PROJECT IN COMMUNICATION SYSTEMS
STOCKHOLM, SWEDEN 2015

Improving the Quality of Web Content through Automated Metrics

An attempt to process course pages at the kth.se website

JENS BAETENS

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Improving the Quality of Web Content through Automated Metrics

An attempt to process course pages at the kth.se website

Jens Baetens

2015-06-08

IK2553 Project Report

Examiner and Academic adviser
Gerald Q. Maguire Jr.

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

Today many organizations make a great deal of content available via a web interface. For this project the main scope will be the public web of KTH Royal Institute of Technology. KTH has lots of data available online. The problem that occurs is that the people who write this content do not get feedback about what they post. The idea of this project is to change that and by giving the responsible person(s) feedback about their content.

The main goal of the project is keeping track of changes of web pages. For example, each course has its own content and information pages - which can be changed by the person(s) responsible for the course. When pages are available in two languages and changes are made to one of them the logical expectation would be that the other language has to be changed as well. However, in the current system there is no systematic way that this change occurs or that someone would be reminded when only one version changed. Similarly for other content pages, although these might not be in multiple languages it would be interesting to see how often a page is updated and to notify the responsible person(s) that a certain page has not been updated in quite some time.

A secondary goal of this project is to provide a proof of concept implementation of a tool that can automatically access web page in KTH Social (a locally developed web service) – thus enabling users to write code that can access, modify, and annotate web pages. Such a tool could be used to compute readability scores for each page and then annotate the page with this score.

The code that was implemented for this project accomplished both goals. Some suggestions are made for how this process can be improved and how alternatively this processing could be done by working directly on the databases used to produce dynamic content.

Keywords

web crawling, readability

Sammanfattning

I dagens läge lägger många organisationer upp stora mängder material på webben. Detta projekt tittar främst på det offentliga webgränssnittet för KTH.

KTH har stora mängder data tillgängligt online. Ett problem är att användare som redigerar och lägger upp denna data inte får återkoppling på den. Iden bakom detta projekt är att ge dessa användare återkoppling om deras material.

Det slutgiltiga målet i detta projekt är att hålla koll på förändring i websidor. Till exempel kurshemsidorna på KTH som har sin egen information och innehåll och kan ändras av kursansvarige. När websidor är tillgängliga i mer än ett språk och det sker en förändring i ena språkets version förväntar man sig att informationen ska ändras på de andra språken också. I det nuvarande systemet finns det dock inget systematiskt sätt som detta ändras på och den ansvarige blir inte påmind att endast den ena ändrats. Man kan tänka sig något liknande för andra typer av innehåll. Till exempel kan det vara intressant att se hur ofta en sida uppdateras och påminna den ansvarige när en sida inte blivit uppdaterad på ett tag.

Ett sekundärt mål med detta projekt är att visa upp ett verktyg som kan nå websidor på KTH Social och möjliggöra ändrandet av websidor med kod. Ett sådant verktyg kan användas för att beräkna “läsbarhetspoäng” (readability points) för varje websida och sedan kommentera sidan med denna poäng.

Den implementerade koden för detta projekt uppnådde båda dessa mål. Några förslag har formulerats för hur denna process kan förbättras till exempel genom att arbeta direkt i databaserna för att producera dynamiskt material.

Nyckelord

webbsökning, läsbarhet

Acknowledgments

I would like to thank Mr. Maguire for providing me the template for this report.

Stockholm, June 2015
Jens Baetens

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

ARI	Automated Readability Index
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transport Protocol
IDE	Integrated DeveLopment Environment
IDLE	(Python) Integrated DeveLopment Environment
KTH	Kungliga Tekniska högskolan (Royal Institute of Technology)
RFC	Request for Comments
SQL	Structured Query Language
URL	Uniform Resource Locator

1 Introduction

This chapter describes the specific problem that this project addresses, the context of the problem, the goals of this project, and outlines the structure of the report.

1.1 Background

The web is constantly changing. This change is often good, but can have some downsides, specifically in a multi-language environment. A problem that often occurs when pages are kept in multiple languages is that the revisions to the page (or content within one page) in one language falls behind the others. A question that arises for end users is which language has the correct information? A question that occurs for content creators is: What is the quality of my content?

1.2 Problem definition

Webpages are maintained by their responsible author(s). The author(s) might forget to update a page in the second (or nth) language or update it poorly because they think that no one will use it. This is a major issue in an environment with many foreign students, such as KTH Royal Institute of Technology. Moreover, KTH has an ambition of being a two language university, hence all essential information should be available in both Swedish and English - and the quality of both versions should be high.

Another problem that occurs is how up to date is each of the various bits of content. For example, there can be an information page about a course that has not been update since last year, but the actual content this course may have changed. In other cases, there are purposely copies of some of the pages for the course over the different years that it has been taught. Is it possible to tell these different cases apart? Is it possible to generate reminders as to when content should be updated for an upcoming course?

In this project both types of problems will be tackled. Information pages about a course can have both types of problems, while the content pages of courses only have the second problem - because in KTH Social pages can be bi-lingual but only within the same page (i.e., there are not separate pages for each language as there are within Polopoly – which is used by KTH for information about the university, departments, policies, calendar, etc.).

1.3 Purpose

The purpose of this report is to improve the content provided on the KTH.se website. The techniques discussed in this report will provide better control over the quality as well as the quantity of this content.

1.4 Goals

The goal of this project is to improve the content on the KTH web. This has been divided into the following two sub-goals:

1. Keeping track of changed pages and
2. Processing text to do spellchecking and computing readability scores.

1.5 Delimitations

The scopes of this project are the course information pages and the content pages within KTH social. While the project only addresses the KTH.se web, it should provide some useful guidance and techniques that can be applied to other websites.

1.6 Structure of the report

Chapter 1 gives some background information about the technologies that have been used. Chapter 1 explains the methods used to achieve the goals. Chapter 1 gives a more detailed explanation of the inner workings of the project and how the proposed solution was realized. Chapter 5 analyses the results of testing the proposed solution. Finally, Chapter 6 concludes this report and suggests some future work.

2 Background

This chapter provides information about the technologies used to achieve the goals of this project.

2.1 HTTP Headers

The HTTP header is a great source of data even without any content being present. There is a difference between response and request fields[1]. The most interesting fields in the header for this project are the response header when making correct requests. For example, you can see if a page has changed since a given data by filling in the `If-Modified-Since` field. If this field is used, then the webserver will return a 304 (Not Modified) [2] when the page has not been modified since the specified date and time. When the page has been modified, then the Response header has an equivalent `Last-Modified` field which will contain the date and time of the last modification in the HTTP date format (defined in RFC 7231 [3]).

Other interesting fields in the response are `Cache-Control` which defines how long a page can be cached before a timeout in seconds. The `Set-Cookie` field gives the info cookie information that needs to be set in order to access a page. Such cookies are frequently used to setup a session after the user has logged in. The `Status Code` is probably the most important field of the header. It contains a code giving the status of the response [2]. The main categories are 1xx information, 2xx success, 3xx redirects, 4xx user error, and 5xx server error, where xx represents two decimal digits.

Interesting fields in the request header are `Cache-Control`, which define whether a caching mechanism is allowed for the response to this request. The `Cookie` field is used to pass information, such as information about the current session.

2.2 Readability scores

This section introduces three aspects of readability scores, i.e., a numeric value indicating how easy it is to read a given set of text.

2.2.1 Flesch–Kincaid Reading Ease

The Flesch–Kincaid Reading Ease[4], also known as Flesch Reading Ease) test is a way to calculate the readability of a text. The computation is based upon the total number of words, sentences, and syllables.

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right).$$

Figure 2-1: Flesch Reading Ease Formula

The score is a number between zero and a hundred. Zero indicates the hardest to read text and a hundred indicates the most easily understood text. For example, a score between sixty and seventy is easily understood by 13- to 15-year olds. In contrast, a score between zero and thirty is best understood by university graduates.

2.2.2 Automated Readability Index

The Automated Readability Index (ARI) is also a means to calculate the readability of a text. As opposed to the Flesh Reading Ease, this test does not return a score, but rather returns a grade in terms of the United States school grade levels. This computation uses the total number of characters, words, and sentences to calculate the grade level.

$$4.71\left(\frac{\text{characters}}{\text{words}}\right) + 0.5\left(\frac{\text{words}}{\text{sentences}}\right) - 21.43$$

Figure 2-2: Automated Readablility Index Fromula

2.2.3 Classification

The reading scores for Flesch–Kincaid Reading Ease can be ordered as shown in Table 2-1.

Table 2-1: Flesch Reading Ease classes

Score	Reading ease
90 – 100	Very Easy
80 – 89	Easy
70 – 79	Fairly Easy
60 – 69	Standard
50 – 59	Fairly Difficult
30 – 49	Difficult
0 – 29	Very Confusing

Automated Readability Index will return a grade which can be translated in to an age range by adding five and six to the grade. This will give a two year range for which the text is written.

2.3 Dynamic content

Dynamic content[5][6] is widely used on the web now days. Dynamic content separates site updates & controlling content from control of the layout. However, when using dynamic content the usefulness of headers, such as `If-Modified-Since` and `Not-Modified` fields, are reduced. Because of demand generated content or when reading from a cache both the Last-Modified and Not-Modified field become useless – as they will often be the exact request time or something close to when the content is read from the cache.

Another major change is the use of a login system to limit the access to pages to specific users or groups of users. Such a system can be used so that a user can edit the content of a page via a web interface after logging in. This login is done via a form, but often this form is used in combination with session cookies and login cookies. These values can be found in the `Set-Cookie` field of the header and need to be provided in the `Cookie` field in any request in order to access the content.

2.4 Summary

Headers are very important when requesting and receiving data via the web. They can provide a great deal of information without having to process the actual data. Although exploiting headers becomes harder in the case of a dynamic content system.

Some pages are protected by a login system so that outsiders, who have not logged in, cannot access the content.

Reading ease scores are ways to determine the difficulty of reading a text based upon some variables. There are a variety of readability scores; the interested reader is referred to “The Writer’s Workbench” set of tools, such as `diction` and `style`, as well as `look` and `spell`.

3 Methodology

The purpose of this chapter is to provide an overview of the research method used in this project. Section 3.1 describes the research process. Section 3.2 focuses on the data collection techniques used for this research. Section 3.3 describes the tools used and the environment they were used in.

3.1 Research Process

Figure 3-1 shows the steps conducted in order to carry out this research. The first step is to get a web page. Next we collect the relevant data concerning this page. For example, page headers and toolbars are irrelevant and can be ignored. We compare the newly collected data against a stored version of the web page in order to see if the content has changed. Finally, we store the content if it has changed – along with a timestamp.

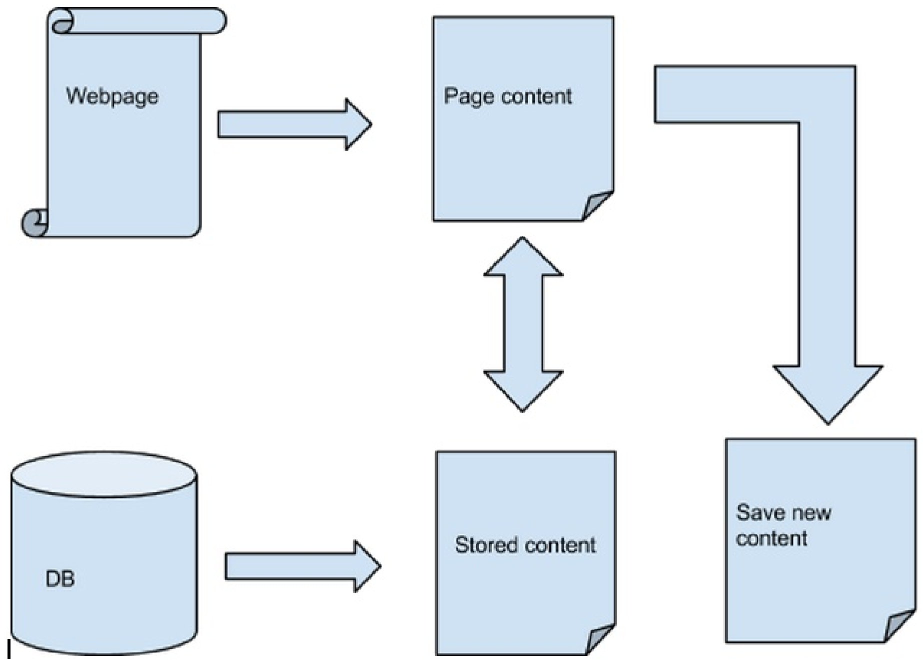
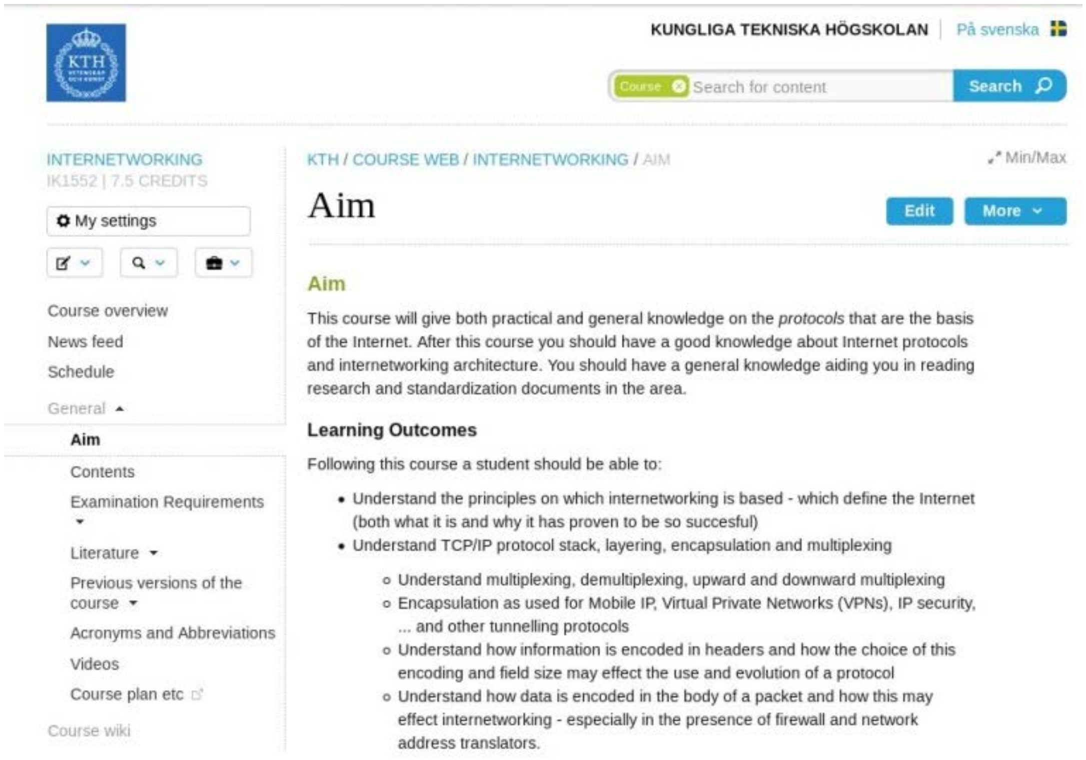


Figure 3-1: Research Process

3.2 Data Collection

All data used for this report come from the web domain kth.se[7]. Part of the aim of this project is to use live data. This data is assumed to be valid and reliable since it is accessible by the general public or by students within one or more of the courses given by KTH. Part of the effort in collecting data is to understand the Hyper Text Markup Language (HTML) structure used by the KTH website. For the following discussion we will use the course page shown in Figure 3-2. The following structures were chosen to find certain parts of this page.



Aim

Aim

This course will give both practical and general knowledge on the *protocols* that are the basis of the Internet. After this course you should have a good knowledge about Internet protocols and internetworking architecture. You should have a general knowledge aiding you in reading research and standardization documents in the area.

Learning Outcomes

Following this course a student should be able to:

- Understand the principles on which internetworking is based - which define the Internet (both what it is and why it has proven to be so succesful)
- Understand TCP/IP protocol stack, layering, encapsulation and multiplexing
    - Understand multiplexing, demultiplexing, upward and downward multiplexing
    - Encapsulation as used for Mobile IP, Virtual Private Networks (VPNs), IP security, ... and other tunnelling protocols
    - Understand how information is encoded in headers and how the choice of this encoding and field size may effect the use and evolution of a protocol
    - Understand how data is encoded in the body of a packet and how this may effect internetworking - especially in the presence of firewall and network address translators.

Figure 3-2: A page from the course IK1552 (https://www.kth.se/social/course/IK1552/page/aim-30/)

Figure 3-3 shows the start of the main content (for the page shown above). The actual start is the `article` tag. However, this tag has no other identifier. Due to not having an `id` or `class` the tag can't be used as a search reference. As it may occur that multiple `article` tags are opened. The next tag, illustrates a typical phenomenon of dynamic content, is the tag that is the most interesting for crawling since it has a very specific identifier. The fact it has both a `class` and an `id` makes it almost impossible to have the exact same tag with `class` and `id` somewhere else on the page. This will be tag used to start the content collection.

```
   <article>
       <div id="article" class="article">
     <div class="contentarea">
       <div class="mainContent">
           <div class="preArticleParagraphs">
             <div class="topControls">
               <div id="top-controls" class="top">

               </div>
             </div>
             <h1 class="title">Aim</h1>
           </div>
           <div class="paragraphs"><h2>Aim</h2>
   <p>This course will give both practical and general knowledge on the
<em>protocols</em> that are the basis of the Internet. After this course you
should have a good knowledge about Internet protocols and internetworking
architecture. You should have a general knowledge aiding you in reading
research and standardization documents in the area.</p>
   <h3>Learning Outcomes</h3>
<p>Following this course a student should be able to:</p>
```

Figure 3-3: Start of the main content

Figure 3-4 shows the start of page navigation. This is embedded in a bigger `ul` which contains some links that are irrelevant for this project. The tag that stands out here is the `ul` with class `pagetree`. Every a element where the `href` attribute does not start with http or https can be considered part of the course's page. This `ul` tag will be used to collect all pages for a specific course.

```
<li> <ul class="pagetree"><li class="admin_pages no-initial-hide trans-
title"><div class="menuRow"> <span>General</span> </div> <ul><li
id="content_54771411f276540f54a4b916" class="current"><div class="menuRow">
<span>Aim</span> </div></li><li id="content_54771442f2765401658d779d">
                <div class="menuRow">
                    <a href="/social/course/IK1552/page/contents-
5/">Contents</a>
                </div></li><li id="content_54771470f2765412a25ed1cb">
                <div class="menuRow">
                    <a href="/social/course/IK1552/page/examination-
requirements-5/">Examination Requirements</a>
                </div> <ul><li id="content_55129949f276543ba3cb619c">
                <div class="menuRow">
```

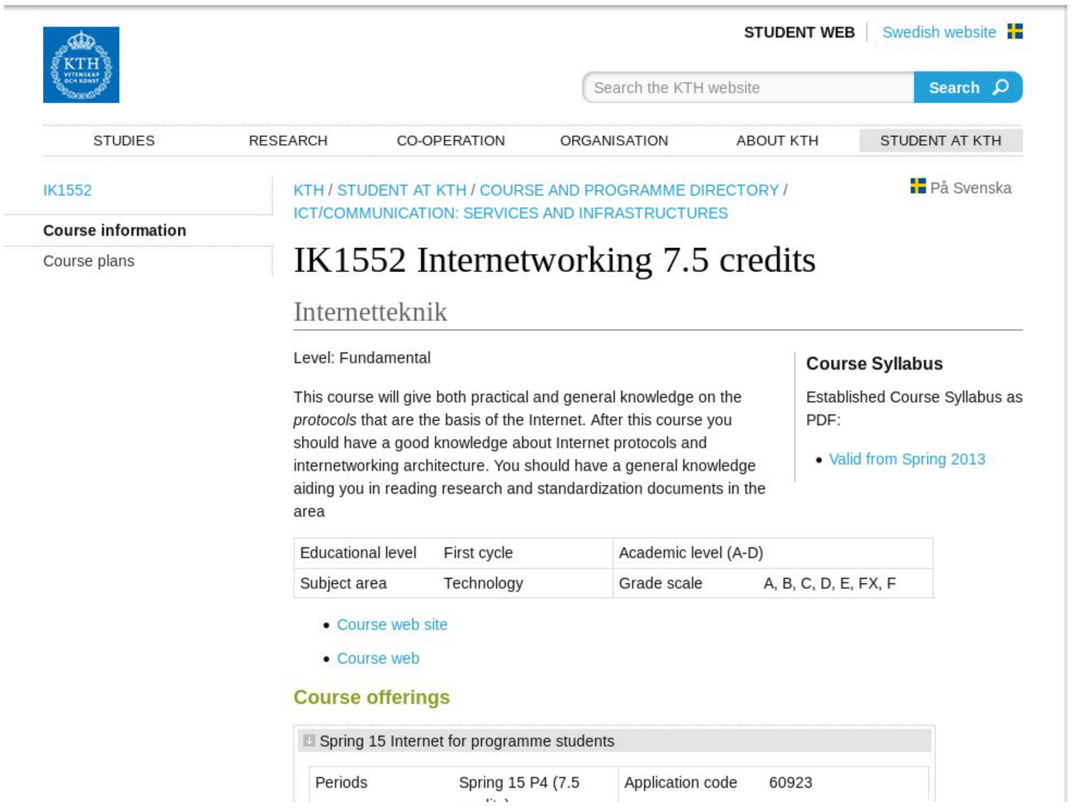
Figure 3-4: Start of the page navigation

The course pages that were used for testing during this project are listed in Table 3-1.

Table 3-1: List of course pages used for testing

Course id	Course information page	Course page in KTH Social
IK1550	https://www.kth.se/student/kurser/kurs/IK1550	https://www.kth.se/social/course/IK1550/
IK1552	https://www.kth.se/student/kurser/kurs/IK1552	https://www.kth.se/social/course/IK1552/
IK1501	https://www.kth.se/student/kurser/kurs/IK1501	https://www.kth.se/social/course/IK1501/
IK135U	https://www.kth.se/student/kurser/kurs/IK135U	https://www.kth.se/social/course/IK135U/
IK131V	https://www.kth.se/student/kurser/kurs/IK131V	https://www.kth.se/social/course/IK131V/

It is easy to notice the pattern in these Uniform Resource Locator (URLs), the course information pages base URL is `https://www.kth.se/student/kurser/kurs/` followed by the course id. For the English version of the information page you add `?l=en` to the URL. For the KTH social page the base URL is `https://www.kth.se/social/course/` followed by the course id. There is no second language possible on the KTH social page. Figure 3-5 shows an example of the IK1552 course page in English and Figure 3-6 in Swedish.


STUDENT WEB | Swedish website

Search the KTH website [Search]

STUDIES | RESEARCH | CO-OPERATION | ORGANISATION | ABOUT KTH | STUDENT AT KTH

IK1552

Course information

Course plans

KTH / STUDENT AT KTH / COURSE AND PROGRAMME DIRECTORY / ICT/COMMUNICATION: SERVICES AND INFRASTRUCTURES

På Svenska

IK1552 Internetworking 7.5 credits

Internetteknik

Level: Fundamental

This course will give both practical and general knowledge on the *protocols* that are the basis of the Internet. After this course you should have a good knowledge about Internet protocols and internetworking architecture. You should have a general knowledge aiding you in reading research and standardization documents in the area

Course Syllabus

Established Course Syllabus as PDF:

- Valid from Spring 2013

Educational level	First cycle	Academic level (A-D)	
Subject area	Technology	Grade scale	A, B, C, D, E, FX, F

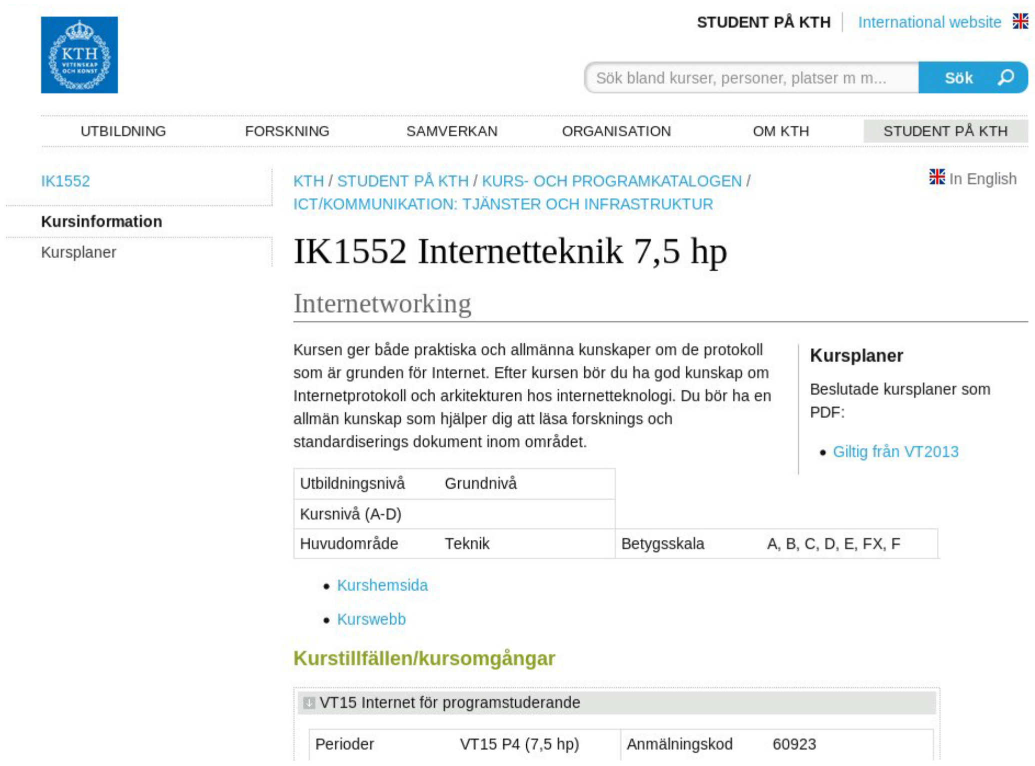
- Course web site
- Course web

Course offerings

Spring 15 Internet for programme students

Periods	Spring 15 P4 (7.5	Application code	60923

Figure 3-5: **Course information page in English (https://www.kth.se/student/kurser/kurs/IK1552?l=en)**


STUDENT PÅ KTH | International website

Sök bland kurser, personer, platser m m... [Sök]

UTBILDNING | FORSKNING | SAMVERKAN | ORGANISATION | OM KTH | STUDENT PÅ KTH

IK1552

Kursinformation

Kursplaner

KTH / STUDENT PÅ KTH / KURS- OCH PROGRAMKATALOGEN / ICT/KOMMUNIKATION: TJÄNSTER OCH INFRASTRUKTUR

In English

IK1552 Internetteknik 7,5 hp

Internetworking

Kursen ger både praktiska och allmänna kunskaper om de protokoll som är grunden för Internet. Efter kursen bör du ha god kunskap om Internetprotokoll och arkitekturen hos internetteknologi. Du bör ha en allmän kunskap som hjälper dig att läsa forsknings och standardiserings dokument inom området.

Kursplaner

Beslutade kursplaner som PDF:

- Giltig från VT2013

Utbildningsnivå	Grundnivå		
Kursnivå (A-D)			
Huvudområde	Teknik	Betygsskala	A, B, C, D, E, FX, F

- Kurshemsida
- Kurswebb

Kurstillfällen/kursomgångar

VT15 Internet för programstuderande

Perioder	VT15 P4 (7,5 hp)	Anmälningskod	60923

Figure 3-6: **Course information page in Swedish (https://www.kth.se/student/kurser/kurs/IK1552?l=sv)**

3.3 Needed tools

A number of tools were needed to access the web pages and to achieve the goals stated in Section 1.4 on page 15. This section describes both the test environment that was created in order to experiment with these tools and the tools themselves.

3.3.1 Test environment

In order to be able to rerun any of the tests you need access to kth.se. The course information pages are publically accessible, so no extra information is needed. For most of the course pages on KTH social a login is needed. Having a valid login to access kth.se should give full access to most courses. Although some pages are only accessible for students enrolled in a specific course.

3.3.2 Hardware/Software to be used

The software in this project is written in Python 2.7.9 [8][*]. For development of Python almost any text editor will do or the standard Python Integrated DeveLopment Environment (IDLE). However, I used Pycharm[9], which is a more advanced Integrated DeveLopment Environment (IDE) for Python. The library used to handle sessions and cookies is RoboBrowser [10].

MySQL is used for the database. This project made use of Wampserver 2.5 [11], which contains Apache 2.4.9, MySQL 5.6.17, PHP 5.5.12, and PHPMyAdmin 4.1.14. With all these tools is possible to locally host a database server and have easy access to it using the PHPMyAdmin [12] interface.

[*] Any version of Python 2.7.x should be useable together with this software.

4 Implementation

This chapter explains the implementation of the different parts of this project. Section 4.1 describes the structure of the database that has been used for storing content. Section 4.2 describes how one can perform spell checking and compute readability scores for the web pages. Section 4.3 describes how one can crawl the entire kth.se website.

4.1 Database design

The database is designed to store all the content of the web pages that have been accessed. The scheme for this database is shown in Figure 4-1. `Courses` is the main table which holds a course ID and the corresponding course name. A course id is used as the primary key (PK) and is the foreign key (FK) in the other database tables. The course id is a very important field because it is used to form the URL for the course. `CourseTexts` is the table that holds information about the public course information pages. These pages are available in both English and Swedish. The time fields are used to keep track of when the last update was pushed to the database. In the `coursePage` table you can find the course specific pages. These KTH Social pages are only available in one language (although a given page can have content in one or more languages) and a course can have multiple subpages. For this reason there is a double PK with the id and the pageName. (The list of course ids and their corresponding pages that were used for testing are listed in Table 3-1 on page 23.)

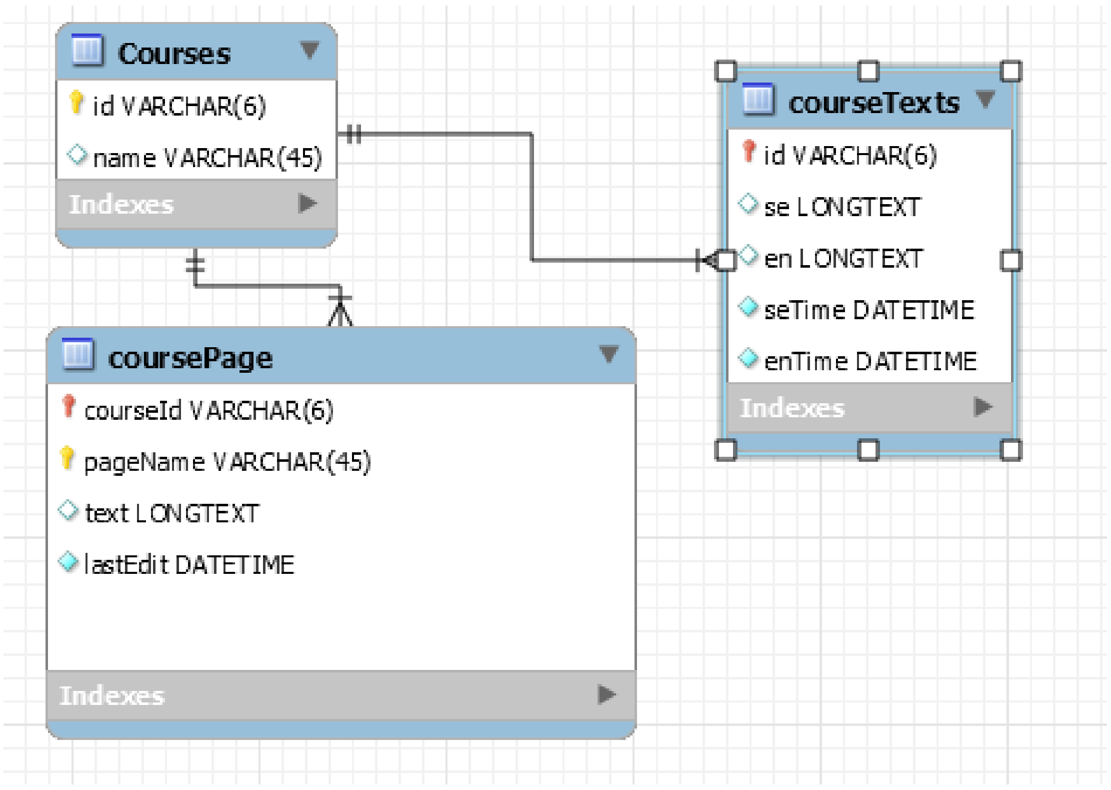


Figure 4-1: Database design

4.2 Spell Checking and readability

As an initial set of operations on the content of a page we have implemented spell checking and readability score computation. These are each described in the following subsections.

4.2.1 Spell Checking

Spell checking can be done by various different approaches. The first option is to write a spell checking program and provide a dictionary file to compare the words in the text against [13]. The second option and the option used in this project is to use an existing spell checking library. The library used for spell checking is called Pychant [14]. Pychant is an open source [15] spellchecking library for python. Pychant was selected because it is easy to use. An example of using this library's enchant module for a single word is shown in Figure 4-2.

```
>>> import enchant
>>> d = enchant.Dict("en_US")
>>> d.check("Hello")
True
>>> d.check("Helo")
False
>>> d.suggest("Helo")
['He lo', 'He-lo', 'Hello', 'Helot', 'Help', 'Halo', 'Hell', 'Held', 'Helm', 'Hero', "He'll"]
```

Figure 4-2: An example of using a spell checker in Python

For this project a small helper method (shown in Figure 4-3) was implemented to make it easy to check whole texts for spelling mistakes. The function returns a count indicating the number of (potential) spelling errors. This is a `SpellChecker` class that by default checks against the `en_US` dictionary. The dictionary can be replace with a different language (eg. `en_GB`) using the `change_to_dict` funtion, a wordlist in a file using the `change_to_wordlist` function ro a combination of both. By either adding a word list to the current dictionary with `add_wordlist_to_dict` or `change_to_dict_and_wordlist`. By using a file as wordlist custom dictionary can be made that contain field specific words.

```
def check_text(self, text):
    count = 0
    words = text.split()
    for word in words:
        word = word.lower().strip(".:;?!)(")
        if word != "":
            if not self.check(word):
                count += 1
    return count
```

Figure 4-3: Helper method

4.2.2 Readability scores

For the readability scores a list of functions was implemented to evaluate a text. The main function (shown in Figure 4-4) calculates the Flesch Reading Ease and the ideal readers' age based on ARI.

```
# Calculates and prints different text stats
def calculate(text):
    print get_readability(text)
    print "Readers age: " + get_readers_age(text)
```

Figure 4-4: Readability calculation

ARI and Flesch values are calculated using a helper function (shown in Figure 4-3) to compute word count, character count, and syllables.

```
def automated_readability_index(text):
    chars = char_count(text)
    words = word_count(text)
    sentences = sentence_count(text)
    a = float(chars)/float(words)
    b = float(words)/float(sentences)
    ARI = 4.71 * round(a, 2) + 0.5*round(b, 2) - 21.43
    return round(ARI, 1)


def flesch_reading_ease(text):
    words = word_count(text)
    sentences = sentence_count(text)
    syllables = syllable_count(text)
    a = float(words)/float(sentences)
    b = float(syllables)/float(words)
    felsch = 206.835 - 1.015*a - 84.6*b
    return round(flesch, 2)
```

Figure 4-5: Helper functions for ARI and Flesch computations

4.3 Crawling the web

The main loop of the program is shown in Figure 4-6. This main loop consists of two parts. The first part sets up a connection to the database and connects to the `courses` table. The program uses this to get a list of courses to process. The next part is the actual web crawling. This part is multi-threaded to decrease the crawling time and to take advantage of the local CPU's capabilities.

```
conn = Connection()
table = conn.get_table("courses")

print 'Courses'
threads = []
for (id, name) in table:
    thread = CrawlThread(id)
    thread.start()
    threads.append(thread)
for thread in threads:
    thread.join()
```

Figure 4-6: **Main loop**

The first class used is Connection. Connection is a class which contains functions to facilitate interaction with a database. The functions in the Connection class are shown in Table 4-1.

Table 4-1: **Functions in the Connection class**

```
get_table(self, name)
insert(self, table, columns, data)
update(self, table, columns, data, where)
query(self, query)
```

The `get_table` function is used to fetch all of columns for all rows in a given table. The `insert` and `update` function both implement their MySQL equivalent operations. The parameters are the table name to insert or update, the columns to insert or update in an array, the data for the responding columns is also in an array, and for update there is also a where clause (An example of such a where clause would be: id=10).

The second class used is CrawlThread. This is the main execution class that delegates the crawling within a thread. Figure 4-7 shows the pseudo code that performs the crawling of web pages.

Get the texts from the database Get the Swedish and English text from the web	
If texts are <u>not</u> in database	**If texts are in database**
Insert new texts and current time into the database	Compare texts **If text is updated** update value in database (text and timestamp)

Figure 4-7: **Crawling of the course information pages**

Crawling of content pages is very different from crawling the course information pages because of the potential need to login. For access to most of the content pages you have to be logged in to kth.se. The pseudo code for content page crawling is shown in Figure 4-8. As can be seen in the pseudo code, a major part of this code concerns logging in. Once logged in all of the doors are open and you can use the session cookie to walk all of the pages.

```
Get the login page
Get the JSESSIONID from the cookie
Extract the hidden fields from the form
Post the form with the JSESSION cookie in the header and the login data (including the hidden fields) in the post data
Get the SESSION cookie from the post response
Walk the navigation for links using the SESSION cookie

Get the text form the database
Get the text from the web using the SESSION cookie
If text is not in database
    Insert new text and current time into the database
If text is in database
    Compare text
    If text is update
        update value in database
        (text and timestamp)
```

Figure 4-8: Crawling the content pages

The file html.py is used to get webpages. This file contains allof the tools necessary to crawl and extract webpages. The main classes inside this file are `HTMLTagFinder`, which is used to find all of the content within a certain tag with a given class (within the kth.se pages the main content can be found in a `div` with class `article` as shown in Figure 3-3); `NavWalker`, which returns a list of all of the links inside of a given tag with a given class (for KTH Social this is a `ul` tag with class `pagetree`, as was shown in Figure 3-4); and `InputFieldExtractor`, which is used to find input fields and return their values based on their type and name (two hidden fields in the login form are `lt` and `execution`, these are fields have random values that change every time the login page is refreshed). All of these classes are derived from `HTMLParser` [16]. This is a base class provided in the standard python library. It has all of the functions needed to parse Hyper Text Markup Language (HTML) documents. The most interesting funtions in `HTMLParser` are:

```
handle_starttag(self, tag, attrs)
handle_endtag(self, tag)
handle_data(self, data)
feed(self, data)
```

`Handle_starttag` will be called every time a new tag is opened. The `tag` parameter holds the tag name and the `attrs` parameter will hold a list of tuples with each attribute. `Handle_endtag` will be called on every every closing of a tag. `And handle_data` will be called when the parser hits content text inbetwen tags. `Feed` is used to start the parsing. Feeding either a string or Unicode to this function will trigger all of the other funtions. In this project the feed method will return the requested data in the subclasses.

Additional useful methods are:

```
get_page(url, headers={}, post_data=None)
get_page_raw(url, headers={}, post_data=None)
is_page_reachable(url)
get_cookie(url, headers={}, post_data=None)
```

Get_page will return the content of a page. The function get_raw_page returns the page *before* .read() is executed, thus giving access to headers and more control over the processing of the page. The function is_page_reachable is a quick test to see if the page is there and the access does **not** return a status code of 404 [2]. The function get_cookie is a short hand to get the set-cookie field in the header.

4.4 Logging in to kth.se

At first I tried to login in to kth manually with the use of a library. The following code resulted from that idea.

```
page = html.get_page_raw(base_url_login)
session_id = page.headers.dict['set-cookie'].split(';')[0]
page_content = page.read()
lt = html.get_input_field(page_content, "lt", "hidden")
execution = html.get_input_field(page_content, "execution", "hidden")
cookie = html.get_cookie(base_url_login + "?l=en;jsessionid=" +
session_id.split('=')[1],
                          post_data={'username': 'jbaetens',
                                     'password': 'Redacted',
                                     '_eventId': 'submit',
                                     'submit': 'Log in',
                                     'lt': lt,
                                     'execution': execution
                                     },
                          headers={'Cookie': session_id})
links = html.get_nav_links(html.get_page(base_url_social + name,
headers={'Cookie': cookie}), 'ul', 'pagetree')
```
Figure 4-9: Manual Approach

The manual approach consist of requesting the page and getting the session id from the from the header. Followed by reading the content of the page and gathering the generated fields lt and exectution. Figure 4-10 show the headers and formdata used to login using Google Chrome.

```
Header:
POST /login?l=en HTTP/1.1 Host: login.kth.se Connection: keep-alive Content-Length: 134
Cache-Control: max-age=0 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 Origin:
https://login.kth.se User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36 Content-Type: application/x-www-
form-urlencoded Referer:
https://login.kth.se/login;jsessionid=B3AFE336DC762C3E918E41D3322A6E4A?l=en Accept-
Encoding: gzip, deflate Accept-Language: nl-NL,nl;q=0.8,en-US;q=0.6,en;q=0.4 Cookie:
JSESSIONID=B3AFE336DC762C3E918E41D3322A6E4A


Form Data:
username=jbaetens&password=Redacted&lt=LT-1331734-OD7RFY3PzOLqnzi5JA7yedk9aiVyXj-
login01&execution=e3s1&_eventId=submit&submit=Log+in
```
Figure 4-10: Headers and Form Data during login

Although trying to mimic the request header and form data as good as possible this did not workout. I was never able to gett the correct cookie as a return value.

After lots of tweaks and tries I gave up on the manual approach and switched to using a library. With RoboBrowser the approach was easier.

```
browser = RoboBrowser()
browser.open(base_url_login)
form = browser.get_form(id="fm1")
form['username'] = 'jbaetens'
form['password'] = 'Redacted'
browser.submit_form(form)
browser.open(base_url_social + name)
links = html.get_nav_links(browser.response.content, 'ul', 'pagetree')
```

Figure 4-11: RoboBrwoser Approach

RoboBrowser is in essence a normal browser inside a Python class. It works by getting the page and focuses on the form. Just filling in the fields and submitting will set the cookies exactly as it would in a normal browser. Simply opening the KTH Social page of a course with RoboBrowser after doing the log in sequence will guarantee that the cookies are used. The code will return version of the page that is the same as a logged in user would receive. The variable `browser.response.content` contains the page's HTML data. Parsing it thru the `get_nav_links` function will get all links of a KTH Social page. This includes sub links as theses are just `a` elements netsted deeper. The `NavWalker` doesn't care about a page being linked under a dropdown in the navigation menu. It just gets all links in the navigation whether they are top level or sub level links.

4.5 Applying these functions to a web page

The first step is to decide what you want to extract form the page: the links in the navigation menu or the content of the page. Figure 4-12 shows how to get a page's content.

```
page_html = html.get_page(base_url_course + name)
page_content = html.get_element(page_html, "div", "article")
```

Figure 4-12: Getting a page content

In the first step, the HTML for the whole page is gotten by using the `html.get_page` function. The whole page is then passed to the `html.get_element` function which will create a `HTMLTagFinder` and `feed` the page content to it. The `HTMLTagFinder` will return the actual textual content of the page, without or with very little formatting. This content can then be parsed by the `SpellChecker` and/or reading score calculation functions. For checking whole texts `SpellChecker` has a `check_text` function which will return the total number of spelling mistakes in the given text. To calculate the reading scores running the calculate function will print both the reading ease and reader's age (range). Or you can simply run *one* of these functions independent(e.g., `get_readability` or `get_readers_age`).

Getting the navigation links on a KTH Social page has a similar apporoach. Passing the page's content to the `html.get_nav_links` function with the tag `ul` and class `pagetree` will give you all of the links contained within `a` elements within the `ul` tag. Internally the function creates a `NavWalker` and feeds the page to it. Links starting with "http" or "https" typically do not need to be checked (or walked) since they will most likley are external to the kth.se web site.

5 Analysis

In this chapter we analyze the design decisions that were made and the results of the testing that was conducted. The design and evaluation carried out during this project will also be reflected upon.

5.1 Code base

The code base, except for the crawling package, is written to be general purpose. The code in the `html` and `db` package are desgined to be highly re-usable. Although this code is written for a very specific environment, the database connection code has most of the functions needed to interact with a database (except for a delete function). For example, the query function can execute all queries passed to it. The `html` package has all basic functions needed to get and extract web pages. The classes are written to extract certain fields within a webpage. These classes are more specific to the problem at hand, but can be used in a wider context. However, the code in the crawling package is very specific to the kth.se web and was written explicitly for this purpose. The overall code flow will be the same for other sites, but the specifics will change. The general approach to extracting webpages should remain the same. The code in the `text` package contains the spelling and readability code. This code does all the required calculations on a given text. This code can be used on any string.

The souce code for this project is available from the same page in DiVA* where this report can be found.

5.2 Database

The data extracted from the web is put into a database. This database can utilized in multiple ways. A daily walk over the records of the courses' information pages could detect any changes made to either language. For example, a program could e-mail the responsible person that their page in a certain language was updated, when the corresponding page in the other language was not updated. Similarly a program could tell them that they have not updated a page for a certain amount of time. One of the issues with the database in its current form is that very long pages do not fit, hence the page should be placed in a blob. This would require some changes in the code since when using a blob SQL will return a byte array rather than a Unicode string.

5.3 Testing results

The following results were computed for the courses listed in Table 3-1. Spellchecking was only performed against the default en_US dictionary. If pages are in Swedish, then the spell checking would need to use a Swedish dictionary. However, that was not evaluated within this project. For the full results see Appendix A. Note that not all of the results from testing the KTH social pages are shown in the tables, but all of these results are shown in Appendix A.

* www.diva-portal.org

5.3.1 Spell checking

Table 5-1 gives the number of spelling mistakes on the page. High amounts of spelling mistakes can be due to very technical and specific language which are not supported by the default dictionary. It is obvious from these numbers that one needs to use an augmented dictionary when spell checking such pages. There is also a need to learn which words are considered mis-spelled so that they could be added to a page specific or topic area dictionary.

Table 5-1: SpellCheck results

Course id	Course information page (English version)	Course pages in KTH Social
IK1550	~235	aim-28/ ~103 contents-4/ ~74
IK1552	~262	aim-30/ ~103 contents-5/ ~74
IK1501	~126	course-material-p2-2013/ ~218 ?in_wiki=1575 ~948
IK135U	~9	
IK131V	~24	

5.3.2 Readability scores

Table 5-2 shows that the readability scorese give a very clear indication of the age of the students that could be expected to read these pages.

Table 5-2: Readability score results

Course id	Course information page	Course pages in KTH Social
IK1550	39.03 (Difficult) Readers age: 23.1 - 24.1	aim-28/ 42.34 (Difficult) Readers age: 22.3 - 23.3 contents-4/ 58.5 (Fairly Difficult) Readers age: 22.9 - 23.9
IK1552	43.83 (Difficult) Readers age: 21.7 - 22.7	aim-30/ 42.14 (Difficult) Readers age: 22.3 - 23.3 contents-5/ 58.23 (Fairly Difficult) Readers age: 22.9 - 23.9
IK1501	42.0 (Difficult) Readers age: 21.6 - 22.6	course-material-p2-2013/ 65.05 (Standard) Readers age: 19.8 - 20.8 ?in_wiki=1575 56.41 (Fairly Difficult) Readers age: 22.3 - 23.3
IK135U	27.13 (Very Confusing) Readers age: 20.7 - 21.7	Note that there is **no** course page for this course, as this course is only given for contract education for specific companies..
IK131V	53.15 (Fairly Difficult) Readers age: 17.5 - 18.5	Note that there is **no** course page for this course, as it is not currently offered.

6 Conclusions and Future work

This chapter concludes the report. Section 6.1 provides information about the goals that were reached and some insights. In Section 6.2 the limitations of this project are discussed. Section 6.3 presents some suggestiong of future work that can be done based upon this project.

6.1 Conclusions

The goals for this project where to keep track of changing pages and to process text to calculate readability and to count spelling mistakes. The result of this project is a code base that can perform both tasks. In this project I have learned to use a new scripting language and gained greater insght in to the working of cookies and their value in building login sessions.

Some suggestions I would give for a similar projects are to start off by inspecting the page to crawl. By looking at the HTML you can learn a lot about the structure of the page. This is especially true for dynamically generated pages. Dynamically generated pages will mostly have a large number of tags wrapped around data. This can be intimidating, but start by eliminate the tags that are there simply for structure. This will enable you to find those tags that are most relevant for your task.

6.2 Limitations

Part of this project was to keep track of page changes. The easiest and most convient way would be to access the database behind the view. However, this data was inaccessable to me. Hence this was the part of this project that had the biggest limitation.

The other limitation I expierenced is the horrible and complex structure of the login system. While it works perfectly for normal browsing, automating it was extremely hard. I did not get it to work completely. The code should mimic the login process, but I was not able to return a proper working session without using an external library.

6.3 Future work

The next task would be to write scripts to inspect the database. For example, a script that runs every day could check which pages have changed and take some action based upon these changes. Alternatively a script could mine data from this database. This project’s code base was written with an eye on the future, hence I choose to keep it as abstract as possible whereever I could so in the expectation that the code could be reused in other web related projects.

Note that an alternative method of accessing the web pages would be to access them directly from the KTH database that is used to produce the course information (KOPPS) and the database used by KTH Social. This would require that the code be developed and executed by someone who has access to these databases. An advantage of having this direct access is that one could utilize database triggers to invoke a set of operations when a page is changed. Additionally, this approach would avoid the problem of the web login, at the cost of having to have the code executed by privileged users.

References

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "'HTTP/1.1: Header Field Definitions', Internet Request for Comments, vol. RFC 2616 (Draft Standard)," Jun-1999. [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html. [Accessed: 01-Jun-2015].

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "HTTP/1.1: Status Code Definitions, Internet Request for Comments, vol. RFC 2616 (Draft Standard)." [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. [Accessed: 16-Apr-2015].

[3] R. Fielding and J. Reschke, "'Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content', Internet Request for Comments, vol. RFC 7231 (Proposed Standard)." [Online]. Available: https://tools.ietf.org/html/rfc7231. [Accessed: 01-Jun-2015].

[4] J. P. Kincaid, R. P. Fishburne, R. L. Rogers, and B. S. Chissom, *Derivation of New Readability Formulas (Automated Readability Index, Fog Count, and Flesch Reading Ease formula) for Navy Enlisted Personnel. Research Branch Report 8-75. Chief of Naval Technical Training: Naval Air Station Memphis.* 1975.

[5] W3C, "Static vs. Dynamic Web Sites - W3C Wiki." [Online]. Available: http://www.w3.org/wiki/How_does_the_Internet_work#Static_vs._Dynamic_Web_Sites. [Accessed: 03-Jun-2015].

[6] M. Gabbrielli, M. Machiori, and F. S. de Boer, "Dynamic web sites." [Online]. Available: http://www.w3.org/People/Massimo/papers/rmm.pdf.

[7] "KTH | Välkommen till KTH." [Online]. Available: https://www.kth.se/. [Accessed: 16-Apr-2015].

[8] "Python Downlad 2.7.9," *Python.org*. [Online]. Available: https://www.python.org/downloads/release/python-279/. [Accessed: 27-May-2015].

[9] "Python IDE & Django IDE for Web developers : JetBrains PyCharm." [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed: 27-May-2015].

[10] "robobrowser." [Online]. Available: http://robobrowser.readthedocs.org/en/latest/. [Accessed: 01-Jun-2015].

[11] "WampServer," *WampServer*. [Online]. Available: http://www.wampserver.com/en/. [Accessed: 27-May-2015].

[12] phpMyAdmin contributors, "phpMyAdmin," *phpMyAdmin*. [Online]. Available: http://www.phpmyadmin.net/home_page/index.php. [Accessed: 27-May-2015].

[13] "How to Write a Spelling Corrector." [Online]. Available: http://norvig.com/spell-correct.html. [Accessed: 16-Feb-2015].

[14] "PyEnchant." [Online]. Available: http://pythonhosted.org/pyenchant/. [Accessed: 16-Feb-2015].

[15] "rfk/pyenchant," *GitHub*. [Online]. Available: https://github.com/rfk/pyenchant. [Accessed: 16-Feb-2015].

[16] "19.1. HTMLParser — Simple HTML and XHTML parser — Python 2.7.10rc1 documentation." [Online]. Available: https://docs.python.org/2/library/htmlparser.html. [Accessed: 18-May-2015].

Appendix A: Full script result

```
Courses
Started crawling IK1550
Started crawling IK1552
Started crawling IK1501
Started crawling IK135U
Started crawling IK131V
IK135U 27.13 (Very Confusing) Readers age: 20.7 - 21.7
IK131V 53.15 (Fairly Difficult) Readers age: 17.5 - 18.5
IK135U Spelling mistakes: ~9
IK131V Spelling mistakes: ~24
IK1501 42.0 (Difficult) Readers age: 21.6 - 22.6
IK1550 39.03 (Difficult) Readers age: 23.1 - 24.1
IK1552 43.83 (Difficult) Readers age: 21.7 - 22.7
IK1501 Spelling mistakes: ~126
IK1550 Spelling mistakes: ~235
IK1552 Spelling mistakes: ~262
Done crawling IK135U (3393 ms)
Done crawling IK131V (3499 ms)
IK1550/social/course/IK1550/page/aim-28/ 42.34 (Difficult) Readers age: 22.3 - 23.3
IK1552/social/course/IK1552/page/aim-30/ 42.14 (Difficult) Readers age: 22.3 - 23.3
IK1501/social/course/IK1501/page/course-material-p2-2013/ 65.05 (Standard) Readers
    age: 19.8 - 20.8
IK1550/social/course/IK1550/page/aim-28/ Spelling mistakes: ~103
aim-28 has updated for course IK1550
IK1552/social/course/IK1552/page/aim-30/ Spelling mistakes: ~103
aim-30 has updated for course IK1552
IK1501/social/course/IK1501/page/course-material-p2-2013/ Spelling mistakes: ~218
course-material-p2-2013 has updated for course IK1501
IK1550/social/course/IK1550/page/contents-4/ 58.5 (Fairly Difficult) Readers age: 22.9 -
    23.9
IK1550/social/course/IK1550/page/contents-4/ Spelling mistakes: ~74
contents-4 has updated for course IK1550
IK1552/social/course/IK1552/page/contents-5/ 58.23 (Fairly Difficult) Readers age: 22.9 -
    23.9
IK1552/social/course/IK1552/page/contents-5/ Spelling mistakes: ~74
contents-5 has updated for course IK1552
IK1501/social/course/IK1501/createpage/?in_wiki=1575 56.41 (Fairly Difficult) Readers
    age: 22.3 - 23.3
IK1501/social/course/IK1501/createpage/?in_wiki=1575 Spelling mistakes: ~948
?in_wiki=1575 has updated for course IK1501
Done crawling IK1501 (4415 ms)
IK1550/social/course/IK1550/page/examination-requirements-4/ 63.81 (Standard)
    Readers age: 19.1 - 20.1
IK1552/social/course/IK1552/page/examination-requirements-5/ 63.65 (Standard)
    Readers age: 19.2 - 20.2
IK1550/social/course/IK1550/page/examination-requirements-4/ Spelling mistakes: ~138
examination-requirements-4 has updated for course IK1550
IK1552/social/course/IK1552/page/examination-requirements-5/ Spelling mistakes: ~138
examination-requirements-5 has updated for course IK1552
IK1552/social/course/IK1552/page/ik1552-paper/ 72.15 (Fairly Easy) Readers age: 17.9 -
    18.9
```

```
IK1550/social/course/IK1550/page/code-of-honor-and-regulations-4/ 69.14 (Standard)
        Readers age: 20.7 - 21.7
IK1552/social/course/IK1552/page/ik1552-paper/ Spelling mistakes: ~53
ik1552-paper has updated for course IK1552
IK1550/social/course/IK1550/page/code-of-honor-and-regulations-4/ Spelling mistakes:
        ~49
code-of-honor-and-regulations-4 has updated for course IK1550
IK1552/social/course/IK1552/page/code-of-honor-and-regulations-6/ 69.14 (Standard)
        Readers age: 20.9 - 21.9
IK1552/social/course/IK1552/page/code-of-honor-and-regulations-6/ Spelling mistakes:
        ~49
code-of-honor-and-regulations-6 has updated for course IK1552
 IK1550/social/course/IK1550/page/grading-24/ 67.76 (Standard) Readers age: 15.8 - 16.8
IK1550/social/course/IK1550/page/grading-24/ Spelling mistakes: ~56
grading-24 has updated for course IK1550
IK1552/social/course/IK1552/page/grading-31/ 67.43 (Standard) Readers age: 15.9 - 16.9
IK1552/social/course/IK1552/page/grading-31/ Spelling mistakes: ~55
grading-31 has updated for course IK1552
IK1550/social/course/IK1550/page/suggestions-when-writing-your-report-3/ 40.34
        (Difficult) Readers age: 22.7 - 23.7
IK1550/social/course/IK1550/page/suggestions-when-writing-your-report-3/ Spelling
        mistakes: ~117
suggestions-when-writing-your-report-3 has updated for course IK1550
IK1552/social/course/IK1552/page/suggestions-when-writing-your-report-4/ 40.19
        (Difficult) Readers age: 22.8 - 23.8
IK1550/social/course/IK1550/page/literature-29/ 63.96 (Standard) Readers age: 21.1 -
        22.1
IK1550/social/course/IK1550/page/literature-29/ Spelling mistakes: ~122
literature-29 has updated for course IK1550
IK1552/social/course/IK1552/page/suggestions-when-writing-your-report-4/ Spelling
        mistakes: ~117
suggestions-when-writing-your-report-4 has updated for course IK1552
IK1550/social/course/IK1550/page/lecture-notes-64/ 70.71 (Fairly Easy) Readers age:
        19.3 - 20.3
IK1550/social/course/IK1550/page/lecture-notes-64/ Spelling mistakes: ~43
lecture-notes-64 has updated for course IK1550
IK1552/social/course/IK1552/page/literature-34/ 62.94 (Standard) Readers age: 22.2 -
        23.2
IK1552/social/course/IK1552/page/literature-34/ Spelling mistakes: ~118
literature-34 has updated for course IK1552
IK1550/social/course/IK1550/page/supplementary-readings-3/ 56.79 (Fairly Difficult)
        Readers age: 22.4 - 23.4
IK1552/social/course/IK1552/page/lecture-notes-81/ 73.25 (Fairly Easy) Readers age:
        20.4 - 21.4
IK1552/social/course/IK1552/page/lecture-notes-81/ Spelling mistakes: ~57
lecture-notes-81 has updated for course IK1552
IK1550/social/course/IK1550/page/supplementary-readings-3/ Spelling mistakes: ~210
supplementary-readings-3 has updated for course IK1550
IK1552/social/course/IK1552/page/supplementary-readings-4/ 56.6 (Fairly Difficult)
        Readers age: 22.5 - 23.5
IK1552/social/course/IK1552/page/supplementary-readings-4/ Spelling mistakes: ~210
supplementary-readings-4 has updated for course IK1552
```

```
IK1550/social/course/IK1550/page/useful-urls-3/ 65.39 (Standard) Readers age: 20.7 -
        21.7
IK1550/social/course/IK1550/page/useful-urls-3/ Spelling mistakes: ~69
useful-urls-3 has updated for course IK1550
IK1550/social/course/IK1550/page/other-on-line-course-related-material-3/ 64.56
        (Standard) Readers age: 22.7 - 23.7
IK1552/social/course/IK1552/page/useful-urls-4/ 56.3 (Fairly Difficult) Readers age: 21.6
        - 22.6
IK1550/social/course/IK1550/page/other-on-line-course-related-material-3/ Spelling
        mistakes: ~44
other-on-line-course-related-material-3 has updated for course IK1550
IK1552/social/course/IK1552/page/useful-urls-4/ Spelling mistakes: ~61
useful-urls-4 has updated for course IK1552
IK1552/social/course/IK1552/page/other-on-line-course-related-material-4/ 63.13
        (Standard) Readers age: 23.0 - 24.0
IK1552/social/course/IK1552/page/other-on-line-course-related-material-4/ Spelling
        mistakes: ~44
other-on-line-course-related-material-4 has updated for course IK1552
IK1550/social/course/IK1550/page/sources-for-further-information-3/ 51.28 (Fairly
        Difficult) Readers age: 24.5 - 25.5
IK1552/social/course/IK1552/page/sources-for-further-information-4/ 51.21 (Fairly
        Difficult) Readers age: 24.6 - 25.6
IK1550/social/course/IK1550/page/sources-for-further-information-3/ Spelling mistakes:
        ~930
sources-for-further-information-3 has updated for course IK1550
IK1552/social/course/IK1552/page/sources-for-further-information-4/ Spelling mistakes:
        ~946
sources-for-further-information-4 has updated for course IK1552
IK1550/social/course/IK1550/page/sample-papers-2/ 71.07 (Fairly Easy) Readers age:
        20.4 - 21.4
IK1550/social/course/IK1550/page/sample-papers-2/ Spelling mistakes: ~112
sample-papers-2 has updated for course IK1550
IK1552/social/course/IK1552/page/sample-papers-3/ 71.07 (Fairly Easy) Readers age:
        20.4 - 21.4
IK1552/social/course/IK1552/page/sample-papers-3/ Spelling mistakes: ~112
sample-papers-3 has updated for course IK1552
IK1550/social/course/IK1550/page/previous-versions-of-the-course-3/ 89.6 (Easy)
        Readers age: 22.3 - 23.3
IK1550/social/course/IK1550/page/previous-versions-of-the-course-3/ Spelling mistakes:
        ~44
previous-versions-of-the-course-3 has updated for course IK1550
IK1550/social/course/IK1550/page/file-for-previous-versions-of-the-intern/ -6.22 (Very
        Confusing) Readers age: 48.9 - 49.9
IK1550/social/course/IK1550/page/file-for-previous-versions-of-the-intern/ Spelling
        mistakes: ~55
file-for-previous-versions-of-the-intern has updated for course IK1550
IK1552/social/course/IK1552/page/previous-versions-of-the-course-4/ 60.87 (Standard)
        Readers age: 34.9 - 35.9
IK1552/social/course/IK1552/page/previous-versions-of-the-course-4/ Spelling mistakes:
        ~101
previous-versions-of-the-course-4 has updated for course IK1552
IK1550/social/course/IK1550/page/acronyms-and-abbreviations-3/ 67.43 (Standard)
        Readers age: 22.5 - 23.5
```

```
IK1550/social/course/IK1550/page/acronyms-and-abbreviations-3/ Spelling mistakes:
        ~45
acronyms-and-abbreviations-3 has updated for course IK1550
IK1552/social/course/IK1552/page/exam-requirements-2014/ 64.09 (Standard) Readers
        age: 19.1 - 20.1
IK1552/social/course/IK1552/page/exam-requirements-2014/ Spelling mistakes: ~140
exam-requirements-2014 has updated for course IK1552
IK1550/social/course/IK1550/page/videos/ 65.76 (Standard) Readers age: 32.9 - 33.9
IK1550/social/course/IK1550/page/videos/ Spelling mistakes: ~101
videos has updated for course IK1550
IK1552/social/course/IK1552/page/acronyms-and-abbreviat/ -121.06 (Very Confusing)
        Readers age: 82.1 - 83.1
IK1550/social/course/IK1550/createpage/?in_wiki=2274 56.41 (Fairly Difficult) Readers
        age: 22.3 - 23.3
IK1552/social/course/IK1552/page/acronyms-and-abbreviat/ Spelling mistakes: ~465
acronyms-and-abbreviat has updated for course IK1552
IK1550/social/course/IK1550/createpage/?in_wiki=2274 Spelling mistakes: ~948
?in_wiki=2274 has updated for course IK1550
IK1552/social/course/IK1552/page/videos-5/ 81.37 (Easy) Readers age: 28.4 - 29.4
IK1550/social/course/IK1550/page/test-page-13/ 71.69 (Fairly Easy) Readers age: 19.9 -
        20.9
IK1550/social/course/IK1550/page/test-page-13/ Spelling mistakes: ~47
test-page-13 has updated for course IK1550
IK1552/social/course/IK1552/page/videos-5/ Spelling mistakes: ~84
videos-5 has updated for course IK1552
Done crawling IK1550 (9291 ms)
IK1552/social/course/IK1552/createpage/?in_wiki=185086 56.41 (Fairly Difficult)
        Readers age: 22.3 - 23.3
IK1552/social/course/IK1552/createpage/?in_wiki=185086 Spelling mistakes: ~948
?in_wiki=185086 has updated for course IK1552
Done crawling IK1552 (9586 ms)
```

TRITA-ICT-EX-2015:92