SHIVA BESHARAT POUR

Shiva Besharat Pour

2018-06-09

Bachelor’s Thesis (Alpha Draft)

Examiner
Gerald Q. Maguire Jr.

Academic adviser
Anders Västberg

Abstract

As the era of digitalization dawns, the need to integrate separate silos into a synchronized connected system is becoming of ever greater significance. This thesis focuses on the Canvas Learning Management System (LMS) and the Digitala vetenskapliga arkive (DiVA) as examples of separate silos.

The thesis presents several methods of automating document handling associated with a degree project. It exploits the fact that students will submit their thesis to their examiner via Canvas. Canvas is the LMS platform used by students to submit all their coursework. When the examiner approves the thesis, it will be archived in DiVA and optionally published on DiVA. DiVA is an institutional repository used for research publications and student theses.

When manually archiving and publishing student theses on DiVA several fields need to be filled in. These fields provide meta data for the thesis itself. The content of these fields (author, title, keywords, abstract, …) can be used when searching via the DiVA portal. It might not seem like a massive task to enter this meta data for an individual thesis; however, given the number of theses that are submitted every year, this process takes a large amount of time and effort. Moreover, it is important to enter this data correctly, which is difficult when manually doing this task. Therefore, this thesis project seeks to automate this process for future theses.

The proposed solution parses PDF documents and uses information from the LMS in order to automatically generate a cover for the thesis and fill in the required DiVA meta data. Additionally, information for inserting an announcement of the student's oral thesis presentation into a calendar system will be provided. Moreover, the data in each case will be checked for correctness and consistency.

Manually filling in DiVA fields in order to publish theses has been a quite demanding and time-consuming process. Thus, there is often a delay before a thesis is published on DiVA. Therefore, this thesis project's goal is to provide KTH with an automated means to handle thesis archiving and publication on DiVA, while doing so faster, more efficiently, and with fewer errors. The correctness of the extracted meta data will be evaluated by comparing the results to the previously entered meta data for theses that have previously been achieved in DiVA. Regarding the efficiency of the automatic process, it takes roughly 50 seconds to prepare the information needed to publish a thesis to DiVA using the new method, compared with 1 hour in the previous manual method.

Keywords

RESTful API, Canvas, DiVA, Calendar announcement, data mining

Sammanfattning

När digitaliseringens tid uppstår, så blir behovet av att integrera separata silor i ett synkroniserat anslutet system större. Denna avhandling fokuserar på Canvas Learning Management System (LMS) och Digitala vetenskapliga arkivet (DiVA) som är exempel på separata silor.

Avhandlingen presenterar flera metoder för automatisering av dokumenthantering för examensarbeten. Projektet utnyttjar det faktum att eleverna kommer att skicka sin avhandling till sin examinator via Canvas. Canvas är den LMS-plattform som används av eleverna för att lämna in sitt kursarbete. När examinatorn godkänner avhandlingen kommer den att arkiveras i DiVA och eventuellt publiceras på DiVA. DiVA är ett institutionellt arkiv som används för forskningspublikationer och studentavhandlingar.

När man manuellt arkiverar och publicerar studentuppsatser på DiVA måste flera fält fyllas i. Dessa fält ger metadata för själva avhandlingen. Innehållet i dessa fält (författare, titel, nyckelord, abstrakt, ...) kan användas vid sökning via DiVA-portalen. Även om det inte är en stor uppgift att skriva in denna metadata för en individuell uppsats så blir det en mycket arbetsam process för många examensarbeten. Dessutom är det viktigt att ange dessa uppgifter korrekt, vilket är svårt när man manuellt utför den här uppgiften. Därför syftar detta avhandlingsprojekt till att automatisera denna process för framtida avhandlingar.

Lösningen som presenteras i denna avhandling kommer att analysera PDF-dokument och använda annan information från LMS för att automatiskt skapa en fram- och baksida för avhandlingen och fylla i de nödvändiga DiVA-metadata. Grunden för införandet av denna data i ett kalendersystem för att ge ett meddelande om studentens presentation kommer också att ges. Dessutom kontrolleras uppgifterna för korrekthet.

Manuell fyllning av DiVA-fält för att publicera avhandlingar har varit en ganska arbetsam och tidskrävande process. Således är det ofta en fördröjning innan en avhandling publiceras på DiVA. Därför ska detta projektet ge KTH ett automatiserat system att hantera avhandlingar och publicering på DiVA, samtidigt som det gör det snabbare, mer effektivt och med färre fel. Korrektheten hos de extraherade metadatan kommer att utvärderas genom att jämföra resultaten med de tidigare inmatade metadatan för examensarbeten som redan ligger uppe på DiVA. När det gäller effektiviteten i den automatiska processen tar det ungefär 50 sekunder att förbereda resurer för att publicera en avhandling till DiVA med hjälp av den nya metoden jämfört med 1 timme i tidigare manuell metod.

Nyckelord

RESTful API, Canvas, DiVA, kalendrar, data mining

Acknowledgments

We would like to thank the following people for their guidance and support during the Connecting Silos project:

Professor Gerald Q. Maguire Jr. – Examiner

Anders Västberg - Supervisor

Stockholm, June 2018
Shiva Besharat Pour

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

API	Application Programming Interface
CDC	Canvas Data Collection module
CMS	Content Management System
DE	Data Extraction module
DiVA	Digitala vetenskapliga arkive (Swedish)
DP	DiVA Publication module
EECS	School of Electrical Engineering and Computer Science
ISRN	International Standard Technical Report Number
ISSN	International Standard Serial Number
IDE	Integration Development Environment
IT	Information Technology
JSON	JavaScript Object Notation
KDC	KTH Data Collection module
KTH	KTH Royal Institute of Technology (English) / Kungliga Tekniska högskolan (Swedish)
LMS	Learning Management System
LTI	Learning Tools Interoperability
MODS	Metadata Object Description Schema
ORCID	Open Researcher and Contributor ID
PDF	Portable Document Format
QA	Quality Assurance
SSH	Secure Shell
TA	Test Automation module
XML	Extensible Markup Language

1 Introduction

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

In order to achieve efficiency, it is desirable to automate routine tasks that are demanding with respect to time and effort when done manually. This thesis presents the design, implementation, and evaluation of an automation solution applied to processing some elements of degree projects. The aim is to provide increased efficiency, increased accuracy, and reduce the time and effort needed by *all* involved. Note that the goal is with respect to everyone who is involved with the process, thus it includes the students who author the thesis, the faculty (as an adviser or examiner), and administrative staff.

1.1 Background

Canvas[*] is a learning management system (LMS) used by many schools and institutions for assignments and coursework. One of the main purposes of Canvas is for the teachers to create coursework/assignments, and for students to submit their work and receive a grade. Canvas is quite well organized and automated when it comes to handling student submissions. It is also useful when it comes to its automation regarding organizing the assignments by consistently managing them in terms of their priority.

DiVA[†] (Digitala vetenskapliga arkive (in Swedish)) is an archiving and publishing system for research and student theses. KTH Royal Institute of Technology (here after simply KTH) uses DiVA as an archive for student theses. One of the goals of this project is to facilitate publishing approved student theses that have been submitted via Canvas to DiVA. However, to do so, specific meta data has to be entered into DiVA. Currently, this meta data is entered via fields presented using a web interface to the DiVA Portal. These fields need to be filled in with information about the thesis (title, abstract (in both English and Swedish), keywords (often in both English and Swedish), number of pages, etc.; the names of the student, examiner, and advisers; the defense date; student's degree program; etc. The thesis is uploaded to DiVA for archiving and optionally the full text of the thesis is published via DiVA. This entire process is currently done manually. This requires a significant amount of time (roughly an hour per thesis) by staff members to enter the meta data and thesis into DiVA. Moreover, before a thesis can be uploaded to DiVA it has to be assigned a report number, the cover made, and the cover attached to the front and back of the thesis.

[*] https://www.instructure.com/

[†] http://kth.diva-portal.org

Currently, automation is lacking when it comes to connecting Canvas to other digital platforms, such as DiVA. More detailed information about the Canvas LMS and DiVA portal will be provided in Sections 2.4 and 2.5 (respectively).

1.2 Problem

If the number of theses that are submitted were few, then the time taken for the process of entering these theses into DiVA would be insignificant. However, considering the number of these that are submitted is high (see Table 1-1 and Table 1-2), thus some problems arise regarding the efficiency of the complete process. It is worth mentioning that, especially towards the end of every academic year, many students are submitting their thesis or dissertation. Therefore, not only are there a large number of theses to enter into DiVA, but the work is frequently very concentrated in a small part of the year.

Table 1-1: Number of degree project reports in DiVA for all of KTH

Year	Total number	Full-text in DiVA	Full-text not available in DiVA
2017	2287	2053	234
2016	2376	2182	194
2015	2601	2316	285
2014	2384	2050	334
2013	2356	2035	321
2012	2500	1873	627
2011	2282	1640	642
2010	504	486	38

Table 1-2: In 2017, the organizations that comprise the School of Electrical Engineering and Computer Science (EECS) as of 1 January 2018 had 697 theses (only 24 without full text)

Organization	Number
School of Computer Science and Communication (CSC)	338
School of Information and Communication Technology (ICT)	154
School of Electrical Engineering (EES)	47
Electric Power and Energy Systems	30
Automatic Control	29
Media Technology and Interaction Design, MID	18
Information Science and Engineering	17
Electromagnetic Engineering	14
Space and Plasma Physics	10
Robotics, perception and learning, RPL	10

Manually extracting the meta information required by DiVA for each thesis is quite repetitive, demanding of both time and concentration, and takes an unnecessarily large amount of time. This work can take months, whereas automating it would only require the examiner to press a button when approving the thesis, subsequently a computer can complete the rest of the process. Therefore, the main problem that this thesis project will try to solve is “How can approved student theses submitted via Canvas be automatically entered into DiVA?”.

1.3 Purpose

The purpose of this bachelor degree project is to design, implement, and evaluate a system to automate the entry of an approved thesis into DiVA. The requirements are quite similar regarding parsing data from a submitted document and filling in fields to create an event in the university’s calendar system. Therefore, the project will also try to provide the fundamental elements to automate such an event-creation to announce a student’s oral presentation. Thus, the repetitive task of extracting information from theses submitted via Canvas and using this information to publish the theses into DiVA or create Calendar events will be done automatically as soon as the appropriate button is pushed (by the responsible person).

The solution is to use automation. This automation will be thoroughly presented and described in sufficient detail for others to utilize the proposed solution or adapt it to a similar need[*]. The methods used for solving the problem will be evaluated and compared to existing methods (if any). The algorithm developed for this automation will be presented. A number of tests will be carried out to demonstrate the correctness and consistency of the results of applying the algorithm. Moreover, an estimate of the time saved with the introduction of this automation will be made. This saved time can be used for other tasks that actually require human interaction (for example, better supporting the advising of students).

1.4 Goals

The goal of this degree project is to provide the fundamental elements to automate the process of taking a (1st or 2nd cycle) thesis submitted via Canvas and entering it into DiVA or generating a Calendar event for an oral presentation. This goal has been divided into the following two sub-goals:

1. Once an examiner has scheduled an oral presentation, the proposed extension to Canvas will automatically extract the relevant information needed to create a Calendar event for a given degree project presentation based upon the submitted beta draft and the time and place of the scheduled presentation.
2. Once an examiner has approved a thesis submitted via Canvas, the relevant information will be extracted from the thesis itself and combined with other data that is available in Canvas to automate the full process of publishing theses via DiVA.

Achieving the above sub goals should provide greater efficiency than the current manual process for theses publication and oral presentation event creation.

1.5 Research Methodology

The research method that this thesis will use is qualitative research. Qualitative research means that this research is primarily exploratory research [1]. The qualitative research that is carried out for this thesis project will focus on understanding the reason, opinions, and motivation [1] for the structure of the data inside a Portable Document Format (PDF) file and the methods to insert and extract data, to and from the Canvas LMS. In particular, we need to know how to parse a PDF document in order to extract the relevant data (such as title, abstracts, and keywords). This action will be followed by generation of a cover for the thesis, as well as combining the front and back covers with the thesis. Eventually, research will be carried out to check that the data that is to be automatically entered into DiVA based upon the extracted information is correct and consistent. This correctness and consistency will be compared to data previously manually entered into DiVA.

[*] For example, there is a need to take the so-called "spikblad" for a licentiate or doctoral defense and automatically create a calendar event for the student's presentation. This one page announcement of a defense is formally published and entered into DiVA several weeks before a defense. Note that unlike 1st and 2nd cycle degree projects the two different types of 3rd cycle dissertations are published *before* the oral defense.

This thesis project is generally based upon parsing information from documents and inserting the extracted data into the relevant fields of records in other systems, hence the overarching goal is connecting what are today separate silos. The details of this parsing and extracting will be described later in the thesis.

An implementation choice is which programming language will be used and what algorithm is best to extract the data from the relevant source. In this context, “best” can be evaluated in terms of both development efficiency as well as run-time efficiency. Development efficiency needs to be taken into account so that the project team has enough time to complete the project as much as possible. Run-time efficiency is of course vital because the whole purpose of the project is providing efficiency.

The code provided by previous work (described in Chapter 2) is written in python; hence it would be simpler to implement the algorithm for this project if it too were written in python. Python is an interpreted high-level programming language for general-purpose programming [2]. How the algorithm is implemented will also depend on the Canvas Application Programming Interface (API) and how we will interact with DiVA.

1.6 Delimitations

The Connecting Silos project aims to provide the fundamental means of automating thesis publication into DiVA as well as creation of the respective presentation calendar event. Automatic publication into DiVA would require accessing the DiVA API which is based on Fedora Repository. Also, in order to create a calendar event on KTH’s Calendar system, Polopoly API needs to be used. However, neither the DiVA API, nor the Polopoly API were accessible by the project team’s members. Thereby, some limitations were imposed on the scope of the project, especially with regards to the development of the project. Apart from resource accessibility, the timespan of the project defined additional development limits for the project.

As the DiVA API was not accessible to the project team, the project will terminate its automation of thesis publication by generating a MODS XML file (described in Section 2.5.1). This file is a MODS XML file in order for it to be importable by DiVA. The project will thus, leave the actual insertion into DiVA, using the MODs file provided by Connecting Silos, for future work. The MODS file will contain most of the information that is necessary for DiVA to complete a thesis publication. However, some information that is not necessarily required for completing the publication, but that is useful to have[*] will be generated by future work. An example of data not available in the currently provided MODS file is the language of the title of the thesis, as this cannot be reliably extracted because it is not specifically identified in any of the data sources. Therefore, an algorithm is required to determine the title language based on the language of the thesis; however, thisis left to be done in future work.

[*] For example, the ORCID identifiers for the examiner and adviser(s).

Unfortunately, the project's limited duration did not allow time to pursue extracting further information. However, the project does provide a means for the automation of some work, but needs to be completed in future work.

The Calendar event creation in the KTH Calendar requires accessing the Polopoly API and using its functionalities for event creation. However, this API was inaccessible to the team members throughout the project's timespan. Therefore, the information required to create the corresponding presentation calendar event for each thesis is provided as a JSON string. This JSON string can be used in future work or perhaps by administrative staff who have access to the Polopoly API in order to complete automation of calendar event creation.

1.7 Structure of the thesis

The following thesis will begin by providing sufficient background knowledge of the tools, libraries, external packages, etc. that are fundamental to implementation of the project. This information presented in Chapter 2 was acquired by project team members through preparatory research and study at the beginning of the project.

Chapter 3 gives an overview of the project's timeline from the preparatory stage to implementation, evaluation, and completion of the project. The outline of the project in this chapter is quite a general overview and the development phase is described in detail in Chapter 4. Chapter 3 also presents the testing methodology of the system (see Section 3.4).

Chapter 4 provides detailed knowledge about the implementation and development of the system. This chapter begins by defining the main sub-tasks of the project and further how each of these tasks were completed. The chapter also takes a step back from the sub-tasks and explains how these separate tasks were then put together to build the entire interconnected system. The chapter concludes by presenting how the system is to be used along with a brief description of minor tools and packages as used.

A discussion of the achievements of the project is in Chapter 5. In this chapter, the test results are discussed to confirm the validity and reliability of the system.

The thesis concludes in Chapter 6, with a discussion of the benefits and drawbacks of the prototype, specifically as compared to the manual process. Moreover, an outline of the usage limitations that the system imposes on users is presented. This is followed by potential developments that can be done. The chapter concludes by discussing the social and economic aspects of the project.

2 Background

The Connecting Silos project is meant to simplify and automate the 1st and 2nd cycle degree project administrative processes. This chapter begins with a rough description of the workflow in a 1st and 2nd cycle degree project to clarify the overall administrative process for a degree project. As the project will be tested and developed based on the current version of the Canvas LMS running at KTH, DiVA, and the Polopoly platform (used for Calendar events), these three systems are introduced in Sections 2.4, 2.5, and 2.6. This will be followed by introductions to details of these three systems, how one can interact with them, and background information about some of the tools to be used. Additionally, Section 2.7 describes the KTH APIs that will be used to extract some information about the examiners and supervisors for insertion into DiVA. Following this a number of other helpful technologies will be described. Finally, the chapter ends with a section on related work section 2.10, analysis of reliability and validity (Sections 2.11 and 2.12), and concludes with a discussion of the material presented in this chapter (Section 2.13).

2.1 Workflow of degree project at KTH (1st and 2nd cycle)

A Bachelor's thesis is the result of a first cycle degree project. This degree project is done as 15 credits (in the European Credit Transfer and Accumulation System). The project is typically done at the end of the last study year of students in the first cycle [3], [4].

As Figure 2-1 demonstrates, in order to be able to register for the degree project course, students need to meet certain requirements of the university. Once they have met these prerequisites, then depending on the cycle and program of study of an individual student, the student will register for a particular thesis course under different course codes. This student will be added to a Canvas course for degree projects of a given cycle (in this work we will only consider 1st and 2nd cycle degree projects). Next the student will be placed by an administrator (or potentially automatically) into a section within this course depending on their education program. During the first cycle, the student will proceed with their thesis project either in a group of two people or alone [3], [4]. Note that 2nd cycle students do their degree project individually.

Once the student has been added to the Canvas course, then each student or group of students is required to submit a project proposal via Canvas. This project proposal describes the proposed project and methodology that will be used in the project. The student may also fill out a survey to provide additional information about the proposed degree project (such as suggested examiners, supervisors, whether they give their approval for the full text of the thesis to be published via DiVA, etc.). If the proposed project meets the requirements of a given program, then the Program Administrator will assign an examiner and supervisor for the student based on the topic of the project proposal. The potential examiner will be recorded in the Canvas Gradebook (for details see Section 2.4.3). This examiner will read the project proposal and decide if the proposal has sufficient quality to be accepted and if they are a suitable examiner. If the proposal has low quality, then the examiner will ask the student to improve the proposal based on the examiner's feedback. This process continues until the proposal is accepted by the examiner. Additionally, the assigned examiner may reject being the examiner for a particular student (typically because the topic is out of the scope of this examiner's expertise, the examiner is already overloaded, there is a conflict of interest, etc.). In this case, the Program Administrator will assign another examiner to this student [3], [4]. There are two potentially automated parts of this workflow and they have been indicated in Figure 2-1. The Connecting Silos project will support the automation for these 'potentially automate' parts of the workflow.

At the completion of the degree project students submit a draft thesis and present it in a seminar (referred to as an oral thesis defense or presentation). It is desirable to announce these presentations via Calendar events in Polopoly. Following the public presentation, the thesis is typically revised and the final version submitted via Canvas to the examiner for evaluation. If the thesis is accepted then a grade is assigned by the examiner and the thesis should be entered into DiVA with a unique document number (at KTH this is called a TRITA number). The information in the thesis will be used to generate the front and back cover pages of the thesis via the Book Cover Generator (see Section 2.7.2).

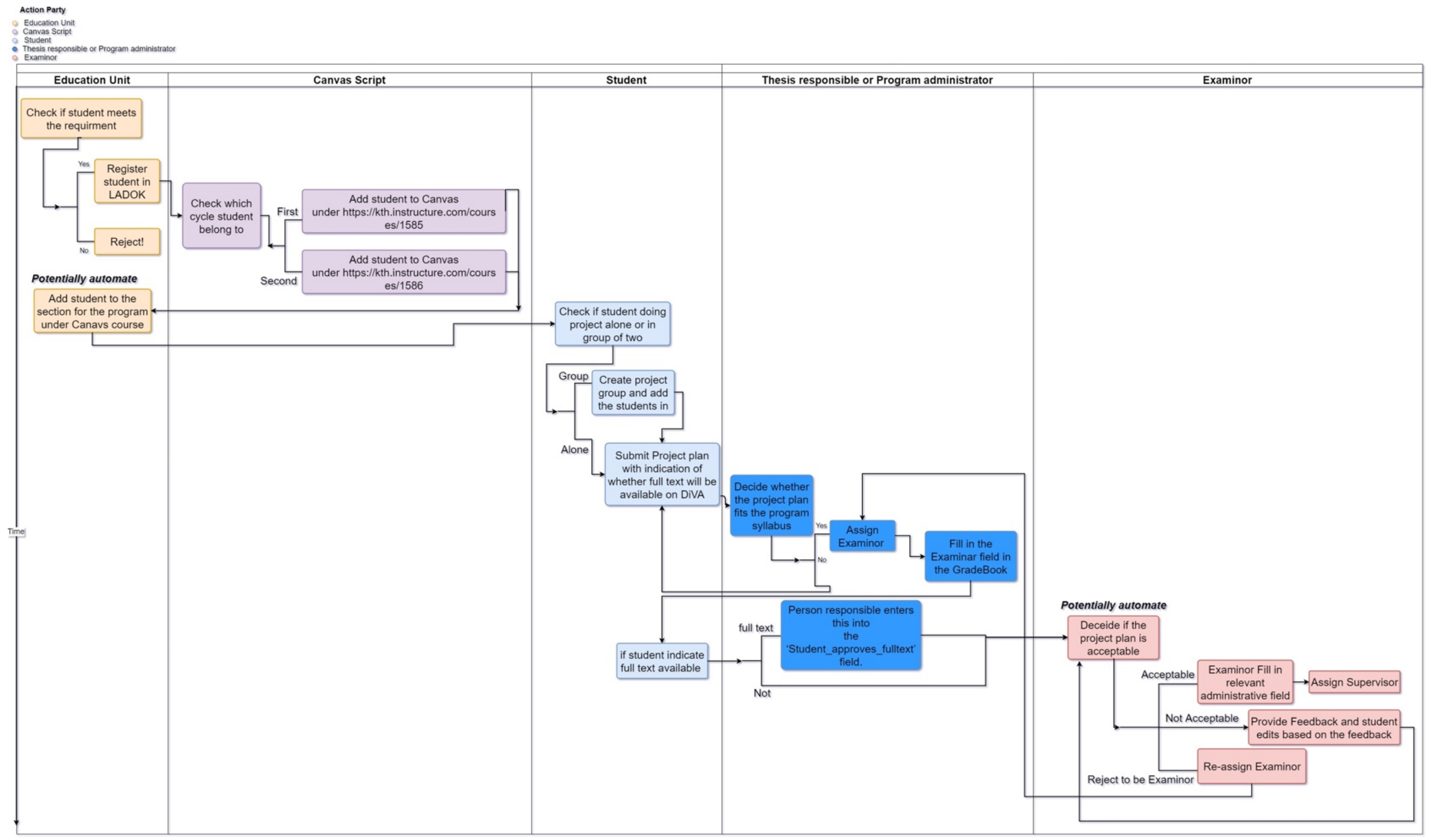

Action Party
Education Unit
Canvas Script
Student
Thesis responsible or Program administrator
Examinor
Education Unit
Canvas Script
Student
Thesis responsible or Program administrator
Examinor
Check if student meets the requirment
Yes
Register student in LADOK
No
Reject!
Check which cycle student belong to
First
Add student to Canvas under https://kth.instructure.com/courses/1585
Second
Add student to Canvas under https://kth.instructure.com/courses/1586
Potentially automate
Add student to the section for the program under Canavs course
Check if student doing project alone or in group of two
Group
Create project group and add the students in
Alone
Submit Project plan with indication of whether full text will be available on DiVA
Decide whether the project plan fits the program syllabus
Yes
Assign Examinor
No
Fill in the Examinar field in the GradeBook
if student indicate full text available
full text
Person responsible enters this into the 'Student_approves_fulltext' field.
Not
Potentially automate
Deceide if the project plan is acceptable
Acceptable
Examinor Fill in relevant administrative field
Assign Supervisor
Not Acceptable
Provide Feedback and student edits based on the feedback
Re-assign Examinor
Reject to be Examinor
Time

Figure 2-1: Workflow for Degree Project[*]

[*] The diagram is based on the document Gerald Q. Maguire Jr., "Facilitating Degree projects & Connecting Silos", School of Electrical Engineering and Computer Science (EECS), KTH Royal Institute of Technology, 25-Mar-2018

2.2 Jira Cloud

Jira is an agile team management program that is created by Atlassian [5]. The purpose of Jira is to create a remote development workflow and real-time collaboration by moving the team backlog into the cloud [5]. Jira is also a good tool for planning and tracking the progress of a project, without spending extra time and human resources. Most of the calculation and process is done automatically. For example, by using 'finish the sprint' function, Jira will automatically generate a burn down chart for the previous sprint. In the Atlassian catalog, Jira is usually called Jira Software. The reason that Jira is called 'Software' is because Jira can either be run as individual program with an installation sequence in different operating systems or as a rental service in the Jira cloud [5]. At the enterprise level, a company usual buys perpetual license in order to reduce the cost of this software. In the Connecting Silos project the project team uses Jira Cloud to avoid the time-consuming effort to locally deploy Jira.

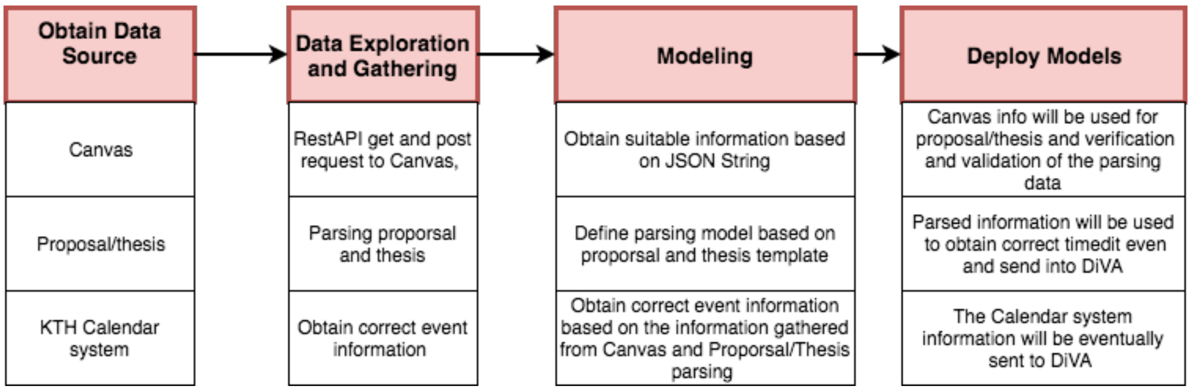
Jira supports different kinds of agile team management frameworks. In the Connecting Silos project, the project team uses a scrum for team management. Jira is used as a remote scrum board for seamless team communication. Additionally, Jira has been used to plan and track the progress within the project.

2.3 Data Mining

Data mining utilizes a computer algorithm to discover patterns in large data sets by using machine learning, statistics, and database systems [6].

As depicted in Figure 2-2, the data mining process can be divided into four steps: obtaining data sources, data exploration/gathering, modeling, and deploying models [7].

Figure 2-2: The process of Data Mining [7]


Obtain Data Source
Data Exploration and Gathering
Modeling
Deploy Models
Canvas
RestAPI get and post request to Canvas,
Obtain suitable information based on JSON String
Canvas info will be used for proposal/thesis and verification and validation of the parsing data
Proposal/thesis
Parsing proporsal and thesis
Define parsing model based on proporsal and thesis template
Parsed information will be used to obtain correct timedit even and send into DiVA
KTH Calendar system
Obtain correct event information
Obtain correct event information based on the information gathered from Canvas and Proporsal/Thesis parsing
The Calendar system information will be eventually sent to DiVA

Data sources are the source files that are to be mined. These data sources will be mined by extracting specific data from the sources [7]. For the Connecting Silos project, the main data source is the PDF version of a thesis proposal or the thesis itself[*]. By means of data exploration/gathering, the pattern in the data sources is derived and the resulting extracted data is transformed into a human or machine readable and-process able format. The data exploration/gathering method that Connecting Silos project will use is PDF parsing. The developer needs to specify a model to use in the data mining process. The model can be a pattern in the data, specific keywords to look for, etc. Finally, the deploy model will be created by the four steps in the data mining process and the extracted data will be used as input to other software [7].

The data that is mined from the data sources has to remain consistent. In the Connecting Silos project, this data consistency will exploit an ORCID (Open Researcher and Contributor ID) [8], KTH ID, or other identification system. ORCID is a persistent digital identifier that is used to identify authors and contributors [8]. KTHID is a digital identification that is used inside KTH for students, faculty, and administrative staff [9]. In the Connecting Silos project, the KTHID will be used to ensure data consistency regarding the examiner, supervisor, and student. The ORCID identifier will also be added to the DiVA records when known[†].

2.4 Canvas Learning Management System

This subsection first introduces what a learning management system is, followed by the specific platform (Canvas) that is used at KTH, the Canvas Gradebook which links to all of a user's submissions and other information, and finally a tool (Speedgrader) to assist teachers in reading and grading submitted assignments.

2.4.1 Learning Management System

A Learning Management System (LMS) is an information technology (IT) solution that provides support for administration, documentation, tracking, reporting, and delivery of educational courses or training programs [1]. At KTH , an LMS is used to deliver course material, share documents, manage assessments and facilitate communication between students, faculty, and other education staff [10].

KTH adopted Canvas as their LMS starting in period 1 of 2017 [10]. Before Canvas, KTH used a variety of different systems, such as Bilda (known commercially as PingPong and a product of Ping Pong AB), KTH Social (a locally developed tool), and Daisy (an LMS developed by the Department of Computer and Systems Sciences (DSV), a department that was operated jointly for many years by both Stockholms Universitet and KTH) as LMSs [10].

[*] As an extension, it should also be possible to mine DOCX files as they are well structured XML files.

[†] It is possible using the KTH API (https://www.kth.se/api/profile/v1/user/*user_name*) to get a user's ORCID identifier.

2.4.1.1 Learning Tools Interoperability

Learning Tools Interoperability (LTI) is a LMS extension standard that IMS Global Learning Consortium created to enable the full potential of a LMS [11]. By using an LTI application in an existing LMS, schools can ensure better teaching and learning experiences by exploiting the functionality of an external tool.

Figure 2-3 shows how an LTI Interface is used as communication channel between a target LMS and an LTI application. In order to be able to install an LTI application in a target LMS, the user needs to send a sequence of requests to the LMS. The purpose of this sequence of requests is to inform the target LMS that there is a LTI application that needs to set up the LTI interface and to initiate communications with the LMS. These sequences of requests can be one or more GET, PUT, or POST requests - depending on the LTI application and the settings in the LMS. The benefit of using LTI comes after the install request. Every time a LTI application starts up, the LTI application will inform the LTI interface that a 'LTI application with external tool id A is starting up. The LTI Interface will then request the LMS to access its database and provide a response that the LTI interface can send as a response to the LTI application. The response includes the fields that the LTI application's main class requires as input parameters. The content of the response can be a course id, assignment id, and so on. The beauty of LTI applications is due to the response that the LTI application received during startup, the user of the software does *not* need to manually input the same data again and again. Some sensitive data can also be avoided as user input, for example a password. Using Oauth2, the user of a LTI application will only get those permissions that are sufficient for the application's workflow and without the user needing to log into the system and give a password.

After the LTI application successfully starts up with the feedback provide by LTI Interface and LMS, the LTI application can request or modify data in the target LMS database via the LTI Interface. For example, an LTI application can automatically grade an assignment and send a grade back to the LMS database as soon as a student hands in their assignment.

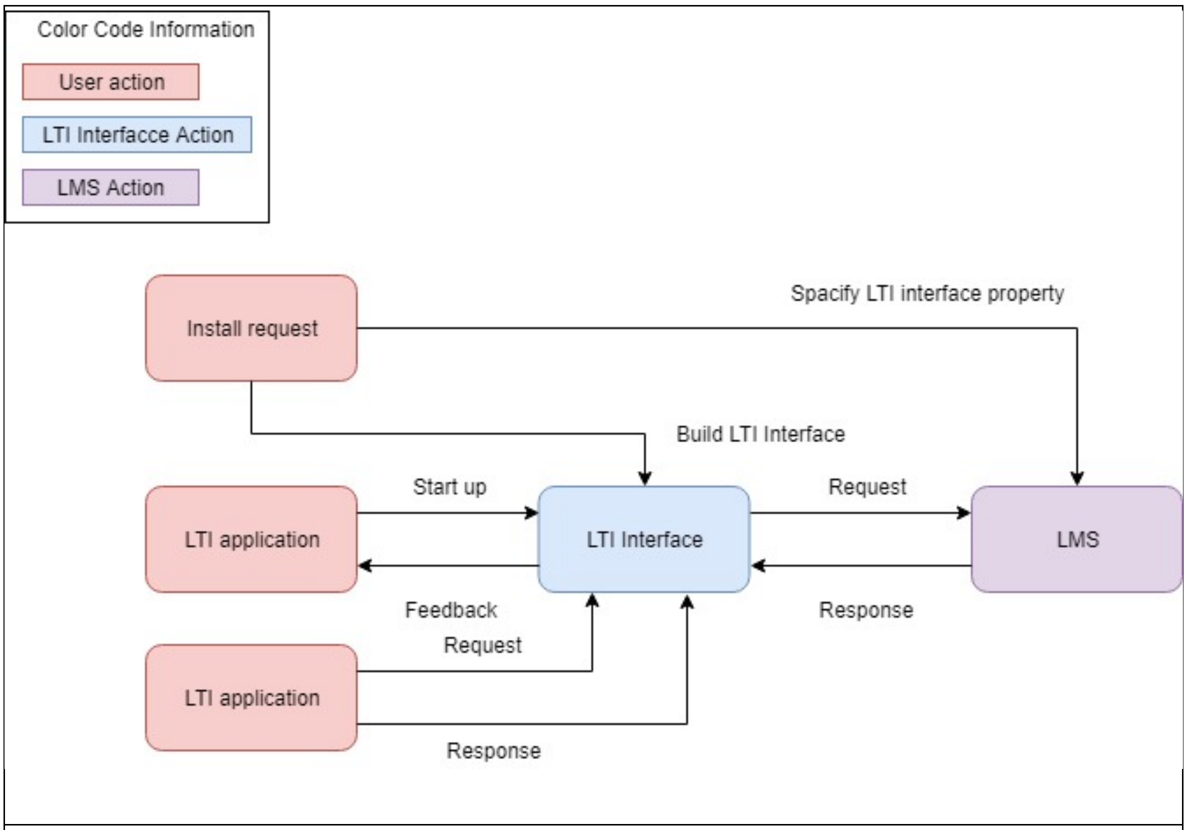


Figure 2-3: LTI Communication Architecture

2.4.2 Canvas Platform

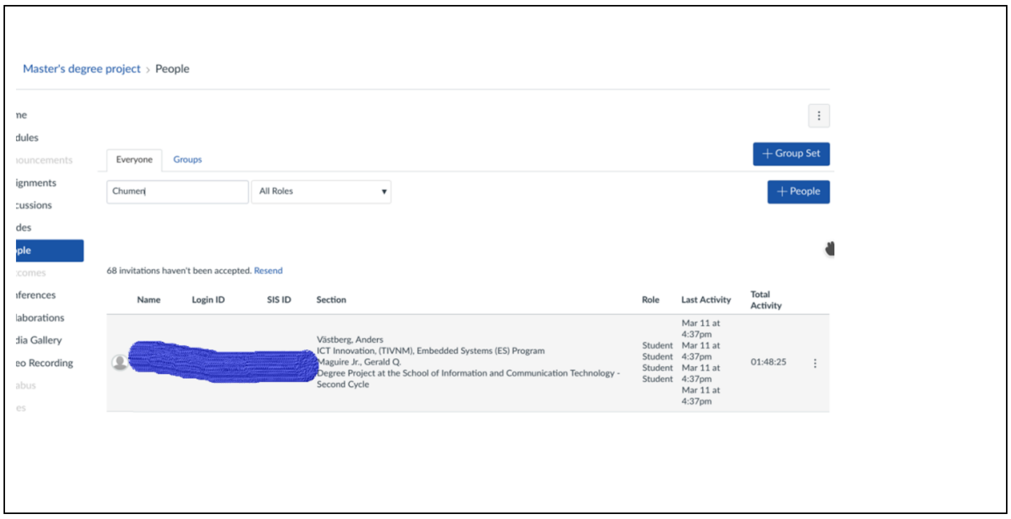
Canvas is a cloud-based and user-focused LMS developed by Instructure, Inc.* and widely used in higher education institutions worldwide [12]. Via Canvas, students can find course material and guides provided by their teachers; keep track of courses, news, and events via an announcement board; keep in touch with teachers and fellow students through discussion forums and instant messaging; hand in assignments; and take course quizzes and exams [13].

Canvas supports a Restful API for external applications to access and modify data in the Canvas database [14]. The Connecting Silos project will achieve its goal through a combination of GET/POST requests via the Canvas API. This API can be accessed via an OAuth2 Authentication Token that is generated for each user on request. When using this token, the application has all of the same permissions as those associated with the user's Canvas account. Using this API, it is possible to access and move data between Canvas and other programs through for example the Canvas Files API†.

* In europé: Instructure Global Ltd.

† https://canvas.instructure.com/doc/api/files.html

In addition to the Canvas Restful API, Canvas also implements a Content Management System (CMS). Via the Canvas web user interface, a user can create content. For example, a user could create a survey that can be used as form (to replace the existing thesis application form) to collect data from the student. Subsequently this information could be used to separate the students into additional sections (beyond the initial section based upon the student's education program). In the case of a degree project, each student is added to a section based upon the student's Examiner and (Academic) Supervisor. An example of a student in several such sections is shown in Figure 2-4. The use of sections enables an examiner, supervisor, or program administrator to easily see a view of the gradebook that shows only their students.



Chumer

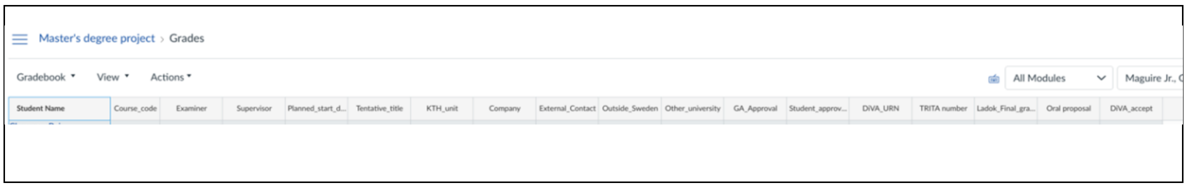
68 invitations haven't been accepted. Resend

Name	Login ID	SIS ID	Section	Role	Last Activity	Total Activity
[redacted]			Västberg, Anders ICT Innovation, (TIVNM), Embedded Systems (ES) Program Maguire Jr., Gerald Q. Degree Project at the School of Information and Communication Technology - Second Cycle	Student Student Student Student	Mar 11 at 4:37pm Mar 11 at 4:37pm Mar 11 at 4:37pm Mar 11 at 4:37pm	01:48:25

Figure 2-4: Example of a student placed in several sections

2.4.3 Canvas Gradebook

The Canvas gradebook contains information about students' submissions for assignments, their grade for each assignment, and other information. In conjunction with the degree project course, the Canvas gradebook has been extended with additional custom fields to store information relevant to a degree project. Figure 2-5 shows an example of the custom columns that have been added to a gradebook for the 2nd cycle Master's degree project course. In this case the custom columns are those shown to the right of the column "Student Name". Some of the values stored in these columns will be used later to generate the cover for the thesis and to record the document number (TRITA number) assigned to the thesis, the DiVA URN that is assigned once the DiVA system has accepted a thesis, information about whether the student has given their permission for the full text to be published via DiVA, etc.



Student Name	Course_code	Examiner	Supervisor	Planned_start_d...	Tentative_title	KTH_unit	Company	External_Contact	Outside_Sweden	Other_university	GA_Approval	Student_approv...	DiVA_URN	TRITA number	Ladok_Final_gra...	Oral proposal	DiVA_accept

Figure 2-5: Example of the custom columns that have been added to a gradebook for the 2nd cycle Master's degree project course

The full set of custom columns includes: Course_code, Examiner, Supervisor, Planned_start_date, Tentative_title, KTH_unit, Company, External_Contact, Outside_Sweden, Other_university, GA_Approval, Student_approves_fulltext, DiVA_URN, TRITA number, Ladok_Final_grade_entered, Oral proposal, and DiVA_accept.

2.4.4 Speedgrader

Canvas Speedgrader allows a user to view and grade student assignments in one place with a simple point scale and complicated rubric [15]. Speedgrader enables the teacher to grade a submission and either directly add markup to the submission or use an external tool to do markup after downloading the submission as a file. Additionally, the user can easily provide (written or multimedia) comments to the student using Speedgrader. Speedgrader allows the user to grade, track, and provide feedback on the assignment in the Canvas without requiring additional tools [15]. In this project, Speedgrader will be used as an interface by examiner for entering feedback on a student's submission.

2.5 DiVA Platform

DiVA (Digitala vetenskapliga arkivet) - Academic Archive Online, is a publishing system for research and student theses and a digital archive for long-term preservation of publications [16]. Currently, there are ~47 different universities and research institute in Sweden that use DiVA for publication registration [16]. At KTH, DiVA has been used to register researchers' publications [17]. These publications include doctoral dissertations, licentiate theses, reports, students' theses, and other research publications where at least one author is from KTH. The purpose of the DiVA platform is to make publications visible and accessible via the DiVA platform [17]. DiVA also makes publications accessible by exchanging data with other database services, such as the Swedish national publication database (SwePub), Google Search Engine, and Google Scholar [17].

Unfortunately, the DiVA API is not accessible to the project team. This issue will be solved by generating a MODS import file for subsequent processing — as DiVA supports the MODS format for modifying or creating records of publications. This file format will be described in Section 2.5.1. Additionally, there may be a possibility to insert records into the DiVA database using an existing API, as DiVA is based upon a Fedora Repository.

2.5.1 MODS

Metadata Object Description Schema (MODS) is a schema for bibliographic elements that can be used for several different purposes, especially in a library system [18]. The MODS schema language is encoded using the Extensible Markup (XML) format. MODS files can be generated by various tools to provide mandatory input data [19]. DiVA supports MODS as one of its publication import formats, thus enabling the automation of the workflow from other systems to DiVA. As MODS is fundamentally a XML file, it will be referred to as MODS file and or MODS XML file interchangeably throughout the thesis.

2.5.2 SwePub MODS

The actual version of MODS used by DIVA is specified in [20]. The details of the DiVA records can be found in [21]. Additionally, U. S. Library of Congress, MARC Code List for Relators: Term Sequence [22] specifies the codes used for describing the thesis examiner, supervisor, etc.

2.5.3 Bibutils software

Gerald Q. Maguire Jr. has adapted Chris Putnam's bibutils v6.2 software [23] to work with DiVA, specifically to read MODS files produced by DiVA and to write BibTeX files[*] . This software provides a rich variety of functions for working with MODS records. Additionally, he has written a set of tools to extract information in MODS format from DiVA[†]. For example, the program diva-get_bibmods_theses_school.py gets the MODS entries for theses for a whole school within KTH for one or more years. This can be used to get the thesis meta data and URLs to extract a set of full-text theses for use in testing the accuracy of the tool being developed in this degree project.

2.6 Polopoly

Polopoly is a content management system at KTH which makes edit management and content publishing on the KTH website more natural [24]. Polopoly also has a function that can be used to publish a calendar event into a specific calendar inside a specific user group. The event creation function in Polopoly is a useful function when it comes to announcements for thesis presentations. As described previously, the project should create a means to automatically create a calendar event in Polopoly based on the information extracted from the PDF version of a thesis draft in Canvas and the place and time for the oral presentation.

[*] See https://github.com/gqmaguirejr/bibutils_6.2_for_DiVA

[†] See https://github.com/gqmaguirejr/DiVA-tools

2.7 KTH

KTH Royal Institute of Technology Sweden has developed a number of tools that are relevant to this degree project. Specifically, a tool to get information about users and an application to create covers for theses. Each of these are described below.

2.7.1 KTH APIs

The KTH APIs give public access to widely used KTH systems. With the help of this API one can obtain the information about courses and program catalogues, KTH Social, KTH profiles, WebTex (for web publishing), KTH directory, KTH web publish system, and KTH places. Within the Connecting Silos project, the KTH APIs' profile access function has been used to obtain detailed information about the student, examiner, and academic supervisor. For example, the API end point https://www.kth.se/api/profile/v1/user/maguire returns the following (note that the user's research related identifiers are highlighted):

```
{
 "kthId": "u1d13i2c",
 "username": "maguire",
 "homeDirectory": "\\\\ug.kth.se\\dfs\\home\\m\\a\\maguire",
 "title": {
  "sv": "PROFESSOR",
  "en": "PROFESSOR"
 },
 "streetAddress": "ISAFJORDSGATAN 26",
 "emailAddress": "maguire@kth.se",
 "telephoneNumber": "",
 "isStaff": true,
 "isStudent": false,
 "firstName": "Gerald Q",
 "lastName": "Maguire Jr",
 "city": "Stockholm",
 "postalCode": "10044",
 "remark": "COMPUTER COMMUNICATION LAB",
 "lastSynced": "2018-05-27T20:54:02.000Z",
 "researcher": {
  "researchGate": "",
  "googleScholarId": "HJgs_3YAAAAJ",
  "scopusId": "8414298400",
  "researcherId": "G-4584-2011",
  "orcid": "0000-0002-6066-746X"
 },
 "courses": {
  "visibility": "public",
  "items": [
   {
    "code": "II2202",
    "koppsUrl": "https://www.kth.se/student/kurser/kurs/II2202",
    "courseWebUrl": "https://www.kth.se/student/kurser/kurs/II2202?l=sv",
    "roles": [
     "courseresponsible",
     "teachers",
     "examiner"
    ],
    "title": {
     "sv": "Forskningsmetodik och vetenskapligt skrivande",
     "en": "Research Methodology and Scientific Writing"
    }
```

```
       },
    ...
     },
     "worksFor": {
      "items": [
       {
        "key": "app.katalog3.J.JF",
        "path": "j/jf",
        "location": "ELECTRUM 229, 16440 KISTA",
        "name": "KOMMUNIKATIONSSYSTEM",
        "nameEn": "DEPARTMENT OF COMMUNICATION SYSTEMS"
       },
       {
        "key": "app.katalog3.J.JF.JFB",
        "path": "j/jf/jfb",
        "location": "ELECTRUM 229, 16440 KISTA",
        "name": "RADIO SYSTEMS LAB",
        "nameEn": "RADIO SYSTEMS LAB"
       }
      ]
     },
     "links": {
      "visibility": "public",
      "items": [
       {
        "url": "http://people.kth.se/~maguire/",
        "name": "Personal web page at KTH"
       }
      ]
     },
     "description": {
      "sv": "<p>Om du verkligen vill kontakta mig eller hitta information om mig, se min hemsida:&nbsp;<a
href=\"http://people.kth.se/~maguire/\">http://people.kth.se/~maguire/</a></p>\r\n",
      "en": "<p>If you actually want to contact me or find information related to me, see my web
page:&nbsp;<a
href=\"http://people.kth.se/~maguire/\">http://people.kth.se/~maguire/</a></p>\r\n\r\n\r\n\r\n\r\n",
      "visibility": "public"
     },
     "images": {
      "big": "https://www.kth.se/social/files/576d7ae3f2765459470e7b0e/chip-identicon-
52e6e0ae2260166c91cd528ba0c72263_large.png",
      "visibility": "public"
     },
     "room": {
      "placesId": "fad3809a-344b-4572-9795-5b423e0a9b2a",
      "title": "4478"
     },
     "socialId": "55564",
     "createdAt": "2006-01-09T13:13:59.000Z",
     "pages": [],
     "avatar": {
      "visibility": "public"
     },
     "isAdminHidden": false,
     "acceptedTerms": true,
     "defaultLanguage": "en",
     "visibility": "public"}
```

2.7.2 KTH Book Cover Generator

KTH's Book Cover Generator* is used to generate the front and back cover pages of a Bachelor or Master's thesis. The following information is needed to use the tool [25]:

- Cycle and number of credits of the degree project,
- Degree,
- Main field or subject of education degree,
- Title,
- Subtitle,
- Author(s),
- (Optional) Image to be used on the front page,
- School at KTH where the degree project was examined,
- Year, and
- TRITA number (as a unique document number).

Most of the information that is required by the Book Cover Generator can be obtained from thesis itself, from Canvas, and from the KTH user database. Eventually, the front and back cover will be combined with the approved thesis via an existing PDF modification tool (such as PyPDF2 described in Section 2.9.3).

2.8 PDF Parsing

The Portable Document Format (PDF) is a document format that is maintained by the International Organization for Standardization (ISO). PDF is a widely used standard for documents. Moreover, PDF is independent of the underlying operating system [26]. PDF supports links, buttons, form fields, audio, video, and business logic in a document's file [26]. To extract information from a PDF file, PDF Parsing is done to parse and analyze PDF documents [27]. PDF Parsing tools can extract the desired raw data from PDF documents and in some cases, even extract data from a damaged PDF file. PDF Parsing is one means of data mining, then the information that is extracted by the PDF Parsing tool can be used by other systems.

Regarding the tools that will be used for parsing PDF - some prior work has been done by Elias Kunnas with his PDF Parsing tool called pdfssa4met [28]. Based upon this tool Gerald Q. Maguire Jr. wrote a program called kthextract to extract data from a thesis proposal or the thesis itself. Both tools are written in Python. The Connecting Silos project will implement its tools based on pdfssa4met and kthextract. More specifically, the project will further develop pdfssa4met and kthextract.

2.9 Python libraries and packages

A number of Python libraries and tools are used in this project. Some of the most important of these are described below.

* https://intra.kth.se/kth-cover

2.9.1 Python Selenium

Selenium is a browser automation plugin for python that simplifies the work flow for a developer with large number of repetitive tasks to perform on a specific website [29]. For example, Selenium can auto-fill in a form on a website based on a data set, set the profile for the browser, and click a button or link on the website. In order to be able to use Selenium, the developer needs to specify the path for the selected browser driver. For example, with Firefox one uses Geckodriver[30] and with Chrome ones uses ChromeDriver[29]. This plugin can be used for future work in order to complete the automation by filling in the Polopoly form to create an event or filling in DiVA fields for thesis publication.

2.9.2 Python Package Manager (pip and conda)

The Python Package Manager is a mechanism to import packages into a system's python library[31]. The most well-known package managers are pip and conda. The Python Package Manager in use varies depending on the Python version that is being used. For example, if one uses python 3.x.x for development, then the developer should use pip3 instead of pip. The same goes for conda. For the Connecting Silos project, python 2.7.14 is used for kthextract (see Section 2.10.2) and python 3.6.5 is used for access to Canvas. The Canvas Data Collection module frequently uses additional external packages. However, for kthextract only two external packages are used.

2.9.3 PyPDF2

PyPDF2 is a python coded PDF toolkit developed from the pyPdf project. It is currently maintained by Phaseit Inc. (http://phaseit.net/) [32]. PyPDF2 can create, extract, edit, merge, and encrypt & decrypt specific data from one or more PDF files [32, p. 2]. PyPDF2 will be used to add the cover pages created by the Book Cover Generator to the thesis. More specifically, PyPDF2 will be used to merge the thesis content with the front and back cover pages.

2.9.4 Other add-ons

The Connecting Silos project is also using other add-ons in addition to the ones mentioned above to maintain the project on the correct track. For example, the Connecting Silos project frequently uses the Python os plug in to manage the input and output files for each module. Some advanced file management functions such as move and delete cannot be done via an os plug in. In this case, we use Shutil a high-level file operations add-on for python file system management [33]. With the help of Shutil, the project has a more stable architecture and easier data validation process. The reason that Shutil makes the data validation process easier is by enabling the use of a cache folder. After data has been validated, all the data in the cache folder will be moved into an output folder instead of outputting the data directly into the output folder and validating it afterwards.

To be able to download all the proposals and thesis files at one time, Canvas allows API developers and users to download a zip-packaged file that contains all the proposals or thesis documents stored under the same course code. To decompress the zip file and make use of the PDF files inside the zip file, the Connecting Silos project use python's zipfile plug-in. Using this plug in, the project can further automate the thesis workflow

2.9.5 Django

Django is a python web framework focused on creating a clean and rapid development environment with fewer lines of code. By automating most of the connection and web platform process, Django allow developers to focus on development [34].The Django framework architecture is shown in Figure 2-6. In this figure, a user is interacting with a template layer which is the HTML and JavaScript front-end page that is shown in the browser. When a user performs a certain action to interact with the template layer, the template layer will trigger a certain function in the view layer. Inside the view layer is the application that a developer wrote. These applications can be a previously written standalone local or internet based python module or function. By using the Django framework, any python module or function can be changed to a web-based application. After the view layer is done processing the data, the view layer will either send the result back to the template layer or store it in a model layer. The model layer is an abstract layer for manipulating the data for the application running in the view layer. In other words, the model layer is the interface between an application and a database. This model layer can be written in different database languages. The model layer allows a view layer to either update information in the database or request information. Without changing any settings, Django uses SQLite as its default database language. If required the user can change the database's location and settings in the file 'settings.py' to use a personal database [34], [35].

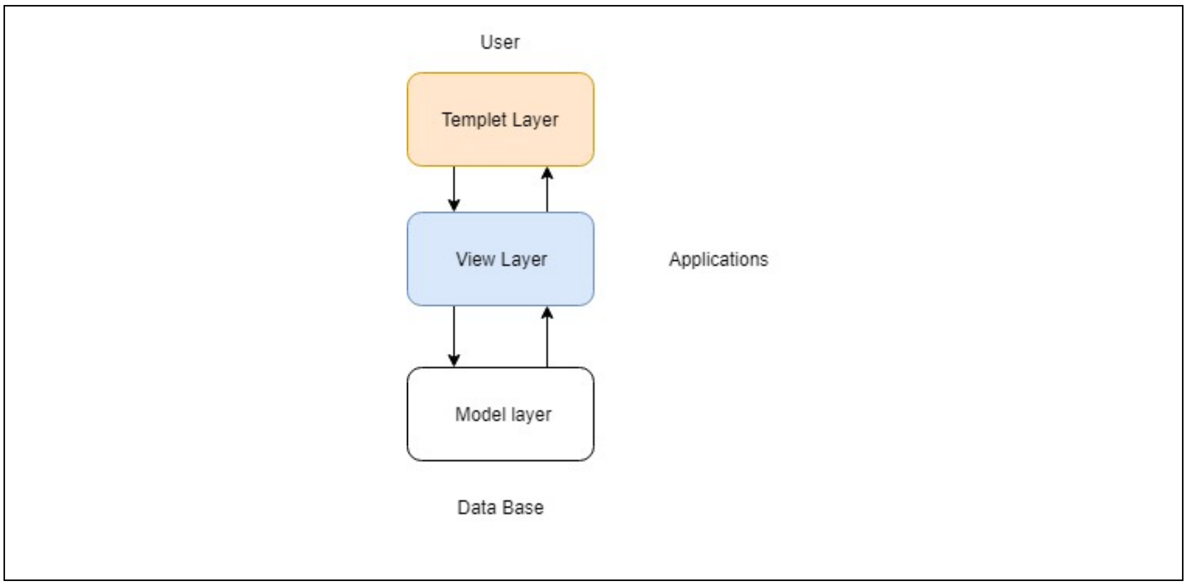


Figure 2-6: Django framework architecture

Django also supports a number of different add-ons. Some of these add-ons need some manual adjustment in ‘settings.py [35]. An example of a modified setting for a Django add-on is shown in Figure 2-7. Under Installed_apps, the folllwiong applications are always present by default because these add-ons are the basic components that Django needs to operate properly: 'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', and 'django.contrib.staticfiles'. In Figure 2-7, there are several additional add-ons: 'oauth2_provider', 'polls', ‘django_extensions', and 'trees' . The application 'polls' is written in view layer, because the other modules outside the view layer call polls (for example, to access the database), therefore ‘polls’ is also present as an add-on here.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'oauth2_provider',
    'polls',
    'django_extensions',
    'trees'
]
```

Figure 2-7: Django installed application setting example

When it comes to python development, compatibility is always an issue. As python 2.7.14 and python 3.x.x are not compatible with each other, the Django framework versions 1.11 and 2.0 also have large differences. Since the Connecting Silos project main class is based on python 2.7.14, but some modules are based on python 3.x.x, the Django version that used here is 1.11 to maintain the greatest compatibility.

The basic structure of Django Framework after initialization is shown in Figure 2-8 below. Inside the 'polls' folder lies the application that the developer has created. The 'mysite' folder is the project's management folder. The functionality of the python files in the 'mysite' folder is to configure the project, the properties of the client-server based application, and external add-ons. The most important python file here is the 'settings.py'. By configuration of 'settings.py', a user can change the properties and configuration of the project. For example, if the developer only wishes a certain individual to access the project, the developer can set the 'Allowed_Host' to that individual's specific IP address.

```
├── mysite
│   ├── __init__.py
│   ├── __init__.pyc
│   ├── settings.py
│   ├── settings.pyc
│   ├── urls.py
│   ├── urls.pyc
│   ├── wsgi.py
│   └── wsgi.pyc
├── polls
│   ├── admin.py
│   ├── __init__.py
│   ├── migrations
│   │   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
│   └── url.py
└── manage.py
```

Figure 2-8: Architecture of a Django project [36].

Figure 2-10 shows the directory architecture of the Connecting Silos project under the Django framework. The connection class is specified in the 'views.py' file. Each webpage has an HTML front-end that is a button linking the page to a function in 'view.py'. The fule 'views.py' contains the back-end action that each webpage does before sending out the HTTP page to the user. The corresponding front end of each view lies inside the 'url.py'. For example, the code in Figure 2-9 creates a front end with the link 'http://<server_ip>/<polls>/ retrive_session_baseon_id' that is runs the backend function retrive_session_baseon_id() in 'views.py'. If developer wishes to have user-interaction through an HTML front-end page, the HTML page will be inside the 'templates' folder shown in Figure 2-10. The template can be called in 'views.py' and be used to return the desired front-end HTML page.

```
url(r'^retrive_session_baseon_id$',
views.retrive_session_baseon_id,
name='retrive_session_baseon_id'),
```

Figure 2-9: Front-end creation example code

The code in ‘manage.py’ is used for running different management operations for the project. The most frequent command for ‘manage.py’ is ‘runserver <ip_address>:<port>’. The ‘runserver’ command will start a test server and provides standard output and test log for the developer to test an application.

```
.
├── django_heroku
├── mysite
├── polls
│   ├── Documentation
│   ├── download_targz
│   ├── ffdriver
│   ├── library
│   │   ├── Canvas-master
│   │   └── pdfssa4met_original
│   ├── migrations
│   ├── output
│   │   └── parse_result
│   ├── Source
│   ├── src
│   │   ├── Canvas
│   │   ├── Diva
│   │   ├── KTH
│   │   ├── mods
│   │   └── parse
│   ├── templates
│   │   └── polls
│   └── venv
│       ├── bin
│       ├── include
│       ├── lib
│       └── local
├── polls_docs
└── test
    └── testproject
```

Figure 2-10: Django project directory architecture

2.10 Related work

The project builds on few related works that have been done previously. The first one is kthextract. This python program provides general functionality to extract fixed-paged content and flexible paged content. Based on kthextract, the Data Extraction module will be adapted to the KTH School of Electrical Engineering and Computer Science (EECS) 2018 template with a few additional functions that will be added by the Connecting Silos project.

The second related work is the Canvas programs written by prof. Maguire. For example, several of his Canvas programs, export an Excel file by using the python pandas plugin [37]. This excel file can include different categories of information from Canvas. Since the Connecting Silos project does not need all the information in the output Excel file, the code from these programs will be modified, so that only the data relevant to this project will be returned.

2.10.1 Pdfssa4met

Pdfssa4met is a python open source project by Elias Kunnas from Finland that aims to provide metadata extraction and tagging based on structural and syntactic analysis of content in XML [28]. The project is based on the pdf2xml project, which is a tool to convert a PDF file into an XML file. After converting the PDF file to an XML file, pdfssa4met can search in the XML for a specific keyword or XML tag. For example, to look for a reference (see the file 'reference.py'), the program first looks for the keyword "Reference" and then for the tag <reference></reference> [28].

2.10.2 kthextract

Based on Pdfssa4met, prof. Maguire developed kthextract.py to extract certain information from a Bachelor or Master's thesis. The data mining part of the Connecting Silos project will be based on 'kthextract' project and will further develop this program to achieve the goal of the project.

The function pdf2xml is the core of pdfssa4met. Most of the operations in pdfssa4met require pdf2xml. Since Pdfss4met requires pdf2xml, the kthextract project also requires pdf2xml. The purpose of pdf2xml is to convert a PDF document into XML for easier processing. The conversion between PDF and XML is useful when it comes to datamining. The function pdf2xml builds a tree structured XML object for search purposes based on the chapter or line property in the PDF document [38]. To use pdf2xml, a user needs to download the executable file or core via the link: https://sourceforge.net/projects/pdf2xml/ [39]. The compilation process is different depending on the local operating system.; therefore, a user needs to download a specific version of pdf2xml depending on their operating system. After pdf2xml is downloaded, the user needs to modify the 'config.py' file in order to specify the path to the pdf2xml file. An example of config.py is shown in Figure 2-11.

```
""" Configuration for pdfssa4met

Created on Mar 1, 2010
@author: John Harrison
"""

# Path to pdf2xml executable
# this should have been downloaded from:
# http://sourceforge.net/projects/pdf2xml/

pdf2xmlexe = "pdf2xml/current/pdf2xml_osfit"

# API Key for OpenCalais Web Service
# You'll need to register for one yourself at:
# http://www.opencalais.com/user/register
# then request a key
OpenCalais_API_Key = "XXX"
```

Figure 2-11: An example configuration file for pdf2xml

The content in pdf2xmlexe is the location of the pdf2xml file which in the example shown above is "pdf2xml/current/pdf2xml_osfit".

2.11 Reliability Analysis

The application is based upon the specific combination of course ID and assignment ID specified in the installation phase followed by the student ID(s) selected to be processed by examiner. Thus, to some extent the reliability depends on the accuracy of the examiner's actions in terms of for example choosing the correct student ID(s) to be processed.

However, reliability can be an issue in the Data Extraction module due to the parsing system's attempting to identify the format of the text (for example bold face, font name, and font size), position of the text (for example page, block, and line), and the content of the text. This system will not cause a significant reliability issue as long as students follow the proposal and thesis report template that is provided by KTH examiners. Unfortunately, the developer cannot control the user's activity once the application is in use. To maintain reliability and validity, specific unit test cases and integration test cases are used. These test cases can be found in Section3.4.2 on page 37.

2.12 Validity Analysis

Considering the reliability issues of the Data Extraction module described above, the Connecting Silos project attempts to extract as much information about the student, examiner, and academic supervisor as possible from Canvas rather than by parsing the PDF file. Moreover, the Data Extraction module will compare each respective meta data field from a MODS record for an existing thesis with the values extracted from PDF to validate whether the parsing is being done correctly and if the parsing results are reasonable. If an error is detected during the validation process, then the system will either generate an error report in the Data Extraction module or utilize the results from Canvas as the correct result. The decision as to which approach the Connecting Silos system will choose is based on the specific type of error. The extra information that is extracted from Canvas will be injected into the output of the current parsing session as an extra field.

Before the project is finally deployed, there should ideally be two rounds of testing. During the initial test sessions, the project team will perform tests, while in the second round of testing the test program would be deployed to KTH administrative staff and feedback from this staff collected. This feedback will describe the accuracy of the output data, user feedback (if the program makes the user uncomfortable), and sustainable development feedback (whether the program will be used in future work). Eventually, from the effectivity and efficiency point of view, we will estimate how much time the program saves.

2.13 Summary

The sections above attempted to give an overview and provide basic knowledge of the main tools used in the Connecting Silos project. Understanding these tools will hopefully provide the user with sufficient background to be able to follow the rest of this thesis, specifically where the implementation and developed will be described.

The tools in use by the project that were described above, were constantly validated against their usage complexity as well as relevance to the development goals. For example, some tools such as Python Selenium (Section 2.9.1) were used by the project in the beginning, but where replaced by other methods/tools later in the development phase. These changes of methodology/approach were mainly caused by the team members learning about new tools and approaches throughout the project. For example, python Selenium was initially used in order to log in to a specific KTH account from which submission files could be downloaded from Canvas. Whereas later on, the team members discovered that the Canvas Files API (Section 2.4.2) is a safer and more straight-forward approach since it did not require logging in nor was there a need to provide user credentials as input parameters.

Some specific python libraries that were used very breifly and only for a very small number of functions inside modules, are not specifically mentioned in the sections above. However, they will be referred to in the Chapters3 and 4, as appropriate.

3 Methodology

The purpose of this chapter is to provide an overview of the methods used in this thesis. Section 3.1 describes details of the research process. Section 3.2 further describes the project's management methodology. Section 3.3 focuses on the development and design of the automation system in detail. Section 3.4 explains the techniques used to evaluate the reliability and validity of the data collected, as well as testing the correctness of the system. Section 3.5 analyzes and evaluates the methods used in the project.

3.1 Project preparation

In order to be able to strategically outline the methodology of the project, The Connecting Silos project started with a short period of preparation and research. This period was concentrated on understanding the scope of the project, the problem that needs to be solved, how it can be solved, and of course, what specific approach should be used to solve the problem. Thus, the preparatory period of the project's timespan was divided into 3 stages: literature study, problem formulation, and solution realization – each is described in a subsection below.

3.1.1 Literature study

The literature study was the first step towards gathering the information necessary to start the project. The information gathered throughout the literature study phase provided the contents of Chapters 1 and 2.

3.1.2 Problem formulation

The main goal that the Connecting Silos project is to provide the fundamental elements for an automation system that can publish student theses from Canvas to DiVA and to create events in KTH's Calendar system. The basic functionality provided by the Connecting Silos project towards this automation will subsequently be used by others in future work to complete the platform and to facilitate the publication process.

For the purpose of automation, the project needed to realize the manual steps currently required to publish student thesis on DiVA. (Inserting data into the Calendar system follows a quite similar pattern.) As of Spring term 2018, students upload their proposal, thesis drafts, andtheir final thesis to the Canvas LMS. Any administrative staff member who is in charge of processing degree project documents will have to extract certain information from these documents. The information that needs to be extracted depends on the information that is required by DiVA to publish every student thesis. This information includes general profile information about the author(s), examiner, and supervisor(s) of the project. Furthermore, information about the contents of the thesis itself (such as abstract, title, organization and cooperation, keywords) and information about the oral presentation are needed. Currently, ahe staff member then manually fills in the corresponding fields via the DiVA user interface with this gathered information, uploads the thesis, and completes the publication process. Therefore, the problem formulation began with the Connecting Silos team identifying the individual tasks necessary to perform each step of the process so that it could be done automatically.

3.1.3 Solution Realization

The goal is to automate every step of the manual process described above. For this purpose, the system will have to step by step access the student's submitted thesis, check the status of this thesis in terms of being graded (as approved), and decide whether to parse and extract the desired information from the PDF version of the thesis. Once the thesis is parsed, the relevant information will be pushed to DiVA and the Calendar system respectively. Ideally a staff member should only need to accept the results of the different stages of the automation or modify the automatically entered data before approving the meta data for publication via DiVA.

The development of this solution mainly relies on studying the access and usage of the APIs for the relevant platforms, specifically Canvas and DiVA. The implementation will be based on Pdfssa4met and kthextract for parsing, and using the Canvas API to extract information from Canvas. In this way, the Connecting Silos project will build upon previous work.

Once these stages of the solution have been identified, then the project will identify specific tasks that need to be done at each stage. These tasks will be distributed amongst the members of the Connecting Silos project team.

3.2 Project management

In order to manage the project's development, the project followed an agile project management format. Agile project management is an approach based on delivering requirements iteratively and incrementally throughout the project life cycle [40]. Agile is based on dividing up the work into different tasks that are prioritized based on their importance relative to one another [40].

An agile project's defining characteristic is that it produces and delivers work in short bursts (or sprints) of durations of up to a few weeks. These sprints are repeated to refine the working deliverable until it meets the client's requirements [40]. One of the agile methodologies is known as Scrum methodology and this is the agile approach adopted by the Connecting Silos team.

The Connecting Silos project management identified each week of project work as one sprint. Each sprint has a specific goal upon which the tasks of the sprints are based. These small tasks are separate pieces to achieve the sprint's goal and are distributed amongst the team members and identified on a task board.

The specific task board used in this program is an online tool called Jira cloud (described in detail in Section 2.2). This online task board categorizes the tasks as "to do", "in progress", and "done". The title of these categories is quite self-explanatory. At the end of each sprint, those tasks that are not moved to the "done" category are moved to the next sprint. Each week, the Connecting Silos team members will meet to update each other on what has been done, what needs to be done, and if the plans needs to be modified. During this meeting, the team members exchange their tasks from the previous sprint so that both team members get a chance to contribute to every task in the project.

Exchanging the tasks may have drawbacks in terms of negatively affecting the efficiency of the project. Since once the tasks are exchanged, the team member who did not start with a certain task will probably need to spend as much time as the previous member who started the task in order to figure out how to continue. However, the Connecting Silos team realized that this drawback would be minimized by a thorough discussion of what each member has previously done. Moreover, this approach had the benefit of both members having confident awareness of every step in the project's development.

3.3 Development Methodology

As shown in Figure 4-1 on page 47, the methodology proposed for carrying out the project was divided into 4 modules namely: Canvas Data Collection (CDC), KTH Data Collection (KDC), DiVA Publication (DP), and Data Extraction (DE) module. Each of these modules will be implemented individually and eventually integrated into a system.

The CDC module extracts information directly from Canvas by using the Canvas API. This method is used to extract information about the author(s) of the thesis, the examiner, and supervisor(s). Additionally, this module accesses the degree project documents submitted by the student.

The KDC module is similar to the CDC module in terms of collecting information. However, the KDC module utilizes the KTH API (described in Section 2.7.1) in order to extract detailed information about students, examiner, and supervisors that is not provided by Canvas.

The DP module should ideally use the DiVA API in order to push the extracted and parsed information to DiVA. However, since the DiVA API is unavailable to the team members, the project will instead use the extracted information to generate a MODS file that can later be imported to populate the DiVA fields. The MODS file was described in detail in Section 2.5.1. Automatically importing the MODS file to insert the required meta data into DiVA fields will be left for future work.

The DE module interconnects the CDC, KDC, and DP modules into an integrated system. This module handles extraction and parsing of the information collected from Canvas that will later be used in order to generate the MODS file for DiVA. As the title of the DE module suggest, this module extracts the input information retrieved from Canvas (both from the PDF document(s) and other data in Canvas) to generate the output which will be written to a MODS file.

This section aims to provide the reader with an overview of the project's development methodology. Each of the modules will be further described in subsections below. Once the student(s) that is/are to be processes are chosen, the CDC module begins processing.

3.3.1 Canvas Data Collection module (CDC module)

The CDC module is centered on accessing Canvas via the Canvas RESTful API and gathering data. The gathered data can be considered as the input data for the project. Because it is the data collected from the Canvas database that will eventually be inserted into DiVA.

Thesis students have 3 phases of submissions that involve submitting PDF files. Phase 1 is known as the alpha draft of the thesis, phase 2 is the beta draft, and phase 3 is the final version of the thesis.

The CDC module processes those students whose beta draft has been submitted, graded, and approved. These students can potentially be scheduled for their oral presentation; hence a Calendar event can be generated for them. Those students who have submitted their final thesis and for which the thesis has been graded and approved should have their PDF file processed to extract the desired meta data.

Processing begins by obtaining a link via which all submissions can be accessed and downloaded. Each submission is linked to its specific student author(s), supervisor(s), and examiner. The Canvas API can be used to obtain a list of the students in a given Canvas course. Therefore, author information, supervisor information, and examiner information can also be gathered from Canvas.

The Canvas File API is one of the Canvas API functions (described in Section 2.4.2). This API is used to download the student's proposal, the beta draft, and or final thesis. The CDC module eventually invokes the DE module in order to apply the parsing algorithm to the downloaded PDF submissions from Canvas.

3.3.2 Data Extraction module (DE module)

The DE module mainly parses a source PDF file to extract information that can later be inserted into DiVA (or other system). The source documents are those submitted by the student as thesis project submissions. These documents are initially converted to XML when the DE module begins. The DE module then parses the XML to extract the desired information. There may be duplicate information (for example the student's name) can be obtained both from the thesis via the DE module and from the Canvas RESTful API via CDC module. The system uses this redundant data to check for data consistency.

The information that needs to be extracted from thesis is of 2 major different types. Either they are (1) coming from a fixed location and the system knows where to look for them or (2) they may appear on any page, or even a block of the page, for example the abstract. In this later case, the module needs to search for them in order to find them.

Information that has a fixed location is located according to the standard structure of a thesis document. For example, the (inside or exterior) cover page always is the first page. Therefore, if some of the information that needs to extract is present on the cover page, then this information will appear in the first page of the thesis document. The extracted information will be stored in a JSON file. aAt the end of DE module, this JSON file will contain a combination of extracted data per degree project (i.e., a combination of a degree project's thesis drafts) as well as the corresponding thesis proposal. This output JSON file is located in a folder that is created per session and per user of the current execution of the program. This folder is named based on the authors of the degree project being processed, the session ID, andthe time and date of the session.

Information (such as the abstract) that does not have a strict position in the document is extracted in a similar manner. However, the difference is that there needs to be added functionality to search for the location of the information. Since the abstract is usually in one of the early preface pages of the thesis, the program searches for it by going through the thesis XML block by block until it finds the abstract (based upon a list of strings that include Abstract, Sammanfattning, Summary, ...). Once the position of the information is found, then the information is extracted. Some specific information (i.e., general information about the thesis) are pushed to the Canvas gradebook, where specific custom columns are created per course[*].

Extracted data is pushed to the Canvas gradebook is for three purposes: (1) if the administrative staff wants to keep track of certain information for other purposes, this information will be present in relevant columns in the Canvas gradebook; (2) if for some reason the automation failed to automatically extract the desired information, then the user (in this case a staff member) can manually enter the data into the custom column cell for this student; and (3) some specific data will eventually be present in gradebook columns once the processing module has completed, as this information will be used later by the KDC module to fetch additional information from other systems at KTH (as will be described in the subsection below).

3.3.3 KTH Data Collection module (KDC module)

The KDC module mainly serves as a data source for data that cannot be obtained by either the CDC module (RESTful API) or DE module (the thesis document). This

[*] These custom columns are created if they do not already exist.

data includes further information about the examiner and supervisor and that can be retrieved from other systems at KTH via the KTH APIs.

Once the DE module has extracted the names of the supervisor and examiner from the XML produced from the thesis document, this information is passed to the KDC module. Within the KDC module a KTH API is used to obtain additional detailed information concerning the examiner and the supervisor (for example, their KTHID, username, affiliation, position, ORCID identifier, etc.). This information will be appended to the JSON file that was generated from the DE module.

3.3.4 DiVA Publication module (DP module)

The DP module is not as complex as the CDC, KDC, or DE modules. At this stage, all of information from the degree project's PDF documents, Canvas API, andKTH API have been extracted and combined in the corresponding JSON file(s) per degree project. The next step is to generate the MODS file that will subsequently be used to insert the meta data into DiVA and potentially upload the PDF file to DiVA. MODS is used because it supports content in multiple languages (hence a thesis can have both Swedish and English language abstracts, titles, and keywords). Moreover, MODS can also be used to provide meta data about the student, examiner, and advisor(s).

3.4 Testing validity and reliability of the system

The system developed by the Connecting Silos project handles degree project data that are to be published in association with a degree project. Therefore, verifying the reliability and validity of the system as well as ensuring data-correctness is important, otherwise the integrity of author(s) and or KTH may be in jeopardy. Therefore, this bsection outlines the test environment for the system, as well as the particular test cases used to validate the reliability of the system (Sections 3.4.1 and 3.4.2 respectively). The section concludes by describing how the test cases and the test results were used to validate the reliability and correctness of the output data of the system (Section 3.4.3).

3.4.1 Testing environment (test bed)

Details of the test environment (test bed) used throughout the development phase of the Connecting Silos project is outlined below. The test environment consisted of a computer with the configuration shown in Table 3-1. This test environment was used to test every module that was implemented (i.e., the CDC, DE, KDC and DP modules described in Section 3.3). Testing the final integrated system must be done via deployment of the server test bed. Details of the deployment of the server are outlined at the end of this subsection.

Table 3-1: Test environment configuration

Operating System	Ubuntu 14.0, Mac OSX
IDE (Integration Development Environment)	Pycharm 181.5087.37
Compiler	Python 2.7.14 Anaconda Distribution,3.6.5 Offical distribution
Framework	Django 1.11.13
External Package Manager	pip 10.0.1, Anaconda
External Package List	Django == 1.11.13 lxml == 4.2.1 PyVirtualDisplay==0.2.1 requests == 2.18.4 selenium==3.11.0 urllib3==1.22 pandas==0.22.0 lti==0.9.2 djangorestframework==3.8.2 eulxml==1.1.3 simplejson==3.15.0

Since the system developed by the Connecting Silos project is client-server based software utilizing the LTI Interface (see Section 2.4.1.1), testing all of the individual modules combined into an integrated system was performed on a deployment server with the properties shown in Table 3-2.

Table 3-2: Deployment server

Server host	Digital Ocean
Server operating system	Ubuntu 14.01
Server Initialization Method	Digital Ocean Django One-Key deployment
Communication method between server and developer test bed	SSH (Secure Shell)
Deployment method	GitHub Deployment Branch under project repository
Compiler	Python 2.7.11 Official distribution, Python 3.0.1 Official distribution
External package manager	pip 10.0.1, Anaconda
Server service engine	Guincore, NGIX
Framework	Django 1.11.13
External Package List	Django == 1.11.13 lxml == 4.2.1 PyVirtualDisplay==0.2.1 requests == 2.18.4 selenium==3.11.0 urllib3==1.22 pandas==0.22.0 lti==0.9.2 djangorestframework==3.8.2 eulxml==1.1.3 simplejson==3.15.0

To deploy the project, user needs to clone the project into the server by entering: 'git clone git@github.com:GiantPanda0090/connecting_silos_kththesis_TCOMK_CINTE.git' on the command line.

If user uses the Digital Ocean Django One-key deployment package, the user will only needs to copy the folder 'polls' into the 'home/Django/Django_project' folder. Thereafter, the user builds the test bed by running, './install_requisition.sh' located in the 'polls' folder inside the bash command line window. The 'install_requisition.sh' script will do the remaining operations, such as installing all the external modules, installing a suitable version of python, and exporting the appropriate paths. Once 'install_requisition.sh' has been executed, the user can start to install the add-on developed by Connecting Silos project in Canvas and thus create an LTI Interface between the Canvas LMS and the deployed server.

To install the add-on and to create the LTI Interface, user needs to run the Installation module (see Section 4.3) by entering 'http://<deployment_server_ip>/polls/install' in the web-browser's search box. This will initiate the installation. The '<deployment_server_ip>' in the web-address above needs to be replaced by the IP address of the deployment server. The host name “localhost” works also works, but must be followed by the specification of the port number where the server listens. The installation process requires the user to input the specific Canvas course_id, the assignment_id for the proposal, beta draft assignment_id, and the final thesis draft assignment_id. If the installation was successful, user will see ‘Installation done’, otherwise the error message ‘installation failed’ will be output. During the installation, the Installation module will also install a suitable pdf2xml core and Selenium core that are compatible with the operating system. The details of the Installation module are described in Section 4.3.

3.4.2 Test Cases

To ensure the quality and viability of the developed system, specific unit test cases are used for each of the individual modules (CDC, DE, KDC and DP) integrated into the final product. Moreover, the integration testing was carried out repeatedly, throughout the entire project’s development phase. Therefore, all stages of the project from the preparation and literature study until the end of implementation have been under Quality Assurance (QA) testing. QA testing implies that the product undergoes testing throughout the entire project in order to ensure maintenance of a certain level of quality.

The unit test cases for each particular development module used in the Connecting Silos project are stated in Table 3-3. Since the project development has been divided into 4 different modules (see Section 3.3), the unit test cases for each module were designed based on the functionality and properties of the respective module. The purpose of the unit test cases for each of the modules is to ensure that each module generates consistent, correct, and reliable output as well as ensuring the correctness of functionality of the module.

Integration of test cases inside the automation system is referred to as the Test Automation submodule (TA) for the Connecting Silos project (see Section 4.4). The unit test cases have been implemented for the purpose of testing the complete automation system (i.e., all of the individual modules integrated into a system). Some of these test cases are not currently implemented inside the TA submodule, but the user can manually invoke them. The project team will also manually invoke these test cases on the final integrated system before the final deployment.

The test-cases outlined in Table 3-3 are categorized as: “integrated”, “not integrated”, or manual. Most test-cases are automated and implemented by a program. However, some of these test cases are integrated inside the TA submodule and thereby referred to as “integrated”[*]. Whereas other automated test-cases have not been integrated in the TA submodule, hence they arereferred to as “not integrated”. However, the “not integrated” tests are still separately applied to the

[*] Some of the integrated test-cases are integrated by the Connecting Silos project team and the rest were already integrated in the tools the project team used. For example, the Django framework will throw error reports upon failure. Some of such error reports are directly the results of some of the test-cases.

system. The remaining test-cases that are referred to as “manual” are test cases that are manually tested and require the developer’s observations as they are not implemented with any automated testing program. The category of each test-case is specified inside each “Test Case” cell under the title of the test-case as shown in Table 3-3.

Table 3-3: Unit test cases

Nr	Test Case	Passing requirements	Status
	CDC module		
U_1	Extracting the correct document “Manual”	The document that is extracted from the zip package which is downloaded from Canvas fits the correct assignment and course code	Pass
U_2	The student list is correct and complete “Integrated”	The student list that is extracted before the DE module begins is correct and complete	Pass
U_3	The pdf file is successfully downloaded “Not integrated”	The Canvas File API successfully downloads the file The pdf file titled with the student id can be found under the “Source” folder	Pass
	KDC module		
U_4	The student information is correct “Implemented”	The student information that is extracted from KTH Profile API:er is correct	Pass
	DE module		
U_5	The PDF file is accurately converted to XML “Integrated”	The PDF is successfully input to the system The transformation from PDF to XML t is done correctly	Pass
U_6	The fixed-location parsing is correct “Not integrated”	The parsing result from fixed location parsing function is correct and complete without less or extra information	Pass

U_7	The dynamic location parsing is correct “Not integrated”	The parsing result from dynamic page parsing function is correct and complete without less or extra information	Pass
U_8	The system is parsing the correct PDF file at the correct stage “Manual”	1. The PDF that is being parsed belongs to one of the students in the student list obtained from Canvas 2. The proposal is being parsed in the proposal phase 3. The thesis is being parsed in the thesis phase	Pass
U_9	The folder that is created for each session should contain output.json “Integrated”	The folder in output/parse_result/<author1>_<autho r2>_<session_id>_<datetime> that is created for each session should not be empty and contain only one file which is ‘output.json’ If the folder that is created for current session is empty in the end of session, a warning should be thrown. In this case test U_10 failed.	Pass
U_10	The data stored inside each output folder should be updated with the newest session results if applicable “Manual”	1. If two keys are the same between the old and new JSON string, the system should choose the new data in the key as output. 2. If there are keys in the old JSON string but not in the new JSON string, the keys should be added into output as extra key:data pair 3. The new JSON string should always d be compare with the old JSON string. If there is no JSON string exists, the new JSON string will be directly used as output.	Pass
	DP module		
U_11	The MODS file is successfully created	1. No error report occurred on client side 2. The MODS file is in the same output folder as its corresponding ‘output.json’ file for the particular session and user 3. The MODS file should end with. mods	Pass

	"Integrated"		
U_12	The MODS file is downloadable "Integrated"	1. The MODS download link is created successfully and is downloadable 2. On client side, the download window can successfully pop up and the file can be successfully downloaded without error report	Pass
U_13	The MODs data should be consistent with 'output.json' data "Manual"	The data under each key should be same as the same as corresponding key data in 'output. json' in the respective output folder	Pass

The Integration test cases are stated in Table 3-4 below. The integration test cases focus on the functionality and overall outcome of the entire system. Even if each module functions correctly and generates correct and consistent result individually, it is not a reliable evidence for the entire integrated system functioning correctly as a whole. Therefore, to ensure the reliability and functional correctness of the system as a whole, the integration test cases in Table 3-4 are executed. The categorization of test-cases is specified by "Integrated"*, "not integrated" or "manual" under the at the bottom of each "Test Case" block as above.

Table 3-4: Integration test cases

Nr	Test Case	Passing requirements	Status
		Implementation	
I_1	The output data in the final mods file are correct "Not integrated"	The output after CDC and DE modules is accurate and provides sufficient means to fill out all the mandatory fields in DiVA publication form	Pass

* Some of the integrated test-cases are integrated by Connecting Silos project team and the rest are integrated already in the tools the project team uses such as Django framework that will throw error reports upon failure. Some of such error reports are directly the results of some of the test-cases.

I_2	The system execution terminates successfully “Integrated”	The program will terminate without error report at any stage	Pass
I_3	The data duplicated in several modules is consistent “Not integrated”	The same key should have the same value in both CDC, KDC and DE modules: 1)when a key value is updated in one module, the corresponding key should be updated in all other modules it appears in	Pass
I_4	Data consistency in Thesis approval stage relative to Oral presentation scheduling stage “Manual”	Same key in ‘Oral presentation scheduling’ stage and ‘Thesis approval stage’ should have the same value in both stages: 1. When data in Thesis approval stage is modified, the previously generated Polopoly JSON string should be updated 2. After ‘Thesis approval stage’, the unmodified data should be the same as corresponding data in ‘Oral presentation scheduling’ stage 3. When user has a spelling mistake in the thesis document, the system should print an error message and notify user to check the program, LMS database and the thesis document	Pass
I_5	The program architecture is always consistent “Manual”	1. Downloaded thesis PDF file is always in ‘Source’ folder 2. The ‘output.json’ file for a particular session is always in the corresponding output/parse_result/<author1>_<author 2>_<session id>_<datetime> folder 3. The three parameters in <author1>_<author2>_<session id> are correct according to the currwent session. <datetime> should always change according to the current session time 4. The source code should always be in src 5. The project structure should be compatible with Django framework	Pass
I_6	The ‘Oral presentation scheduling’ stage has	According to thesis workflow, the ‘Oral presentation scheduling’ stage has to be executed before ‘Thesis Approval’ phase. If the ‘Thesis Approval’ is executed before	Pass

	to be executed before 'Thesis Approval' "Integrated"	'Oral presentation scheduling' stage, the system should throw a warning.	
Others			
I_7	The literature study should provide sufficient knowledge to prepare for project development "Manual"	Sufficient knowledge should be obtained before start of the project development phase in order for the team members to be able to understand the scope of the project and outline a general methodology plan	Pass

3.4.3 Verification and validation method

The validity of the program is tested using the test cases outlined in Section 3.4.2 above. Each development module is tested against its corresponding test cases as specified in Section 3.4.2. The observations from the tests are described in Chapter 5. Ideally all the test cases should be implemented in the TA submodule and thereby automated. However, currently only a few of them have been implemented inside the TA submodule. Although, those test cases that are currently implemented inside the TA submodule are sufficient to ensure the validity of the program as well as the validity of the results. Moreover, the user can perform the test cases that are not implemented inside the TA submodule manually if desired. The integration of these test cases inside the TA submodule is left for future work as it will be mentioned in Section 6.3.

In order to confirm the reliability of the system's output files, the system results are compared with results obtained from manual processing of the document. The SHA-1 hash[*] of the output of the system developed by the Connecting Silos project will be compared to that of the output from manual processing of the document. The same experiment is repeated three times to confirm the reliability of the result as well as consistency of the resulting data. If all three rounds of this experiment successfully generated the same corresponding SHA-1 hashes, the program is considered to be consistent and thereby reliable in terms of the correctness of the data it outputs.

[*] SHA-1 (Secure Hash Algorithm 1) is a cryptography hash function which takes certain input and produce 160-bit hash value. The SHA - 1 generation function is a one-way function and the file that generates SHA - 1 code cannot be restored from sha-1 hash code. SHA-1 value can be used to validate the completeness of the file which in the project. [41]

3.5 Analysis and evaluation of development methodology

The development model selected for the Connecting Silos project was thoroughly described in Section 3.3. Categorizing the development work into four categories has its advantages and disadvantages.

Dividing up a development project and categorizing its sub-tasks based upon their different characteristics can be both negative and positive for progress as well as successful completion of the project. This positivity and/or negativity may depend on the specific characteristics upon which the subtasks are categorized or the extent to which development tasks are appropriately broken down. The following section will discuss these aspects of the project.

3.5.1 Categorization of subtasks

In the particular development approach selected by the Connecting Silos team, the tasks were categorized depending upon their use of a particular API and or module. The development model focused on two major APIs (the Canvas RESTful API and the KTH API), one specific module (kthextract), and the MODS file format throughout the project. This approach made the tasks in each category interconnected with regards to their area of study. For example, all tasks in the CDC module were more or less concentrated on the Canvas RESTful API and the team member who was developing the CDC module could learn from each task and hence be more efficient and effective when doing the next task.

If the categorization did not take into account the specific module or API of each task, then the team members would have had to study multiple APIs or modules for their sprint tasks, hence reducing their effectivity. This occurs because learning the Canvas RESTful API or the KTH API requires 1-2 days. Following this it takes another 1-2 days to actually use the API and do the desired task. Similarly, it would take a team member perhaps the same 2-4 days for the next task, such as working with the MODS file format. However, if both tasks of a given team member are concerned with the same API or file format, then this team member will likely not need additional study and preparation time for their second task as he/she is already familiar with the API or format and knows how to use it.

However, the specific task categorization of the Connecting Silos project development may be inconsistent with respect to the data sources from which the project obtains its data. As previously mentioned in Section 3.3, some data are obtained directly from the Canvas API (such as examiner, supervisor(s), and student info); while other data is obtained from the thesis itself (which is done by kthextract within the DE module). However, errors may occur, such as in the thesis, the author may have misspelled the examiner’s name, hence the examiner’s name as extracted from the thesis in the DE module will not correspond to the examiner’s name as obtained via the Canvas RESTful API within the CDC module.

While the inconsistency of data from different sources and from separate modules may seem to be a disadvantage, it may actually provide better validity in the final data — as it may be possible to decide which data source is the most reliable, Moreover, misspellings and mistaken inconsistency can be invalidated by the system. Thus, the system must either check for data consistency or choose the most reliable data to proceed with.

3.5.2 Extent of task division

Extent of task division refers to the extent to which the project's tasks are broken down. Tasks should be specific enough to be manageable in one sprint and yet cover enough of the entire project's goal so that the project is successful in achieving its goal. The Connecting Silos team divided its development tasks in such a manner that each team member would take responsibility for 2-5 tasks. These tasks were designed to be roughly manageable in a period of one sprint (normally corresponding to 1 week), but in the worst case, the tasks could require 1-3 days of the next sprint as well.

The project's progress will confirm that the sprint tasks covered enough of project objectives to complete the project in time. The use of agile scrum project management guided the task subdivisions. As of the beginning of the project, the development team made an overview of the maximum number of weeks that was to be spent on development. Thereafter, the project team could identify the objective of each week (sprint) in order for the project to be complete by the end of the last week of development period. Furthermore, the sprint's (weekly) goals identified what tasks needed to be the focus of each sprint in order for the sprint's goal to be met at the end of the sprint. Thus, from a general perspective, the scrum management approach helped the project team to identify tasks that were both sufficient for their respective sprint and effective for the project as a whole.

There were still tasks that perhaps were not specific enough and the developer had to further break down the tasks he/she was assigned to during the sprint. However, if all of the tasks had been divided down into the smallest possible task, then the project would risk having several unfinished tasks and insufficient time left for development before the deadline. Keeping the tasks from being too specific also prepared the project team for those projects which they will contribute to later in their careers. In industry today, the tasks one is put in charge of for a period of 1 week are sometimes not specific or even clear, hence it is up to the developer to identify what the task is about, what it requires, and what partial tasks need to be done in order for the whole task to be completed.

4 Project implementation

The project development was divided into four main modules. These modules are: Canvas Data Collection (CDC), Data Extraction (DE), Diva Publication (DP), and KTH Data Collection (KDC) modules as mentioned earlier in Section 3.3. These modules will be referred to by their abbreviations in this chapter for the purpose of simplification. These four modules cooperate with each other and construct the automation project workflow shown in Figure 4-1. Each column represents a module and the content in the column represents the workflow of that module.

The project was developed in two different stages: “Oral presentation scheduling” stage and “Thesis approval” stage. The reason for this division into stages is the workflow of degree project administrative process. The oral presentation is scheduled based on the content of the thesis proposal or beta draft of the thesis. The time, data, and place of the oral presentation are combined with specific information from the information provided by the proposal or beta draft. This combined information will be used to schedule an oral presentation and generate an event in the school’s calendar in Polopoly. The beta draft will be revised following the final oral presentation and the final thesis will be submitted for approval by the examiner. Thesis approval stage uses the final thesis as submitted by the student in order to extract data from this PDF file. However, some of the extracted data from the final thesis might have previously been extracted during the Oral presentation scheduling stage from the beta draft. As the final thesis is an updated version of the beta draft, the corresponding data in the Oral presentation stage needs to be updated based upon the thesis submitted in the thesis approval stage. An example of data that might appear both in the Oral presentation scheduling stage and thesis approval stage are the names of the examiner and supervisor(s). If any of these names are extracted with a misspelling during the oral presentation stage; then if the misspelling is corrected in the final draft, thesenames will be updated in the final “output.json” file generated in the particular folder specific to the session of the program execution. This generated file contains the parsed and extracted results that may be updated until the program terminates. Thetwo stages interact with each other by using the “output.json” file which contains the essentiaoutput of the system.

Both the thesis approval and oral presentation scheduling stages begin by interacting with the user (presumably an examiner) who selects the students that are to be processed. Next, the CDC Module will obtain the information that the DE module needs as input parameters. The DE module will parse, select, and categorize the information and check for data consistency based on the information that was obtained from the CDC module. The Oral presentation scheduling stage, terminates by outputting a JSON string that is based on the result of the DE module. This JSON string can subsequently be used to automate the Oral presentation scheduling process to generate a Calendar event in Polopoly. Similarly, the thesis approval terminates with the generation of a MODS formatted XML file that can be imported into DiVA to complete the process or publication and archiving of a student's[*] thesis.

The CDC, DE, DP and KDC modules are the main backbone of the project development. The workflow of these modules will be described in more detail in Section 4.1 below.

The project is powered by the Django Framework and the LTI support from the Django framework. The LTI support [42] is provided by Django LTI support from OpenCraft [43]. The Django framework can make a local python application run as a web-based client-server application. This gives project the possibility to be deployed into a specific server or hosted by a test server with a given IP address or domain. The project is hosted on a server managed by Digital Ocean [44] and utilizes a secure end-to end-connection with Canvas. In this case the project team did not need to set up port forwarding in the team member's home router, but rather Digital Ocean provides a computer that is up and running for the duration of the project. Hosting the project on Digital Ocean also provided the project team members with the possibility to develop and deploy anywhere, anytime and the system can be accessed by any project member without restriction. Integration of the project using the Django framework is further described in detail in Section 4.2.

The remaining sections of this chapter focus on installation description for the user (Section4.3), followed in Section 4.4 by the implementation of the Test Automation (TA) module mentioned in Section 3.4.2 . The chapter concludes with a brief description of external packages used throughout project implementation, some of which were not sufficiently fundamental to the project development to be mentioned in Chapter 2.

[*] Note that here and elsewhere we refer to the student in the singular, but in the case of a 1st cycle thesis there could actually be two authors who are responsible for a given thesis.

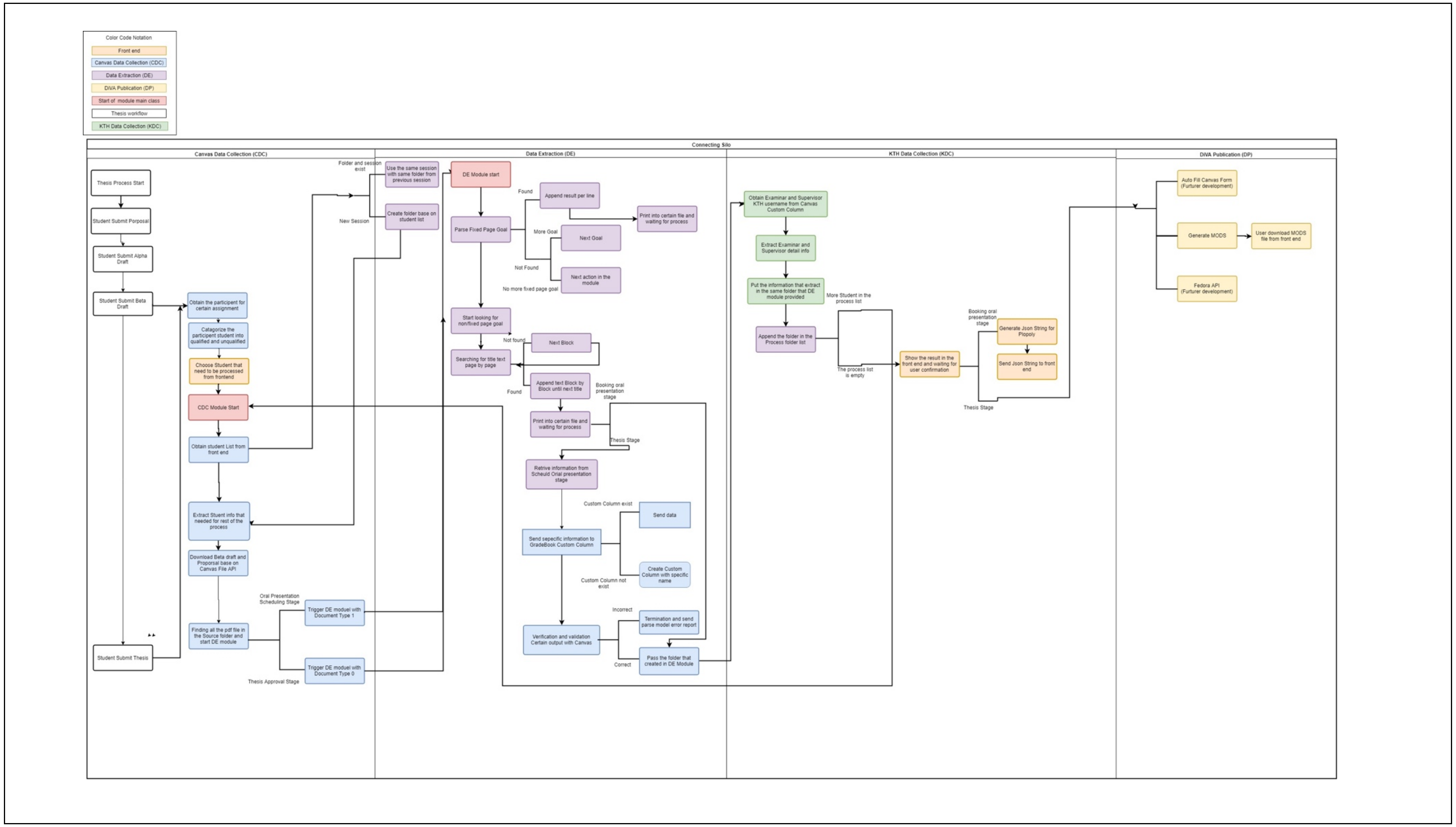

Color Code Notation
Front end
Canvas Data Collection (CDC)
Data Extraction (DE)
DiVA Publication (DP)
Start of module main class
Thesis workflow
KTH Data Collection (KDC)
Connecting Silo
Canvas Data Collection (CDC)
Data Extraction (DE)
KTH Data Collection (KDC)
DiVA Publication (DP)
Thesis Process Start
Student Submit Porposal
Student Submit Alpha Draft
Student Submit Beta Draft
Student Submit Thesis
Obtain the participent for certain assignment
Catagorize the participent student into qualified and unqualified
Choose Student that need to be processed from frontend
CDC Module Start
Obtain student List from front end
Extract Stuent info that needed for rest of the process
Download Beta draft and Proporsal base on Canvas File API
Finding all the pdf file in the Source folder and start DE module
Oral Presentation Scheduling Stage
Trigger DE moduel with Document Type 1
Thesis Approval Stage
Trigger DE moduel with Document Type 0
Folder and session exist
Use the same session with same folder from previous session
New Session
Create folder base on student list
DE Module start
Parse Fixed Page Goal
Found
Append result per line
Print into certain file and waiting for process
More Goal
Next Goal
Not Found
Next action in the module
No more fixed page goal
Start looking for non/fixed page goal
Not found
Next Block
Searching for title text page by page
Found
Append text Block by Block until next title
Booking oral presentation stage
Print into certain file and waiting for process
Thesis Stage
Retrive information from Scheuld Orial presentation stage
Custom Column exist
Send data
Send sepecific information to GradeBook Custom Column
Create Custom Column with specific name
Custom Column not exist
Incorrect
Termination and send parse model error report
Verification and validation Certain output with Canvas
Correct
Pass the folder that created in DE Module
Obtain Examinar and Supervisor KTH username from Canvas Custom Column
Extract Examinar and Supervisor detail info
Put the information that extract in the same folder that DE module provided
Append the folder in the Process folder list
More Student in the process list
The process list is empty
Show the result in the front end and waiting for user confirmation
Booking oral presentation stage
Generate Json String for Plopoly
Send Json String to front end
Thesis Stage
Auto Fill Canvas Form (Furturer development)
Generate MODS
User download MODS file from front end
Fedora API (Furturer development)

Figure 4-1: Flow chart of the Connecting Silos Project

4.1 Backbone modules

As stated above, the CDC, DE, DP, and KDC modules serve as the backbone of the backend functionality of the project. Moreover, they provide an overview of the project's development workflow.

These modules mainly interact with each other by passing input/output data to one another. For example, the output of CDC module is the input data to the DE module. However, the functionality developed within each module can be independently invoked when necessary.

4.1.1 Canvas Data Collection (CDC) module

The CDC module is the main source of data extracted during the Connecting Silos project's development phase. This module is built upon the Canvas API [14] and its basic functionality was provided in a set of python programs written by Professor Maguire. More information about the library and a link to the library itself can be found in KTH's Canvas instance in "Chip's sandbox"* [45]. Depending on the required backend functions needed by the project, the library can be modified to fit the requirements and purpose of the project.

The CDC module is generally triggered whenever the system needs to communicate with Canvas. The phases during which the system needs to communicate with Canvas and thus invoke the Canvas module are shown as blue boxes in Figure 4-1.

Both stages begin by entering the CDC module. From the CDC module, the input data that is later extracted by kthextract in the DE module can be utilized. Depending on the stage in the thesis workflow, the CDC module will trigger the kthextract function in the DE module with an input parameter of 0 for Thesis Approval stage or 1 for the Oral presentation scheduling stage. In Figure 4-1, one can see that the system can initially be triggered once the beta draft has been submitted for scheduling the oral presentation. The second trigger occurs after the student has submitted their final thesis for approval following their oral presentation. The functionality of the CDC Module for both stages is quite similar. However, the CDC module will be invoked on different (beta or Final) drafts, depending on the stage (Oral presentation scheduling or Thesis approval) at which it is being executed as shown in Figure 4-1 above and Figure 4-2 below. During the oral presentation stage, data is requested from Canvas based on the thesis proposal or beta draft. On the other hand, during the Thesis approval stage, the CDC module extracts data from the final thesis draft.

In Figure 4-2 one can see that the CDC module begins by interacting with user. The user specifies the student(s) for whom an Oral presentation is to be scheduled

* https://kth.instructure.com/courses/11

and or whose thesis is to be approved. The front-end provided for the user to specify students will then request the CDC module be initiated. The CDC module will then be responsible for obtaining (from their proposal, beta draft, or final thesis) information relevant to the specified students. This is done through execution of 'list_submissions' in Table 4-1 on page 53.

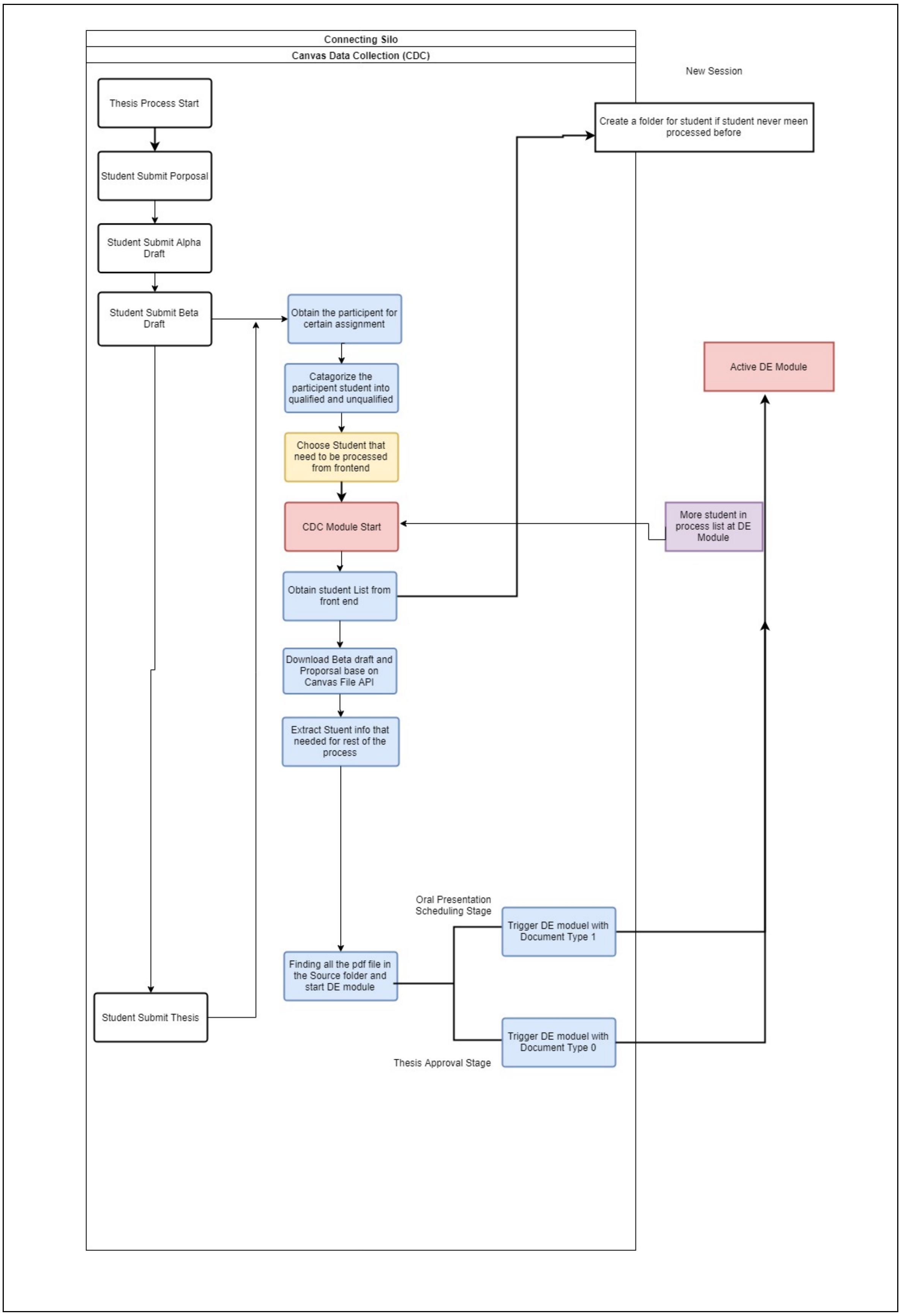
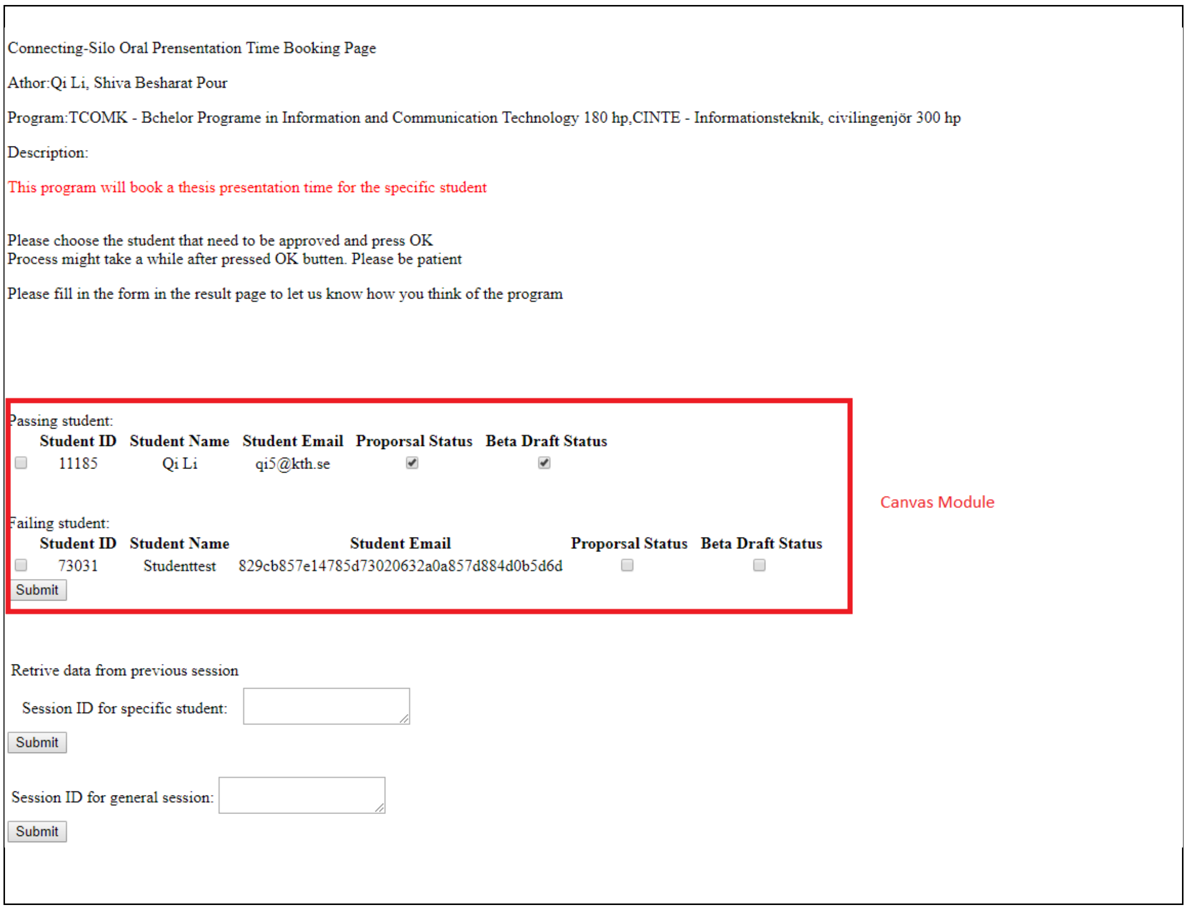

Connecting Silo
Canvas Data Collection (CDC)
Thesis Process Start
Student Submit Porposal
Student Submit Alpha Draft
Student Submit Beta Draft
Obtain the participent for certain assignment
Catagorize the participent student into qualified and unqualified
Choose Student that need to be processed from frontend
CDC Module Start
Obtain student List from front end
Download Beta draft and Proporsal base on Canvas File API
Extract Stuent info that needed for rest of the process
Finding all the pdf file in the Source folder and start DE module
Student Submit Thesis
Oral Presentation Scheduling Stage
Trigger DE moduel with Document Type 1
Trigger DE moduel with Document Type 0
Thesis Approval Stage
New Session
Create a folder for student if student never meen processed before
Active DE Module
More student in process list at DE Module

Figure 4-2: The flow chart of the CDC Module

The front-end user interaction and the front-end HTML page are based on the Django framework which will be discussed in Section 4.2. An example of user-interaction through the Django framework is shown in Figure 4-3. The red box highlights where the user chooses the student(s) for whom an Oral presentation is to be scheduled.



Connecting-Silo Oral Prensentation Time Booking Page

Athor:Qi Li, Shiva Besharat Pour

Program:TCOMK - Bchelor Programe in Information and Communication Technology 180 hp,CINTE - Informationsteknik, civilingenjör 300 hp

Description:

This program will book a thesis presentation time for the specific student

Please choose the student that need to be approved and press OK
Process might take a while after pressed OK butten. Please be patient

Please fill in the form in the result page to let us know how you think of the program

Passing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status
☐	11185	Qi Li	qi5@kth.se	✓	✓

Failing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status
☐	73031	Studenttest	829cb857e14785d73020632a0a857d884d0b5d6d	☐	☐

Submit

Canvas Module

Retrive data from previous session

Session ID for specific student:

Submit

Session ID for general session:

Submit

Figure 4-3: Example interface the index page

Once the user has chosen the student(s) to be processed, the CDC module's main function will be triggered as highlighted in Figure 4-2, by a red box with the caption: "CDC module start". The CDC module will then receive the list of students whom are to be processed. It will extract the information about each student in this list using the Canvas API. The CDC module will use the information obtained as an input parameter to call the Canvas Files API. The Canvas File API is used to download the submission file for each student. This action is implemented in 'get_submission_as_file' in Table 4-1.

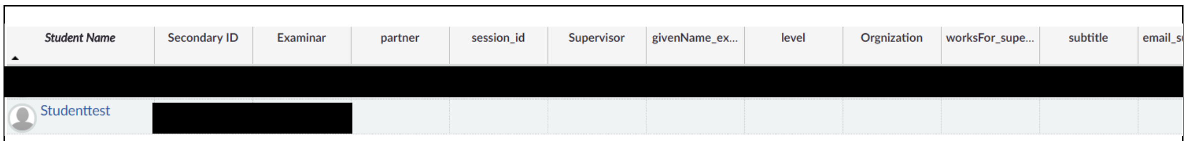
The CDC module needs to interact with the DE module as will later be described in detail in Section 4.1.2 in order to store specific data in the relevant custom columns inside the Canvas course's gradebook. This interaction occurs by executing 'custom_column_data' as described in Table 4-1. The purpose of this module is to store specific data in the custom column in the Canvas course's gradebook. If the specified custom column does not exist, then 'custom_column_data' will trigger ‘create_proposal_gradebook_columns’ to create a custom column with a specific title. Once a custom column is created, ‘fill_proposal_gradebook_columns’ is executed to store the data into Canvas. ‘fill_proposal_gradebook_columns’ executes an information filtering algorithm to select the information that is to be stored in the gradebook. For example, the Abstract and Introduction are not to be stored in custom columns in the gradebook and thus are filtered out. One reason for excluding the Introduction and Abstract from the gradebook is that each custom column is limited to a maximum of 255 words. The code shown in Figure 4-4 shows how some data can be filtered out in order to prevent them from being pushed to the gradebook. This process purposely excludes the heading, Orignization_supervisor, Sammanfattning (the Swedish version of the Abstract), Introduction, Examiner info, Supervisor info, and any field that has the keyword author, given name, and family name.

```
if filename != "heading.txt" and filename !="Orignization_supervisor(en).txt" and
filename !="log.txt" and filename!="abstract(en).txt" and filename !="abstract(sv).txt" and
filename != "introduction(en).txt" and filename != "Examiner.txt" and
filename !="Supervisor.txt" and "author" not in filename and "givenName_" not in filename
and "familyName_" not in filename:  # filename filter
```

Figure 4-4: The example code of a filter

Once the filtering is complete, ‘fill_proposal_gradebook_columns’ sends a ‘PUT’ request to the Canvas API to update the content in the custom column. ‘custom_column_data’ and ‘fill_proposal_gradebook_columns’ creates a dynamic content-update interaction. ‘custom_column_data’ is responsible for the existence check. The system needs to detect if there is any new data available before invoking ‘custom_column_data’. ‘custom_column_data’ will decide if this new data needs to be pushed to the gradebook or not and if a new custom column needs to be created before pushing the data.

The purpose of custom columns in the gradebook is to give the administrative staff, examiner, and supervisor, a clear overview of the status of any specific degree project student. An example of custom columns created by the Connecting Silo project in a Canvas gradebook is shown in Figure 4-5.



Student Name	Secondary ID	Examinar	partner	session_id	Supervisor	givenName_ex...	level	Orgnization	worksFor_supe...	subtitle	email_s
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
Studenttest	[redacted]	[redacted]									

Figure 4-5: Example interface of the custom column created by Connecting Silos

There are some test functions implemented within the CDC module. These test functions are meant to build specific test benches during different phases of development and to manually validate specific test data throughout the CDC module. For example, ’delete_custom_column’ deletes a specific custom column that was created by a previous test or one that is now considered to be unnecessary or irrelevant. This module can minimize the potential errors that might occur during the current test session.

Table 4-1: Project structure of the relevant CDC modules

Module name	Role in the system
create_proposal_gradebook_columns	Used for creating custom column in the grade book Input: course_id
custom_column_data	Input data into custom column in Canavs. If the column does not exist, create a custom column Input: course_id, column_number, user_id, value,position_data
delete_custom_column	Delete certain custom column. Used for building and resetting the test benches. Input: course_id,column_id
fill_proposal_gradebook_columns	Called by ‘custom column data’ module to fill in the data inside custom column Input: Course_id, file_list
canvas.get_file	Used to download the file from canvas. Based on get submission Input:file_id
get_submission	Used to obtain the submission download link. Based on list_submission. Input: course_id assignment_id user_id
get_submission_as_file	Combination of get_file and get_submission. Used to get the download link and

	download the submission file with Canvas File API. Input: course_id assignment_id user_id
list_assignments	Outputs the information about each of the assignments.
list_custom_column_entries	Used to list the entries of custom column. Order by row order of the gradebook Input: course_id column_number
list_custom_columns	List the property and identity of the custom columns Input: course_id
list_studentList_assignment	Returns a list of submitted assignments and the student who submitted the assignment with the formate of [user_id, name, log in id'] Input: course_id,assignment_id
list_students_in_course	List the students that participate in the course Input: course_id
list_submissions	Lists the submissions of specific assignment and returns the student id, name, email, passing state and kthid for students who have submissions for the assignment. 0 is failed and 1 is passed Input: course_id, assignment_id

4.1.2 Data Extraction (DE) module

The DE Module is responsible for searching, collecting, and processing data that can be used to generate DiVA MODS XML files and Polopoly JSON files, specifically using the proposal, beta draft, and final version of the thesis. As noted earlier, this module is thus based on the previous work done on Pdfssa4met described in Section 2.10.1 and kthextract. The DE module is also responsible for outputting and tracing the data within the correct session for a legitimate user. Figure 4-5 showsan overview of the DE module. The system initially checks if user data already exists, if so the existing session is updated. Checking for whether the student data is already in 'output.json' file of the session folder or not can ensure data consistency and avoid duplicate session folders which may result in system failure. The red box with the caption "DE module start" is the 'main()' function in 'pdfssa4met.kthextract' (Figure 2-1). The file format that is compatible with 'kthextract' is PDF Format. Before the main function starts parsing a submission, the 'kthextract' module will convert the PDF format into XML format via executing 'pdfssa4met.pdf2xml' (see Table 4-2). This module relies on the external module pdf2xml (see Section 2.10.1). The installation of the correct version of pdf2xml for the specific operating system is described in Section 4.3.

Table 4-2: Project structure of the DE Module

Module	Description
config	Configuration file for pdfss4met. Includes the setting for location of pdf2xmlexe and the key for OpenCalais_API_Key
headings	Parsing the heading in the PDF file fundamental of 'pdf2xml' Input: OPTIONS[*] FILEPATH
kthextract	The main class of the process module. This file includes the parsing model of thesis template and proposal. The output should be under polls/output/parse_result/author1_author2_sessionid_datetime Input: pdf_path, documentation type[†]
openCalais	Not used
pdf2xml	The core of kthextract and other component in process module
references	Parsing the reference from pdf. fundamental for 'pdf2xml'
socialtags	Parsing the social tag from pdf. fundamental for 'pdf2xml'

[*] OPTIONS: --help, -h Print help and exit, --noxml Do not tag individual headings with XML tags. Default is to include tagging. --title Only print title then exit, --author Only print author then exit

[†] Document type: 0 is thesis,1 is proposal

utils	The tool that is provided by pdfssa4met. Used by kthextract for different operations fundamental for ‘pdf2xml’

The ‘pdf2xml’ module creates a folder with a name of the form: ‘_author 1 name _author 2 name _session id_session date’. For example, the folder name for Connecting Silos team with members Shiva Besharat Pour and Qi Li approved on 2018-05-25 12:47:10.953676 will be: ‘_Shiva Besharat Pour_Qi Li_20824655-5d2b-11e8-a3d7-dab20c77adf1_2018-05-25 12:47:10.953676’. The session id is generated only if none of the authors’ names is a part of any other folder name in the output folders. If an already used author name is input to the ‘pdf2xml’ module, the module will use the same session id as was previously used for that particular author, and then update the processed date and time for the new folder’s name. Afterwards, ‘pdf2xml’ will merge the results of previous and current sessions and create a single unique folder.

There are two different parsing models in ‘kthextract’: (1) a thesis template and (2) a proposal template. Each parsing template has two data categories. One data category focuses on data that is available at a fixed position, in this case the module knows where to look for this data. The second data category concerns data that does not have a fixed or known position, such as the Abstract. As noted earlier this category of data requires the module to search for this data. Figure 4-7 shows that right after “DE module start”, the parsing functionality for each data category (represented as a fixed page goal and non-fixed page goal) is triggered. The system first looks for the data at fixed locations before searching for data at non-fixed locations. The reason for the module starting with data with known positions is that most of the fixed-page data mining occurs at the beginning of the thesis draft and or proposal document. Thereafter, the module searches through a document from the beginning to end in order, thus there is no need to return back to the beginning of the document when part way through the document. The idea behind this specific parsing order is to optimize the process with regards to time-efficiency, especially for the case of thesis documents as they may have a large number of pages. The fixed-page locations will be parsed for data that is always going to be on the same page. For example, the title, subtitle, and authors name will always be on the first page, while the examiner and supervisor(s) names are likely to be on this first page or the second page. The fixed-page data extraction process goes through each block on the page where the desired data is, until some specific condition has been reached. This condition can specify the font size, font face (such as bold or italic), or a specific keyword. For example, the code shown in Figure 4-6 looks for a block that has text with font size larger than 20 (points) and bold font face, or text that has font size larger than 15 (points) and bold font face.

```
        title_headers = trial_title_node.xpath(".//TOKEN[(@font-size > {0} and @bold = 'yes') or (@font-size > {1} and @bold = 'yes')]".format(20,15))
```

Figure 4-6: Example of matching criteria for a block

If all the requirements of a given condition are met, then the system will assume that the required data has been found and the specific block containing this data will be the result. The system output the information into the ‘output.json’ file of the current session’s folder.

The fixed page goal only outputs the information that meets certain requirement in a block whose position is known. Thereafter, the ‘kthextract’ module will output the result by creating the ‘output.json’ file in the current session folder. The flow of the fixed page goals can be found on the right branch of the purple box titled “parse fixed page goal” in Figure 4-7.

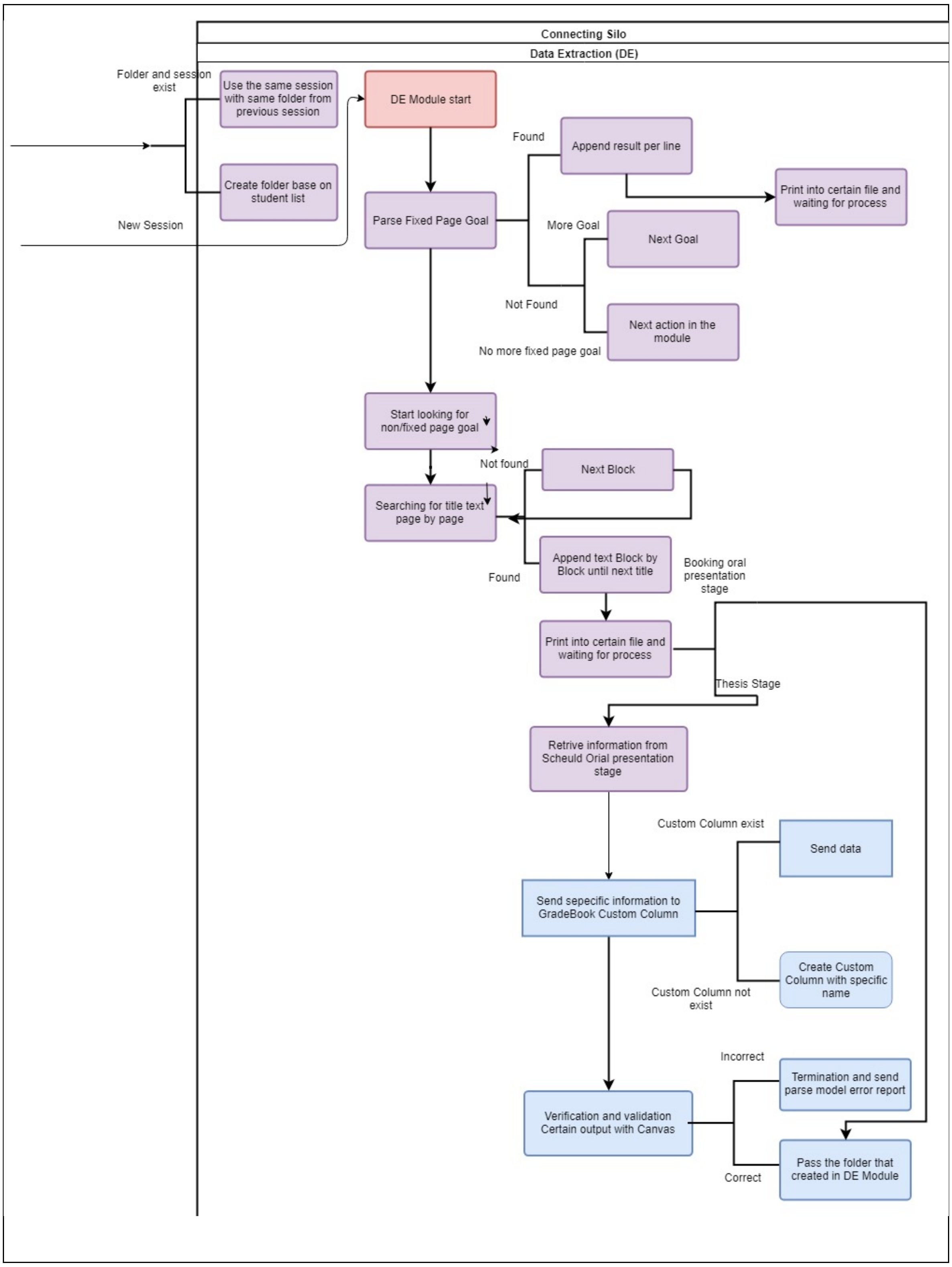

Connecting Silo
Data Extraction (DE)
Folder and session exist
Use the same session with same folder from previous session
Create folder base on student list
New Session
DE Module start
Parse Fixed Page Goal
Found
Append result per line
Print into certain file and waiting for process
More Goal
Next Goal
Not Found
Next action in the module
No more fixed page goal
Start looking for non/fixed page goal
Not found
Next Block
Searching for title text page by page
Found
Append text Block by Block until next title
Booking oral presentation stage
Print into certain file and waiting for process
Thesis Stage
Retrive information from Scheuld Orial presentation stage
Custom Column exist
Send data
Send sepecific information to GradeBook Custom Column
Custom Column not exist
Create Custom Column with specific name
Incorrect
Termination and send parse model error report
Verification and validation Certain output with Canvas
Correct
Pass the folder that created in DE Module

Figure 4-7: Flow chart of the DE module

After completing all of the fixed page extractions, 'pdf2xml' performs dynamic page extractions. Dynamic page extraction was referred to as the non-fixed page goal in the previous figure. The naming of this data category is due to the uncertainty regarding which page and block is to be parsed to extract the desired data. For example, the "Abstract" can be before or after the "Table of contents" depending on the author's choice. Another example is that the Introduction and Methodology usually come after the "Table of contents"; however, depending on the length of the "Table of contents", the exact page where the Introduction and Methodology starts varies. Thus, 'pdf2xml' must search through all of the pages to ensure the accuracy of the data collection. Except for the fact that dynamic page processing searches through all pages, the implementation of appending the results and setting the stopping point of data collection is also different in Dynamic page extractions compared to fixed-page extractions. The fixed page extraction only extracts the information that matches the requirement for a particular block, but the dynamic page goal outputs all the block that matches the requirement and *continues until the next heading*. The reason that dynamic page extraction extracts all of the block is that the data required all occurs under a specific heading, for example "Abstract", rather than being specific data *within* a block. For example, the requirement to find the "Abstract" needs to extract all the content under the heading "Abstract" rather than just the line that begins with 'Abstract'. Figure 4-8 shows all the functions in 'pdf2xml'. The section highlighted in red, shows the difference between print line and print block methods. The print line method is used in fixed page extraction and the print block is used in dynamic page or non-fixed page extraction.

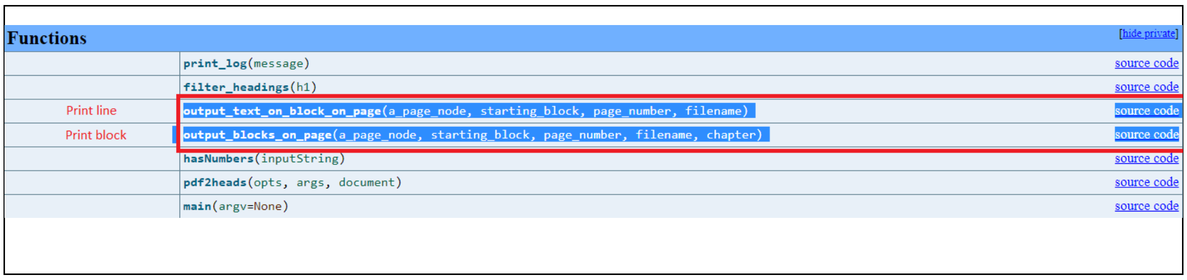

Functions		[hide private]
	print_log(message)	source code
	filter_headings(h1)	source code
Print line	output_text_on_block_on_page(a_page_node, starting_block, page_number, filename)	source code
Print block	output_blocks_on_page(a_page_node, starting_block, page_number, filename, chapter)	source code
	hasNumbers(inputString)	source code
	pdf2heads(opts, args, document)	source code
	main(argv=None)	source code

Figure 4-8: The function that prints line and block

After all the required data have been found and extracted, the CDC module 'custom_column_data' (Table 4-1) will be invoked to store specific information in the Canvas Gradebook to provide the user with a clear view of the extracted information. Afterwards the system will compare specific information with the corresponding data that was extracted from Canvas, in order to maintain data consistency and accuracy. The purpose of validating and verifying extracted results from the DE module is to validate and verify this data as the DE module is the final step in data collection before the data is sent to output. This output concerns both the 'Oral presentation scheduling' stage and 'Thesis approval' stage. A detailed discussion regarding 'custom_column_data' module is provided in Section 4.1.1 and the validation module can be found in Section 3.4.2.

4.1.3 KTH Data Collection (KDC) module

By now, most of the information that DiVA requires has been extracted and is now ready. However, some information is still missing. The reason is that not all of this information is available in Canvas or the thesis document. For example, the Open Researcher and Contributor ID (ORCID ID) of the examiner and supervisor [8], as well as some profile and contact information for both the author(s), supervisor(s), and examiner are not in Canvas or necessarily in the thesis document itself. Fortunately, the KTH profile of individuals often provides the detailed information about this individual. Therefore, the KTH API is used to access this data. The extraction of data via the KTH API occurs within the KDC module. The flowchart of this module is given in Figure 4-9 below.

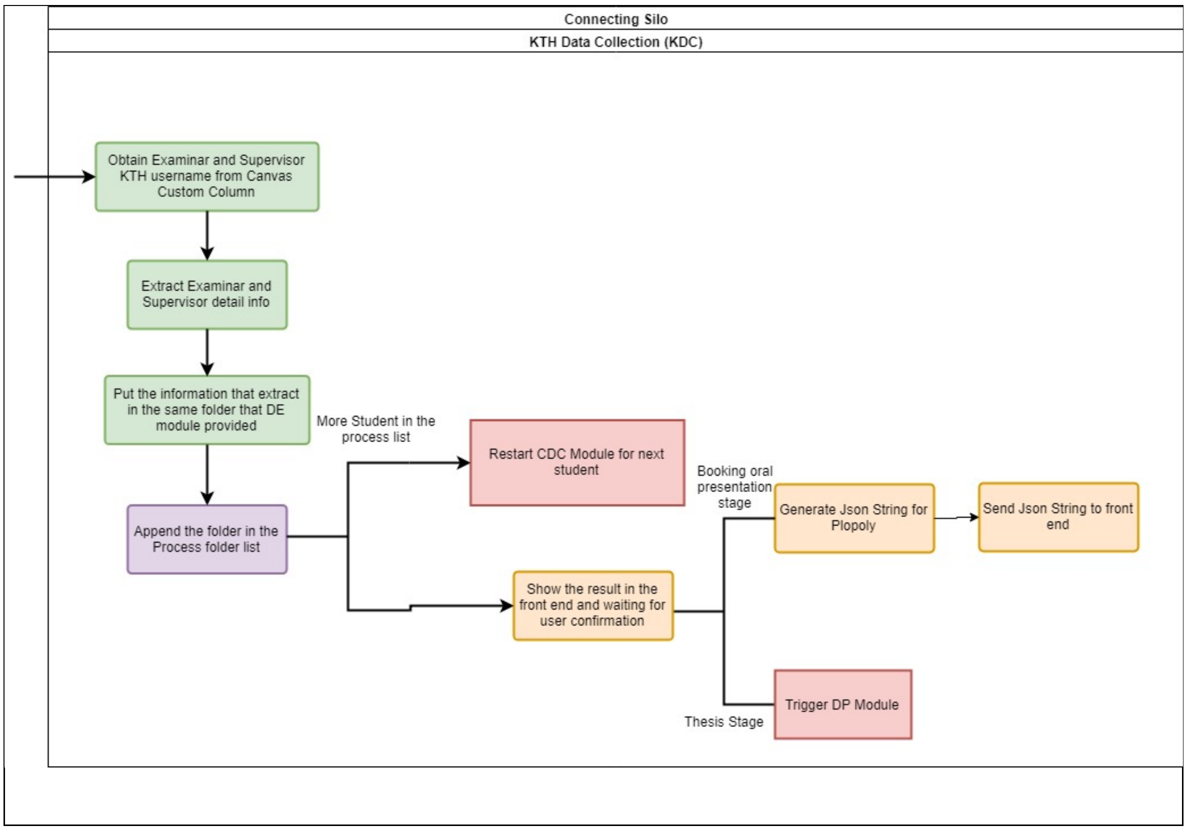

Connecting Silo
KTH Data Collection (KDC)
Obtain Examinar and Supervisor KTH username from Canvas Custom Column
Extract Examinar and Supervisor detail info
Put the information that extract in the same folder that DE module provided
Append the folder in the Process folder list
More Student in the process list
Restart CDC Module for next student
Show the result in the front end and waiting for user confirmation
Booking oral presentation stage
Generate Json String for Plopoly
Send Json String to front end
Thesis Stage
Trigger DP Module

Figure 4-9: Flow chart of the KDC module

The KDC module is integrated into the system, as two python modules and a JSON configuration file. These files together realize the KDC module and connect this module to the rest of the project. Details of this code will be introduced in this section.

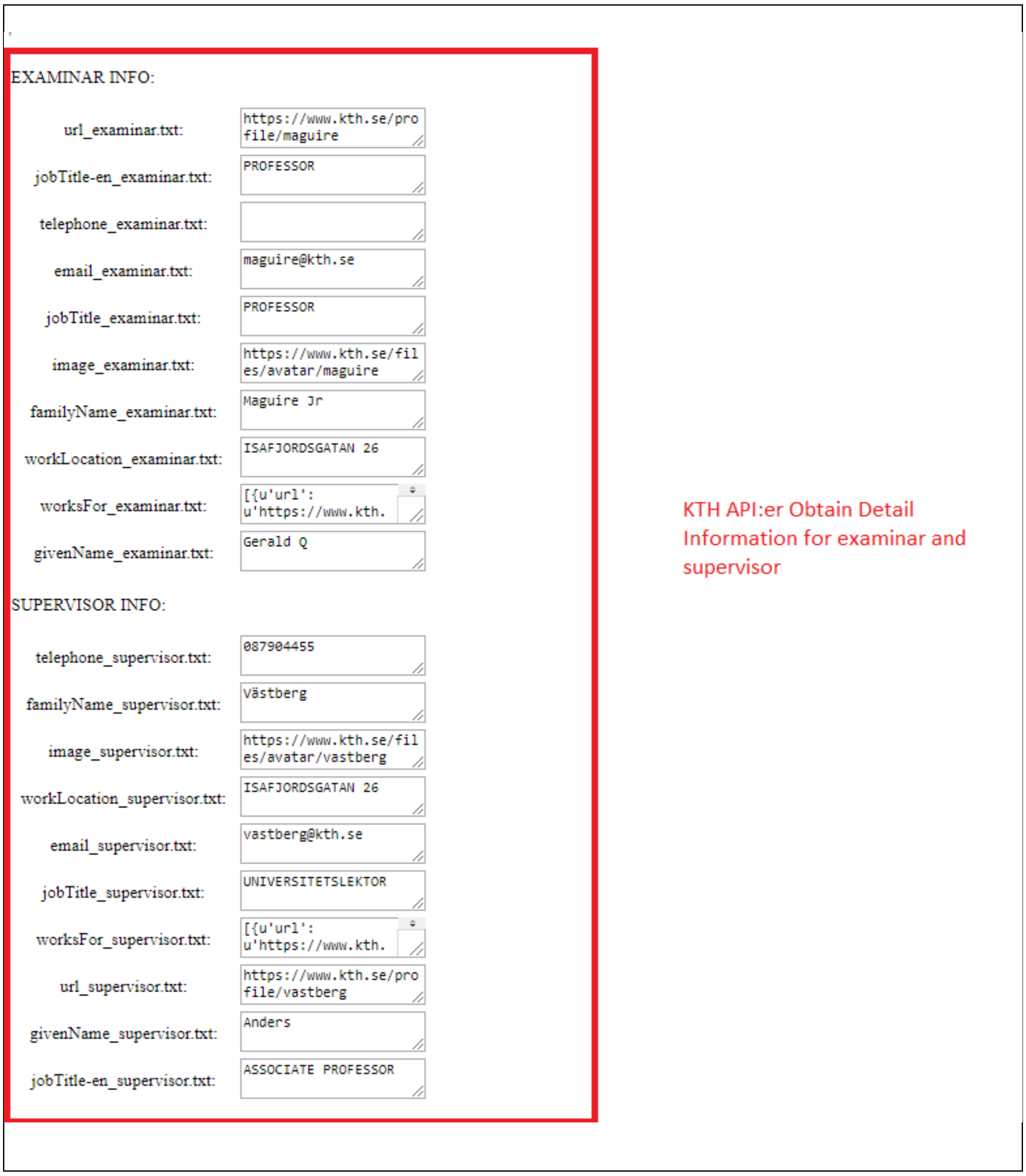
The structure of KDC module is shown in Table 4-3 below. The KTH API is accessed by using a specific URL which is specified in the module that accesses the API. The URL's host is taken from a file named configuration.json. Using this json file, a module named parse_profile.py configures the URL to access the KTH API.

Once the URL has been successfully configured, a function called get_user_info(userid) is called. Given this userid, the function constructs the specific URL needed to access the desired information about a specific user. A HTTP GET request with this URL returns the desired user information via the API. The content of the corresponding HTTP response is in JSON format. This data is then stored as the contents of a variable list inside the module. Upon invocation of the parse_profile.py module, this content is returned and is used in the system, to complete the previously generated 'output.json' file by the DE module (section 4.1.2). A third constructor module __init__.py imports the parse_profile module to be used within the system.

Table 4-3: The structure of the KDC Module

Module	Description
config.json	Configuration file that specifies the host of the URL address to access the KTH API:er
parse_profile	Returns all the detailed info found in the profile of the staff member or student via KTH API
--init--	It imports parse_profile and calls upon it

The output and application in the project for the KTH APIs can be seen in Figure 4-10 below. As it can be seen from the figure, the KTH API provides rather detailed profile information about individuals at KTH. Whereas Canvas API provides the necessary basic information for individuals such as name, program, and their role at KTH. Therefore, Connecting Silos found it necessary to combine results from both the CDC module and KDC module in order for the resulting 'output.json' file to be more complete (as is desirable for input into DiVA).



EXAMINAR INFO:

url_examinar.txt:	https://www.kth.se/profile/maguire
jobTitle-en_examinar.txt:	PROFESSOR
telephone_examinar.txt:	
email_examinar.txt:	maguire@kth.se
jobTitle_examinar.txt:	PROFESSOR
image_examinar.txt:	https://www.kth.se/files/avatar/maguire
familyName_examinar.txt:	Maguire Jr
workLocation_examinar.txt:	ISAFJORDSGATAN 26
worksFor_examinar.txt:	[{u'url': u'https://www.kth.
givenName_examinar.txt:	Gerald Q

SUPERVISOR INFO:

telephone_supervisor.txt:	087904455
familyName_supervisor.txt:	Västberg
image_supervisor.txt:	https://www.kth.se/files/avatar/vastberg
workLocation_supervisor.txt:	ISAFJORDSGATAN 26
email_supervisor.txt:	vastberg@kth.se
jobTitle_supervisor.txt:	UNIVERSITETSLEKTOR
worksFor_supervisor.txt:	[{u'url': u'https://www.kth.
url_supervisor.txt:	https://www.kth.se/profile/vastberg
givenName_supervisor.txt:	Anders
jobTitle-en_supervisor.txt:	ASSOCIATE PROFESSOR

KTH API:er Obtain Detail Information for examinar and supervisor

Figure 4-10: The output of KTH API

4.1.4 DiVA Publication (DP) module

Once the CDC, KDC, and DE modules have been implemented and integrated into a connected system hosted by the server, the data that is required for creating an entry in DiVA is available and now ready for insertion. Even if this insertion is done manually, since the data is categorized in a well-structured way, the goal of this project to provide improved efficiency in thesis publication via DiVA would be met. However, the Connecting Silos project included automation of data-insertion into DiVA in the project plan; therefore, this last step of the project has concentrated on providing a means of inserting the data output by the system into DiVA. Hence, this module is referred to as the DiVA Publication (DP) module. A flowchart of this module is shown in Figure 4-11.

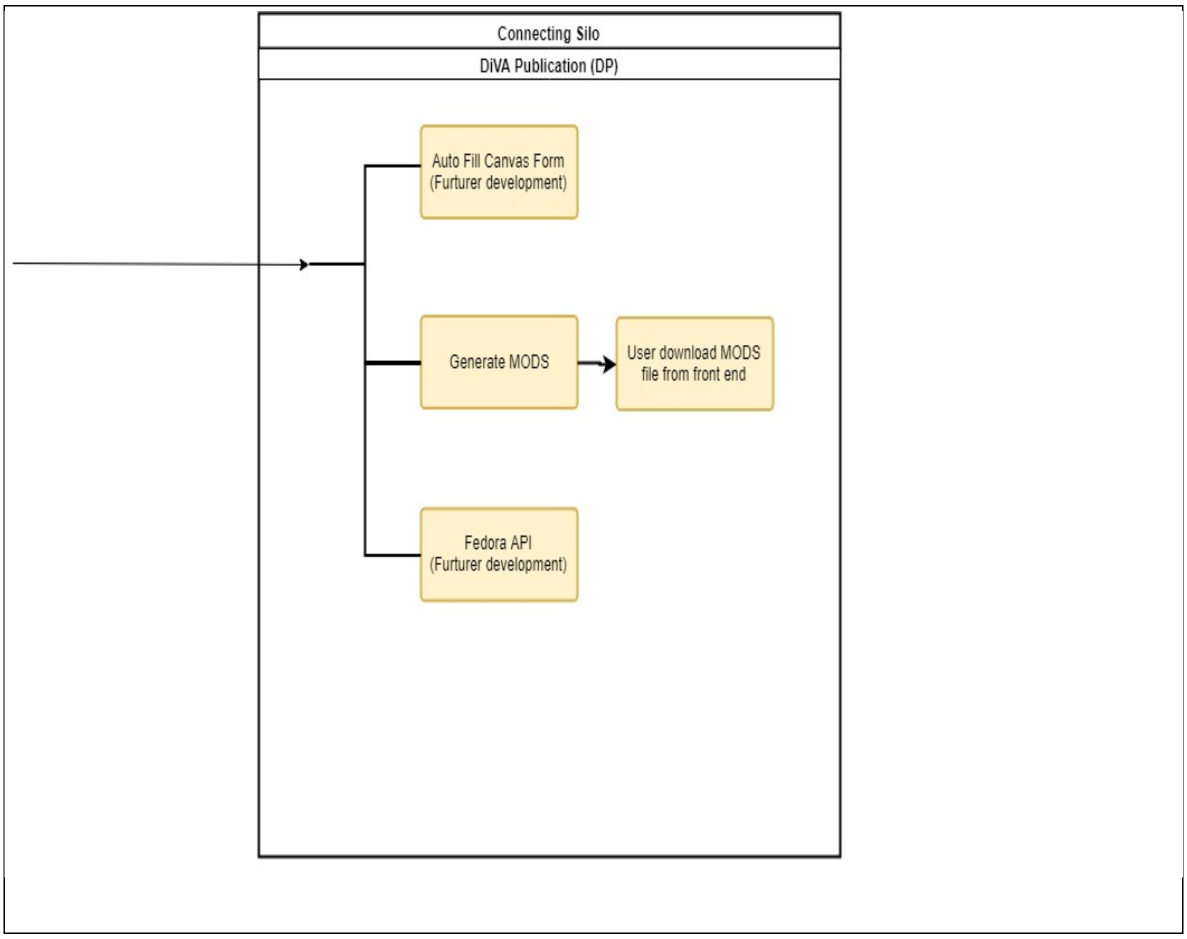

Connecting Silo
DiVA Publication (DP)
Auto Fill Canvas Form (Furturer development)
Generate MODS
User download MODS file from front end
Fedora API (Furturer development)

Figure 4-11: DP module flowchart

Although DiVA is based upon the Fedora API, we do not have access to this API for DiVA. As a result, we decided to simply generate a MODS XML file, as DiVA accepts MODS files for importing data. As a result, the DP module consists of only one module in its workflow where the MODS file is generated (see Table 4-4 below) . The structure of a MODS file was briefly introduced in Section 2.5. This section will give further details of this file format.

Table 4-4: The structure of the DP module

Module	**Description**
xmlGenrtor.py	Generate the required elements in the desired MODS file, extracting relevant data from “output.json” and generating the MODS file by generating a XML file and changing its extension to “.mods”.

As MODS utilizes XML we will begin by describing the relevant XML syntax. A MODS file has a number of element tags where each element has a title and a value, optional attribute(s), or perhaps both. Figure 4-12 shows an example of a MODS XML element.

```
<role>
   <roleTerm authority="marcrelator" type="code">ths</roleTerm>
</role>
```

Figure 4-12: An example of a MODS element

Each MODS element, similar to XML elements, starts and ends with the tag that identifies the element (for example, <role>...</role> and <roleTerm>...</roleTerm> in Figure 4-12). The element may have a value which is the data in between its starting and ending tags. In the example shown in Figure 4-12 the value is the string “ths”. Elements may or may not have attributes. These attributes are stated inside the starting tag. For example, authority and type in Figure 4-12. Elements may or may not have sub elements. For example, in Figure 4-12, roleTerm is a sub element of role.

Generation of a MODS file for importing into DiVA was mainly based upon the library describing the format’s specification as well as a system description of DiVA [46]. For example, according to this library, the Abstract element inside the MODS file would look like <abstract lang="eng">Abstract-text</abstract> (from [35]) or the example shown in Figure 4-13.

```
<abstract lang="eng">&lt;p&gt;With Ericsson’s goal to have optimal user experience …
&lt;/p&gt;</abstract><abstract lang="swe">&lt;p&gt;Med Ericssons mål att ha optimal
användarupplevelse … &lt;/p&gt;</abstract>
```

Figure 4-13: Another example of an abstract, in this case in two languages (from the Master’s thesis of Patrik Hagernäs, see urn:nbn:se:kth:diva-172760)

As mentioned in the DE module (Section 4.1.2), the extracted data are all stored in the ‘output.json’ file. The MODS generator module will read these files to generate a MODS file. Note that the file could begin with a line of the following form: <?xml version="1.0" encoding="UTF-8"?>. This line indicates what version of XML is used and what character encoding is used.

The MODS generator module begins by defining the root element of the MODS file (<modsColelction>) where a collection of MODS records is defined. The overall MODS file has the following form:

<modsCollection xmlns="http://www.loc.gov/mods/v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.loc.gov/mods/v3
http://www.loc.gov/standards/mods/v3/mods-3-2.xsd">
...

</modsCollection>

The attributes xmlns, xmlns:xsi, and xsi:schemaLocation define which XML name space is used (with xmlns), the Schema Instance Namespace (xmlns:xsi).

Inside the modCollection is one or more mods elements of the form:

<mods xmlns="http://www.loc.gov/mods/v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" version="3.2"
xsi:schemaLocation="http://www.loc.gov/mods/v3
http://www.loc.gov/standards/mods/v3/mods-3-2.xsd">...

</mods>

The <mods> sub element follows after with following data:

xmlns=http://www.loc.gov/mods/v3 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:xlink=http://www.w3.org/1999/xlink version=3.2 xsi:schemaLocation= http://www.loc.gov/mods/v3
http://www.loc.gov/standards/mods/v3/mods-3-2.xsd

To define these 2 elements and every other element of the file, the xml.etree.ElementTree library was used [47]. This library provides an API for accessing, creating, iterating, and updating the elements of a XML file as a tree structure. Once the 'modsCollection' element is inserted at the top of the XML tree, the module can insert a mods element and begins insertion of the meta data extracted from the CDC, DE and KDC modules.

The module uses the python os library [48] to navigate to the corresponding 'output.json' file of the current session. Thereafter, getting the relevant data from the JSON file is done by specifying the key of the data inside the JSON file. This is done by using json.load(output.json) to load the JSON data inside a variable for example VAR. each data key can then be accessed using VAR. For example, once the appropriate element with its subelements as attributes for the author has been created, the author name can be retrieved from the JSON file by VAR["author_1_frontname"]. In this particular example, the 'output.json' file holds the value of the front name of author 1 by the key that is titled as "author_1_frontname". To create an element mainly the functions xml.etree.ElementTree.Element and xml.etree.ElementTree.Element.append are used.

The function xml.etree.ElementTree.Element creates an element and makes the element's nametag the same as a string parameter. Depending on where this element is supposed to be positioned inside the XML tree structure, xml.etree.ElementTree.Element.append is used to position the element. For example, as mentioned, the root element of the mods file is <modsCollection> followed by a sub element <mods>. To create and position these two elements the following code is executed:

1. The root element creation:

```
root = ET.Element("modsCollection")
root.text ="xmlns =http://www.loc.gov/mods/v3 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://www.loc.gov/mods/v3 http://www.loc.gov/standards/mods/v3/mods-3-2.xsd "
```

2. The mods element creation:

```
mods = ET.Element( "mods")
```

Positioning the "mods" element as a sub element of the "modsCollection" element is done with:

```
root.append(mods)
```

Therefore, it can be seen that it is important to specify ParentElement.append(childElement) throughout the tree to structure the tree. All the elements that are created for the mods data of a single thesis are a child element of the same mods element. Therefore, all of them have been positioned using the code line mods.append(Element_To_Append). Of course, the child-elements of the any data-element within the mods element must be positioned under their respective parent element.

Once the elements are all created and positioned, then every element needs to be assigned its corresponding attributes and/or values as specified by the DiVA format specification. The values of each element were specified by ElementName.text("value as string") and their attributes specified as ElementName.set("the attribute title as string" , "the attribute value as string"). An example is shown below of how the abstract element was created and positioned along with setting the element's value and attributes. In this code, we assume that "absEnglish" is a variable pointing to the abstract element. This element is appended to its parent element ("mods"). Additionally, its "lang" attribute is set to "lang = eng" which specifies the language used it this abstract is English. This attribute is important as it allows us to distinguish between the Swedish and English abstract both of which should be present in every thesis document. Execution of this code will result in the element shown in Figure 4-13.

```
    absenglish = ET.Element("abstract ")

    mods.append(absenglish)

    absenglish.set("lang", "eng")

absenglish.text = content["abstract(en)"]
```

The remaining elements are implemented in a similar manner to the abstract element described above. Once all of the elements have been created, the ElementTree now contains all of the data. Finally, a string version of this data is created using ElementTree.tostring(root) and written to a newly created XML file.

Upon execution of the module, a file named as “xmlMODS.xml” is created. This file contains the elements of the XML tree. The file extension is finally changed from .xml to. mods which provides the MODS file that can be later imported by DiVA, thus, achieving the goal of providing an automated means for the process of entering the meta data into DiVA for publication of the thesis.

4.2 Integration of the Project

The backbone modules described previously need to be integrated/connected to form a complete system and provide the property of Client-Server based application. Django Framework is used to hook all the modules together and allow the program to communicate between user, Canvas. and the server that hosts the application.

The backbone modules are copied into the Django framework and the architectures are shown in Figure 4-14 and Figure 4-15. The backbones modules act as individual python modules under the ‘polls’ folder. These backbone modules are connected via ‘views.py’ and interact with the front-end user via the templates in templet folder with a link provided in ‘url.py’. Some automated integration test cases are implemented as self-check functions in ‘views.py’. Each button on the page leads to an individual function in ‘views.py’.

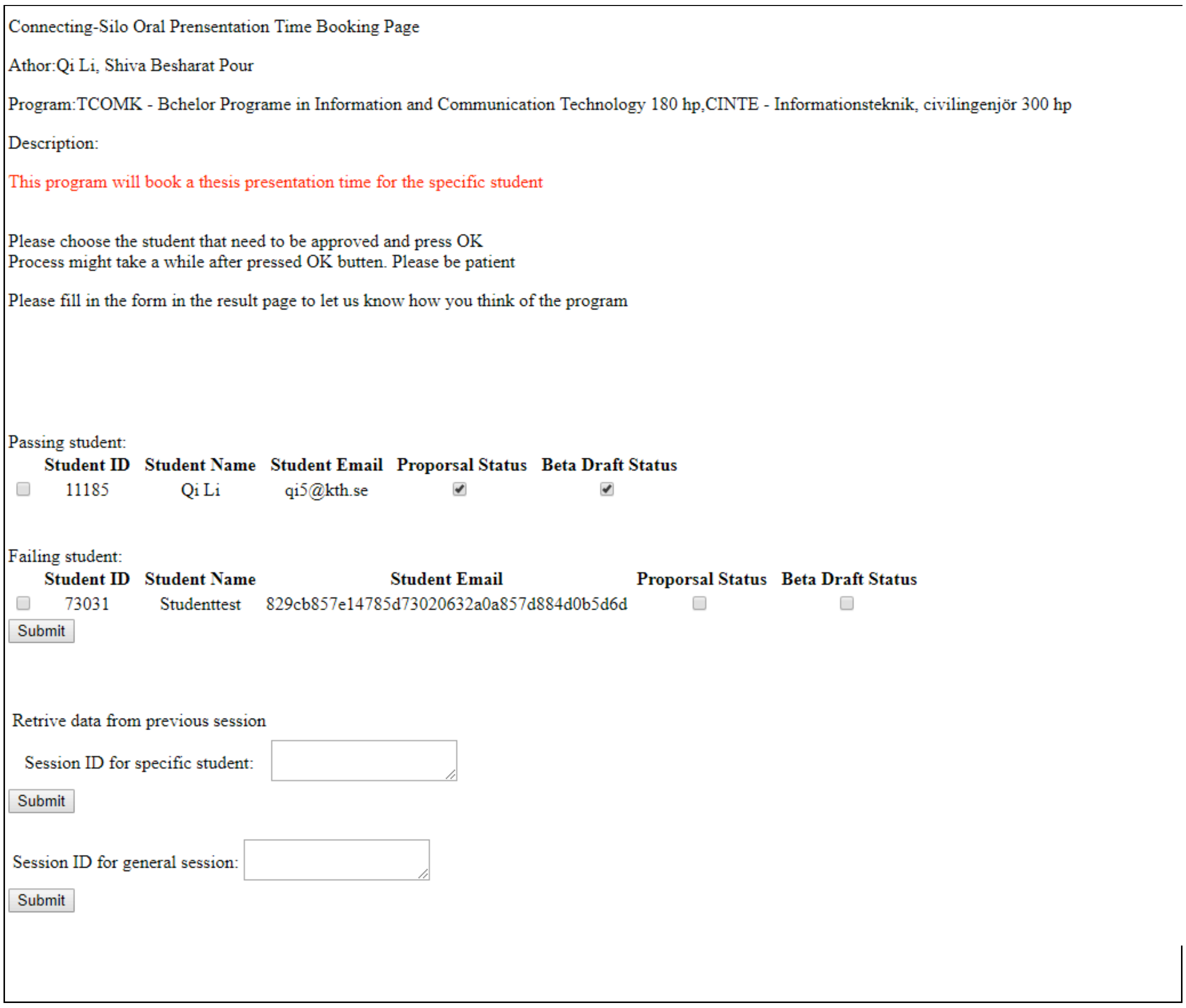
The file ‘views.py’ acts as a main function for the entire Connecting Silos project. The functions that are included in ‘views.py’ are stated in Table 4-5 on page 73. Oral presentation scheduling stage is handled inside ‘lti_proporsal’ function and Thesis approval stage is handled inside ‘lti_thesis’. The importance of the ‘lti_proporsal’ and ‘lti_thesis’ in ‘views.py’ is because these two functions control the primary interaction between user and the program. The ‘lti_proporsal’ and ‘lti_thesis’ will lead to the page shown in Figure 4-16 below. Figure 4-16 is the ‘Oral presentation scheduling’ front-end page. The ‘Thesis approval’ stage looks similar but with the addition of Final draft status. ‘Passing student’ in Figure 4-16 means that the student is qualified for booking an oral presentation (or alternatively can be processed at thesis approval stage). If student has not passed either proposal or beta draft once selected to be processed at Oral presentation scheduling stage, then the student will be categorized as a failing student. The same condition applies for the ‘Thesis Approval’ stage; the student is categorized as failing student if any of proposal, beta draft, or final thesis have not been approved by the examiner.

```
├── mysite
│   ├── __init__.py
│   ├── __init__.pyc
│   ├── settings.py
│   ├── settings.pyc
│   ├── urls.py
│   ├── urls.pyc
│   ├── wsgi.py
│   └── wsgi.pyc
├── polls
│   ├── admin.py
│   ├── __init__.py
│   ├── migrations
│   │   ├── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
│   └── url.py
└── manage.py
```

Figure 4-14: Architecture of a Django project

```
.
├── django_heroku
├── mysite
├── polls
│   ├── Documentation
│   ├── download_targz
│   ├── ffdriver
│   ├── library
│   │   ├── Canvas-master
│   │   └── pdfssa4met_original
│   ├── migrations
│   ├── output
│   │   └── parse_result
│   ├── Source
│   ├── src
│   │   ├── Canvas
│   │   ├── Diva
│   │   ├── KTH
│   │   ├── mods
│   │   └── parse
│   ├── templates
│   │   └── polls
│   └── venv
│       ├── bin
│       ├── include
│       ├── lib
│       └── local
├── polls_docs
└── test
    └── testproject
```

Figure 4-15: Django project directory architecture



Connecting-Silo Oral Prensentation Time Booking Page

Athor:Qi Li, Shiva Besharat Pour

Program:TCOMK - Bchelor Programe in Information and Communication Technology 180 hp,CINTE - Informationsteknik, civilingenjör 300 hp

Description:

This program will book a thesis presentation time for the specific student

Please choose the student that need to be approved and press OK
Process might take a while after pressed OK butten. Please be patient

Please fill in the form in the result page to let us know how you think of the program

Passing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status
☐	11185	Qi Li	qi5@kth.se	✓	✓

Failing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status
☐	73031	Studenttest	829cb857e14785d73020632a0a857d884d0b5d6d	☐	☐

Submit

Retrive data from previous session

Session ID for specific student:

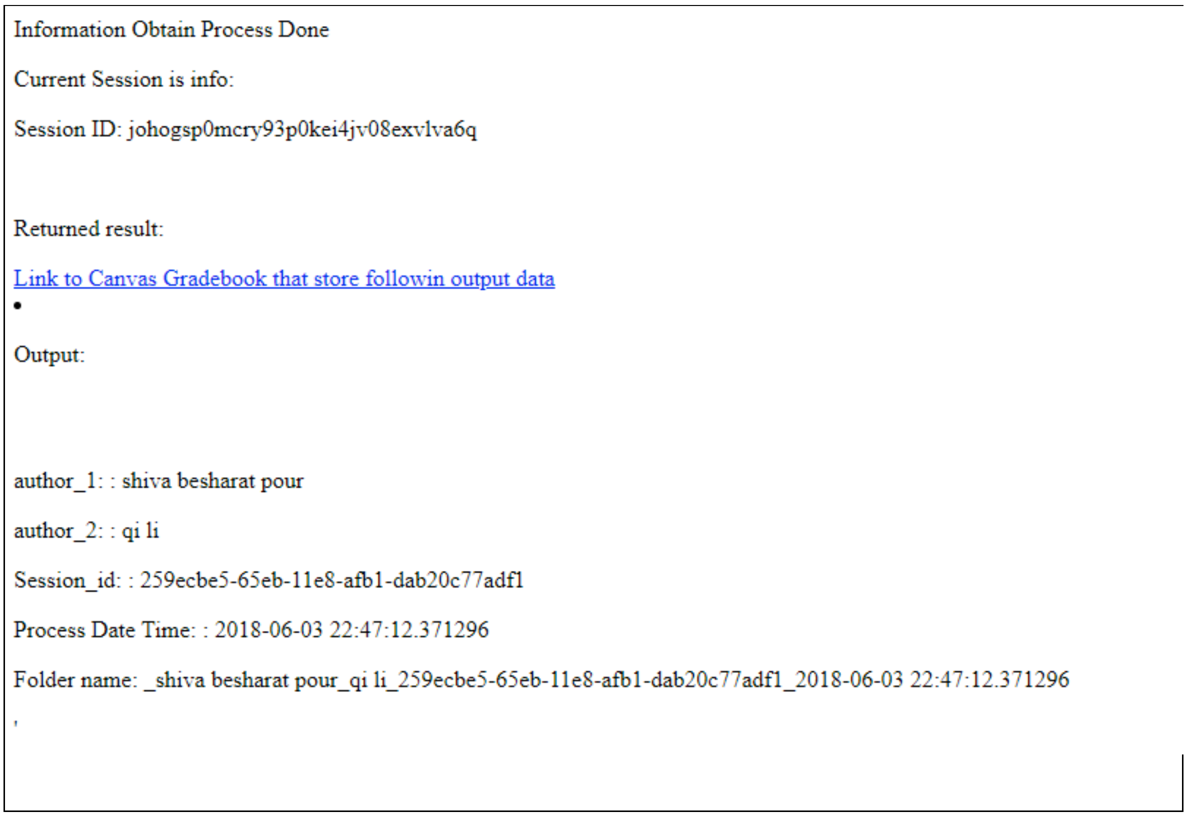
Submit

Session ID for general session:

Submit

Figure 4-16: Oral presentation scheduling stage front-end

At the end of the page shown in Figure 4-16, there are two information retrieval boxes to retrieve the information from a previous session. User can either retrieve information for a specific student or for the entire process session. The button under 'Session ID for specific student:' will trigger the function 'retrive_session_baseon_id' in Table 4-5 and retrieve the information for individual student or student pair. The button under 'Session ID for general session:' will retrieve the entire session by executing the function 'retrive_generalsession_baseon_id'. The retrieval session id for individual student or student pair and entire process session can be seen in the result output page as Figure 4-17 shows. Figure 4-17, shows an example of session ID which is the 'Session ID: johogspomcry93pokei4jvo8exvlva6q' at the beginning of the page. To retrieve an individual student or student pair, user will need to input the session id as the 'Output:' text. For example, in Figure 4-17, the session id to retrieve an individual student or student pair is 'Session_id: 259ecbe5-65eb-11e8-afb1-dab20c77adf1'.



Information Obtain Process Done

Current Session is info:

Session ID: johogsp0mcry93p0kei4jv08exvlva6q

Returned result:

Link to Canvas Gradebook that store followin output data

•

Output:

author_1: : shiva besharat pour

author_2: : qi li

Session_id: : 259ecbe5-65eb-11e8-afb1-dab20c77adf1

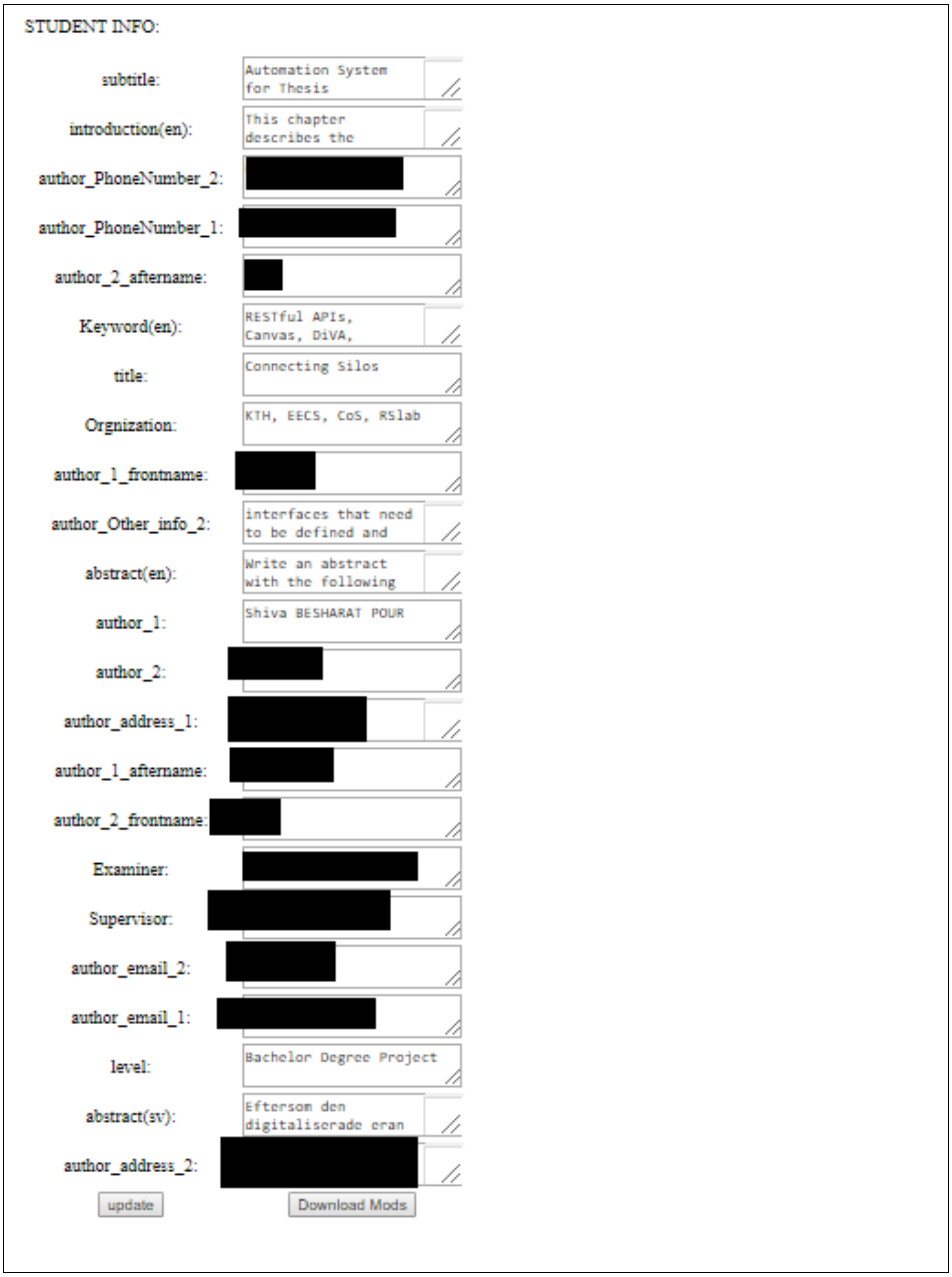
Process Date Time: : 2018-06-03 22:47:12.371296

Folder name: _shiva besharat pour_qi li_259ecbe5-65eb-11e8-afb1-dab20c77adf1_2018-06-03 22:47:12.371296

,

Figure 4-17: An example of session ID

After pressing ‘submit’ in Figure 4-16, the system will trigger the function ‘index_submit’ for ‘Oral presentation scheduling’ and ‘index_submit_thesis’ for ‘Thesis Approval’ (Table 4-5). ‘index_submit’ and ‘index_submit_thesis’ will trigger the CDC module and the DE module. The result will be sent as output into the result page which contains the information that is shown in Figure 4-18. The result output page also contains the output that both DE module and KDC module sent as output. An example of the output from KTH API can be seen in Figure 4-10 on page 62 and for DE module, Figure 4-18 below. By dragging the triangle in the right bottom corner of each text box in Figure 4-18 one can see more information inside each field. The purpose of the triangle in the right bottom corner is to save space on the page and to simplify the information presented to the user. If any data in the text box is incorrect, the user can edit the text inside the text box. Each text box is editable and the updated data will be updated in the ‘output.json’ file mentioned earlier. The system will not force the user to use the data if the data is not accurate. After the user has potentially modified the data in a specific text field, the update button will trigger the function ‘update_database’ (Table 4-5) and send the data to the sever. The user is then provided with the most recently updated data on the output page. The data under each text field is consistent with relevant data in other text fields, implying that for example, if user changes the name of author_1, the author_1_frontname and author_1_aftername will be changed too. In other words, the related data fields are linked to each other and interconnected.



STUDENT INFO:

subtitle:	Automation System for Thesis
introduction(en):	This chapter describes the
author_PhoneNumber_2:	[redacted]
author_PhoneNumber_1:	[redacted]
author_2_aftername:	[redacted]
Keyword(en):	RESTful APIs, Canvas, DiVA,
title:	Connecting Silos
Orgnization:	KTH, EECS, CoS, RSlab
author_1_frontname:	[redacted]
author_Other_info_2:	interfaces that need to be defined and
abstract(en):	Write an abstract with the following
author_1:	Shiva BESHARAT POUR
author_2:	[redacted]
author_address_1:	[redacted]
author_1_aftername:	[redacted]
author_2_frontname:	[redacted]
Examiner:	[redacted]
Supervisor:	[redacted]
author_email_2:	[redacted]
author_email_1:	[redacted]
level:	Bachelor Degree Project
abstract(sv):	Eftersom den digitaliserade eran

Figure 4-18: An example of output result page

The 'Download Mods' button in Figure 4-18 only appears in the 'Thesis Approval' stage. The 'Download Mods' function will generate a MODS file for a specific user. This file can subsequentlybe imported into DiVA. After the user presses the

'Download Mods' button, the 'modsOut' function in Table 4-5 will be invoked. The 'modsOut' function provides a download link for the user to download the MODS file and the browser's download window will thereafter pop-up.

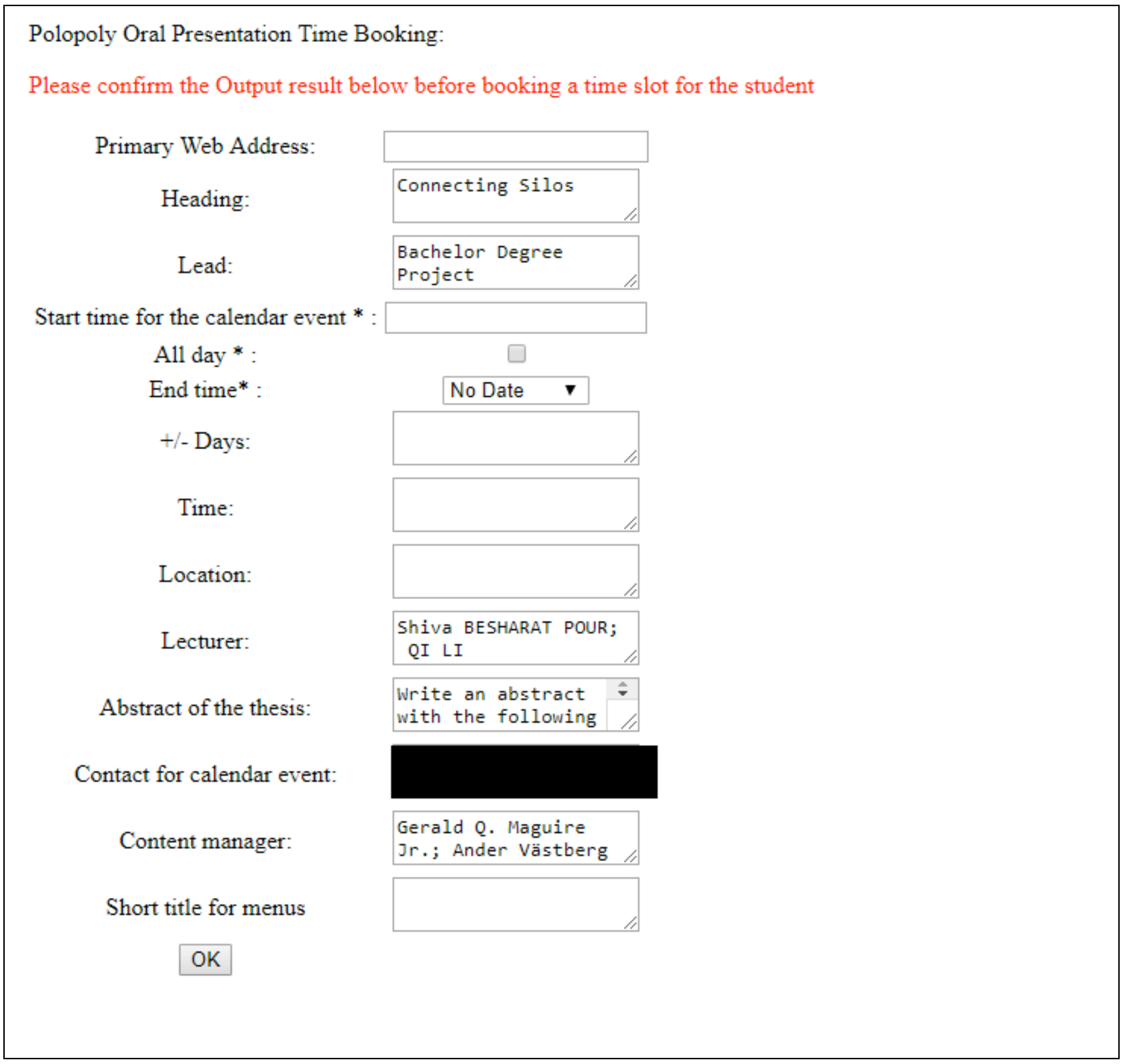
Table 4-5: List of functions of 'views.py'

Functions under views.py	
Functions	**Description**
update_database(request)	This function provides the functionality to update certain field in the result ouput page
retrive_generalsession _baseon_id(request) https://connectingsillodoc.azurewebsites.net/polls.views-pysrc.html - retrive_generalsession_baseon_id	Provides functionality to retrieve previous unfinished/finished session base on general session id
retrive_session_baseon_id(request)	Provides functionality to retrieve previous unfinished/finished session base on individual user session id
install(request)	Installation module first step
accept_form(request)	Accept course_id from install(request) function
install_2(request)	Installation module second step
install_final(request)	Installation module final step
tool_config(request)	Generate xml page for 'Booking Oral Presentation' installation
tool_config_thesis(request)	Generate xml page for 'Thesis Approval' installation

print_http_response(f)	Redirect console output to a Django HttpResponse[49] Reference: https://chase-seibert.github.io/blog/2010/08/06/redirect-console-output-to-a-django-httpresponse.html
lti_thesis(request)	Main class/first interaction with user for Thesis Approval'
lti_proporsal(request)	Main class/first interaction with user for "Booking Oral Presentation'
generate_polopoly_json(request)	Generate Polopoly json string for colander even creation
modsOut(request)	Generate MODS file and trigger download window in the browser
index_submit_thesis(request**)**	Process submitted student list by trigger canvas and process module from index_thesis.html
index_submit(request**)**	Process submitted student list trigger canvas and process module from index.html

During the 'Oral presentation scheduling' stage, the final output is the json string generated for Polopoly Calendar system. The output page in Figure 4-18 has an extra interface that is showed in Figure 4-19 below. The data is that already extracted in DE Module and from KTH API will be automatically filled in. For example, in Figure 4-19 below, the heading has already been filled in due to the title having been already extracted from the thesis proposal. The fields that do not have data from DE module and KTH API will be left empty. However, a user is allowed to fill in these fields manually, if desired.

The data in Figure 4-19 is consistent with the output result that is showed in Figure 4-18 above. A certain key with a specific value in Figure 4-18, will correspond to the data of the same field with the same key. This consistency applies for any modifications applied as well.



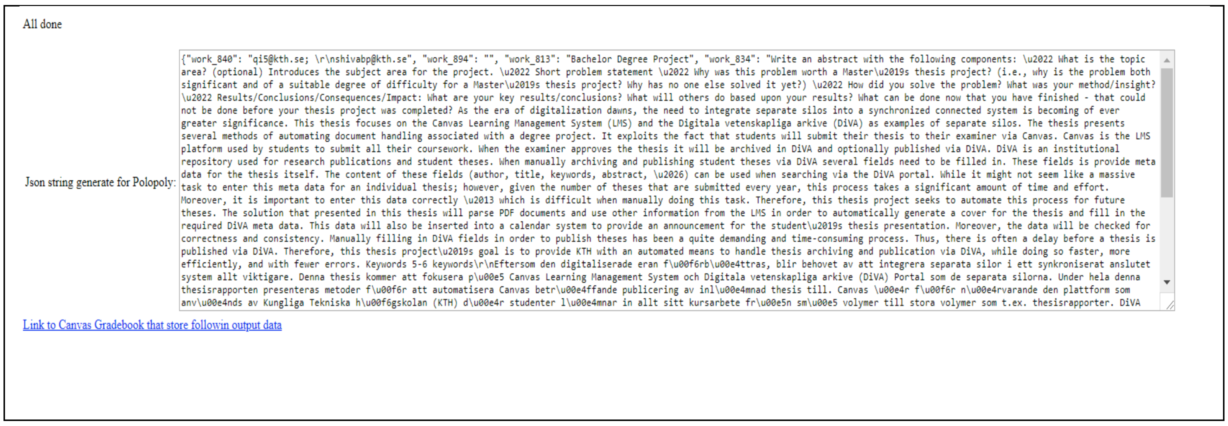
Polopoly Oral Presentation Time Booking:

Please confirm the Output result below before booking a time slot for the student

Primary Web Address:	
Heading:	Connecting Silos
Lead:	Bachelor Degree Project
Start time for the calendar event * :	
All day * :	☐
End time* :	No Date ▼
+/- Days:	
Time:	
Location:	
Lecturer:	Shiva BESHARAT POUR; QI LI
Abstract of the thesis:	Write an abstract with the following
Contact for calendar event:	[redacted]
Content manager:	Gerald Q. Maguire Jr.; Ander Västberg
Short title for menus	

Figure 4-19: Example of Polopoly calendar JSON generation interface

By pressing the OK button under the Polopoly calendar JSON generation interface (Figure 4-19), the system will start generating a JSON string based on the data in the data of the text fields of the interface. Figure 4-20, is an example Polopoly JSON result page. In this page, one can for example use the Polopoly JSON string for automation of event creation in the calendar system by sending it to the Polopoly API. The blue link under the JSON string is linked to Canvas Gradebook.



All done

Json string generate for Polopoly:

```
{"work_840": "qi5@kth.se; \r\nshivabp@kth.se", "work_894": "", "work_813": "Bachelor Degree Project", "work_834": "Write an abstract with the following components: \u2022 What is the topic
area? (optional) Introduces the subject area for the project. \u2022 Short problem statement \u2022 Why was this problem worth a Master\u2019s thesis project? (i.e., why is the problem both
significant and of a suitable degree of difficulty for a Master\u2019s thesis project? Why has no one else solved it yet?) \u2022 How did you solve the problem? What was your method/insight?
\u2022 Results/Conclusions/Consequences/Impact: What are your key results/conclusions? What will others do based upon your results? What can be done now that you have finished - that could
not be done before your thesis project was completed? As the era of digitalization dawns, the need to integrate separate silos into a synchronized connected system is becoming of ever
greater significance. This thesis focuses on the Canvas Learning Management System (LMS) and the Digitala vetenskapliga arkive (DiVA) as examples of separate silos. The thesis presents
several methods of automating document handling associated with a degree project. It exploits the fact that students will submit their thesis to their examiner via Canvas. Canvas is the LMS
platform used by students to submit all their coursework. When the examiner approves the thesis it will be archived in DiVA and optionally published via DiVA. DiVA is an institutional
repository used for research publications and student theses. When manually archiving and publishing student theses via DiVA several fields need to be filled in. These fields is provide meta
data for the thesis itself. The content of these fields (author, title, keywords, abstract, \u2026) can be used when searching via the DiVA portal. While it might not seem like a massive
task to enter this meta data for an individual thesis; however, given the number of theses that are submitted every year, this process takes a significant amount of time and effort.
Moreover, it is important to enter this data correctly \u2013 which is difficult when manually doing this task. Therefore, this thesis project seeks to automate this process for future
theses. The solution that presented in this thesis will parse PDF documents and use other information from the LMS in order to automatically generate a cover for the thesis and fill in the
required DiVA meta data. This data will also be inserted into a calendar system to provide an announcement for the student\u2019s thesis presentation. Moreover, the data will be checked for
correctness and consistency. Manually filling in DiVA fields in order to publish theses has been a quite demanding and time-consuming process. Thus, there is often a delay before a thesis is
published via DiVA. Therefore, this thesis project\u2019s goal is to provide KTH with an automated means to handle thesis archiving and publication via DiVA, while doing so faster, more
efficiently, and with fewer errors. Keywords 5-6 keywords\r\nEftersom den digitaliserade eran f\u00f6rb\u00e4ttras, blir behovet av att integrera separata silor i ett synkroniserat anslutet
system allt viktigare. Denna thesis kommer att fokusera p\u00e5 Canvas Learning Management System och Digitala vetenskapliga arkive (DiVA) Portal som de separata silorna. Under hela denna
thesisrapporten presenteras metoder f\u00f6r att automatisera Canvas betr\u00e4ffande publicering av inl\u00e4mnad thesis till. Canvas \u00e4r f\u00f6r n\u00e4rvarande den plattform som
anv\u00e4nds av Kungliga Tekniska h\u00f6gskolan (KTH) d\u00e4r studenter l\u00e4mnar in allt sitt kursarbete fr\u00e5n sm\u00e5 volymer till stora volymer som t.ex. thesisrapporter. DiVA
```

Link to Canvas Gradebook that store followin output data

Figure 4-20: Example of Polopoly JSON result page

```
{"work_840": "email1; \r\n email2", "work_894": "", "work_813": "Bachelor Degree Project", "work_834": "Write an abstract with the following components: \u2022 What is the topic area? (optional) Introduces the subject area for the project. \u2022 Short problem statement \u2022 Why was this problem worth a Master\u2019s thesis project? (i.e., why is the problem both significant and of a suitable degree of difficulty for a Master\u2019s thesis project? Why has no one else solved it yet?) \u2022 How did you solve the problem? What was your method/insight? \u2022 Results/Conclusions/Consequences/Impact: What are your key results/conclusions? What will others do based upon your results? What can be done now that you have finished - that could not be done before your thesis project was completed? As the era of digitalization dawns, the need to integrate separate silos into a synchronized connected system is becoming of ever greater significance. This thesis focuses on the Canvas Learning Management System (LMS) and the Digitala vetenskapliga arkive (DiVA) as examples of separate silos. The thesis presents several methods of automating document handling associated with a degree project. It exploits the fact that students will submit their thesis to their examiner via Canvas. Canvas is the LMS platform used by students to submit all their coursework. When the examiner approves the thesis it will be archived in DiVA and optionally published via DiVA. DiVA is an institutional repository used for research publications and student theses. When manually archiving and publishing student theses via DiVA several fields need to be filled in. These fields is provide meta data for the thesis itself. The content of these fields (author, title, keywords, abstract, \u2026) can be used when searching via the DiVA portal. While it might not seem like a massive task to enter this meta data for an individual thesis; however, given the number of theses that are submitted every year, this process takes a significant amount of time and effort. Moreover, it is important to enter this data correctly \u2013 which is difficult when manually doing this task. Therefore, this thesis project seeks to automate this process for future theses. The solution that presented in this thesis will parse PDF documents and use other information from the LMS in order to automatically generate a cover for the thesis and fill in the required DiVA meta data. This data will also be inserted into a calendar system to provide an announcement for the student\u2019s thesis presentation. Moreover, the data will be checked for correctness and consistency. Manually filling in DiVA fields in order to publish theses has been a quite demanding and time-consuming process. Thus, there is often a delay before a thesis is published via DiVA. Therefore, this thesis project\u2019s goal is to provide KTH with an automated means to handle thesis archiving and publication via DiVA, while doing so faster, more efficiently, and with fewer errors. Keywords 5-6 keywords\r\nEftersom den digitaliserade eran f\u00f6rb\u00e4ttras, blir behovet av att integrera separata silor i ett synkroniserat anslutet system allt viktigare. Denna thesis kommer att fokusera p\u00e5 Canvas Learning Management System och Digitala vetenskapliga arkive (DiVA) Portal som de separata silorna. Under hela denna thesisrapporten presenteras metoder f\u00f6r att automatisera Canvas betr\u00e4ffande publicering av inl\u00e4mnad thesis till. Canvas \u00e4r f\u00f6r n\u00e4rvarande den plattform som anv\u00e4nds av Kungliga Tekniska h\u00f6gskolan (KTH) d\u00e4r studenter l\u00e4mnar in allt sitt kursarbete fr\u00e5n sm\u00e5 volymer till stora volymer som t.ex. thesisrapporter. DiVA \u00e4r en institutionell f\u00f6rvaringsplats som anv\u00e4nds f\u00f6r thesisrapporter. D\u00e4rf\u00f6r m\u00e5ste flera f\u00e4lt fyllas i n\u00e4r man publicerar thesisrapporter fr\u00e5n Canvas till DiVA manuellt. Att fylla i DiVA-f\u00e4lt f\u00f6r att publicera thesisrapporter kanske inte verka som en stor uppgift om det bara fanns n\u00e5gra f\u00e5 antal thesisrapporter. Men med
tanke p\u00e5 det stora antalet thesisrapporter som varje \u00e5r skickas till
Canvas, kommer denna process att ta en betydande tid och mankraft som inte ska
underskattas. D\u00e4rf\u00f6r kommer detta bachelorsprojekt att fokusera
p\u00e5 att ta itu med detta problem f\u00f6r framtida dessa inl\u00e4gg.
L\u00f6sningen som presenteras i hela denna rapport kommer att best\u00e5 av
metoder f\u00f6r att analysera Portable Document Format (PDF) -dokument och
skapa ett omslag med hj\u00e4lp av den analyserade informationen och fylla i
de \u00f6nskade DiVA-f\u00e4lten automatiskt. Dessa data kommer ocks\u00e5 att
inf\u00f6ras i ett kalendersystem och kommer att kontrolleras b\u00e5de
automatiskt och manuellt, f\u00f6r korrekthet och konsistens. Manuell fyllning
av DiVA-f\u00e4lt f\u00f6r att publicera thesisrapporter har varit en ganska
kr\u00e4vande och tidsbesparande process f\u00f6r KTH under de senaste
\u00e5ren, d\u00e4r ingen automatisering f\u00f6r denna process
tillhandah\u00f6lls. Detta har lett till att en upps\u00e4ttning av
thesisrapporter som v\u00e4ntar p\u00e5 att publiceras p\u00e5 DiVA och inte
tillr\u00e4ckligt med tid f\u00f6r att n\u00e5gon anst\u00e4lld tar ansvar
f\u00f6r att ta hand om problemet. D\u00e4rf\u00f6r kommer det h\u00e4r
projektet att kunna utrusta KTH:s personal med ett automatiskt
publiceringssystem, vilket ger effektivitet till thesisrapporternas
publikation i DiVA. Nyckelord 5-6 nyckelord", "work_857": "Gerald Q. Maguire
Jr.; Ander V\u00e4stberg", "work_820": "", "work_807": "", "work_831":
"author_2;\r\n author_1", "work_824": "none", "work_827": "", "work_829": "",
"csrfmiddlewaretoken": "secret token", "work_809": "Connecting Silos"}
```

Figure 4-21: Example of Polopoly JSON result page (extended version)

Another example of using the Polopoly JSON string is showed in Figure 4-22 below. In this example, the user uses the JSON string for Selenium and browser automation. Selenium will first find the field on the Polopoly page (Figure 4-19) that has id “work_840”. When the ID “work_840” has been found, Selenium will start to automatically input the content “email_1” into the text field under of the id. After the automation has been done, Selenium will start looking for the submit button of the text field and click on it.

```
email = browser.find_element_by_id("work_840")
email.send_keys(“email_1”)
rowser.find_element_by_name("submit").click()
```

Figure 4-22: Usage of Polopoly JSON in Selenium

The functions ‘install’,’install_2’,’install_final’ and ‘accept form’ are used in Installation module (section 4.3). These four functions are connected via ‘ok’ button under each step of the installation process. The details regarding the installation module are provided in section 4.3.

4.3 Installation module

This section describes the installation of the modules of the system developed by Connecting Silos. This description is intended to facilitate the deployment and installation process of the Connecting Silo project’s resulting Canvas LTI-application by providing dynamic and automated installation process for the user. To launch the installation module, an IT administrator first needs to move the source code into the server. During the development phase of the project, the code transfer between server and developer was done via a deployment branch on GitHub. The developer pushes the code into the deployment branch and the server administrator pulls the code via the server's console. After the source code has been transferred into the server, one can launch the installation module at: *address of the server*/application_name/install'. If the installation is successful, then the user should see the page shown in Figure 4-23. This page is the first step of the installation process. The purpose of this page is to enable the user to specify the course ID of the Canvas course in which the tool needs to be installed. Currently only one course at a time may be selected. The deployment process will be time consuming if the tool needs to be installed in a large number of courses.

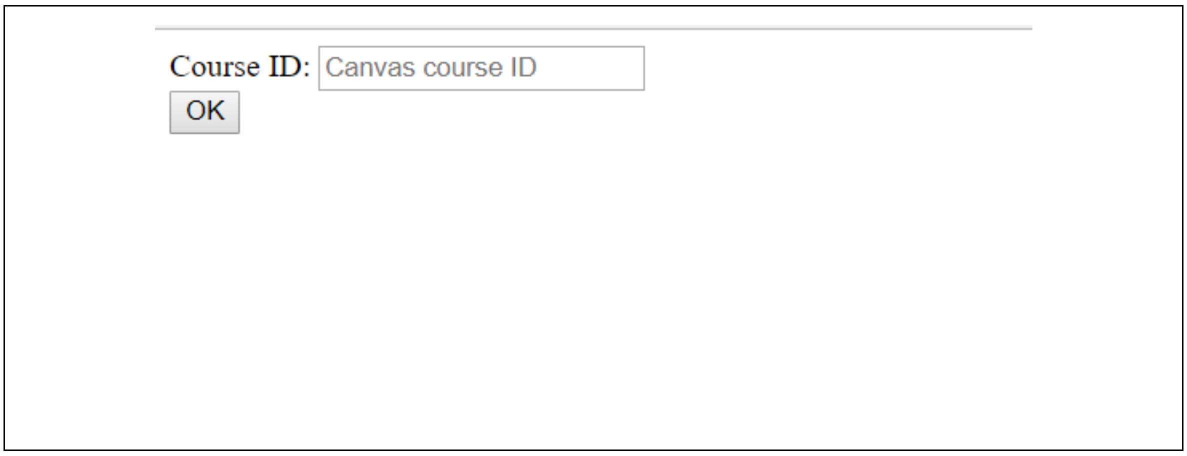
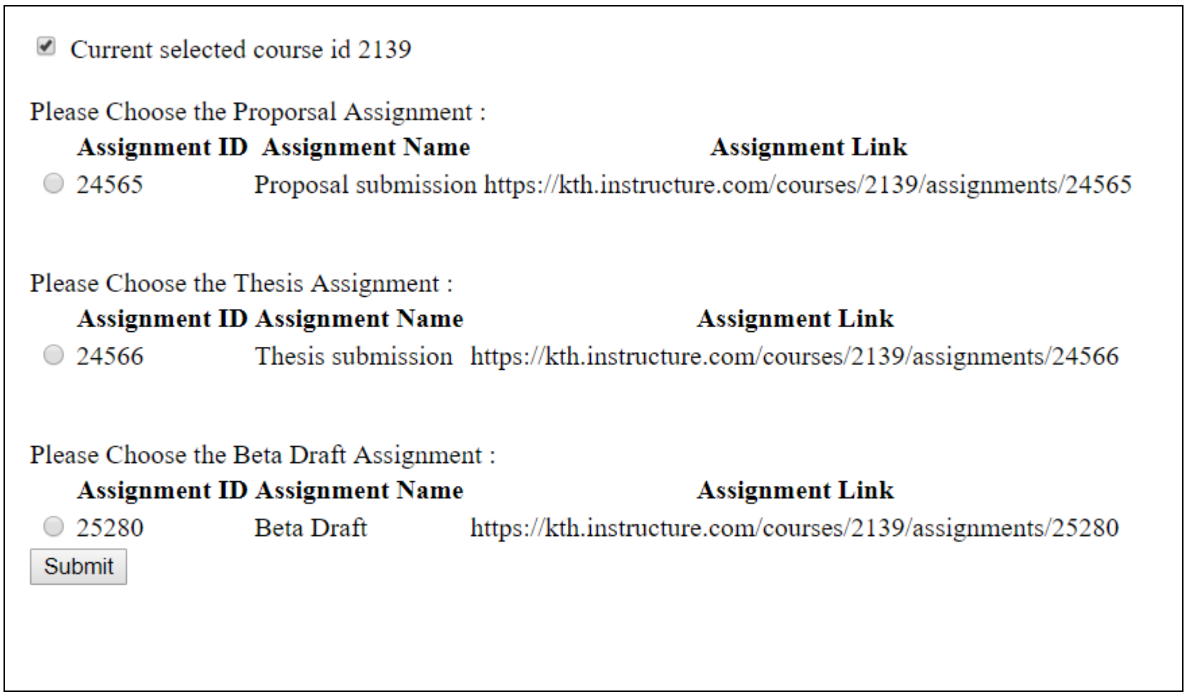

Course ID: Canvas course ID

Figure 4-23: The first step of Installation module

The next stage of the installation process is to specify which assignment was created for the proposal, Beta draft, and final draft as shown in Figure 4-24. The installation module will go through all the assignments and look for those assignments that have the keywords 'proposal', 'beta', 'final' **and** 'Thesis' in their name. In case there are several assignments with these keywords mentioned, the system lists all of the matching assignments - as shown in Figure 4-24. The user can choose among this list of assignments, which assignments to include the tool into.



☑ Current selected course id 2139

Please Choose the Proporsal Assignment :

	Assignment ID	Assignment Name	Assignment Link
○	24565	Proposal submission	https://kth.instructure.com/courses/2139/assignments/24565

Please Choose the Thesis Assignment :

	Assignment ID	Assignment Name	Assignment Link
○	24566	Thesis submission	https://kth.instructure.com/courses/2139/assignments/24566

Please Choose the Beta Draft Assignment :

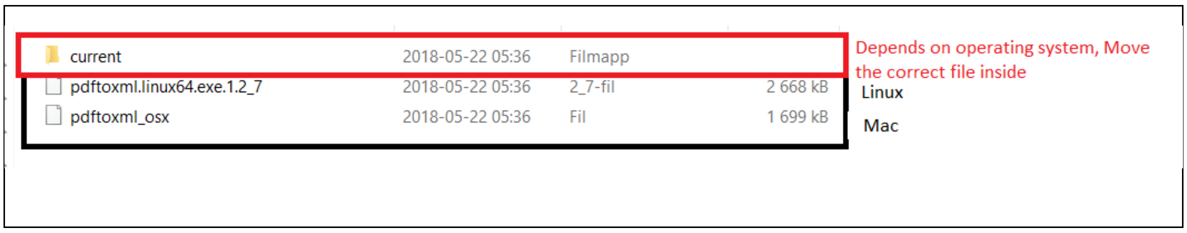
	Assignment ID	Assignment Name	Assignment Link
○	25280	Beta Draft	https://kth.instructure.com/courses/2139/assignments/25280

Figure 4-24: The assignment selection page

If the user wants to double check whether the correct assignment has been selected, the links to the assignments' pages are also shown (see Figure 4-14).

After the user has selected the assignments, the system will install the software environment for the server. This stage installs all of the external python modules using 'pip', while the installation of pdf2xml and selenium are done based on the underlying operating system and configuration of a specific system path. The goal of the Installation module is to minimize the manual setup that needs to be done after the installation module has completed its execution.

Since the DE module depends on the pdf2xml program whose location depends on the operating system of the computer, the Installation module, dynamically installs the correct version of pdf2xml. Under the directory – '<app folder>\src\parse\kth_extract\pdfssa4met\pdf2xml', there are two core pdf2xml files: 'pdftoxml.linux64.exe.1.2_7' is for Linux operating system and 'pdftoxml_osx' is for Mac OSX operating system. Currently the dynamic download module does not support Windows yet, but Windows compatibility is one of the future development goals. If the dynamic pdf2xml installation module detects that the server is running a Linux operating system, then the 'pdftoxml.linux64.exe.1.2_7' file will be copied into the current folder and a similar procedure occurs for Apple's Mac OSX. Figure 4-25 shows the structure of the pdf2xml folder. The settings file for the DE module has a link pointing to the current folder so that the DE module knows where each function can access the pdf2xml core.



```
current                      2018-05-22 05:36   Filmapp
pdftoxml.linux64.exe.1.2_7   2018-05-22 05:36   2_7-fil    2 668 kB
pdftoxml_osx                 2018-05-22 05:36   Fil        1 699 kB
```

Figure 4-25: The structure under pdf2xml folder in DE module

Once the environment has been successfully set up, the Installation module will send request to Canvas via Canvas API to install the 'Booking Oral Presentation' and 'Thesis Approval' LTI application. The configuration for 'Booking Oral Presentation' and 'Thesis Approval' is stated in two different 'payload tuples'. An example of a payload tuple can be seen in Figure 4-26.

```
# payload_proposal= {
          'name': app_title,
          'privacy_level': "public",
          'description': app_description,
          'consumer_key': "N/A",
          'shared_secret': "",
          'course_navigation[text]': app_title,
          'course_navigation[default]': "enabled",
          'course_navigation[enabled]': "true",
          'course_navigation[visibility]': "public",
          'config_type': "by_url",
          'config_url': config_url,
          'custom_fields[assignment_id]': assignment_id,
          'custom_fields[assignment_id_beta]': assignment_id_beta

     }
```

Figure 4-26: Payload for 'booking oral presentation

Figure 4-27 shows the LTI application configuration that uses the content of the variable 'app-title' as its name, sets the privacy level as public, adds a description with the content of 'app_description', activates the course_navigation placement, and sets the name of the button to be the contents of 'app_title' and eventually sets the configuration type to obtain XML configuration via URL and sets the URL as the content in 'config_url'. In the last two lines of the payload, (show in Figure 4-26) there are two custom fields that provide the assignment id for the proposal and Beta draft assignments.

The Canvas LTI application allows triggering of the LTI application in four different ways. Different triggering mechanisms provide different information as feedback. If a user wants extra information to be sent from the LMS to the LTI application, then the user needs to specify this information in the configuration manually. To be able to provide a configuration field manually, the user passes in a custom field as shown in the last two lines of Figure 4-26.

In figure 4-16, one can see the configuration method as the URL obtains XML configuration under 'config_type' and the source of the XML is under the data of ‘config_url’ key. Tool_config is a function provided by add-on ‘django-lti-provider 0.3.3’ [42] that creates such XML and provides a URL for LMS and LTI Interface to load the XML configuration. ‘Django-lti-provider 0.3.3’ is one of the Django plug ins that adds LTI application support to the development environment and can be installed via ‘pip’[42]. An example of tool_config can be found in Figure 4-27.

```
#tool config proporsal -creat xml config
#reference: https://github.com/pylti/lti
def tool_config(request):
    app_title = 'Proporsal Approval'
    app_description = 'KTH Automation proporsal'
    launch_view_name = 'lti_CS_launch'
    launch_url = request.build_absolute_uri(reverse('lti'))

    # maybe you've got some extensions
    extensions = {
        'my_extensions_provider': {
            # extension settings...
        }
    }
    print app_title

    lti_tool_config = ToolConfig(
        title=app_title,
        launch_url=launch_url,
        secure_launch_url=launch_url,
        extensions=extensions,
        description = app_description

    )

    return HttpResponse(lti_tool_config.to_xml(), content_type='text/xml')
```

Figure 4-27: Example code for tool config

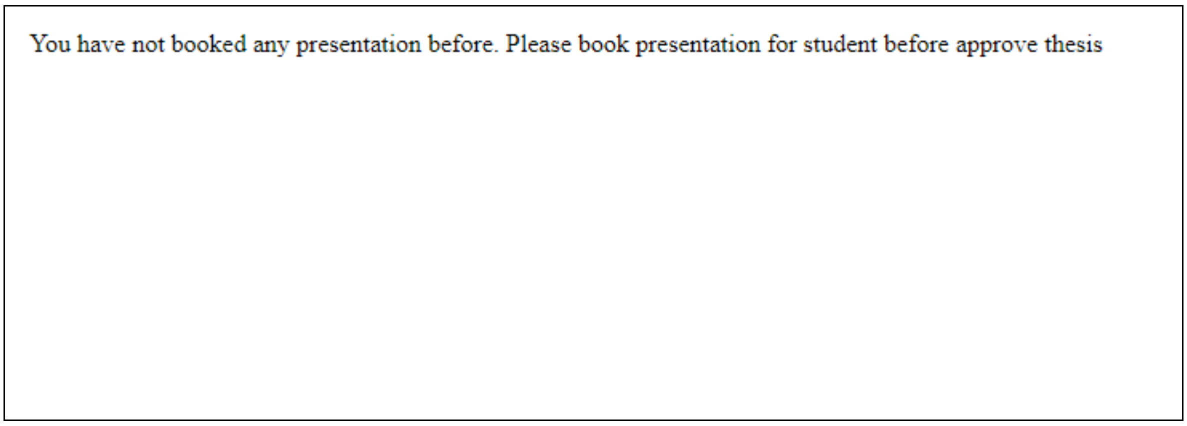
Figure 4-27 shows example code for tool config. The title for the LTI application has been set to 'Proposal Approval', application description is 'KTH Automation proposal', and the launch URL is dynamically set to the current server’s URL. The dynamic URL set up has also been used in the payload method of setting up the LTI application. By using ‘request.build_absolute_uri(reverse('lti'))’, the Django framework will detect the address of the module inside the parameter of the ‘reverse’ function and use it as the output of the ‘request.build_absolute_uri’.

Some fields in tool_config and payload cannot be read during initial installation sequence. To be able to create a value for the unreadable fields and use the functions these fields provide, the user needs to obtain the id of the external tool and send a ‘PUT’ request to update the setting of the LTI application. One example is the ‘course_navigation’ field. If this field has been sent via an installation request to Canvas, then Canvas will not read it. So, this field needs to be updated later during the update stage.

After the LTI installation has completed, the Installation module will send a Canvas external tool information obtaining a request with the external id from the application that was just created. If the response of the request is 200 and content exists, then the application is successfully created, and the front-end will notify the user of a successful installation. Any other response code will be treated as installation failure and an error message will be sent to the front-end to notify the user.

4.4 Test Automation (TA) sub-module

The TA submodule aims to provide system self-check functionality. The reason for it is called a submodule instead of a module is because the TA submodule exists in each of the backbone modules stated in section 4.1 above as a unit testing module. One can see in Table 4-6 below, the 'unit_testing' folder exists in both CDC and DE modules as a submodule.[*] The reason that only CDC and DE modules have TA submodule is because DP module has very few test cases amongst which the test-cases that are marked as "Integrated" in table Table 3-3, are self-check functions implemented inside the DP module's source code. Some test cases have been tested based on developer's manual observations and the integration of some automated test-cases are left for future work. Current implementation of the TA submodule is sufficient for the program to run properly and provide informative error reports to the user when there is a failure. In 'view.py' and 'proporsal.py', there are integration testing modules implemented as checking algorithms inside the code. These self-check algorithms are using the test cases specified for both unit testing and integration testing specified in Section 3.4.2. If any test case fails in TA submodule, an error report will be shown. Figure 4-28 shows the error report for failing the I_6 integration test case ('The 'Oral presentation scheduling' stage has to be executed before 'Thesis Approval') (see Table 3-4). The error message in Figure 4-28 was printed when the developer was testing the test case I_6.



You have not booked any presentation before. Please book presentation for student before approve thesis

Figure 4-28: Example of error report for a failed test case

In Table 4-6, one can see the project structure for the TA submodule. Since the Integration testing is spread out in different modules and source code files, the

[*] KDC module has only 1 test case which is combined with CDC test- cases since both modules are used for data collection.

integration test-cases do not have a specific file devoted to them. The error message that is shown in Figure 4-28 is generated from 'index_error.html'. If any test case fails, this html file will be sent as feedback to the user with the error message for the test case's failure. The error message will include the reason that the test case has failed. Some error messages will even specify the specific test that has failed. The reason that the error message includes the reason and the name of failing test case is to help the developer as well as future users, such as IT staff at KTH, to identify and fix the problem.

Table 4-6: The structure of the TA submodule

Module	Description
templet.index_error.html	HTML templet for showing error message to the user
Canvas/unit_testing	Unit_testing module for CDC (and KDC) module unit test cases
Parse/unit_testing	Unit_testing module for DE module unit test cases

4.5 External Packages

External packages provide extra functionality that can be directly used in the program. An external add-on is downloaded via the Python Package Manager pip. The functionality and usage of each external package are specified in Table 4-7 below.

The project Installation module already downloads the core packages based on the user's operating system and provides the basic setup as explained in Section 4.3. The instructions for installing the rest of the packages required are described in Section 3.4.1.

Selenium and PyVirtualDisplay are left for future implementation. These two external modules provide the user with an option to use the output of the project. Since Selenium is based on a graphical interface, PyVirtualDisplay creates a virtual display to let Selenium do the automation.

Table 4-7: External packages used in the project

Package	**Description**
Django	Django framework described in section 4.2
lxml	XML generation by specifying the tree structure of the XML.
PyVirtualDisplay	Python virtual disaplay. Not used in the project but used for setting up selenium for future work.
requests	Add functionality to send POST, GET and PUT requests to specific API or IP address
Selenium	Browser Automation Engine. Not used in the project. The project set up for user for future automation of polopoly json string and autofill of DiVa fields
urllib3	Same functionally as requests but more advanced. Depends on the usage, urllib3 and requests are used in different places.
pandas	An Excel document generator
lti	Provides LTI functionality to the program
eulxml	Python module used for interaction with an XML file and its tree structure
simplejson	A module used as JSON encoder and decoder

5 Achievements and analysis

The purpose of Connecting Silos project as stated in Sections 1.3 and 1.4 is to provide the fundamentals of an automated system that can be used to speed the currently manual administration process for degree projects and theses. The sections below will discuss whether the project has achieved its goals and how accurate and reliable the program is.

5.1 Achievement of project goals

As outlined in Section 1.4, the Connecting Silos project was expected to achieve the following 2 sub-goals:

1. Once an examiner has scheduled an oral presentation, the proposed extension to Canvas will automatically extract the relevant information needed to create a Calendar event for a given degree project presentation based upon the submitted beta draft and the time and place of the scheduled presentation.
2. Once an examiner has approved a thesis submitted via Canvas, the relevant information will be extracted from the thesis itself and combined with other data that is available in Canvas to automate the full process of publishing theses via DiVA.

Based on the Chapter 4 of this thesis report, upon successful installation of the LTI application developed by the project, a Canvas page will be augmented with two plugins (named 'Schedule Oral Presentation' and 'Thesis Approval') as shown in Figure 5-1 shown below.

The first sub-goal was to provide an automated means for creating a calendar event based on information provided by the beta draft of the thesis and information about the time, place, date, etc. of the oral presentation of a degree project. Ideally this information aout be injected into the calendar system by using the Polopoly API. The thesis referred to this sub-goal as "Oral presentation scheduling' throughout the development phase. The completion of this stage would imply that the first sub-goal has been achieved. Hence, once the user (an administrative staff member) has installed the application for the specific course that is devoted to degree projects for the term, the respective Canvas course will be customized with the "Schedule Oral Presentation" LTI add-on as shown in Figure 5-1. By clicking on this add-on, the user will be directed to the page shown in Figure 5-2. The preceding actions and functionalities are described in Section 4.2. Eventually, upon termination of this stage of the program, the user will be provided with a JSON string that can later be used to automate the presentation event creation via the Polopoly API, hence completing sub-goal 1. However, at present this final step is not yet automated, so a user has to manually import the content into a Polopolgy calendar event.

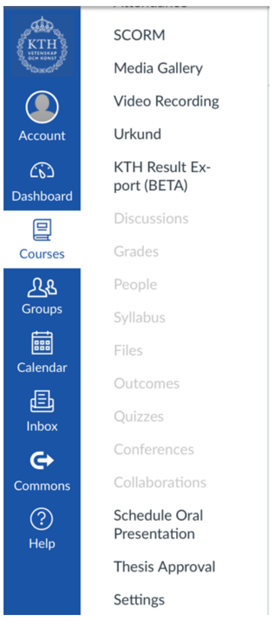
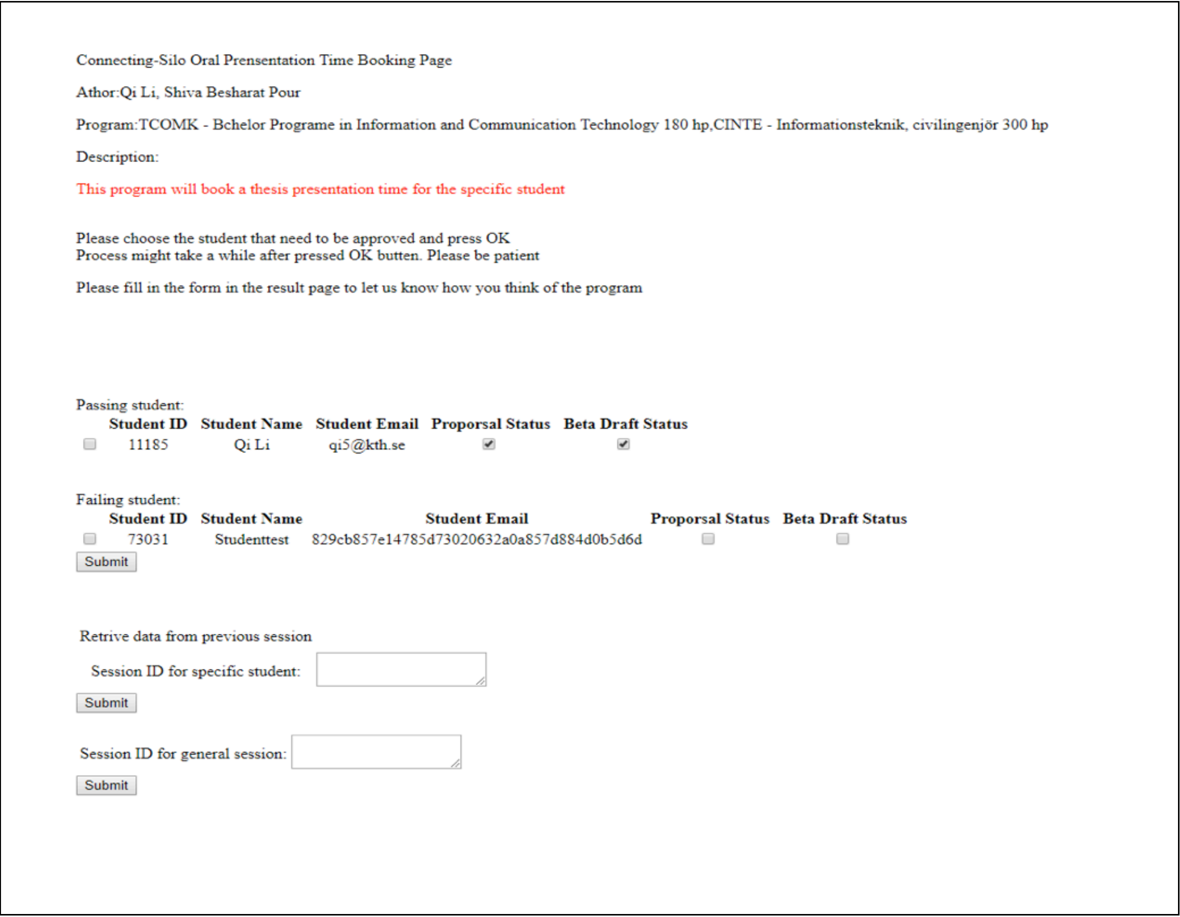


Figure 5-1: The assignment selection page


Connecting-Silo Oral Prensentation Time Booking Page

Athor:Qi Li, Shiva Besharat Pour

Program:TCOMK - Bchelor Programe in Information and Communication Technology 180 hp,CINTE - Informationsteknik, civilingenjör 300 hp

Description:

This program will book a thesis presentation time for the specific student

Please choose the student that need to be approved and press OK
Process might take a while after pressed OK butten. Please be patient

Please fill in the form in the result page to let us know how you think of the program

Passing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status
☐	11185	Qi Li	qi5@kth.se	✓	✓

Failing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status
☐	73031	Studenttest	829cb857e14785d73020632a0a857d884d0b5d6d	☐	☐

Submit

Retrive data from previous session

Session ID for specific student:

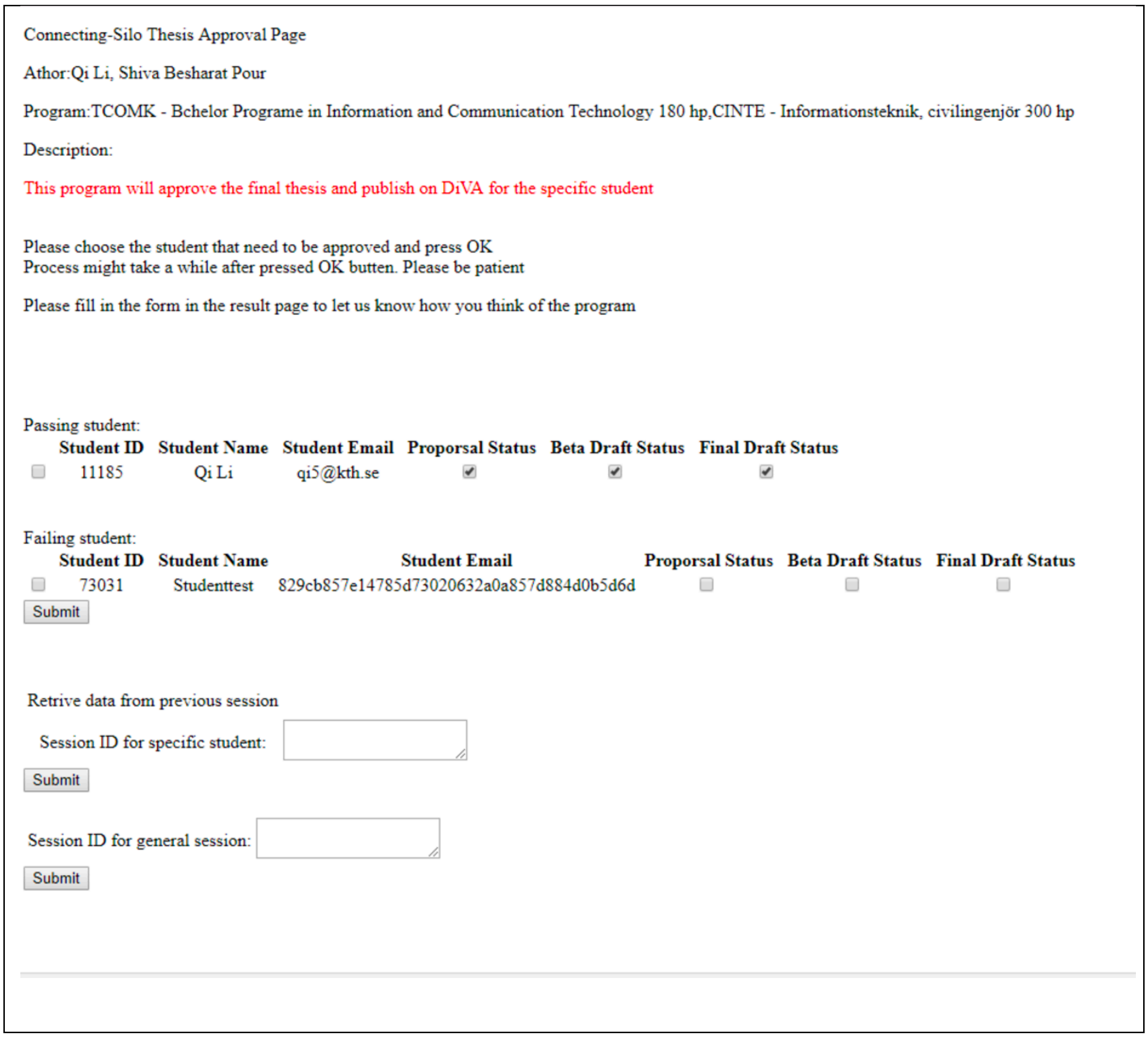
Submit

Session ID for general session:

Submit

Figure 5-2: Oral presentation scheduling page

In addition to the‘Schedule Oral presentation’ button shown in Figure 5-1, there is an additional add-on (button) titled ‘Thesis Approval’. This is the add-on developed by the project for the purpose of the second sub-goal regarding extracting information for automation of thesis publication via DiVA. This sub-goal was referred to as the “Thesis Approval” stage and was developed into the add-on named “Thesis Approval”. This add-on is installed together with “Schedule Oral Presentation” stage add-on; hence, the installation process only needs to be performed once in order to install both add-ons. Once the user clicks on “Thesis Approval”, the user will be taken to the page shown in Figure 5-3. This realizes most of sub-goal 2. The remaining procedures to be performed on the “Thesis Approval” page are further described in Section 4.2. These procedures terminate with the generation of a MODS file containing all of information to be imported into DiVA.


Connecting-Silo Thesis Approval Page

Athor:Qi Li, Shiva Besharat Pour

Program:TCOMK - Bchelor Programe in Information and Communication Technology 180 hp,CINTE - Informationsteknik, civilingenjör 300 hp

Description:

This program will approve the final thesis and publish on DiVA for the specific student

Please choose the student that need to be approved and press OK
Process might take a while after pressed OK butten. Please be patient

Please fill in the form in the result page to let us know how you think of the program

Passing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status	Final Draft Status
☐	11185	Qi Li	qi5@kth.se	✓	✓	✓

Failing student:

	Student ID	Student Name	Student Email	Proporsal Status	Beta Draft Status	Final Draft Status
☐	73031	Studenttest	829cb857e14785d73020632a0a857d884d0b5d6d	☐	☐	☐

Retrive data from previous session

Session ID for specific student:

Session ID for general session:

Figure 5-3: Thesis Approval page

5.2 Reliability and validity analysis

Section 3.4.3 proposed the approaches that the project used to verify the reliability and validity of the program. This verification is very important since the developed system handles sensitive student and staff information. Therefore, uncertainties regarding the validity and reliability of the program may have considerable consequences for especially for the administration staff and everyone whose data is involved.

Validity of the program was tested using the test cases presented earlier in Section 3.4.2. As mentioned earlier, not all of these tests were integrated inside the automated Test Automation (TA) module and not all of them were automated because it was more suitable to manually perform some test cases. However, the team members make sure to perform all the test cases, in order to verify the validity of the program. Of course, it was not always the case that the system passed the test in the first round of testing. That is why, the team constantly improved the system until no test case failed or terminated with an error report. This claim can be confirmed by looking at the "Status" column of the test case tables where all test cases' status is marked with "Pass". Thus, the validity of the developed system is confirmed.

The reliability of the program was tested using the SHA-1 hash code experiment performed for 3 rounds (see Section 3.4.3). The program passed all the rounds, meaning that the hash of the results of the manual process when compared to the automated system were identical; thereby, the automated system was proven to be reliable.

6 Conclusions and Future work

This chapter begins by presenting the conclusions drawn by the project team members after developing the automated system for the two goals: (1) to provide an automated means for publishing student thesis in DiVA and (2) to create a calendar event for a student's oral presentation in KTH's calendar system via the Polopoly API. As confirmed in Chapter 5, a major part of these means are now ready to be successfully applied. Moreover, they were validated and their reliability has been confirmed. The purpose of automating some of the degree project administration processes is mainly to provide improved efficiency over the currently manually performed actions required when administrating degree projects. The program has proved to have speeded up the process as will be explained in Section 6.1; however, it may have disadvantages due to issues that would not have occurred in the manual administrative process. The respective drawbacks and advantages of the program will be discussed and compared against each other in Section 6.1. Furthermore, the limitations that the program imposes on users will be outlined in Section 6.2 as well as their causes and effects on the usage of the program.

Due to the limited timespan of the project as well as limited access to some tools and libraries (as well as relevant APIs), the project only some of the desired functionality has been implemented. The rest remains for future work. The specific tasks that are required in order to complete the automation system will be outlined in Section 6.3.

Finally, Section 6.4 will conclude this thesis with a brief discussion of the social, economic, and ethical aspects of this Connecting Silos project.

6.1 Conclusion

Section 1.1 mentioned that manual processing of degree projects to enter them into DiVA takes roughly 1 hour per thesis. This process now has been partially automated, thus reducing the time needed for the whole process to roughly 50 seconds. Upon program termination, a MODS file is generated that can be imported into DiVA. However, since this last step has been left for future work, we can only estimate the additional time that will be required.

Realistically, it is safe to say that the complete process of thesis publication in DiVA will be a matter of few minutes, and most likely less than 5 minutes. Thereby the process has been speeded up by the difference of 60 minutes compared to 5 minutes which is 55 minutes. This corresponds to a 92% speedup for publishing a single degree project in DiVA.

If we extrapolate this from an example year, in this case 2017. Table 1-1 suggests that the total number of thesis that was submitted during this year was 2287 theses at KTH. Manually publishing 2287 degree projects in DiVA at roughly an hour per thesis would require 2287 hours. To comprehend the significance of this time, 2287 hours is approximately 95 days for faculty & staff to process degree projects and upload them to DiVA for a single year. Moreover, this huge amount of this work is concentrated towards the end of every academic year.

To estimate the time required when using the automation system developed by the Connecting Silos project, we will assume 5 minutes (as was estimated above to be the worst case) for the complete process for one thesis. This correspond to roughly 191 hours. One case see that this is a huge difference as 191 hours corresponds roughly to 8 days (rounded upwards). Basically 1 week of work for the same number of degree projects (2287) for 1 year, compared to 95 days (approximately 3 months) for the same process when done manually. Therefore, the speed up goal of the "Thesis approval" stage of the project, corresponding to sub-goal 2 (Section 1.4) has been achieved. This is despite the fact that the process has not been completely automated, with further automation the time can be reduced even further.

It is also worth mentioning that although the most time consuming part of administrating degree projects is publication via DiVA, it seems realistic to assume that manually scheduling an oral presentation takes perhaps several minutes which in aggregate will be a quite considerable amount of time for the large number of oral presentations done each year. As the automation system, outputs a JSON string of the required data for event creation in roughly 48 seconds – if there were an automated means to inject this into Polopoly, then the time difference would be considerable when considering this savings occurs for ~2000 degree projects every year. Thus, the speedup goal for the "Oral presentation scheduling" stage of the project corresponding to sub-goal 1 (Section 1.4) has also been achieved. In summary, the project has (basically) achieved its goals.

6.2 Limitations

There are some Limitation imposed on the users of the system developed by the project. The system requires the degree-related submissions to be submitted to Canvas as opposed to being sent to the examiner via email. The reason for this requirement is quite obvious as otherwise the system would be unable to access these submitted documents to process them.

The system is using some tools in the background that have different versions and differ in compatibility with different operating systems. The system developed by the project can currently install compatible version of these tools (such as pdf2xml mentioned in Section 2.10.1) dynamically based on the operating system but only for Linux and Mac OS. Moreover, the system has not been tested on a computer running Windows. Therefore, a Windows user would need to manually install the Windows-version of these operating system-dependent tools. The tools that require manual installation for windows are pdf2xml and Python Selenium (Section 2.9.1).

The installation process explained in Section 4.3 has been tested only for one course; however, it will require more time if this needs to be performed for multiple courses. However, the installation only needs to be done once, and the installaiotn time will be small in comparison with the amount of time saved later on by automatically handling some of the degree project administration.

The JSON files that the event-creation automation and DiVA publication automation generate, were not actually tested by importing them into by Polopoly or DiVA respectively. This is due to the fact that the project team did not have access to the required APIs and therefore this is left for future work.

6.3 Future work

The automated system implemented for publishing student thesis in DiVA generates a MODS file containing the data required by DiVA. These MODs files can be imported into DiVA given that they have the correct structure and element definition as required by DiVA. However, as the team did not have access to the DiVA API to complete the insertion process; thus this final step in automation is left for future work. Note that in the absences of a DiVA API developers may take the generated MODS file and then use python Selenium and PyVirtualDisplay (see Section 2.9.1 and Table 4-7) in order to emulate a user manually importing an entry (thus further automating the insertion of the content into DiVA).

A large amount of the data that is required by DiVA to complete the publication process is provided by the generated MODS file; however, some of the desired information has not been extracted. Due to the limited duration for the project, further data extraction remains for future work.

As mentioned in Section 3.4.2, some of the test cases are automated but not yet integrated in the Test Automation (TA) module. The system developed by project has been tested for these test cases and all others until each test was successfully passes. However, in the future a developer might wish to automate all of these test-cases and integrate them inside the TA module.

6.4 Reflections

Subsections below will reflect on the Social as well as Economic and Ethical aspects of the program developed respectively.

6.4.1 Social aspects

The General Data Protection Regulation (referred to by GPDR in short) is a regulation introduced by the European Union for the purpose of protecting individual data as well as enhancing privacy.[50] Since the automated system is directly accessing personal information about KTH students and staff, it is important to keep in mind that the system must not violate GPDR. The project respects GPDR by avoiding storage of personal data at any stage. Moreover, the project follows KTH regulations by using a KTH database as a storage of the final results (the MODS file contents or calendar entry) will be stored in a KTH data repository that is managed by the university to ensure that it is compliant with GDPR).

This source code for the project will be published to benefit society by providing an efficient means to that similar administrative processes can exploit this work. Therefore, future work or any desired modifications may be made by anyone who desires to continue developing this automation system further.

6.4.2 Economic aspects

Of course, such time-taking process of administrating degree projects, specially concentrated on the last period of the academic year requires significant amount of time and human resources if done manually. Moreover, Table 1-1 and Table 1-2 give an indication of the potential considerable cost of this process considering the total amount of work that needs to be done. The efficiency provided by the automation will provide significant economic benefits for similar processes.

6.4.3 Ethical aspects

Automatically handling administration process for every degree project has an advantage that can be considered as an Ethical aspect of the program. A computer program handling degree projects, does so by treating every degree project equally. Every degree project will be input to the same procedures, algorithms and functions.

This may not however always be the case if degree project administration is done manually. For example, misspelling and or mistakes that may occur for administrating one degree project may not necessarily occur in another.

Since degree project administration is dealing with sensitive personal data and student work, making sure that every degree project is treated equally and with respect to correctness as well as time required is essential.

References

[1] R. K. Ellis, “Learning Managament Systems,” Alex. VI Am. Soc. Train. Dev. ASTD, 2009.

[2] D. Kuhlman, A Python Book: Beginning Python, Advanced Python, and Python Exercises. Platypus Global Media, 2011.

[3] “Degree project | Bachelor’s Programme in Information and Communication Technology (TCOMK, 180 cr) | KTH.” [Online]. Available: https://www.kth.se/social/program/TCOMK/page/degree-project-5/. [Accessed: 30-Mar-2018].

[4] “Degree project process | Bachelor’s Programme in Information and Communication Technology (TCOMK, 180 cr) | KTH.” [Online]. Available: https://www.kth.se/social/program/TCOMK/page/routine-for-degree-project/. [Accessed: 30-Mar-2018].

[5] Atlassian, “Jira | Issue & Project Tracking Software,” *Atlassian*. [Online]. Available: https://www.atlassian.com/software/jira. [Accessed: 28-May-2018].

[6] Soumen Chakrabarti *et al.*, “Data Mining Curriculum: A Proposal (Version 1.0),” 30-Apr-2006.

[7] Margaret Rouse, “What is data mining? - Definition from WhatIs.com,” *SearchSQLServer*, 01-Dec-2008. [Online]. Available: http://searchsqlserver.techtarget.com/definition/data-mining. [Accessed: 01-Apr-2018].

[8] “ORCID.” [Online]. Available: https://orcid.org/. [Accessed: 30-Mar-2018].

[9] KTH, Department of Leaerning, Language and Communication Unit, “Frequently Asked Questions,” 28-Apr-2014. [Online]. Available: https://www.kth.se/en/ece/avdelningen-for-larande/sprak-och-kommunikation/verksamhet/tandem/frequently-asked-questions-1.378107. [Accessed: 05-Apr-2018].

[10] “Learning Management Systems | KTH.” [Online]. Available: https://www.kth.se/en/student/kth-it-support/learning-platforms/learning-management-systems-1.517117. [Accessed: 28-Mar-2018].

[11] “LTI Advantage FAQ | IMS Global Learning Consortium.” [Online]. Available: https://www.imsglobal.org/lti-advantage-faq#what-is-LTI-Advantage. [Accessed: 28-May-2018].

[12] Instructure Global Ltd., “Canvas by Instructure,” 05-Apr-2018. [Online]. Available: https://www.canvaslms.eu/. [Accessed: 28-Mar-2018].

[13] “Canvas | KTH.” [Online]. Available: https://www.kth.se/en/student/kth-it-support/learning-platforms/canvas/canvas-1.784659. [Accessed: 28-Mar-2018].

[14] “Canvas LMS REST API Documentation.” [Online]. Available: https://canvas.instructure.com/doc/api/index.html. [Accessed: 28-Mar-2018].

[15] “What is SpeedGrader? | Canvas Community.” [Online]. Available: https://community.canvaslms.com/docs/DOC-10712. [Accessed: 28-Mar-2018].

[16] “DiVA portal is a finding tool for research publications and student theses written at the following 47 universities and research institutions.” [Online]. Available: http://www.diva-portal.org/smash/aboutdiva.jsf?dswid=-1313. [Accessed: 28-Mar-2018].

[17] “About DiVA | KTH.” [Online]. Available: https://www.kth.se/en/kthb/publicering/kths-publikationsdat/om-diva-1.569302. [Accessed: 28-Mar-2018].
[18] “Metadata Object Description Schema: MODS (Library of Congress Standards).” [Online]. Available: http://www.loc.gov/standards/mods/. [Accessed: 28-Mar-2018].
[19] “MODS: Uses and Features (Metadata Object Description Schema: MODS).” [Online]. Available: https://www.loc.gov/standards/mods/mods-overview.html. [Accessed: 28-Mar-2018].
[20] C. Smith, “SwePub MODS metadata format specification,” p. 43.
[21] “Formatspecifikation - DiVA: info - Confluence.” [Online]. Available: https://wiki.epc.ub.uu.se/display/divainfo/Formatspecifikation. [Accessed: 03-Jun-2018].
[22] “Relator Code and Term List -- Term Sequence: MARC 21 Source Codes (Network Development and MARC Standards Office, Library of Congress).” [Online]. Available: https://www.loc.gov/marc/relators/relaterm.html. [Accessed: 03-Jun-2018].
[23] “bibutils,” *SourceForge*. [Online]. Available: https://sourceforge.net/projects/bibutils/. [Accessed: 03-Jun-2018].
[24] “User’s guide to Polopoly | KTH.” [Online]. Available: https://intra.kth.se/en/administration/kommunikation/webbpublicering/polopoly/manual/att-jobba-med-polopoly-1.8432. [Accessed: 28-Mar-2018].
[25] “KTH Book Cover Generator.” [Online]. Available: https://intra.kth.se/kth-cover?l=en. [Accessed: 30-Mar-2018].
[26] “What is PDF? Adobe Portable Document Format | Adobe Acrobat DC.” [Online]. Available: https://acrobat.adobe.com/us/en/acrobat/about-adobe-pdf.html. [Accessed: 28-Mar-2018].
[27] “InfoSec Handlers Diary Blog - PDF Babushka,” *SANS Internet Storm Center*. [Online]. Available: https://isc.sans.edu/diary.html. [Accessed: 28-Mar-2018].
[28] E. Kunnas, PDF Structure and Syntactic Analysis for Metadata Extraction and Tagging: https://code.google.com/p/pdfssa4met/. 2017.
[29] “Selenium - Web Browser Automation.” [Online]. Available: https://www.seleniumhq.org/. [Accessed: 20-Apr-2018].
[30] geckodriver: WebDriver <-> Marionette proxy. Mozilla, 2018.
[31] “Installing Packages — Python Packaging User Guide.” [Online]. Available: https://packaging.python.org/tutorials/installing-packages/. [Accessed: 20-Apr-2018].
[32] “PyPDF2 Documentation — PyPDF2 1.26.0 documentation.” [Online]. Available: https://pythonhosted.org/PyPDF2/. [Accessed: 30-Mar-2018].
[33] “10.10. shutil — High-level file operations — Python 2.7.15rc1 documentation.” [Online]. Available: https://docs.python.org/2/library/shutil.html. [Accessed: 24-Apr-2018].
[34] “The Web framework for perfectionists with deadlines | Django.” [Online]. Available: https://www.djangoproject.com/. [Accessed: 28-May-2018].
[35] “Django documentation | Django documentation | Django.” [Online]. Available: https://docs.djangoproject.com/en/2.0/. [Accessed: 28-May-2018].
[36] “How To Use the Django One-Click Install Image for Ubuntu 16.04,” *DigitalOcean*. [Online]. Available:

https://www.digitalocean.com/community/tutorials/how-to-use-the-django-one-click-install-image-for-ubuntu-16-04. [Accessed: 03-Jun-2018].
[37] "Python Data Analysis Library — pandas: Python Data Analysis Library." [Online]. Available: https://pandas.pydata.org/. [Accessed: 22-Apr-2018].
[38] P. Lopez, pdf2xml convertor based on Xpdf library: modified version. 2018.
[39] "pdf2xml download | SourceForge.net." [Online]. Available: https://sourceforge.net/projects/pdf2xml/. [Accessed: 28-May-2018].
[40] "Agile project management – the what and the why | APM." [Online]. Available: https://www.apm.org.uk/blog/agile-project-management-the-what-and-the-why/. [Accessed: 22-May-2018].
[41] "SHA-1 Hash Code Generator | My Tec Bits." [Online]. Available: http://www.mytecbits.com/tools/cryptography/sha1generator. [Accessed: 06-Jun-2018].
[42] django-lti-provider adds LTI functionality for the Django web framework. This work began as a port of MIT&#39;s LTI Flask Sample, which demonstrates a sample LTI provider for the Flask Framework ba.. Columbia Center for Teaching and Learning, 2018.
[43] "OpenCraft," *GitHub*. [Online]. Available: https://github.com/open-craft. [Accessed: 22-May-2018].
[44] "DigitalOcean: Cloud Computing, Simplicity at Scale," *DigitalOcean*. [Online]. Available: https://www.digitalocean.com/. [Accessed: 22-May-2018].
[45] "Chip sandbox." [Online]. Available: https://kth.instructure.com/courses/11. [Accessed: 22-May-2018].
[46] "Formatspecifikation och systembeskrivning - DiVA: info - Confluence." [Online]. Available: https://wiki.epc.ub.uu.se/display/divainfo/Formatspecifikation+och+systembeskrivning. [Accessed: 28-May-2018].
[47] "19.7. xml.etree.ElementTree — The ElementTree XML API — Python 2.7.15 documentation." [Online]. Available: https://docs.python.org/2/library/xml.etree.elementtree.html. [Accessed: 28-May-2018].
[48] "15.1. os — Miscellaneous operating system interfaces — Python 2.7.15 documentation." [Online]. Available: https://docs.python.org/2/library/os.html. [Accessed: 28-May-2018].
[49] "Redirect console output to a Django HttpResponse," *Chase Seibert Blog*, 06-Aug-2010. [Online]. Available: https://chase-seibert.github.io/blog/2010/08/06/redirect-console-output-to-a-django-httpresponse.html. [Accessed: 03-May-2018].
[50] "General Data Protection Regulation (GDPR) – Final text neatly arranged," *General Data Protection Regulation (GDPR)*. [Online]. Available: https://gdpr-info.eu/. [Accessed: 06-Jun-2018].