NTNU
Norwegian University of Science and Technology

A support system for student exchange

An exploration of course comparison, and the design of a system aiding student counselors working with exchange

Magnus Lund
Andreas Røyrvik

Master of Science in Informatics
Submission date: June 2017
Supervisor: Rune Sætre, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

The goal of this thesis is to propose a system for student counselors working with exchange students, and study its usefulness. The project aims to define the requirements for such a system, and to explore the possibility of automatically determining course similarity based on the metadata available.

To identify the requirements, a questionnaire was sent out to student counselors and interviews were conducted with potential users of the system. Based on findings from the data generation methods, a paper prototype of the proposed system was developed. The requirements of the system were derived from the results of testing the paper prototype. Based on these requirements, a prototype of the system was developed.

This prototype was tested on potential end users of the system, with interviews concluding the usability study. To evaluate the quality of the functionality for automatically determining course similarity, a set of courses already approved by student counselors were fed into the system, and the output was compared.

The results are promising, especially regarding the usefulness of the system.

Sammendrag

Målet med denne oppgaven er å foreslå et system for studieveiledere som jobber med utveksling, og studere systemets nytteverdi. Formålet med prosjektet er å definere krav (requirements) til hvordan et slik system bør være, samt utforske muligheter for å automatisk anslå likheten mellom fag basert på tilgjengelig metadata.

I første omgang ble en spørreundersøkelse sendt ut til studieveiledere, og intervjuer satt opp med potensielle brukere av systemet. Basert på funnene fra disse datagenereringsmetodene, ble en papirprototype av systemet utviklet. Kravene for systemet ble utledet basert på resultatene av testing av papirprototypen. Med utgangspunktet i disse kravene, ble til slutt en softwareprototype utviklet.

Denne prototype ble testet på potensielle brukere av systemet, der det til slutt ble gjennomført intervjuer for å avgjøre nytteverdi og brukbarhet. For å demonstrere kvaliteten på funksjonaliteten for automatisk sammenligning av fag, ble fag allerede godkjent av studieveiledere matet inn i systemet, og resultatene sammenlignet.

Resultatene fra prosjektet er lovende, spesielt i forhold til nytteverdien av det foreslåtte systemet.

Preface

This thesis was written by Magnus Lund and Andreas Røyrvik during the autumn of 2016 and spring of 2017. It is the final delivery for the Master of Science degree in Informatics at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway.

We would like to thank the project supervisor Rune Sætre for his guidance and input throughout the project.

Trondheim, June 1, 2017

Contents

List of Figures

List of Tables

List of source code listings

Glossary

course curriculum Refers to all textual documents part of the curriculum of a given course.

course dependency A course dependency is a relationship where course A is a prerequisite needed in order to take course B. Course B is then said to be dependent on course A.

course link An undirected link between two courses, confirming that they have been approved as similar enough to be regarded as equivalent.

course metadata Refers to all structured data available about a given course.

student counselor An academic employee, usually affiliated with a specific department, responsible for helping students choose what courses to take, either abroad or domestic.

Acronyms

API	Application Programming Interface.
CSS	Cascading Style Sheets.
IDF	Inverse Document Frequency.
IDI	Department of Computer Science (Institutt for Datateknologi og Informatikk).
JSON	JavaScript Object Notation.
MVC	Model-View-Controller.
NLTK	Natural Language Toolkit (for Python).
NTNU	Norwegian University of Science and Technology.
ORM	Object-Relational Mapping.
SQL	Structured Query Language.
TF	Term Frequency.
TF-IDF	Term Frequency-Inverse Document Frequency.
UI	User Interface.
UUID	Universally Unique Identifier.
WSGI	Web Server Gateway Interface.

Chapter 1

Introduction

In 2015 it was decided that the Norwegian University of Science and Technology (NTNU) would merge with Aalesund University College, Gjøvik University College and Trondheim University College [*NTNU merging with HiAls HiG and HiST* 2016]. As a result of this the new university was, at the beginning of this project, reworking its organizational structure to accommodate for the merge. That includes going through the study programs of the old university and university colleges, in order to find overlapping courses [*NTNU merge FAQ* 2016]. Having multiple courses with the same syllabus is redundant, and these could be merged into one course. The university merge was initiated in January 2016.

After the merge, the new NTNU offered over four thousand courses for its students [*NTNU list of subjects* 2016]. Some courses will likely be similar to each other. These courses needs to be evaluated manually; an enormous job which takes a considerable amount of time and resources.

1.1 Motivation

The motivation behind this work is based on a thesis suggestion (Figure 1.1) made by Rune Sætre at the Department of Computer Science (IDI) at NTNU.

Now that NTNU has merged with HiST, HiGjøvik or HiÅlesund we should look at how the different courses offered at all these institutions overlap or complement each other. And to make the job easier, for student exchange and collaboration, a useful IT tool should be made.
Suggested functionality for the tool:

- It should be possible to make a map of how the different classes depend on each other, within a specified field of study.
- It should be possible to calculate how many percent two classes from the same or different institutions overlap with each other.
- A scalable algorithm should be made to intelligently calculate the overlap between classes based on textual descriptions, including keywords, curriculum, bullet points, etc.

Figure 1.1: The original project proposal by Rune Sætre

When NTNU merged with the colleges previously mentioned, many new and potentially similar courses were added to the course catalogue. The initial goal of this project was to help departments compare the new courses from the old university colleges. The departments needed to map which courses at NTNU that corresponded to the courses taught at the old colleges.

The idea was to create a system that kept track of all courses at the newly merged university. Within that system, counselors and administrative employees at different departments could compare, and find similar courses within the course catalogue.

During the initial research phase, it was discovered that the process of comparing and keeping track of course similarity, is a common use case for counselors working with student exchange as well.

Because the situation of the merge at NTNU is temporary, and the system proposed would only be useful in a transitional period, it was decided to pivot the project towards exchange studies instead.

From the information presented, we wish to design, implement and evaluate a prototype of a system that can be used by student counselors to compare, and keep track of similar courses at other universities and colleges.

The goal of the system is to help increase the efficiency of exchange counseling, for both students, counselors and lecturers. We want to achieve this by reducing the amount of manual work done when aiding students that are considering traveling abroad on exchange studies.

If the creation of such a system is successful, it might also solve the need for a course comparison system at NTNU.

1.2 Research questions

During this project, we hope to achieve advances in the knowledge of what digital tools are required in order for student counselors to work more efficiently. We will do this by investigating the digital needs of student counselors, and look at what the requirements of the proposed system proposed should be. The research of this paper will address the following research questions:

- RQ1: What are the requirements for a digital tool for counseling potential exchange students, and is there a need for such a system?
- RQ2: To what extent would it be possible to automate the process of determining course similarity?

Throughout this project, we will use NTNU as an example university. The requirements of the system will be derived based on data collected from counselors at this university.

1.3 Contributions

The main contributions of this project will be a set of requirements for a system that helps student counselors keep track of courses and course relations, an architectural proposition for the system, and a software prototype of that system. In addition, this project will contribute with survey data on the needs student counselors have for a digital tool.

1.4 Thesis structure

This section contains a brief overview of the thesis structure.

- **Chapter 1: Introduction** presents the motivation behind the thesis, the research questions and its contributions.
- **Chapter 2: Method** presents the research methods, research process and the approach taken for data collection and analysis.
- **Chapter 3: Prestudy** consists of two parts; a section about findings, including results from the conducted questionnaire, and a section where concepts from the literature are presented.
- **Chapter 4: Architecture and Design** explains the architectural planning of the system. Presents the project's stakeholders and defines the architectural requirements (functional requirements, quality requirements and business requirements). The chapter also contains details about the design and evaluation of the paper prototype.
- **Chapter 5: Implementation** gives details about how the prototype was built, and how the various requirements detailed in Chapter 4 were met. The chapter explains the various parts of the system, and how they interact.
- **Chapter 6: Results** describes the results from testing and evaluating the prototype.
- **Chapter 7: Discussion and conclusion** concludes the project based on the results, and gives recommendations for future work.

Chapter 2

Research method

This chapter describes the research methods used in this project, and the reasoning behind choosing them. Furthermore, it will discuss the data generation methods chosen, and how the data from these methods will be analyzed.

Using the model of the research process defined by [Oates 2006], the research method *design and creation* was chosen as the research strategy. The model can be seen in Figure 2.1, where the selected methods are highlighted. Based on the defined research questions, an IT artifact which can be used by student counselors will be developed. A *questionnaire* will be designed and conducted to get a better understanding of the problem domain. This will give us quantitative data for analysis. Additionally, *interviews* will be conducted with potential users of the system. This will produce qualitative data. Using more than one data generation method is called method triangulation [Oates 2006]. These two data generation methods combined can help us verify the findings. Data gathered will be analyzed, and used in the process of designing a paper prototype.

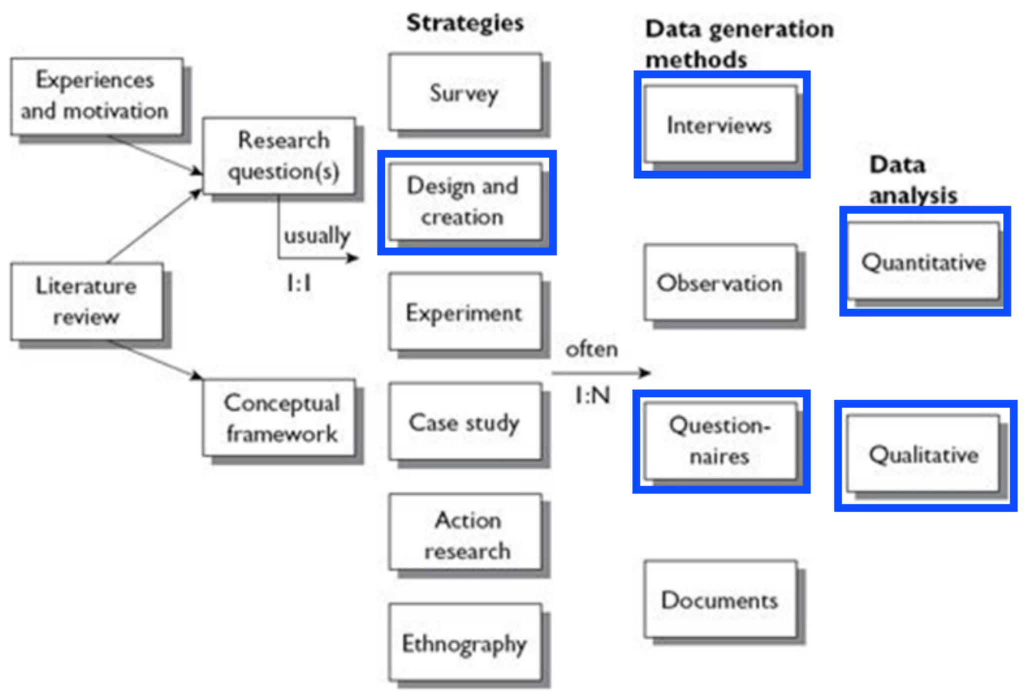


Figure 2.1: Research processes, as modelled in *Researching Information Systems and Computing* [Oates 2006]

The research process of the project is summarized in Figure 2.2. The design and creation iteration will be completed twice: first for the paper prototype, and then for the software prototype. The project will consist of four iterations, completed in consecutive order:

1. Prestudy: Interviews and Questionnaire
2. Design and Creation: Paper prototype
3. Design and Creation: Software prototype
4. Final evaluation

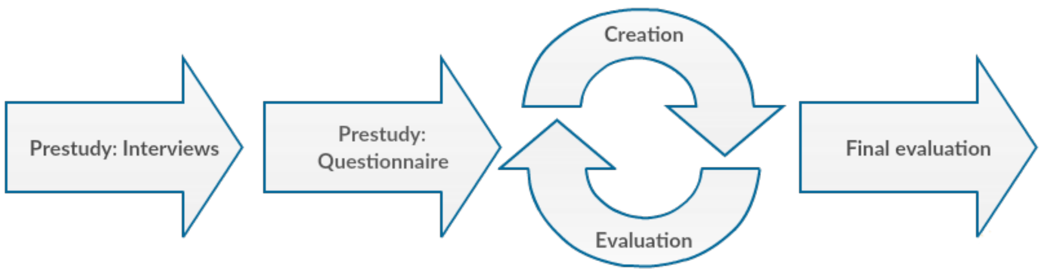


Figure 2.2: Research process

2.1 Interview

To learn more about the subject area, interviews were conducted with student counselors. The purpose of research interviews is to explore experiences, views and beliefs of individuals on specific matters [Gill et al. 2008]. It was decided to conduct unstructured interviews because they have the advantage of not being scripted to the same degree as structured interviews. This allows the interviewer to explore a topic by letting the interview subject talk freely based on open ended questions. Unstructured interviews are well suited where very little are known about the subject area [Gill et al. 2008].

2.2 Questionnaire

In order to validate the concept of the suggested system, and to get a better overview of the situation today, a questionnaire was created. A questionnaire is a research instrument for the collection of data [Oppenheim 2000]. In its simplest form it is a list of questions that are to be filled out by respondents. The resulting data are used to create an understanding of the population which the respondents are a part of [Wohlin et al. 2000]. For example, by asking 50 students of a particular course what they think about the lectures, the opinion of the entire class of 200 students can be assessed.

Collecting data through a questionnaire has both advantages and disadvantages. It's an easy and cost effective tool to reach a large amount of respondents in a short period of time. Collected data can be used for quantitative analysis. Compared to e.g. conducting interviews with relevant subjects, it is a time saving tool. However, one must also be aware of the disadvantages related to questionnaires as a tool for data collection. There is no way of verifying the truthfulness of a respondent, how much thought they have put into their answers and whether they have misinterpreted the questions [Wright 2005].

The questionnaire used in this paper was designed to be as short and concise as possible, in order to maximize the amount of replies. Questionnaires with a large number of questions are tiresome to fill out, and might affect data quality [Wohlin et al. 2000].

More details about the questionnaire is presented in Section 3.1.

2.3 Prototype

2.3.1 Paper prototype

Paper prototyping is a well-established method of designing, testing, communicating and improving user interfaces [Snyder 2003]. The method allows developers to validate a suggested design before implementing it. Problems with the design will quickly be identified by running usability tests with end users. In a usability test, the test subjects are given a set of tasks they should solve. The test is done entirely by using the paper prototype, where the test subject interacts with it as they would with a full-fledged system. The tasks should be focused around exploring how users interact with the parts of the system that the tester wants to learn more about. A good task covers questions important to the success of the product, and should have a reasonably narrow scope and a clear end point [Snyder 2003].

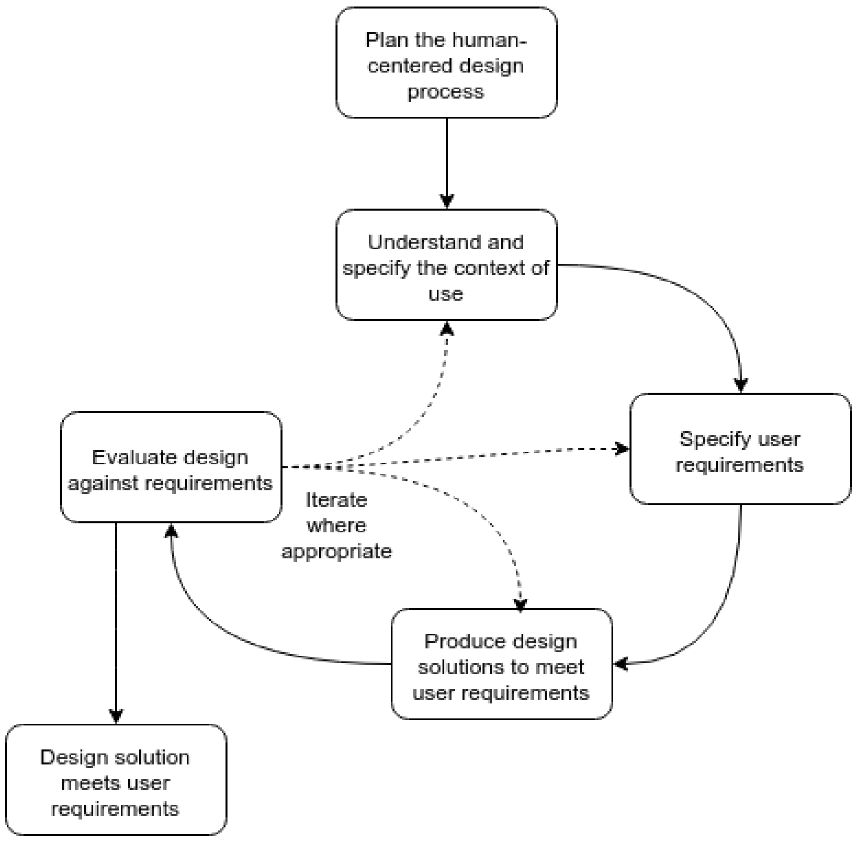

Plan the human-centered design process
Understand and specify the context of use
Specify user requirements
Evaluate design against requirements
Iterate where appropriate
Produce design solutions to meet user requirements
Design solution meets user requirements

Figure 2.3: ISO 9241-210 - Human-centred design for interactive systems [ISO 2010]

The design process followed while planning the system is formalized in ISO 9241-210 - Human-centred design for interactive systems [ISO 2010], as shown in Figure 2.3. The standard describes an iterative process that focuses on communication with the end user throughout all design phases and iterations. This entails active involvement of the end user in order to get an understanding of their needs, what problem the system will solve for them, and in what environment the system is to be used. This is called the *context of use*.

The process was initiated with the questionnaire, and subsequent interviews, as a tool for understanding the context of use. Furthermore, a list of requirements were sketched based on the results. Based on the requirements derived from the questionnaire and interviews, a paper prototype of the system was created. More on this in Section 4.2. Evaluation of the paper prototype with end users concluded the first iteration of the process.

Evaluation of the paper prototype

To evaluate the paper prototype, a set of tasks were designed, which covered the most important functionality of the prototype. Test subjects were presented with the tasks, and asked to attempt to solve them. Feedback from the test subjects, and how they solved the tasks were used in the next iteration of the research project. Section 4.2.2 contains more information about the evaluation of the paper prototype.

2.3.2 Software prototype

When the iterative process of designing and evaluating the paper prototype was concluded, the process of implementing the software prototype was initiated. Chapter 5 describes this process in detail.

Evaluation of the software prototype

After the software prototype was implemented, it was evaluated by test subjects through interviews and a usability test. Chapter 6 contains results of this software prototype evaluation.

Chapter 3

Prestudy

This chapter will first give an overview of the questionnaire and its results. Secondly, it will present a summary of findings derived from the questionnaire and interviews, that are relevant in order to understand the architecture of the system proposed in Chapter 4. Lastly, it covers the theoretical knowledge necessary to understand the technical implementation of the prototype.

3.1 Questionnaire

The finished questionnaire consisted of nine questions. The questions were directed to assess how much work the counselors spend each week on course comparison, how they store and maintain that information today, and to what extent they felt the proposed system would make their work flow more efficient. The full list of questions can be viewed in Appendix C.

The student counselors that were added as recipients to the questionnaire, were found in a list of counselors maintained by the university[1].

The article contains shortcuts to every faculty's list of student counselors. The questionnaire was created using Google Forms[2], which is a free tool for survey creation

[1]https://innsida.ntnu.no/wiki/-/wiki/Norsk/Studieveiledning (as of April 29, 2017)

[2]https://docs.google.com/forms/ (as of May 06, 2017)

and gathering of results.

The questionnaire was sent out with instructions on how to fill it out, along with an introduction to the research project and how their answers would help it going forward. At the end of the questionnaire, the participants could leave their email if they were interested in providing additional feedback before, during and after the development of the system.

3.1.1 Distribution and response

The questionnaire was distributed to a total of 44 student counselors from 21 different departments and five different faculties at NTNU (Faculty of Natural Sciences, Faculty of Engineering, Faculty of Medicine Health Sciences, Faculty of Humanities, and Faculty of Information Technology and Electrical Engineering). The distribution of the questionnaire was divided into two iterations. The first iteration included 25 recipients. Due to a low response rate it was decided to send the questionnaire to an additional 19 recipients. A total of 17 responses ($\approx 39\%$ response ratio) were recorded.

3.1.2 Results from questionnaire

This section presents selected findings from the questionnaire. The full set of results can be viewed in Appendix C.

How does your organization store documentation about subjects taken by students during exchange semesters, for example which NTNU course best correspond to a course from another university?
An interesting find is that approximately 31% of respondents state that they do not store documentation about the similarity of exchange courses and their equivalent on NTNU (Figure 3.1). This indicates that there is room for improvement, and that a system like the one proposed in this project might be of use for student counselors.

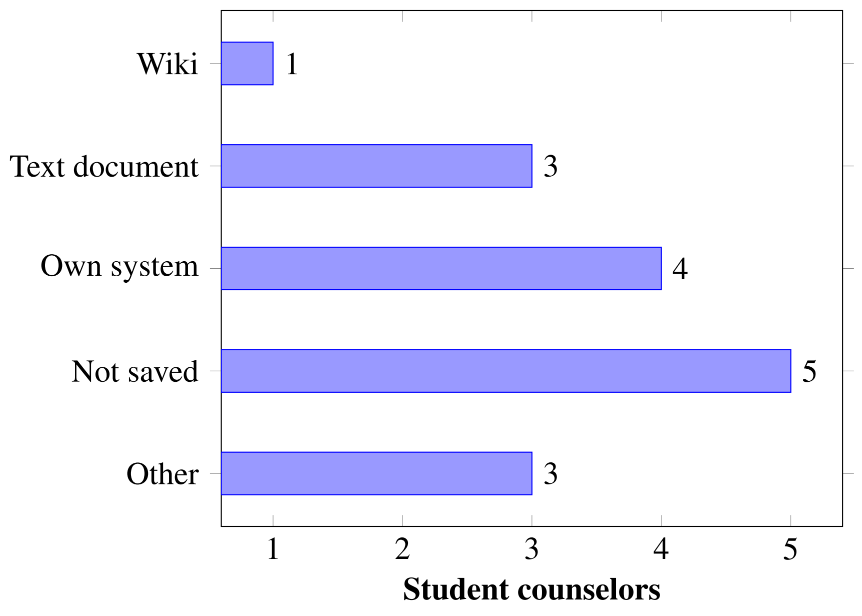


Figure 3.1: Systems used today

To what degree would a tool that automatically compares courses based on their curriculum make your work day more efficient? 16 out of 17 respondents state that a system which automatically compares courses based on their curriculum would make their work day more efficient to some degree or to a large degree (Figure 3.2).

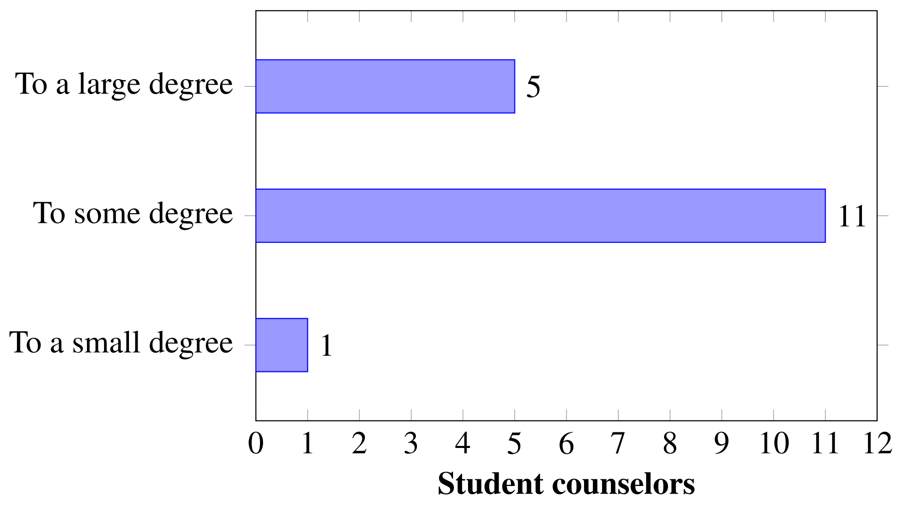


Figure 3.2: Course curriculum - degree of usefulness

In general, the results of the questionnaire suggests that there may be a need for a digital tool when counseling potential exchange students. Even though the process of comparing courses might not be as time consuming as first assumed, the survey results indicate that there is a need for centralizing the information of courses that have already been evaluated.

Based on the feedback from the questionnaire, the first draft of the architectural requirements was created. These are described in Section 4.3.

3.2 Interviews

From the questionnaire, seven student counselors left their contact information for further participation to the project. Interviews were scheduled with two of them, with the following pre-planned talking points:

The interviews aimed to explore the following subjects:

- The process of finding relevant courses for student exchange on another university
- The process of approving courses that students wishes to take while on exchange
- Current digital tools for student counselors
- The usefulness of the proposed system
- How, and if, student counselors share information about approved courses with colleagues

The findings from the interviews are presented in Section 3.3.

3.3 Summary of questionnaire and interviews

This section will present findings that contributed to creating the requirements for the system. It will first describe collected information relevant for designing a prototype that works at NTNU. Secondly, it will discuss findings that are related to the design of a prototype that works with other universities as well. The source of these findings are interviews with student counselors, and information available online [Innsida 2017].

3.3.1 NTNU

Exchange process

In order to derive requirements for a system that can help student counselors, it is important to understand how the process of guiding exchange students works today. If a student wants to study abroad, the process is as follows:

1. The student needs to decide what university they want to study at, and what courses they want to take. The selected courses should correspond to the ones they would have taken at NTNU the same semester(s).
2. The courses that the student decides on needs to be approved by NTNU. The student counselor contacts the lecturer of each corresponding NTNU course, to make sure that the courses the student has decided on are valid substitute courses.
3. When all courses are approved, the student is ready to travel.
4. During the exchange, it often happens that the courses the student got approved are unavailable, or in some other way not possible to take. The student is then responsible for finding a suitable replacement. When the student returns from the exchange, the student counselor contacts the lecturers of the corresponding courses that the student could not participate in, and ask them to review the suggested replacement course.
5. When all courses are approved, the process is complete, and the student receives the same amount of credits as they would during a normal semester.

Student exchange at NTNU

NTNU has a goal of having 40% of its students traveling abroad on exchange studies [Kommunikasjonsavdelingen 2014]. During the period of 2010-2013, 30% of its students traveled abroad.

There exists many different solutions at NTNU today for keeping track of relations between courses at two universities. These relations will hereby be referred to as course links.

Among the systems that are used by counselors stating that they use a structured system, wiki solutions and Ephorte[3] are the most popular ones. All systems discovered during the prestudy are maintained manually, i.e. no data is collected automatically. A common problem by storing course links in a software system, is that the links become outdated. It is therefore important to note the date when the course link was approved by the lecturer. If the course link is suspected to be outdated, the student counselor has to contact the lecturer to get the link re-approved.

Common metrics used by student counselors to determine if courses might be similar include course descriptions[4], course level[5], credits[6] and curriculum.

To determine course level, some counselors draws graphs of which courses a student needs in order to take a certain course, e.g. what introductory course a student need to take in order to be able to take a certain intermediary course. These types of links will hereby be referred to as course dependencies. Creating such graphs is often done with pen and paper, and can be a time consuming process.

NTNU has its own database for keeping track of how many credits a semester at NTNU corresponds to at another university. This system is referred to as GNAG[7], and is maintained by the Office of International Relations[8] at the university.

[3]https://www.evry.com/no/bransjer-og-tjenester/tjenester/forretningslosninger/ecm/ephorte/ (as of May 16, 2017)

[4]usually found in the course catalogue

[5]an attribute used to describe if a course is introductory, intermediary or advance

[6]a number that describes how comprehensive the course is

[7]https://www.ntnu.no/international/studentweb/gnag/gnag.htm (as of May 16, 2017)

[8]https://www.ntnu.edu/international (as of May 16, 2017)

Having a system that keeps track of approved course links may also be beneficial for students searching for course to take during their exchange. As mentioned in Section 3.3.1, students are responsible for finding suitable courses during their exchange semester. The proposed system can improve this process, by letting students easily see what courses have already been approved by student counselors.

3.3.2 Universities in general

This section is based on interviews with student counselors, and research of other universities online.

The prestudy has not found any universities that do not provide a description of the courses they have in their catalogue online. Many universities also offers APIs for course information. The data quality of these APIs differs greatly, and not all of them are public.

Credits given for a course are in no way standardized between universities. If courses are to be considered equivalent by student counselors, the number of credits of the courses in question needs to be approximately the same (relative to the number of credits at their university).

Curriculum and course data

In order to make course comparison as accurate as possible, the curriculum of a given course was considered as a source of information. In the context of this paper, curriculum will refer to all textual documents part of the curriculum of a given course.

The first issue of using curriculum as a basis for calculating course similarity, is that information about a course's curriculum is hard to find. Few universities have open and maintained overviews of their curriculum. Another issue is getting the content of a given book. Most books costs money, and the prestudy has not uncovered any data sources that would allow the proposed system to analyze contents of books for free.

Some of the books indexed by Google Books[9] can be browsed without paying, and some had their most frequent keywords listed. However, this service was not regarded as a viable data source for this project. Most publishers keep table of contents of their books openly available in their web stores. Table of contents can be seen as a summary of the contents of a book, which may be suitable for the experimental curriculum comparison functionality.

In addition to curriculum, course metadata should be used when comparing courses. In this context, metadata will refer to all structured data available about a given course. The findings in the previous sections suggest that course descriptions and other textual descriptions are important factors when evaluating course similarities.

3.4 Similar systems

In the early stages of the prestudy, research was conducted in order to discover any systems that are similar to the system proposed in this paper. No such system has been discovered.

3.5 Theory

In this project, we will work with substantial amounts of course metadata, such as course descriptions and curriculum. The software prototype should support both comparison of courses and searching for similar courses. This section explains key concepts from the field of information retrieval that are relevant for this project. Information retrieval is an area of Computer Science focused on retrieving documents from a document collection that satisfies an information need [Manning, Raghavan, and Schütze 2008].

[9]https://books.google.com (as of May 18, 2017)

3.5.1 Vector space model

The vector space model is an algebraic model for representing documents and queries as vectors. These vectors should capture the relative importance of a document's individual terms, i.e. they should be assigned some weight. More on this in Section 3.5.2. Documents and queries are represented as t-dimensional vectors, where t is the total number of index terms. A consequence of this vector representation is the loss of word order. E.g. the documents *"Mary is quicker than John"* and *"John is quicker than Mary"* are identical. This is called the *bag of words* model [Manning, Raghavan, and Schütze 2008].

Given a collection of document vectors and/or query vectors, it is possible to represent them in a common vector space. The cosine of the angle between any two vectors is a measure of their similarity. This is known as the *cosine measure*. Figure 3.3 shows the cosine similarity of two documents: $sim(d_1, d_2) = \cos\theta$

The vector space model is a well established information retrieval model, first introduced by the SMART system developed in the 1960s. It allows for partial matching and ranking of results.

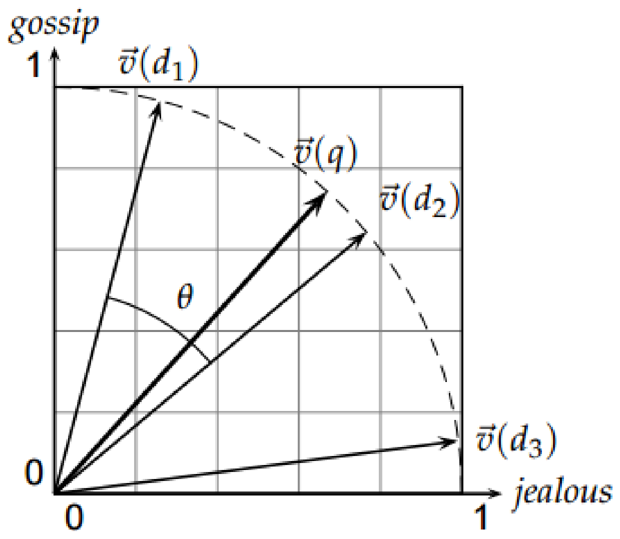


Figure 3.3: The vector space model [Manning, Raghavan, and Schütze 2008]

3.5.2 Term weighting

Not all terms are equally useful for describing the contents of a document. In information retrieval systems, documents are indexed, and index terms are given weights. Index terms that will describe the document contents should be given higher weights than terms which are not useful in describing the same content. Common term weighting schemes include *term frequency* and *inverse document frequency*, as described below.

Term frequency

The frequency f_{ij} of a term is simply how many times it occurs in a document. However, the relevance of the term does not increase proportionally with its frequency. There are several variants for weighting term frequency (TF). The simplest ones are the binary variant (assigns 1 to TF if the term is present, 0 if not), and the raw variant (uses the occurrence count directly). The latter approach is problematic because the importance of a term does not scale linearly with the number of occurrences. Applying a logarithmic scale to term frequency will dampen the effect of terms that have high frequencies [Baeza-Yates and Ribeiro-Neto 2011]. Figure 3.4 shows the formula for log term frequency, where $f_{i,j}$ is the raw term frequency for term i in document j.

$$tf_{i,j} = \begin{cases} 1 + \log_2 f_{i,j} & f_{i,j} > 0 \\ 0 & otherwise \end{cases} \tag{3.1}$$

Figure 3.4: Log term frequency

Inverse Document Frequency

The document frequency df_t of a term t is defined to be the number of documents where t occurs. Common words in a document collection are not good discriminators (i.e. a characteristic which enables documents to be distinguished from each other), while terms that occurs in few documents are often more useful [Manning, Raghavan, and Schütze 2008]. In other words, the document frequency indicates how little important a term is. For measuring a word's importance, we can invert the document frequency. This is called the Inverse Document Frequency (IDF). Thus the IDF of rare words are high, and the IDF of common words are low. IDF weights provide a foundation for term weighting schemes and are used by almost any modern information retrieval system [Baeza-Yates and Ribeiro-Neto 2011]. IDF is defined as follows, where N is the total number of documents in the collection and n_i is the number of documents where the term t_i occurs.

$$idf_i = \log \frac{N}{n_i} \tag{3.2}$$

Figure 3.5: Inverse Document Frequency

TF-IDF

We now combine the term frequency and inverse document frequency. This results in a weighting scheme where the weight of a term increases to the number of occurrences, but is offset by the frequency of the term in the document collection [Manning, Raghavan, and Schütze 2008]. Terms that occur in virtually all documents are given the lowest weight, while terms that occur frequently in a small number of documents are given high weights [Salton and Buckley 1988]. Very rare terms like typos and foreign words are given low weights. Figure 3.6 shows the formula for TF-IDF.

$$TFIDF_{t,d} = tf_{t,d} \times idf_t \tag{3.3}$$

Figure 3.6: TF-IDF

3.5.3 Document preprocessing

Document preprocessing consists of five stages, as described in [Baeza-Yates and Ribeiro-Neto 2011]:

1. Lexical analysis
2. Elimination of stop words
3. Stemming
4. Selection of index terms
5. Construction of thesauri

Lexical analysis (tokenization): The goal of the tokenization step is to determine the words of the document. Punctuation marks and whitespace are usually treated as word separators. Letter casing can be removed, but in some IR-systems it is vital to keep casing. After the lexical analysis step is finished we are left with an array of words.

Stop words: Stop words are words that appear very frequently in a document collection. They are usually of little value when searching, as they are not good discriminators [Baeza-Yates and Ribeiro-Neto 2011]. We are interested in extracting the most descriptive terms. Prepositions, articles and conjunctions are typical stop words. Removal of stop words can be done based on a predefined word list, or by removing the top *n* percentage of the most common words.

By using TF-IDF weighting, most stop words will be ignored because of their high frequency, but will clutter the vector making the computation time longer and more memory intensive. It is typical to obtain a compression of the index of 40% or more by removing stop words [Baeza-Yates and Ribeiro-Neto 2011].

Stemming: A stem is the portion of a word that is left after the removal of its affixes. For example, by removing the suffix in the words *connection* and *connected*, we are left with the stem *connect*. Stems are thought to be useful for information retrieval purposes, as they reduce variants of the same root word into a common base word. Stemming comes in many different forms, with affix removal being the most widely used [Baeza-Yates and Ribeiro-Neto 2011].

Index term selection: Not all terms are equally useful for describing the contents of a document. Instead of using a full text representation, where all terms in a document are used, the most descriptive and important terms can be selected. In a library system, index terms are often selected manually by experts. In other systems, a more automatic approach must be taken. A simple approach is to consider the terms with the highest TF-IDF values as index terms.

Thesauri: A thesaurus is a precompiled list of important terms in a domain, and for each of these terms, a set of related words or synonyms.

Chapter 4

Architecture and design

This chapter explains the architectural requirements that were derived from the findings in the previous chapter. It will first describe the stakeholders of the system. Secondly, it will describe the design and implementation of the paper prototype, followed by a discussion of the architectural requirements for the software prototype. Lastly, it will describe the most important architectural views of the system.

4.1 Stakeholders and concerns

A stakeholder is anyone having an interest in a given project. People affected by a software system are not limited to those using it, but everyone involved in maintaining, testing, operating, developing and paying for the system [Rozanski and Woods 2011]. Each of these groups have their own concerns about the system. The remainder of this section presents the stakeholders of the project, and their concerns.

4.1.1 Student

The student will use the system to browse courses before choosing which courses to take during the exchange. Consequently, the student will mostly be concerned with what the system offers in terms of usability and having a database that is as complete and up to date as possible. Students want a system that is easy to use. They also do not want to run into bugs or errors which can ruin their experience.

4.1.2 Student counselor

The student counselor will use the system to keep track of courses and relations between courses. This includes:

- Keeping course relations up to date
- Keeping track of courses that have previously been approved as equivalent to courses on other universities
- Adding new courses and course relations

The counselor is also responsible for contacting the relevant lecturer when a new course relation is suggested by a student. The counselor inherits the concerns of the student, as they are also users of the system. In addition, a system that is easy to use would reduce the workload for the counselor, because students would require less help to find courses.

4.1.3 Lecturer

The lecturer is responsible for reviewing curriculum. They ensure that a course relation to a course on another university is only approved if their similarity is satisfactory. Because the lecturer should be contacted by the counselor when a new course need to be reviewed, the lecturer will most likely not use the system directly. The lecturer's concern is that the system is available and easy to use, so that students does not contact them, but rather use the system and contact the counselor for any enquiries.

4.1.4 System maintainer

The system maintainer will be responsible for ensuring that the system runs as it should. They will also be responsible for importing new universities through APIs, spreadsheets and other data sources. The maintainer wants a system that is easy to maintain, modify and with a good interface for running import scripts.

4.1.5 University

The university is interested in having a work flow that is as streamlined and efficient as possible. The administration want a system that is cheap and easy to maintain, and that reduces the cost of approving courses for students traveling abroad. The university is also concerned with reaching their goal of sending at least 40% of its students on exchange for one or two semesters [Kommunikasjonsavdelingen 2014].

4.2 Paper prototype

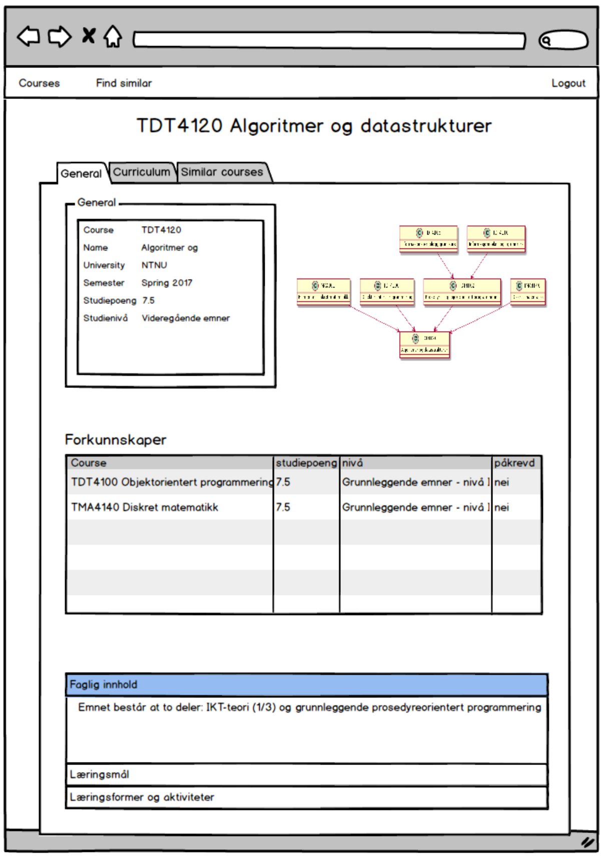
This section describes the design and evaluation of the paper prototype.

4.2.1 Design of the paper prototype

Balsamiq[1] is a software tool for rapidly mocking up graphical user interfaces, and was used to create the prototype. This section presents two important prototype pages: the course overview page, and the compare courses page. The entire paper prototype can be found in Appendix D.

Figure 4.1 shows the overview page for a course. It contains all the information about a course that the user needs. The metadata included here is based on feedback from potential users (as presented in Section 3.3). The image on the right hand side of the page is the course dependency tree.

[1]https://balsamiq.com (as of 26 April, 2017)



TDT4120 Algoritmer og datastrukturer

General

Course	TDT4120
Name	Algoritmer og
University	NTNU
Semester	Spring 2017
Studiepoeng	7.5
Studienivå	Videregående emner

Forkunnskaper

Course	studiepoeng	nivå	påkrevd
TDT4100 Objektorientert programmering	7.5	Grunnleggende emner - nivå I	nei
TMA4140 Diskret matematikk	7.5	Grunnleggende emner - nivå I	nei

Faglig innhold

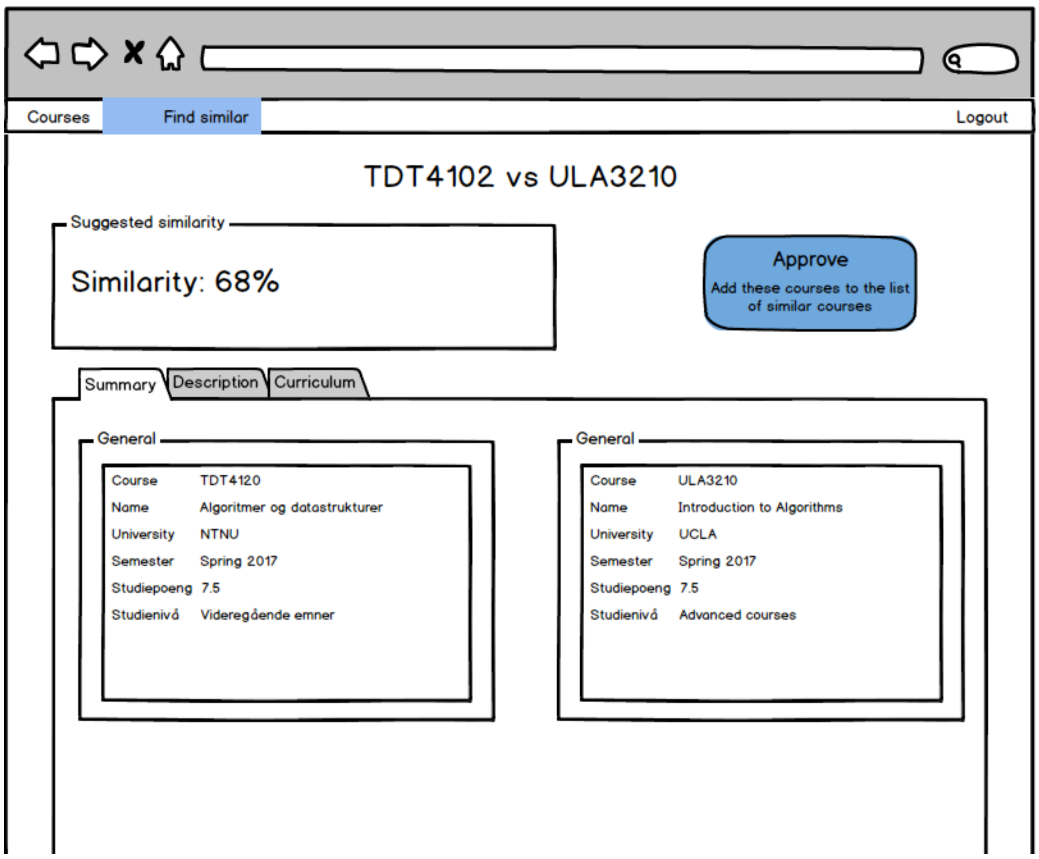
Emnet består at to deler: IKT-teori (1/3) og grunnleggende prosedyreorientert programmering

Læringsmål

Læringsformer og aktiviteter

Figure 4.1: Paper prototype: Overview of a course

Figure 4.2 shows the course comparison view. Users can compare courses side by side, and get a similarity suggestion. If the user is logged in as an admin, the *'Approve'-button* is visible.



Similarity: 68%

Course	TDT4120
Name	Algoritmer og datastrukturer
University	NTNU
Semester	Spring 2017
Studiepoeng	7.5
Studienivå	Videregående emner

Course	ULA3210
Name	Introduction to Algorithms
University	UCLA
Semester	Spring 2017
Studiepoeng	7.5
Studienivå	Advanced courses

Figure 4.2: Paper prototype: Course comparison

4.2.2 Evaluation of the paper prototype

As previously mentioned, the questionnaire contained a field where the subjects could leave their contact information if they wanted to contribute further in the development of the system. Paper prototype tests were conducted with two student counselors.

When meeting with the test subjects, they were presented with a printed out version of the paper prototype. They were then given a series of five tasks. The tasks presented were based on what was seen as the most important system requirements:

1. Look up the course TDT4120: Algorithms and Data Structures
2. Find the curriculum for TDT4120: Algorithms and Data Structures

3. Check if there exists an approved replacement for TDT4120: Algorithms and Data Structures at University of Waterloo
4. Find the suggested similarity of TDT4120: Algorithms and Data Structures and CS341: Algorithms
5. Approve the course CS341: Algorithms from University of Waterloo as a valid replacement for TDT4120: Algorithms and Data Structures

The test subjects had no problems solving the tasks. However, they missed functionality for directly comparing two courses. Some important course meta data was also omitted. After all test sessions were completed, the data that had been collected was analyzed, and a new iteration of the system requirements were created.

4.3 Architectural requirements

In this section, we describe the functional requirements, quality requirements, and business requirements that are regarded as having the most significant impact on the system architecture.

Throughout this section, a student counselor and anyone that needs to make modifications to the underlying data, is defined as an *admin*.

4.3.1 Functional requirements

This section contains the functional requirements (FR) derived from the findings covered in the prestudy.

General

The most fundamental part of the system is the ability to store and access course information. For the system to be a success, it must support basic administration of courses and universities.

FR 1: The system should keep track of courses belonging to a university

FR 1.1 The system should display relevant course information

FR 1.2 Admins should be able to create courses

FR 1.3 Admins should be able to edit courses

FR 1.4 Admins should be able to make notes on a course

FR 2: The system should support multiple universities

FR 2.1 Admins should be able to create universities

FR 2.2 Admins should be able to edit universities

Computing similarity

The prestudy found that most student counselors thought a system for aiding them in comparing courses would be useful, to some extent. The following requirements are derived from the feedback:

FR 3: The system should be able to assist users in determining the similarity of two courses

FR 3.1 The system should be able to calculate the similarity of two courses based on their respective metadata

FR 3.2 Users should be able to search for similar courses to any given course

Searching and browsing

NTNU's course database consists of over four thousand courses. A feature for searching, filtering and browsing the course database is therefore important.

FR 4: The system must allow users to search through all courses in the database

FR 4.1 Courses must be searchable

FR 4.2 Courses must be browsable

Authentication

The authentication requirements are not directly acquired through feedback from student counselors, but are more implicitly acquired through understanding the importance of data quality in this system. Not everyone should have access to manipulate data.

FR 5: The system should support user login

FR 5.1 Admins should be able to log in with their university account

FR 5.2 Admins should be able to log out

Course link administration

Keeping track of approved course links will be the most important feature in this system. To replace the different old solutions discovered during the prestudy, the system will have to fulfill the following requirements:

FR 6: Admins should be able to approve course links

FR 6.1 Admins should be able to create new course links

FR 6.2 Admins should be able to remove course links

Course dependencies and dependency management

Keeping track of course dependencies was discovered as a feature many student counselors would appreciate. The following requirements regarding course dependencies were derived:

FR 7: The system should keep track of the dependency between courses

FR 7.1 Admin should be able to add new course dependencies (prerequisites)

FR 7.2 Admin should be able to remove course dependencies

FR 7.3 The system should be able to visually represent all course dependencies for a given course

FR 7.4 The system should be able to differentiate between required and optional course dependencies.

4.3.2 Quality requirements

Quality requirements, often referred to as non-functional requirements, are requirements that specifies criteria that can be used to judge how a system operates, rather than specific behaviors [Bass 2012].

This section describes the quality attributes selected for this system, along with their requirements. Quality attributes refers to the qualification of requirements, rather than concrete functional requirements [Bass 2012]. All quality requirements are derived from the findings covered in the prestudy.

Usability

The usability attribute is about how easy it is for users to perform what they want to with the functionality that the system provides, and how satisfied they are with it [Bass 2012].

Usability was chosen as a quality attribute because it is important that student counselors are satisfied with the functionality of the system. If they are not satisfied, they won't use it. In this system, the student counselors are the main data providers. If a student counselor doesn't use it, the system loses some of its value.

Quality attributes of type usability are prefixed with the letter U.

U1: Easy to learn system features and start managing courses

In order for the system to be a success, it must be easy to get started. Users should not have to use a lot of time learning to know the system before using it.

Source of stimulus	Student counselor
Stimulus	Visits website
Artifacts	System
Environment	Runtime
Response	The user has logged in and learned all the main features of the system
Response measure	Within 10 minutes of experimentation

Table 4.1: U1: Easy to learn system features and start managing courses

U2: User satisfaction

If users don't see the value of the system, they won't continue using it. For the system to work as intended, it is imperative that as many student counselors as possible uses it as their primary course database. It is therefore important that users are satisfied with the system. It should also avoid frustrating the user with non-intuitive user interface (UI).

Source of stimulus	Student counselor
Stimulus	Tries the system for the first time
Artifacts	System
Environment	Runtime
Response	The user evaluates the system after 30 minutes of usage
Response measure	Gives an overall impression rating higher than 7 out of 10

Table 4.2: U2: User satisfaction

Modifiability

The modifiability attribute is about the cost of change to a system, in the form of work hours and complexity [Bass 2012]. This attribute was chosen mainly because the prestudy showed that student counselors already have a lot of course information stored in spreadsheets and similar solutions. To make the transition over to a new system easier, it must be possible to batch import course data from such sources. Modifiability was also chosen because of the experimental nature of course comparison. It should be easy to substitute the comparison module with another one, as more knowledge is gained on the subject.

Quality attributes of type modifiability are prefixed with the letter M.

M1: Batch add courses and universities

If a university wants to start using the system, it should be easy to import course information for all courses at that university.

Source of stimulus	Developer
Stimulus	Add spreadsheet of approved course relations to database
Artifacts	Data / Code
Environment	Runtime
Response	Courses added and are visible in frontend
Response measure	2 hours (excluding parsing spreadsheet)

Table 4.3: M1 - Batch add courses and universities

M2: Modify courses and course relations

An important feature of the proposed system is the functionality of keeping track of course relations. Maintaining these links must be easy and efficient for student counselors.

Source of stimulus	Student counselor
Stimulus	Approve link between two courses
Artifacts	Data
Environment	Runtime
Response	Course relation is added and is visible in frontend
Response measure	Less than 1 minute

Table 4.4: M2 - Modify courses and course relations

M3: Modify comparison algorithm

The course comparison algorithm in this project is experimental. In order to make improvements to the algorithm, it should be decoupled from the rest of the code, and easy to replace.

Source of stimulus	Developer
Stimulus	Replace comparison algorithm
Artifacts	Code
Environment	Design time
Response	Change made and tested
Response measure	3 hours

Table 4.5: M3 - Modify comparison algorithm

Interoperability

The interoperability attribute is about the extent of which two or more systems can exchange meaningful information between each other [Bass 2012]. The attribute was chosen because data stored in this system very likely will be useful for other systems as well.

Quality attributes of type interoperability are prefixed with the letter I.

I1: Easy to collect course data from external APIs

It should be easy to extract data from the system.

Source of stimulus	Developer
Stimulus	Create a new system that uses the course API as a data source
Artifacts	Code
Environment	Design time
Response	New system can use course data from API
Response measure	2 hours

Table 4.6: I1: Easy to collect course data from external APIs

4.3.3 Business requirements

The primary business requirement for this system is that the university improves and streamlines its process of approving courses before students travel abroad. This can significantly improve the way student counselors work today by reducing time spent for both counselor and lecturer.

4.4 Selection of Architectural Views

Software architecture deals with design of the high-level structure of a software system, and the abstraction and decomposition of this design. The *4+1 architectural view model* is a model for describing the architecture of a software system. The model is the result of a proposal to organize the description of a system's architecture using several *views*, each one addressing a specific set of concerns [Kruchten 1995]. A view is a representation of an entire system from different perspectives.

Each of the views in the "4"-part of the model represents the system from the view-point of different stakeholders. These four views are the following:

1. Logical view
2. Process view
3. Physical view
4. Development view

This section describes these four views in light of the architecture of the system. The "+1" part of the model is the scenario view. The scenario view describes sequences of interactions between processes and between objects. It contains instances of general use cases which describes the most important requirements. Its purpose is to identify the architectural elements of a system, and to validate an architectural design. The view is redundant with the other four views, hence the "+1" [Kruchten 1995]

4.4.1 Logical View

The logical view supports the functional requirements. It focuses on addressing the end user's concerns about the system supporting all the desired functionality. The resulting view is a representation of the system at module or class level. Figure 4.3 shows the logical view for the proposed system. The frontend will have four views, where all views are designed to lead the user to the *CourseDetails* view, where most of the course functionality resides. *CourseDetails* will have four tabs, each containing different types of course information.

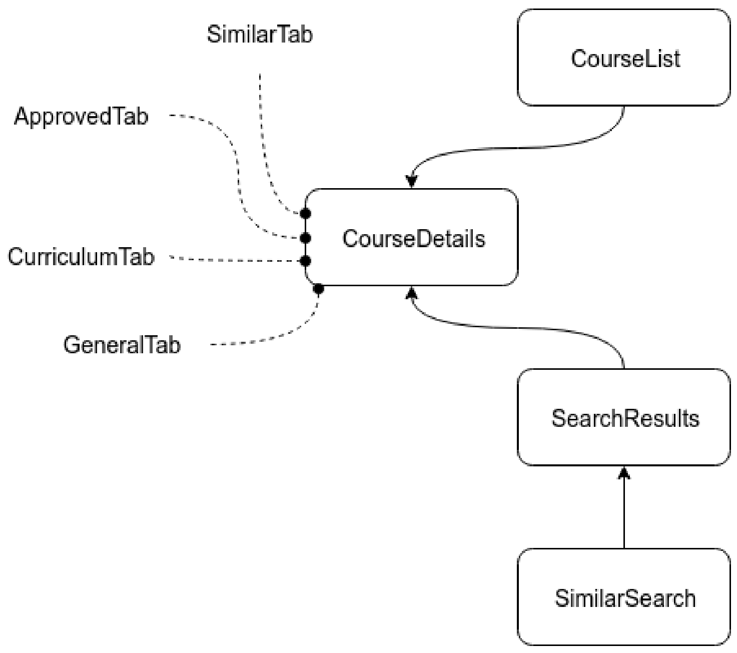


Figure 4.3: Logical view

4.4.2 Development View

The development view focuses on the software module organization of the development environment [Kruchten 1995]. It describes the architecture that supports the development process. The stakeholders of the view are software developers and testers [Rozanski and Woods 2011].

Figure 4.4 shows the system's development view. The server (backend) will rely on Python and Django, while the client (frontend) uses JavaScript/React (see Chapter 5). The client and server will communicate over HTTP using REST.

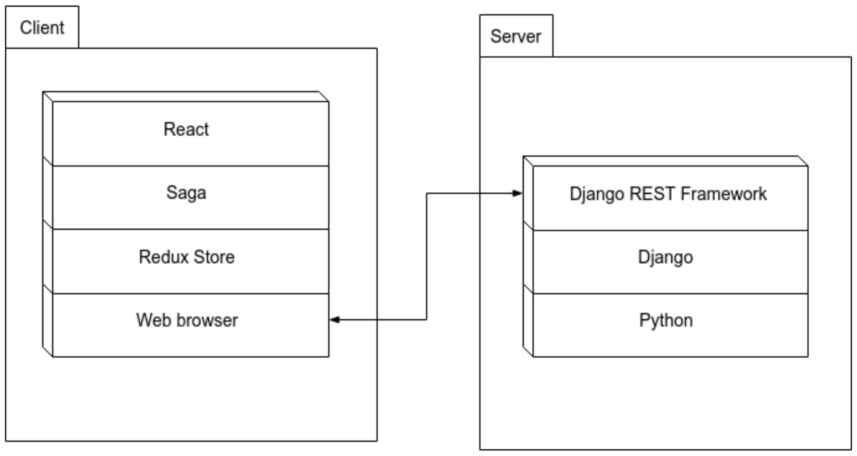


Figure 4.4: Development view

4.4.3 Process View

A process is a group of actions that together form an executable unit. The process view attempts to capture the significant processes in the system. The stakeholders of the process views are the system integrators and developers.

Figure 4.5 shows the process view (in the form of an activity diagram) for approving a course relation, which is an architectural important process in the system. After logging in to the system, the user finds one of the two courses in the new course relation. In the tab containing approved courses, the user searches for the other of the two course in the new course relation. If that course already exist, the user can simply click to create the course relation. If that course does not exist, the user first creates it, and then creates the course relation.

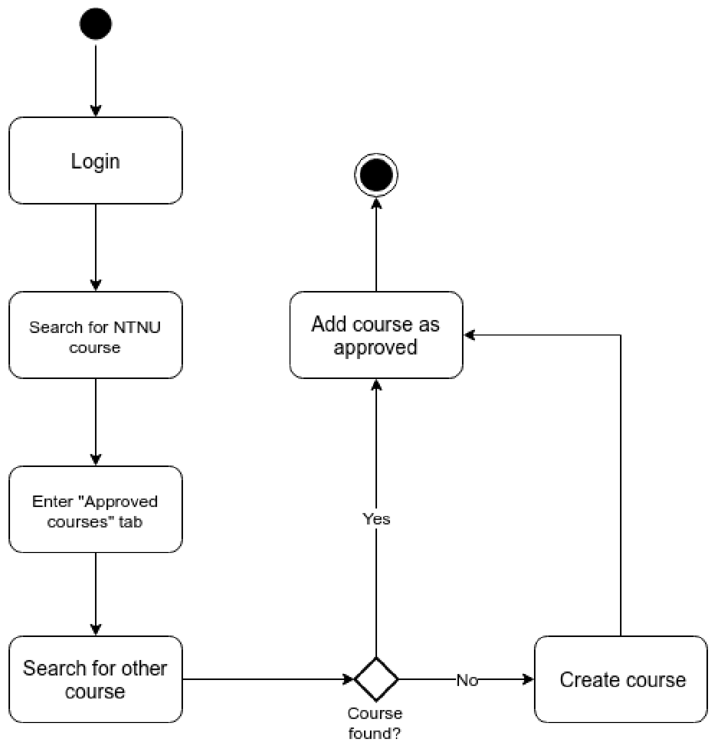


Figure 4.5: Process view: Approving a course relation

4.4.4 Physical View

The physical view attempts to capture the mapping of software to hardware. The view focuses on non-functional requirements related to the stability of the system, such as availability, reliability and performance. The view is also concerned with representing the physical connections and separation of nodes in the system [Kruchten 1995]. The stakeholders of the physical view are system maintainers and technical administrators.

Figure 4.6 shows the physical view of the system. The client and the server communicates with each other over HTTP. Both of them also communicates with the university authentication system over HTTP. The server comunnicates with the database over SQL.

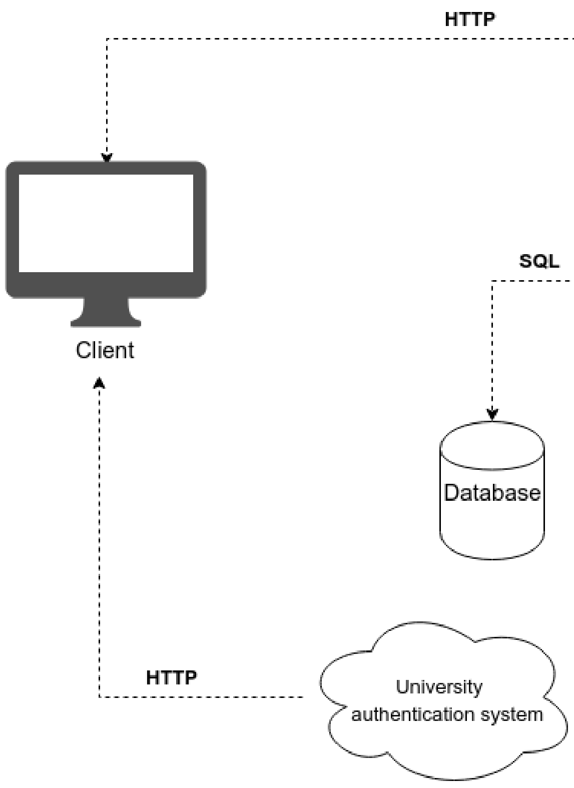


Figure 4.6: Physical view

Chapter 5

Implementation

This chapter presents how the architecture described in the previous chapter was implemented. The implementation of the software prototype is split into two separate applications - a backend application and a frontend application. The frontend application is the part that users interact with directly. Backend is the server-side application, and is concerned with storing and processing data. The backend is based on Python[1], and the frontend is written in JavaScript[2].

5.1 Backend

5.1.1 Database

The system uses PostgreSQL[3], a relational SQL database. Figure 5.1 shows an ER diagram of the schema used in the final prototype. Because of the rapid development of the prototype, it has throughout the project been attempted to keep the data as normalized and decoupled as possible. This has made the models easy to edit and extend.

[1]https://www.python.org/ (as of April 2, 2017)
[2]https://www.javascript.com (as of April 2, 2017)
[3]https://www.postgresql.org/ (as of May 22, 2017)

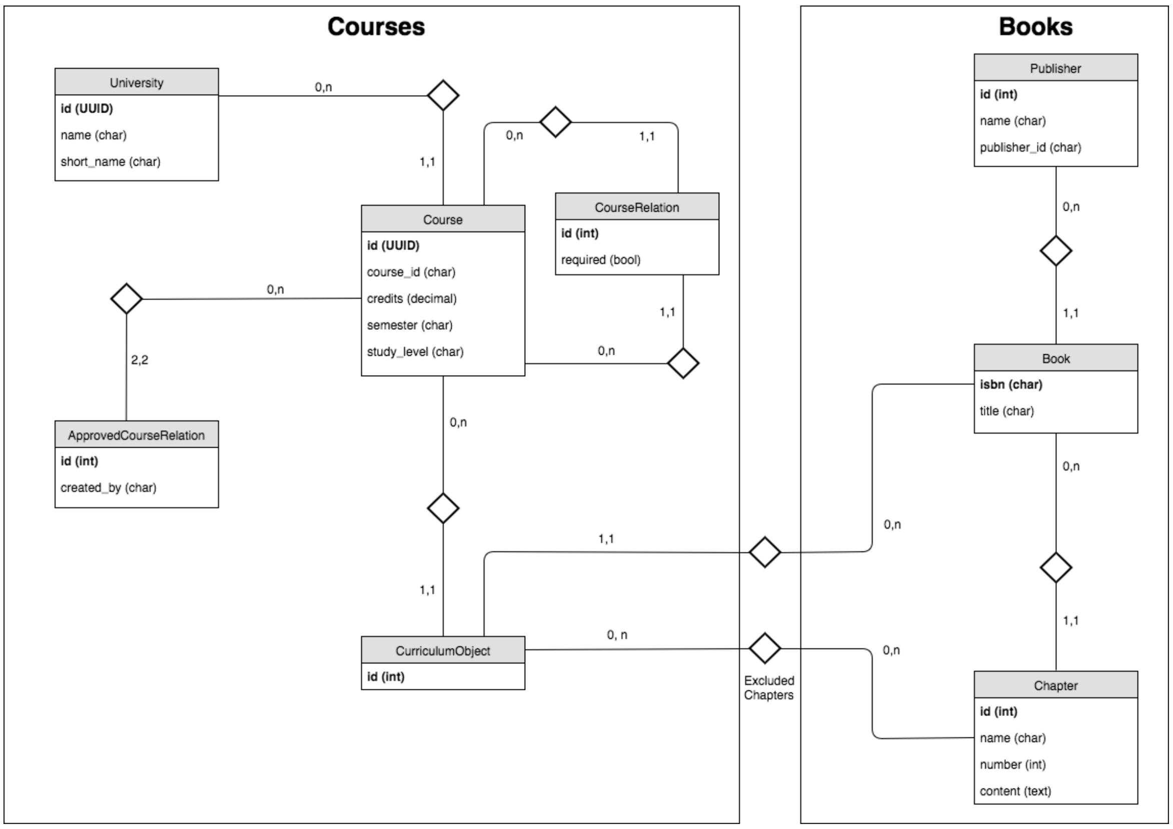


Figure 5.1: ER Diagram of final prototype

The models are split into two applications (see Section 5.1.2 on the Django framework), where the *Courses* application is seen as the core application. Relationships between applications (e.g. foreign keys) are designed to only go one way. This entails that no application can have both incoming and outgoing relations to another application.

The core model within the course application is the *Course* model. It has an UUID [*RFC 4122* 2005] as its primary key. Additionally, it has a composite key [Elmasri and Navathe 2016] consisting of *university* and *course_id*, where *course_id* is the ID that the course is referred to in the course catalogue. The course model also contains some optional metadata fields, prefixed with *"meta_"*. These fields are omitted from the ER diagram for brevity, but can be found in the source code of the prototype.

5.1.2 Django

Django[4] is an open-source Python web framework that was chosen for its maturity, high modularity and heavy focus on reusability. It is built on the MVC pattern, and splits its modules into what is referred to as *applications*.

The project consists of three applications:

- Courses - contains everything related to courses and universities
- Books - contains everything related to curriculum and text analysis
- API - is the foundation for the API, connecting it to the views of the other applications

Django has many features that allows for rapid development of a prototype. This section will briefly discuss the most important features that are relevant for this project.

Django ORM and migrations

Django comes with its own built in Object-Relational Mapping (ORM) [O'Neil 2008]. This allows the developer to perform database operations in Python, without having to think about the underlying SQL. Additionally, Django has support for automatic database migrations, making it easy to change the database schema.

Admin panel

In order for the Django ORM to work, a model of the schema must be specified. An example of such a model can be seen in Listing 1.

[4]https://www.djangoproject.com (as of April 5, 2017)

```python
class University(models.Model):
    name = models.CharField(max_length=50)
    short_name = models.CharField(max_length=10, unique=True)
```

Listing 1: Example model

Based on this model, Django can automatically generate an admin panel where users can list all objects, create new objects, and edit or delete existing objects. This is done by adding the following lines:

```python
@admin.register(University)
class UniversityAdmin(admin.ModelAdmin):
    list_display = ('name', 'short_name',)
```

Listing 2: Register example model in admin

The automatically generated admin panel proved incredibly useful throughout the implementation, because data manipulation and retrieval could be done without having to interact with the database through code or SQL statements.

Management commands and shell

The system created in this project relies on its content in order to be considered useful by its users. This content is largely decentralized today, scattered throughout different spreadsheets and wiki solutions at different faculties and departments. It must therefore be easy to add large amounts of data through scripts (Section 4.3.2). Django allows developers to create custom scripts that can be run through the command line interface (CLI). These scripts have access to the same environment as the actual server, making it easy to create web scrapers that collects data from other sources, such as the IME Course API [*IME API* 2017].

5.1.3 API

Django REST Framework[5] is a Python package for building RESTful APIs [Richardson and Ruby 2008] in Django.

The package allows for rapid prototyping of APIs, by using the Django Model as a foundation for endpoint generation. An example can be seen in Figure 3:

```
class UniversitySerializer(serializers.ModelSerializer):
    class Meta:
        model = University
        fields = ('id', 'name', 'short_name', )
```

Listing 3: Example Serializer

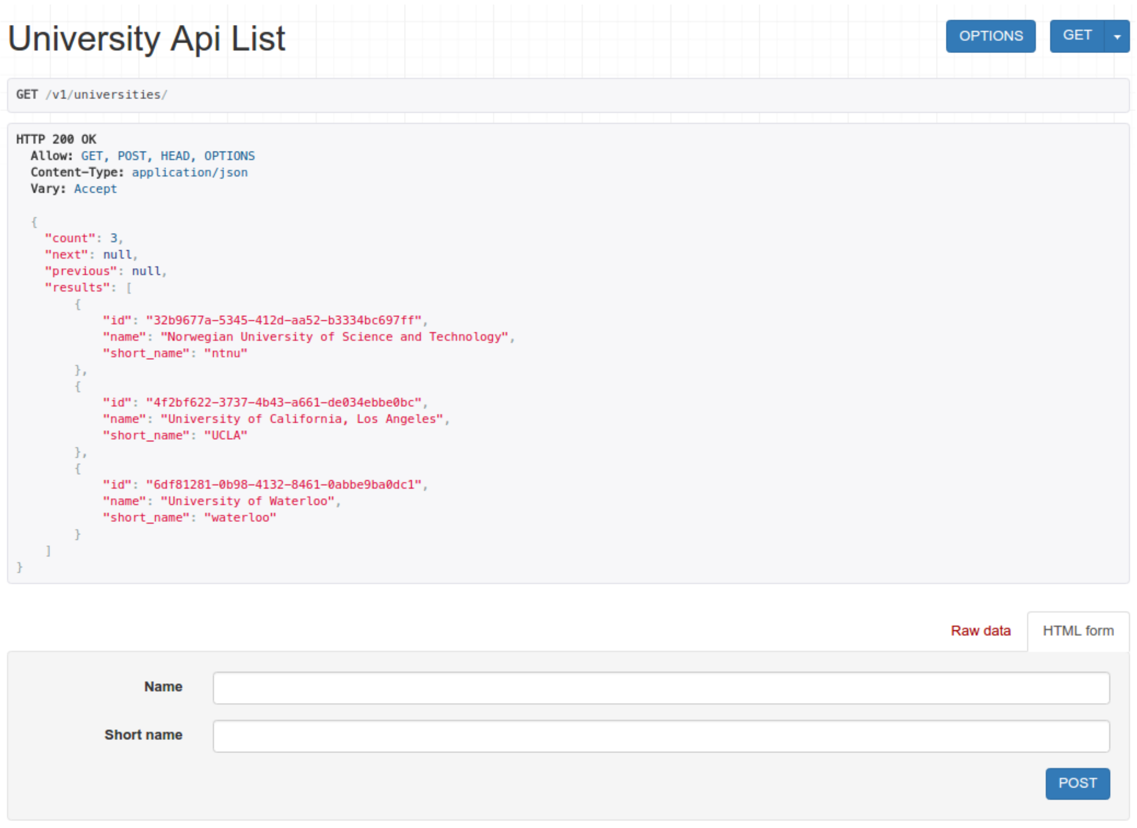
The viewset in Figure 4 automatically generates API endpoints for creation, deletion, editing and retrieval of model instances. The serializer specifies which fields to display, and how those fields are validated. All data is serialized and deserialized as JSON[6]. The framework also supports permission handling, as can be seen in the same viewset.

[5]http://www.django-rest-framework.org/ (as of April 3, 2017)
[6]http://www.json.org/ (as of April 3, 2017)

```python
class UniversityAPIViewSet(viewsets.ModelViewSet):
    queryset = University.objects.all()
    serializer_class = UniversitySerializer

    def get_permissions(self):
        if self.action in ('list', 'retrieve'):
            self.permission_classes = []
        else:
            self.permission_classes = [IsAuthenticated,
 ↪  AdminAccessPermission]
        return super(UniversityAPIViewSet, self).get_permissions()

    def get_object(self):
        queryset = self.get_queryset()
        pk = self.kwargs[self.lookup_field]
        try:
            uuid.UUID(pk)
            obj = get_object_or_404(queryset, pk=pk)
        except ValueError:
            obj = get_object_or_404(queryset, short_name=pk)

        self.check_object_permissions(self.request, obj)
        return obj
```

Listing 4: Example Viewset

Documentation of a viewset is automatically generated, and is browsable through any of its own endpoints. A screenshot from the documentation can be seen in Figure 5.2.



```
GET /v1/universities/
```

```
HTTP 200 OK
  Allow: GET, POST, HEAD, OPTIONS
  Content-Type: application/json
  Vary: Accept

  {
    "count": 3,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": "32b9677a-5345-412d-aa52-b3334bc697ff",
            "name": "Norwegian University of Science and Technology",
            "short_name": "ntnu"
        },
        {
            "id": "4f2bf622-3737-4b43-a661-de034ebbe0bc",
            "name": "University of California, Los Angeles",
            "short_name": "UCLA"
        },
        {
            "id": "6df81281-0b98-4132-8461-0abbe9ba0dc1",
            "name": "University of Waterloo",
            "short_name": "waterloo"
        }
    ]
}
```

Name	
Short name	

Figure 5.2: Screenshot of API

5.2 Frontend

5.2.1 React

React[7] is a JavaScript frontend library for building user interfaces, developed by Facebook. It was chosen because of its active community, and component based philosophy.

The component based structure made it easy to create modular, reusable components, something that allowed for rapid prototyping. Because of the active community, common components are available in support libraries. An example of this

[7]https://facebook.github.io/react/ (as of April 3, 2017)

is React-Bootstrap[8], a wrapper for the Twitter Bootstrap framework[9]. Bootstrap comes with components used to build the Graphical User Interface (GUI) of the application, such as a CSS grid (i.e. rows and columns), tables, buttons and modals. This is useful for fast prototyping.

5.2.2 React Boilerplate

In order to quickly get started creating the prototype, React Boilerplate[10] was chosen as a foundation for the code base. React Boilerplate takes care of all necessary configuration, letting the developer quickly get started with the development. Part of the boilerplate is Webpack[11], a module bundler that minifies the code, transpiles it from JSX[12] to ES5[13], and outputs a small set of files that can be deployed to a server as static files. The minification allows the code do be downloaded to the client faster. The transpilation allows the development to be done in JSX, a useful JavaScript abstraction when working with React.

5.2.3 Redux and Saga

Redux[14] is a state container for JavaScript frontend applications that eases the task of creating systems that behave consistently. This entails managing the global state of the application, allowing components to share state. Together with Saga[15], a library for handling side effects and assist with asynchronous behaviour, it handles the global state and data flow throughout the application.

The project is separated into components and containers. *Components* are simple reusable components with no global state, while containers are more advanced com-

[8]https://react-bootstrap.github.io (as of May 11, 2017)
[9]http://getbootstrap.com/ (as of May 11, 2017)
[10]https://github.com/react-boilerplate/react-boilerplate (as of May 11, 2017)
[11]https://webpack.github.io (as of April 3, 2017)
[12]https://facebook.github.io/react/docs/introducing-jsx.html (as of April 3, 2017)
[13]http://www.ecma-international.org/memento/TC39.htm (as of April 3, 2017)
[14]http://redux.js.org (as of April 3, 2017)
[15]https://redux-saga.js.org/ (as of May 22, 2017)

ponents connected to the Redux store. Each container consists, in addition to the actual component, of the following files:

- **actions.js:** Actions are events that when dispatched, triggers a state change. An example of an action is *LOAD_COURSES*, which changes the state of the application, so that it starts loading courses from the API. All actions relevant to the container resides in this file.
- **reducers.js:** While actions describe an event that happens, the reducer describes how the state changes when a given action is triggered. For example, if the *LOAD_COURSES* action is triggered, *loading_courses* should be set to *true*. This file keeps track of state changes triggered by the container's actions.
- **sagas.js:** Sagas are responsible for all side effects and asynchronous behaviour within the container. An example of a saga, is one that watches when the *LOAD_COURSES* action is triggered (side effect), and starts the API call that actually loads the courses. All sagas relevant for the container resides in this file.
- **constants.js:** All actions have a *type* attribute which uniquely identifies them and which they are referred to by, e.g. *LOAD_COURSES*. To avoid hard coding these references, they are specified as constants. All such constants are specified in this file.
- **selectors.js:** Selectors makes it easier to load data from the Redux data store. A component that triggers the *LOAD_COURSES* action, might want to subscribe to the current value of *loading_courses* and *courses* (the actual courses) in the data store. Instead of selecting each of them manually, the selector creates an abstraction layer that returns a set of values in the data store which are relevant to each other. In this case *loading_courses* and *courses*. All selectors that are relevant of the container resides in this file.

5.3 Client-server relationship

This section describes how the client-server architecture proposed in Chapter 4 was implemented in the prototype. It will first describe how authentication has been implemented, before summarizing how the frontend and backend interacts with each other.

5.3.1 Authentication

To ensure that only authorized users are able to manipulate data in the database, the prototype incorporated Dataporten[16] as the authentication system. Dataporten is a OAuth 2.0[17] service created by Uninett[18]. By default, the service allows developers to configure a login for anyone with an educational account in Norway, but its architecture is designed to also support other institutions. This entails that a university in e.g. South Africa can start using Dataporten as a login service, by integrating with Dataporten as a login provider. Uninett is also working on integrating Dataporten with eduGAIN[19], a service for authentication and authorization for educational federations internationally.

5.3.2 Communication

The frontend and the backend are integrated with Dataporten in two separate ways. The backend is registered as an API Gatekeeper[20], which means that it is registered as a data source available to Dataporten clients[21]. The frontend is registered as a client.

Figure 5.3 illustrates how the frontend and the backend communicates with each other. This is what happens when an authenticated user creates a new course in the

[16]https://docs.dataporten.no (as of May 23, 2017)
[17]https://oauth.net (as of May 23, 2017)
[18]https://www.uninett.no (as of May 23, 2017)
[19]https://www.geant.org/Services/Trust_identity_and_security/eduGAIN (as of May 24, 2017)
[20]https://docs.dataporten.no/docs/apigatekeeper (as of May 24, 2017)
[21]https://docs.dataporten.no/docs/gettingstarted (as of May 24, 2017)

frontend:

1. The request to the API is triggered by a Saga.
2. The frontend does not communicate directly with the API, but uses a proxy provided by Dataporten. This proxy strips the OAuth token of the client, and provides the API with a temporary token on behalf of the user. The proxy can also be configured to only allow authenticated requests.
3. When the request reaches the backend, Apache handles the request and passes it to the API through WSGI[22].
4. Django runs the request through all of its middleware, including a custom Dataporten middleware. This middleware collects the token provided by the proxy, and makes a request to the Dataporten Groups API using this token[23]. This API returns all groups that the user is part of, i.e. what university the user belongs to, and which faculties and departments it is part of.
5. If the user belongs to the admin group predefined in the Django configuration file, the requests proceeds, and the course is created.
6. Django returns a Response to Apache via WSGI, which in turn sends that response through the proxy to the client. The asynchronous Saga resolves, the the GUI in the client reflects the changes made on the server.

[22]http://wsgi.readthedocs.io/en/latest/what.html (as of May 24, 2017)
[23]https://docs.dataporten.no/docs/groups/ (as of May 24, 2017)

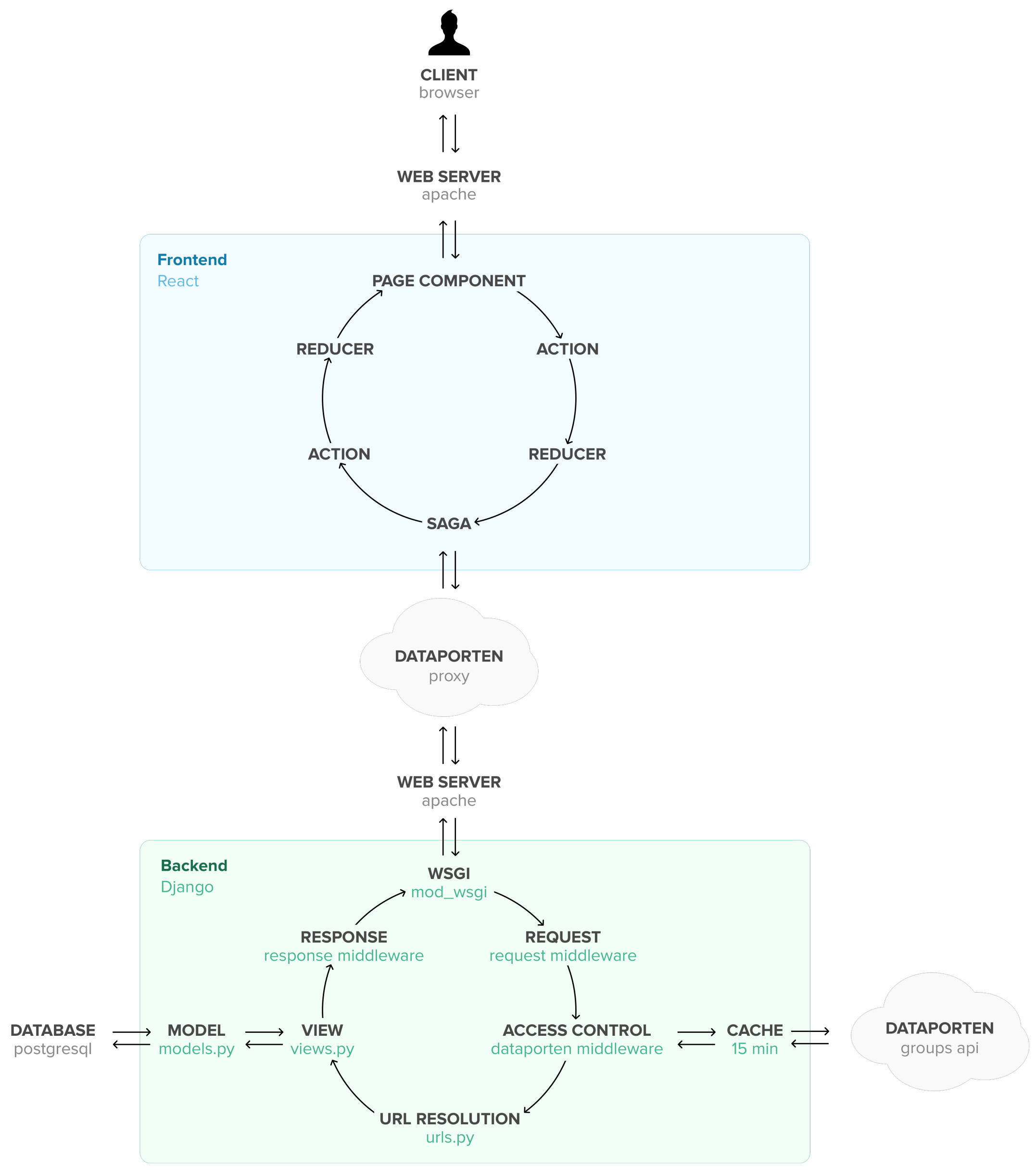


Figure 5.3: System communication

5.4 Data collection

This section describes the process of acquiring a data foundation for the application. The project relies on having course data from at least two different universities in order to fully test all the functionality in the prototype. In addition to using NTNU as an example university, the University of Waterloo[24] will be used as a proof of concept to test the course link and comparison functionality. Both universities offers well-documented APIs, which allows for easy data collection.

5.4.1 Dataset from NTNU

The Faculty of Information Technology and Electrical Engineering (IE) at NTNU offers an API with course data [*IME API* 2017]. In order to structure the data the way we wanted (see ER diagram in Figure 5.1), a web parser was created to fetch data from the API. An overview of the parser can be seen in Figure 5.4. The parser works by fetching one course from the API at a time. For each course, it saves the required data, like course code, course description, credits etc. The course metadata also contains the fields *required previous knowledge* and *recommended previous knowledge*. Unfortunately, these fields does not simply contain a list of course codes, but an unstructured text. When encountering these fields, the parser uses a regular expression (regex) to extract course codes from the unstructured text. A *CourseRelation* is created to represent a dependency link between the courses.

Course codes at NTNU uses the format *[2-4] characters* + *[2-6] numbers*. This regex can be replaced when using the parser on other APIs.

[24]https://uwaterloo.ca (as of May 18, 2017)

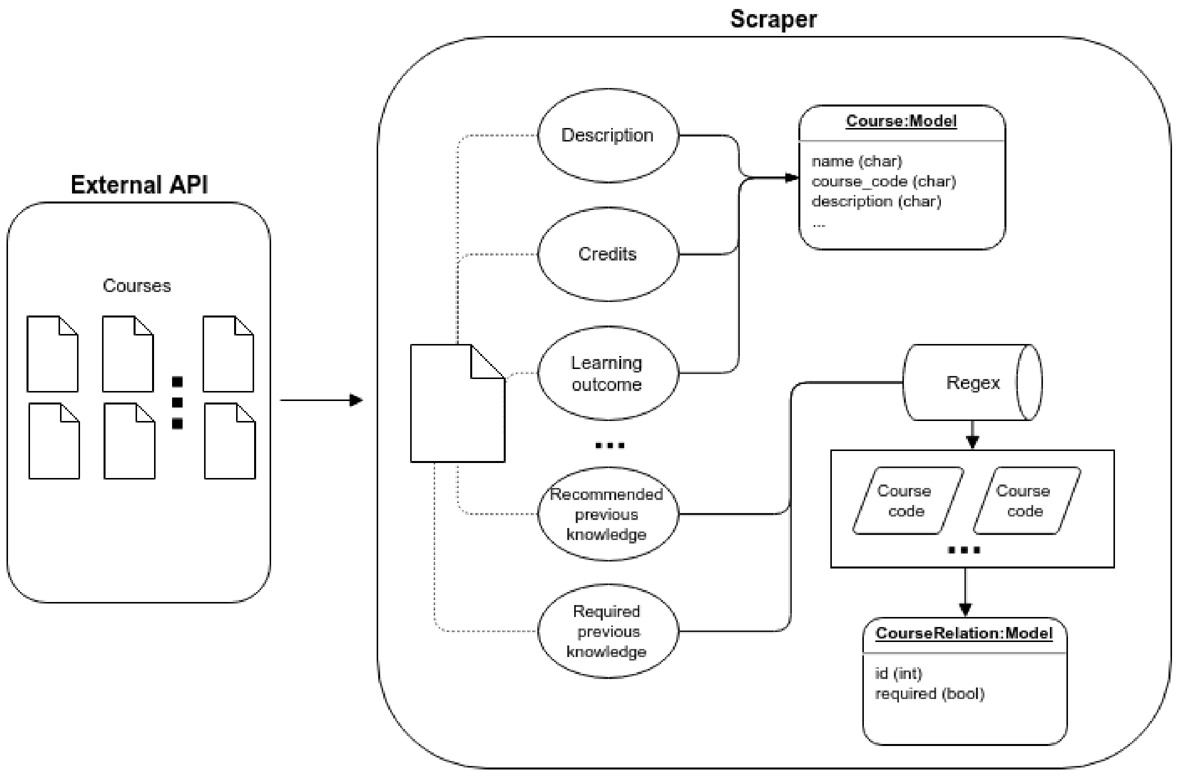


Figure 5.4: Parser process

5.4.2 Dataset from another university

University of Waterloo offers an API with all the required course metadata [*The University of Waterloo Open Data API* 2017]. A modification of the NTNU parser were used to gather the data. Only the regex and some of the field names were changed.

5.5 Curriculum

Functional requirement *FR 3* states that the system should support comparison of courses based on metadata. Web scrapers for getting data about textbooks were created in order to meet this functional requirement. Textbooks from the publishers O’Reilly and Person were supported in the prototype, as a proof of concept. The process of curriculum comparison is detailed in Section 5.7.

5.6 Course dependency tree

5.6.1 Dependency graph

Several different methods were considered when constructing the relationship that describes how courses depends on each other, for example how a student needs to take an introductory course to algorithms before he takes an advanced algorithms course. The conclusion from this research was to implement this relationship as a Directed Acyclic Graph (DAG) [Cormen et al. 2009].

A DAG was chosen because of its main attributes:

- The direction of the relationship matters (course A is required in order to take course B)
- A parent course can never be a child of any of its descendants, as this would create an endless loop of pre-requisite courses. The relationships must therefore be acyclic (if course A is required in order to take course B, course B cannot be required in order to take course A)

5.6.2 Graph visualization

PlantUML[25] was chosen to visually represent course dependencies. This is a language that allows users to create UML diagrams. The backend generates a course dependency tree by fetching it from the database, building the courses and their relationships with PlantUML on the fly. When the PlantUML text string is completed, the string is encoded and returned to the client as a link to the PlantUML API. This API decodes the string, and generates the graph as an SVG.

An example of a PlantUML graph, can be seen in Figure 5.5.

[25]http://www.plantuml.com (as of May 24, 2017)

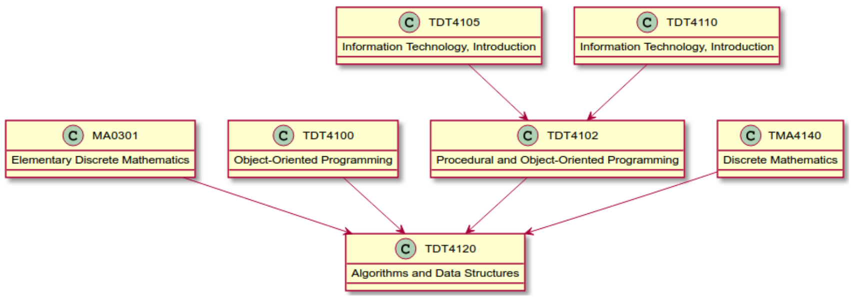

TDT4105
Information Technology, Introduction
TDT4110
Information Technology, Introduction
MA0301
Elementary Discrete Mathematics
TDT4100
Object-Oriented Programming
TDT4102
Procedural and Object-Oriented Programming
TMA4140
Discrete Mathematics
TDT4120
Algorithms and Data Structures

Figure 5.5: Dependency visualization for TDT4120

5.7 Course comparison and similarity analysis

This section describes the implementation of the functionality for finding similar courses and comparing courses. This includes preprocessing course metadata, applying TF-IDF weighting and using the vector space model to finally calculate similarity. The theory behind the information retrieval concepts implemented here, are described in detail in Section 3.5.

Scikit-learn is an open-source collection of tools for data mining, data analysis and machine learning[26]. It provides state-of-the-art Python implementations of many widely used machine learning algorithms. These are available through an easy-to-use interface [Pedregosa et al. 2011]. In this project, Scikit-learn has been used to compare courses, and to search for similar courses given a course as input. This have been achieved by using the framework's text preprocessing technology, and Term Frequency-Inverse Document Frequency (TF-IDF) implementation [*Scikit-learn: Feature extraction documentation* 2016].

Natural Language Toolkit (NLTK) is a collection of libraries for working with human language[27]. The platform is written in Python, and contains tools for natural language processing. NLTK's library for stemming has been used in this project [Bird, Klein, and Loper 2009]. More details about this is presented in this Section 5.9.

[26]http://scikit-learn.org (as of May 11, 2017)
[27]http://www.nltk.org (as of May 11, 2017)

5.7.1 Stop word removal

Scikit-learn filters out words from a predefined list of stop words. The list contains 309 very common words, like *a*, *at*, *the* and *for*, and words that do not carry any meaning[28].

5.7.2 Stemming

The implementation features usage of the Porter algorithm. The Porter algorithm is a well-established suffix removal algorithm [Baeza-Yates and Ribeiro-Neto 2011]. The algorithm applies a series or rules to the words based on a suffix list. Figure 5.6 shows a few examples of suffix removal steps applied by the algorithm.

$$s \rightarrow \phi$$

$$sses \rightarrow ss$$

$$fulness \rightarrow ful$$

Figure 5.6: A sample of the steps in the Porter suffix-removal algorithm

The algorithm checks if the input word ends with the suffix on the left side of the equation. By going through its entire list of possible suffixes and performing the appropriate replacements, the finished terms are all in their base form. The longest sequence of letters that matches the suffix is used.

[28]http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words (as of May 11, 2017)

$$cats \rightarrow cat$$

$$stresses \rightarrow stress$$

$$hopefulness \rightarrow hopeful$$

Figure 5.7: Example of words before and after being processed by the Porter algorithm

In the implementation, the PorterStemmer module from NLTK was used. Figure 5.8 shows the course description for TDT4120 Algorithms and Data Structures. This course description will be used to illustrate the document preprocessing steps.

> Methods for analyzing the efficiency of algorithms, divide and conquer techniques, recursive solution methods. Methods for ordering, searching and sorting. Data structures for efficient retrieval of data, dynamic programming and greedy algorithms. Data structures for implementing graphs and networks, as well as methods for traversals and searches. Algorithms for finding the best path(s) and matchings, spanning trees and maximum flow. Theory of problem complexity. Algorithms are expressed in a language independent fashion.

Figure 5.8: TDT4120 course description

From this it is possible to choose a number of descriptive terms that well represents what is taught in the course: *algorithms*, *efficiency* and *methods* to name a few. Ideally, any course that contains these index terms should be a good match. However, if another course has a description containing the index terms *algorithm* and *efficient*, no match would be made for these terms. This is where a stemmer is useful. Applying the Porter algorithm to the description gives the result shown in Figure 5.9. Stop word removal has also been applied in this example.

```
method analys effici algorithm divid conquer techniqu recurs solut method method order search sort data structur effici retriev data dynam program greedi algorithm data structur implement graph network method travers search algorithm find best path match span tree maximum flow theori problem complex algorithm express languag independ fashion
```

Figure 5.9: Document after stop word removal and stemming

5.7.3 Index term selection

Scikit-learn was also used in the index term selection stage of text preprocessing. By supplying the $max\_features = n$ argument to the class used for TF-IDF computation, only the top n terms (sorted by TF-IDF value) will be considered in the comparison. These n terms will effectively act as index terms.

Using $max\_features = 10$ selects the terms shown in Figure 5.10. Experiments were made to decide what value to use for n. A set of courses with confirmed similarity (credit reduction) were compared. The results showed that setting $max\_features = None$ yielded the most consistently good results. *None* means that all features should be considered.

```
algorithm data effici method network np problem search structur techniqu
```

Figure 5.10: Document after index term selection

5.7.4 TF-IDF and cosine similarity calculation

The TF-IDF computation is done by the *TfidfVectorizer* class. Listing 5 shows an extract of the implementation for finding similar courses.

Line 9 passes in the entire document collection, and results in a weighted document-term matrix. The matrix is used in line 12 to calculate the cosine of the angle between the base course and the other courses. A one-dimensional matrix is the result of this calculation, where the *n-th* element is the similarity of the *base course* and the *n-th* course. In this example, a rank of the top 20 similar courses are retrieved, as seen at line 15.

```
1   # Initialize
2   vect = TfidfVectorizer(
3       stop_words='english',
4       analyzer='word',
5       max_features=20,
6   )
7
8   # Learn vocabulary and idf. Return term-document matrix
9   tfidf = vect.fit_transform(all_courses_content)
10
11  # Do vector space model calculations
12  result_vector = (tfidf * tfidf.T).A[base_course_index]
13
14  # Get indexes of top n results
15  top_results_indexes = sorted(range(len(result_vector)),
16                               key=lambda i: result_vector[i])[-n+1:]
```

Listing 5: TF-IDF computation using Scikit-learn

The process of preprocessing the document collection and computing TF-IDF, is expensive and time consuming. Django's caching module has therefore been used actively to increase performance and decrease server load.

Chapter 6

Results

This chapter contains the results yielded from the implementation and testing of the prototype. Firstly, it goes through the list of functional and quality requirements, describing the status of each. Secondly, it will describe the performance and results of the course similarity algorithm. Lastly, it will summarize the findings derived from the final round of usability testing and interviews.

6.1 Requirement results

This section gives an overview of which functional and architectural requirements were covered, and which were not implemented. The requirements that were not covered will be discussed in detail.

6.1.1 Functional requirements

Figure 6.1 gives an overview of the status of each functional requirement after the implementation of the prototype. Two requirements were not implemented, and one requirement requires further elaboration:

FR	**Implemented**	**Comment**
FR1	Partially	1.4 was not implemented due to time constraints
FR2	Partially	2.2 was not implemented in the frontend. Super admins can however edit universities via Django Admin
FR3	Yes	-
FR4	Yes	-
FR5	Yes*	See details below
FR6	Yes	-
FR7	Yes	-

Table 6.1: Functional requirements covered in prototype implementation

FR1.4: Functionality for adding notes to courses was not added due to time constraints. This is still a requested feature from several student counselors, and something that should be implemented if the actual system is developed.

FR2.2: Allowing admins to edit universities was left out of the prototype due to time constraints. As noted in the table, it is still possible to edit universities through the Django admin panel.

FR5.1: It can be argued that this requirement would be better suited as a quality requirement, as it is possible to support this requirement to a certain degree (e.g. 70% of all universities). As mentioned in Section 5.3.1, Dataporten was chosen when implementing this requirement. This solution solved the problem for NTNU, and supports a large part of the universities word wide through eduGAIN[1]. Because eduGAIN is open to other federations (i.e. universities) joining, this requirement is still regarded as successfully implemented.

[1]https://technical.edugain.org/status.php (May 27, 2017)

6.1.2 Quality requirements

Only half of the quality requirements established in Section 4.3.2 are regarded as successful. Figure 6.2 illustrates the status of reach requirement:

QR	Success	Comment
U1	Yes	-
U2	Yes	-
M1	No	Not tested
M2	Yes	-
M3	No	Not tested
I3	Yes	This was tested when implementing the frontend of the prototype

Table 6.2: Functional requirements covered in prototype implementation

M1 was not tested due to time constraints. It is worth noting that data from two different APIs were loaded into the database through scripts during implementation of the prototype. On both occasions, this was done within the given response measure of the requirement.

M3 was not tested due to time constraints. During the development of the prototype, parts of the algorithm was changed on several occasions. Changing the entire algorithm was never needed.

6.2 Course similarity results

This section will present how the course comparison functionality of the system was tested, as well as results from these tests. In order to evaluate how accurate the course similarity suggestions are, lists of courses that have already been approved by a student counselor will be used. The lists used in this section are available from the wiki page on exchange, maintained by student counselors at IDI[2]. The wiki page contains lists of course pairs, where one is a NTNU course and the other is an exchange course. This will serve as a baseline for evaluating the performance of the course similarity functionality.

Table 6.3 shows a set of courses from NTNU Gjøvik (left column), and their approved counterpart at NTNU (middle column). Only courses with sufficient data foundation were included in the tests. The right column shows the similarity suggestion made by the comparison algorithm. The comparison does not include curriculum. The testing yielded mostly good results, with five out of seven comparisons suggesting over 50% similarity. Based on the results from this test, we assume that 50% similarity is a good threshold for assuming that two courses are similar.

To attempt to explain why the bottom two comparisons yielded a low percentage suggestion, their metadata was manually evaluated. It was discovered that the courses IMT3441 and TDT4145 have little in common based on their course description and learning goals. TDT4145 is a course with focus on database theory, while IMT3441 is more of a "hands-on" practical course. The similarity suggestion of 38.69% might be reasonable. The same can be said for IMT4751 and TTM4135, which have considerable differences. It is worth noting that both these course pairs are entered into the wiki article with a credit reduction.

[2]https://www.ntnu.no/wiki/pages/viewpage.action?pageId=78024468 (as of May 25, 2017)

Course A	**Course B**	**Suggested similarity**
IMT2243 Software Engineering	TDT4140 Software Engineering	82.53%
IMT2282 Operating Systems	TDT4186 Operating Systems	75.30%
IMT1082 Objectoriented Programming	TDT4102 Procedural and Object-Oriented Programming	67.73%
IMT2021 Algorithmic Methods	TDT4120 Algorithms and Data Structures	61.11%
IMT2521 Network Administration	TTM4100 Communication, Services, Network	53.00%
IMT3441 Database and application administration	TDT4145 Database Management Systems	38.69%
IMT4751 Wireless communication security	TTM4135 Information Security	26.18%

Table 6.3: NTNU Gjøvik comparison results

Table 6.4 shows the results from testing the course comparison functionality on courses from University of California. Course descriptions, learning goals and any available metadata were manually entered into the system. Again, the results were satisfactory, with the lowest similarity suggestion at 49.55% and an average of approximately 62%.

Course A	**Course B**	**Suggested similarity**
ECON 173A Financial Markets	TIØ4145 Corporate Finance	74.94%
ECON 106 Microeconomics	TIØ4117 Microeconomics	66.74%
CS 420.1 Fundamentals of Information System Security	TTM4135 Information Security	65.44%
MAE 119 Introduction to Renewable Energy	TEP4175 Design of a Wind Turbine	56.10%
MGMT 497.610 Developing a Business Plan	TIØ4250 Entrepreneurship	49.55%

Table 6.4: University of California comparison results

As mentioned in Chapter 5, the system has functionality for using curriculum as a component in the comparison. However, finding the curriculum a course is using is not trivial, and is often not available for non-students. Testing of this functionality was therefore limited. Three course pairs where both courses had available curriculum lists were found. Table of contents of the books were used in the testing. The results shows that the comparison actually returns a lower similarity score without curriculum, than it does with.

Course A	**Course B**	**Without curriculum**	**With curriculum**
CS420.1 Fundamentals of Information System Security	TTM4135 Information Security	65.40%	49.51%
CS 151C Design of Digital Systems	TDT4240 Software Architecture	50.72%	30.77%
INFR08014 Object-Oriented Programming	TDT4100 Object-Oriented Programming	51.59%	50.89%

Table 6.5: Comparison with and without curriculum

To verify that the comparison functionality does not report any false positives, a number of tests on courses that are *not* similar were ran. The courses were selected by going over their curriculum and available metadata, and verifying that they have little or nothing in common. Table 6.6 shows the results of the tests. At most, a suggested similarity of 4.3% were reported. The results were satisfactory, and no false positives were given.

Course A	**Course B**	**Suggested similarity**
TEP4100 Fluid Mechanics	TTT4145 Radio Communications	4.30%
AE302010 Business economics	TDT4120 Algorithms and Data Structures	3.12%
TKP4110 Chemical Reaction Engineering	TDT4240 Software Architecture	2.57%
IB204112 Road construction	ANT1101 Ancient History	0.55%
FY1001 Mechanical Physics	PSY1014 Social Psychology I	0.48%
TMA4100 Calculus 1	AFR2850 Modern African History	0.30%
JAP0502 Japanese II	MA2401 Geometry	0.10%

Table 6.6: Non-similar courses

6.3 Evaluation of prototype

To evaluate the software prototype, interviews were set up with two student counselors and one employee from the Office of International Relations at NTNU. The first part of the interview was a usability test, where the subjects were explained the purpose of the system, and then presented with the same tasks as in the paper prototype test (see Section 4.2.2). The second part was an unstructured interview, with the following prearranged questions:

- What is your impression of the system?
- Would you use it?

- What must be done to get your colleagues to use it?

The overall impressions from the subjects were positive. They all stated that the system would require little or no improvements before being ready to use. One of the student counselors stated that the system also would be useful for students searching for courses to take, because they would be able to browse courses previously approved at the university they are researching. All subjects stated that they thought colleagues would be interested in using it. The interviews did however uncover some shortcomings of the system. Below is a list of improvements suggested by the subjects:

Course relation comments: Two of the subjects pointed out that it should be possible to comment on a course relation, in addition to commenting on courses. There is often a reasoning behind why a course link gets approved, and this is useful information for other counselors as well.

Course relation cardinality: It was pointed out that course relations are not necessarily a 1-1 relation. Often due to credits mismatch, there might not be a course at university A that can be counted as similar to a course on university B. The course relation should therefore be changed to 1-n.

More detailed course dependency tree: The current tree shows all courses that a given course depends upon. Often, a course C is only dependent upon one of a group of courses A or B. This should be incorporated into both the database and the graphical representation of the dependency tree.

More options when searching for similar courses: Both student counselors mentioned that it would be useful with a more detailed search, when searching for similar courses. Students often know which country they want to study in, so it would be useful to filter on similar courses by country.

Credits conversion: As mentioned in Section 3.3.1, NTNU has a database (GNAG) for keeping track of how many credits a semester at a foreign university is worth, in order to be able to compare them with semesters at NTNU. This allows them to compare the number of credits of a foreign course with a course at NTNU. It was suggested to add support for automatically calculating the number of NTNU credits of a given foreign course.

Integrate with the Office of International Relations' database: The last suggestion was to incorporate the system with the Office of International Relations' database over approved courses.

Chapter 7

Conclusion

7.1 Conclusion

This project has identified that there is a need for a digital tool for counseling potential exchange students, and explored what those needs are. Through interviews and a questionnaire we have collected relevant data to support this statement. Furthermore, the project has collected data in order to properly derive the requirements for such a system. Based on these requirements, a working prototype has been created.

The testing shows that the functionality for automatically determining course similarity yielded good results. Automating the process of course comparison is possible, but the accuracy of the results are not perfect. False negatives and false positives will occur occasionally. The similarity suggestions should be used as a basis for determining similarity, and not as a solution. The use of curriculum in the comparison process should be further researched.

As such we consider the project to be successful, and hope that it can be useful for student counselors in the future.

7.2 Limitations of the current solution

This section describes limitations of the prototype and the research project.

Real world usage: In order for student counselors to start using the system, all their existing data have to imported. Although the system has a well-defined interface for importing data from APIs, spreadsheets and other data sources, the amount of work that needs to be put into this transition might not be for everyone. Student counselors might feel that the overhead of getting started with the system is too much.

Prototype performance: The prototype exhibits problems with performance as the amount of data grows.

Survey breadth: A sample size of 17 respondents might not be enough to represent the entire population of student counselors. The system requirements and the resulting prototype are to a large degree based on the knowledge acquired from the respondents.

7.3 Future work

There is still work to be done before the system can be considered complete. The feedback gathered from the interviews in Section 6.3 should be formalized into additional requirements, and implemented. Additionally, the following features should be considered:

Performance improvements: Searching for courses will scale poorly as the course database grows. Indexing courses by their *course_id* and *name* should improve this. A task queue should be considered for running resource intensive tasks such as preprocessing of courses.

The algorithm that generates the course dependency tree should be rewritten to reduce its runtime. The algorithm grew in complexity during the development of the prototype, and is currently performing poorly for large trees. This is largely due to a suboptimal implementation of transitive reduction (i.e. remove transitive course dependencies), where it should be possible to reduce the runtime to $O(V(V+E))$. This should be researched and implemented.

Implement with other groups: In parallel with this project, two other projects with the same student counselors as this project have been worked on. All projects were related to each other, where one of them specialized in course comparison. That project has been conducted by Audun Liberg (NTNU). The results from the research has not yet been published, but should be studied when complete.

Usability improvements: Even though all usability requirements were successfully supported, there are still improvements that should be done to the user interface. Generally, the design should be reworked to be more consistent and user friendly than the current implementation.

”

Bibliography

NTNU merging with HiAls HiG and HiST (2016). URL: `http://www.ntnu.edu/ntnu-merges-with-university-colleges` (visited on 05/05/2016).

NTNU merge FAQ (2016). URL: `https://www.ntnu.no/fusjon/faq` (visited on 05/05/2016).

NTNU list of subjects (2016). URL: `https://www.ntnu.no/studier/emner` (visited on 05/05/2016).

Oates, Briony J (2006). *Researching Information Systems and Computing*. SAGE Publications Inc.

Gill, P. et al. (2008). “Methods of data collection in qualitative research: interviews and focus groups”. In: *British Dental Journal*, pp. 291–295.

Oppenheim, A. (2000). *Questionnaire design, interviewing and attitude measurement.* Continuum.

Wohlin, Claes et al. (2000). *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers. ISBN: 0-7923-8682-5.

Wright, Kevin B. (2005). “Researching Internet-Based Populations: Advantages and Disadvantages of Online Survey Research, Online Questionnaire Authoring Software Packages, and Web Survey Services”. In: *Journal of Computer-Mediated Communication* 10.3, pp. 00–00. ISSN: 1083-6101. DOI: `10.1111/j.`

1083-6101.2005.tb00259.x. URL: http://dx.doi.org/10.1111/j.1083-6101.2005.tb00259.x.

Snyder, C. (2003). *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*. ITPro collection. Morgan Kaufmann. ISBN: 9781558608702. URL: https://books.google.no/books?id=YgBojJsVLGMC.

ISO (2010). *ISO 9241-210:2010*. International Organization for Standardization. URL: https://www.iso.org/standard/52075.html.

Innsida (2017). *Utenlandsstudier*. URL: https://innsida.ntnu.no/utenlandsstudier (visited on 05/26/2017).

Kommunikasjonsavdelingen (2014). *NTNUs internasjonale handlingsplan*. URL: http://www.ntnu.no/documents/10137/981312606/internasjonal-handlingsplan-ntnu.pdf/ (visited on 04/11/2017).

Manning, Christopher, Prabhakar Raghavan, and Hinrich Schütze (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Baeza-Yates, Ricardo and Berthier Ribeiro-Neto (2011). *Modern Information Retrieval: The Concepts and Technology behind Search*. second. Addison-Wesley Professional.

Salton, Gerard and Christopher Buckley (1988). “Term-weighting Approaches in Automatic Text Retrieval”. In: *Inf. Process. Manage.* 24.5, pp. 513–523. ISSN: 0306-4573. URL: http://dx.doi.org/10.1016/0306-4573(88)90021-0.

Rozanski, N. and E. Woods (2011). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. second. Addison-Wesley Professional.

Bass, Len (2012). *Software Architecture in Practice*. Addison-Wesley Professional.

Kruchten, Philippe (1995). “The 4+1 View Model of Architecture”. In: *IEEE Softw.* 12.6, pp. 42–50. ISSN: 0740-7459. DOI: 10.1109/52.469759. URL: http://dx.doi.org/10.1109/52.469759.

RFC 4122 (2005). URL: https://tools.ietf.org/html/rfc4122 (visited on 03/21/2017).

Elmasri, Ramez and Shamkant B. Navathe (2016). *Fundamentals of Database Systems, Global Edition*. Global. Pearson Education Limited. ISBN: 9781292097619.

O’Neil, Elizabeth J. (2008). “Object/Relational Mapping 2008: Hibernate and the Entity Data Model (Edm)”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’08. Vancouver, Canada: ACM, pp. 1351–1356. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376773. URL: http://doi.acm.org/10.1145/1376616.1376773.

IME API (2017). URL: http://www.ime.ntnu.no/api/ (visited on 02/15/2017).

Richardson, Leonard and Sam Ruby (2008). *RESTful web services*. " O’Reilly Media, Inc."

The University of Waterloo Open Data API (2017). URL: https://uwaterloo.ca/api/ (visited on 02/20/2017).

Cormen, Thomas H. et al. (2009). *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press. ISBN: 9780262033848.

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Scikit-learn: Feature extraction documentation (2016). URL: http://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting (visited on 04/15/2017).

Bird, Steven, Ewan Klein, and Edward Loper (2009). *Natural Language Processing with Python*. 1st. O’Reilly Media, Inc. ISBN: 9780596516499.

Appendix A

Setup

This document describes how to set up the applications. The project consists of two applications; a frontend application and a backend application.

The project requires that the following system dependencies are installed:

- Python 3.5
- Node.js
- Virtualenv

A.1 Backend

1. Create and activate a Virtualenv:
   ```
   $ virtualenv -p /usr/bin/python3.5 env
   $ source env/bin/activate
   ```
2. Install dependencies:
   ```
   $ pip install -r requirements.txt
   ```
3. Copy the `dev-template.env` template file to `.env`
4. Run database migrations:
   ```
   $ python manage.py migrate
   ```

5. Populate database:
   ```
   $ python manage.py loaddata coursedump.json booksdump.json
   ```
6. Start server:
   ```
   $ python manage.py runserver
   ```
7. To create a admin superuser:
   ```
   $ python manage.py createsuperuser
   ```
   Visit the admin panel at: `http://localhost:8000/admin`

A.2 Frontend

1. Install dependencies:
   ```
   $ npm install
   ```
2. Start the client:
   ```
   $ npm start
   ```
3. The frontend application can be reached at: `http://localhost:3000`

Appendix B

API documentation

The RESTful methods provided by the API are listed here.

Resource	**Path**	**Method**	**Result**
Books	/books	GET	Returns all books
Books	/books/{id}	GET	Returns the book with the specified id
Books	/books/{id1}/compare/{id2}	GET	Returns the similarity of books with id1 and id2

Table B.1: Endpoints for Books

Resource	Path	Method	Result
Courses	/courses	GET	Returns all courses
Courses	/courses/{id}	GET	Returns the course with the specified id
Courses	/courses/{id1}/compare/{id2}	GET	Compares course with id1 and course with id2, and returns comparison data
Courses	/courses/{id}/find_similar	GET	Returns a ranked list of similar courses
Courses	/courses/{id}/find_similar/ {uni_name}	GET	Returns a ranked list of similar courses, limited to university with name uni_name
Courses	/courses/{id1}/ add_prerequisite/{id2}	POST	Add course with id2 as a prerequisite for course with id1
Courses	/courses/{id1}/ remove_prerequisite/{id2}	DELETE	Remove course with id2 as a prerequisite for course with id1
Courses	/courses/{id1}/approve/{id2}	POST	Approve course with id1 as a verified similar course as course with id2
Courses	/courses/{id1}/ remove_approve/{id2}	DELETE	Remove approval of course with id1 as a verified similar course as course with id2
Courses	/courses/search/?q={text}	GET	Search for course with name or course_code text

Table B.2: Endpoints for Courses

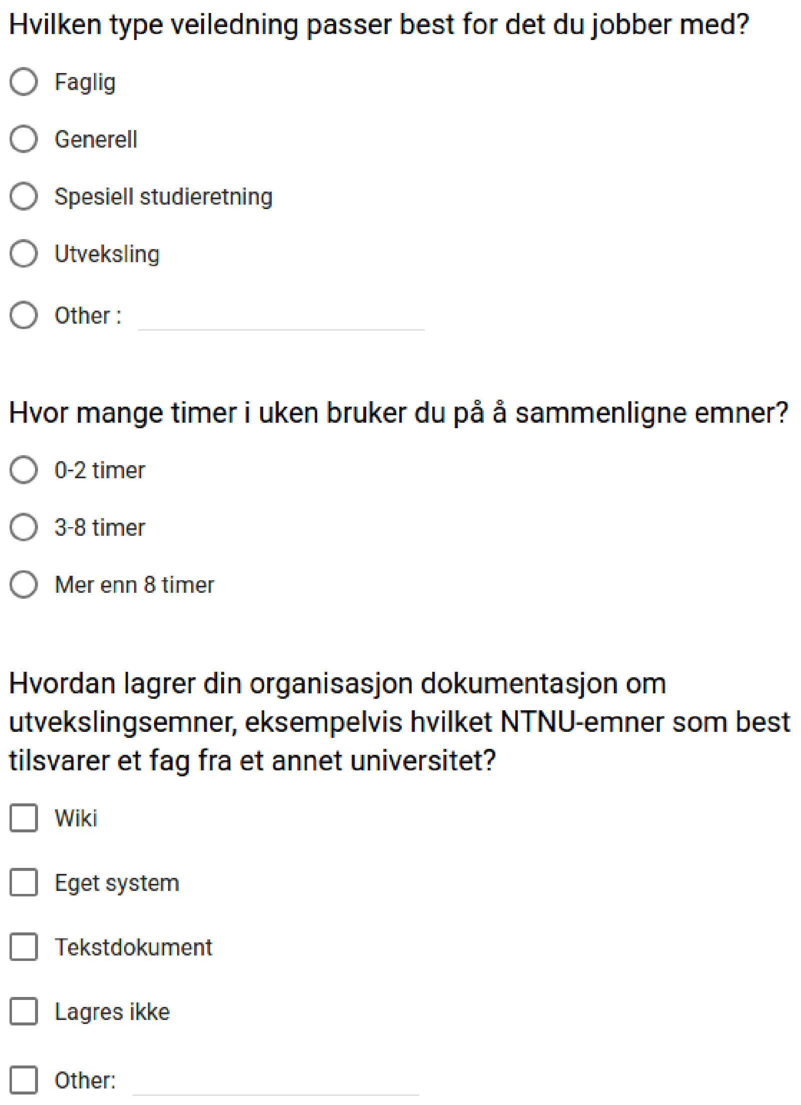
Resource	**Path**	**Method**	**Result**
Books	/books	GET	Returns all books
Books	/books/{id}	GET	Returns the book with the specified id
Books	/books/{id1}/compare/{id2}	GET	Returns the similarity of books with id1 and id2

Table B.3: Endpoints for University

Appendix C

Questionnaire

C.1 Questions



Hvilken type veiledning passer best for det du jobber med?

○ Faglig

○ Generell

○ Spesiell studieretning

○ Utveksling

○ Other : ____________

Hvor mange timer i uken bruker du på å sammenligne emner?

○ 0-2 timer

○ 3-8 timer

○ Mer enn 8 timer

Hvordan lagrer din organisasjon dokumentasjon om utvekslingsemner, eksempelvis hvilket NTNU-emner som best tilsvarer et fag fra et annet universitet?

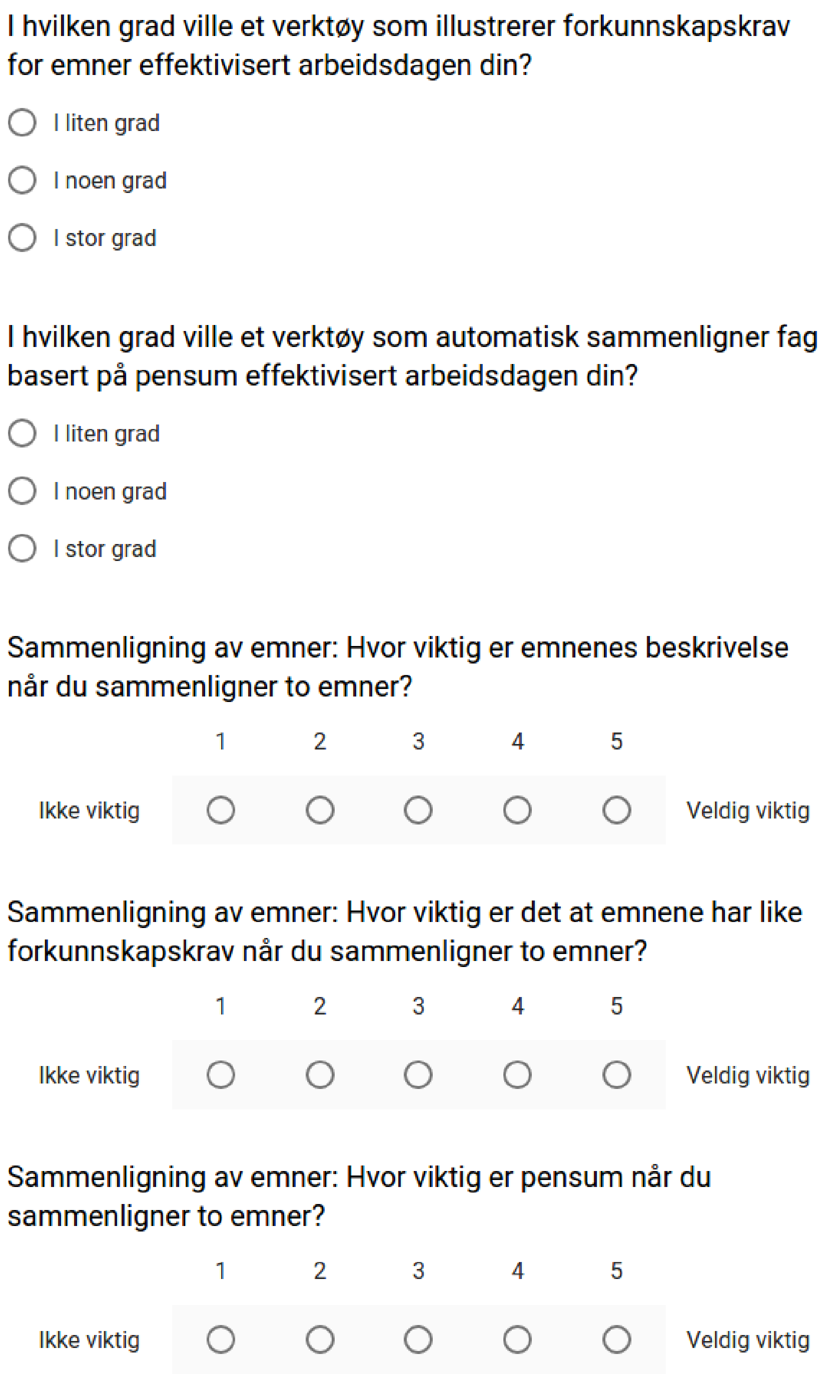
☐ Wiki

☐ Eget system

☐ Tekstdokument

☐ Lagres ikke

☐ Other: ____________



I hvilken grad ville et verktøy som illustrerer forkunnskapskrav for emner effektivisert arbeidsdagen din?

○ I liten grad

○ I noen grad

○ I stor grad

I hvilken grad ville et verktøy som automatisk sammenligner fag basert på pensum effektivisert arbeidsdagen din?

○ I liten grad

○ I noen grad

○ I stor grad

Sammenligning av emner: Hvor viktig er emnenes beskrivelse når du sammenligner to emner?

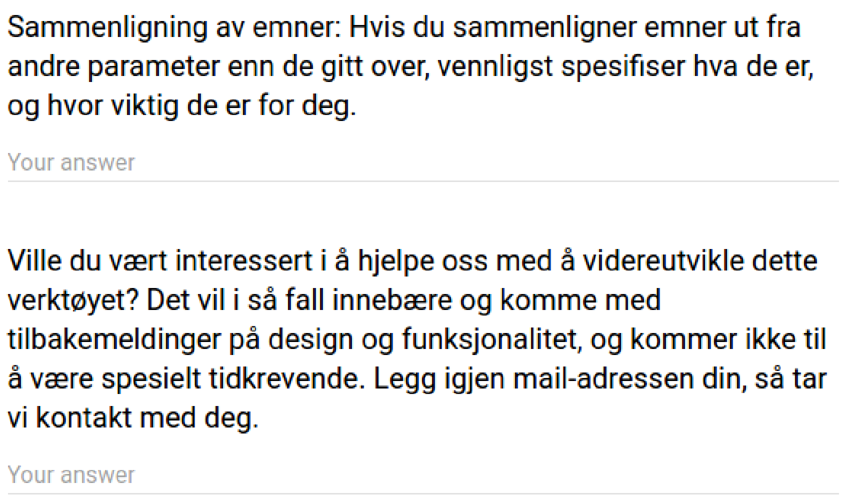
	1	2	3	4	5	
Ikke viktig	○	○	○	○	○	Veldig viktig

Sammenligning av emner: Hvor viktig er det at emnene har like forkunnskapskrav når du sammenligner to emner?

	1	2	3	4	5	
Ikke viktig	○	○	○	○	○	Veldig viktig

Sammenligning av emner: Hvor viktig er pensum når du sammenligner to emner?

	1	2	3	4	5	
Ikke viktig	○	○	○	○	○	Veldig viktig


Sammenligning av emner: Hvis du sammenligner emner ut fra andre parameter enn de gitt over, vennligst spesifiser hva de er, og hvor viktig de er for deg.

Your answer

Ville du vært interessert i å hjelpe oss med å videreutvikle dette verktøyet? Det vil i så fall innebære og komme med tilbakemeldinger på design og funksjonalitet, og kommer ikke til å være spesielt tidkrevende. Legg igjen mail-adressen din, så tar vi kontakt med deg.

Your answer

Figure C.1: The questionnaire

C.2 Results

This section contains the results from the questionnaire. 17 study counselors answered the questionnaire.

What kind of counseling best describes what you are working with?

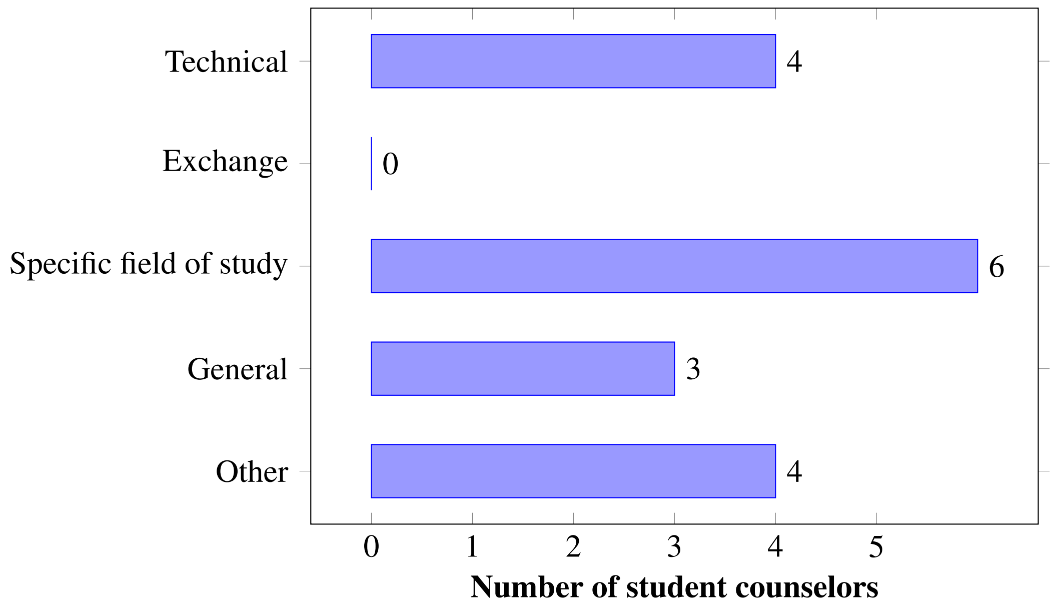


Figure C.2: Results (1)

How many hours a week do you spend comparing courses?

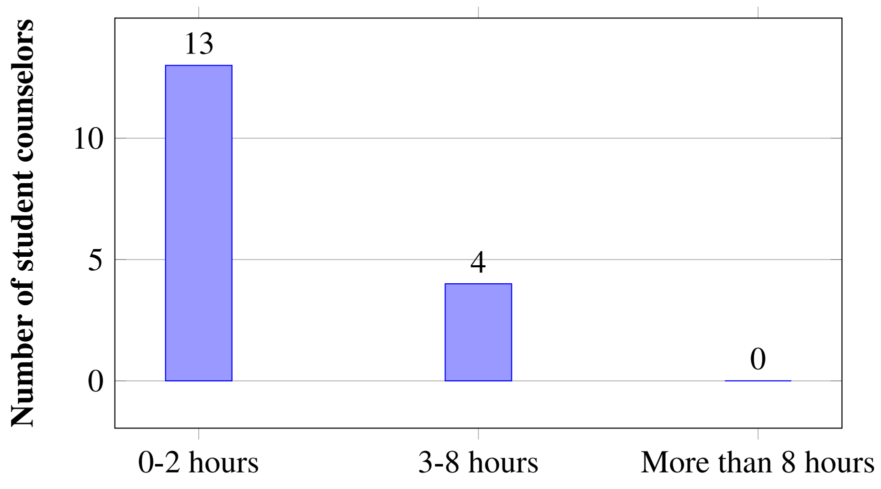


Figure C.3: Results (2)

How does your organization store documentation about subjects taken by students during exchange semesters, for example which NTNU course best correspond to a course from another university?

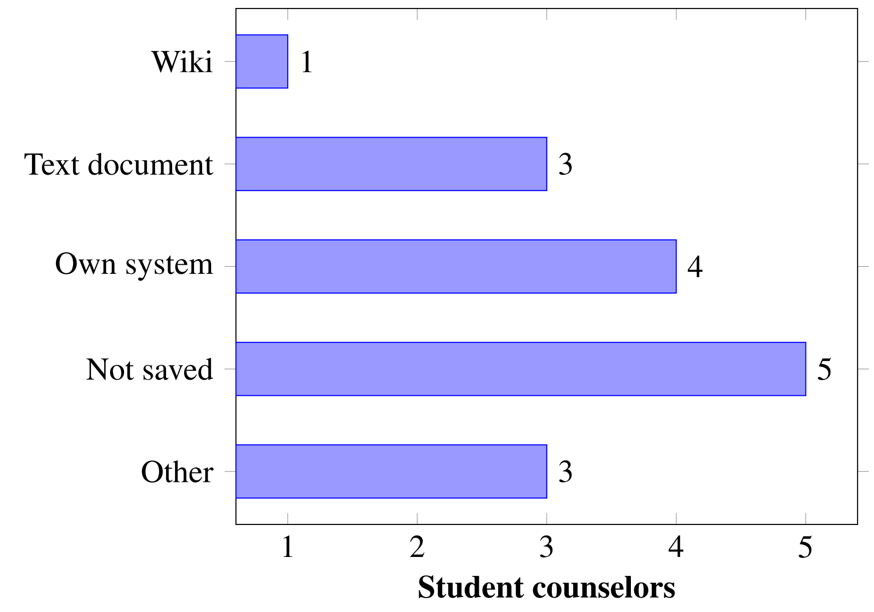


Figure C.4: Results (3)

To what degree would a tool that illustrates required prerequisite knowledge for courses make your work day more efficient?

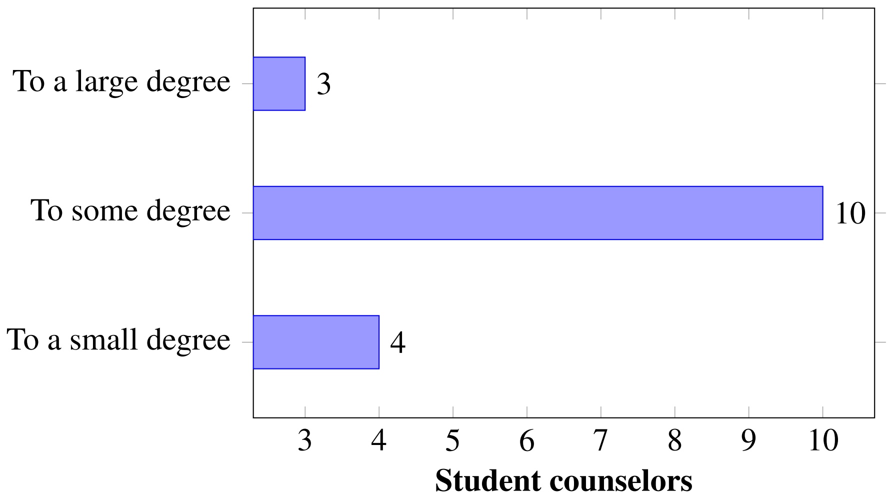


Figure C.5: Results (4)

To what degree would a tool that automatically compares courses based on their curriculum make your work day more efficient?

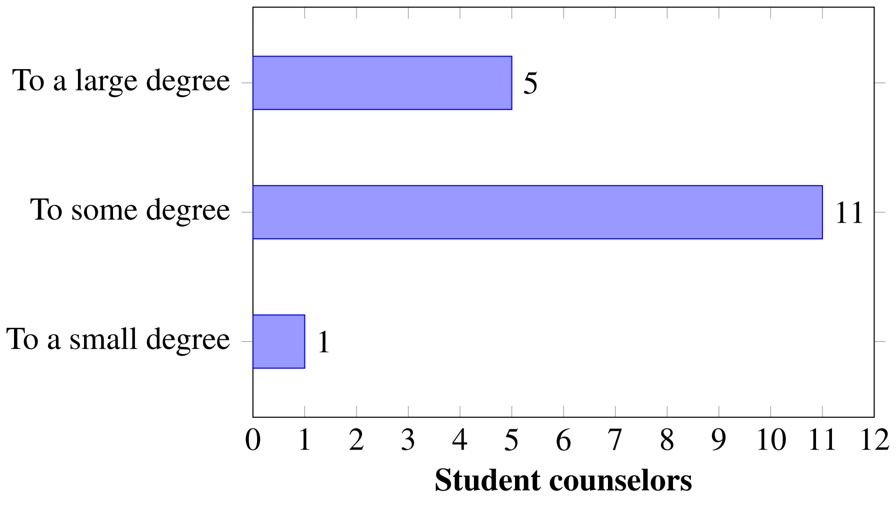


Figure C.6: Results (5)

Course comparison: On a scale from 1 to 5, how important is course descriptions when comparing two courses?

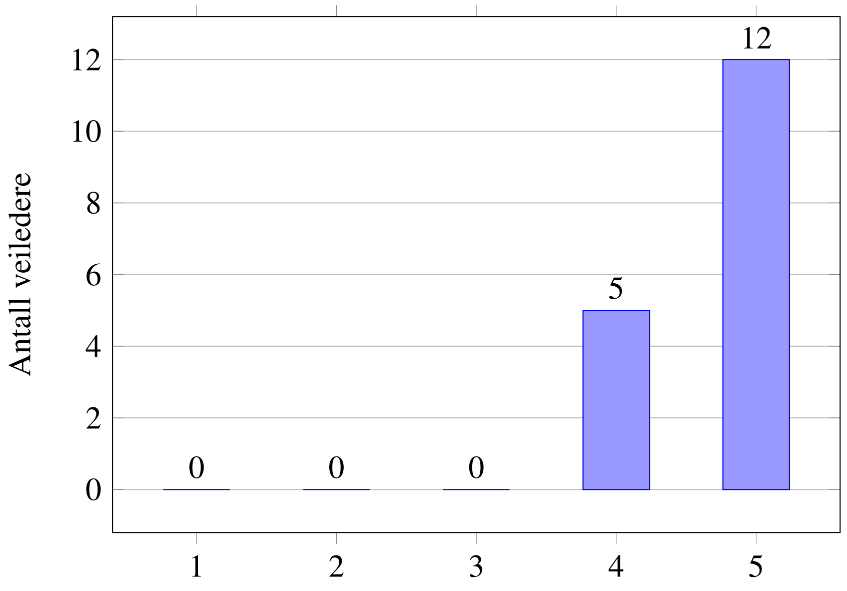


Figure C.7: Results (6)

Course comparison: On a scale from 1 to 5, how important is it that the courses have equal prerequisite requirements?

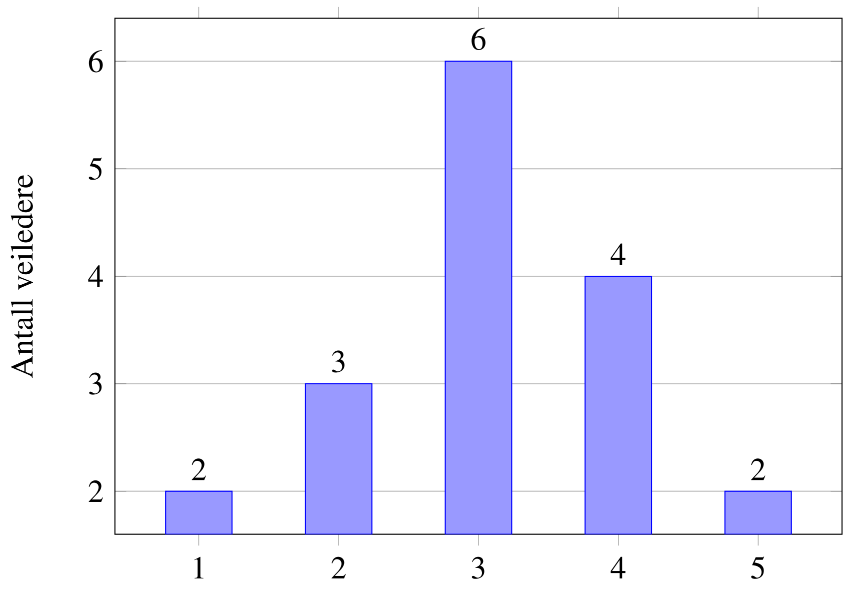


Figure C.8: Results (7)

Course comparison: On a scale from 1 to 5, how important is course curriculum when comparing two courses?

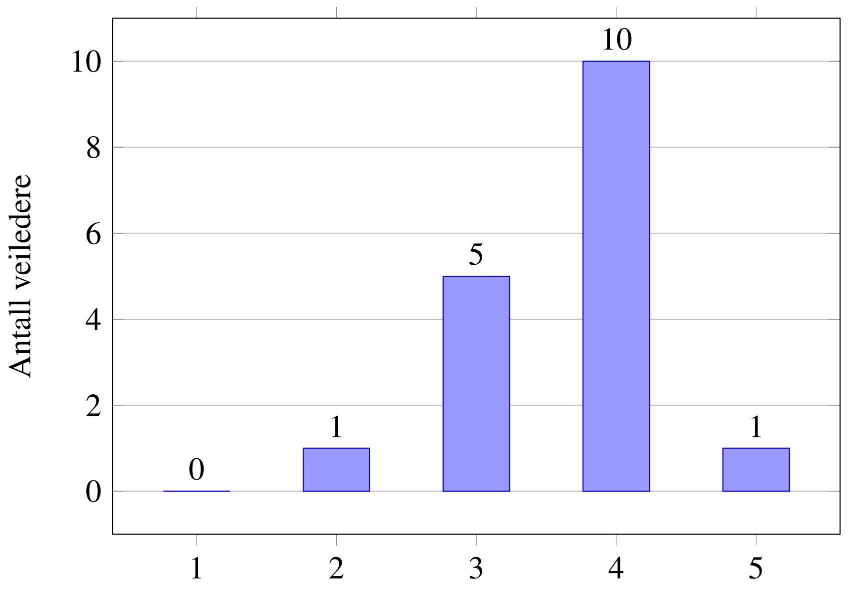
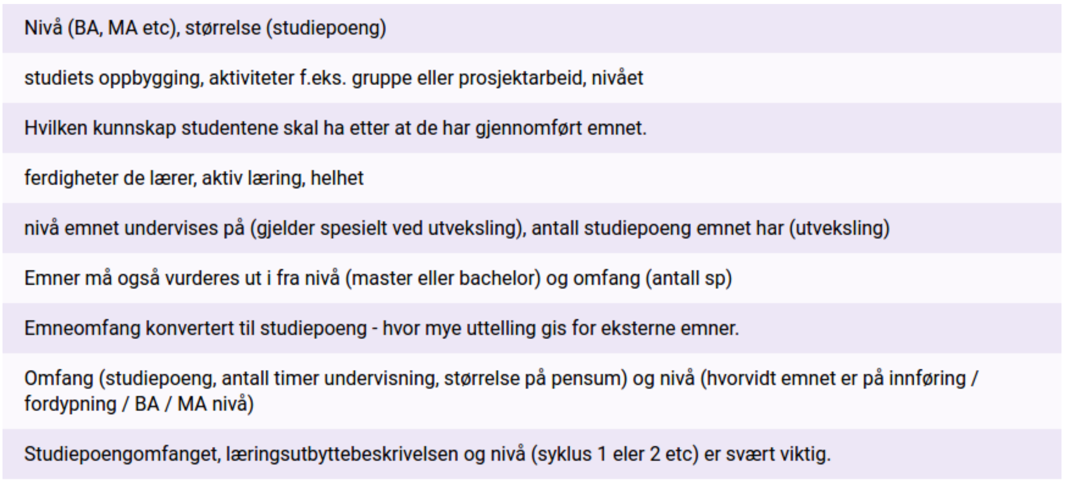


Figure C.9: Results (8)

Course comparison: If you compare courses based on other parameters than those given above, please specify what they are and how important they are.



Nivå (BA, MA etc), størrelse (studiepoeng)

studiets oppbygging, aktiviteter f.eks. gruppe eller prosjektarbeid, nivået

Hvilken kunnskap studentene skal ha etter at de har gjennomført emnet.

ferdigheter de lærer, aktiv læring, helhet

nivå emnet undervises på (gjelder spesielt ved utveksling), antall studiepoeng emnet har (utveksling)

Emner må også vurderes ut i fra nivå (master eller bachelor) og omfang (antall sp)

Emneomfang konvertert til studiepoeng - hvor mye uttelling gis for eksterne emner.

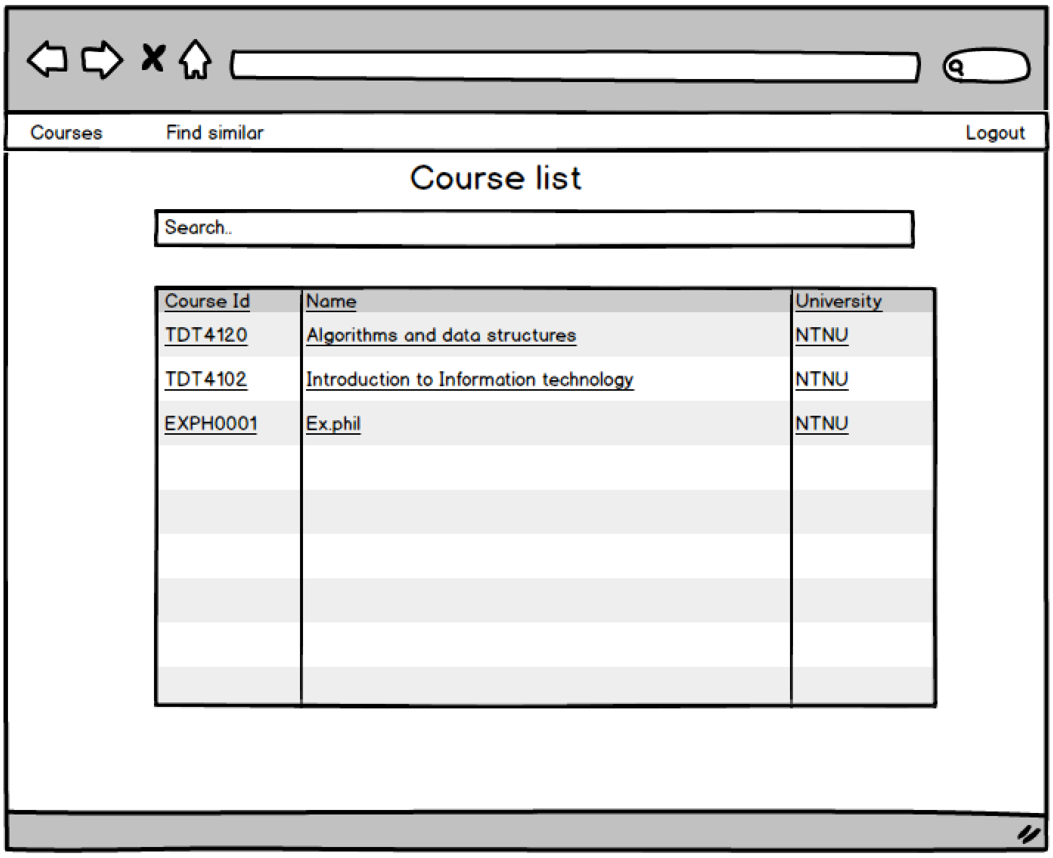
Omfang (studiepoeng, antall timer undervisning, størrelse på pensum) og nivå (hvorvidt emnet er på innføring / fordypning / BA / MA nivå)

Studiepoengomfanget, læringsutbyttebeskrivelsen og nivå (syklus 1 eler 2 etc) er svært viktig.

Figure C.10: Results (9)

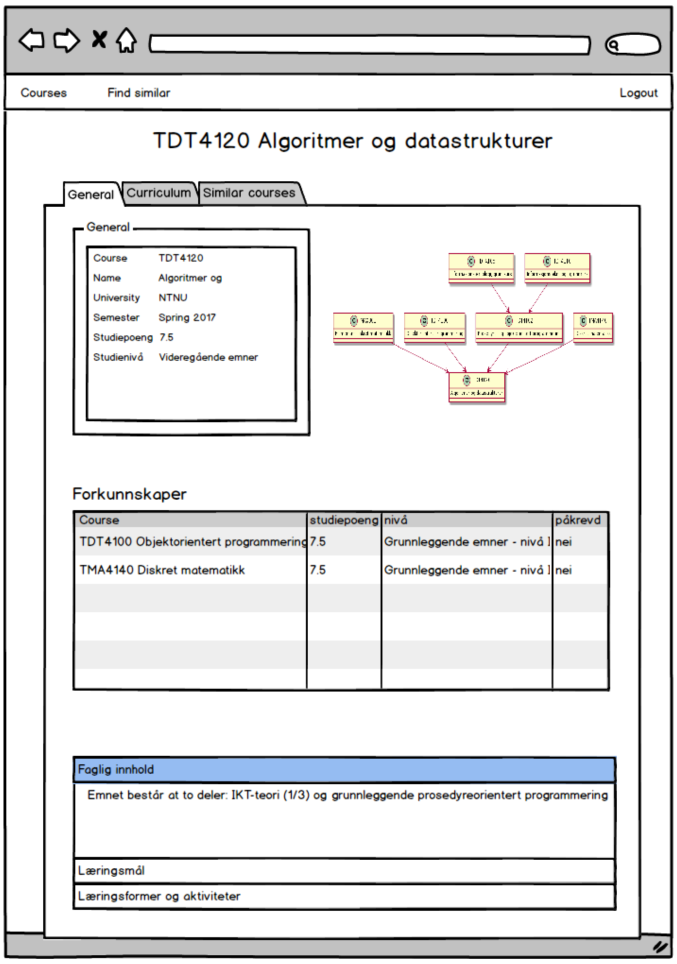
Appendix D

Paper prototype



Course Id	Name	University
TDT4120	Algorithms and data structures	NTNU
TDT4102	Introduction to Information technology	NTNU
EXPH0001	Ex.phil	NTNU

Figure D.1: Frontpage



TDT4120 Algoritmer og datastrukturer

General

Course	TDT4120
Name	Algoritmer og
University	NTNU
Semester	Spring 2017
Studiepoeng	7.5
Studienivå	Videregående emner

Forkunnskaper

Course	studiepoeng	nivå	påkrevd
TDT4100 Objektorientert programmering	7.5	Grunnleggende emner - nivå I	nei
TMA4140 Diskret matematikk	7.5	Grunnleggende emner - nivå I	nei

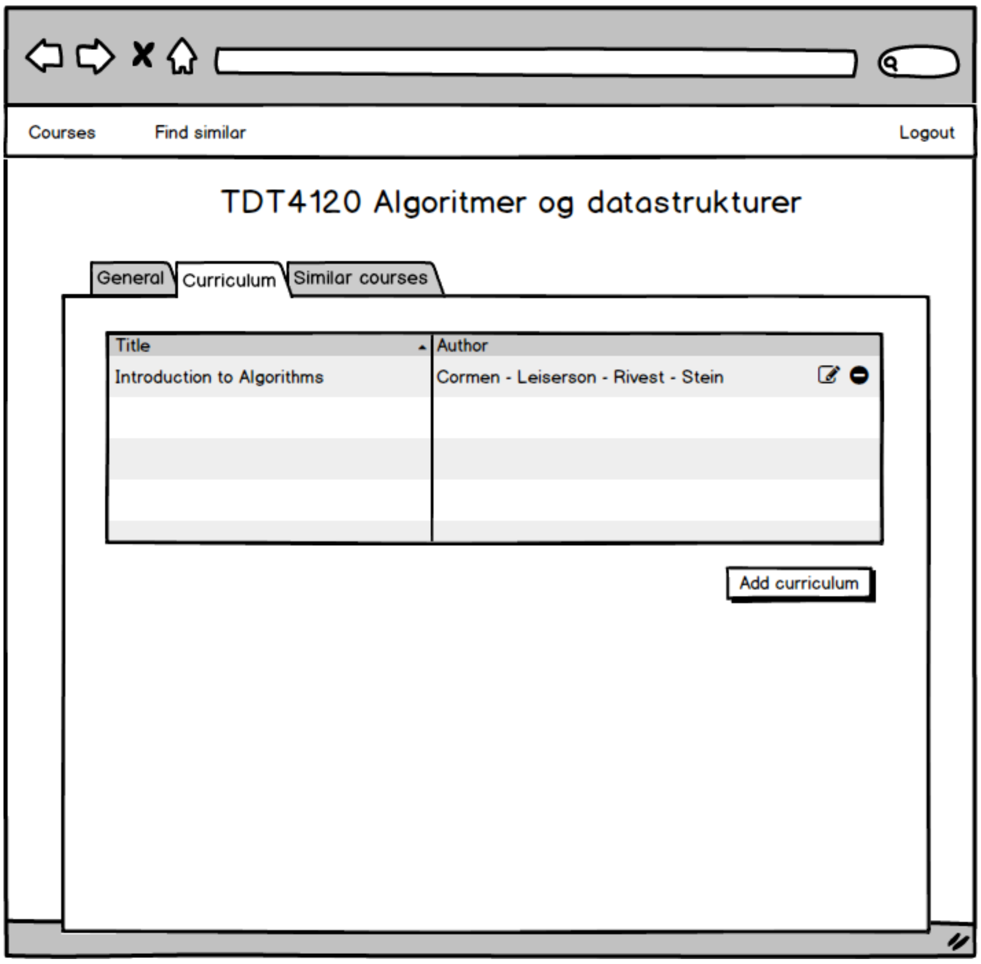
Faglig innhold

Emnet består at to deler: IKT-teori (1/3) og grunnleggende prosedyreorientert programmering

Læringsmål

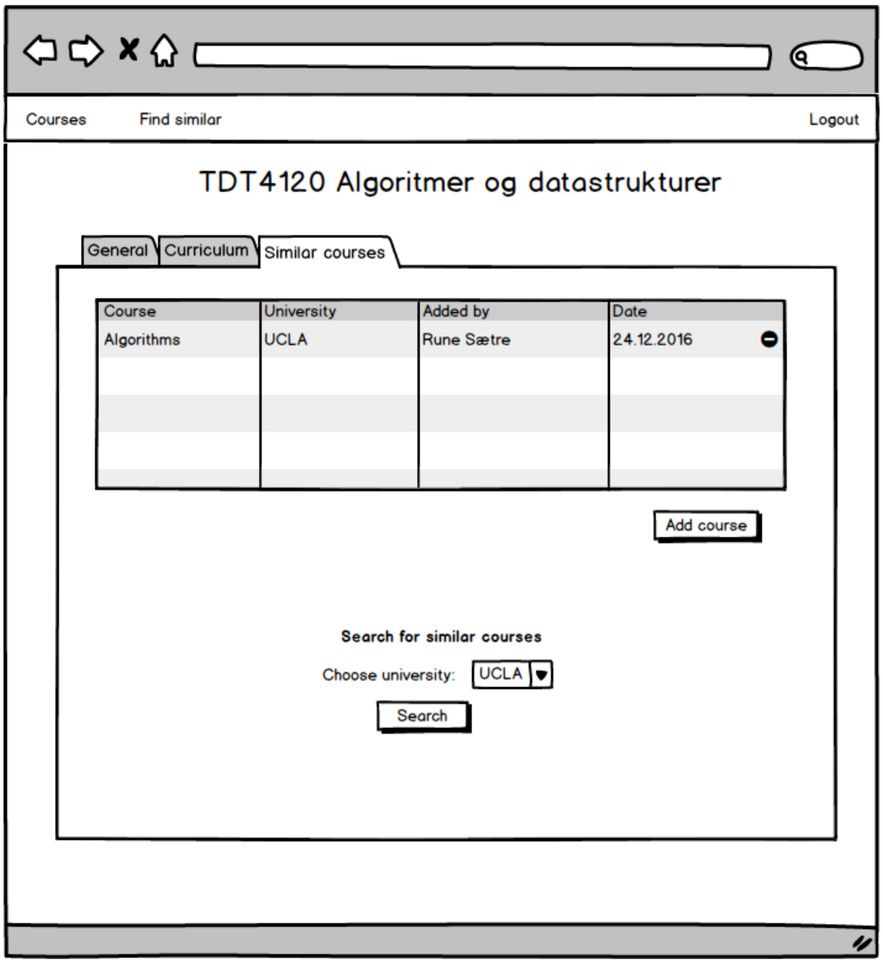
Læringsformer og aktiviteter

Figure D.2: Course details: General



Title	Author
Introduction to Algorithms	Cormen - Leiserson - Rivest - Stein

Figure D.3: Course details: Curriculum



TDT4120 Algoritmer og datastrukturer

Course	University	Added by	Date
Algorithms	UCLA	Rune Sætre	24.12.2016

Search for similar courses

Choose university: UCLA

Figure D.4: Course details: Similar courses

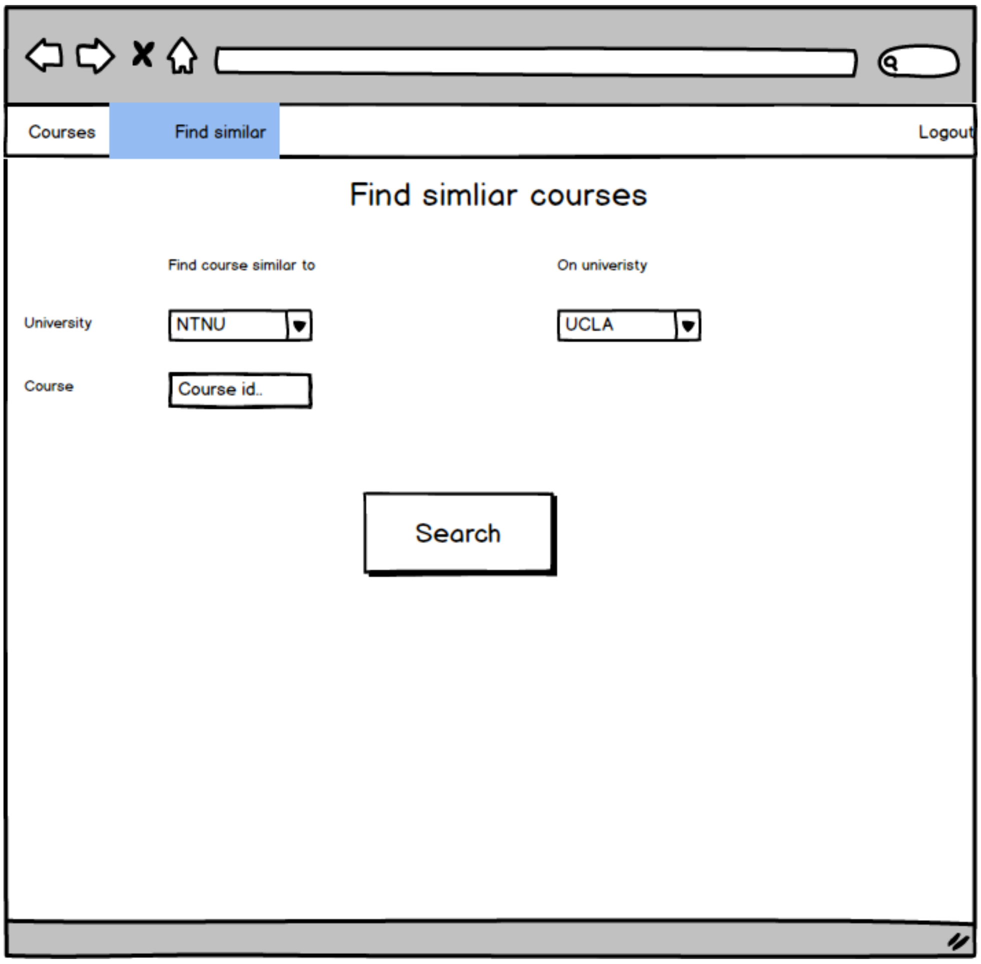
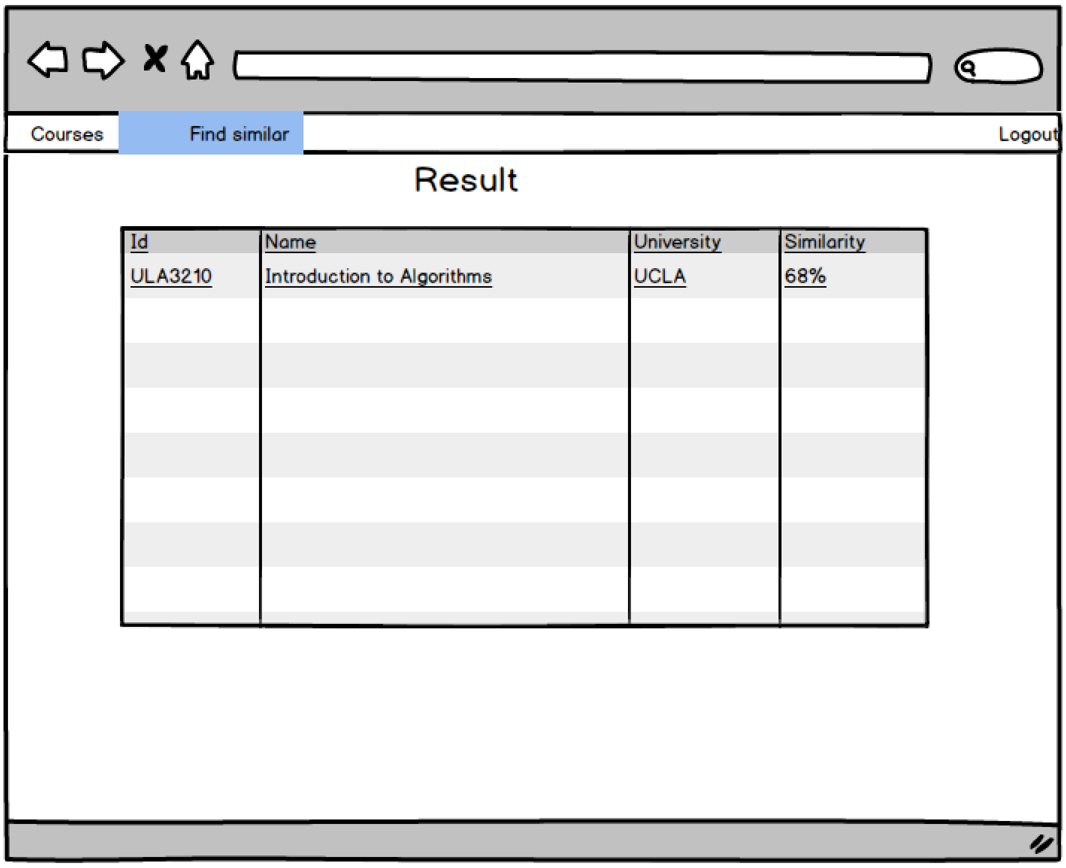


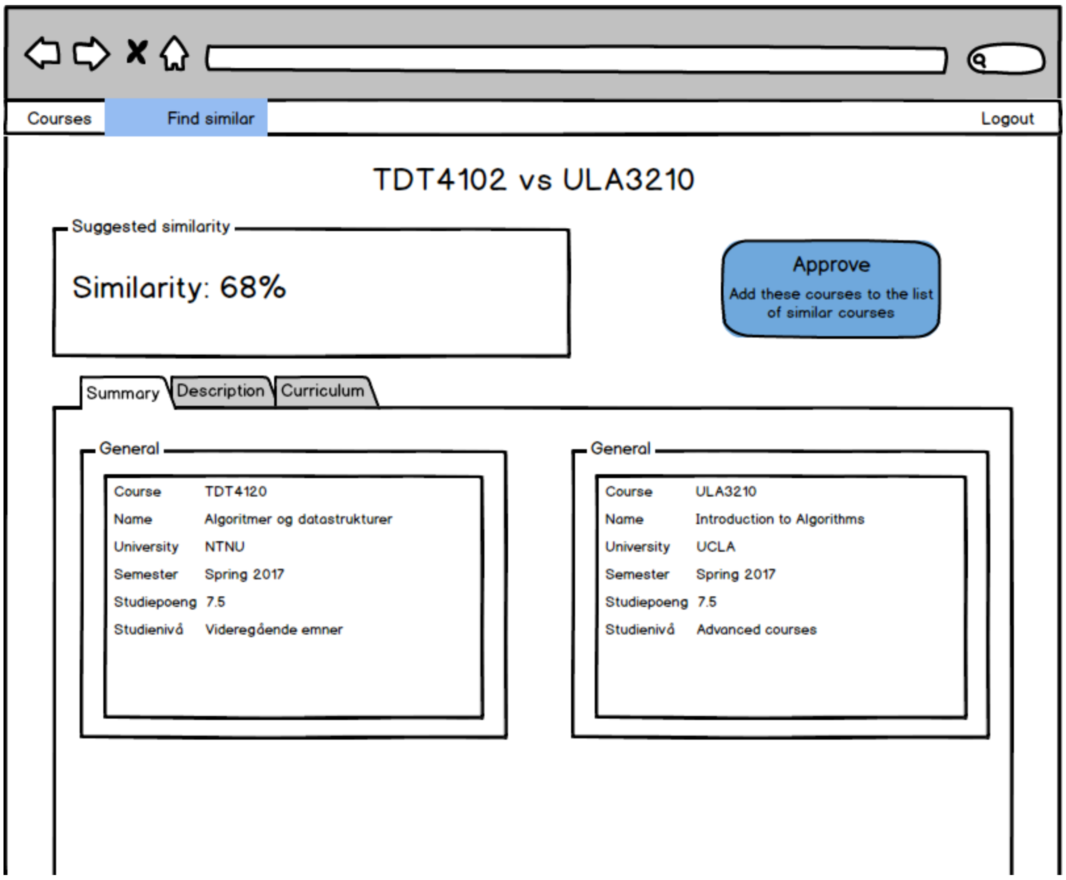
Figure D.5: Find similar courses



Result

Id	Name	University	Similarity
ULA3210	Introduction to Algorithms	UCLA	68%

Figure D.6: Find similar courses results



TDT4102 vs ULA3210

Suggested similarity

Similarity: 68%

General

Course	TDT4120
Name	Algoritmer og datastrukturer
University	NTNU
Semester	Spring 2017
Studiepoeng	7.5
Studienivå	Videregående emner

General

Course	ULA3210
Name	Introduction to Algorithms
University	UCLA
Semester	Spring 2017
Studiepoeng	7.5
Studienivå	Advanced courses

Figure D.7: Compare courses

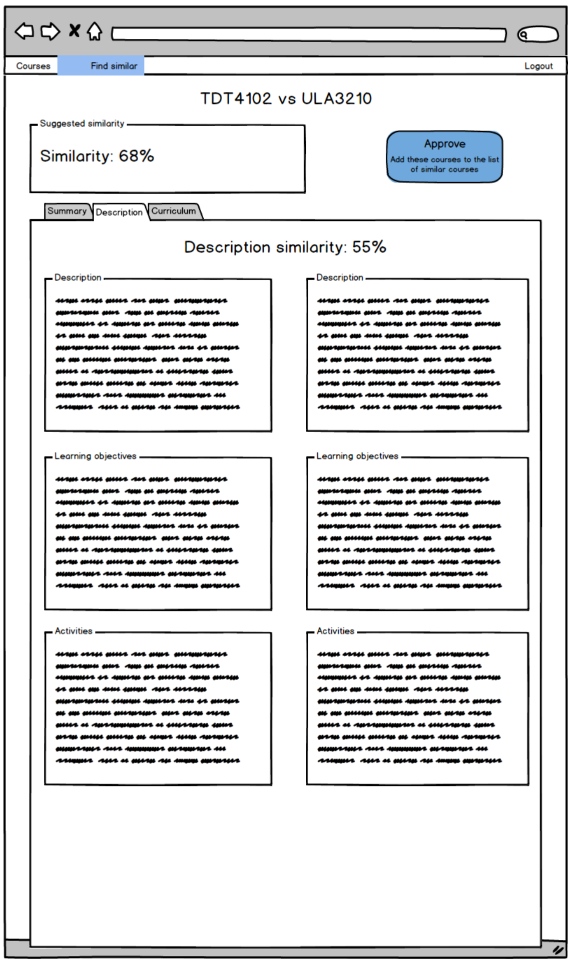
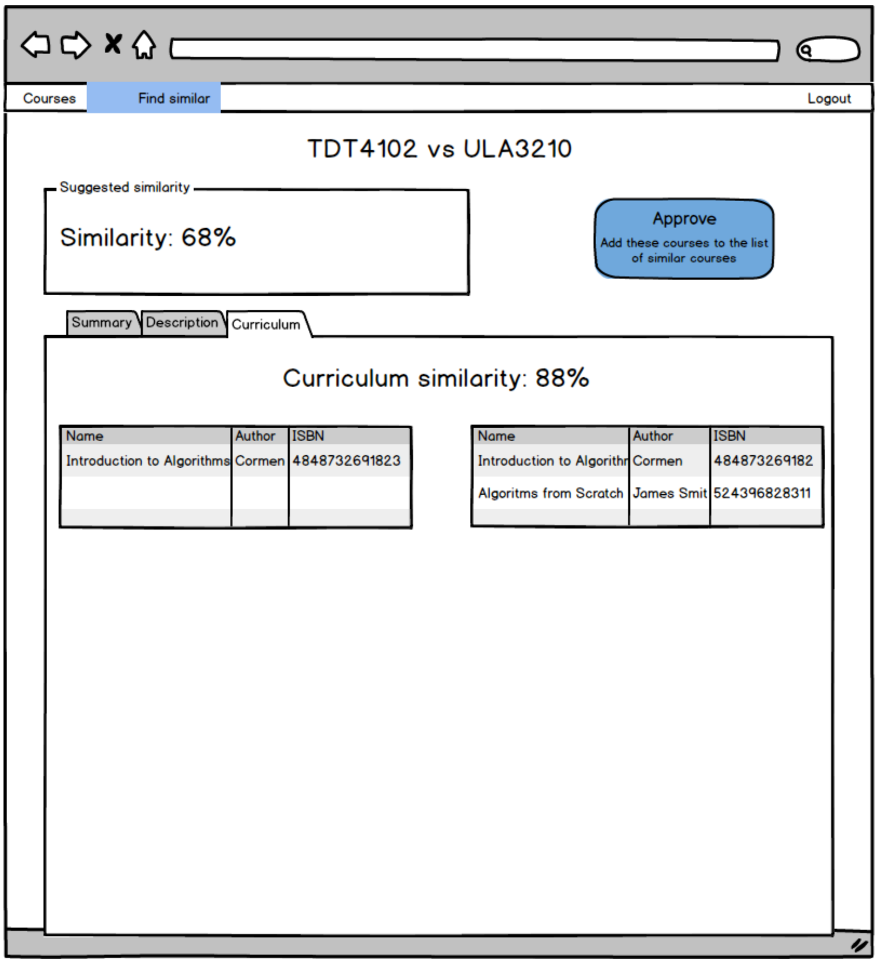


Figure D.8: Compare courses - description



TDT4102 vs ULA3210

Suggested similarity

Similarity: 68%

Curriculum similarity: 88%

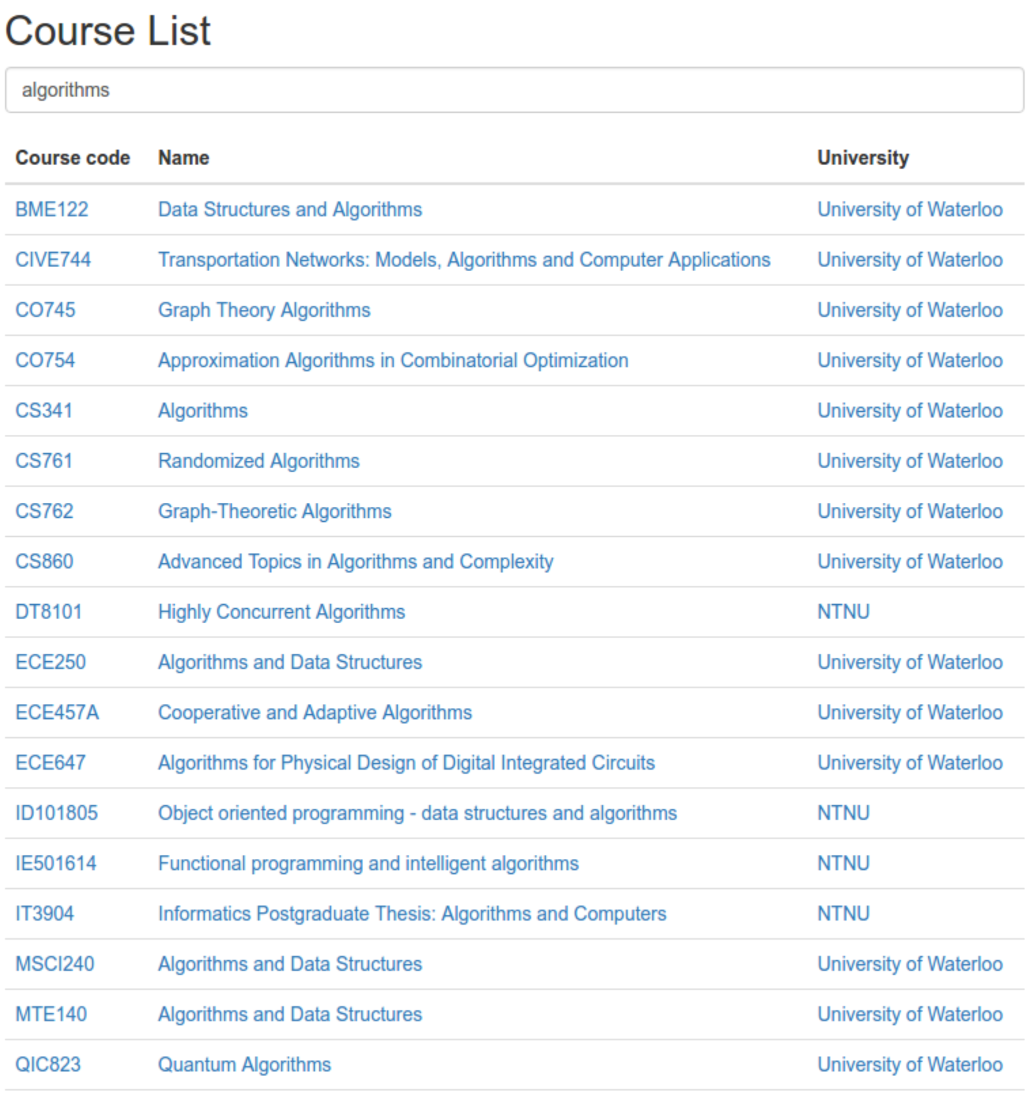
Name	Author	ISBN
Introduction to Algorithms	Cormen	4848732691823

Name	Author	ISBN
Introduction to Algorithr	Cormen	484873269182
Algoritms from Scratch	James Smit	524396828311

Figure D.9: Compare courses - curriculum

Appendix E

Screenshots of the final system

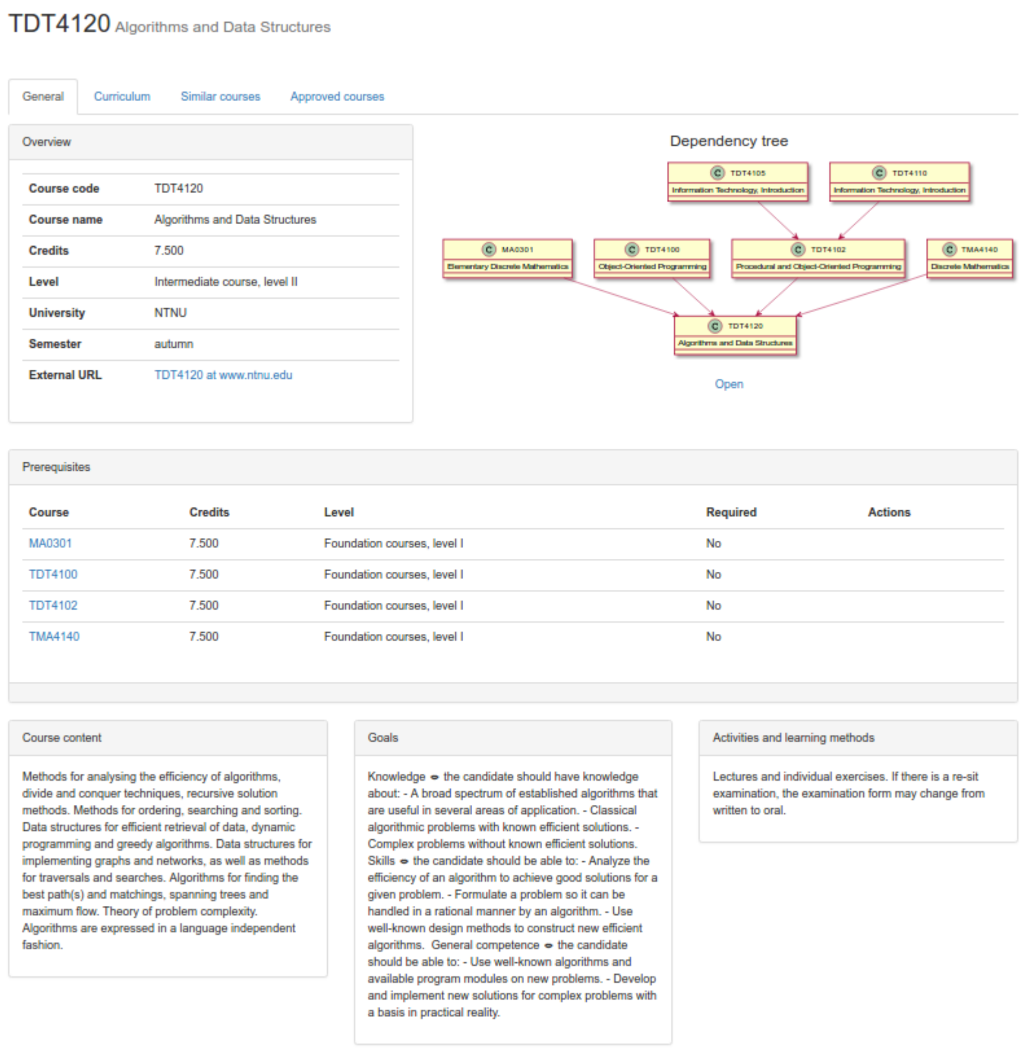


Course List

algorithms

Course code	Name	University
BME122	Data Structures and Algorithms	University of Waterloo
CIVE744	Transportation Networks: Models, Algorithms and Computer Applications	University of Waterloo
CO745	Graph Theory Algorithms	University of Waterloo
CO754	Approximation Algorithms in Combinatorial Optimization	University of Waterloo
CS341	Algorithms	University of Waterloo
CS761	Randomized Algorithms	University of Waterloo
CS762	Graph-Theoretic Algorithms	University of Waterloo
CS860	Advanced Topics in Algorithms and Complexity	University of Waterloo
DT8101	Highly Concurrent Algorithms	NTNU
ECE250	Algorithms and Data Structures	University of Waterloo
ECE457A	Cooperative and Adaptive Algorithms	University of Waterloo
ECE647	Algorithms for Physical Design of Digital Integrated Circuits	University of Waterloo
ID101805	Object oriented programming - data structures and algorithms	NTNU
IE501614	Functional programming and intelligent algorithms	NTNU
IT3904	Informatics Postgraduate Thesis: Algorithms and Computers	NTNU
MSCI240	Algorithms and Data Structures	University of Waterloo
MTE140	Algorithms and Data Structures	University of Waterloo
QIC823	Quantum Algorithms	University of Waterloo

Figure E.1: Course list



TDT4120 Algorithms and Data Structures

General | Curriculum | Similar courses | Approved courses

Overview

Course code	TDT4120
Course name	Algorithms and Data Structures
Credits	7.500
Level	Intermediate course, level II
University	NTNU
Semester	autumn
External URL	TDT4120 at www.ntnu.edu

Dependency tree

TDT4105
Information Technology, Introduction
TDT4110
Information Technology, Introduction
MA0301
Elementary Discrete Mathematics
TDT4100
Object-Oriented Programming
TDT4102
Procedural and Object-Oriented Programming
TMA4140
Discrete Mathematics
TDT4120
Algorithms and Data Structures
Open

Prerequisites

Course	Credits	Level	Required	Actions
MA0301	7.500	Foundation courses, level I	No	
TDT4100	7.500	Foundation courses, level I	No	
TDT4102	7.500	Foundation courses, level I	No	
TMA4140	7.500	Foundation courses, level I	No	

Course content

Methods for analysing the efficiency of algorithms, divide and conquer techniques, recursive solution methods. Methods for ordering, searching and sorting. Data structures for efficient retrieval of data, dynamic programming and greedy algorithms. Data structures for implementing graphs and networks, as well as methods for traversals and searches. Algorithms for finding the best path(s) and matchings, spanning trees and maximum flow. Theory of problem complexity. Algorithms are expressed in a language independent fashion.

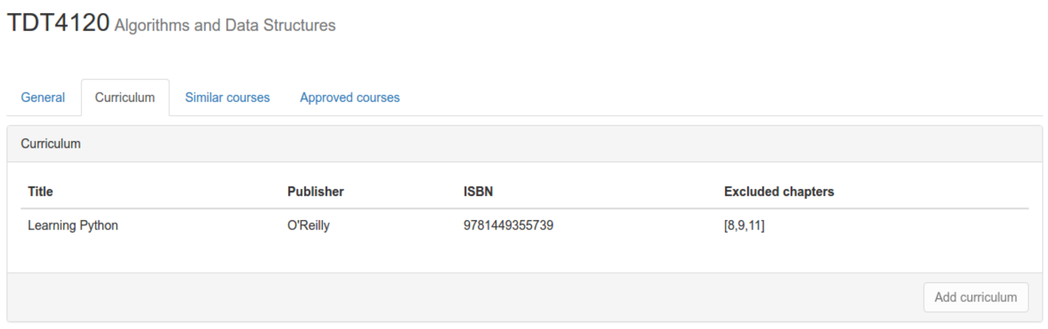
Goals

Knowledge ⬬ the candidate should have knowledge about: - A broad spectrum of established algorithms that are useful in several areas of application. - Classical algorithmic problems with known efficient solutions. - Complex problems without known efficient solutions. Skills ⬬ the candidate should be able to: - Analyze the efficiency of an algorithm to achieve good solutions for a given problem. - Formulate a problem so it can be handled in a rational manner by an algorithm. - Use well-known design methods to construct new efficient algorithms. General competence ⬬ the candidate should be able to: - Use well-known algorithms and available program modules on new problems. - Develop and implement new solutions for complex problems with a basis in practical reality.

Activities and learning methods

Lectures and individual exercises. If there is a re-sit examination, the examination form may change from written to oral.

Figure E.2: Course details



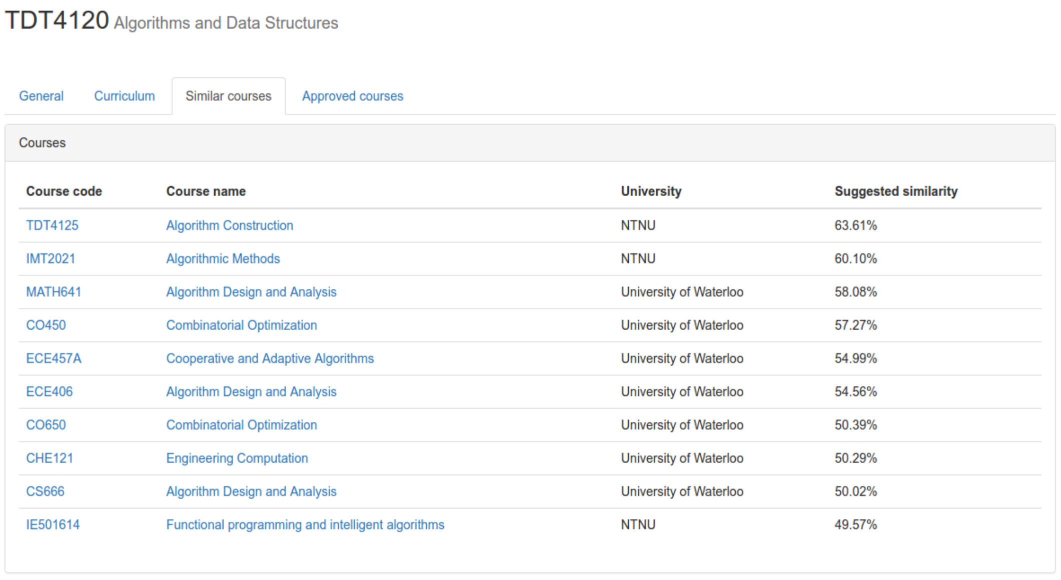
TDT4120 Algorithms and Data Structures

General | Curriculum | Similar courses | Approved courses

Curriculum

Title	Publisher	ISBN	Excluded chapters
Learning Python	O'Reilly	9781449355739	[8,9,11]

Figure E.3: Course details - curriculum



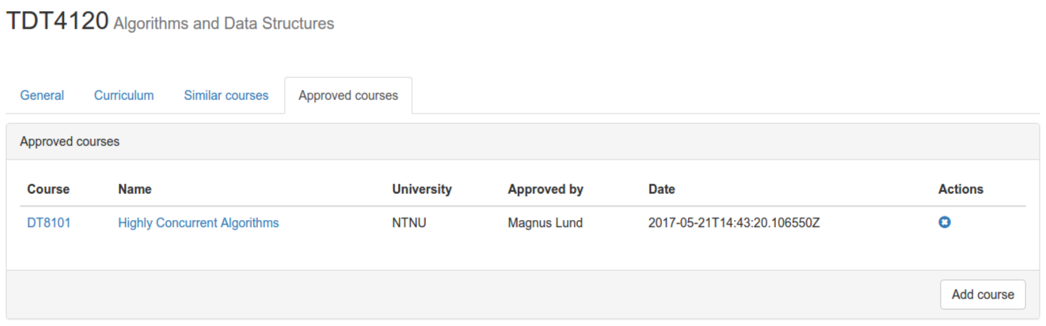
TDT4120 Algorithms and Data Structures

General | Curriculum | Similar courses | Approved courses

Courses

Course code	Course name	University	Suggested similarity
TDT4125	Algorithm Construction	NTNU	63.61%
IMT2021	Algorithmic Methods	NTNU	60.10%
MATH641	Algorithm Design and Analysis	University of Waterloo	58.08%
CO450	Combinatorial Optimization	University of Waterloo	57.27%
ECE457A	Cooperative and Adaptive Algorithms	University of Waterloo	54.99%
ECE406	Algorithm Design and Analysis	University of Waterloo	54.56%
CO650	Combinatorial Optimization	University of Waterloo	50.39%
CHE121	Engineering Computation	University of Waterloo	50.29%
CS666	Algorithm Design and Analysis	University of Waterloo	50.02%
IE501614	Functional programming and intelligent algorithms	NTNU	49.57%

Figure E.4: Course details - similar courses



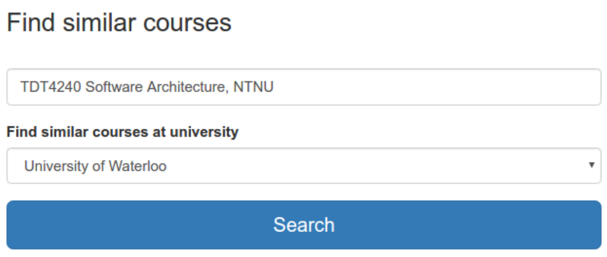
TDT4120 Algorithms and Data Structures

General | Curriculum | Similar courses | Approved courses

Approved courses

Course	Name	University	Approved by	Date	Actions
DT8101	Highly Concurrent Algorithms	NTNU	Magnus Lund	2017-05-21T14:43:20.106550Z	⊗

Figure E.5: Course details - approved courses



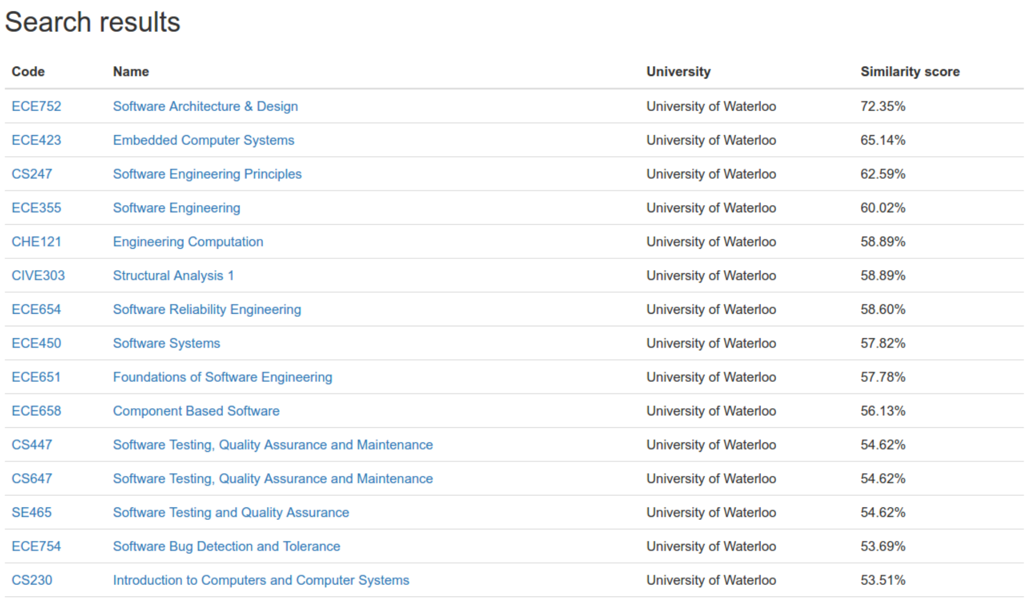
Find similar courses

TDT4240 Software Architecture, NTNU

Find similar courses at university

University of Waterloo

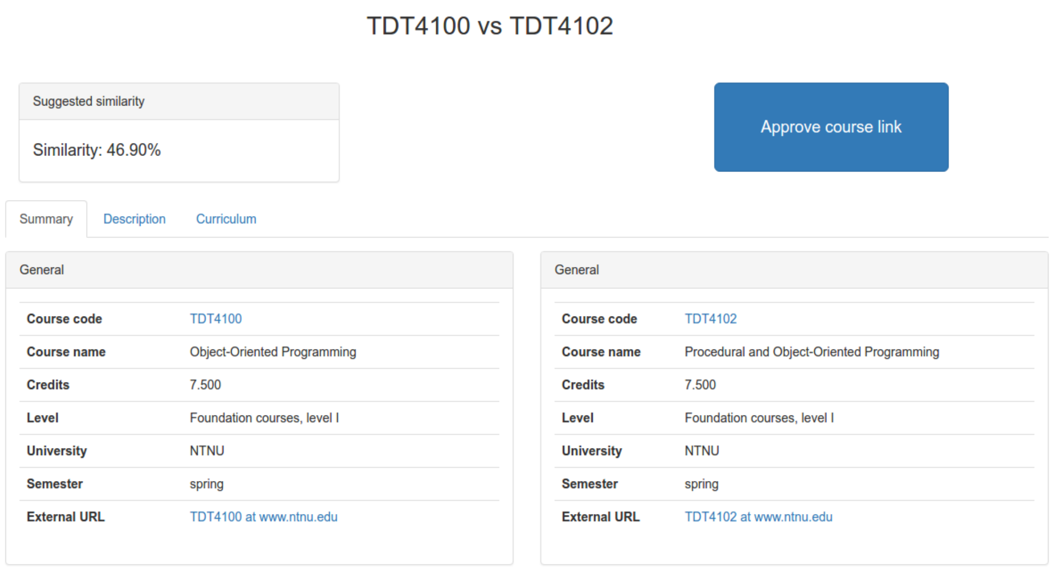
Figure E.6: Search for similar courses on a given university



Search results

Code	Name	University	Similarity score
ECE752	Software Architecture & Design	University of Waterloo	72.35%
ECE423	Embedded Computer Systems	University of Waterloo	65.14%
CS247	Software Engineering Principles	University of Waterloo	62.59%
ECE355	Software Engineering	University of Waterloo	60.02%
CHE121	Engineering Computation	University of Waterloo	58.89%
CIVE303	Structural Analysis 1	University of Waterloo	58.89%
ECE654	Software Reliability Engineering	University of Waterloo	58.60%
ECE450	Software Systems	University of Waterloo	57.82%
ECE651	Foundations of Software Engineering	University of Waterloo	57.78%
ECE658	Component Based Software	University of Waterloo	56.13%
CS447	Software Testing, Quality Assurance and Maintenance	University of Waterloo	54.62%
CS647	Software Testing, Quality Assurance and Maintenance	University of Waterloo	54.62%
SE465	Software Testing and Quality Assurance	University of Waterloo	54.62%
ECE754	Software Bug Detection and Tolerance	University of Waterloo	53.69%
CS230	Introduction to Computers and Computer Systems	University of Waterloo	53.51%

Figure E.7: Result of search


TDT4100 vs TDT4102

Suggested similarity

Similarity: 46.90%

General

Course code	TDT4100
Course name	Object-Oriented Programming
Credits	7.500
Level	Foundation courses, level I
University	NTNU
Semester	spring
External URL	TDT4100 at www.ntnu.edu

General

Course code	TDT4102
Course name	Procedural and Object-Oriented Programming
Credits	7.500
Level	Foundation courses, level I
University	NTNU
Semester	spring
External URL	TDT4102 at www.ntnu.edu

Figure E.8: Compare courses

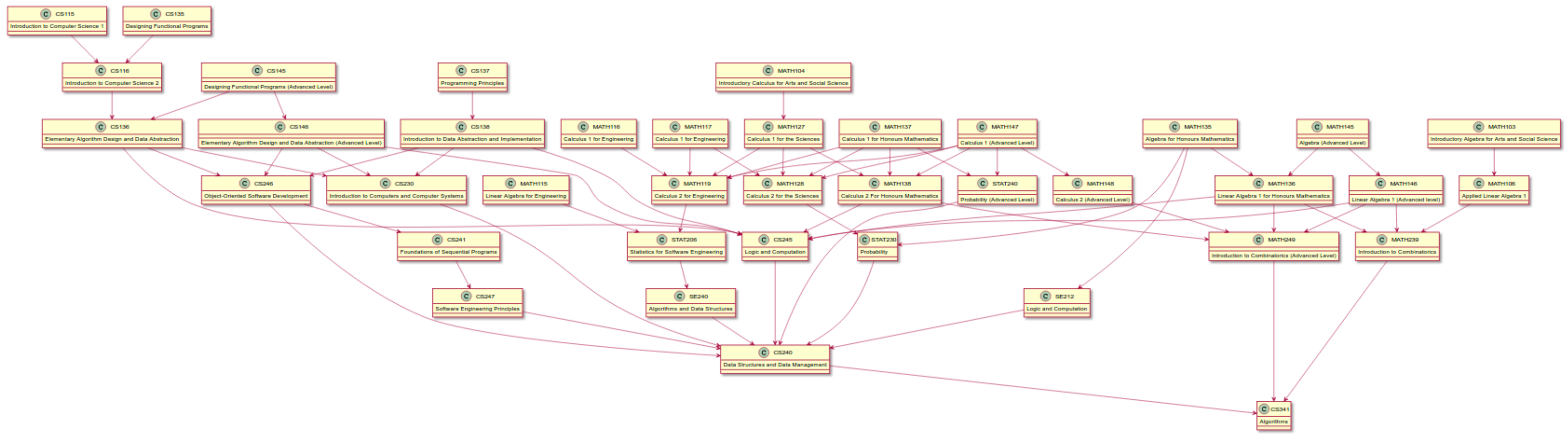

CS115 Introduction to Computer Science 1
CS135 Designing Functional Programs
CS116 Introduction to Computer Science 2
CS145 Designing Functional Programs (Advanced Level)
CS137 Programming Principles
MATH104 Introductory Calculus for Arts and Social Science
CS136 Elementary Algorithm Design and Data Abstraction
CS146 Elementary Algorithm Design and Data Abstraction (Advanced Level)
CS138 Introduction to Data Abstraction and Implementation
MATH116 Calculus 1 for Engineering
MATH117 Calculus 1 for Engineering
MATH127 Calculus 1 for the Sciences
MATH137 Calculus 1 for Honours Mathematics
MATH147 Calculus 1 (Advanced Level)
MATH135 Algebra for Honours Mathematics
MATH145 Algebra (Advanced Level)
MATH103 Introductory Algebra for Arts and Social Science
CS246 Object-Oriented Software Development
CS230 Introduction to Computers and Computer Systems
MATH115 Linear Algebra for Engineering
MATH119 Calculus 2 for Engineering
MATH128 Calculus 2 for the Sciences
MATH138 Calculus 2 For Honours Mathematics
STAT240 Probability (Advanced Level)
MATH148 Calculus 2 (Advanced Level)
MATH136 Linear Algebra 1 for Honours Mathematics
MATH146 Linear Algebra 1 (Advanced level)
MATH106 Applied Linear Algebra 1
CS241 Foundations of Sequential Programs
STAT206 Statistics for Software Engineering
CS245 Logic and Computation
STAT230 Probability
MATH249 Introduction to Combinatorics (Advanced Level)
MATH239 Introduction to Combinatorics
CS247 Software Engineering Principles
SE240 Algorithms and Data Structures
SE212 Logic and Computation
CS240 Data Structures and Data Management
CS341 Algorithms

Figure E.9: Example of a large, generated dependency tree. Note that these are both required and optional course dependencies