UNIVERSITY OF READING

UNDERGRADUATE THESIS

Beamer Slide Composition

Author:
Joseph Dyer

Supervisor:
Dr Julian Kunkel

A thesis submitted in fulfilment of the requirements
for the degree of BCs Undergraduate of Computer Science

in the

Reading University
The School of Mathematical, Physical and Computational Sciences

July 9, 2019

Declaration of Authorship

I, Joseph Dyer, declare that this thesis titled, "Beamer Slide Composition" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

UNIVERSITY OF READING

Abstract

The School of Mathematical, Physical and Computational Sciences

BCs Undergraduate of Computer Science

Beamer Slide Composition

by Joseph Dyer

This report describes the action taken to improve the reuse of individual slides from presentations, intending to aid the dissemination of knowledge especially in an academic setting. This goal is to be achieved by managing large collections of presentations made with LaTeX, using machine learning to automate archiving and improve retrieval. This report documents the design and development process of a candidate created to prove the usefulness of such a system.

Acknowledgements

I would like to express my gratitude to Dr. Julian Kunkel for his support and guidance through this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, the current situation is analysed for where any problems arise and suggest possible solutions. Also, goals are set for this project to solve a problem that is found and define how the thesis will explore the creation of the solution.

1.1 Motivation

This report describes a of unit work with the aims to investigate how to increase productivity in reusing existing presentation material and produce a prototype candidate to implement the ideas.

The current standard method for building a presentation is to write it from scratch or to copy and paste slides from existing presentations which entails work to make it fit the style.

It would more advantageous to build new presentation mixing and matching from many existing presentations which could produce much better results.
Without any tools to aid the search for presentations would see the user having to search through possibly hundreds of files that have promising titles and then assessing each slide one at a time. While no tools are needed to perform the process, it is very time consuming and also reliant on user skill to accurately assess the presentations for their content from only a file name.

As presentations in academia are often recycled (for example lecture courses are usually heavily based on last year's material) academics time is much better spent by leveraging work already produced. Therefore, a better solution can be provided for users that allows for quick access to old content. By adding metadata tags that describe a summary of topics of each presentation would significantly speed up the search and give far more detail than just a file name on its own. The search could be achieved by using a database of presentations that can then be searched via their metadata to give a subsection of presentations which relate to the given topic.

The assessing of each presentation by their name can then be much more targeted but is still time-consuming as the presentations still need to be manually searched for useful content which might only be contained in a few slides. A problem of this approach is how accurately the user can assign the topics. If there are too few metadata tags assigned to presentations, then the results are too limited. Alternatively, too many tags would produce such a vast quantity of results to invalidate the purpose of the search in the first instance.

If instead of summarising each presentation, individual slides are instead summarised and given metadata tags then the user can search with a much higher focus and removes the need to scan large presentations. The slide search can be implemented by expanding the database to include slides, so a search produces presentations and an index to the useful slides. However, this is still a time-consuming process and gives a problem of manageability due to the necessity of obtaining the data from the list of search results, which could be in the hundreds or even thousands of slides. The list of slide locations also presents an issue in transferring the required content from the search results into the user's new presentation.

Therefore a tool that could retrieve the required slides from a presentation is not feasible with the more popular file types, such as Powerpoint (.pptx). Instead, using LaTeX allows the reading and manipulation of the source without issue. LaTeX also easily facilitates for 'copy and paste' to be a viable method of export, as it is formatted via tags these can be transferred entirely.

To ensure the success of the tool, there is a new requirement for every slide in a presentation to be tagged with metadata. This prerequisite is unlikely to be completed due to the sheer amount of additional work required by the user. It also raises essential consistency issues with tagging, for example, individual users are likely to value data differently, and one presentation could have one tag per slide with another having thirty tags per slide. An excellent way to extract topics would be to use artificial intelligence to review each slide and assign keywords. This algorithm needs to be selected carefully as often slides will have a limited amount of text to analyse to ensure an accurate assignment. As single word topics are quite narrow and subjective, help should be provided with additional similar suggested topics when searching, such as a search for pets also includes the words animals and dogs.

Currently, large amounts of people are using LaTeX to produce presentations, about 18% of hard science papers are produced with it [1]. The popularity can be attributed to LaTeX's many useful features such as the ability to show animations and complex artefacts with little end-user formatting [2]. LaTeX is also written as a plain text format which makes it very accessible for scripting in both generating the documents and modifying after creation [3] [4].

Existing tools only provide the ability to store, use and share as a whole presentation. This new tool should be able to retrieve individual slides from all stored presentations based on some key-words. By providing global cloud locations for storage of slides this will be able to facilitate the sharing of presentations between a wide range of people.

1.2 Goals

The aim of this thesis is to create a tool that aids the creation of new presentations, by providing access to a large collaborative repository of slides found via their topics covered. From the user's perspective, this should be achieved with as little effort as possible. Six core objectives need to be completed to provide a minimum set of functionality, which is capable of fulfilling this need. These are:

1. **Large slide storage space**
   The tool will need to be able to have the capacity to archive a large number of slides.
2. **Database of metadata**
   The metadata that will be generated needs to be stored in a structured way which is easily searchable, so the tool knows what slides to suggest.
3. **UI previews of slides**
   The slides are visual in nature, to enable them to be evaluated properly they must be displayed graphically.
4. **Automated file handling**
   The tool needs to be able to archive and retrieve slides without user intervention.
5. **Automated metadata creation via ML analysis**
   Each slide added with the tool must gain metadata to be searched by so it can be retrieved at a later date.
6. **Formatting to consistent style or template specified template**
   Once the new slides are selected they should be presented with a consistent format.

1.3 Thesis outline

This thesis thus aims to explore the process of creating a tool that can fulfil the requirements stated. The preceding chapter has outlined the problem space and the general shape of a possible solution. In the proceeding, chapters will define the design, implementation and verification work completed to building the tool. Next, in Chapter 2, the existing software and literature for technologies to be

leveraged are explored in detail. Then in Chapter 3, the design will plan the tool and lay out a set of detailed requirements to be achieved. Chapter 4 will describe the process that will be employed to ensure the quality of the tool to be prototyped. Chapter 5 will discuss the specifics in the implementation that occurred in the tool and how unexpected problems were handled; this leads to Chapter 6 in which the implemented tool is tested to assess its ability and verify its functionality. Finally, Chapter 7 will conclude the thesis, which will include an evaluation to judge its successfulness and then suggest improvements.

Chapter 2

Literature review

This chapter will focus on reviewing existing literature and software available. The sources were found to allow a basis of knowledge that allows innovative work to be built.

2.1 LaTeX

According to the LaTeX Wikipedia article, "LaTeX is a high-quality typesetting system; it includes features designed for the production of technical and scientific documentation. LaTeX is the de facto standard for the communication and publication of scientific documents." [5]

LaTeX provides a markdown style type set to write documents. This is achieved by composing the document in plain text with tags to describe the formatting which is then compiled into a new file type, most commonly PDF. The format later nature has the advantage of easy editing and allows the user to concentrate on content while the software handles the formatting. This feature is particularly useful for preventing edits from completely changing and therefore ruining the document. The stability povided has made LaTeX a very attractive choice when writing formal documents with strict design requirements, such as scientific papers [1]. LaTeX also has ready-made tools to support the creation of presentations, which are as feature rich as their contemporaries. The fact that LaTeX is written in plain text makes it much easier to manipulate programmatically and supports the use of custom tools for managing presentation slides.

2.1.1 LaTeX Compilers

As LaTeX is formatted via keywords the raw source files are hard to review. Therefore to be reviewed these should be compiled to display the documents in a friendly format for a user.

This is a laborious operation that should use a pre-existing tool to complete the task. Therefore, the task was handled by the compiler distributed on home page of LaTeX, due to the fact it used by most other editors.

Other compliers are available, two of the most popular are MikeTex [6] and Tex Live [7]. However most come with some form of editors which is unnecessary or are made to run on Unix systems when this tool will target Windows operating systems.

2.1.2 LaTeX example

The following listing 2.1 is a short LaTeX source code for an example:

LISTING 2.1: lst:Short LaTeX example

```
\documentclass{article}
\usepackage{graphicx}

\begin{document}

\title{Introduction to \LaTeX{}}
\author{Author's Name}
\maketitle

\begin{abstract}
The abstract text goes here.
\end{abstract}

\section{Introduction}
Here is the text of your introduction.

\subsection{Subsection}
Simple equation
\begin{equation}
    \label{simple_equation}
    \alpha = \sqrt{ \beta }
\end{equation}

\end{document}
```

The example snippet listing 2.1 can produce the following page fig. 6.1 as a PDF file.

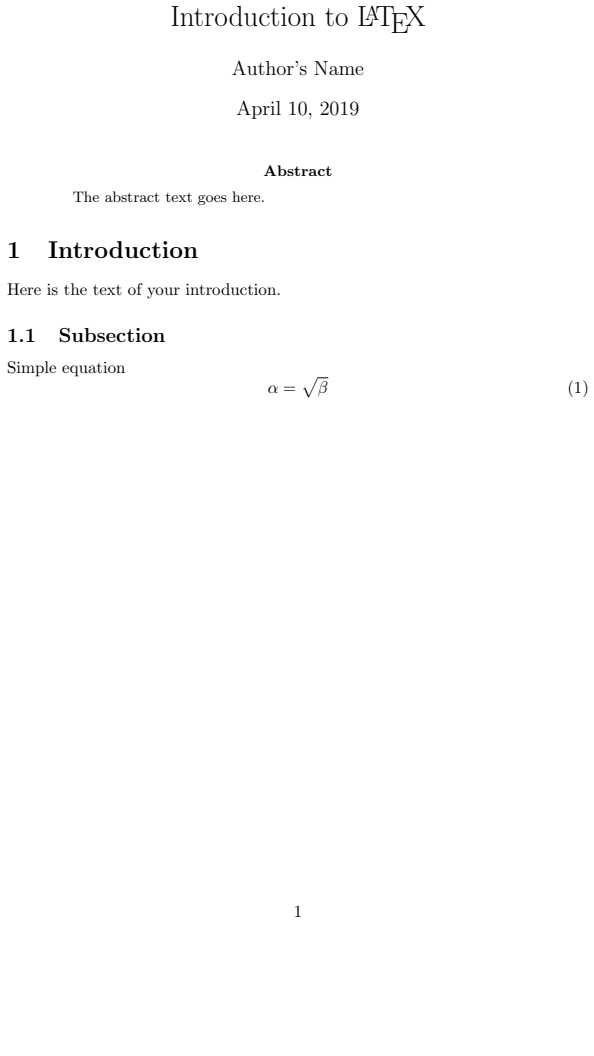

FIGURE 2.1: fig:Example LaTeX slide

2.2 Existing tools

The earliest forms of presentation management, when presentation software was first being created in the late 1970s, would be the file name and file system organisation. The advent of GUIs from the late 1980s to the 1990s gave the ability to review presentations more easily. Once Windows Vista released in 2008, file explorer gained the ability to display a thumbnail which gave some ability to see what is in a file or presentation before opening. As discussed earlier, relying on file names and file structure to find individual slides is a time-consuming

task demanding the need for dedicated software to manage presentations. Due to the importance of presentations and their sheer quantity, there already exists a range of technologies to organise file content and present it in a usable format. These have gradually evolved into more and more competent services, some of the more popular examples will be reviewed now.

2.2.1 Slide Share

Slide Share [8] is the official LinkedIn presentation sharing service and one of the most widely used tools for transmitting slides. Several key features are present that would be excellent for this paper's tool. The most prominent being its search engine, which is critical for finding presentations that are incidentally linked to the topics given. The searching is particularly useful as the search does not rely on the specific key terms but can infer the user's desired content. The second advantage is the global nature of the tool as it is a single repository of slide decks shared by everyone; such level of collaboration is something that should be replicated. By providing a shared repository in this project, the usefulness would increase due to the plethora of more slides to choose from. The final advantage of Slide Share is its quick preview of slides. Once selected, the presentation can be instantly viewed without having to launch an external viewer. A complete workflow contained in the tool makes it much easier to review large amounts of content quickly.

Unfortunately, there are several flaws in using Slide Share as a slide deck building tool. The first being the tool relies on users to provide most of the metadata to give accurate search results, which often is just a title and short description. The searches also lacks any granularity on a slide level meaning presentations have to be searched slide by slide. The final major disadvantage is the presentations are stored in a variety of formats which makes it hard to incorporate a slide directly into a new presentation easily, and often the file types are not conducive to editing, such as PDF.

2.2.2 Copac Collection Management

Copac Collection Management (CCM) [9] is a project to provide better management of presentations and other teaching materials. CCM has a strong focus on completing internal collection by purchasing material from other organisations and removing unnecessary material to free up resources. This ensures that organisations can manage their data through a toolset that facilitates efficient reviewing and searching for presentations. As such, it needs to be able to search presentations, find specific subjects and present similar materials so that users can review presentations on similar topics at the same time. It facilitates collaboration between different users' organisations by integrating their presentation collections into one search, increasing efficiency in finding applicable presentations.

CCM carries extensive metadata of presentations that are focused on assigning its worth and the tools to review extensive collections and assess presentations.

2.2.3 ShareTex and Overleaf

There have also been several tools created for the creation and editing of LaTeX documents. The leading example is the combined services of ShareTex [10] and Overleaf [11], which provide an online collaborative real-time editor with excellent links to storage options. As such, it provides live compilation for its raw LaTeX to a preview, making it very easy to design a LaTeX document. It also has integration to Git and other archiving tools to provide good backups and allow collaboration between separate organisations, on top of the collaboration in the editor presented. The tools integration makes retrieving the section of wanted LaTeX to be reused in a new project very easy, however, the LaTeX snippets still have to be found manually with only the file names and file structure to help.

2.3 Natural language processing (NLP)

Ever since computers have been invented there has always been ideas to allow them to understand language and communicate as humans do. The father of computing, Alan Turing famously proposed his test in 1950 to judge when machines had breached this milestone; the Turing Test was given in his paper "Computing Machinery and Intelligence" [12]. The latest date that the Turing test was achieved was during a competition held by the University of Reading, where the chatbot Eugene Goostman convinced 33% [13] of experts they were talking to a human. However, there have been previous attempts, such as Cleverbot [14], managing to convince 55% of the general public that it was human in 2011. The software used to pass the Turing Test is not adequate for this paper as the test is now considered to be a naive understanding of human language as it only tests a surface level understanding that focuses more on the illusion of a conversation. To allow machines to extract more meaning, other software has been designed to perform natural language processing (NLP).

An early success of NLP came from the program STUDENT [15], which was a highly limited program that could read and solve textbook algebra questions and give an answer. One example is "The sum of two numbers is 96, and one of the numbers is 16 larger than the other. Find the numbers" gives the outputs "One of the numbers is 56" and "The other number is 40" [15]. There were several other notable NLP attempts from the period such as SHRDLU [16] and ELIZA [17].

A breakthrough came in the 1980s, when NLP shifted from using ever increasing complex handwritten rules to use machine learning instead. The less hands on approach proved more successful as the effort of a human defining every rule in

a language, which often has at least one exception, is more suited to a machine which can compute a vast amount of data faster than a human. Eventually, this develops to building more fuzzy sets of rules based on the statistical analysis of a text, which better suits the fluidity of language. The effectiveness increased as computing power became ever more abundant to process the massive amounts of data necessary and can utilise more complex algorithms, starting with large lists of "if else" statements to neural networks with millions of neurons.

2.3.1 Graph-based ranking algorithms

One of the critical advantages of graph-based ranking algorithms is their unsupervised learning which allows the use of extensive collections of unannotated data that is abundant everywhere. Another advantage is its ability to analyse a corpus of text without any prior training required, making it especially useful when analysing the text of a niche topic. These represent sentences as nodes in a graph and edges represent the similarities. What constitutes a similarity is the central area of study along with how to represent it as an edge.

Two of the first successful graph-based ranking algorithms came in the form of Google's PageRank [18] and Kleinberg's HITS algorithm [19]. PageRank was Google's first attempt at designing a web crawler to build their search engine and focused on linking web pages with particular attention given to the links between pages. PageRank excels at sorting the importance between different pages and extracting key pieces of information. A similar system would be excellent to review slides in a presentation for the proposed tool.

PageRank is optimised to analyse web pages and would not find any meaning in objects unconnected by hyperlinks. Instead, a new tool modified from PageRank was introduced in 2004 called TextRank [20] that was designed to work on plain text. TextRank modifies the processes of PageRank to allow it to study text corpora. The approach works by taking the corpus of text and breaking it down into parts, for this project this would be individual words, each of them becoming a vertex on a graph. The edges are then drawn which can be directed. The greatest advantage of TextRank and its approach over other attempts is the lack of domain knowledge required to understand the text it is presented. Other methods such as supervised machine learning algorithms require to be taught on large corpus that are relevant to the text that will be summarised. As well as being far more accurate when compared to leading efforts utilising other underlying principles such as that proposed in 'Improved automatic keyword extraction given more linguistic knowledge' [21] which achieves an accuracy of 13% when TextRank can achieve 33% accuracy.

2.3.2 Training graph-based ranking algorithms

While TextRank uses graph-based ranking algorithms to build a new graph for every corpus given, a graph can also be built to be later queried. The methods to do this originated at the same time as the above use with the SMART system [22]. The ability of graph-based ranking algorithms to be trained in an unsupervised manner is a significant boon to training. The unsupervised nature means large quantities of data can be collected without particular care to noting what it relates to. Large data sets are critical to the accuracy of any machine learning attempt, to give them the broadest view and reduce any statistical outliers in the data collected. Unannotated data is also unlikely to be corrupted by the user implementing the system with their limited understanding or biases.

Graph-based ranking algorithms also have the advantage of only consuming large amounts of resources when a model is being trained. Once the graph has been built and trained, it can be reduced to a set of vectors that can be easily transported, making it a lightweight method of including machine learning analysis. The compactness is much better for this project as it will need to be included with the code base putting sharp limits on the suitable size of any models included.

Chapter 3

Design

The design has completed from the start of the development to allow a planned approach to be undertaken during the tool's creation. The documents produced will be referenced during the building of the tool and provide requirements to know when the project is completed

3.1 Requirements

The requirements have been made by consulting multiple different sources.

Firstly, the motivation to know what functionality to include and then the literature review for the technology available to use. Inspiration has also been drawn from existing tools, questions to likely users, tutorials such as the guide for dummies [23], personal experience and feedback from supervisor and peers. The listed goals from Chapter 1, were as follows:

1. **Large slide storage space**
2. **Database of metadata**
3. **UI previews of slides**
4. **Automated file handling**
5. **Automated metadata creation via ML analysis**
6. **Formatting to consistent style or template specified template**

This can be used to build an initial design to help organise the separate pieces of the tool. From this initial discussion, a more detailed plan can then be formed.

For point 1, there are two possible avenues to provide the storage, the first is to use the machine the program is running on and the other is to use a cloud-based storage solution. A local solution is low maintenance and easy to set up while being more secure and cheaper. Cloud providers such as Git Hub or One Drive provide far more space and allow for slides to easily be shared. Both should be used as they each have distinct benefits and costs, which depending on a user's

need will decide on the preferred approach. There can also be hybrid approaches with both cloud and local storage locations in use.

The same logic applies to point 2 as point point 1 above with its cloud providers being services like AWS. The technologies picked to build and interact with the databases will need to be done with care as there are many considerations such as being lightweight, which can cause unnecessary cost in cloud environments or bloat a user's machine.

For the fulfilment of point point 3 is for ease of use of the tool as it will need to be simple to interact with and extract the necessary slides. Also due to the visual nature of slides they should be displayed as they will be used, and not in their LaTeX format. This will allow the searches for slides to be quickly evaluated and checked otherwise, a simple iteration over files with something like Notepad++ could provide similar functionality.

The purpose of point 4, is if slides are going to be stored in cloud locations or a local system the process of exporting presentations from input needs the retrieval of slides to be seamless.

Requirement point 5 aims to further ease the adding of new presentations to the collection, by removing the need to handwrite the metadata for all slides. Instead, it should be generated automatically as the slides are uploaded. The choice to use ML is to allow more precise metadata generation than general rules would allow.

The final requirement point 6 is to automate the formatting of each given slide to a consistent style. When many slides by different authors are generated there will be style differences which would be time-consuming to standardise even to such an extent that it could be better to build the presentation from scratch. Instead, the format of the slides should all be consistent and be exportable in conformance to any given template.

3.1.1 Low-Level requirements

From the high-level requirements, a set of precise low-level requirements can be made, these are distinct goals for the proposed tool to achieve. The requirements can be listed with three different levels (must, should, could) to their criticalness of a successful project. These requirements can be referred to at the end of the project to judge whether it has been successful. For the project to be considered successful all 'must' items need to be completed, along with most 'should' items. Items with 'could' can be considered to be quality of life upgrades but are not necessary to complete the core workflows.

1. Large storage area
    (a) Must hold large volumes of data

   (b) Must store many files (individual slides of the presentations)
   (c) Must allow single file access/download
   (d) Must not be prone to data loss
   (e) Must limit access with authentication
   (f) Must retrieve slides with appropriate methods
   (g) Should provide easy access
   (h) Could track history of files edits
   (i) Could allow collaborations between users
   (j) Could integrate with existing solutions
   (k) Could selectively publish slides for wider sharing
2. Database to hold metadata
   (a) Must not be prone to data loss
   (b) Must limit access with authentication
   (c) Must quickly search large volumes of data
   (d) Must search a database with appropriate methods
   (e) Should provide easy access
3. Central code and interface
   (a) Must upload new slides
   (b) Must allow database queries
   (c) Must generate slide preview
   (d) Must export newly generated presentations
   (e) Must be able to assign metadata to slides
   (f) Should ensure authors are properly accredited
   (g) Should format slides when exported
   (h) Could provide GUI for all functionality
4. Metadata generator
   (a) Must extract details of slides that can be searched for
   (b) Must accurately summarise slides
   (c) Should require minimal effort on user's behalf
   (d) Could detect updates on slides

5. Non-functional
   (a) Must fail gracefully
   (b) Should give an indication of progress (not hang while working)
   (c) Should be intuitive to use interface
   (d) Should scale to an unlimited amount of slides
   (e) Could allow for integration with other tools
   scripting

3.2 Architecture of Solution

This project has many requirements that can be fulfilled by existing and readily available technologies. These should be leveraged as they complete their tasks to a much higher standard than could be achieved in this project's scope, which will allow for the effort to be focused on the end requirements and produce a much higher quality end product. Whenever external work could be incorporated into the project, the possible choices had to be evaluated to make sure the chosen implementation was best suited for what was needed. The evaluation of alternative implementations were based on inspections of their specifications and reviews by experts.

3.2.1 High-level overview

To fulfil the requirements and allow for a clear approach to be devised a high-level component diagram was made.

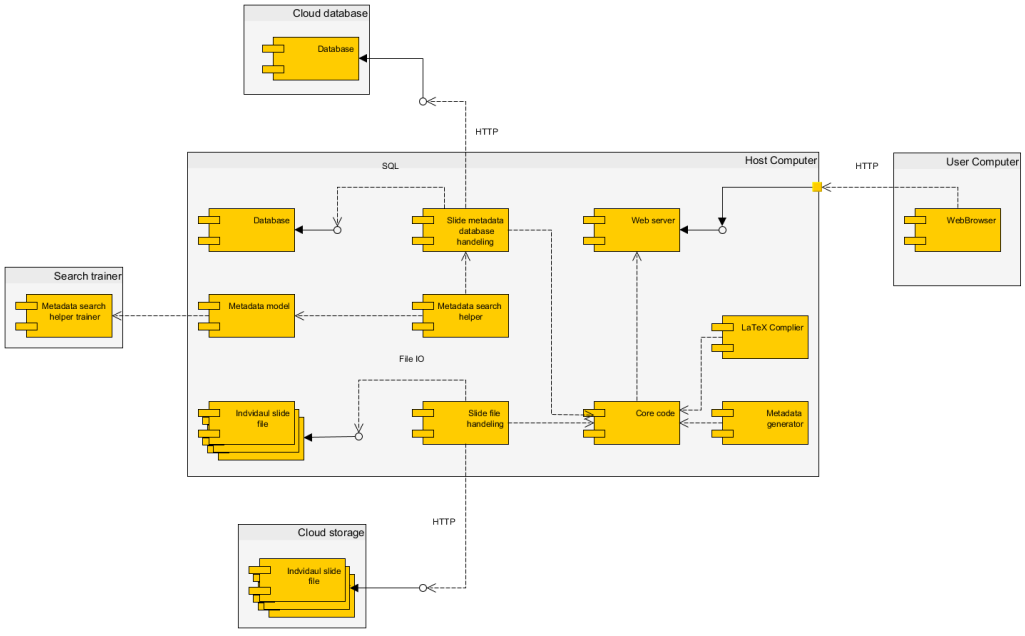

FIGURE 3.1: fig:High-Level Component Diagram

To fulfil these requirements a tool must be written, to plan this, the high-level diagram fig. 3.1 has been created. In the following, the individual components are described in more detail.

3.2.2 Core code

First, there is the core code; this is the central interface that connects the whole program and acts as a common interface. The single communication route helps to simplify communications through the code to one single path and stops multiple cross dependencies. The responsibilities also include maintaining the integrity of the internal information, and any new input should be checked to stop errors.

3.2.3 Interface

The GUI for this project can be assumed to be simple, as using LaTeX suggests a certain level of competence from the user. However, the GUI should still be an aid to the user and not so poorly designed to be a hindrance. The GUI will be accessible to a range of operating systems and should promote collaboration between different users; therefore it will be made on a web page. The web page will allow multiple users to access the same set of resources to promote collaboration in small teams. The tool also becomes more accessible to many more types of system which could not support the tool easily, such as Linux PCs or tablets. The two most popular web frameworks are Flask and Django. There are many articles online discussing the pros and cons of both, such as the one referenced [24], using these have allowed an informed choice to be taken. Therefore for its more lightweight implementation Flask will be used, as the needs for the GUI are simple and should consume the least amount of resources possible.

3.2.4 Data requirements

There are many choices in how the data will stored in the tool. Generally, there is a choice in whether to use local or cloud solutions, both of which have advantages and disadvantages. A local solution is much simpler to implement and is cheaper. However, this limits the total size of the instance. While the cloud is practically unlimited in size only limited by what a user is willing to pay, it also provides the ability to have multiple instances using the same set of files. However, there is now the added complexity of using a service's API which can stall development as it does not allow for direct access. This also requires users to have a cloud solution to be able to use in this purpose and is also less secure than keeping everything internal. The two approaches are not mutually exclusive and would support each other very well. An initial implementation can be created with a local approach which can then be expanded into cloud offerings.

Database

There is a need to be able to order many attributes of the slides into an easily searchable format, which is perfect for a database. With the arguments of cloud against local in mind, any cloud implementation will provide a fixed solution. Therefore any that can be used should be supported or available for inclusion. More criteria effect the local solution, the most pressing concerns are the storage consumption and ease of use. Therefore any database to use locally must have a comprehensive Python library to facilitate its use, and should only create a simple database with no extras to minimise memory usage. Therefore sqlLite [25] has been chosen, for its ability to provide a lightweight database. The other advantage is it offers a direct connection to a database removing the overhead (in resources and code) for a client-server relation.

The structure for the databases needs to be planned as well. The databases will be implemented in two phases, first will be bare necessities the second providing much more functionality. The basic requirements will have the databases using a single table with four attributes, the slide identifier, metadata list, storage location and storage type. This basic set is all that is needed to allow the critical workflow.

Once the core of the project has been made the second phase can commence. The first extended functionality is to track authorship of slides, and this would entail addition tables. The first table would be for author's details, the second would hold entries for every edit of slides, and the elements would be a primary key for a slide and author with a timestamp. The additional information will allow a timeline of edits to a particular slide.

Storage mediums

Large amounts of data can be collected especially when including multimedia such as pictures and videos, therefore how the data is stored is important.

There is a challenge in how to store the slides in a way which makes them easily retrievable.

They could be stored as their presentations as given, however this causes problems in how to identify individual slides which require scanning files extensively for identifiers which is slow. Loading large files can then use massive amounts of memory, which is wasteful to collect the small wanted sections.

Then they could be put into a database, but this will cause problems with the multimedia like images not being able to be included. Therefore the slides are to be saved in their own files that way to store multimedia items, which are packaged separately in LaTeX, can use the same file name with an appended letter.

3.3 Workflows

There are three critical workflows with their respective sub-flows which can define the core functionality of the project, which is the addition of new databases and storage locations, saving slides and retrieving slides. These are defined below with a short description and UML diagram.

3.3.1 Adding resources

The first step needed is to notify the tool of an existing storage location and database or an area where new ones can be created. The following flowcharts highlight the steps needed to initialise a database.

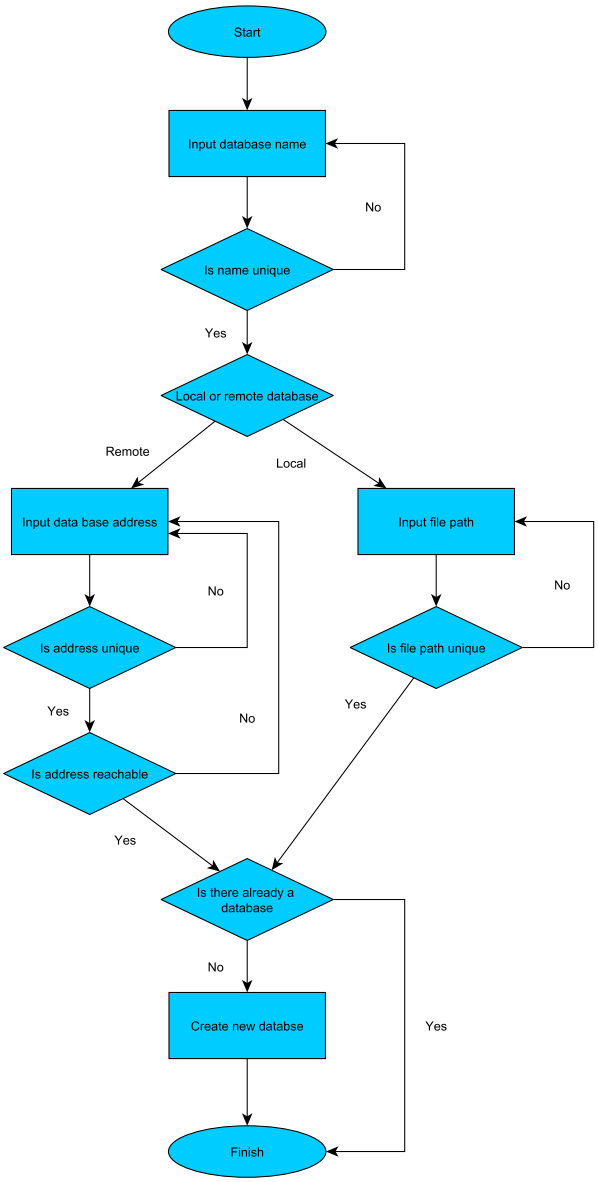

FIGURE 3.2: fig:Process to Add New Database

This diagram fig. 3.2 has been made to plan the process to add a new database to the tool. The key is in using the correct APIs for interfacing with the remote databases.

There is also the critical stage of checking if there is already an existing database, as writing a new blank database would orphan large amounts of slides. Instead, a simple select all of the expected tables with a limit of one will not return an error if the table already exists. In that case, there is no more work to be done, if not then the tool can create a new database and initialise all the necessary tables.

The same process is also used to add new storage locations with the test being a save of a blank file called test.

Slide saving

The next process is to save slides.

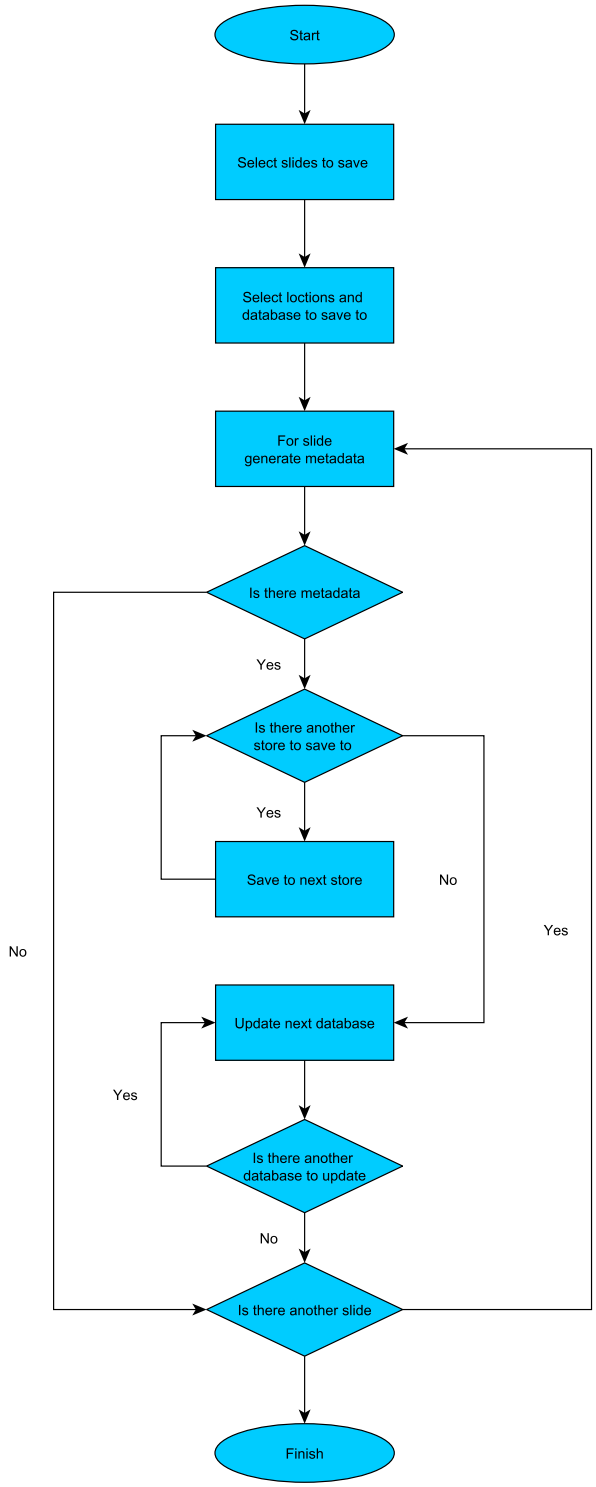

FIGURE 3.3: fig:Process for Slide Saving

The diagram fig. 3.3 shows the processes for which slides will be saved, it will be using a simple nested loop structure to save each slide for each database for each storage location. The slides all get assigned metadata for use in their retrieval, and if they do not have any then they cannot be recovered so they are not saved. Then the individual slide is saved to each selected storage location with the list

of storage location saved to the database can be updated. The update adds a new location with the slide name, locations and metadata this update happen to all selected databases. This process is repeated for every slide to be saved.

The databases and storage locations are contacted for every slide individually to stop any unexpected errors in one slide from preventing every other being saved. Individual saving does incur a time cost as it is much more efficient to bundle all operations into one call, but this cost should be negligible due to the small time cost of the operations.

3.3.2 Search for slide

Once the system has slides stored it will need to search for them, the process for is shown below.

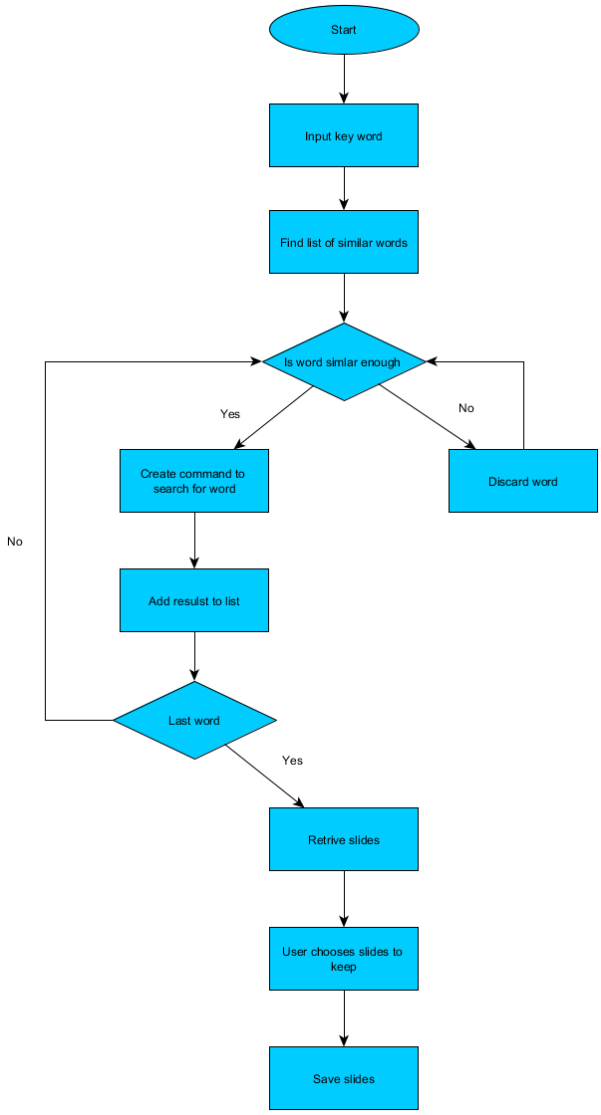

FIGURE 3.4: fig:Search for Slides Process

The diagram fig. 3.4 shows the steps for how slides will be searched and then retrieved. The critical parts of this process are the suggestions for new words and retrieving slides. As discussed in Chapter 2 the suggestions will be provided by a trained graph-based ranking algorithm. The other key point is the interaction between file systems for database and slide retrieval.

An essential process is retrieving the slides, which will be accomplished by checking the type of location of them in the database record. Then the storage type will be checked to allow the correct retrieval methods to be used. Different precautions will be needed for the different types of stores with some, for example, cloud storage must be reachable and all external storages will need to authenticate in a unique manner.

Chapter 4

Solution approach

This chapter will define the processes and tools to be used when developing the proposed tool. These will ensure good quality work can be produced and ensure the integrity of the final product.

4.1 Process introduction

A well-defined process pipeline assures the ability to produce quality software and allow the achievement of higher quality in the future. The process defined thus should be applied for each new piece of functionality that is produced. As there is little experience in producing software at the beginning of the project the aim is to provide a loose collection of processes that give guidelines to quality software, fulfilling the requirements of the Capability Maturity Model (CMM) [26]. CMM classifies how developed the tools and process are and ranks them from one to five. With a Level 1 representing no processes or documentation on past processes and level 5 representing rigorous tools, processes and documentation for the whole development cycle, with constant data-driven review looking for optimisation. During the project, the processes will be developed with reviews after each major release to refine what worked and discard what did not. The final aim is to develop a set of strict processes that guarantee software quality, fulfilling the requirements of CMM of level 4 at a minimum.

4.2 Development cycle

As new functionality is introduced it is completed in a controlled manner to make sure the new code is properly formed and fit for use. Each unit of work is started by describing the inputs and desired outputs. These requirements can then be used to build the specification for each functions. Expected functions can then be derived and assigned to the program. New functions have been following the Unix methodology [27] of producing small rugged code blocks. Small functions means testing individual functions is easier but increases the dependency complexity. For each function a careful was undertaken to find existing code that

already produces the desired functionality to stop wasted effort due to refactoring code.

For each new function, tests define the definition of done (DOD) are written, which will confirm when the inputs generate the desired outputs. This have been expanded throughout the project as new functionality is added. Due to the creation of several prototypes made to confirm technology selections and correctness of the envisioned architecture at the start of the project, a full test rig was able to be used. Yet to be implemented code was to mock code to allow for top-down testing.

During development, most functions gained unit tests especially when calling on other parts of the code. These were designed to exercise all critical paths with results of success and errors expected. Unit testing also provide a regression test to catch any future changes that degrade functionality or quality. The development continues until the DoD tests confirm the specification has been met. These tests are run with the unit test and a linter during each commit to source control by a continuous integration (CI) system. This can then be released as a new beta version to users which can lead to modification or the creation of an entirely new specification.

Once a new feature is complete from several iterations of the above process a new major version can be published.

4.3 Development cycle processes

4.3.1 Source Control

The use of source control has been critical controlling the software and ensure quality is maintained. Firstly, it represents a secure backup for the code. Then using branches, high-quality code can persist while new functionality introduces instability in their creation.

The source control has been using a branching model with four types of branches, in a hierarchy of master, release, develop and feature. Feature branches have been used to develop new functionality with one feature branch for each piece of functionality. These are then merged into the develop branch once all unit tests are passing and the functionality meets it DOD. Develop is used to collect all new features and functionality, it represents the bleeding edge but has always been kept in a 'ready to release' state. Meaning no new work can be added unless it is directly trying to get the develop branch to pass all its tests, forcing only one specification can be worked on at a time as develop branch tests includes the DoD. Once a new release is ready a release branch is created, this is used to complete preparations, such as minor bug fixes and finalising of metadata and

allow the develop branch to continue. As well as all testing the develop branch has release branch testing will include manual testing and can then be promoted to master once every test passes for major releases or back into develop after beta testing and bug fixes. Master branch is therefore, holding each major version of the project that is released to users. The only exception to this process is hotfixes which are for critical errors in master, of which only a few were used. They are branched from master and merged straight back into master. Testing for different branches must provide timely responses, which is inversely proportioned to the amount of time they will be run.

Name	Purpose	Tests to Merge	Response time to tests
Feature	Develop a new feature, should be the smallest amount of code to provide some additional functionality.	Unit Tests,Plylint	> 5Min
Develop	Used to collect new features	All the above,All DoDs	> 4 hours
Release	Release branches are created to provide testable code for a full suite of testing, once it passes everything it is merged to master	All the above, End user testing	> 1 day
Master	Each version of code released	N/A	N/A

4.3.2 CI

A CI system implemented with Jenkins was created to manage test automation in a sterile environment. The main advantage is freeing time to not focus on tests and provide results which cannot be easily ignored. Webhooks have been created to allow the source control to notify Jenkins of new commits and pull requests so that it can start working. Results are written back to source control, so each commit and each pull request have a tag with its build, enforcing standards by blocking branch promotion until a passing build occurs. The system detects what type of branch is being executed via a naming convention of branches to complete the appropriate tests.

Jenkins has been hosted locally and use dynamical created agents in docker. The agents can be cloned from predefined templates to ensure no contamination from old tests and allows any new dependencies introduced to be detected.

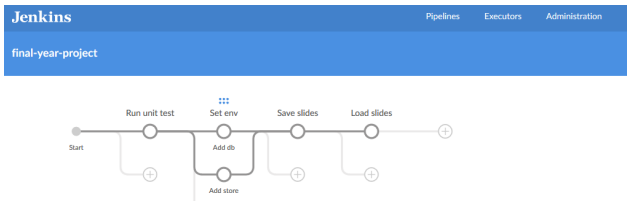

FIGURE 4.1: fig:Example Jenkins Build

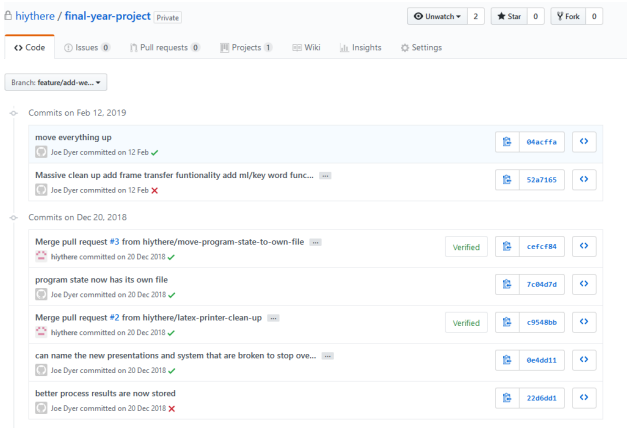

FIGURE 4.2: fig:Example build results in Git Hub

The picture fig. 4.1 shows the typical build on Jenkins in which unit tests are first run then a single run of core functionality. The results of the build can be seen in the ticks and crosses next to each commit in figure fig. 4.2.

Chapter 5

Implementation

In this chapter, the general structure of the tool produced is presented. Also, the interesting points that have arisen in the creation of the tool have been highlighted and discussed.

5.1 Technology used

There were a range of technologies used in the creation of the tool which are as follows.

Python

Python is the primary language and represents nearly all the code written and was chosen due to several reasons. The biggest of which is previous experience in Python which would drastically speed up development times. Python also has a vast array of libraries that can be used to provide common functions.

The biggest disadvantage to using Python is its slow execution speed compared to other languages. However, the computation happening in the project is not too heavy, and produces little noticeable difference to the end user. The other disadvantage is that Python does not naturally produce an executable so is harder to package. However, this is balanced by the source code being able to run on any system with Python installed.

Libraries

Python supports the use of libraries which should be utilised as much as possible as they can be reasonably assumed to be reliable and efficient.

Several libraries that were used that do not come packaged with the default installation were added with the package installer PIP [28]. Many were included to allow for the creation of easy to use interfaces, these were pyInquirer, clint, pyfiglet, termcolor, colorama and flask. Flask was used for the creation of the web interface and the rest are used for improving the text based interface. The library six is used to provide backwards compatibility printing for Python version 2.

Gensim is a library that has been used for it implementation of the TextRank algorithm and its word2vec bag of words models. Gensim was chosen to implement this as it is the most robust library available, so its functions can be used with confidence.

Default libraries have also been used for a variety of purposes. The libraries glob, sys, os, shutil, tempfile, pathlib have all been used for various file handling process. The frame's hash function has been provided by the library hashlib and to run separate threads subprocess has been used.

SQL

To manage the database in the project SQL was chosen. SQL commands are formatted as a string in Python then executed against the necessary database. The decision to use SQL was made as it is easy to use from Python, as it can be treated as a string. SQL also provides a full feature set for the creation and management of database and is industry standard meaning all cloud databases support its use.

5.1.1 Dependencies

Two dependencies were needed to generate the previews of the slides, how they are employed is described later. Both executables will be packaged with the source code so the tool can be run with no set up required. The first of which is pdflatex this is the official tool for compiling LaTeX to a PDF. This was chosen due to its official nature, and its popularity as every surveyed editor also uses it in some form. It also means it is highly likely that it is already available for any user who is editing LaTeX documents.

The second executable is to convert PDF to a PNG which is from Xpdf [29]. Xpdf's pdftopng tool was chosen as the tools are packaged into their own executables so the tool used can be included and it will not take too much space.

5.2 Tool architecture

Large amounts of the code is for keeping track of groups of data, which lends itself to using Object Orientated methods allowing for data to be safely encapsulated. The objects also can be organised to allow inheritance of classes for different database and storage locations. A benefit is the code accessing the databases and storage locations to be blind to their types, greatly simplifying the complexity.

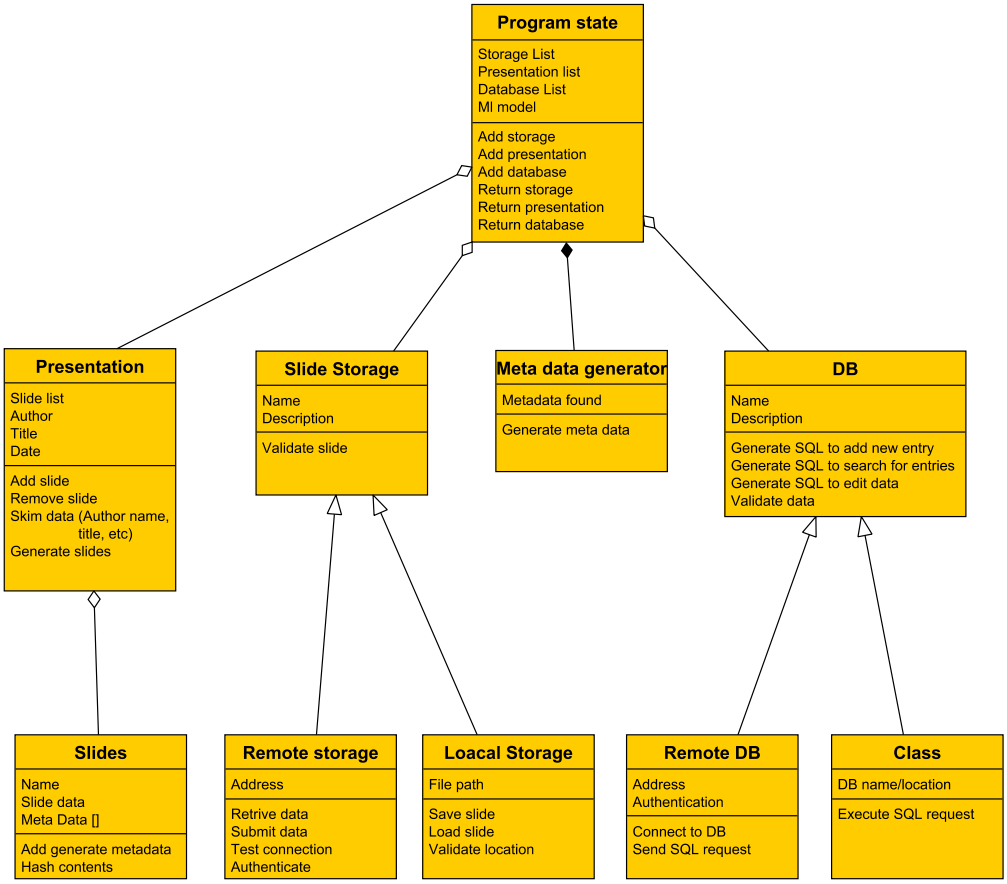

FIGURE 5.1: fig:Class Structure

The class structure diagram fig. 5.1 shows how the classes interact with each other and provides an overview of their key attributes and functions.

5.2.1 Program state

There is a large amount of information that is required throughout the program, critically three dictionaries of all presentation, databases and storage locations. To easily pass these through the code an object was made to house them. This also allows for controls to be implemented to restrict how they are used to decrease error chances. Some of this protection includes preventing overwriting and duplicates of the same object from existing simultaneously. These are critically important to maintain the health when in operation to stop crashes and allow for normal functionality. This also centralises the code which makes tracing bugs easier and neatly organises all the configuration needed, so that it can be written to disk so it can be loaded when the tool is restarted.

5.2.2 Presentation class / Slide class

There is a list of presentations in the program state that will allow the collection of slide objects. The class is used mainly for its ability to organise slides into a

structure in which they will be used, therefore is the logical place to handle the displaying of slides. The process for converting slides from their raw LaTeX to the graphical format is also handled here and discussed later in this chapter.

The slide class holds the contents of a slide; as it is created it will generate its metadata. This is a simple process of calling the TextRank algorithm on its contents. Some complexity comes in the form of the LaTeX keywords polluting the results. This is handled by a simple generator function which will remove any word starting with a \as some in the snippet listing 5.1.

LISTING 5.1: lst:Snippet to remove key words

```
1 resultwords = [x for x in querywords if not x.startswith("\\")]
```

5.2.3 Database class

This class represents one instance of a database used in the program. The key functions carried out by the class is checking if there is already a configured database at the specified location. Therefore during creation, the database will try to query the expected table, if it succeeds then everything is ready and if not a new table can be created.

The class also handles the formation of the queries sent to the database and uses its method interaction to execute them. This is critical for the use of different types of databases as each database type uses separate APIs to interact with and have variations in the queries they use. Another advantage is it concentrates the logic into a few functions to allow them to be produced to a high quality and reduce the chance of errors.

5.2.4 Slide store class

The slide store follows the same logic as the database class except using commands specific for file IO. A difference is in how the testing of the location is undertaken for remote instances, which involves saving a file called test and ensuring a positive response.

To ensure that all file names are unique even when multiple tools are using the repositories a system had to be devised. A possible solution would be to index files and count upwards, which is simple to implement but would require resource locking otherwise it could lead to slides being overwritten in simultaneous writes. A locking mechanism is not supported in many cloud services and could lead to lengthy delays if many users are working at once.
Instead, the contents of a slide were hashed using the sha224 algorithm.

LISTING 5.2: lst:Snippet to hash slides contents

```
1 hashlib.sha224(self.frameText.encode('utf-8')).hexdigest()
```

The snippet shown in listing 5.2 was used as there is little computation power needed and is statistically improbable for files to share the same name. This particular hash was chosen due to its reliable library implementation and the cryptographically secure algorithm. The smallest hash possible as calculated with the formulas from the book 'Benes and Butterfly schemes revisited' [30] would have been 64 bits, with only a collision possibility of 0.1% with 1.9×10^8 slides. However, a 224 bit hash will future proof indefinitely, and its output string is not too long which would be a problem for a hash with even more bits.

5.3 Machine Learning

The existing library Gensim [31] was used, in particular, its models and keywords modules. Gensim was chosen for its robustness and it is using a highly refined algorithm Word2Vec [32]. The most-time consuming process of implementing the system was in the training.

5.3.1 Training

The training for the model to use will have the largest effect on the usefulness of the words generated, as such it is important to do it right. Key to the results would be the breadth and depth of the words that the training is exposed to. Therefore the first model in use was trained on the data given with the tutorial, which was compiled from hotel reviews. Training provided a good understanding of common English words but lacked knowledge of niche topics. To rectify this and with the knowledge that likely users would be of a computer science background, a corpus of text was collected for an improved vocabulary. The text used included hand picked online articles, Wikipedia pages and papers. When used individually these produced poor results but also had little effect on the results when used in conjunction with the hotel data.

Instead, it was decided to use every Wikipedia article to provide a corpus to train on. The collection was a colossal file of 65GB in size from the latest Wikimedia article archive. However, there were problems in training on this data set as the resources available could not cope and failed at around five hours in with a memory error. As this is a popular subject there are many models available which have been trained on a range of sources, the most suitable of which is the results from the book 'Advances in Pre-Training Distributed Word Representations' [33]. This model was trained on Wikipedia articles and then reduced to give the million most important words giving the suggestions an extensive vocabulary.

5.4 Displaying LaTeX

To show a p,resentation it should be in its graphical format for better ease of use. The first part is creating a single string of LaTeX by retrieving all the slides to be displayed and surrounding them by the necessary tags. This can then be given to the global LaTeX library as seen in the snippet listing 5.3.

LISTING 5.3: lst:Snippet to convert create a single LaTeX string

```
def convert_to_png(self):
    combinedFrameText = ''
    for Frame in self.frameList:
        frameText = Frame.give_text()
        if '\frame{\titlepage}' in frameText:
            self.remove_frame(Frame)
            continue
        textToAdd = Frame.give_text()
        combinedFrameText += textToAdd
    LaTeXPackUnpacker.convert_to_png(combinedFrameText, self.Name)
```

To enable the LaTeX presentations to be viewed easily they then need to be compiled before being presented as can be seen in listing 5.4. To do this first, the plain text needs to be compiled which is done with the tool PDFLaTeX. The tool generates a PDF in a specially generated folder using temp file, to ensure there is no interaction with the rest of the system and guarantees a file path.

With the presentation as a PDF it has a visual representation but trying to display a PDF on a web page is complicated. Instead, a much simpler solution would be to transform the PDF to a PNG format. This is a common operation so is supported by a multitude of tools to give the end results of each slide being represented by its own picture, which can then be picked up by the web page and displayed.

LISTING 5.4: lst:Snippet to convert LaTeX to a PNG

```
def convert_to_png(latexText, presName = None):
    previewText = latexText
    previewText = (r'\documentclass{beamer}\begin{document}'
                   + latexText +
                   r'\end{document}')

    try:
        temp_dir = tempfile.mkdtemp(dir=os.path.dirname(os.path.realpath(
            __file__)))
        tempFileName = os.path.join(temp_dir, 'temp.tex')
        with open(tempFileName,'w') as tempFile:
            tempFile.write(previewText)

        #pdflatex like unix paths better
        tempFileName = (pathlib.PureWindowsPath(tempFileName))
        tempUnixName = tempFileName.as_posix()

```

```
17          command = ((r'''pdflatex␣-interaction=nonstopmode␣-output-directory␣
                {1}␣"{0}"''').format(tempUnixName, temp_dir))
18          #print (command)
19          os.system(command)
20
21          tempFileName = (tempFileName.with_suffix('.pdf'))
22          tempPDFName = str(tempFileName)
23
24          if presName is None:
25              command = r'''pdftopng␣{0}␣static/slides'''.format(tempPDFName)
26          else:
27              presNameNoSpace = presName.replace("␣", "-")
28              command = r'''pdftopng␣{0}␣static/{1}-slides'''.format(
                    tempPDFName, presNameNoSpace)
29          os.system(command)
30          print(command)
31          print(tempPDFName)
32
33      finally:
34          shutil.rmtree(temp_dir)
```

5.4.1 Displaying slides on command line interface

To display slides initially and for the text based interface a different approach was used. Instead of using lots of PNG files a single PDF containing all the presentations was used. This PDF now needs to be displayed, but there is no certainty to which PDF viewers are available, therefore the webbrowser library is used. This library allows for the PDF to be opened by the user's default web browser which is almost guaranteed to be available or on a windows system opened by the default PDF application. To allow identification of slides in the displayed presentation the LaTeX was modified to add a water mark with the slide's index.

5.5 Interface

5.5.1 Text based

To provide a intuitive interface the library PyInquirer was used. This was vital for multiple choice questions. Using PyInquirer is an excellent way to build complex text base interfaces, by using a list of dictionaries for each question. The answers are then neatly packaged into a single variable making the following code simpler. The implementation uses the library in a single line function call, which is done to reduce the number of third party calls throughout the code base which the example listing 5.5 shows.

LISTING 5.5: lst:Snippet for command line interface main menu

```
1 options = { 'Import␣from␣disk': load_new_pres,
2         'Add␣slide␣to␣presentaion':add\_slide_to_pres,
3         'save␣to␣storage':save_slides,
```

```
        'Search for slides':search_slides,
        'do db stuff':do_db_stuff,
        'show current config':print_config,
        'copy frames':copy_frames,
        'transfer frames': transfer_frames,
        'DEBUG:print all slides':print_all_slides,
        'DEBUG:GUI print all slides': print_to_gui,
        'DEBUG:List all presentations':list_pres,
        'quit':Exit
{
questions = [
        {
            'type': 'list',
            'name': 'Operation',
            'message': 'What do you want to do',
            'choices': options.keys()
        {
    ]
answer = ask_questions(questions)
```

5.6 Safe file handling

As can be seen in some of the earlier snippets, some external tools required a file to operate on this was handled by the code shown in listing 5.6. Writing files requires some precautions to ensure that there are no interactions between it and other files and provide a method to clean the addition content generated by the tool, such as logs. It was much simpler to use a single temporary file which will be automatically cleaned once closed, however it must be closed before the external tools could work on it. So a temporary directory was made which could then be worked in then deleted once the work was completed, which would clean all the unwanted files as well.

LISTING 5.6: lst:Snippet for creating a temporary work area

```
        temp_dir = tempfile.mkdtemp(dir=os.path.dirname(os.path.realpath(
            __file__)))
        tempFileName = os.path.join(temp_dir, 'temp.tex')
        with open(tempFileName,'w') as tempFile:
            tempFile.write(previewText)
```

There are several sections where a permanent directory is needed for operations, the most prominent being the directory which the web interface's slide previews are displayed from. Anytime slides are to be shown the directory static is wiped clean or created if it is not already existing by the snippet listing 5.7.

LISTING 5.7: lst:Snippet for cleaning permanent directories

```
if os.path.isdir('static'):
    for the_file in os.listdir('static'):
        file_path = os.path.join('static', the_file)
```

```
4         try:
5             if os.path.isfile(file_path):
6                 os.unlink(file_path)
7             elif os.path.isdir(file_path): shutil.rmtree(file_path)
8         except Exception as e:
9             print(e)
10 else:
11     os.mkdir('static')
```

Chapter 6

Verification

In this chapter, the solution is tested to verify that it can complete the required tasks and assess how well those tasks can be achieved.

6.1 Verification methodology

The tool generated has been made to classify small blocks of text and retrieve them at a later date, therefore any test needs a large sample to work on to produce statistically sound results. Unfortunately there are no large repositories of LaTeX presentations available to use, so a system to generate large amounts of data will be needed. Then a smaller test can be run using hand translated presentation of a few similar topics to verify the extensive data set tests.

6.2 Automated approach

To generate large amounts of data needed to be statistically sound a test was devised to generate slides and try and retrieve them programmatically. The source of the slides came from randomly selected Wikipedia articles which, if over at least 2000 words, would then be broken down into several slides. A Wikipedia article is chosen for several reasons; firstly there is a central theme and topic to each article which means the slides can be assumed to be linked. Secondly there is a broad range of topics to provide a range of tests for word vocabulary, and finally, there is a lot of data that can easily be accessed from Wikimedia. The generated presentation would then be uploaded to the tool where its slides are extracted and given metadata, the slides are then archived to local storage location and database. The metadata for each slide is then taken to be used in the tool's search function, and the amount of slides from the same presentation retrieved is noted. The process is repeated for all the slides in the presentation and once all the slides are used a new presentation can be created to repeat the process. For any particular configuration, a couple of hundred presentations would be used to provide an average recall rate.

TeX, stylized within the system as TeX, is a typesetting system (or "formatting system") which was designed and mostly written by Donald Knuth and released in 1978. TeX is a popular means of typesetting complex mathematical formulae; it has been noted as one of the most sophisticated digital typographical systems.

FIGURE 6.1: fig:Example LaTeX slide

For example, the demo slide fig. 6.1 produced from the first paragraph of the Tex Wikipedia article generates the metadata 'tex', 'complex' and 'mathematical' all of which would then be used in searches to try and retrieve the rest of the slides produced from the article.

As well as an indication of the usefulness of the searching and metadata generation this stress tests the tool with the storage of many slides. Both these results are discussed in the following sections.

6.2.1 Verification of retrieval with a script

The script will give each generated slide one to five sentences, and there is the assumption that most should share some metadata. Therefore, the expectation is that most slides will share some metadata; the amount they share can then be used to gauge the effectiveness of the tool.

The initial metadata generation was completed by a word count, this was used with the implemented assisted search. While highly limited the word count gives a good baseline for which any improvements can be then judged as can be seen in the table 6.1.

Model	Corpus source	Recall percent average
No model		4%
WordVec Tutorial	Hotel reviews	5%
Hand made	Scientific articles	4%
Wikipedia	All Wikipedia articles	7%

TABLE 6.1: Slide recall average with word count generator

As can be seen very little of the slides have been recalled. Instead, the TextRank algorithm was used to produce the metadata and a large increase in the number of slides recalled can be seen.

Source	Corpus source	Recall percent average
WordVec Tutorial	Hotel reviews	25%
Hand made	Scientific articles	10%
Hand made and Tutorial	Scientific articles	26%
Wikipedia	All Wikipedia articles	40%

TABLE 6.2: Slide recall average with TextRank generator

The table 6.2 Slide recall average with TextRank generator suggests a consistent set of metadata can be reliably produced for slides, and a user can search for a topic and get a consistent set of results.

6.2.2 Slide storage limit

As the script was allowed to run to create many thousands of slides, this provided a good stress test for the tool. These were allowed to collect in the local archives for this purpose and not cleaned away. There were no noticeable complications when the tool was handling thousands of slides.

There was also a concern that files could be lost or overwritten, to check for this two separate tests were devised. For the first test, the tool was hacked to load random slides from storage and then recalculate metadata to verify it matched the values stored in the database to verify the integrity of the store. The second test went the other way and picked random entries in the database and found the slide it referenced then retrieved it and recalculated its metadata to compare to what it held. The script also counted the number of slides it created so those could be compared to the number of files and the database entries. For all the tests no data appeared to be lost.

6.2.3 Limitations of testing approach

There is a difference between a slide generated in the script with a clear sentence and a slide which might only have several bullet points or no text. These tests are representing the best case for the metadata generation due to the fact slides are made of complete descriptive sentences, therefore any real-world test would produce worse results.

6.3 Translated slides

To gain a significant number of slides needed for testing with real presentation LinkedIn's SlideShare [8] was used. These were then manually translated from their format to a LaTeX presentation, so they could be used by the tool. The metadata generated for the slides would be manually inspected and judged for correctness. The manual inspection showed TextRank was able to accurately extract relevant metadata even when slides contained single word bullet points, showing its suitability for the task.

Then unknowledgeable users were tasked with creating presentations on the topics archived, which proved to be successful as most could generate a fairly complete presentation from their own searches.

6.4 Interface tests

The user test in the above section also tested the interface. The primary menu can be seen in fig. 6.2, this was shown as is and users tasked with archiving then retrieving a presentation. The initial completion rate was poor with only 20% of users competently using the interface, however after simple documentation was provided this rate increased to around 90%.

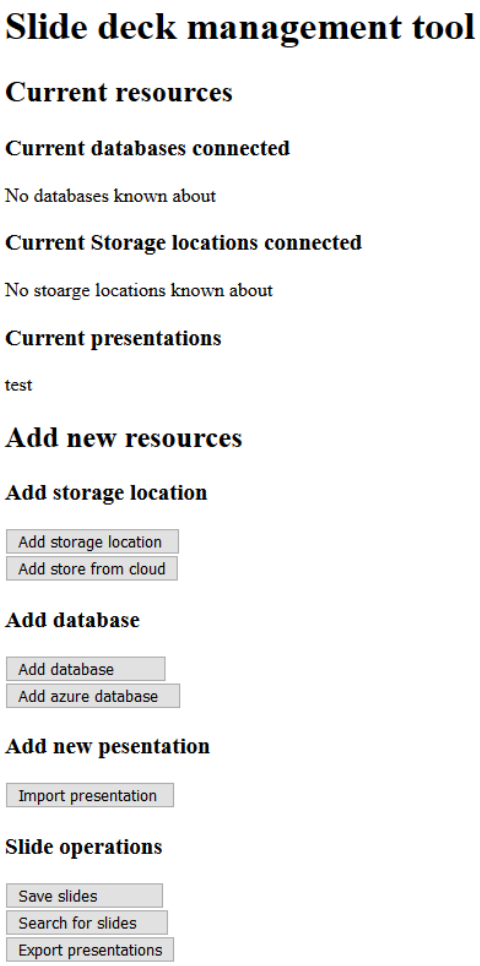

FIGURE 6.2: fig:Interface Primary Menu

Of note, was the poor response to the interface to add new cloud resources, an example of which can be seen in fig. 6.3. The required inputs are not immediately obvious and specific.

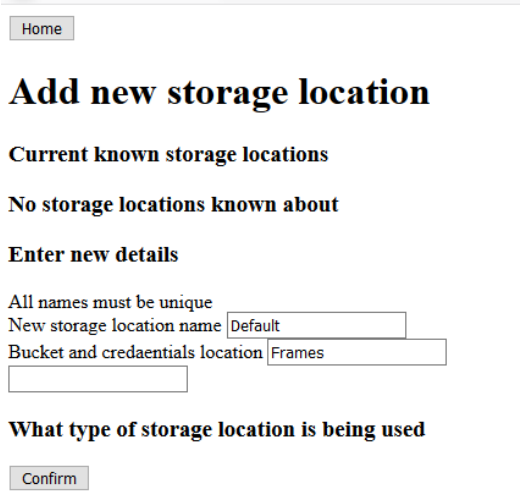

FIGURE 6.3: fig:Interface to Add New Google Store

From these tests the interface provided all the functionality needed to operate the tool reliably, however is not intuitive and requires extensive documentation to explain its operation.

6.5 Requirements evaluation

At the start of this report the primary objective was stated to be:
This report aims to investigate how to increase productivity in reusing existing presentation material, and produce a prototype candidate to implement the ideas.

As proven in this chapter, this objective has been achieved. There was a distinct lack of archiving tools for LaTeX presentations found and a viable prototype has been developed to alleviate the problem.

The design chapter of this thesis laid out requirements for a prototype which needed to be met for it to be considered successful which are now to be reviewed.

Requirement	Criteria completed	Description
Large storage area		
Must hold large volumes of data	Met	The tool can use any file system from the machine it is running on and can use remote resources to store an effectively unlimited number of slides
Must store many files (individual slides of the presentations)	Met	There is no limit to the number of files used in the system so there is no limit on slides, file names are also based on file hashes to stop clashes in file names.
Must allow single file access/download	Met	Each slide is individually retrieved from its source
Must not be prone to data loss	Met	File system and storage approach mean that files will not be lost by the tool. The file storages available for the tool to use has a very high reliability
Must limit access with authentication	Met	The local files are accessed as the user running the tool so share authentication and all global resources use their preferred authentication method
Must retrieve slides with appropriate methods	Met	The object-oriented approach allows all different file structures to be interacted with via the correct APIs

Should provide easy access	Met	The files are easy to access from the tool and all are available
Could track history of files edits	Not met	There is no history kept, any edited slides are treated as new slides
Could allow collaborations between users	Met	Many tools can use the same cloud resources to allow for group slide repositories
Could integrate with existing solutions	Not met	The tool does not directly interact with any existing LaTeX editors, but can theoretically use the same storages as them.
Could selectively publish slides for wider sharing	Met	The tool allows for granularity on which storage locations a slide is published to.
Database		
Must not be prone to data loss	Met	File system and storage approach mean that data will not be lost by the tool. The databases available for the tool to use have very high reliability
Must limit access with authentication	Met	The local files are accessed as the user running the tool so share authentication and all global resources use their preferred authentication method
Must search database with appropriate methods	Met	The object-oriented approach allows all different databases to be interacted with the correct APIs and formats an appropriate SQL string to carry out the request
Central code and interface		
Must upload new slides	Met	The interface allows for LaTeX (.tex) files to be uploaded
Must generate slide preview	Met	The raw LaTeX is compiled into a picture to display on the interface
Must export newly generated presentations	Met	The created presentations can be presented as plain text to be copied into an editor of choice
Must be able to assign metadata to slides	Met	On upload, a presentation's slides all receive metadata automatically
Should ensure authors are properly accredited	Not met	Frame classes allow for an author to be assigned, however, there is no method for tracking authorship
Should format slide when exported	Met	The exported presentation provides a complete LaTeX document and each slide is clearly separated
Could provide GUI for all functionality	Met	All functionality is accessible from the interface

Metadata generator		
Must extract detail of slides that can be searched for	Met	The TextRank algorithm generates key metadata points
Must accurately summarise slides	Met	System testing shows that slides can be reliably recalled with relevant topics
Should require minimal effort on user's behalf	Met	The metadata is generated automatically without any user involvement
Could detect updates on slides	Not met	There is no current method to detect a change to slides
Non-functional		
Must fail gracefully	Met	Nearly all errors will not cause the system to crash and checks are in place to guarantee data integrity
Should give an indication of progress	Not met	There are still times in which the tool is working without giving any indication of what is happening
Should be intuitive to use	Met	The interface allows for new users to operate with little instruction
Should scale to an unlimited amount of slides	Met	With cloud solutions there is effectively unlimited amount of slides usable.

TABLE 6.4: Review of Low-Level Requirements

From the table 6.3, we can see that all 'Must' objectives have been completed, and can conclude the prototype has been successful. The completion of most of the 'Could' and 'Should' requirements further reinforces the fact the prototype has met the stated requirements. With the prototype meeting the requirements laid out and the main objective of this project to produce a viable candidate can lead credence to the success of this project.

Chapter 7

Review and conclusion

In this chapter, a final conclusion of this project has been drawn. Any learning experiences that have been gained during the completion of this project are discussed as well as further work that could be completed on the prototype produced.

7.1 Summary

The thesis set out to make managing large slide decks easier and produce more benefits, this has led to the creation of a tool for managing slide decks. The tool can take any LaTeX presentation, separate each slide and pull out essential metadata, archive the information then retrieve slides with the metadata generated. This tool is presented via a web interface which allows for many users to share large repositories of presentations and have the content consistently catalogued via the TextRank algorithm. There is no practical limit to the number of slides that could be managed with the ability to integrate cloud solutions to hold them and provides significant amounts of flexibility. Of particular interest is where the project has also expanded out of the initial scope, most notably is the added functionality from the search help suggestions. This functionality has made the tool much more reliable at retrieving slides relevant to the topic wanted.
These points taken together shows that this thesis has led to the creation of a valid candidate that could be used and which has been proven in the verification chapter.

7.2 Future improvements

Even though the prototype has met the requirements stated these were generated with the idea of an initial prototype and only covered the basics. With further time and resources, more comprehensive features and functionality could be introduced.

The images included in a presentation were not subjected to analysis only their titles and captions, this would be an excellent area to expand. By taking images and analysing them to produce a short description would allow for more detailed metadata to be extracted, which would prove to be particularly useful with slides

which only have pictures. Analysing images could be accomplished with LaTeX's approach of storing multimedia independently from the main document, allowing tools to be easily leveraged upon them. This functionality would also be a unique selling point and help achieve a more complete description of a slide.

To gain further accuracy of a slide's contents, a review of the whole presentation could be taken for metadata generation. A broader approach would allow for a more cohesive content to be generated to tie all the slides in a presentation together and allow for slides with little content to receive more accurate metadata. The metadata could also be expanded for slides with sufficient words and for the presentation as a whole to summarise them with phrases and sentences. More data will give a user much more to search with, thus increasing the likelihood of finding useful content. However, searching would have to become a lot more comprehensive to handle all the new information.

Another improvement is to expand the supported cloud services and introduce helper functions to aid their initialisation. There are many more cloud services that could be supported such as GitHub or Amazon Relational Database, the more options, the better. Then there should also be a guided process for the user to set up the resources and connect them to the tool.

The models used to suggest additional words for a search require a large amount of resources to use, this places a limit on the size of the model that certain machines can use. A limit means that some users can struggle to gain much value from the tool with the difficulty of finding useful slides unassisted. The solution would be to use a service running in the cloud which users could then connect to. A simple server which presents a restful API to the internet would allow for the requirements to be offloaded to another machine. By hosting this in the cloud it would allow for very large models to be used or even multiple models which specialise in specific topics.

The interface currently is functional but very utilitarian, which can make it uninviting to users. To rectify this, more effort should be placed on the appearance of the interface utilising tools like CSS to make it more appealing. The functionality can also be improved by providing auto-completion on entries and improving the way selections are inputted. Currently, tick boxes are employed to select resources and slides, this becomes a chore when many resources are being used.

The process used was effective for the project and has contributed to its delivery. Most of the work has been undertaken in a measured approach utilising the support available. The systems for developing the tool have been excellent as they have allowed enough flexibility when needed but set a precedent for the code to be of reasonable quality. There were numerous reviews throughout the project to assess how well the tool was developing, of note is the external code reviews. Monthly code would be supplied to other developers for comment and review, this helped fix immediate issues and allow for the developing processes to be

more finely tuned to fix repeating deficiencies. It was quickly found that a ticketing system was not needed to track work as there was a very linear development and a high priority was given to any bugs found to instantly fix them.

However, from this experience there are several improvements which could have been made. Firstly, the support provided could have been more utilised and sought out more often as this may have allowed for a more focused approach and improved the investment of time in certain areas. Secondly, the workload while managed, has spiked considerably nearer the completion of the project. If repeated there would be more focus on writing up the thesis earlier, allowing for additional cycles of reviews and improvements to produce a more refined thesis.

Bibliography

[1] François Brischoux and Pierre Legagneux. "Don't Format Manuscripts". In: *Scientist (Philadelphia, Pa.)* 23 (Apr. 2009), pp. 24–24.

[2] A. Mojžišová and J. Pócsová. "Visualisation of mathematical content using LATEX animations". In: (2018), pp. 536–541. DOI: 10.1109/CarpathianCC.2018.8399689.

[3] M. A. G. Silva, E. F. Barbosa, and J. C. Maldonado. "Model-driven development of learning objects". In: (2011), F4E–1–F4E–6. ISSN: 0190-5848. DOI: 10.1109/FIE.2011.6143024.

[4] M. Benaddy and B. El Habil. "A programming language for matrices operations and LATEX code generation". In: (2016), pp. 1–4. DOI: 10.1109/ICEMIS.2016.7745377.

[5] LaTeX-corp. *LaTeX homepage*. 2019. URL: https://www.latex-project.org/.

[6] miktex. *miktex*. 0. URL: https://miktex.org/ (visited on 06/03/2019).

[7] TeX user groups. *texlive*. 0. URL: http://tug.org/texlive/ (visited on 04/03/2019).

[8] LinkedIn. *Slide share*. 2019. URL: https://www.slideshare.net/.

[9] Copac. *Copac Collection Management*. 0. URL: https://ccm.copac.jisc.ac.uk/ (visited on 04/01/2019).

[10] ShareLaTeX. *ShareLaTeX*. 0. URL: https://www.sharelatex.com/ (visited on 06/04/2019).

[11] Overleaf. *Overleaf*. 0. URL: https://www.overleaf.com/ (visited on 09/03/2019).

[12] Computing Machinery. "Computing machinery and intelligence-AM Turing". In: *Mind* 59.236 (1950), p. 433.

[13] BBC. *Computer AI passes Turing test in 'world first*. 2019. URL: https://www.bbc.co.uk/news/technology-27762088.

[14] Robert W Gehl. "Teaching to the Turing Test with Cleverbot". In: *Transformations: The Journal of Inclusive Scholarship and Pedagogy* 24.1-2 (2014), pp. 56–66.

[15] Daniel G Bobrow. "Natural language input for a computer problem solving system". In: (1964).

[16] Terry Winograd. "What does it mean to understand language?" In: *Cognitive science* 4.3 (1980), pp. 209–241.

[17] Joseph Weizenbaum. "ELIZA—a computer program for the study of natural language communication between man and machine". In: *Communications of the ACM* 9.1 (1966), pp. 36–45.

[18] Lawrence Page et al. "The PageRank citation ranking: Bringing order to the web." In: (1999).

[19] Joel C Miller et al. "Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records". In: (2001), pp. 444–445.

[20] Radim Rehurek and Petr Sojka. "Software framework for topic modelling with large corpora". In: (2010).

[21] Anette Hulth. "Improved automatic keyword extraction given more linguistic knowledge". In: (2003), pp. 216–223.

[22] G Salton. "The SMART system". In: *Retrieval Results and Future Plans* (1971).

[23] GuideForDummies. *Guide for dummies slide reuse*. 0. URL: https://www.dummies.com/software/microsoft-office/powerpoint/powerpoint-2019-slide-libraries-and-ways-to-reuse-slides/ (visited on 11/01/2019).

[24] Gareth Dwyer. *Flask vs. Django: Why Flask Might Be Better*. 0. URL: https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v (visited on 11/04/2019).

[25] sqlite. *sqlite*. 0. URL: https://sqlite.org/index.html.

[26] Wikipedia. *Capability Maturity Model*. 0. URL: https://en.wikipedia.org/wiki/Capability_Maturity_Model (visited on 11/03/2019).

[27] Dennis M Ritchie and Ken Thompson. "The UNIX time-sharing system". In: *Bell System Technical Journal* 57.6 (1978), pp. 1905–1929.

[28] The pip developers. *PIP*. 0. URL: https://pypi.org/project/pip/ (visited on 04/02/2019).

[29] xpdf. *xpdfreader*. 0. URL: https://www.xpdfreader.com/ (visited on 06/02/2019).

[30] Jacques Patarin and Audrey Montreuil. *Benes and Butterfly schemes revisited*. Cryptology ePrint Archive, Report 2005/004. https://eprint.iacr.org/2005/004. 2005.

[31] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[32] Tomas Mikolov et al. "Computing numeric representations of words in a high-dimensional space". In: (2015). US Patent 9,037,464.

[33] Tomas Mikolov et al. *Advances in Pre-Training Distributed Word Representations*. 2018.

Appendix A

PID

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**24010054**
Student Name	**Joe Dyer**
Email	**Joseph.dyer@student.reading.ac.uk**
Degree programme (BSc CS/BSc IT)	**BSc CS**
Supervisor Name	**Julian Kunkel**
Supervisor Signature	
Date	

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	420
1.2	**Project Title**
	Beamer Slide Composition
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	Design and produce a system to facilitate the reuse of presentations made in LaTex.

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Joe Dyer BSc CS joseph.dyer@student.reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
	Julian Kunkel juliankunkel@googlemail.com
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project
	N\A

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
	N\A
1.8	**Company Address**
	N\A
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**
	N\A

SECTION 2 – Project Description

2.1 **Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**

Currently large amounts of people are using LaTeX to produce presentations. This is especially true in academia with Latex's ability to show animations and complex artefacts with little end user formatting [1]. LaTeX is also written as a plain text format which makes it very accessible for scripting in both generating the documents and modifying after creation [2] [3].

As presentations in academia are often recycled (for example lecture course are usually heavily based on last year's material) it can be beneficial to keep track of the material already produced. However, a LaTeX beamer presentation is compiled from plain text to a useable format like PDF, this makes looking over a presentation hard as normally they are large text files with its contents hidden in many markup tags. A better solution can be provided for users that allows quick access to old content. Many presentations are given every day for countless topics.

A review of the tools available commercially for editing and creating and editing LaTeX show there are many tools with one of the most complete being the most complete in features being the combined products of Overleaf and ShareLaTeX [4]. Even these products have no way of allowing quick retrieval of content apart from user made file names, which becomes quickly unmanageable.

The new system should be able to hold large amounts of slides and seamlessly upload/download for any user. By a machine learning system will analyse the slide and generate metadata [5][6] for the slide that can be used to search for it, to ensure good consistent metadata for every slide.

This metadata will then be stored in a searchable from such as a SQL database and linked to the slides stored in another location. Users should then be presented with a simple interface to interact to retrieve slides via searches, possible by natural language searches [6].

[1] Andrea Mojžišová and Jana Pócsová, 'Visualisation of mathematical content using LATEX animations', 2 IEEE, 28 June 2018

[2] Marco Aurélio Graciotto Silva and Ellen Francine Barbosa and José Carlos Maldonado, 'Model-driven development of learning objects', IEEE, 02 February 2012

[3] Mohamed Benaddy and Brahim El Habil, 'A programming language for matrices operations and LATEX code generation', IEEE, 17 November 2016

[4] https://www.overleaf.com/for/authors, https://www.overleaf.com/for/edu

[5] Tao Liu, 'Sparse Topic Model for text classification', IEEE, 08 September 2014

[6] Ruslan Posevkin and Igor Bessmertny, 'Translation of natural language queries to structured data sources', IEEE, 30 November 2015
All URLs were checked and current at date of submission.

2.2 **Summarise the project objectives and outputs in about 400 words.**
These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.

	The initial infrastructure should be able to host many presentations with their slides individually accessible in a reliable and secure format, which then can be retrieved from meta data that is stored in a searchable format. Therefore, there will be **1)** Large storage drive and **2)** Database, these should be hosted in the cloud to provided controlled access while allowing collaboration. These should be then be accessed by a **3)** created program that will provide a range of functions. These functions will initially be simple access to the cloud infrastructure but expand to include. **4)** UI previews of slides from storage. This will allow for searches for slides to be quickly evaluated and checked, otherwise a simple search over files with something like notepad++ could provide similar functionality. **5)** Automated file handling (providing packing and unpacking of presentations). To easily build a large collection of presentations it needs to be easy to present. This should be able to take a complete presentation and disassemble it to individual slides with key information (such as authors) for further processing. This will be most of the work to do the reverse to build a presentation from slides stored. **6)** Automated tagging/meta data creation via ML analysis. To further the ease of adding new presentation metadata for slides should be generated automatically. The choice to use ML is to allow more precise tagging than general rules would allow. **7)** Automated bibliotherapy generation. With the possibility to build a presentation with many authors work it is critical to proper accredit where the work comes from. This should be generated and inserted into the presentation to avoid human error. **8)** Formatting to consistent style or template specified template. When many slides by different authors are generated there will be style difference which would be time consuming to standardise, instead it should automatically apply a standard format. User reviews and surreys could expand or shrink the objectives depending on the response.
2.3	**Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.

Large storage area

1. Must hold large volumes of data
2. Must store many files
3. Must provide granularity in slide retrieval
4. Must be resilient
5. Must be secure
6. Should be accessible
7. Could hold history of files
8. Could allow collaborations
9. Could integrate with existing solutions
10. Could selectively publish slides

Database to hold metadata

1. Must be resilient
2. Must be secure
3. Should be accessible

User friendly interface

1. Must upload new slides
2. Must allow database queries
3. Must search data base
4. Must retrieve slides
5. Must generate slide preview
6. Must save and load slides from disk
7. Must be able to tag slides
8. Must ensure authors are properly accredited
9. Should format slide for upload
10. Should generate slide tag
11. Should provide selected frames in easily usable state
12. Could provide full GUI

Metadata generator

1. Must extract detail of slides that can be searched for
2. Must be accurate
3. Must be precise
4. Should require minimal effort on user's behalf
5. Could use slide hash as primary key
6. Could detect updates on slides

2.4 **Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval? (If your project requires a questionnaire/interview for conducting research and/or collecting data, you will need to apply for an ethical approval)**

There is some concern for copyright infringements if there is carelessness in gathering material. As the aim is collect many academic slides they need to properly accredit and may not be able to be widely shared.

There could also be a questionnaire for end users on their desire for the final product and any features they would like.

2.5 **Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.**
e.g. item 1 name, supplier, cost

	No cost
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**
	Not need but the Department would provide the easiest source of a large collection of presentations made with LaTeX.

SECTION 3 – Project Plan

Total number of weeks from 8/10 24

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	Task description	Effort (weeks)	Outputs
1	**Background Research**		
1.1	Background Research in products available	1	Gaps in current market to produce something original
1.2	Background in research in products and user wants	1	Make sure new ideas and products are being created and will be useful
2	**Analysis and design**		
2.1	Create criteria/specs	1	Fully formed criteria and specs
2.2	Design diagrams	0.5	Components needed, their high-level functions and how they communicate with each other
2.3	Select tech (by testing)	0.5	Final list of infrastructures to be used
2.4	Design Database	0.2	Plan of well-formed database that can hold the metadata.
2.5	GUI Design	0.1	Design of final form of GUI provided
2.6	UML for program	1	Plan for program
3	**Develop prototype**		
3.1	Create storage location and database in cloud	1	Initiate area in cloud for metadata and slides
3.2	Create user tool to interact with cloud infrastructure	2	Be able to interact with database and storage reliably. Be able to easily extend
3.3	Metadata training set	2	Set of slides enough to train ML system on
3.4	ML for slide processing	3	Automated metadata tagging
3.5	Advanced file handling	1	Generate and load presentation in a correct format.
3.6	Slide preview GUI	1	Generate a page with complied slides.
3.7	Automated bibliotherapy	2	Any presentation built will have a complete list of authors work used.
3.8	Formatting to consistent	3	A built presentation will be consistent in style
3.9	Form database structure	1	Add tables and rules for metadata data base
4	**Testing, evaluation/validation**		
4.1	Unit testing	2	Individual components have limited faults
4.2	Integration testing	2	Ensure all components interact properly

4.3	User beta test	1	Gather feedback on usability
4.4	Stress testing	1	Find upper limit of system under different types of load.
5	**Assessments**		
5.1	Write-up project report	3	Project Report
5.2	Produce poster	1	Poster
5.3	Produce demonstration and presentation	2	Demonstration and presentation
TOTAL	**Sum of total effort in weeks**	**33.3**	

SECTION 4 – Time Plan for the proposed Project work

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

Project stage		**START DATE: 14/09** **Project Weeks** 0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1.1	**Background Research in products**													
1.2	**Background in research in**		← First specification created											
2	**Analysis and design**													
2.1	**Create criteria/specs**													
2.2	**Design diagrams**													
2.3	**Select tech (by testing)**													
2.4	**Design Database**													
2.5	**GUI Design**													
2.6	**UML for program**			← specification and design finalised										
3	**Develop prototype**													
3.1	**Create storage location and**													
3.2	**Create user tool to interact with**				← Point of basic usability/ first release									
3.3	**Metadata training set**													
3.4	**ML for slide processing**													
3.5	**Advanced file handling**													
3.6	**Slide preview GUI**													

3.7	**Automated bibliotherapy**													
3.8	**Formatting to consistent**													
3.9	**Form database structure**													
4	**Testing, evaluation/validation**													
4.1	**Unit testing**		A continuous integration approach should be taken with latest versions being regally being released and testing should be constantly run through out the project.											
4.2	**Integration testing**													
4.3	**User beta test**													
4.4	**Stress testing**													
5	**Assessments**													
5.1	**Write-up project report**													
5.2	**Produce poster**													
5.3	**Produce demonstration and**													

RISK ASSESSMENT FORM

Assessment Reference No.		**Area or activity assessed:**	
Assessment date			
Persons who may be affected by the activity (i.e. are at risk)			

SECTION 1: Identify Hazards - *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

1.	Fall of person (from work at height)		**6.**	Lighting levels		**11.**	Use of portable tools / equipment		**16.**	Vehicles / driving at work		**21.**	Hazardous fumes, chemicals, dust		**26.**	Occupational stress	
2.	Fall of objects		**7.**	Heating & ventilation		**12.**	Fixed machinery or lifting equipment		**17.**	Outdoor work / extreme weather		**22.**	Hazardous biological agent		**27.**	Violence to staff / verbal assault	
3.	Slips, Trips & Housekeeping		**8.**	Layout , storage, space, obstructions		**13.**	Pressure vessels		**18.**	Fieldtrips / field work		**23.**	Confined space / asphyxiation risk		**28.**	Work with animals	
4.	Manual handling operations		**9.**	Welfare facilities		**14.**	Noise or Vibration		**19.**	Radiation sources		**24.**	Condition of Buildings & glazing		**29.**	Lone working / work out of hours	✓
5.	Display screen equipment	✓	**10.**	Electrical Equipment		**15.**	Fire hazards & flammable material		**20.**	Work with lasers		**25.**	Food preparation		**30.**	Other(s) - specify	

SECTION 2: Risk Controls - *For each hazard identified in Section 1, complete Section 2.*

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible)*
			High	Med	Low	
5	Display screen equipment - Long time spent working at computers	Ensure 5-minute break each hour			✓	Set timers to inform of hour limit
29	Lone working / work out of hours				✓	Ensure a schedule of following ~9-5 work hours, and healthy sleeping habits are maintained
Name of Assessor(s)			**SIGNED**			
Review date						

Appendix B

Social legal and ethical

B.1 Concerns raised

At the start of the project, there were several issues noted and others have been found during the creation of the project in the project initiation document (PID) appendix A. These will be assessed for if they were valid and if so how they were handled.

B.1.1 Display screen equipment - Long time spent working at computers

There was many hours spent on computer in the creation of the tool and writing the documentation like this thesis. As noted in the PID appendix A the solution was to enforce breaks at a set schedule. This was kept so that every 2 hours a 15 minute break would be taken.

B.1.2 Lone working / work out of hours

With little outside structure to how the work was undertaken could lead unhealthy habits from forming. The control prosed at the start of the

Appendix C

Log Book

Project Log Book

28-04-2019

Final review of thesis
Preping thesis for submition

08-04-2019

Main draft of presentation and demonstration craeted and under review
Demo path for tool set up to provided good demo results
Thesis sections verifcation and conclusion statred
Movtivation competly rewritten most existing parts moved to design
Litiature review completed

28-03-2019

Poster final draft checked and subbmited
Presentation demo begining to be drafted
Full user testing of project
Bug finding and fixing

25-03-2019

The program can now be used to search for slides via a web page
Thesis section 1.1 has been rewritten and expanded
First draft for the poster
Thesis sections now included first drafts for Motivation, Literature review, Design, Solution approach and implermentation.

14-02-2019

project mow uses ML to search for and assign keywords to the slides, leaving very little left to add (but a fair bit of polishing). The report is attached in its early state, the introduction and solution approach have been taken from previous work so will need more massaging into shape. It would be good for feedback on the topics chosen for 2.2 and 5.1 are the right approach and I should continue to work on them.

22-12-2018

The program is now 'complete' in its basic implementation and is ready for ML. I have started working on how to use the ML systems from our meeting (http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/). I am a bit unsure on the best data to train on, I was thinking for computer science presentation/articles based on the likely users.

For the next bits of work I will be focusing research for ML use and writing the thesis

15-12-2018

The last two weeks I have:

Fixed bugs in how slides were stored
Reliable compile slides to PDF files
Draft more for my theses

Next two weeks I plan to have the program able to fully run with ML and start a mini code review.

30-11-2018

I have added new functionality to the software to:

Simple metadata/keyword generator

Add slides to a db
Search for slides in db
recover frames from files based on db search

Next week plan to add:

GUI previews of slides
Asset management

I have also been keeping daily logs of the work done to eventually add to the thesis. All the work has been added to git hub if you want a closer look.

Meeting 2018-11-22

- Status:
    - Prototype
    - Read LaTeX files => saves them to disk
    - GitHuB Repository here
    - Template work
- Thesis
    - Abstract => most important piece
        - Problem
        - Teaser; quantify the results
        - Key contributions (in terms of science)?
        - Conclusions in brief

    - Each Chapter should show the “central theme” through the chapter
        - In this chapter, we do XX, Section X..
        - Support for the reader to follow the “central theme”
    - Goals => short & clear. Have one “bigger” sentence that introduces what is your theses.
        - Research ways of organizing presentation slides to optimize reuse and prototyping a candidate.
    - Goals => partially Requirements => go to Design!
    - 2=> Introduction
    - Appendix
        - Which tools to use for what
- Database structure
    - SQLite
    - Presentation link
    - Presention key, slide key, slide # in presentation
    - Discussion:
        - Relation between slides => parent, derived from slide(s)?
- Workflow
    - Action: Joe sends status report every week
    - Add thoughts about alternatives?
    - Take presentation written in LaTex (1 week)?
        - => Extract slides => Keywords / Metadata into DB
            - Must extract all text but not LaTeX keywords
            - https://www.blog.pythonlibrary.org/2018/04/10/extracting-pdf-metadata-and-text-with-python/
            - creating “preview” PNGs from the slides?
        - Result:
            - Tex Code for 1 Slide
            - Assets needed for 1 Slide
            - Preview PNG
            - PDF of the single slide
            - Metadata => author, institution, date?
    - Search from slides based on keywords (1 week without ML)
        - Python tool text based interface
            - => generates HTML
            - With ML support
        - Visualization tool of the slides after navigation
            - View slide previews
            - HTML ^-^
    - Output: Should help to build new presentation (1week to generate this)
        - Respect authorship and Copyright
        - How to include slides into “presentation” that is to be “built”
        - Should be able to add the slides as one “slide” + directories with assets/pictures and how to include
    - Machine learning (1 week for ML)
        - How does it help searching?
            - producing tags / keywords for slides
            - Fuzzy search ! => did you mean shoe?

                - Someone searches “country” => it should find: county, countries, Germany?
            - Takes all Metadata and Text Data on slides
            - Train yourself => load existing models
            - Python
                - http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/
        - Cloud environment (2 weeks)
            - Upload of slides?
            - Generate the download of slides?
            - django => Python framework
- Do we need normalized database? => No I don’t think so.

Email in between

Meeting 2018-10-19

- Status:
    - Design docs done
    - ER for database
    - GitHub repository
        - Action: access to Julian
            - https://github.com/JulianKunkel
- Next steps
    - Implemention prototype
        - SQLite
    - What info exists for final year project thesis template?
        - Action: Joe sends info to Julian / change stuff :-)
        - Julian supports you
    - Action: Start writing something for the thesis (use template)
        - Structure a bit
    - Structure of the thesis => 50 pages ^-^
        - Introduction
            - Motivation
            - Goals
            - Structure of the thesis
        - Background
            - Whatever is relevant as background for people with CS knowledge
            - LaTeX
        - Related work
            - Related to the research / design/ implementation of this thesis
        - Design (high-level)
            - Overview
            - Data model
            - Architecture
            - UML Diagrams
        - Implementation
            - Selected pieces of interesting code/problems...

        - Evaluation
        - Summary & Conclusions
            - Future Work
        - Biography
        - Appendix ...

Meeting 2018-10-05

- PID
    - Even if database
        - import/export ⇔ Git
    - Local version
        - Command line
        - => Probably not have a normal database? SQLlite?
    - global version
        - RESTful APIs
    - 2.3) Metadata generator box
        - Machine learning
    - 2.6) Virtual Machines to demonstrate the stuff…
    - Have 10 References in 2.1 => cover all topics
        - Maybe ML for metadata in libraries?
    - Put Names from tasks into Section 4
        - “First Prototype”
    - Try my templates if you like
        - git pull http://git.hps.vi4io.org/julian.kunkel/reading-templates.git
    - Agile programming
    - Related work:
        - https://de.slideshare.net/lshtm/preparing-data-for-sharing-the-fair-principles
        - https://www.slideshare.net/
        - http://www.authorstream.com/SignIn/Dashboard
        -

Meeting 2018-09-03

- Presentation of slides
- Background
    - Why LaTeX?
        - Textfile => compiles to PDF, HTML, ...
    - Why repository of slides?
        - \input{fileSlide1}
        - \input{fileSlide2}
- Related work
    - Overleaf => Manages a single project => no reuse
    - Other stuff…
    - => What else?
    - LaTeX editors out there MANY !
        - ATOM with auto completion, templates…
    - PanDoC => Markdown

        - https://pandoc.org/demos.html
    - Office 365
        - PowerPoint ?
    - Grammarly?
    - Periodic system of presentations
        - Structure
        - Grammar
        - Presentation style
        - https://xebialabs.com/periodic-table-of-devops-tools/
    - Develop Criteria
        - Collaborative
        - Cloud-enabled
        - Easy-to-use
        - Language
            - Markdown, …
        - Templating?
-
- Goal
    - Agile presentation creation
    - How to reuse material for presentations?

Suggestions

- Use LaTeX
    - LaTeX beamer
    - https://git.hps.vi4io.org/julian.kunkel/reading-templates
- Slides
    - Single liners
    - Footer
- Design
    - Think about alternatives
    - Alternative Workflows?
        - How does DevOps vs. Presentations conceptially match, what is different in the workflow
        - Right now: Build a new presentation
            - copy old stuff into it
            - modify slides
            - => done
        - Process:
            - Selection of content
            - Modification
            - Templates
                - Auto-filled!
            - New content
        - Additional aspects
            - Collaboration
            - Permissions
                - Private slides...
            - Users

        - Granularity of objects we like to re-use
            - Word
            - Bullet
            - Picture
            - Slide(s)
        - What does testing of “slides” mean?
            - Consistency, presentation style, ...
    - “Raw Slides”
        - Cloud => corporate slide creation
        - SQL database
        - Git Repository? Single file per slide?
    - Tools:
        - command-line-tool would be great
        - GUI: HTML page
        - Alternative: Own GUI
        - Alternative: Integration into existing dev tool as plugin (Vis Studio, ATOM editor)
    - Criteria
        - Ease of use
        - Ease of installation
        - Flexibility
- Future Work
    - Machine learning to compile slides for a certain topic ?

12-09-2018

- Collect thesis template for latex to use
- Inital protypes created and focus on crating scripts for creating enviroment
    - Pip etc

19-08-2018

- Creation of git hub repository https://github.com/hiythere/final-year-project
- Examinig past thesises for insperation

Next steps

- Compile: Related work, research on the core questions
    - Think about criteria; your tool(s) do sth. nobody else has done
- Think about the thesis structure
- Email to Julian beginning of October