessing it. A man in the middle attack refers to a widespread type of attacks where the attacker is using various techniques that can capture data packets traveling between two computers on the network from a client to server as it is shown in figure 3.2.

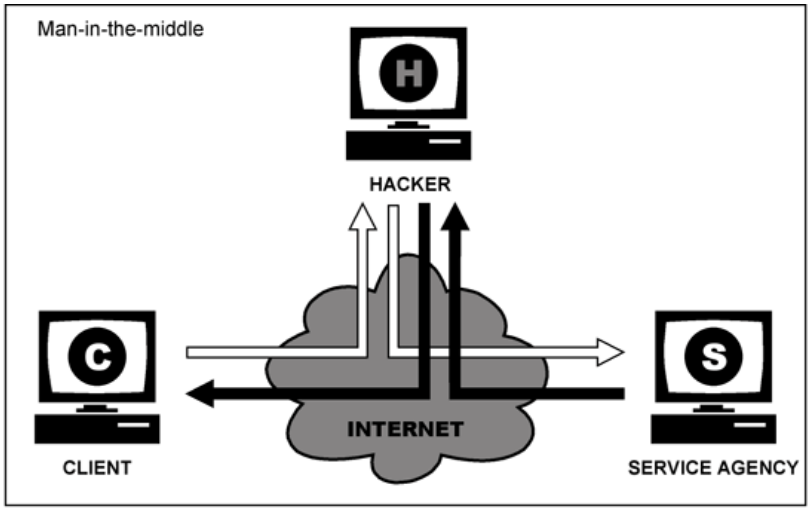

Figure 3.2: A visual presentation of "Man in the middle" attack [37]

However, it is possible to secure the HTTP packets and make sure that a third party is unable to read a message's content or alter them without it being detected. The current state of the art protection from these types of attacks is a Secure Transport layer which can be created by using the security protocols Secure Socket Layer (SSL) and Transport Layer Security (TLS). The SSL layer resides between the TCP and HTTP layers on the OSI stack, as shown in figure 3.3.

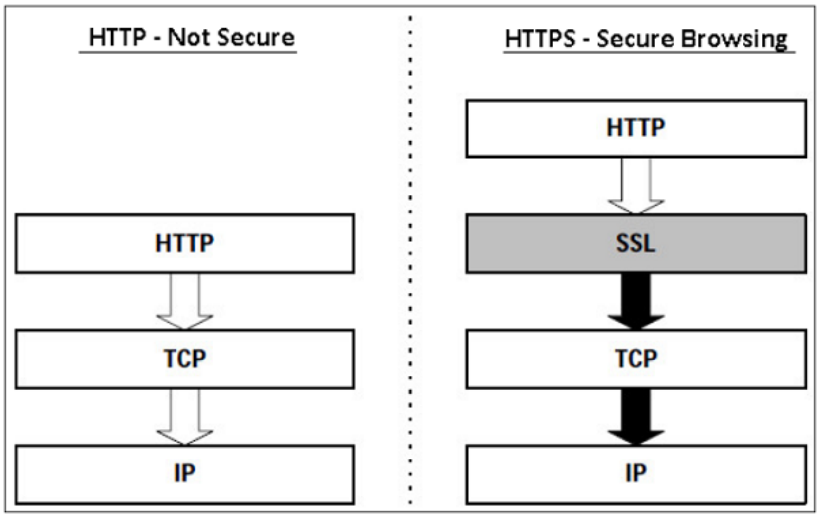

Figure 3.3: SSL position in OSI stack [38]

SSL and TLS provides means to enforce message confidentiality, integrity and authenticity

[39]. Both protocols provide confidentiality by leveraging symmetric cryptographic techniques, thus encrypting the data being exchanged by the communicating parties. The symmetric cryptographic keys are unique and created for each session using the public-key encryption technique during the connection establishing process, which is also called the "SSL handshake" process. Thus no one can can capture the symmetric encryption key, even if the attacker captures all of the messages between two parties. Encryption provides confidentiality of the message, and guarantees that the message can not be read by a third party. Message integrity can be verified by checking the message's hash [40]. But the client also needs to know that he communicates with the right entity (server is who it claims to be), so the server provides the means to verify its identity in form of a certificate which contains information such as the issuer, digital signature, and the owner information. The server's identity can be verified by the client using a public key infrastructure [41].

3.4.2.1 Practical Considerations

From a practical point of view, two components are required in order to secure a website with SSL, namely a web server that supports SSL, and the security certificate itself. Today, all major web servers software provide SSL support, for instance Apache, Nginx, and Lighthttpd have SSL support out of the box. If one is building a self-hosted web application with a built-in web-server, then there are a lot of libraries and frameworks to choose from for SSL support implementation.

When it comes to obtaining an SSL certificate there are two ways of doing so. One can generate and sign certificate himself using "OpenSSL", which is a software utility [42] or get a certificate from a certificate authority. The problem with a self-signed certificate is that it is not issued by a trusted Certificate Authority, and thus will not be trusted by the web browser. Such certificate will not be working in production, since the users will be getting warnings displayed in their user agents. If an SSL certificate should be used in production, then it should be obtained from a certificate authority, as this would result in no warning being shown. Before issuing the SSL certificate, the certificate authority performs a validation of the certificate buyer. There are three different levels of validation available on the market. In *Domain Validation* the certificate authority (CA) only checks if the application has the right to use a domain. In *Organization Validation* the CA checks if the application has the rights to use a domain and conducts some vetting of the company, whereas in *Extended Validation* the right to use a domain is checked in addition to a thorough company vetting [43].

4 Analysis

In this chapter the different possibilities of creating a RS based on the exploration in chapter 3 will be analyzed. The analysis will be a combined effort in trying to specify the different requirements, for such a system to function. A presentation of how the system is expected to function will be shown in a system context diagram and will be elaborated upon. Scenarios will be used to generate a use case diagram to define the functionality that the system should have. Security and privacy aspects of such a system will also be analyzed. At the end of this chapter, requirement specifications will be formulated with argumentation of why the system should be able to perform the defined tasks. The MoSCoW method will be used in order to rank the requirements.

In figure 4.1, a system context diagram is used to visualize how the system should function and what terminators the system communicates with. Four different terminators are defined; a User, Education Institution, Other data sources, and Administrator. The user provides the system with a user profile and in return gets a recommendation on educations that aligns with the provided user information. Based on what we know from section 3.1 about RSs, different inputs might be needed to make a more specific recommendation. This will be analyzed in the Recommender System section below. The system should be able to fetch information from the Educational Institution, as educations might change over time. The Administrator should be able to do maintenance of the system.

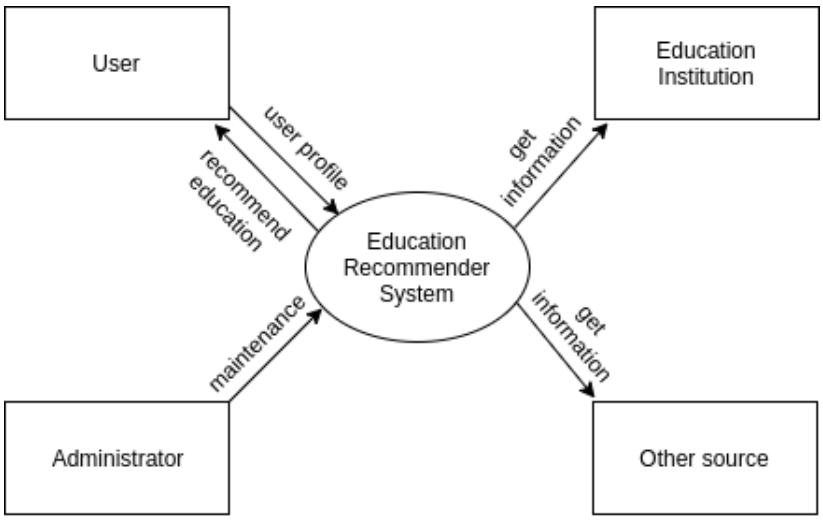

Figure 4.1: A System Context Diagram for this reports RS

In table 4.1 the different terminators are presented with a short description of what they do. If there are no users, the service do not provide any recommendations, and the same can be said about education institutions and other sources, if they do not provide information, then there is nothing to present or base the recommendation on. If the administrator is not able to maintain the service, and at some point fails, the service might become useless if the information about the educations is not correct.

Table 4.1: An overview of of the different terminators from the system context diagram with a description

Terminator	**Description**
User	The user that provides user information. In return, the user gets a recommendation
Administrator	Administrates the service, and has the job of maintaining the system
Education institution	The different education institutions, which contains information about their educations
Other sources	Other sources is where the system is able to collect education information from

The communication between the different terminators and the system are called flows. Descriptions of the flows in the system context diagram can be seen in table 4.2.

Table 4.2: An overview and a short description of the flow in the system context diagram

Flow	Description
User profile	The user communicates with the service, by providing user information. The system uses this information to perform the 'Recommendation education' flow.
Recommend education	Based on the user information sent to the system, it fetches information from the Educational institutions in order to present the recommendation to the user.
Get information	In order to show the information of educations, it first needs to get the information from either the Education institutions, or the other sources terminator.
Maintenance	The administrator should be able to do maintenance on the system, if any errors occur.

4.1 Scenarios

Scenarios will be used, as they will aid in creating use case diagrams for the prototype. Scenarios are useful since they are used to describe a set of tasks that the actor should be able to perform [44].

4.1.1 Scenario 1 - User

Bob just finished his high school education and wants to continue on a higher education. Bob goes on the Internet where he knows there is information about the different educations that each of the institutions offer. Bob is interested in several different things, which makes it hard for him to decide what he wants to study, since all of his interest are equally important. On the web service uddannelsesguiden.dk Bob tries to enter his interests, however, the output he receives does not meet his expectations, this is also due to the very specific categorization, such as IT and electronic, where Bob is only interested in the IT part, but is unable to only select that. Bob tries another web service called Studievælgeren, which also is located on uddannelsesguiden.dk. Bob is only able to choose some predefined categories, where he is only interested in one of the subjects in that category. Afterwards, Bob tries the system being discussed in this report where a recommendation system has been utilized which could help suggest educations based on information about what interests Bob has in regards to selecting an education. This could be location, institution etc. Bob inputs several of his

interests in the input field and chooses a location where he wants to study. Bob presses the recommendation button, and is recommended several educations that covers the interests Bob has inputted. The first education in the list catches Bob's attention, and he would like to know more about the education, so he presses the 'show more information' button, and the education information is expanded, giving him a lot of useful details about the education, such as its location, education level and more. Bob is satisfied with the recommendation, and clicks a link, which then directs Bob the the education institutions website where he can collect contact information, so he can contact the institution which offers the education if he has any questions.

4.1.2 Scenario 2 - Administrator

Alice who works as an administrator for the recommendation service discussed in this report receives a notification that some of the functionality is not running as it should. Alice connects to the service and logs into the service through the administration panel, where she is met by an error log generated by the system. Alice has the option to look at it and try to see if it is something she can correct, or if it should be sent to the service providers. After Alice has sorted out the problem, she can see that a new education has been added to the system, she checks if the education has been run through the system correctly and checks if there is any misbehaviour coming from that education. Alice can see that there is a word which needs to be filtered out, which she has to do. After Alice have checked the education and filtered out the word, she approves the education. At the very end, Alice logs-out of the system, because she is finished maintaining the system.

4.1.3 Use Case

Based on the two scenarios, a use case diagram has been created to visualize the functionality the user and administrator have access to. The use case diagram can be seen in figure 4.2, where the user is on the left and the administrator is on the right. The functionality provided for the user might seem limited, however, since this still is a prototype, a focus will be put on proving that the system works and delivers a satisfying result. More functionality and extra features can be added throughout several development iterations on the prototype. As mentioned in chapter 2 section 2.1, we are working with the Lean software development method, and hence this is not as important as finishing the prototype, and essentially delivering it to a customer. The user has the ability to find and education, and then view certain information about that education.

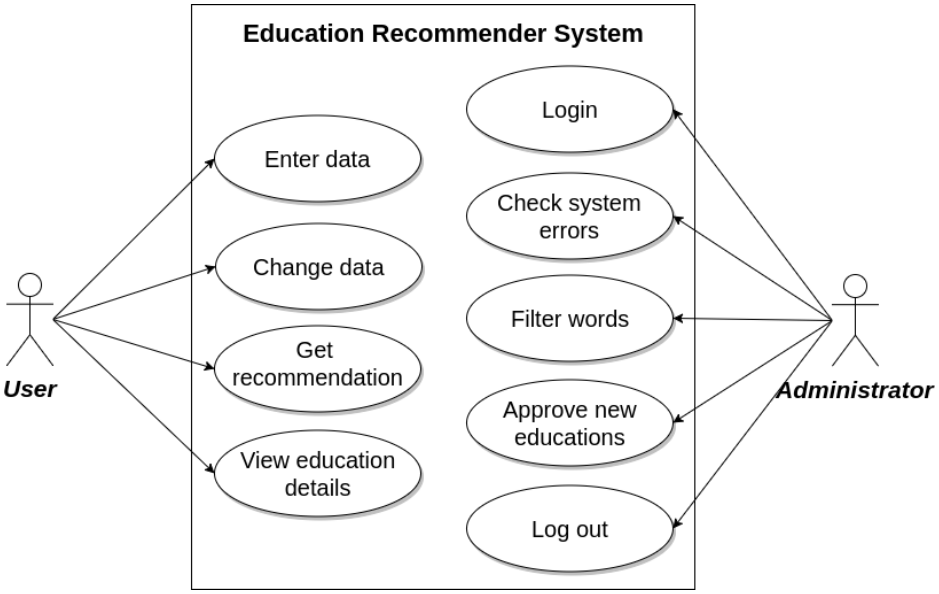

Figure 4.2: Use case diagram for the users and administrators

With the aforementioned use case diagram, three high-level sequence diagrams have been created to give a visual presentation of how different parts of the system could work, and how some of the components communicate. The first sequence diagram can be seen in figure 4.3. This sequence diagram presents the users, and what actions are needed in order to present them with a recommended education.

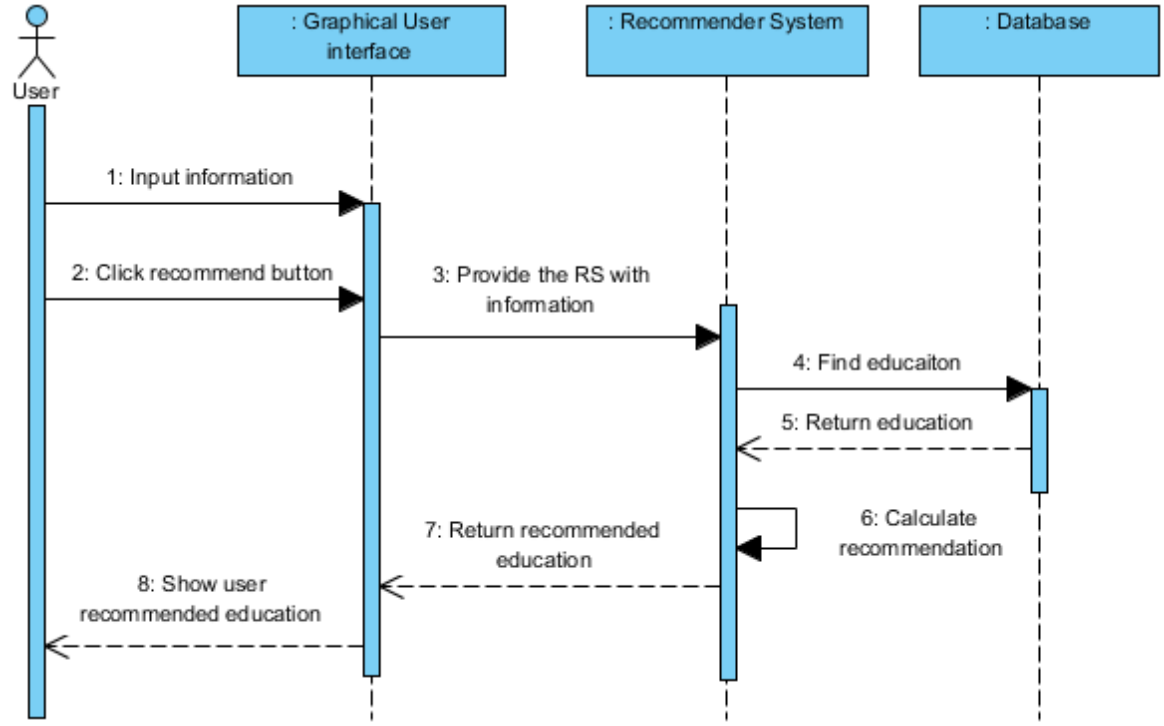

Figure 4.3: Sequence diagram for the user where a recommendation is returned

If a situation occurs where the system is unable to recommend an education to the user, it should have a fall-back mechanism, meaning that the users should receive feedback that they might have to change some of the criteria they have provided to the system. In figure 4.4 a sequence diagram visualizes how this mechanism could be performed.

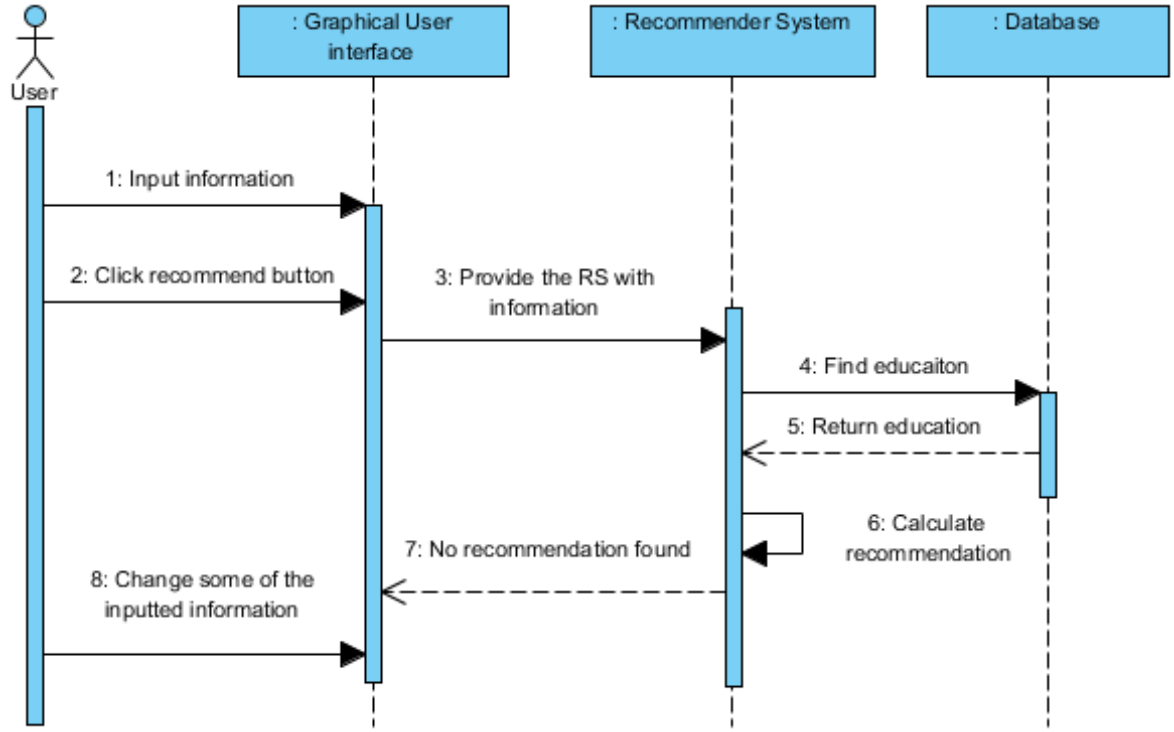

Figure 4.4: Sequence diagram for the user when no recommendation is presented

After no recommendations has been found, the user can change their criteria, and then goes through the process which can be seen in figure 4.3.

A sequence diagram for when the administrator logs in can be seen in Appendix E. After the admin has logged in he has the possibility to filter words, approve educations and check the system error log. How these actions can communicate can be seen in figure 4.5.

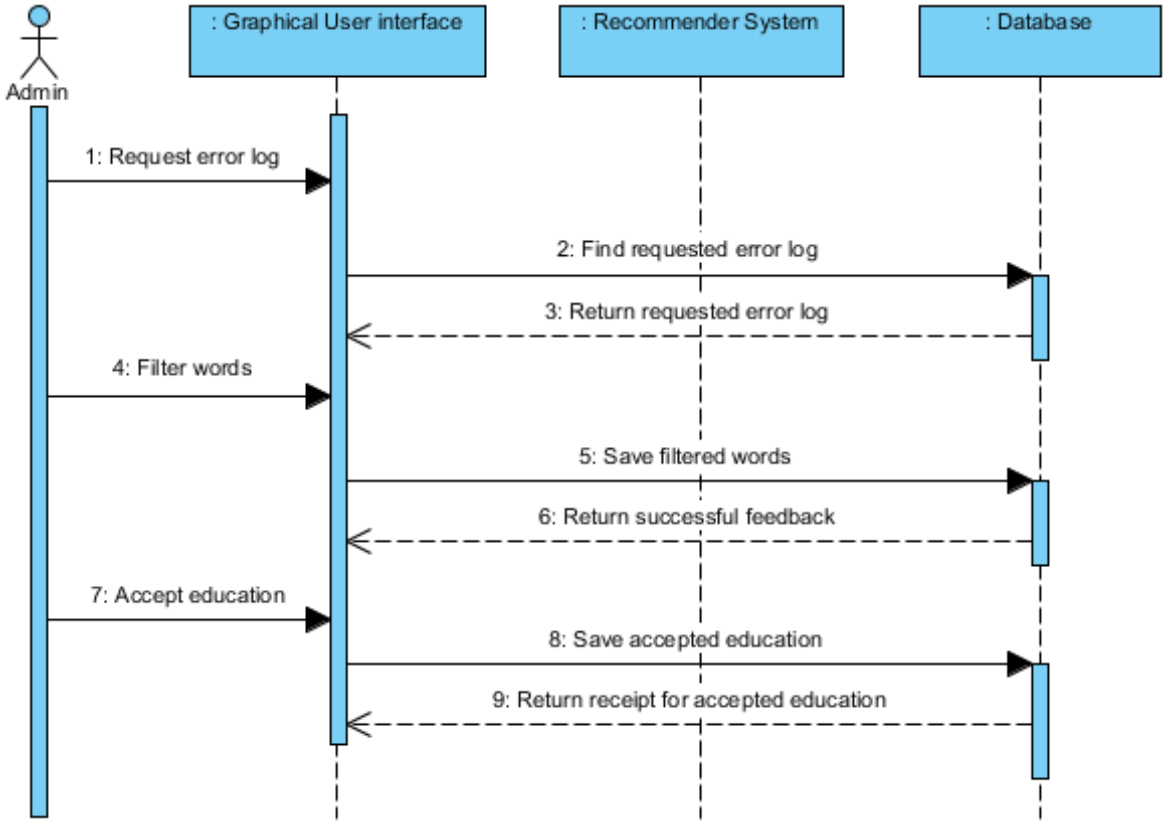

Figure 4.5: Sequence diagram showing the administrator functionality

4.2 Recommender System Technique

As presented in the previous chapter, there are many different RSs that can be used with certain advantages and disadvantages for each individual technique. This section will be used to choose an appropriate RS technique for this project.

The idea of developing a collaborative RS where users can rate or like educations is very interesting, because the current solutions for finding an education, which were previously presented in section 3.3, do not involve other users. However, a user can not rate an education as easily as he can with for example a movie or book. If an honest opinion from the user is needed, then the user would need to finish a 2-4 years long education before being able to rate it. In addition, a collaborative system presents other issues as well. First, it will be hard to gather that kind of data for this project, if not using mock data. Second, the content in educations is often modified and a user's rating might not truly reflect the education if the content changed after a user rated it.

Using a content-based approach would likely be relevant, as there is already a lot of data available from multiple sources, where relevant features could be gathered for most educations. The different data sources were mentioned in section 3.2.2 in the previous chapter. Some examples of features that could be used are title, description and location. However, this approach may require that the user has to provide the system with explicit feedback, which could be ratings or keywords in order to create a user profile. Content-based RSs can struggle with recommending item to new users [20], but this depends on how the system works. If users explicitly answers questions in order to get a user profile, then a benefit is that there would be no cold-start, because information about educations or items is already available and the user provided enough information about themselves for the system to create a user profile.

Using a demographics-based RS could perhaps be useful when it comes to recommending some educations, as the variance between male and females can be very high in some educations, however, none of the data sources mentioned in section 3.2.2 in the previous chapter provide any demographic information about applicants or students enrolled.

Knowledge-based RSs do not try to make a user profile of a user over a longer period of time, but merely aim at comparing a user's profile, which is based on explicitly defined preferences [11]. The two knowledge-based techniques, both case-based and constraint-based, seem applicable for this project. Constraint-based might be of more value since some users may want to find educations based on more strict requirements. An example of why the constraint-based technique is relevant for this project could be that a person already know that his or her education institute has to be located in Copenhagen, and therefore it would be relevant to only recommend educations in Copenhagen. Since the case-based technique requires already established cases in order to function. And since no cases are available, it will not be used, however, if any cases were available it would have been beneficial to take advantage of this method.

Based on the multiple RS techniques presented in section 3.1, it has been decided that a hybrid approach RS would be appropriate for this project. The hybrid approach will consist of a combination of content-based and knowledge-based, where the focus of the knowledge-based part will be on constraint-based. The hybridization that will be used will be weighted, which is mentioned in section 3.1.5. This allows the user to choose the importance of the weights themselves. Throughout the rest of this report the term constraint-based will be used when referring to the knowledge-based part of the RS.

4.2.1 Features

As presented in chapter 3 section 3.1.2, content-based filtering requires that certain features are extracted from items, which thereafter allows a system to perform calculations to compare it with the user profile. There is no rule for which features are best to choose since most RSs are different, and features can therefore not be the same for every system. Thus, it is not easy to know which features should be used before the system can be tested. Choosing features for this system will be based on what information the existing solutions offer and data sources discussed in 3.2.2, and how educations are presented. These solutions were presented in section 3.3, and the solutions provide most of the same information as discussed in section 3.3. In addition, websites such as UddannelsesGuiden and uZoom use similar data as the features chosen. Table 4.3 shows what information is available from the different sources. All of these can be used as features for the RS.

Table 4.3: Overview of chosen features and possible sources it can be gathered from

	UG.dk	**AAU.dk**	**KOT**	**uZoom**
Title	✓	✓	✗	✗
Description	✓	✓	✗	✗
Education form	✗	✗	✗	✓
Location	✓	(✓)	(✓)	✗
Education level	✓	✓	✗	(✓)
Grade	✓	✓	✗	(✓)
Institution	✓	✓	✗	(✓)

The combination of a content- and constraint-based RS means that some of the features will be used purely as content-based, and other features will be used for the constraint-based part. The two features that were deemed most important were education titles and descriptions. The title of an education can be very informative and should express what the education is mostly about. The description of an education contains keywords that describe the main topics of the education. The location can be one of the main criteria for some, since they might already know that their education has to be in e.g. Aalborg. Education level is important for all applicants, since they usually know whether they want to start a bachelor or masters. Titles, descriptions, education level and location can be sourced from the university website as discussed in section 3.2.2.2. Education form is also relevant if a person cares about whether the education requires group work, project work, or if it is mostly self-study. Grades can be highly relevant as well, as some educations require a

minimum grade in specific courses or in average. The grades can be found on university websites and through KOT's data as discussed in section 3.2.2.4. This can be sourced from uZoom's data as seen in section 3.2.2.3. Taking all these features into consideration, table 4.4 shows an overview of which feature will be used for content-based and which will be used for constraint-based.

Table 4.4: Overview of features and where they could be used - either in the content-based or constraint-based part of the RS.

	Content-based	**Constraint-based**
Title	✓	✗
Description	✓	✗
Education form	✗	✓
Location	✗	✓
Education level	✗	✓
Grade	✗	✓
Institution	✗	✓

Based on the features selected and discussed it is safe to argue that some of the features describe an education better than others. This means that certain weights have to be added to the features in order to make sure that some will be more dominant and vice versa. Determining the exact weight for each feature is hard, as it can be difficult to set a percentage on how important each specific feature is, therefore, the weights will be determined later in chapter 5. The title and education are deemed the most important, as they provide the most unique information about each education. As such, these should be weighted higher than other features.

4.2.1.1 Education Information Gathering

As a content-based recommender system technique has been decided to be the main way of providing recommendations, the system will need information about the educations in order to function properly. Based on the discussion in section 4.2.1, several different types of information needs to be gathered in order to support the RS. Fortunately, all of these are available through the sources discussed in section 3.2.2 and table 4.3. This means that all of them are publicly available through either websites or Microsoft Excel documents uploaded to the Internet. The sources that the information has been decided to be gathered from can be seen in table 4.5. Fortunately, there are several methods to gather the information

as discussed in section 3.2.1. Some requires that they are implemented by the service, e.g. public APIs. However, in most cases where an API can not be used, it is possible to develop a web scraper to copy the information from a remote server to a local database.

Table 4.5: List of information required and the source it is gathered from.

Information	**Source**
Title	aau.dk
Description	aau.dk
Education Form	uZoom
Location	uZoom
Education Level	aau.dk
Grade	KOT
Institution	uZoom

As both table 4.4 and 4.5 show, the primary sources for the content-based RS will be the website for Aalborg University, as the titles, descriptions and education level will be collected from there, and uZoom will be used for the education form, location, average duration and institution. Finally, KOT will only be used to collect the grade required to be admitted in recent years. Both uZoom and KOT have the data used by their services uploaded as Excel documents to the Ministry For Higher Education and Science website. This makes it easy to collect the data as it is available through the same web page yearly. As the data is only available through Excel files it is not possible to use a web scraper to gather the data, however, accessing both the files and the data within is possible through scripting.

The data for both the education titles and descriptions will be collected from Aalborg University's website. It is possible to collect all the descriptions from here through a web scraper. As the website is built using a CMS system all web pages follow the same document object model and the same naming convention for the URIs. This makes it possible to develop a web scraper to access a specific element in a specific HTML page and save it a database.

4.2.1.2 Content-Based Processing

After the titles and descriptions have been collected the data needs to be processed in order to extract meaningful classifications, keywords or categories that can be used to construct the vectors that describe each education. Before the description and titles can be processed, it can be beneficial to reduce the size of the feature space, as the computational power required

will be reduced. There are several methods of doing so. In information retrieval the most common ones are stop words removal and stemming.

Preprocessing Stop words refers to the most common words in a language. As these words are usually used several times over all descriptions. In the English language, some of the most common stop words to filter out are "The", "a" and "is". There is no definite list of stop words that can be applied to all systems, as some words might be classified as descriptive in one domain but not in another. For this project a list of 558 stop words was applied and excluded before processing. This list can be found in Appendix D.

Another way to reduce the feature space is by using stemming. The idea of stemming is that in some languages words often contains a stem. The stem of a word is the main body that contains the meaning of the word. Stemming algorithm remove letters from words in order to get the steam of the word, or the essence of the word [45]. The main principle of stemming is to remove common suffixes in the language from words to get the essence of the word. This ensures that two words with the same essential meaning, however spelled differently due to e.g. different tenses or comparison, can be filtered. An example of stemming is the words "speaker" and "speaking". After being stemmed they both turn into "speak". While this can be very useful for reducing the feature space in some languages, others it might not work as well due to the nature of it. In Danish many words are made by concatenation of two words. For example, if two nouns are mentioned after each other they are contracted into one word. While stemming could slightly reduce the feature space in this project, it has been decided not to be used due to nature of the Danish language.

Depending on the size of the feature space, it might become large enough that it would be unreasonable to expect the user to input enough words to provide a proper recommendation. An alternative to stemming is to categorize the words in larger general groups and presenting these to the user. For example, the words "programming", "JavaScript" and "HTML5" can all be classified under the group "Software Development". This allows the users to input only a single group, however, that group will then point to several elements in the user vector.

As mentioned, many words in Danish are two words contracted, e.g. "projektarbejde" (project work), "projektgruppe" (project group) and "projektopgave" (project assignment) all have the essential meaning in the word "projekt" (project). Classifying these similar words that are made from contractions can prove useful in simplifying the feature space presented to the user. Categorizing the words will prove useful since it was decided to not use stemming. While categorizing will most likely require significant amounts of manual labor, it is a useful technique to avoid duplicates of words such as "mathematician" and "mathematics". Due to

this it was decided that the prototype should make use of categorizing. Categories are not based on the structure of the word, but the meaning of it. A major difference between stemming and categorizing is that stemming is done before weighting the terms where categories are done after weighting.

Term Frequency Inverse-Document Frequency One of the most common way of weighting terms in a document is by using term frequency inverse-document frequency (TF-IDF) [18] as written in section 3.1.2. TF-IDF is a powerful algorithm as it takes the frequency of the term over the entire set of document as well as the frequency within each document. This makes it possible to compare documents that are different in length and it ensures that terms are weighted compared to the usage over the entire collection of documents. If a term is used a lot in a single document it should have a high weight. However, if the term is used a lot over the entire collection of documents the initial high weight is compared with the frequency of the entire collection of documents. The TF-IDF weights are usually stored in a vector space model (VSM). In this case, the VSM is representing the education descriptions in a vector space, where each document is a vector in an n-dimensional space, and each query is also a vector in the same space. Each dimension of the vector relates to the terms that are in a collection of documents, in this case the education descriptions. Eventually every document will consist of a vector with weights for each term in the document, which is used to determine how important the term is to the document. Term frequency (TF) refers to how often a term occurs in a document, which is defined by equation 4.1.

$$tf_{t,d} = \frac{n_{t,d}}{\sum_k n_{k,d}} \tag{4.1}$$

Where

$tf_{t,d}$ = Term frequency for term t in document d
n_{td} = How often the term t occurs in document d
$\sum_k n_{k,d}$ = Sum of all words in the document

It might not always be the case that each document is approximately the same length, hence the TF has to be normalized in order to avoid bias. Therefore, the TF is divided by the amount of words in the document in order to get a normalized value between 0 and 1.

Inverse Document Frequency (IDF) is a way of measuring how often a word is used in a collection. By using the IDF it is possible to reduce the weight of words that are mentioned

often in many different documents and increase the weight for the more rare words. This is relevant since e.g. the word "the" can occur in all documents while "software" might only occur in a few, which means that the word "the" does not really have an important and unique meaning to the document compared to "software". IDF is calculated as equation 4.2.

$$idf_t = log\left(\frac{N}{df_t}\right) \tag{4.2}$$

Where

idf_t = Inverse-document frequency of term t
N = The total amount of documents in the collection of documents
df_t = The number of documents which contain the term

The logarithmic function is used to dampen the values which makes it easier to compare large documents with small documents. After TF and IDF have been established through the collection of documents, it is the then possible to weight each term within each document by equation 4.3.

$$w(t,d) = tf_{t,d} \times idf_t \tag{4.3}$$

Where the weight, $w(t,d)$, of a term will be high if it has occurred multiple times in a particular document, and not that often in other documents. This weight will be calculated for each word in each document and later be used to fill in the relevant words in the vector. An advantage of TF-IDF is that it is powerful to compare two documents of different sizes, as it normalizes the weights. However, a major disadvantage of TF-IDF is that it does not take sentence structure into account. If two words are next to each other then they might be related and should be treated as a phrase rather than individual words. For example, the words "Virtual Reality", with TF-IDF it would be split into "Virtual" and "Reality". The words being separated is not descriptive, whereas together they are highly descriptive. Further, some words might be used rarely over both a single document and over the entire set of documents. An example could be domain specific words such as "Patient". These might receive a high value in some educations such as "Medicin" as they might be used often in that education, however in practice they do not offer any unique descriptions about the educations. To avoid this the TF-IDF values should be screened and filtered after being calculated.

4.2.2 User Information

In order to compare the features stored in the system with user preferences, the user needs to submit it, so that it can be compared to the system features as discussed in section 3.1.2. As described in section 4.2.1.2 the features are stored in vector space models. As such the user preferences should also be stored in a vector space model in order to compare them.

A possible way of doing so is by asking the users directly what their preferences and interests are to generate their user profile. This result would then be stored in a vector that can be compared to the education vectors. However, in order to directly compare the education vectors to the user vectors, the elements in the two have to align. For example, if the word "literature" is in element 1 in a education vector, but element 5 in the user vector, then they are not directly comparable. In addition, if the user vector contains a word that is not in the education vector, then the two are not comparable. As such, the user might only be allowed to input words that are already in the education vectors. This would force the structure of the user vector to be similar to the structure of the education vectors, which makes them directly comparable.

The vector that is created from the users input, will from this point on be known as the user profile. See section 3.1.2 where this is represented.

4.2.3 Vector Comparison

Now that the basics for both the education vectors and user vector have been established it is possible to compare the two. When comparing sample sets there are generally two algorithms that are mostly used; the Jaccard Index and cosine similarity. Both algorithms are tools that can easily compare two sets of data and will output a number between 0 and 1, which corresponds to the similarity between the two data sets. In this case the two sets of data would be stored in a VSM. The Jaccard Index takes the similar features directly into account and is defined as the intersection divided by the union. Where the intersection is the amount of elements the two VSMs have in common and the union is the total amount of elements. The algorithm is shown in equation 4.4

$$j(\vec{d_n}, \vec{u}) = \frac{\vec{d_n} \cup \vec{u}}{\vec{d_n} \cap \vec{u}} \tag{4.4}$$

Where

$j(\vec{d_n}, \vec{u})$ = The Jaccard Index of document d_n and user vector $\vec{u}$
$\vec{d_n} \cup \vec{u}$ = The intersection of document d_n and user vector $\vec{u}$
$d_n \cap \vec{u}$ = The union of document d_n and user vector $\vec{u}$

If the VSMs are 100% similar to each other, the result would mean that the intersection and union are equal to each other, which in turn means that the algorithm would divide 1 by 1 and then return 1. An advantage of using Jaccard is that it is very efficient in calculating binary data sets, e.g. where an element in a vector is either 0 or 1. However, in this case, the elements in the VSMs are weighted by TF-IDF which means they can be any value between 0 and 1. This means that the user would have to input a value of an element exactly similar to the value of the element in any of the education vectors. If the values are not exactly equal to each other they would not be counted in the intersection (numerator part in equation 4.4) and would thus reduce the final Jaccard Index.

Another very common method of comparing documents is by calculating the cosine similarity between them. This method calculates how similar two vectors are based on the angle between them. It is possible to calculate the cosine similarity, Θ, by the following algorithm.

$$cos(\vec{u}, \vec{d_n}) = \frac{\vec{u} \cdot \vec{d_n}}{||\vec{u}|| \times ||\vec{d_n}||} \tag{4.5}$$

Where

$cos(\vec{u}, \vec{d_n})$ = Cosine similarity of two vectors
$\vec{u}$ = The VSM representation of the user profile
$\vec{d_n}$ = The VSM representation of document n

First, the algorithm calculates the dot product, that is the sum of the products of the two vectors. This returns a scalar. However, as the dot product is sensitive to the magnitude it might show that two vectors with a similar direction are not similar to each other, due to one having a larger magnitude than the other. This is normalized by dividing the product of the lengths of the two vectors together and calculate the cosine similarity using the unit vector rather than the regular vector.

As the Jaccard Index is inefficient for measuring similarity between vectors with non-binary elements, it would be difficult to compare the education vector with TF-IDF values against the user vector. Chances of the values being equal to each other are very slim. Cosine similarity on the other hand calculates the difference between two elements instead of valuating them based on a boolean expression. Due to this, cosine similarity will be used to

compare the user profile to the education information for both the content-based RS and the constraint-based RS.

4.3 Security And Privacy

In [36] the PbD (Privacy by Design) principles are discussed, and some more generic patterns which can be used when implementing a ICT solution are presented. The two which will be emphasized in this report will namely be the two called, (1) anonimization and pseudonymization, and (2) data minimization. The anonimization and pseudonymization pattern is about removing the identifiable part and linkage between a user or a person to a set of information or data [36]. The idea behind data minimization is that you only collect, store and use information or data that you need, in order to make the service function as intended [36]. With this in mind, the prototype should only collect the information it needs in order to function, and the collected information should be anonimized and pseudonyimzed. For such a prototype it would not make sense to collect information such as the PII mentioned in chapter 3, since it would not add much value to the actual system, and if the prototype were to collect it, then it would need to abide the GDPR legislation mentioned in chapter 3, section 3.4.1, in order to avoid receiving a big fine. However, some useful data that could be collected are queries from the users, as this could be analyzed in order to optimize the systems' most used features. This data should then be encrypted in order to fulfill the anonimization and pseudonymization pattern from [36].

The ability to exchange data with a server in a safe way is a natural part of the user's privacy. In order to set up a private and secure connection between our server and a user agent we are going to use an SSL certificate 3.4.2. There are different types of SSL certificates, and there are several different ways of obtaining an SSL certificate, as mentioned in section 3.4.2. The minimal viable certificate for this project will be the one working securely in the web browsers, and preferably a free one. It is possible to obtain a working free SSL certificate from Let's Encrypt, which should be used in this project. A Let's Encrypt certificate provides Domain Validation, which works in all major browsers, and is free to obtain.

4.4 Requirement Specifications

It is necessary to figure out what the requirement specifications for such a RS described in this chapter has before it can be developed. This section will include all the requirements, and these will be derived from what has been presented earlier in this chapter. The requirement specifications will be prioritized using the MoSCoW method [46]. MoSCoW stands for **M**ust

have, **S**hould have, **C**ould have and **W**on't have. Where "Must have" is the minimum working product and "Won't have" is the fully implemented solution. The method is a powerful tool in projects that have a fixed deadline as it can prioritize the order of tasks and quickly sort out any non critical tasks out.

The requirements have been split up into two categories; functional and non-functional requirements. The functional requirements are classified as requirements that are critical to the service functioning. Where non-functional requirements are classified as e.g. performance, security, reliability or maintainability related.

4.4.1 Functional Requirements

The functional requirements, which will be presented in this section, will be used to show what exactly the system should be able to do. These requirements are typical related to what the user should be able to do, and what kind of functions are needed in the system. These can be seen in table 4.6.

Table 4.6: Table of the functional requirements

No.	**Requirement**	**Motivation**	**Priority**
FR1	Users have to be able to input their interests	To compare with the education vectors. See section 4.2.2	Must
FR2	Users have to be able to choose education location	To allow the user to include or exclude specific education locations. See section 4.2 and section 4.2.1	Must
FR3	The service has to recommend educations for the users based on their user profile that is based on selected words.	The service should recommend educations based on the user profile. See section 4.2	Must
FR4	The users have to be able to view title, description, education form, location, education level, grade, and institution for each education.	The users should be able to get information about the educations. See section 4.1.1	Must

Continued from last page

No.	**Requirement**	**Motivation**	**Priority**
FR5	Users have to be able to choose education form	The service should be able to filter educations based on education form. See section 4.2.1	Should
FR6	Users have to be able to choose education language	The service should be able to filter educations based on education language. See section 4.2.1	Should
FR7	Users have to be able to choose education level (degree)	The service should be able to filter educations based on the education level. See section 4.2.1	Should
FR8	Users have be able to input their grades	The service should be able to filter educations based on the entry grades. See section 4.2.1	Should
FR9	Users have to be able to choose education institution	The service should be able to filter educations based on education institution. See section 4.2.1	Should
FR10	The service has to scrape a website for information.	The service should be able to fetch information in an automated way. See section 4.2.1.1	Should
FR11	The service has to be able to produce error logs (in case of system errors occurrence) for the administrator to access.	To ensure a reliable service. See section 4.1.2	Should
FR12	The admin has to be able to log-in and out	Access to the backend has to be secured to stop people from the outside to enter. See section 4.1.2	Won't

Continued from last page

No.	Requirement	Motivation	Priority
FR13	The admin has to be able to filter out words	Stop words should be filtered out to increase likelihood of a good recommendation. See section 4.1.2	Won't
FR14	The admin has to be able to approve educations	Newly added education descriptions should be approved in order to avoid fraud. See section 4.1.2	Won't

4.4.2 Non-functional Requirements

The non-functional requirements are all the other requirements which do not fit as functional requirement. They are for example not based on user behaviour, but they are more requirements that the software developer needs to consider when building the system. Some typical non-functional requirements are related to latency, security and scalability among others. The non-functional requirements for this system can be seen in table 4.7.

Table 4.7: Table of the non-functional requirements

No.	Requirement	Motivation	Priority
NFR1	Compare the user vector with the education vector using cosine similarity	Cosine similarity was chosen as the comparison algorithm. See Section 4.2.3	Must
NFR2	The system has to be able to compute TF-IDF	TF-IDF was chosen as the information retrieval algorithm. See section 4.2.1.2	Must
NFR3	The feature weights have to be adjustable	To ensure that the service give the most relevant recommendation. See section 4.2.1	Must
NFR4	The users should be able to connect securely to the service through HTTPS	To ensure that the user's perferences are kept private. See section 4.3	Should

Continued from last page

No.	**Requirement**	**Motivation**	**Priority**
NFR5	The words in the vector have to be categorized.	The words in the vector should be categorized to group together similar meaning words. See section 4.2.1.2	Should
NFR6	The system has to pseudonymize user information	To keep user preferences secure and private. To comply with GDPR. See section 4.2.1.2	Should
NFR7	The system has to store user profiles	To optimize the system. See section 4.3	Won't

5 Design

The way the system's architecture is going to be designed can be seen in figure 5.1. In order for the system to be able to recommend anything for the users, it first needs certain information from them which is going to be provided through the graphical user interface (GUI). In the requirement specification this information can be seen as FR1 and FR2. The GUI then delivers this information to the recommender engine which processes it accordingly to NFR1, and uses information previously saved in the database by the Data processor. The Data processor handled the information added to a database by a web scraper, which is defined in FR10. The web scraper scrapes the data source for useful information. The data processor handles the information from the database and creates vectors, see NFR2. When the recommender engine has received a recommendation, it is sent back to the GUI and shown to the user through the user agent seen in FR3.

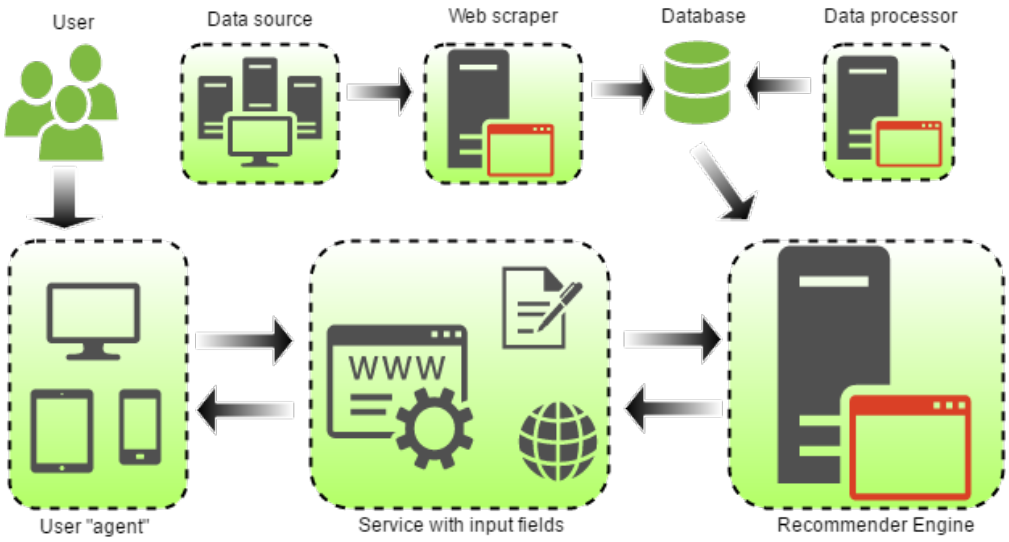

Figure 5.1: Free-form illustration of the system, and how all components communicates

In order to realize what is shown in figure 5.1, we need ways of representing objects to the user and store information in a database. The MEAN stack will be used for implementing this prototype. The MEAN stack consists of four main parts which are: MongoDB database for data storage, the Express framework to process HTTP requests, the Angular UI framework

client-side JavaScript application and Node.js to run JavaScript on the server side. It is a complete stack that provides all the required tools for building a server-side web applications with a NoSQL database and client-side front-end. One of the advantages of this stack is that both server-side and client-side code is written in JavaScript. The LAMP stack is another platform that could be used, however, it requires PHP and renders HTML on the server side while JavaScript would be needed in order to make web pages interactive and dynamic. Using LAMP would require learning both PHP and Javascript. The MEAN stack was therefore chosen, since both the front-end and back-end can be written in one language, which makes it faster for us to implement the prototype in the end.

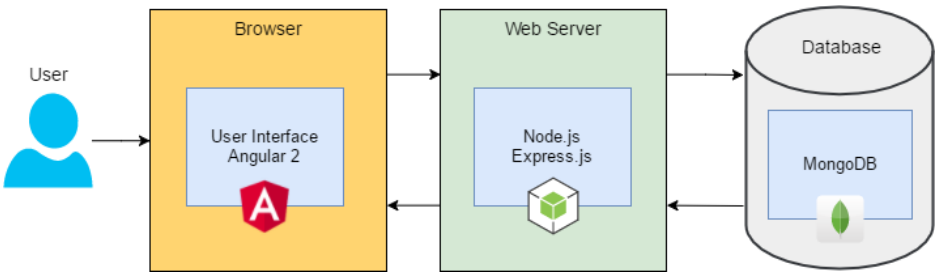

Figure 5.2: Illistration of the MEAN stack's components

5.1 Data Sources

As discussed in section 3.1.2 and 3.1.3 both content-based and constraint-based RSs need information about the content before it can start calculating recommendations. This is specified in requirement FR1 and FR2. Section 4.2.1 discussed what kinds of information each RS technique needs, these can be seen in table 4.4. In addition, the information needed, and sources it can be gathered from, was shown in table 4.5. As seen in table 4.5, three different data sources will be used. Title, description and education level will be sourced from Aalborg University's website. Education form, location and institution will be sourced from uZoom. Finally, the grades will be sourced from KOT, as discussed in section 4.2.2.

The website for Aalborg University does not have an API for easily getting the information into our database. However, the information can easily be gathered using the web scraper. Aalborg University's website uses a common content management system (CMS) called InfoGlue. Using a CMS ensures that all pages look similar, it also relies on the same standard template for the DOM. This means that all pages generally have the same structure in terms of elements position and naming. In addition, the URI's of different educations are also similar, as they too rely on a specific structure. Figure 5.3 shows how the URI structure

from the bachelor's program in biology is similar to the master's program in sociology.

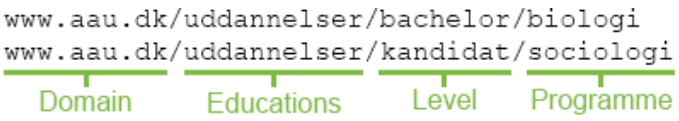

Figure 5.3: Figure showing the similarity between education URIs for AAU.dk

The consistency in URIs and DOMs over both bachelor's and master's programs makes it easy to setup a web scraper and provide it with the URIs and elements to scrape.

Getting data from uZoom and KOT into the system is a little more complex. As neither service have an API and both have their data stored in Microsoft Excel files on their server it is not possible to use a web scraper either. As such the files have to be copied to our server and be prepared for processing for the RS. First, the files need to be copied, then they have to be converted into a format that can be read by the server. Useful information can then be extracted from the files and be saved in the database. Fortunately, there are several tools that are free to use and open source for copying files from one server to another and for converting Microsoft Excel files to e.g. CSV files.

This could potentially be automated by the service, however, that requires the file locations and content of files to be consistent over multiple versions. The data for uZoom has only been uploaded once, which makes it impossible to know if the file format will change next version. Further, the names of the KOT files that contain the grades have changed three times in the last 9 years. This makes it hard to automate, however it is a doable task for the administrator, as it is only required to do once a year when new data is made public.

5.2 Web Scraper

As discussed in section 4.2.1.1 and section 5.1 some information required for the content-based RS will need to be sourced directly from web pages on Aalborg University's website, this is based on FR10. Fortunately, the website relies on a CMS system, which makes the DOM on all pages similar to each other. Before the web scraper can start fetching data, it needs to know from what URIs it should fetch data from. Fortunately, Aalborg University's website has a site with links to all bachelor's programs and one for all master's programs. These will be scraped first in order to produce a list of links to all educations. Listing 5.1 shows a snippet of the structure of the website `www.aau.dk/uddannelser/bachelor` which

contain links to all bachelor's programs. The web page for master's programs have the same structure.

```
<div class="box white">
   <h2><a id="A_undergraduate"></a>A</h2>
   <ul class="unstyled studyProgrammes">
   <li><a href="http://www.aau.dk/uddannelser/bachelor/anvendt-filosofi">
   <h3 class="hasArrow">Anvendt filosofi</h3></a>
   <p class="description">Læses i Aalborg</p>
   </li>
   <li><a href="http://www.aau.dk/uddannelser/bachelor/arkitektur-design">
   <h3 class="hasArrow">Arkitektur og Design</h3></a>
   <p class="description">Læses i Aalborg</p>
   </li>
   <li>...</li>
   </ul>
</div>
<div class="box white"></div>
<div class="box white"></div>
```

Listing 5.1: Snippet of the structure of www.aau.dk/uddannelser/bachelor.

Each of the elements with the class "box white" contains educations starting with a specific letter. The structure continues from A to Ø. This makes it possible to build a web scraper that looks for the first <a> element within each <ul> element and save it to a file. This list will be used as input for the web scraper that will gather education information.

As all the educations on the website have the same structure, they also all have the same sub web pages. One of them being "Fagligt Indhold" (Academic Content). These pages contain, as the name implies, descriptions about the academic content of each education. These descriptions will be used as the input data for the content-based RS as described in section 4.2. Fortunately, it is easy to access the academic content subpage, as all that is required is adding the suffixes "/fagligt-indhold/" to the URIs scraped earlier.

As the academic content pages are a part of the CMS they too share a structure across educations. As such it is possible to setup the scraper to look for a specific element in the DOM and tell it to save the information within this specific element to the database. In this case it is necessary to tell the scraper to look for two elements; first a element with class "spotCon grid g8" and then look for a element with the itemprop "articleBody". The reason for this is that the both elements are used elsewhere on the page. Listing 5.2

shows a code snippet of the academic content subpages. The scraper should be developed such that all content within the are saved to the database.

```
<div class ="wrapper">
   <div class="section">...</div>
   <div class="section">
      <div class="spotCon grid g8">
         <div class="article itemscope itemtype="http://schema.org/Article">
            <div class="box white">
               ::before
               <h2 itemprop="headline">Fagligt Indhold</h2>
               <img src="/digitalAssets/72/72177_colourbox1100848-650x350.jpg" ... >
               <meta itemprop="dateModified" content="2015-03-13T09:56+0100" />
               <meta itemprop="datePublished" content="2013-08-21T10:48+0200"/>
               <div itemprop="articleBody">
               ...
               </div>
```

Listing 5.2: Snippet of the structure of academic content subpage.

5.3 Database

MongoDB will be used since it is a part of the MEAN stack. The database data structure should fit the use case and support the application the best way possible. We have three collections in MongoDB, where each collection serves its purpose. When information about educations is fetched from the data sources, it can be semi-structured and not always in the right format. The first step is to save the education information in the collection, so we have the ability to work with it in a more convenient way. The data required for each education for the service is education name (FR4), location (FR2), education level (FR4), institution name (FR4), and description (FR4). The data processing can start after the education data is fetched and saved in the educations collection. In order to calculate TF-IDF as required by NFR2, and determine stop words, we create an additional collection called *words*. Basically all the words from all educations descriptions are extracted, and placed into that collection. This approach gives us ability to write code in an easier way, and simplify all operations by breaking them down into smaller logical chunks. After TF-IDF is calculated, it is possible to go through the words and define a TF-IDF threshold for when a word is considered a stop word [47]. When stop words are filtered out, the remaining words represent the vector. At the moment we do not know exactly what would be the best way to build a vector. After stop words are filtered out, the system then builds vectors for each education and places

them into a separate collection, in order to improve the performance. This is done in order to compare all the vectors with a user vector. All the data from the collection has to be fetched by the system. Again, the vectors can be placed in the same collection as educations, but it will slightly increase the complexity of the queries.

5.3.1 Data Structure

Figure 5.4 illustrates the relationships between the entities that could have been modelled in the traditional relational database system. But since we chose MongoDB as a primary data storage system, we decided to denormalize the data model, in order to support easier interaction with the database from our application prototype [48, 49, 50].

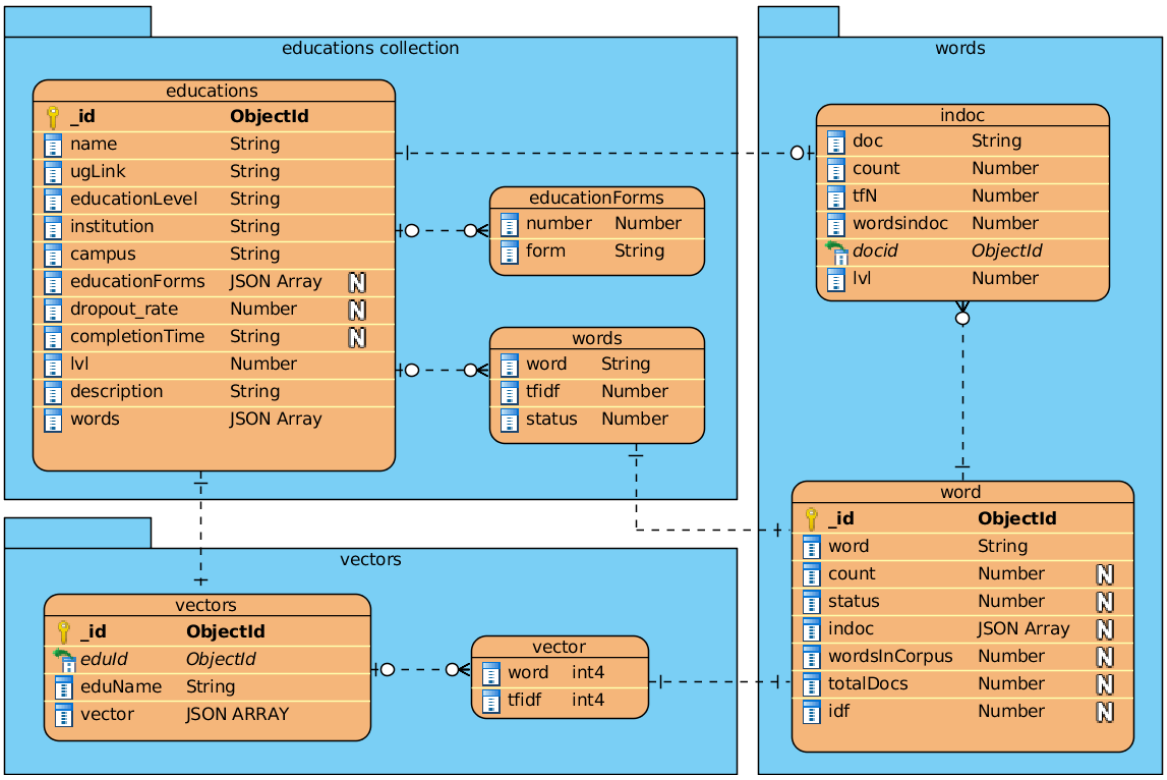

Figure 5.4: Entity Relationship Diagram of how the database should be structured

The blue packages in the diagram represents the collections in the MongoDB. Within every package in the diagram, all entities with one-to-many relationships were denormalized using an "embedding strategy", where we embedded related objects into a main object as JSON arrays. Embedded entities are displayed in the diagram without foreign keys while having

one-to-many relationships. This simplifies queries and selections. The denormalization strategy was chosen using guidelines from MongoDB's official website [48, 49, 50]. The intentional usage for each collection is as follows. The *words* collection is used as a temporary collection that contains all the words from all the education descriptions together with the meta data which is needed to calculate a TF-IDF value. Every *word* entity contains an "indoc" element which is an array of "indoc" entities. Each "indoc" entity have a reference to the education which is modeled as a foreign key and called "docid". The *educations* collection stores data about educations along with descriptions broken down into an array of words, which is stored as an embedded JSON array of objects. Each word contains the word itself, its TF-IDF value, and status showing whether the word is used in the vector or not. This collection serves as a base for constructing vectors. Every time a words status is changed, or new educations are added to the collection, the vectors collection have to be re-created in order to reflect the changes. The *vectors* collection is used in the recommender system, and it is made this way to provide speed and ease of usage from the application side.

5.4 Data Processing

The web scraper accumulates information which is added to the database, but the information is not ready to be used for calculations in the system until some processing has been done. Based on FR1, FR2 the users should be able to input interests and location, which should be matched to educations' description and location. The description for each education has to be processed in order to get the TF-IDF value (NFR2) for each word in every description, and this should be saved in a vector for each education, because this vector can then be used when comparing the user vector with the education vector (NFR1). The location for each education also has to be processed into a vector. The data processing part of the system should also be able to perform another important action before calculating TF-IDF, which is to filter out stop words to decrease the size of the education descriptions.

5.5 Recommender Engine

The system will need to be able to produce recommendations based on a content-based and a constraint-based technique as mentioned in chapter 4 section 4.2, and this section will focus on how the system can be designed for supporting the functional requirements NFR1 and FR3.

FR1 and FR2 shows that the user profile should consist of user interests and preferred education location. This means that the user profile will only consist of interests and education

location. The features mentioned in Chapter 4, which the system must consist of, are descriptions and location. The system therefore needs to make two vectors for the user profile (interests and location) and two vectors for each education (words and location). The location vectors for both the user profile and each education are quite simple and only need to contain three elements [1,0,0], where the first element is Aalborg, second is Esbjerg, and third is Copenhagen. If an element in the user location vector is 1, then that is where the user wants to study. An education can be offered in all three locations, so the element will be 1 for wherever it is offered, this could make an education location vector be [1,1,1]. This means that the location vectors for both the user and education can now be compared using cosine similarity, see section 4.2.3. The description for each education has to be processed where each word in the description will get a TF-IDF value. This will result in a vector with many elements as it will contain all the words in the system. As mentioned in chapter 4 section 4.2.1.2, stop-words should be filtered out before making the vector, which also makes the vector smaller. The vector for user interests has to be compared with the description vector, which means that the user vector needs to contain all the same elements as the description vector as mentioned in section 4.2.2. The vectors for location and words should then be compared to the educations using cosine similarity as discussed in 4.2.3, however, weights have to be added to the two features. Figure 5.5 shows how this can be done.

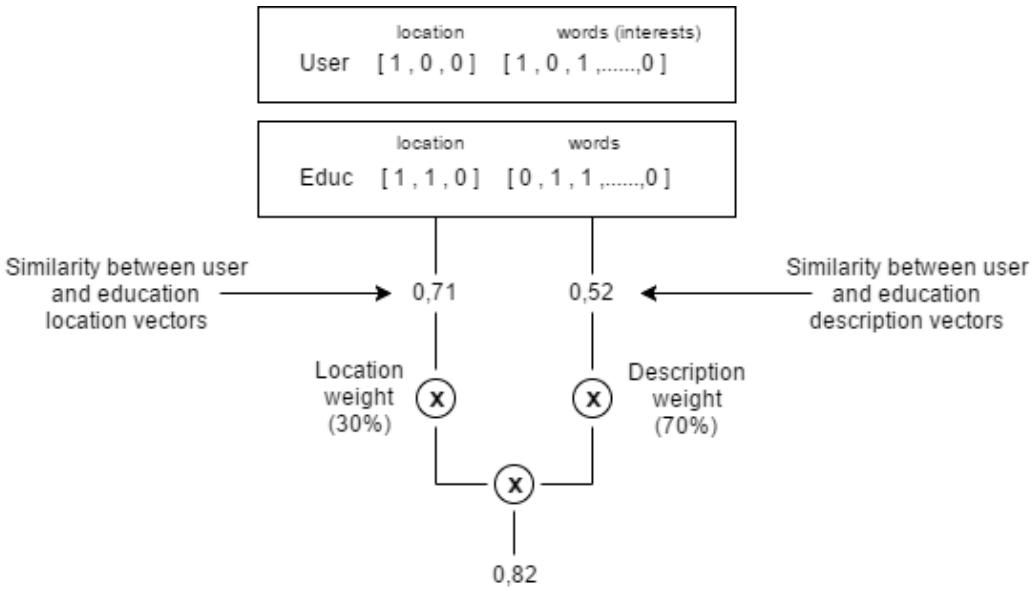

Figure 5.5: Example of how weights can be added to the different features.

Weights should be added after calculating the cosine similarity as discussed in 3.1.5 and

4.2. Figure 5.5 shows that location should have a weight of 30% and description 70%. The cosine similarity result for the location vectors will therefore be multiplied by 0.30, and the description by 0.70. The two values are then multiplied by each other and this results in the final score, where the similarity with the highest number should be recommended. The aforementioned is just an example of what they weighing values could be, which means that the values can be different when implementing the system.

5.6 Web Server

The web server needed for the system will be created using Node.js, and the Express framework will be used on top of Node.js. These are used because they are parts of the MEAN stack, which was discussed in the beginning of this chapter. The system could be created with only Node.js, but Express makes it easier to create the web application and it makes it faster to develop the system as it is simpler to e.g. handle requests compared to only using Node.js. The idea of Express is to write less code and use existing functions while developing the system. As seen in figure 5.1 the web server will be required when users interact with the system through the browser and it is needed for handling requests and routes for the recommender engine.

5.7 User Interface

The user interface will be created using Angular 2, as this is a part of the MEAN stack and is a client-side JavaScript framework. Angular allows building responsive and fast web applications while keeping the code clean and maintainable. This also means that it will be a single-page web application. The user interface (UI) will be simple, in the sense that for the prototype there will only be two input fields, namely the interest and location fields. For introducing convenience for the user, and making sure they apply certain words to the vector, an auto-completer is going to be implemented. This means that when they start typing in their interest, a suggestion of words they could enter will be presented to them. An example of how this could look can be seen in figure 5.6.

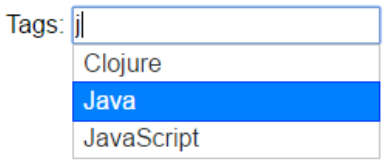

Figure 5.6: An example of how the auto-completer could look like, where the user inputs "j" and are presented with three suggestions: Clojure, Java, JavaScript.

When the user is satisfied with filling out the interests, he can click on a submit button, sending the information to the recommender engine, as it can be seen in figure 5.1. When the user is presented with the recommended education, it should only show information which is useful for the user. This information can be seen in FR4, and will be represented for the user. An example of how this could look can be seen in the figure 5.7 below.

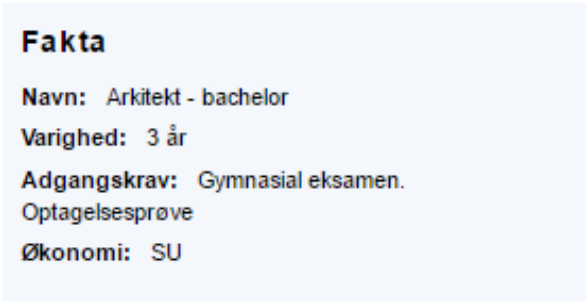

Figure 5.7: An example of how the important information could be presented to the user. The example is taken from ug.dk

If the user is keen on learning more about the education or reading the full description of the education, they will be able to click a button, which then redirects them to details about the education. A visualization of how all of the aforementioned can be combined to create the desired UI can be seen in figure 5.8. Each of the boxes represents different objects. The blue box represents the input field for interests, while the green box is for choosing location. The orange box is the button the user has to press in order to submit the user profile they just created to the system. After the user has pressed the button, the yellow are presented along with the purple box and bright-yellow box.

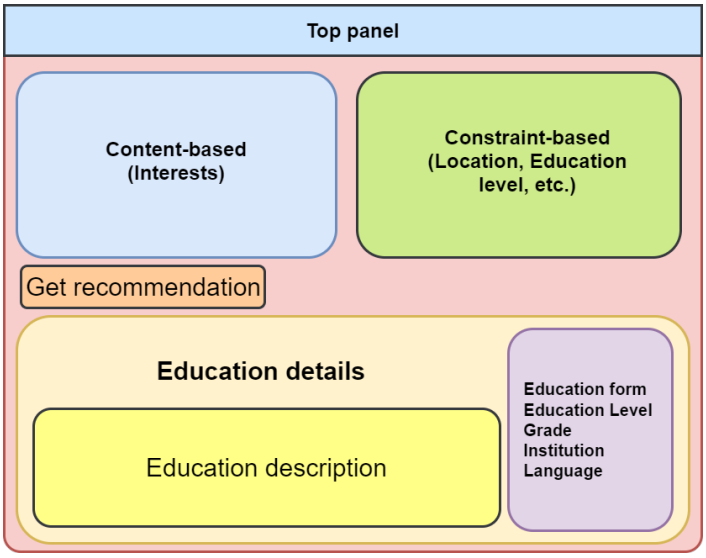

Figure 5.8: An example of the web page could be structured

When designing a web application there are two main patterns which can be utilized in order to guide the users attention to the important areas. These two patterns are the F- and Z-pattern [51, 52]. The F-pattern guides the user to start top-left and then go in a F-pattern. The Z-pattern guides the user to read in a z-like form, starting from the top left corner. For this web application, the F-pattern will be utilized, since we want to make sure that the user goes through all of the different steps, like adding interests, selecting location, pressing the button, reading the education description and finally reading the bullet point information (form, level, grade, etc.).

6 Implementation

This chapter will show how a prototype was developed based on the requirement specifications presented in Chapter 4 and the design in chapter 5. The prototype will not include everything that was discussed in the analysis, since the time limit does not allow it. As discussed in the section 1.3 Delimitations, for the implementation it has been decided to only include 10 educations, five bachelor's educations and five master's educations. Table 6.1 shows the chosen educations. Two of the educations are available in Esbjerg, seven are available in Copenhagen and eight are available in Aalborg. These educations were chosen because some of them are alike and some are different. "It, Communication and New Media" and "Innovative Communication Technologies and Entrepreneurship" are very similar, however the two have nothing to do with e.g. "Socialt Arbejde" (Social Work) or "Anvendt Filosofi" (Applied Philosophy).

Table 6.1: Table of educations included in the implementation.

Bachelor's	**Master's**
Sociologi	Information Studies
Bæredygtigt Design	Socialt Arbejde
It, Communication and New Media	Innovation Communication Technologies and Entrepreneurship
Kemi og bioteknologi, diplomingeniør	Medialogy
Anvendt Filosofi	Technoantropology

In addition, the focus was to implement all the requirement specifications with the *Must* priority. Code snippets will be presented in order to explain certain parts of the implementation, however, some of the examples will be simplified in order to avoid having too large code snippets. Please see Appendix N to see the whole code.

Some libraries were used throughout the development of the prototype, as this made it faster to create a proof of concept. However, some of the libraries might not have been used if the prototype was to be used in production. The libraries used can be seen in table 6.2.

Table 6.2: Table of libraries used for the implementation.

Library	**Purpose**
Compute-cosine-similarity [53]	Calculate cosine similarity between two vectors
Natural [54]	Data processing (tokenization & tf-idf)
Chai [55]	Used for testing
Mocha [56]	Used for testing
command-line-test [57]	Used for testing
pow-mongodb-fixtures [58]	Populate database with data

As mentioned in the Chapter 5, the system should be implemented using the MEAN stack, where external libraries were used, which are not specifically related to the MEAN stack.The libraries are shown in table 6.2.

6.1 User Interface

As presented in section 5.7, the GUI is built based on the F-pattern. This is because a limited amount of UI objects are going to be presented the user. The user would have to look at a bigger area if the Z-pattern were used, since the space between the objects would be far greater than it is now. This could have a negative effect, since the user might get tired of looking around the GUI in order to find the information he is looking for.

The UI of the web application is built with Angular 2 as it is a part of the MEAN framework. In Angular a superset of JavaScript called Typescript is used. The Angular 2 framework allows the UI to be build with components, where each component has an HTML template. Logic is added to components using services. A service is a layer that facilitates communication between the front-end and back-end. Listing 6.1 shows an HTML template used for allowing users to select the location for where they would like to study and to select their interests.

```
1  <div *ngFor="let place of places" class="checkbox">
2      <label><input type="checkbox" value="{{place}}"
       ↪  (change)="onCheckChange($event)">{{place}}</label>
3  </div>
4
5  <h3>Indtast nogle af dine interesser</h3>
6  <p-autoComplete [(ngModel)]="countries" [suggestions]="filteredCountriesMultiple"
7                  (completeMethod)="filterCountryMultiple($event)" styleClass="wid100"
8                  [minLength]="1" [multiple]="true">
9  </p-autoComplete>
10 <button (click)="recommend()" class="btn btn-blue recommend">Recommend</button>
```

Listing 6.1: HTML and Angular template that handles the user inputs - home.component.html.

Line 10 shows the HTML for the button which users can press when they are ready to get a recommendation, and the button calls a function called *recommend()* when clicked. This function is present in the component which this HTML template is tied to, which can be seen in listing 6.2.

```
1  import {RecommenderService} from '../../services/recommender/recommender.service';
2  @Component({selector:'app-home',templateUrl:'./home.component.html',
   ↪  styleUrls:['./home.component.css']})
3  export class HomeComponent{
4    showRecommendations: boolean;
5    recommendations = [];
6    word: any;
7    words: any[];
8    constructor(private recService: RecommenderService) { }
9    recommend(){
10     this.recService.getRecommendation(this.words).subscribe((data)=>{
11       this.recommendations = data;
12       this.showRecommendations = true;
13     });
14   }
15 }
```

Listing 6.2: The Angular component - home.component.ts

The component, seen in listing 6.2, contains the HTML template, which can be seen in line 2. The function called *recommend()* is declared on line 9, and this is where the recommendation service is used. The constructor on line 8 makes use of dependency injection where the

RecommenderService is passed into the component, which means that the service now can be accessed and used in this component. The lines 9-13 make use of the recommender service by calling the *getRecommendation()* function in the recommender service and saving it in an array, as it can be shown in the HTML template. The code for the recommender service can be seen in listing 6.3

```
1   constructor(private http: Http) { }
2   getRecommendation(words){
3     return this.http.post('http://localhost:3000/recommend/edu',words)
4       .map(res => res.json());
5   }
6 }
```

Listing 6.3: Simplified code snippet from the recommender.service.ts file, which shows how the Angular service makes API calls.

The recommender service, seen in listing 6.3, contains the *getRecommendation()* function and line 6 shows how the service makes an API call to the web server.

Listings 6.1, 6.2 and 6.3 give an overview of how the Angular application provides the user with an interface and how Angular can communicate with the routes provided by the web server.

6.2 Web Server

The web server used in development was running locally on each individual developer's machine, since it was easy to start the server with one simple command. As mentioned in Chapter 5, Express was used to handle the routing. In figure 6.1 it can be seen how the routing is done. The browser sends an HTTP post request to the server, then the router situated on the server maps the request to the function that calculates recommendations, which then handles the request with parameters extracted from the request. Express returns the recommended educations to the client's user agent.

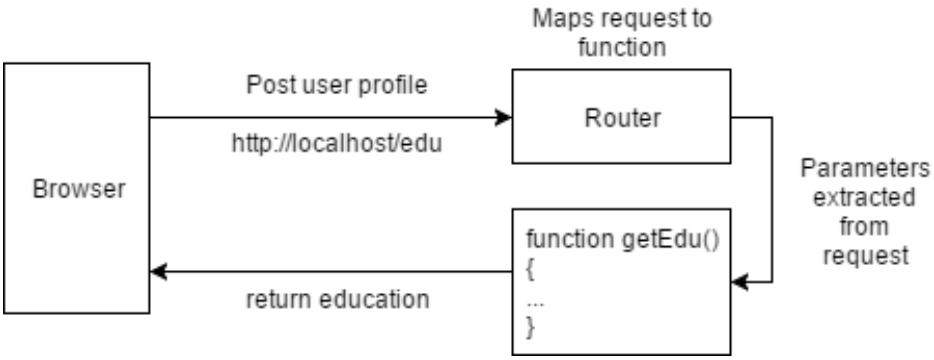

Figure 6.1: Free-form illustration of the system, and how all components communicates

Listing 6.4 shows code from two files, app.js (left) and recommend.js (right). An instance of Express is created in app.js, where the variable app is attached to Express. Routes are then loaded into the app on lines 5-7. The recommend.js file creates a router module, where routes can be defined using for example router.get() or router.post(). Line 5 shows how a route for 'edu' (short for education) was defined. Based on these few lines of code, the application can now handle requests to http://localhost:3000/recommend/edu.

```
1 // In app.js file
2 const express = require('express');
3 const app = express();
4 // Load route
5 var recommend =
6     require('./routes/recommend');
7 app.use('/recommend', recommend);
```

```
1 // In /routes/recommend.js file
2 const express = require('express')
3 const router = express.Router()
4
5 router.post('/edu',function (req, res){
6     ...
7 }
```

Listing 6.4: Code snippets from app.js and /routes/recommend.js

Listing 6.4 shows how the routing is done in the web server. Notice that many routes can be loaded into the application using app.use() (line 7), and in the end the app variable is passed into the http.createServer() function, which creates the server with our Express routes since they are saved in the app variable. This means that the http.createServer() function will run each time someone opens the web application and Express handles all the routing.

6.3 Recommender Engine

Before we can calculate recommendations for a user, we need to create a vector from the user's input, and then compare this vector with the education vectors which are pre-calculated and

already stored in the database. A separate route was created in the API. This route accepts HTTP POST requests containing JSON information in the body such as user's interests and preferred locations. This information is used to create the complete vector for the user, see lines 2-5 in Listing 6.5. Once a vector is created on (line 6), education vectors are fetched from the database (line 8). After the user's vector is compared against all other vectors (lines 9 -13), the vectors with the highest similarity are selected (line 17). Once they are selected, corresponding educations are fetched from the database (line 19), and sent back to the user in JSON format (line 22).

```
1  //CREATING USER's VECTOR
2  db.collection('vectors').find(...).limit(1).map(data => {
3    let wrds = data.vector.map(val => { ... });
4    let places = data.locvect.map(val => { ... });
5    return {'words':wrds,'places':places}
6  }).toArray((error,userVec)=>{
7    //FETCHING EDUCATIONS VECTORS AND COMPARING TO THE USER's VECTOR
8    db.collection('vectors').find({}).map(eduVec => {
9      let wordVectSim = similarity( eduVec.vector.map(word =>
10         {return word.tfidf}),userVec[0]['words']);
11     let placeVectSim = similarity( eduVec.locvect.map(place =>
12         {return place.weight}),userVec[0]['places']);
13     let finalSim = (wordVectSim * 1) * (placeVectSim * 1);
14     return {'eduId' : eduVec.eduId, 'sim' : finalSim }
15   }).toArray((a,eduSimilarity)=>{
16      //FILTERING OUT EDUCATIONS WITH 0 SIMILARITY
17      let closeSimilarity = eduSimilarity.filter(...).map(...);
18      //FETCHING FILTERED EDUCATIONS
19      db.collection('educations').find({'_id':{in:closeSimilarity}},{'words':0})
20         .map(...).toArray((err,data)=>{
21            data.sort(..);
22            res.send(JSON.stringify(data)
```

Listing 6.5: Calculation recommendation snippet

6.4 Database

It was no problem to have the web server run on every developer's machine, since all the code was saved in a Git repository, and it was therefore considered whether the same should be done with a MongodDB, however, this did not seem optimal because every developer needed to be able to work with the data in MongoDB without having to copy and replace

the MongoDB locally. It was therefore decided that mLab should be used. mLab offers hosting for MongoDB on AWS, Azure and Google, it is possible to host a database for free if it is under 500MB in size [59]. This made sure that there was only one database and that every developer had access to the same data.

6.5 Web Scraper

As discussed in 5.2, Aalborg University's website relies on a CMS, which means that structure of the DOM is the same on all of the web pages for the educations. The input URLs were sourced from www.aau.dk/uddannelser/bachelor and www.aau.dk/uddannelser/kandidat. While the rest of the prototype is written in JavaScript, the web scraper is written in Java. The links are stored in a text document, which are read by the scraper.

Before the web pages can be scraped they have to be downloaded to the web scraper's local storage. To do this a library called jsoup was used. jsoup is a a Java library that can parse HTML files to a Java project. In addition it also allows for finding data in web pages using DOM traversal or CSS selectors [60]. Listing 6.6 shows the part of the code that downloads the HTML page to the web scraper. First, the web scraper splits the URL into an array based on the forward slashes in the URI, then it evaluates the length. If the array is split into eight then the web scraper should use the regular education page, otherwise it should add the academic content suffix. This is due to the difference in lengths of URIs in the bachelor's and master's educations web pages.

```
1 String splitLink[] = link.split("/");
2     ...
3 if(splitLink.length == 8) {
4     doc = Jsoup.connect(link).get();
5 } else {
6     doc = Jsoup.connect(link + "/fagligt-indhold").get();
7 }
```

Listing 6.6: Code snippet that is responsible to connect to the HTML page.

In order to get the content of the description box on the website, the scraper has to look for a div element with the itemprop "articleBody" within another div tag with class "spotCon grid g8". Listing 6.7 shows the part of the code that searches the HTML page for the CSS selectors. After the element has been found, the string is cleaned for any HTML tags that may be within such as <img> and <a href>.

```
Element gridG8 = doc.getElementsByClass("spotCon grid g8").first();
Element descriptionJSe = gridG8.select("[itemprop=articleBody]").first();
    ...
descriptionJSe.getElementsByTag("img").remove();
descriptionJSe.getElementsByTag("table").unwrap();
descriptionJSe.getElementsByTag("a").unwrap();

descriptionJSe.text();
```

Listing 6.7: Code snippet that selects the correct div tag to extract data from and cleans it for HTML tags

After the string has been cleaned it is saved to a treemap as a key-value pair. The key is the education name and the value is the extracted data from the itemprop="articleBody". This key-value pair is saved to a treemap, which is then parsed as a json object and written to a file. Listing 6.8 shows the calling of getting name and description methods and save them to a treemap on line 4 to 7. The treemap is then converted to JSON and saved to a file using a filewriter on line 10 to 16.

```
1  Map<String, String> mapBa = new TreeMap<>();
2      ...
3  while((Baurl = aauBALinks.readNextLine()) != null) {
4      String desc = getAAUdesc(Baurl);
5      String name = getAAUname(Baurl);
6
7      mapBa.put(name, desc);
8  }
9
10 try (FileWriter file = new FileWriter(workingDirectory + "/output/aauDescriptionsBA.json")) {
11     String stringMap = gsonBa.toJson(mapBa, mapType);
12     file.write((stringMap));
13     file.flush();
14 } catch (IOException e) {
15     e.printStackTrace();
16 }
```

Listing 6.8: Code snippet that selects the correct div tag to extract data from.

6.6 Data Processing

The data processing is currently implemented as a command line tool, which gives us flexibility and more control over on each operation. There are two goals of the data processing

activities. The first goal is to make sure that data units are formatted properly, so the system can provide a consistent way for presenting the recommendations to the user. The second goal is to generate TF-IDF and vectors which can be used by the recommender engine. Data processing activities are broken down to several steps, where intermediate results of each step are preserved. These steps can be outlined as:

1. Processing files, validating information scraped from the web and saving it in the MongoDB collection.
2. Calculating TF, IDF and then TF-IDF
3. Manually adjusting threshold / filtering out words.
4. Building vectors for each education.

6.6.1 Step 1 - Gathering Information

In this step, files created by the web scraper are parsed by a Node.JS-based script, and inserted into a MongoDB collection. On line 7 in the Appendix G we create a JSON object from the file's content. Prior to inserting the educations in the database, we are adding an additional field to each education called 'lvl'. This is done in line 17 and 20. This field holds the education's level, where a value of 1 is for bachelors, and a value of 2 is for masters. After the educations are saved, we remove educations without description, name, institution name, location, link or level, so our data is consistent.

6.6.2 Step 2 - TF-IDF Processing

After the educations are inserted into the database, the TF-IDF can be calculated. The TF-IDF is calculated in a few steps (as mentioned in section 5.4), and the intermediate results are saved in the MongoDB collection. We implemented a function (Appendix H) that creates an array of objects, where each object contains a word, education name, and a count of how many times the word is used in that education's description. This array is assigned to a variable "loosewords" which is initialized on line 1 in Appendix H. Afterwards, we are using a reduce function on this array on line 1 to produce entries for the intermediate collection. On line 9 in Appendix I we are calculating the normalized TF value. Afterwards we are calculating IDF in Appendix J on line 4. After IDF is calculated, it is simple to calculate TF-IDF. TF-IDF is calculated on line 6 in K.

6.6.3 Step 3 - Word Filtering

The third step involves some manual work, as this is discussed in section 4.2.1.2 in chapter 4. A simple UI was created, where it is possible to reject words, based on their TF-IDF value. It can be done by setting the threshold of the TF-IDF, filtering out words that are below, since these are considered non-relevant. Another function available in the UI allows to go through educations and inspect the words with TF-IDF, and also reject non-relevant words with a simple click of a button. On this step we can also apply the stop words list file with the help of a command line function, which can be seen in Appendix M. The function rejects all the words that are present in the stop list.

6.6.4 Step 4 - Creating Vectors

Excerpt form the vector calculating function can be seen in the Appendix L. After all non-relevant words are filtered out, we can create vectors. We take all "accepted" unique words and use them for a vector from line 5 to 15. Then we loop through the educations and create a vector for each of them, and then saving them in the database on line 74.

6.7 Summary

This chapter started by introducing ten educations that should be used in the prototype in order to provide a proof-of-concept. The UI was built based on the F-pattern, since the prototype would have minimum front-end components. One of the components was a field that would help users to only select words that was predefined based on the education vector. The front-end was developed using Angular 2, which communicated with Express.js running a Node.js server. The Express framework handled the routing, and the recommendation engine was built on a specific Express route. Sending an HTTP post with a user profile in the request body to the route would result in recommended educations being returned in the response. The recommender engine compared the user profile to the education vectors using cosine similarity and returned the educations that align the most with the user profile. MongoDB was used as a database, and mLab was used to provide a MongoDB running in the cloud. The web scraper was developed using Java, which saved all the necessary data into the MongoDB. The data processor was then built, which processed the data gathered by the web scraper and calculated TF, IDF, and TF-IDF. The data processor was also used to filter out words, and thereafter create vectors for each education.

7 Evaluation

In this chapter we will present an evaluation of the prototype that has been built6. The evaluation of the prototype has been divided into two parts, (1) front-end functionality and testing of functional requirements, and (2) unit testing and testing of non-functional requirements.

As the aim of this project is not to test the usability of the system, but rather test if the system answers the problem statement defined in chapter 1 section 1.2, unit testing will be used to test the prototype. Table 7.1 gives an overview of what were tested and the outcome of the tests.

Table 7.1: An overview of the different test, and their result

Test case	**Description**	**Result**
Front-end functionality	The service should be able to take user interests and preferred location as input, and provide the users with a recommended education and information regarding it. See FR1, 2, 3, and 4	Passed
Cosine Similarity	The cosine similarity needs to be tested in order to confirm that it returns a correct result, see NFR1	Passed
TF-IDF	The TF-IDF is tested in order to confirm that it returns the correct result, see NFR2	Passed
Adjustable Weights	The weights should be adjustable, and the test should confirm this, see NFR3	Passed
User Test	The service should provide a recommendation that aligns with the user's preferences	Passed

In the following sections, it will be presented how each of these tests were performed, and the outcome of them.

7.1 Front-end Functionality

The UI for the prototype can be seen in figure 7.1. The UI fulfills FR2 by allowing the user to input the locations they would like to study in the section *Hvor i landet vil du gerne studere?* ("Where in the country would you like to study?"). Checkboxes have been used, as currently there are only three possible locations. FR1 by allowing the user to input their interests into the field *Indtast nogle af dine interesser* ("Input some of your interests"). The possible words to input in this field is based on the selected words from the education descriptions that make up the structure of the education vectors.

Figure 7.1: The final UI which was presented for the users in the small test. This figure is without any recommendation

Figure 7.2 shows that FR3 has been fulfilled, as a recommendation is given to the user, based on the information they have provided for the system. FR4 has also been fulfilled, however, not as it was discussed in section 5.7, but instead it is shown in the recommended education's details. Right below the title of the education, the level and location of the recommended education are shown, this can be seen in figure 7.2. It can also be seen that the interests field also takes several inputs and "stores" it in the interests field. The auto-completer presented in section 5.7 allows this, while suggesting words to the user.

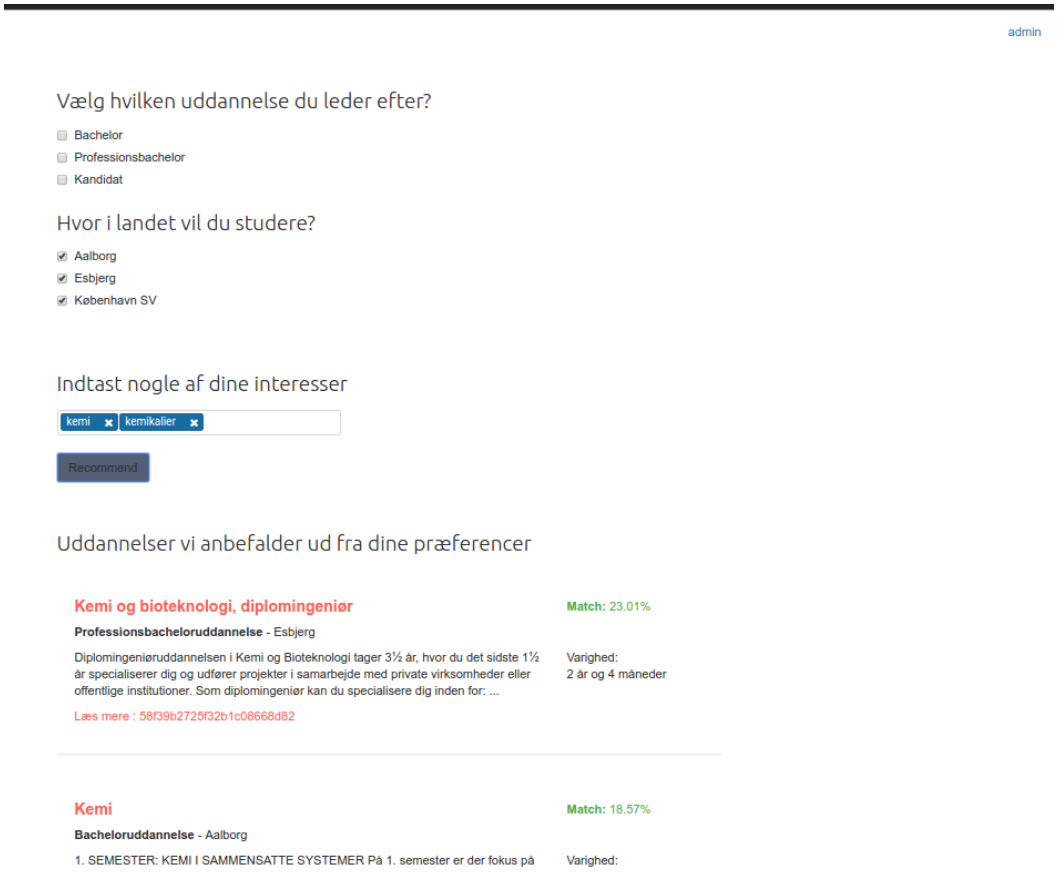

Figure 7.2: The final UI which was presented for the users in the small test. This figure contains a recommendation, based on the user information

7.2 Unit Testing

As previously mentioned unit testing will be utilized in order to make sure that the important parts of the system function as they should. The units that will be tested are those presented in the non-functional requirements list (NFR1, 2, 3, and 4). In order to test the aforementioned non-functional requirements two frameworks have been used. The frameworks are called Mocha [56], and Chai [55]. The Mocha framework provides the necessary tools for writing a tests, while Chai provides helpers for assessment of different conditions. Furthermore, command-line-test [57] was used to test the cm.js script.

7.2.1 NFR1 - Cosine Similarity

The code for testing cosine similarity can be seen in listing 7.1 where three different scenarios are presented in order to test it with different values. The equation tested can be seen in equation 4.5 and the code tested can be seen in listing 6.2.

The input values for scenario one can be seen in line 1-2. The test calculation can be seen in lines 9-10. The test expects that the return value of the similarity function is equal to 1.00, as the two compared vectors are the same. The mathematical equation for this scenario can be seen in equation 7.1 below.

$$\frac{0 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 0}{\sqrt{0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2} * \sqrt{0^2 + 1^2 + 0^2 + 1^2 + 0^2 + 1^2 + 0^2}} = 1.00 \qquad (7.1)$$

For scenario two, the input values can be seen in lines 3-4 and the test calculation can be seen in lines 12-13. Here the expected result is NaN (Not a Number), as the result of both the numerator and denominator are equals to zero the equation will be zero over zero, which equals to NaN. Fortunately, this is not a scenario that ever should happen in the RS, as all words in the user vector points directly to a word in one or more educations.

Finally, lines 5-6 are the input values for scenario three, and lines 15-16 is where the test calculation can be seen. These three tests confirm that the cosine similarity is calculated properly.

```
var userVector1  = [0,1,0,1,0,1,0];
var eduVector1   = [0,1,0,1,0,1,0];
var userVector2  = [0,0,0,0,0,0,0];
var eduVector2   = [1,1,1,1,1,1,1];
var userVector3  = [0.5,0.2,0.1,0.9,0.3,0.4,0.9];
var eduVector3   = [0.1,0.7,0.8,0.1,0.9,0.2,0.7];

describe('Similarity', function(){
  it('It should return 1', function(){
    expect(parseFloat(similarity(userVector1,eduVector1).toFixed(2))).to.be.equal(1.00);
  });
  it('It should return NaN', function(){
    expect(parseFloat(similarity(userVector2,eduVector2).toFixed(2))).to.be.NaN;
  });
  it('It should return 0.58', function(){
    expect(parseFloat(similarity(userVector3,eduVector3).toFixed(2))).to.be.equal(0.58);
  });
});
```

Listing 7.1: Code snippet containing the Chai and Mocha test

Each of the scenario's outcome can be seen in figure 7.3, where each of the scenario's criteria has been fulfilled. This indicates that the code used for calculating the cosine similarity between education's vector and user's vector returns the correct result.

```
Similarity
  ✓ It should return 1
  ✓ It should return NaN
  ✓ It should return 0.58


3 passing (8ms)
```

Figure 7.3: The return message from Mocha that all three tests have passed.

7.2.2 NFR2 - TF-IDF

A command-line-tool (mentioned in table 6.2) was used to execute commands manually, which was located in the cm.js file. The TF-IDF was calculated in the cm.js file and the vector for educations were generated through it as well. The TF-IDF calculation is tightly coupled with the command-line tool, and the database, which made it challenging to do a unit test on the TF-IDF calculation. Because the TF-IDF calculation consisted of several

steps, and every subsequent step relied on the results produced by the previous stage which are saved in the database. This means that the TF-IDF calculation can not be tested without the database. It was possible to re-factor the code and make it more loosely coupled, and thus more testable, but due to the time constraint we decided to create database fixtures which set the test database to a known state prior to running the function, and verify the database state after the function was executed. For every stage of the TF-IDF calculation test two files were created. One that sets the database to an initial state, and one that defines how the database should look like after function ran. Fixtures were created using the 'pow-mongodb-fixtures' library (mentioned in table 6.2) and tests were written using the 'command-line-test' library.

7.2.3 NFR3 - Adjustable Weights

The NFR3 requires that the weights of features should be adjustable. The service allows the weights being manipulated based on two key-value pairs sent with the HTTP post query. The keys are called "nameWeight" and "locWeight", which are the weights for the description and location respectively. The test script can be seen in listing 7.2. Three tests will be conducted, one for the word weight, one for the location weight and one where the weights are equal.

```
1 it('First recommendation name should be "Historie"', (done) => {
2     let input = {"words": ["historie","historiefaget"],"places": ["Aalborg"],
      ↪  "nameWeight":1, "locWeight":1}
3     chai.request(server)
4         .post('/recommend/edu')
5         .set('Content-Type', 'application/json').send(input).end((err, res) => {
6             expect(res.body[0].similarity).to.equal("Historie");
7           done();
8     });
9 });
```

Listing 7.2: Test HTTP post command to test the weights.

With both "nameWeight" and "locWeight" equals to 1 the cosine similarity is equal to 0.36922205253261187. If the "nameWeight" is changed to 10 then the cosine similarity is expected to be $0.36922205253261184 \times 10.0$. The test was conducted three times. First with nameWeight to be 10 and locWeight to be 1. Then nameWeight 1 and locWeight 10 and finally nameWeight 1 and locWeight 1. The results can be seen in figure 7.4. These tests show the weights can be adjusted as stated in requirement NFR3.

```
POST /recommend/edu 200 26941.868 ms - 25116
     ✓ Cosine Similarity should be equals to 3.6922205253261187 * 10.0 * 1.0 (26973ms)
Closing connection
POST /recommend/edu 200 23364.595 ms - 25116
     ✓ Cosine Similarity should be equals to 3.6922205253261187 * 1.0 * 10.0 (23384ms)
Closing connection
POST /recommend/edu 200 22148.575 ms - 25121
     ✓ Cosine Similarity should be equals to 0.36922205253261187 * 1.0 * 1.0 (22165ms)
```

Figure 7.4: Unit test with three different weighting ratios. 10:1, 1:10 and 1:1

7.3 User Test

After the prototype has successfully passed all the requirement specification tests, it is ready for user testing. In the test the participants were asked to input their preferences for an education and see if the the recommendation aligned with their preferences. For the first test (P1) the ten educations shown in table 6.1 were used, however the second test (P2) had all educations included. This means that the test participants have to be currently studying or have previously studied one or more of them. The prototype was tested on two students from Aalborg University. P1 is a 25 year old female bachelor's student of *It, Communication and New Media*, she is currently on her 6^{th} semester. P2 is a 26 year old female master's student of *Innovative Communication Technologies and Entrepreneurship*, she is currently on her 2^{nd} semester.

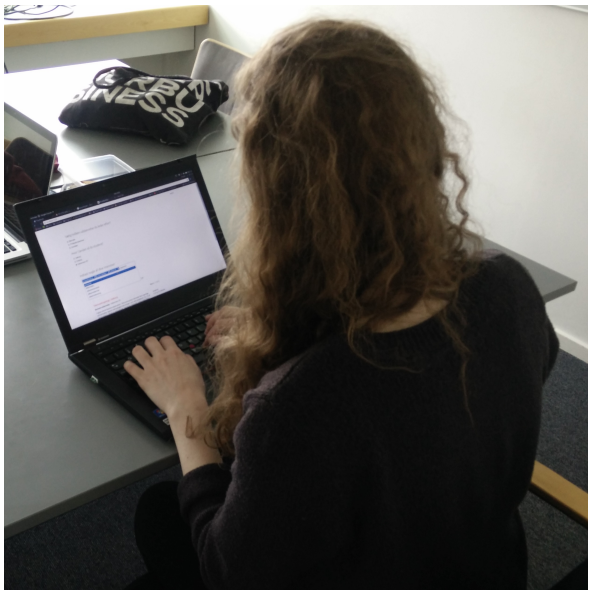

Figure 7.5: Picture from the P1 test.

P1 used the prototype for 7 minutes. The query she inputted can be seen below in table 7.2. This query gave her the recommendation for *It, Communication and New Media* and *Bæredygtigt Design* (Sustainable Design). She felt that these fitted her query and interests well.

Table 7.2: Inputted query from P1

Level	Master's
Location	Copenhagen
Words	"Programmering", "Kommunikation", "Sikkerhed", "Applikationer", "Internet"

P2 used the prototype for 5 minutes. The query she inputted can be seen below in table 7.3. The query gave her recommendations for *International Virksomhedskommunikation i Engelsk* (International Business Communication in English) and *International Virksomhedskommunikation i Tysk* (International Business Communication in German). Further, it suggested *Informatik* (Informatics) and *Informationsteknologi* (Information Technology). She felt that some of the recommendations fit well compared to her query, however, she had no interests in International Business Communication and would much rather prefer Information Technology.

Table 7.3: Inputted query from P2

Level	Bachelor's & Master's
Location	Aalborg & Copenhagen
Words	"Design", "Software", "Kommunikation"

Both of the users' input were rather short queries in terms of interests, which for P2 it could be reason she was suggested business communication over information technology. If she had provided a query with more interests, then business communication might have been filtered out, as it could have reduced the cosine similarity of her vector and the education vectors.

8 Discussion

A discussion of the chosen development method, Lean, and how it was used throughout the project will be presented. Afterwards, the prototype and the different components are discussed.

8.1 Development Methodology

As mentioned in chapter 2 the Lean software development approach was used. Overall the approach was followed, however, it could have been followed more strictly, which could have allowed for a service of a higher quality, both front-end and back-end wise. The first principle about limiting waste helped us to define requirements which can be seen in table 4.6 and 4.7, where the most important requirements (those with a must) are fulfilled. For a minimal viable service like this, it does not need to take input like education form and grade, since it will function without it. In terms of the Lean approach, these would be labeled as extra features. Principle two about taking your discoveries into the software, which can be seen in the form of trying out different things as well as applying what has been written in chapter 3 and 4 to the development of the service, was useful in the development phase. However, this could have been emphasized even more if the service had been tested at an early stage.

As mentioned in chapter 2, Gantt charts were created for writing the report and developing the prototype in order to give an overview of the timeline for the project. The deadlines for the report (seen in Appendix B.1) were not always followed, and they should have been broken down into smaller tasks, as this proved to work well when developing the prototype. Throughout the development of the prototype a project management tool was used called Trello. In Appendix B figure B.3, a picture of the tasks used in Trello can be seen. The tasks were divided into small tasks where each person could assign his name to a task and move it to the "doing" task-list. When a task was completed it was moved to the "done" task-list. The small tasks made the transition from having small pieces of software to connecting it all into a full piece of software smooth.

The Lean development method helped this project by pushing it forward with defined tasks and using the theory that was researched in chapter 3.

8.2 Features

The main feature was the education descriptions where we collected words that represented the education, which in the end would be used to fill out the majority of the user vector. Our choice of using description for the content-based part and the location for the constraint-based might have had an affect on the recommendations. A problem of using the descriptions was that some of them did not properly describe the education. The other feature that was used in the service was location. Location was an important feature to include into the system, as the constraint-based RS was based on it. Other features such as language, grade, education form and level could also have been used. These features could have helped the service to provide a more reliable recommendation, as more information would be provided to the service. However, as it can be seen in table 4.6, these have a MoSCoW prioritization of "should", meaning that a minimal functioning service would still work without them.

The weights for the two features were defined by ourselves, however, this way is not suitable, as each user is different and has different preferences meaning that they would probably also weigh the two features differently. Instead of having "predefined" weights, it would be more beneficial and optimal if the users could do the weighing themselves. A way this could be done would be to provide the users with a slider so they would be able to control the weighing themselves. This would give a much more precise recommendation for each user.

8.3 Low Matching Percentage

One of the obstructions encountered was that the value calculated from the cosine similarity was quite low. This was due to the elements in the education vectors included very small numbers as the TF-IDF values were stored in the vectors. This number was then being compared to the user vectors that only contained either 0 or 1 in each element. Thus the difference between the elements were very high returning a small cosine similarity. However, this did not have an affect on the recommendations as all educations are compared to the same number in the user vector (1 or 0). As the users did not rate the importance of each word, the ideal weight in the user vectors are 1 for each word, as this means the words are a part of the most important ones. Setting the number lower might skew the results as if a TF-IDF weight was above the weight in the user's vector then it would reduce the similarity, despite not being less similar.

8.4 Categorization And Grouping

In this project's implementation, categorization was not used as the objective of the report was to answer the problem statement in section 1.2. If categorization had been used, we could have categorized words together with other words, which are in the same field. An example of words that could have been in the same category could be "JavaScript", "C#", "Python", "programming", where the phrase characterizing the category could be "software development". However, if the categorization was done, the user should be presented with a possibility to weigh the words within that category, otherwise the user would experience the same problem as mentioned in scenario 1, in chapter 4, section 4.1.

Another problem that occurred when the prototype was evaluated was that there were a lot of words which had the same meaning, but did not look alike, e.g. different prefix or suffix. As discussed in section 4.2.1.2, stemming could have been used if we had education descriptions in English. However, since the descriptions were in Danish, this was not a possibility. A way to solve this problem could have been to create a structure, where similar words with different endings had one word which then was presented through the UI. This would help making the education vectors smaller, since instead of having seven words that mean the same, but with a different ending, they would only be presented with one word.

8.5 Meaning Of Words

The TF-IDF value was calculated for each word in every education description, however, this approach does not take the meaning of words into account. This means that the system for example would make "big data" into two separate words, and they would receive separate TF-IDF values. This is a problem since the meaning of "big data" compared to "big" and "data" (separated) is not the same. Instead external services could have been used to enable natural language understanding. For example, IBM Bluemix allows developers to access their Natural Language Understanding API as a RESTful service [61]. As opposed to TF-IDF, Natural Language Understanding takes phrases and the context that words are used in into account and weighs them accordingly. The three words with the highest TF-IDF in the bachelor's education Medialogy are "Interactive", "Sleep" and "Guitarists". Whereas using IBM Bluemix the phrases with the highest weights are "Sound Effects", "Interactive Systems Design" and "Physical Interface Design". IBM Bluemix was not used for this project as they currently only support a small selection of languages such as English and Spanish.

8.6 Poor Education Descriptions

When the education descriptions were scraped from the Internet, it became apparent that some of the descriptions were poor with almost no or little description. An example of two educations which have poor descriptions can be seen in Appendix F. For the service, this could have an impact since the the amount of words are low and not specific enough to say what that education is about. A way to overcome this problem could be to collect information about the education from several different websites, and then choose the description which says the most about the education. It could also be possible to make use of the descriptions from UG.dk as described in 3.2.2.1, as these are generally shorter while still containing rich information. Another way of overcoming the problem, would be to use Domain experts.

8.7 Domain Experts

Using domain experts could have been beneficial for optimizing the service. The domain experts could have helped us by defining words that would describe each education. However, before the domain expert should be used, some manual labor would still need to be done. Examples of the manual labor can be filtering out stop words, performing stemming, or similar tasks that reduces the amount of words which are presented for the domain expert to make sure they do not have to go through every single word sourced from the descriptions.

Examples of domain experts could have been the teachers who are teaching each education, and student who have graduated from a specific education. Using a mix of those two could have proven to be beneficial since each of them might provide different words, since they have different views of what the education is about.

As it can be seen below in table 8.1, a domain expert of an education, Innovative Communication Technologies and Entrepreneurship, provided a set of words which indicates what the education is about. The domain expert chosen was Henning Olesen, the one responsible for the education. It is clear to see that there is a difference between the domain expert's words and the ones in our system. This adds further support to the importance of using domain experts in RSs such as this one, as the difference between the TF-IDF words and the domain expert phrases are significant.

Table 8.1: Descriptive words of Innovative Communication Technologies and Entrepreneurship.

TF-IDF	**Domain Expert**
ICT	Business Development
Services	Content Networking
Business	Network Technologies
Governance	Software Engineering
Content	Identity Management
Application	Service Development
Media	Interaction Design
Security	Data Analytics
Design	ICT Regulation
Management	Techno Economics

8.8 User Preferences

The user profile was constructed by asking the users explicit questions about their preferences. The user test found that users are not likely to input more than a couple interests before asking for a recommendation. As discussed in section 8.2, it is desirable to include more features into the system. However, as the users did not even input comprehensive information in the current system, they might input even less information if more features requires additional information. The user profile could be enriched easily by sourcing user information from other websites. For example their YouTube video history, Twitter followings and tweets, Facebook likes and posts. This enrichment could provide the user profile with predefined interests that would help fill out the user vector, giving the service more information of the user to base a recommendation on. It could also introduce a more convenient way for the user to choose their interests, since they would not have to fill out the interests field manually. However, this could also prove to be problematic, because the user would maybe also have to remove some of the interests that were based on fetching information from e.g. Facebook, since they might not align with the user's current interests. If the user did not remove those interests, they would get a recommendation which might be skewed due to incorrect interests.

8.9 Privacy And Security

As mentioned in chapter 3 section 3.4.2, SSL would be essential in a system running in production, since the data that the users input has to be pseudonymized according to the GDPR. This was not necessary for a small prototype as it is not public available, but it would be a requirement if it was developed into a full system. If any user data would be stored in the system, it would need to be pseudonymized as well, which would mean that the data would not directly be linkable to an identity.

The prototype had the functionality where the administrator could filter words, which was seen in the use-case diagram (figure 4.2). However, there was no login and logout function implemented. This means that anyone running the prototype locally could go to localhost:4200/admin/edu and start filter out words without needing to login. It would therefore be needed to limit access to that route so that only administrators with valid credentials would be able to login. Using proper cryptographic techniques would then be needed for authenticating and authorizing the administrator [62].

8.10 Design And Implementation

The MEAN stack was used to create a proof-of-concept of this project, however, the computational requirements in a large application could mean that the MEAN stack is not a good choice, since Node.js is particularly good at handling many request, but does not perform as well when it comes to CPU intensive tasks. If the architecture of our prototype was used for a large application, then it would result in having large vectors for users and educations, which could require a lot computational power if many users were using the application.

All the data processing operations mentioned in chapter 6 were performed through one script called cm.js, which can be found in Appendix N. This meant that the operations were run manually through that script (e.g. TF, IDF, cosine similarity) when processing the data. This worked for the prototype, however, it would not be ideal in production. Therefore, it would be more appropriate to automate those operations. Reducing the amount of code could also optimize the system, since there was some bloat code within the prototype. The operation for calculating TF-IDF in the cm.js file could for example been simplified and broken down to two functions; one for TF and one for IDF. This strategy would allow to write more simple unit tests as well, so the code maintainability could also be improved.

As mentioned in Chapter 6 section 6.4, the database was deployed using a free solution from mLab, however the performance was not optimal. The performance was unstable, where

it sometimes could take 30 seconds from when the "recommend" button was pressed until the recommendation was shown in the application. A simple test was done by copying the database to a local MongoDB instance and run the application, and the response time was always under 1 second. A faster solution should be investigated before full deployment. Further, a non-functional requirement on the performance speed should be defined.

9 Conclusion

This aim of this project was to develop a solution where users can find educations based on their personal preferences. In order to achieve the aforementioned, research within the field of recommender systems was done and the findings were analyzed and applied to this project. A research question was defined in order to specify what this project's outcome and contribution should be. The research question was as follows:

How can a recommender system based application support people to find an education that aligns with their preferences?

Five sub-questions were defined to answer the problem statement. Each of the sub-questions opened up for new important aspects of the problem statement that should be researched.

Different recommender system techniques were investigated, where a hybrid of content- and constraint-based was chosen. This combination provided accurate recommendation results. However, some of the educational information used for the content-based was not precise and rich enough. The use of domain experts could have helped solve this problem.

As many recommender systems require initial input data to compare the users against, it was decided to research what education information should be used. In order to support the content-based and constraint-based RSs, education description and location were used. As discussed earlier, using this information created some problems that could hinder a reliable recommendation of an education. A higher quality recommendations could have been achieved by using more educational information. The educational information was obtained through a scraper that collected information from the education institution's website. It occurred that the collected information required processing before it could be used in the RS. Another way of acquiring information could prove to be beneficial for obtaining the needed information.

Meaningful data about the users had to be gathered to compare with the education vectors. The user's preferences were collected using an HTML form on the service's front end. While this was suitable for the prototype, a better way of providing user preferences to the system could be beneficial for a fully developed service. User preferences could for example be sourced from the user's Facebook account. The user preferences were used to recommend educations for each user by comparing it with the different education vectors. Security

and privacy was also researched as there is an increased focus on keeping user data secure and private. The prototype did not store any information about the user, however, user preferences were sent through HTTP requests. The General Data Protection Regulations was therefore researched since user preferences can be considered personal information. Using the Secure Socket Layer protocol was therefore suggested in order to help protect personal information from being intercepted by third parties.

The service was first designed based on the requirement specifications derived from our analysis, and thereafter implemented using the MEAN stack. In addition to providing a full stack solution, which only required knowledge of a single programming language in order to build powerful web applications, the components work very well together and do not require extensive knowledge to get started on.

Based on this it can be concluded that a recommender system can be used to help people to find the right education for them. More specifically, a content-based recommender system can be used. It was found that the most important aspect is the input data for both the educations and the users. Some of the input data used for the educations in this project did not provide sufficient descriptions of the educations and did not have many unique words. This made it difficult to recommend these educations, as they had a very small unique vector.

The input data provided by the users should also be treated differently. In order to provide a meaningful recommendation, the users had to input a lot of words. Using categories could have prevented this problem. It would be useful to research into alternative and automated ways that does not require the user to input a lot of data in order to generate the user vector.

10 Future Works

In this chapter several aspects, which could be implemented into the project if it was further developed, are presented. These aspects are merely some of the areas that could improve the quality of the service.

For improving and adding more functionality to the service, machine learning algorithms, such as k-means clustering would be beneficial to use. K-means clustering would add another layer on-top of the existing functionality by introducing another way of recommending education to the user. Demographic information from the user could be used to give them recommendations on what similar users have received as a recommendation. The user ratings of the recommendation could also be used to make the system smarter. If a recommendation has only received low ratings based on similar user profiles, the system could use it to learn that the education should not be recommended if a similar user profile is presented.

Feedback from the users would be useful in order to improve the system. Data such as how much the recommendation they received suited them could be vital for improving the system's recommendation engine vastly. This could also be used to create predefined profiles, which could be accessible for the users, so they would not have to start completely from scratch when using the service. If the user gave the recommendation a high rating it could be used to showcase it to other users who might have similar interests, since they might like the same education.

The prototype did not include a login for the users, but this could be necessary if more features were added to the system, which was suggested in Chapter 8 section 8.2. Including more features would mean that the users would need to provide more information about their preferences, and the users therefore have to answer more questions. Having a login could allow the users to save the preferences, which could be relevant if users needed to leave the application without providing all their preferences.

As mentioned in Chapter 8 section 8.10, some of the code was not well optimized and required running scripts manually. A fully developed system should be scalable while avoiding the need for running scripts manually. Only educations from AAU.dk were used in the prototype, but the system should be scalable enough to be able to add other Universities easily.

Bibliography

[1] Tim O'Reilly. What is Web 2.0, 2009. URL http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1. Accessed: 12-04-2017.

[2] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender systems handbook*. Springer, 1 edition, 2015.

[3] Ministry for Higher Education and Science. Tal og fakta om søgning og optag på de videregående uddannelser, 2017. URL http://ufm.dk/uddannelse-og-institutioner/statistik-og-analyser/sogning-og-optag-pa-videregaende-uddannelser.

[4] Pernille Mainz and Jacob Fuglsang. Eksperter: Der er alt for mange uddannelser, 2015. URL http://politiken.dk/indland/uddannelse/art5583902/Eksperter-Der-er-alt-for-mange-uddannelser.

[5] Salvador Garcia-Martinez and Abdelwahab Hamou-Lhadj. *Educational Recommender Systems: A Pedagogical-Focused Perspective*, pages 113–124. Springer International Publishing, Heidelberg, 2013. ISBN 978-3-319-00375-7. doi: 10.1007/978-3-319-00375-7_8. URL http://dx.doi.org/10.1007/978-3-319-00375-7_8.

[6] Mei-Hua Hsu. A personalized english learning recommender system for esl students. *Expert Systems with Applications*, 34(1):683–688, 2008.

[7] Nguyen Thai-Nghe, Lucas Drumond, Artus Krohn-Grimberghe, and Lars Schmidt-Thieme. Recommender system for predicting student performance. *Procedia Computer Science*, 1(2):2811 – 2819, 2010. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2010.08.006. URL http://www.sciencedirect.com/science/article/pii/S1877050910003194.

[8] Git. About version control, No date. URL https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control. Accessed: 19-05-2017.

[9] Paul Beynon-Davies, Chris Carne, Hugh Mackay, and Douglas Tudhope. Rapid application development (rad): an empirical review. *European Journal of Information Systems*, 8(3):211–223, 1999.

[10] Mary Poppendieck. Lean software development. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 165–166. IEEE Computer Society, 2007.

[11] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002. ISSN 0924-1868. doi: 10.1023/A:1021240730564. URL http://dx.doi.org/10.1023/A:1021240730564.

[12] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: Introduction and challenges. In *Recommender Systems Handbook*, pages 1–34. Springer, 2011.

[13] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, pages 3:1–3:10, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-075-3. doi: 10.1145/1409540.1409544. URL http://doi.acm.org/10.1145/1409540.1409544.

[14] Marko Balabanović and Yoav Shoham. Fab: Content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, March 1997. ISSN 0001-0782. doi: 10.1145/245108.245124. URL http://doi.acm.org.zorac.aub.aau.dk/10.1145/245108.245124.

[15] Robin Burke. Hybrid web recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web: Methods and Strategies of Web Personalization*, pages 377–408, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-72079-9. doi: 10.1007/978-3-540-72079-9_12. URL http://dx.doi.org/10.1007/978-3-540-72079-9_12.

[16] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[17] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. Effective attack models for shilling item-based collaborative filtering systems. In *Proceedings of*

the 2005 WebKDD Workshop, held in conjuction with ACM SIGKDD, volume 2005, 2005.

[18] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_3. URL http://dx.doi.org/10.1007/978-0-387-85820-3_3.

[19] Marie-Francine Moens, Juanzi Li, and Tat-Seng Chua. *Mining User Generated Content*. Chapman & Hall/CRC, 2014. ISBN 1466557400, 9781466557406.

[20] C.C. Aggarwal. *Recommender Systems: The Textbook*. Springer International Publishing, 2016. ISBN 9783319296579. URL https://books.google.dk/books?id=Wpf3jgEACAAJ.

[21] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. *Developing Constraint-based Recommenders*, pages 187–215. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_6. URL http://dx.doi.org/10.1007/978-0-387-85820-3_6.

[22] Fabiana Lorenzi and Francesco Ricci. *Case-Based Recommender Systems: A Unifying View*, pages 89–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31655-8. doi: 10.1007/11577935_5. URL http://dx.doi.org/10.1007/11577935_5.

[23] Cleidson RB de Souza, David Redmiles, Li-Te Cheng, David Millen, and John Patterson. Sometimes you need to see through walls: a field study of application programming interfaces. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 63–71. ACM, 2004.

[24] Tim Brady. Rfc 7159 - the javascript object notation (json) data interchange format, 2014. URL https://tools.ietf.org/html/rfc7159.

[25] Eloisa Vargiu and Mirko Urru. Exploiting web scraping in a collaborative filtering-based approach to web advertising. *Artificial Intelligence Research*, 2(1):44, 2012.

[26] UddannelsesGuiden. About uddannelsesguiden, no date. URL https://www.ug.dk/programmes/aboutugdk. Accessed: 22-04-2017.

[27] Simon Kjølby Larsen. Bekendtgørelse om indberetning til uddannelsesguiden (ug.dk), 2009. URL https://www.retsinformation.dk/forms/r0710.aspx?id=125466. Accessed: 24-04-2017.

[28] David Vestergaard Eriksen. http://ufm.dk/uddannelse-og-institutioner/videregaende-uddannelse/sogning-optag-og-vejledning/uddannelseszoom, 2017. URL http://ufm.dk/uddannelse-og-institutioner/statistik-og-analyser/uddannelseszoom. Accessed: 27-04-2017.

[29] David Vestergaard Eriksen. http://ufm.dk/uddannelse-og-institutioner/statistik-og-analyser/uddannelseszoom, 2017. URL http://ufm.dk/uddannelse-og-institutioner/statistik-og-analyser/uddannelseszoom. Accessed: 27-04-2017.

[30] Ministry for Higher Education and Science. Den koordinerede tilmelding, 2014. URL http://ufm.dk/uddannelse-og-institutioner/videregaende-uddannelse/sogning-optag-og-vejledning/kot. Accessed: 22-04-2017.

[31] Lotte Vinkel Hansen. Hovedtal - den koordinerede tilmelding (kot) — uddannelses- og forskningsministeriet, 2017. URL http://ufm.dk/uddannelse-og-institutioner/statistik-og-analyser/sogning-og-optag-pa-videregaende-uddannelser/grundtal-om-sogning-og-optag/kot-hovedtal. Accessed: 22-04-2017.

[32] UddannelsesGuiden. Studievælgeren, no date. URL https://www.ug.dk/vaerktoej/studievaelgeren/. Accessed: 27-04-2017.

[33] UddannelsesGuiden. Adgangskortet, no date. URL https://www.ug.dk/vaerktoej/adgangskortet/. Accessed: 27-04-2017.

[34] Erika McCallister, Timothy Grance, and Karen A. Scarfone. Sp 800-122. guide to protecting the confidentiality of personally identifiable information (pii). Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2010.

[35] EU GDPR. Key changes with the general data protection regulation, No date. URL http://www.eugdpr.org/key-changes.html. Accessed: 20-05-2017.

[36] Jeroen van Rest, Daniel Boonstra, Maarten Everts, Martin van Rijn, and Ron van Paassen. Designing privacy-by-design. In *Annual Privacy Forum*, pages 55–72. Springer, 2012.

[37] Sumit Gaur. Man in the middle attack using bt5 ettercap tutorial, 2013. URL http://www.justhackitnow.com/2013/11/man-in-middle-attack-using-bt5-ettercap.html. Accessed: 20-05-2017.

[38] Tutorials Point. Network security transport layer, No date. URL https://www.tutorialspoint.com/network_security/network_security_transport_layer.htm. Accessed: 20-05-2017.

[39] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006. ISBN 0132390779.

[40] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, RFC Editor, April 2006. URL https://tools.ietf.org/html/rfc4346#section-1. Accessed: 21-04-2017.

[41] J. Du, X. Li, and H. Huang. A study of man-in-the-middle attack based on ssl certificate interaction. In *2011 First International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pages 445–448, Oct 2011. doi: 10.1109/IMCCC.2011.117.

[42] Inc. OpenSSL Foundation. Openssl, 2017. URL https://www.openssl.org/. Accessed: 13-04-2017.

[43] GlobalSign. What are the different types of ssl certificates?, No date. URL https://www.globalsign.com/en/ssl-information-center/types-of-ssl-certificate/. Accessed: 20-05-2017.

[44] Bonnie A Nardi. *The use of scenarios in design*. Hewlett-Packard Laboratories, Technical Publications Department, 1992.

[45] Chris D Paice. An evaluation method for stemming algorithms. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–50. Springer-Verlag New York, Inc., 1994.

[46] Dave Ensor and Ian Stevenson. *Oracle Design: The Definitive Guide*. " O'Reilly Media, Inc.", 1997.

[47] miguelmalvarez. How to select stop words using tf-idf? (non english corpus), No date. URL https://stackoverflow.com/questions/16927494/how-to-select-stop-words-using-tf-idf-non-english-corpus. Accessed: 20-05-2017.

[48] William Zola. 6 rules of thumb for mongodb schema design: Part 1, 2014. URL https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1. Accessed: 13-04-2017.

[49] William Zola. 6 rules of thumb for mongodb schema design: Part 2, 2014. URL https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-2. Accessed: 13-04-2017.

[50] William Zola. 6 rules of thumb for mongodb schema design: Part 3, 2014. URL https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-3. Accessed: 13-04-2017.

[51] Brandon Jones. Understanding the z-layout in web design, No date. URL https://webdesign.tutsplus.com/articles/understanding-the-f-layout-in-web-design--webdesign-687. Accessed: 27-04-2017.

[52] Brandon Jones. Understanding the z-layout in web design, No date. URL https://webdesign.tutsplus.com/articles/understanding-the-z-layout-in-web-design--webdesign-28. Accessed: 27-04-2017.

[53] Compute.io. Computes the cosine similarity between two arrays., No date. URL https://github.com/compute-io/cosine-similarity. Accessed: 16-05-2017.

[54] Natural. General natural language facilities for node, No date. URL https://github.com/NaturalNode/natural. Accessed: 16-05-2017.

[55] Chai, No date. URL http://chaijs.com/. Accessed: 19-05-2017.

[56] Mocha, No date. URL https://mochajs.org/. Accessed: 19-05-2017.

[57] Zhiye Li. Command-line-test, No date. URL https://github.com/xudafeng/command-line-test. Accessed: 20-05-2017.

[58] Charles Davison. pow-mongodb-fixtures, No date. URL https://github.com/powmedia/pow-mongodb-fixtures. Accessed: 20-05-2017.

[59] mLab. Mongodb hosting: Database-as-a-service by mlab, No date. URL https://mlab.com/. Accessed: 15-05-2017.

[60] Jonathan Hedley. jsoup: Java html parser, 2016. URL https://jsoup.org/. Accessed: 16-05-2017.

[61] IBM. Natural language understanding, No date. URL https://www.ibm.com/watson/developercloud/natural-language-understanding.html. Accessed: 19-05-2017.

[62] Stephen A. Thomas. *SSL and TLS Essentials: Securing the Web*. John Wiley & Sons, Inc., New York, NY, USA, 2000. ISBN 0471383546.

[63] Christos Svitis. Lean software development, 2015. URL https://gupea.ub.gu.se/bitstream/2077/38521/1/gupea_2077_38521_1.pdf. Accessed: 20-05-2017.

Appendix A

Lean Development Method

A.0.1 Lean Software Development

For a software development method, the Lean method will be used. The Lean development method focuses on delivering value, in the form of limiting the waste, and lowering the cost, which are useful when developing software. The Lean development process boils down to seven principles, which are [63, 10]:

1. Eliminating waste - Removing objects and elements which does not need to be there in order to add value for the users. Examples of waste are extra features, extra processing steps, and requirements.
2. Amplifying Learning - Incorporating your discoveries from e.g. a test into the software. Here the phrase "Examine and Adjust" is the main focus.
3. Decide as Late as Possible - in order to avoid unnecessary testing, changes, features etc., one should take decisions as late as possible. A usage of short loops in the initial planning phase can emphasize the aforementioned.
4. Deliver as Fast as Possible - Smaller pieces of software are easier to handle that one large pieces of software, hence, delivering small packages of software is desired. Using this allows for a much smoother transition between the iterations, if a important item is removed from the objective queue.
5. Empower the Team - An example of this is allowing the development team to take part in the technical decision making, since they have knowledge about the problem, and have an understanding of how it can be solved.
6. Build Integrity In - There is internal and external integrity. Internal means that all the different parts in the system should work together. External is the consistency between user demands and the systems performance. Integrity can be added to the product, by using user feedback.

7. Optimize the Whole - The whole process should be optimized right from the get go, meaning, that when you receive a request from the customer, the whole stream should be optimized.

With these seven principles software development cost and quality should increase [63, 10]. However, as stated in [63], a good approach to using Lean development is to customize it to benefit your project, instead of picking it up "as-is". With this in mind, as the length of this project is only 5 months, we will not be able to fulfill all of the principles of the Lean development method.

Appendix B

Timeline

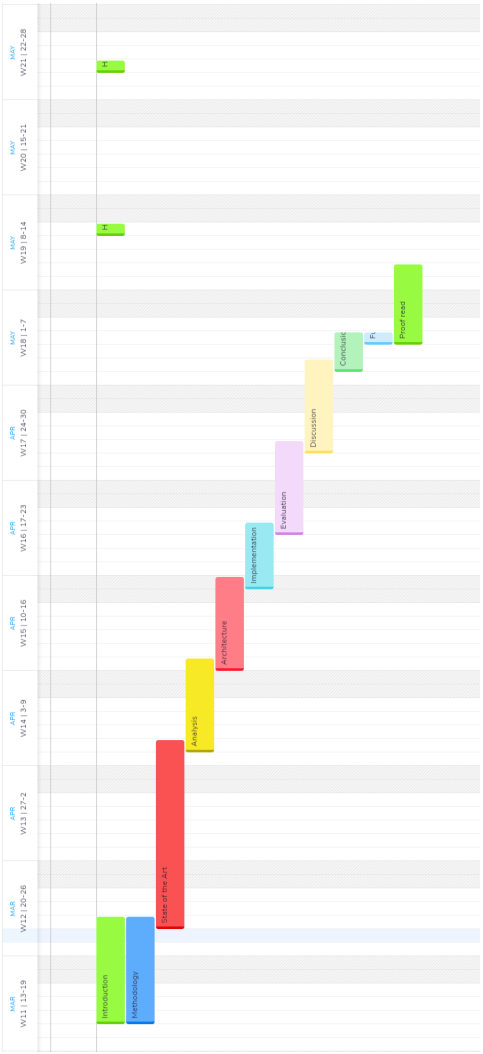

Figure B.1: Report Timeline

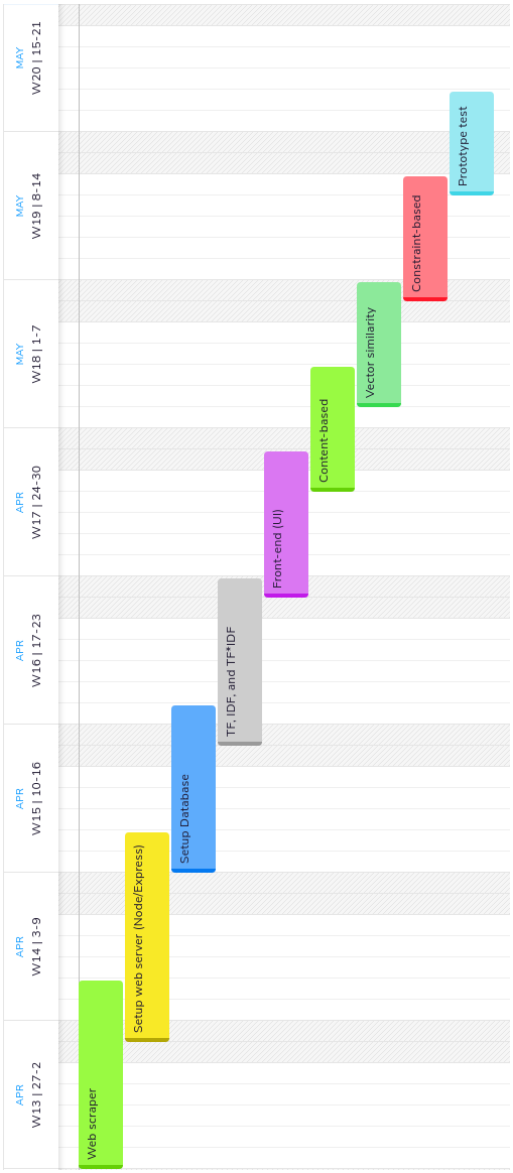

Figure B.2: Prototype Timeline

Figure B.3: Picture of tasks in Trello

Appendix C

Whiteboard

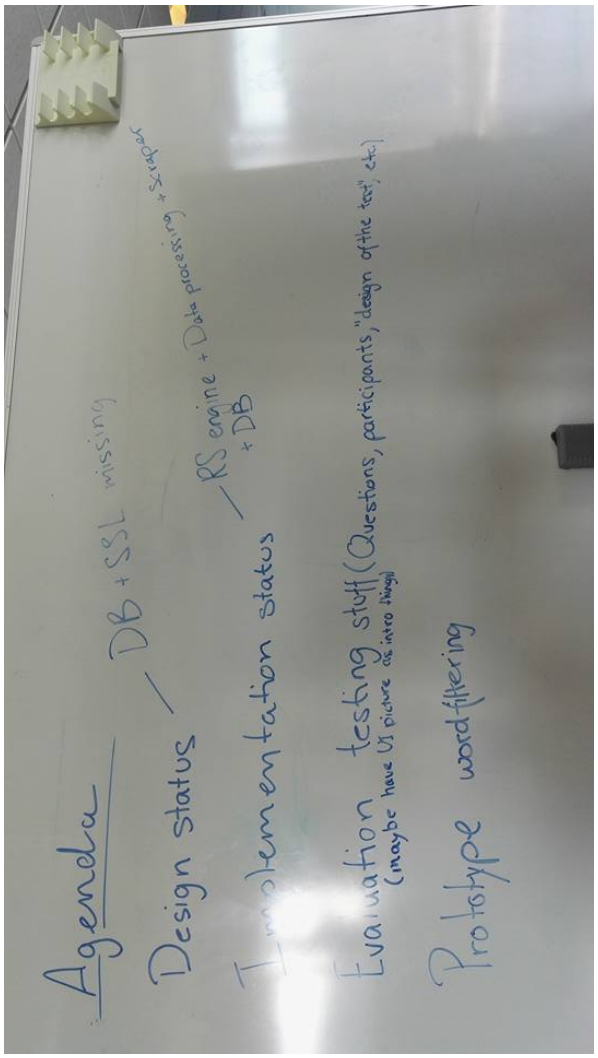

Figure C.1: Picture of a meeting agenda.

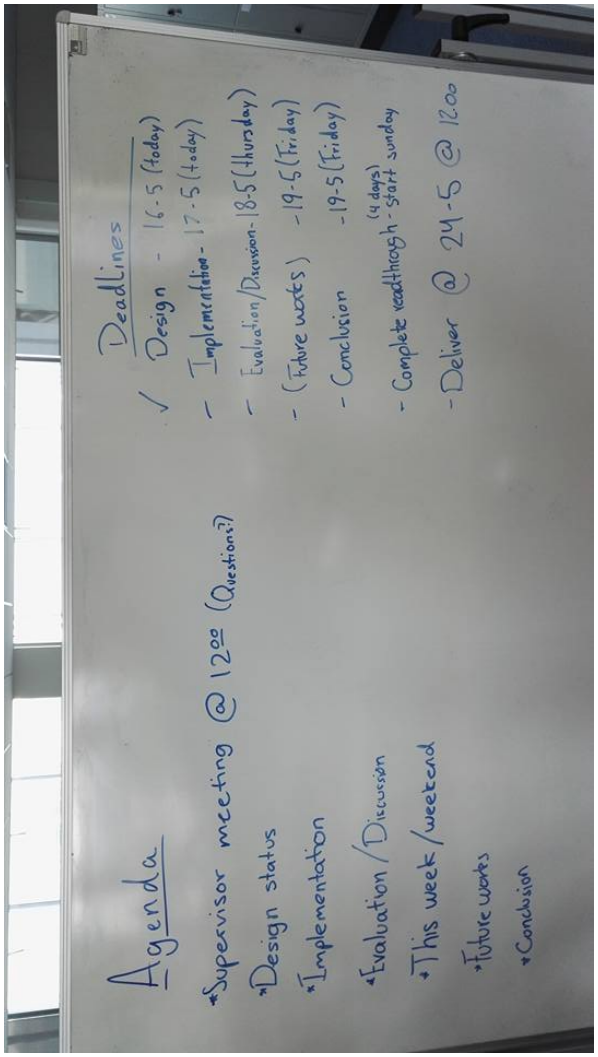

Figure C.2: Picture of a meeting agenda and deadlines.

Appendix D

Stopwords List

Please see https://github.com/nicfol/Edu-RS/blob/master/resources/Stopwords.txt for a list of the stopwords used in the preprocessing.

Appendix E

Admin Sequence Log-in diagram

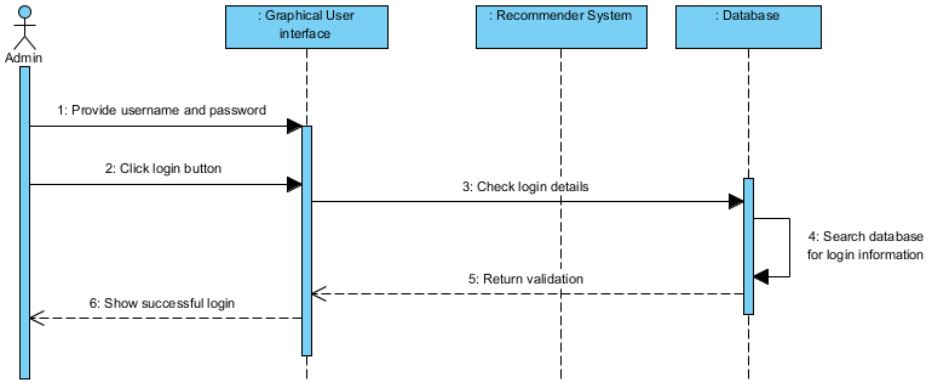

Figure E.1: Sequence Diagram of the admin log-in

Appendix F

Poor Education Descriptions

Figure F.1: idræt

Figure F.2: Architecture

Appendix G

Saving Educations Code

```
fs.readFile(path.join(__dirname, filename), 'utf8', function (err, data) {
    if (err) {
        console.log(err);
        process.exit(1);
    }
    content = util.format(data);
    let baseobj = JSON.parse(content);

    if (subdok) {
        var objects = baseobj[subdok];
    } else {
        var objects = baseobj;
    }
    function* values(obj) {
        for (let prop of Object.keys(obj)) {
            if (coll === 'bachelors') {
                obj[prop]['lvl'] = 1
            }
            if (coll === 'masters') {
                obj[prop]['lvl'] = 2
            }
            yield obj[prop];
        }
    }

    let arr = Array.from(values(objects));
    collection.insertMany(arr, function (err, result) {
        assert.equal(err, null);
        callback(result);
    });
});
```

Listing G.1: Parsing files and inserting them into MongoDB.

Appendix H

Counting Words Occurrences In The Descriptions

```
let loosewords = [];

for (let doc of docs) {
    let matchedWords = tokenizer.tokenize(doc.description.toLowerCase());
    wordsInCorpus += matchedWords.length;

    wordsPerDocument.push({
        docname: doc.name,
        wordsindoc: matchedWords.length,
        docid: doc._id,
        lvl: doc.lvl
    });
    let counts = matchedWords.reduce(function (stats, word) {
            let element = stats.findIndex((elem) => {
                return elem.hasOwnProperty("word") && elem.word === word
            });
            if (element !== -1) {
                stats[element].count += 1
            } else {
                stats.push({word: word, count: 1, doc: doc.name, docid: doc._id});
            }
            return stats;
        }, []
    );
    loosewords = loosewords.concat(counts);
}
```

Listing H.1: Counting words occurances.

Appendix I

Generating Database Entries

```
let entries = loosewords.reduce(function (words, word) {
    let w = wordsPerDocument.find(e => {
        return e.docid.equals(word.docid)
    });

    let temp = {
        doc: word.doc,
        count: word.count,
        tfN: word.count / w.wordsindoc,
        wordsindoc: w.wordsindoc,
        docid: w.docid,
        lvl: w.lvl
    };
    let element = words.findIndex((elem) => {
        return elem.hasOwnProperty("word") && elem.word === word.word
    });
    if (element !== -1) {
        words[element].count += word.count;
        words[element].indoc.push(temp)
    } else {
        let dword = {
            word: word.word,
            count: word.count,
            status: 0,
            indoc: [temp],
            wordsInCorpus: wordsInCorpus,
            totalDocs: totalDocs
        };
        words.push(dword);
    }
    return words;
}, []);

let totalcounts = db.collection('words');
totalcounts.drop((err, reply) => {
```

```
36      totalcounts.insertMany(entries, function (err, r) {
37          callback();
38      })
39  })
```

Listing I.1: Counting words occurances.

Appendix J

Calculating IDF Code

```
db.collection("words").find({}).toArray((err, data) => {

    for (let doc of data) {
        let idf = 1 + Math.log(doc.totalDocs / (doc.indoc).length);
        let objectId = new ObjectID(doc._id);
        db.collection("words").updateOne({_id: objectId}, {$set: {idf: idf}}, {upsert:
        ↪  true}, (er, r) => {

        });
    }
    callback();
});
```

Listing J.1: Calculating IDF.

Appendix K

Calculating TFIDF Code

```
for (let word of words) {
    for (let doc of word.indoc) {

        let tmpwrd = {
            word: word.word,
            tfidf: word.idf * doc.tfN,
            status: word.status
        };

        let element = alldoc.findIndex((elem) => {
            return elem.hasOwnProperty("docid") && elem.docid.equals(doc.docid)
        });

        if (element === -1) {
            alldoc.push({docid: doc.docid, lvl: doc.lvl, words: [tmpwrd]});
        } else {
            alldoc[element].words.push(tmpwrd);
        }
    }
}
```

Listing K.1: Calculating IDF.

Appendix L

Generating Education Vectors

```
db.collection("educations")
.find()
.limit(1100)
.toArray((err, educations) => {
    let metaVector = educations.map(education => {
            return education.words.filter(word => {
                    return word.status !== -1
                }).map(word => {
                    return word.word
                });
    }).reduce((acc,words)=>{
        return [...acc,...words];
    },[])
        .filter((v, i, a) => a.indexOf(v) === i)
        .sort();

    let locMetaVect = educations.map(el => {return el.campus})
        .filter((v, i, a) => a.indexOf(v) === i)
        .sort();

    let eduVectors = educations.reduce(function(eduVectors, education) {

        let eduVector = [];

        let locVector = locMetaVect.map(
            el => {
                if (el === education.campus){
                    return {
                        'campus':el,
                        'weight':1
                    };
                }else{
                    return {
                        'campus':el,
                        'weight':0
```

```
                    };
                }
            }
        );
        let validWords = education.words.filter(word => {
                return word.status !== -1
        }).map(word => {
            return {'word':word.word,
                    'tfidf':word.tfidf}
        });

        for(let word of metaVector){
            let validWord = validWords.find((el)=>{
                return el.word === word
            });
            if(validWord !== undefined){
                eduVector.push(validWord)
            }else{
                eduVector.push({
                    'word':word,
                    'tfidf':0
                })
            }
        }
        eduVectors.push({
            'eduName':education.name,
            'eduId':education._id,
            'vector':eduVector,
            'locvect':locVector,
        });
        return eduVectors
    },[]);

    db.collection('vectors').drop(function () {
        db.collection('vectors').insertMany(eduVectors, function (err, r) {
            callback();
        })
    });
})
```

Listing L.1: Calculating vectors.

Appendix M

Applying stopwords

```
var collection = db.collection("words");
var filename = file;
var content;
fs.readFile(path.join(__dirname, filename), 'utf8', function (err, data) {
    if (err) {
        console.log(err);
        process.exit(1);
    }
    content = tokenizer.tokenize(data);
    for (let word of content) {
        collection.updateOne({word: word}, {$set: {status: -1}}, {}, function (err, r) {

        });
    }
    callback();
});
```

Listing M.1: Applying stopwords.

Appendix N

Prototype

The full code for the prototype can be found:

- In a Github repository at: https://github.com/nicfol/Edu-RS/
- In the .zip file attached to the project