MapMyNotes

Final Report for CS39440 Major Project

Author: Ryan Gouldsmith (ryg1@aber.ac.uk)
Supervisor: Dr. Hannah Dee (hmd1@aber.ac.uk)

4th March 2016
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in
Computer Science (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name .....

Date .....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name .....

Date .....

Acknowledgements

I am grateful to...

I’d like to thank...

Abstract

Include an abstract for your project. This should be no more than 300 words.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

1.1 Background

Handwriting notes is still considered to be an important aspect of note taking. Smoker et al. [16] conducted a study comparing handwritten text against digital text for memory retention and out of 61 adults 72.1% preferred to take notes using pen and paper, rather than on a computer. Smoker et al. concluded that recollection rates for handwritten text was greater than that of typed text proving that handwritten notes are better for a user's memory retention.

Technology has advanced and people are becoming more connected with distributed services through the cloud as well as tracking things in their life digitally; Google Calendar is an example of this. Therefore, there's a need to ensure that memory retention with handwritten notes is carried forward into the digital age.

1.1.1 Taxonomy of notes

When notes are made they will often be very different from any other note. Some are semi-structured and some are "back of the envelope" kind of notes. When thinking about an application to analyse notes, first there has to be consideration for what a note will consist of. A taxonomy, by definition, is a biological term for a classification of similar sections, showing how things are linked together [20].

Notes can be thought of as a collection of similar classifications, whether this is the pure textual descriptions of a note or whether this is purely pictorial form or a mixture of both. However, the notes are normally split into three distinct categories:

1. Textual descriptions
2. Diagrams
3. Graphs

In Figure 1.1 it shows a taxonomy of the different aspects which may form a part of a note. Textural descriptions form the core content of a note, this is essentially the important aspect that a note-taker is trying to remember and write down. Different note-takers form their notes in different

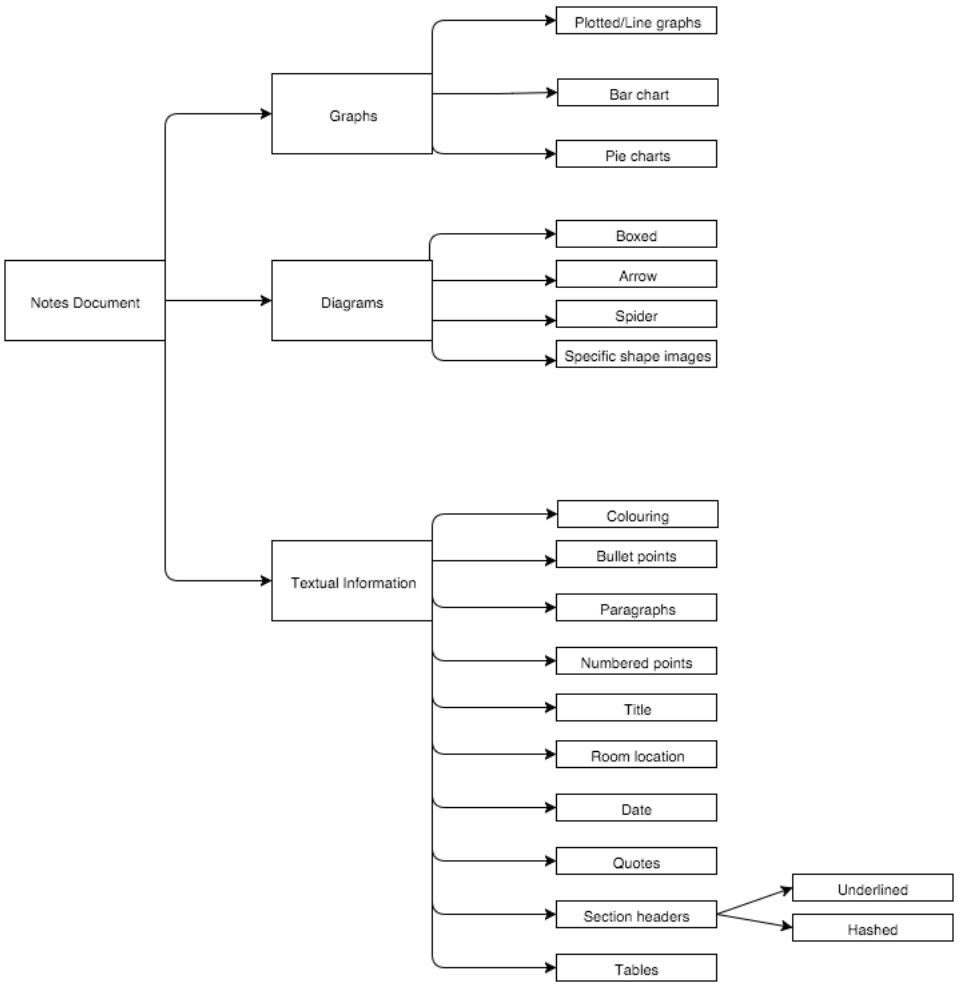

Figure 1.1: A taxonomy showing the structure and classification of different types of notes and what is contained in a note.

ways, for example the headings may be underlined or hashed - if they were adopting a mark-down style approach. These sections help to show that there's a break in the content, and should be sub-sectioned. Textural points that are short, but important, are often characterised by a colon or a bullet point; these are the most common form of concise note building, in the classification.

Coloured text is often used for a variety of reasons: it stands out on the page and improves memory retention of that text [3]. Both congruent and incongruent coloured text helped to increase memory retention of post-graduate learners [12]. With congruent text, Olurinola et al. showed that for 30 students with a 20 word list, a 10.9 mean retention rate was achieved and a 8.1 mean retention rate was achieved for incongruent text. The studies conducted show that coloured items would improve a user's retention rate in a lecture.

Finally, tables help to represent textual content in tabular form - this is often good in notes for comparisons.

Graphs are great visual tools for users to help to convey important textural information easily. Naturally, they have their limitations such as they come in different shapes and sizes, such as a line-graph, pie chart or a bar chart. Coupled with graphs, notes often consist of diagram drawings.

In Figure 1.1, there are different sections and classifications of a diagram: boxed, arrow etc. Each one has its own purpose and arrowed and box can overlap; UML diagrams are a case of this. Spider diagrams are probably the hardest to represent, due to the varying sizes and whether the user draws circles or clouds. Furthermore, specific shape diagrams are conceptually hard to think about as it depends on the domain in which the user is drawing the note. For example, a person in biology may draw a stick person, whereas someone in Computer Science may draw a computer.

Identifying a taxonomy of notes is imperative when considering what to parse from a note as it helps to define a domain of possible classifications. By identifying the classifications it will acknowledge what sections can be parsed by a specific technique. For example, textural information and bullet point lists can be parsed via text-recognition however, for diagram recognition that would involve image manipulation.

1.1.2 Handwriting recognition

Analysing a user's handwriting is a complex process and one which requires lots of research. Handwriting recognition has had successes in the past from Machine learning techniques, such as neural networks [7]. Knerr et al. yields a 10% rejection rate and a 1% error rate, with the use of Neural Networks on the U.S. Postal service data collection, identifying that handwriting recognition is still very much a research oriented to produce a reliable system to extract and identify handwriting, generically.

Another approach is to analyse handwriting via an OCR (optical character recognition) tool. In [14] the open-source tool, Tesseract [18] was used to analyse Roman scripts. The system was trained on handwriting identified from those scripts. Rakshit et al. recorded an 83.5% accuracy on 1133 characters. It is noted in the paper that "over-segmentation" is a problem with the Tesseract engine.

Another issue to overcome when analysing handwriting is disjointed characters; this is where sections of the letter are split off from the main body of the character, for example, the letter i. Rakshit et al. concludes that this is an issue which is ever-present in the Tesseract engine, with around a 53% misclassification on this character alone.

The problem of handwriting recognition has not been solved - but tools such as Tesseract offers support for improvement in this field. Ask Rakshit et al., discusses training had to be conducted with Tesseract to ensure that it can identify handwriting successfully. As a result, every implementation of handwriting recognition needs significant amounts of data of varying quality, so that a system can succeed. However, considering Tesseract's high recongition rate of 83.5% experienced by Rakshit et al., it is a viable option to use for handwriting recognition.

1.1.3 Similar systems

With note-taking on digital devices becoming more widely available, there has become an influx in note-taking and organisational applications available for users. These are predominately WYSIWYG (What you see is what you get) editors - which allow a great deal of flexibility. When evaluating existing systems three were predominantly used and they were:

- OneNote

- EverNote
- Google Keep

1.1.3.1 OneNote

OneNote [9] is a note-taking and organisational application made by Microsoft, offering the functionality to add text, photos, drag and drop photos onto a plain canvas. In recent times, OneNote has developed functionality to analyse a user's handwriting, from say a stylus, and interpret the text they entered [11]. In OneNote you can insert a note into a document and then it would interpret the text from the note.

There is a wide range of product support from mobile based applications to web versions of their software. Office Lens [10] can be used in conjunction with the OneNote to help to take photos and automatically crop the image and then save them to OneNote. This feature is important and should be considered for the *MapMyNotes* application. The process requires the user to sign in with a Microsoft account. When creating notes, OneNote formats collated notes into a series of "notebooks".

One feature which was noted when analysing the system is automatic saving of the note, reducing the need for a user to click save. Additionally, when using OneNote it feels very much like Microsoft Word - with the similar layout that gives most users a similar user experience feel with its intuitive WYSIWYG editor.

1.1.3.2 EverNote

EverNote [4] is a note-taking and organisional application, it is supported as a web application, bespoke desktop application and a mobile application. EverNote is widely used and provides a wide range of functionality a user would need to digitise their notes.

EverNote have released development articles [1] stating that OCR recognition on images is possible. This would allow the user to upload an image outputting a list of potential words for the image found. Like OneNote, the notes are collated into Notebooks, offering a WYSIWG editor, giving the user full control of the content that is entered. When uploading an image to the web version it gives the option to edit the PDF and images, however it seems as though an additional application has to be downloaded, specific to the user's platform, to be able to utilise this functionality.

According to the website, it does do OCR recognition, however whilst using the web application there was no information regarding extracting of text from the application. Additionally, there seemed to be no way to save the note to a calendar item - only the option to send via a link.

1.1.3.3 Google Keep

Google Keep [5] is a note taking application produced by Google with mobile and website support. Google Keep allows a user to attach an image to their note, attempt to extract the text from an image and save this in the body of the note. In addition it allows the user to tag a title and add an associated body.

An important design feature that it does not offer is the support of a WYSIWG editor; a default text box has been preferred, offering a more raw feel to the application. They have the option of a "remind me" feature, which will get synced to their calendar as a reminder - but there's no easy way to add it to a calendar event.

Google Keep seems as though it's more suited for TODO lists and jotting down quick notes, rather than an archiving tool suitable for substantial note taking. Nevertheless, the tagging with labels is a nice feature and the filter by image is a smart tool; this only shows notes with specific images. The simplicity of the user interface and the ease in which text can be extract provides a great reference going forward.

1.1.3.4 Reflection on the systems

These three existing products are widely used by the every day note taker. They have been developed to a high quality and give the user full control of what their notes can consist of. The automatically cropping of an image is an important feature and should be considered when for the application going forward. *MapMyNotes* aims to try and give the user full control of their lecture notes content, so that they can find their notes easily again.

MapMyNotes intends to differ from EverNote's text extraction by providing a one to one comparison of the text, rather than a list of potential words.

After the analysis of the existing products there are certain aspects which would be regarded as necessary: a simple way to view the notes, a way to filter the notes and a simple UI which feels more like an application rather than a website.

1.1.4 Motivation

The author handwrites his notes during lectures and often are discarded in crowded notebooks until they are needed for an assignment or examination.

A calendar event is already stored for every lecture that the author goes to, so it would be useful if there was a way to associate each of the notes taken to that calendar event. This would ensure that all the information is located in one easy place that can be found again, instead of trawling through lots of paper and trying to find the content again. This would aid in reducing the chances of lost notes from paper slipping out of the notebook or pages being damaged due to rain or creases.

1.2 Analysis

As the project was originally proposed by Dr Harry Strange, a meeting was arranged to discuss the initial ideas that he wished the application would follow. It was here that it was highlighted that Dr Harry Strange wants to take a photo of his notes, archive them with specific data, make them searchable and integrate them with existing calendar entries he had for a given date.

1.2.1 Parsing a note

In conjunction with the information gathered a taxonomy of notes was collated, helping to deconstruct what a note consists of. Analysing the taxonomy produced a comprehensive breakdown of what could be parsed as text. After seeing that text formed a main component of a note the key efforts of the application would focus on parsing the text. Diagrams, graphs and images were considered when thinking of what should be extracted from an image, however this was placed as a task for the future. This required a task to investigate how to reliably extract the text from an image.

1.2.2 An OCR tool

Handwriting recognition has been an active research project for a while. There could have been the possibility of creating a bespoke handwriting recognition tool, using machine learning techniques, but that would distract from the actual problem which is this available tool to archive notes.

Therefore an OCR tool would have to be chosen to analyse the text. Choosing a sensible OCR tool with good recognition rates would be important - so a task was created to explore and look at possible solutions.

1.2.3 What to parse from the note

From research conducted into Google Keep it was clear that analysing the text would be a great aspect to include in the application. The real question is what should be parsed from the note? By looking at the overall structure of the application and what it entailed then it was agreed to just parse the note's associated meta-data: the title, lecturer, date and module code. Recalling that Google Keep parses all the text and EverNote gives a list of suggested words, it was decided that a tool would be developed to suggest the meta-data but it does not automatically tag the meta-data.

1.2.4 Structuring of notes

In conjunction with analysing what to parse, a sensible structure would have to be applied to notes used in the application. A task to create and find a good set of rules would have to be collated to ensure that notes could be parsed confidently. This reduced the complexity of incorporating natural language processing in the application, which would be more than can be achieved in the timeframe.

1.2.5 OCR for the authors handwriting

After research into OCR technologies, such as Tesseract [18], it was established that analysing handwriting is a complicated process. Instead of trying to train it on a lot of dummy data, it would be trained to recognise the author's handwriting. A task was created to train the user's handwriting data and this would run throughout the duration of the project.

1.2.6 What platform is most suitable

During the meeting with Dr Harry Strange one of the core features that was needed was for the application to be accessible regardless of where the user is. After the research was conducted all the aforementioned software tools have web application version of their application. A mobile application was considered but only one version of the application would be made, either Android or iPhone, therefore preventing other phone users from using the application. A bespoke desktop application was considered for a long time, however, the user would have to ensure infrastructure decisions, such as databases, are correctly set up. As a result a web application was chosen - following research found; the next steps were to consider appropriate tools to use.

1.2.7 What should the application do

From analysing all three of the chosen research systems, it was clearly identifiable that they all have the ability the view all notes, searching, deleting, adding and editing a note. Taking these ideas on-board, they were set as a high-level task and something that the core system *must* do.

Reflecting on the premise of the application, that it was to aid the organisation lecture notes, it was concluded that the best way to search for notes would be by module code, as most University students would want to find specific module notes. This created the high level task that notes must be searchable by their module code.

1.2.8 Calendar integration

From evaluating the systems it was noticed that there was not a clear way to integrate into a calendar. Reflecting on the conversations with Dr Harry Strange, integrating with the calendar was important for keeping the different systems together. From an AYTM survey, in December 2015, [13] Google calendar is the most popular calendar application, therefore due to time constraints Google calendar was the choice of integration and other competitors such as Microsoft would not be implemented. This formulates the task of integrating the calendar into the application to save the url of the note to a specific event.

1.2.9 Objectives

As a result of the analysis of the problem, the following high-level requirements were formulated:

1. Investigate how to extract handwriting text from an image - this will involve looking into ways OCR tools can interpret handwriting.
2. Train the OCR to recognise text of the author's handwriting.
3. Produce a set of a rules which a note must comply to.
4. Produce a web application to form the core part of the product. This includes allowing a user to upload an image, display the image. Add appropriate tagging to a note such as module code.

5. The user must be able to search for a given module code, showing the fill list of notes based on the module code entered.
6. The backend of the application must conduct basic OCR recognition, analysing the first 3 lines of the notes.
7. The backend must integrate with a calendar to archive the notes away later to be found again.

1.2.10 Compromising with objectives

Some additional compromises were made in-light of the analysis due to the complexity of the tasks at hand.

- It would be nice to have image extraction from a note and incorporating a WYSIWG editor into the application, like OneNote.
- Full OCR on all the characters. This would then output the text to a blank canvas.
- Make the handwriting training generic enough to identify a wide range of users handwriting.

It is worth noting down that the project supervisor Dr Hannah Dee felt as though the handwriting training would be too much for the dissertation and should be done as a "maybe". After much deliberation it was decided to include it, but as a background process.

1.3 Process

Software projects often have a degree of uncertainty with requirements at the beginning, these projects lend themselves to an Agile approach. Whereas more structured applications with requirements which are well known are suited to a plan-driven approach.

For this project there are a lot of tasks which are not 100% definable at the start of the project. In addition to this certain tasks, such as training the author's handwriting data, can not be truly estimated down to a fixed time. Often new requirements would emerge from weekly meetings and only high level requirements were in-place from the start of the project. As a result, a plan-driven approach such as the Waterfall model would not be appropriate, and an Agile methodology was implemented.

1.3.1 Scrum overview

Scrum [15] is a methodology used by teams to improve productivity where possible. Due to this being a single person project, a Scrum approach has to be modified. Sprints are set time-boxes where tasks are completed. These vary from one to four weeks in length but a shorter sprint means the developer can act on quicker feedback.

Scrum organises its work into user stories to ensure client valued work is being completed. They are normally collected at the start of the project and put into the backlog, which is a collection of client values work. At the start of each sprint user stories are selected from the backlog with an estimation on complexity performed. Finally, at the end of the sprint a review and retrospective is conducted to analyse the sprint, identifying what went well and what could be improved.

1.3.2 Adapted Scrum

During the project this methodology was embraced and adapted. A one week long sprint was adopted which coincided with a weekly supervisor meeting. Epics (a high level version of a story) were identified at the start of the project to reflect the work completed in the analysis phase.

The epic was then broken down into user stories. Each user story was formulated as: "As a <role >I want to <feature >so that <resolution >". This gave specific client value that was known to have a purpose. Each of these stories were estimated on their complexity and compared to a "goldilock" task [1].

For planning a sprint, the planning poker [17] technique was adopted; user-stories are estimated on a scale of 1, 2, 3, 5, 8 etc. When a task was estimated about 15 story points, it would be reflected upon to ensure the scope was fully understood - this would be broken down to sub-stories where appropriate.

At the end of a sprint a review and retrospective was conducted in the form of a blog post [6], instead of in a team. The retrospective was used to analyse what was achieved in the sprint, what went well and what needed to be improved upon. During this time, pre-planning was conducted to formulate a series of tasks to complete in the next sprint; this was agreed by the customer (Dr Hannah Dee).

Communication with the project supervisor was key to determine what needed to be completed. It was discussed if what was suggested was achievable in that weeks sprint, based on the total story points completed in the previous sprint; if twenty story points were completed in sprint 3 then 20 story points were estimated for sprint 4 - associated user stories were brought forward.

The project was managed on the open source management tool Taiga.io [19] which was invaluable, and provided built in functionality such as burndown charts per sprint. This shows how well story points are being completed and are used as an analytical tool for how well progression was being made.

Daily stand-ups were informally conducted with a peer. Cut from the usual 15 minutes to around 5 minutes, the conversation helped to identify if there were any issues, what had been completed yesterday and what would be completed that day. This provided a good way to analyse what needed to be achieved and keep in perspective how the sprint was going.

1.3.3 Incorporated Extreme Programming

In tandem with Scrum, Extreme programming [2] principles were integrated into the development process; merciless refactoring, continuous integration and test-driven development were borrowed from its principles.

1.3.3.1 Test-driven development

Test-driven development (TDD) is the process of writing tests prior to the implemented code. This allows the developer to think about the design prior to its implementation and can form part of the documentation [8]. This was implemented throughout the project, with both unit and acceptance

[1] A task which all other tasks are evaluated against.

tests being written before the code implementation.

TDD follows three cycles: red, green, refactor. Initially the test fails, then it passes then refactoring is performed to keep the simplest system.

1.3.3.2 Continuous Integration

Continuous Integration tools were a core part of the process in this project. Typically used to ensure that code is checked into a repository it was used to ensure that the application could be built in an isolated environment and pass all the tests.

1.3.3.3 CRC cards

Class, responsibilities and collaboration (CRC) cards [21] were used during the design section to consider how different classes were to be created and the responsibilities they share. This principle from Extreme Programming helped to keep the design simple and not convoluted.

Chapter 2

Design

As the application was developed in an iterative manner, then no upfront formal class diagrams were conducted to design the system. Regardless, from the initial requirements it was clear that there was a set of design decisions that had to be made - these mainly were the implementation features. These would not change too significantly throughout the iterations.

2.1 Implementation tools

The following discusses the design decision regarding the implementation tools that were used during the project. This section provides rationale for key design decision which would impact the application.

2.1.1 Programming language

As the application was determined to be a web application then a design decision had to be made regarding the specific language to use for the server side. Normally for a server side application there would be a choice of languages to use such as: PHP, Ruby, Python, Java or more recently Node has increased popularity [CITE].

It was quickly decided in the first few meetings that OpenCV [CITE] would be needed to be implemented in the application at some point. Having previously used OpenCV with Python bindings it was known that Python could successfully integrate with a Python based application. Research was conducted to see if PHP or Ruby had some form of OpenCV wrapper, but after a little research there was not a lot of wrappers out there that looked promising.

As a result it was quickly established I'd have to use either Python or Java's wrapper for the application. Java applications tend to be more useful for large commercial applications, using enterprise software systems like Java EE[CITE]. That level of complexity was considered to be overkill for this application, therefore Python was chosen for the main language implementation due to the lightweight abilities, the ability to create web applications and knowledge of integrating with OpenCV.

2.1.2 Database management system

When considering the database aspect, one has to consider the different types of management systems available. The general consensus falls in the NoSQL vs SQL category. SQL management systems by their very nature are suited to data which is structured and has little variations. Whereas NoSQL is more suited to applications where there may be different values from time to time between the data.

After spending a long time evaluting the system and what could be achieved a SQL management system was selected. To do the basic tagging of meta-data would not warrant a NoSQL database structure, due to the information being structured and similar for every note. Couple this with the fact that there are a specific set of rules in which the notes must follow to tag this meta-data really suggested that the data will be nice and formulated, following the same structure for every note.

After determining that an SQL management system would be appropriate, a suitable technology would have to be chosen. When considering which technology to use, it tends to fall in the: SQLLite, PostgreSQL or MySQL. [CITE DIGITAL OCEAN LINK]

2.1.2.1 SQLLite

SQLLite was considered as a viable option; with most of the Flask documentation discussing implementing a database with SQLLite, it is intuitive to include in the application. However, considering the larger picture: would SQLLite scale well with multiple users using it. Testing and development would be perfectly fine, with a single person using it, but with more read and writes to the file, it probably wouldn't scale well.

2.1.2.2 PostgreSQL

PostgreSQL is the most advanced of the RDMS's available. The additional advanced, especially in the types give the developer more freedom to change their structure and implement more complex SQL queries if it were needed. For example its support for JSON means that Google responses could be asaved if needed.

2.1.2.3 MySQL

MySQL is a viable option as a RDMS for a web application development. It does offer a wide range of support. The documentation for MySQL is good, but compared to Postgresql there is better support [CITE THING.].

After evaluating the possible relational database management systems (RDMS) it was decided that there is not a lot of difference between PostgreSQL and MySQL - it probably would not yield a great difference in return. As a result, PostgreSQL was chosen due to being able to offer a wider support which may have been beneficial for future improvements to the application.

2.1.3 Choice of Framework

Just to recall Python was chosen as the language of choice for the application. With this in mind a framework had to be chosen to help aid the development of the project. Frameworks can be interesting things, they are their to aid you but sometimes you're constrained to their rules and regulations. This can lead to many frameworks being bloated with additional options that you would never want or need to use. That said, choosing a framework was an important process.

After analysing what Python web frameworks were available, I shortlisted three different ones: Django [CITE], Flask [CITE] and Bottle [CITE]. The latter two were classified as a "micro-framework" whilst Django being the full blooded framework - like Ruby on Rails [CITE]. After establishing that maybe a heavyweight framework was probably too much for the application and I wanted to have freedom to include my own libraries and feeling as though there's more flexibility, I opted to rule out Django.

This left Flask and Bottle frameworks. On the face value of things, Flask and Bottle look quite similar. They are both lightweight with similar syntax, however when delving a little deeper into the technologies I found that Flask had more of a support community compared to Bottle. As I was not familar with either framework then choosing one which provided a good level of community support in tangent with the documentation was important. On making the decision based on this factor then Flask seemed the more appropriate framework to choose for this application.

2.1.4 Continuous Integration tool

Although Continuous Integration (CI) is normally used for development teams to ensure that all code is checked into the repository[cite], there was value in using it for a single person project. Prodomently for the build processes on the branches to ensure that the application builds in an isolated enviornment.

Having used Jenkins CI before [CITE] this was my immediate choice when considering what to use for a continous integration tool. However, whilst considering the posibilities on how to integrate this to the application I discovered the CI tool, Travis CI. This CI tool seemed easy to set up: you link it to the repository, add a travis.yml file (with configurations) and it will run and build the application. This then gives useful messages such as passing, errors or failing.

One key decision whilst considering CI tools was to show be able to check the build process whereever I am. After finding out the Travis CI is in the cloud and links directly with the GitHub repo I opted to use Travis CI for the project.

2.2 Overall Architecture

2.2.1 CRC cards

To recap, Extreme Programming uses the concept of CRC cards to help to aid the developer to think about the design of the application. Unlike UML, they are throw-away cards which can be edited and changed every time a new feature is implemented. The CRC cards which have been drawn up in the application have been formalised, and a selection of the interesting design decisions and justifications are stated below.

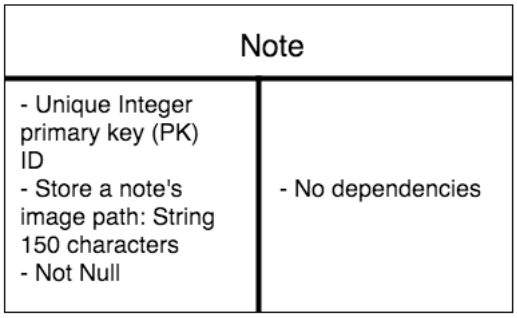

Figure 2.1: An example from Sprint 3, showing a CRC card at the very beginning of creation.

These little design diagram cards helped to consider the design of the system by decomposing the problem into more logical steps. For example, when creating the note CRC there could have been a time in which a separate class was made for the image link - however a simple design was ensured with a reflection on the card.

2.2.2 Class Diagram

- Use of service objects
- Explain why sections are linked together and describe the behaviour of the application classes and why they would be used.
-

2.2.3 Database diagram

Coupled with the CRC cards a database UML diagram has been produced which shows the implementation of the relations in the database.

Figure 2.2, shows the output from the result of continuous design on the database diagram through the iterations.

2.2.3.1 Justification of design

The design has been carefully considered. The reason the note is in its table is intuitive: a note needs to be persisted in the database. The attributes best collate a high-level representation of what a note consists of. First and foremost, a note will consist of an image link - which is a relative path, this was stored to persist where the image on the file store is located for that image. A user id was added to the note, to link a note to a user; not originally in the scope of the application, but a design feature added so that it was more accessible.

The calendar url was added to the database to persist which calendar event was associated with the note. As a note will only have one calendar event, then saving it in the database was an easier way to link to the event - rather than making an external API request to Google.

Finally with the note relation, a note-meta-data foreign key id was added to link additional note-meta-data to a note.

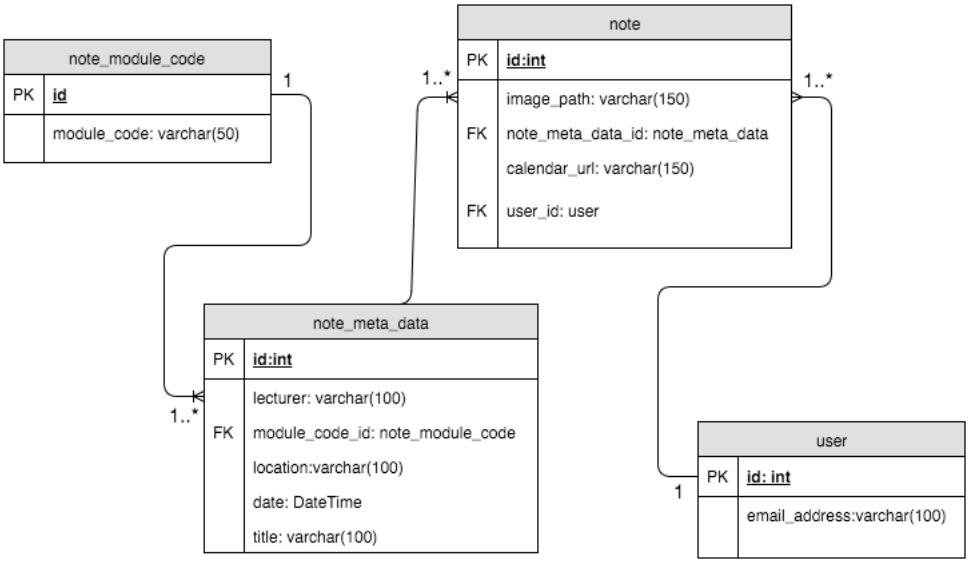

Figure 2.2: The final result of the database diagram. After a series of iterations.

The Note-meta data relation is its own relation because of reducing data-redundancy, this is a key requirement in normalisation of databases [CITE]. The note Meta-data may be duplicated in a note if a user takes more than one note per lecturer - and wants to tag the same meta-data to the note. As a result a relation was created for this use-case.

The meta-data consists of a lecturer, location, and title as variable characters up to 100 in length, which is a valid length as none of them realistically should be more than that length. A date field was added so the meta-data can be associated to a date and time - this aids in the calendar integration. Finally, the module code is identified as a foreign key constraint.

The module code is its own relation, and therefore a foreign key in the meta-data relation is because of the same reasons as meta-data is it's own relation: data redundancy. A valid use-case is that a user would enter in multiple notes for any given module code. This module code does not need to be duplicated in the database - therefore a foreign key was suitable.

Finally, a user relation was established after the application decided it would be expanded to involve users. An email address was created to parse the response from the oAuth log-in. Furthermore, this was added as a foreign key to notes, so that every user would be associated to a note.

Overall, a succinct collection of relations have been developed, which follow a good normalization process to produce a design from a database side which has been well considered.

2.2.4 MVC Structure

Early on in the process it was decided that an MVC approach would be approached. This was an upfront design decision, that would not likely to be changed.

2.2.4.1 About MVC

MVC is a design pattern where you split the logic into a Model View and controller. As displayed in figure 2.3 the MVC approach helps to decouple logic from the view file. Therefore all the information passed to the view is renderable content. The controllers do not integrate with the database directly. The primary job for the controllers is to interact with the model, any services and ask to render the view files.

The model in the MVC structure has no acknowledgement of the view file. Instead of rendering any form of HTML in the model it is purely data-driven. The sole purpose of the model is to interact with the database and perform any business logic that does not fit in the controller and the view file.

Finally, the view files contain HTML logic with dynamic content passed to the file from the controller. There may be specific logic which impact the HTML displayed, but no direct calls are made to the database layer or the controller. It uses the dynamic content passed in.

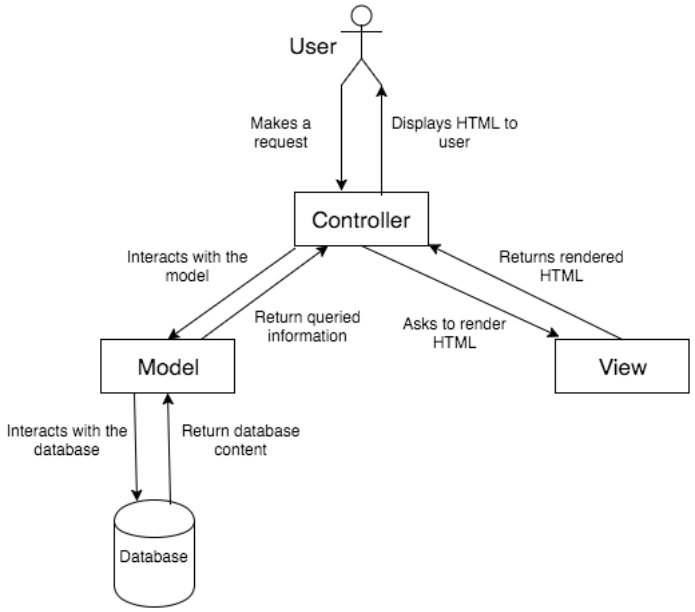

Figure 2.3: A example of how the model-view-controller(MVC) framework integrates.

2.2.4.2 Structuring the Web application

The application was chosen to follow the MVC approach so that there was a clear distinct design pattern used, which did not obfuscate where logic and presentation layers interact. With a well structured system it makes testing easier.

In addition to the structuring, components can be reused again easily. Take for example service classes, these are reusable components in the system where they can be easily instantiated from inside any controller. If an MVC approach was suggested for the design then the reusability of code would decrease. This could only be achieved from a nice decoupled system where independent logic can be processed separately.

Although the MVC approach was desired, Flask does not support an MVC structure out of the box. During a lot the documentation it expects all routes to be placed in a single file. This design approach was not considered for a number of reasons: it reduces the readability of the code - if there is more than one class per file, then readability begins to be impacted. Furthermore, when considering the design of the controller objects, as well as the models, then dependencies between the classes would become obfuscated. From identifying an architecture perspective, if the routes were on one file and models in another, it would be hard to explicitly identify corresponding links between different files.

To overcome the Flask routing issue, in recent versions [FIND VERSION NUMBER] Blueprints [CITE] have been included. These blueprints, act almost as controller files, like Ruby, using the routing annotation to create a route. Therefore, similar routes can be place in the same file, but a different file can be created for unrelated routes. This offers a good controller feel to the application.

The models consist of both persistence logic as well as service and helper objects. All the business logic is kept in side this group. Finally, all the views are allocated to the view directory.

2.2.4.3 Extending views

It is worth mentioning that Flask uses the Jinga templating engine[CITE]. By default Jinga would expect the DOM to be represented again in every view file. A design decision was made to extend the view files. Figure 2.4 illustrates the view file design.

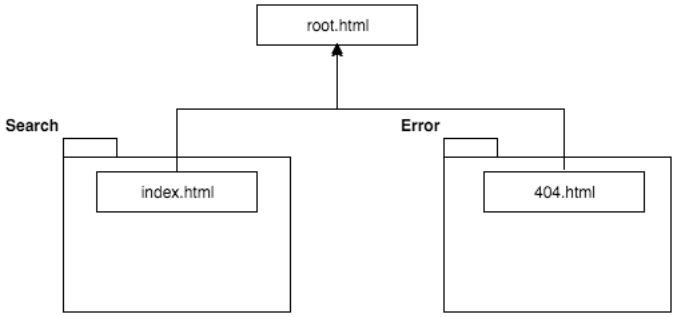

Figure 2.4: A diagram illustrating how extension in Jinga html template engine works.

Each of the view file templates are separated into their own directories. Inside the directories, the HTML files extend the index.html root file. This overrides the block “content”, dynamically generating a new content for every new URL. The advantages of this design is that meta-data, headers, stylesheets and navigation can be placed in the index.html page and all subpages will inherit these automatically, sticking to the principle of DRY.

2.2.5 Application URLs

When considering the application URLs would form an integral part of helping the users to identify where they are, what the page is aiming to achieve and, where applicable, bookmarkable.

Therefore designs into the URL structure was performed. Typically there are two types of URL structures which websites take: query strings or REST URLs.

Query strings aid in the creation of URLs such as: /search?name=foo. The query string is represented as a key value pairing of the searched criteria. This form of URL representation is acceptable for search criteria, but what if the user was editing a note - would a good structure contain all the contents of that note?

For parts of the application to overcome this issue REST [CITE] URLs were considered. By exposing the URLs to RESTful interface, it is clear what the application is intending to do on that page. For example $/show_note/1$ compared to $/show_note?note_id = 1$. Therefore, the URLS will follow a hierarchal structure, showing exactly where the notes are, in this case [CITE].

It is worth noting that RESTful URLS are a good concept for external API usage. However, due to the application being a web application some modifications were taken place. For example, the URL: $/view_notes/$ would be a better RESTful url as /notes/. However, the design decision was to adopt the RESTful pattern to help make the URLs a bit more readable, but still keep the principles of REST.

2.2.6 HTTP methods

Flask supports all the REST HTTP methods, such as delete, put and patch[CITE]. However, normal web browsers have this common issue where they do not support them and instead only support GET and POST.

As a comparison in Ruby, the templating language - embedded ruby - allows being able to fake certain methods on forms and submit buttons. Unlike Flask, who uses pure HTML templates, with the Jinga syntax added on top, then developers are restricted to the default GET and POST.

Another layer of code could have been added in the form of posting everything to the URL routes on the server as AJAX requests - but that would first and foremost change the entire structure of the application. This would mean that a user would have to have Javascript enabled on the page to even interact with the application.

As a result, the design decision to use default POST and GET requests was decided. DELETE would not be used for deleting content from the database.

2.2.7 Interaction with the application

When decomposing a problem it is useful to identify how a user would intend to interact with the system.

2.3 User Interface

Although the application was developed in an iterative approach, wiremocks were completed when UI changes occurred so that I could reflect on them.

To begin with the user interface (UI) went through a few iterations of wiremocks to sketch out what it should look like. This iterative approach was applied when actually designing the website. Firstly it was important that the application felt as thought it was an application, rather than a traditional website. Colours from Google Style guide [CITE] was used to keep with nice, clean and bold colours. Bootstrap[CITE] was considered to be used on the application and it did

give a default responsive theme built in - however, keeping with the idea of minimalist and not over-bloated, then personally I feel that that bootstrap comes with a lot of the additional material which would not be needed.

With this being a web application, a responsive navigation was always important. A user may take a photo of their note and then upload this to the website off their mobile. This resulted in the web application having to be thought about from a responsive view. I should have done this better and considered this from the beginning and built up from the responsive to the desktop version. However, I did retrospective responsive design and performed media-queries when I needed to.

- Wiremock examples
- Media query code examples.

2.4 Other relevant sections

- Why did I use a Calendar Item class to story the class and then format the date from that instance.
- Why Parsing the notes returned the first 3 lines. Why did this have to be done?

2.4.1 Tesseract

Comparisons between different OCR tools was researched. Evaluations into systems which would be a suitable aid was undertaken. [CITE - Case study] performs a case study using Tesseract as their OCR tool to analyse printed text in an image. Patel et al, also discuss the comparison against a proprietary OCR tool, Transym [CITE].

The results concluded by Patel et al is that Transym only yielded a 47% accuracy on 20 images compared to 70% accuracy using the Tesseract engine.

Analysing the outputs concluded that Tesseract would be the best performing on an image. Tesseract was chosen for its open source and it's highly commended recognition rate as well as my supervisor echoing the compliments of Tesseract and it should be considered.

2.5 Optimising tesseract

2.5.1 ImageMagick

Once Tesseract was chosen as the OCR tool training was started promptly. Due to there not being a formal way images were supposed to be prepared for the tool then the first few iterations were converted to grayscale using ImageMagick [CITE]. This yielded a poor return rate and characters were not identified consistently.

At this point in the design, the notes were being written on original lined paper. Research work was looked into to try and get the image as clean as it could be for the Tesseract engine. This resulted in monochrome from ImageMagick to be used; an increase in recognition rate was

received, however the image was very messy with pixels from the lines obfuscating the response from Tesseract.

2.5.2 OpenCV

During a meeting with Dr Hannah Dee, it was suggested that OpenCV should be used. It was already a design consideration - as to why Python was chose - so it should be used to facilitate the binarisation script.

OpenCV is an open source image processing library. Natively written in C++, but there are Python bindings available - so the Python version was selected, to fit into the web application.

- Realised it wasnt good so redesigned it to custom lined paper Why did I do this instead of normal lined paper?
- Why the colour blue ?

2.6 Version control

Chapter 3

Implementation

3.1 Image processing

3.1.1 Optimising Tesseract

After the design considerations to use OpenCV was chosen, specific algorithms would need to be implemented. After a discussion with Dr Hannah Dee, it was suggested that investigations into different binarisation scripts would be useful to see how an image can be converted to black and white from an image.

3.1.1.1 OTSU

[CITE CITE, Lab brooks page] OTSU, created by Nobuyuki Otsu, is a binarisation technique which allows the conversion of an image to black and white. Otsu is a global thresholding algorithms, where it takes all the image and compares that, rather than local thresholding, where it's pixel by pixel. [CITE Automatic thresholding for defect detection].

OTSU requires uniform lighting conditions to be fully effective, this is naturally a problem when taking notes due to shadows being placed over the note.

With OTSU, the image is seperated into a histogram of grey-level values. OTSU determines the optiminal threshold value from the histogram, by "maximising the discriminant measure"[CITE OTSU PAPTER][CITE OPENCV]. Once the threshold has been established, it will serparate them into the foreground and background pixels. [CITE LAB BROOKS PAGE].

[CITE HP PAPER]Tesseract uses OTSU as its preprocessing method for the tiff files presented to the system. From analysis and experimentation using the OTSU method, it was clear to see how exactly the image was failing with simple just a grayscale value. Fig X, shows how Tesseract would intepret the image, making it almost impossible to see characters on the page.

3.1.1.2 Adaptive Threshold

From the enlighting analysis of OTSU it was realised that an OTSU thresholding ontop of an OTSU threshold from the Tesseract engine would not be beneficial. To account for the changing

of light source across an image, an adaptive threshold was chosen [CITE somewhere which says this].

Adaptive threshold works by doing local thresholding across the image, rather than the global thresholding OTSU does. Therefore, there is more chance that shadows on an image would be eliminated from the local thresholding.

During the trials of optimising the images for Tesseract, other thresholds were tested: Fig x shows the binarisation and thresholding algorithms implemented on a note.

From Fig x, it's unoquivacal that the adpative threshold using mean and gaussian returns the best binarisation success rate. The text on the image is legiable, and it produces an image which clearly segments the text from the background; the mean has more noise on the image, but these would be cleared up with morphological events.

As a result the adaptive threshold function with Gaussian blur was chosen to be the thresholding used for the pre-processing.

3.2 Lined paper

Initially, standard lined paper was used; the noise that was produce was far too much to make Tesseract analyse characters correctly.

Due to Tesseract requiring text to be in horizontal lines [CITE HP PAPER] plain paper would often skew the text and make it harder for Tesseract to interpret the text.

3.2.1 Filtering the blue lines

As a result, custom blue lined paper Fig X, gives equal spacing between the lines as well as a horizontal structure. Blue was initially chosen as the pre-processing script would filter the blue lines and leave the black, binarised, text. The text would then appear as though it was written on normal paper and should not be included with the noise from the image.

This process went through a series of iterations to try and overcome issues identified through each iteration. The first process was to extract all the values on the page which fell between a predefined grey-black range. However, this had its obvious problems that it would extract some of the line where a dark blue would contain black-like elements of colour.

Morphological operations such as erroding [CITE] and dilation [CITE] was performed on the image, but to no avail - there was still noice in the image which causing Tesseract to interpret them incorrectly. This also had an unwanted side-effect: the binarised characters would then become so undistinguishable that they would not be able to be located correctly by the OCR tool. They became so pixelated that even from a human eye it was hard to identify their meaning.

3.2.2 Only extracting the text

The following optimisation would identify the lines, but only extract the text that was needed. [CITE EXAMPLE ON OPEN CV] A gaussian adaptive threshold operation was performed over a median blurred, grayscale version of the note; successfully binarising both the text and the

lines. The horizontal lines on the page were extracted using OpenCV's structuring element, $MORPH\_RECT$ [CITE].

Further errosion and dilation was applied to remove the black lines and any noise residue from the extraction. A series of intermediate blank image masks were created to transfer the text from the image across to the mask. Although this carried all the text over, further line noise was also transferred across.

This suffered similar problems are the previous iteration, until connected components, via contours was discovered. Due to the morphological operations on the horizontal lines the connectivity between the pixels was very small. Thereby, choosing to use connected components would be able to extract the text from the image with little interference from the horizontal line noise. These connected components were then transferred to the new mask - only carrying the text from the image, not the lines.

Final morphological operations were included to improve the image quality. Overall the binarisation script works well. It can take a photo of an image in a terrible light source Fig x, for example, and it would succesfully output the binarised image which would have distinguishable text, coinciding with little noise on the image. The noise on the image has little interference with the Tesseract OCR and it yields better results than greyscale images.

3.3 Handwriting Training

During the start of the handwriting training phase problems occurred such as it not reading the characters from a greyscale image correctly.

3.3.1 Training process

When using training data to be worked with Tesseract, it requires it to be in a specific file format. Such as: $< lang > . < font > .exp < number > .tiff$. When ran through a language, Tesseract outputs a box file which contains on each line: the character and the coordinates of this box. Trying to analyse this box was almost impossible. On the Tesseract wiki page [CITE] there's a link to jTessBoxEditor [CITE]. This tool was used to tag the boxes with their associated content.

Whilst using this editor the boxes could be expanded or shrunk to give the best possible fit to the characters. This was often utilised due to erroneous characters picked up by the editor.

3.4 Web application

3.4.1 OAuth

3.4.2 Reoccuring events

Reocurring events were discovered as an issue in the pre-beta user testing. During the design phase when thinking about the calendar, it was forgotten that reocurring events and all-day events could exist.

All-day events do not have the dateTime key response from the Google Calendar API. As a result the code would fail when trying to access the dateTime from the event start date. This resulted in a redesign and re-think of the possible issues which could arise from Google Calendar.

Eventually, the dateTime key was checked and the all day events issue was solved.

However, the reocurring events problem was still existing. When querying for an event, if the event was reocurring then it would group the reocurring events by the first time in which the event was created. This resulted in an image, which was taken on the 12th March 2016 for example, to show events in February - if there was a reocurring event on the 12th March. However, it had an important reoccurence event ID key.

This resulted in a further query being created which would return all the instances that were reocurring. This had to pass in the $event['id']$ to the query to return all these instances; it was filtered down by querying for the start and end date.

When editing a reocurring event, Google calendar performs some unexpected behaviour: instead of silently modifying the event and returning the grouped event, again, it instead returns both the grouped event and the edited event. A succinct solution for this has not been found and has been a slight issue.

3.4.3 Tesseract Confidence

During a meeting with Dr Hannah Dee, it was suggested that some form of confidence score couldbe outputted to the user to show how well Tesseract identifies the text from the image.

The Tesseract command line does not output the confidence of the characters identified; only the C++ library can output the confidence. Due to time constraints, a wrapper for the C++ Tesseract API could not be implemented - so a third party library was chosen, tessocr[CITE].

Tessocr offered the implementation to access the confidence values for the associated words. The algorithm to execute the identification of the characters was quite simple.

Firstly we get all the text lines; Tesseract deals with the lines as a series of text lines. This is then enumerated over and for each line a corresponding list of confidence words is collected from tehe map_all_words API.

Due to this returning a tuple which is an immutable object in Python, modifications on the list could not be made easily. This resulted in the view file checking the tuple content and calculating whether it is above the threshold; 75 for green, 70 for orange and below 65 for red.

3.4.4 Displaying calendar events

3.4.5 Parsing Exif data

3.4.6 Editing calendar events

3.5 Review against the requirements

Chapter 4

Testing

Testing would form the core part of the application for *MapMyNotes*. Due to it prodomently being a web application then testing was an important process. In conjunction, there needed to be a wide-range of testing included to cover all possible approaches.

4.1 Overall Approach to Testing

To recall an agile approach was adopted throughout the project. As a result, a test-driven development approach was conducted.

4.1.1 TDD

Test-driven development was adopted throughout all aspects of the application. A solution always had an associated test and all code in the application has a purpose with a test behind it. Figure 4.1 shows the TDD cycle.

A sensible test was created and, following the cycle, this would fail when run. The following steps would be to ensure that the tests pass by adding the associated code needed to make sure that it passes - afterwards refactoring occurs to ensure that design is kept simple and as clean as

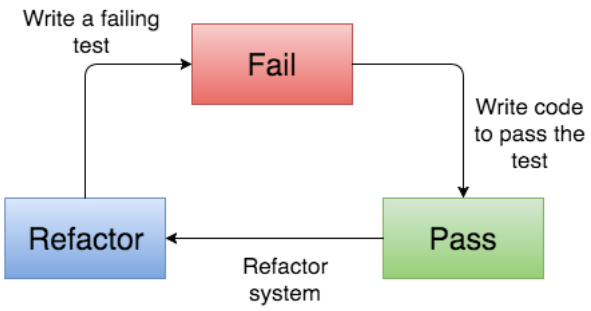

Figure 4.1: The cycle of TDD during the development stages of the application

possible for the current implementation.

This approach could have been modified so that a group of tests were created for a feature and then implement a set of functions. This testing strategy was rejected and a pure one test for one bit of functionality was used. This was mainly to ensure that the design was not being over complicated.

Reflecting on the testing strategy and the design, it really did help to think of the design through a test-driven-development way. It ensured that the domain was fully understood before creating a test. This left the codebase tested fully - through a variety of aspects.

4.2 Automated Testing

One thing to note is that Flasks testing documentation is very sparse and is of low quality.

To begin with py.test was originally being used to run the test suite and tests were created with just unittest.TestCase.

4.3 Mocking Tests

During testing it would be established quickly that tests would need to integrate with the Google API as well as the Tesseract helper class.

The need to for mocks vary between applications but thinking about the Google API example. [Cite] It is best practice not to hit a live production URL unless its in production. For testing Google's API's do not offer such luxury. After looking around they suggest that mocking would be a suitable idea.

The Tesseract example it may first seem odd to mock the response of certain functions. However, consider the possibilty of training data and then testing the application, then training some more. The results are likely to change - therefore to ensure that the tests have consistency and will work, data has been mocked to return a sensible output. Personally, the test cases was complex and not exactly the most intuitive.

During the first few iterations, whilst getting used to how it works, then working how to deal with mocks were conducted via annotations above the test function.

```
@mock.object(GoogleOauthService, 'authorise')
@mock.object(GoogleCalendarService, 'execute_request')
def test_return_correct_response(self, authorise, calendar_response):
    authorise.return_value = some_json
    calendar_response.return_value = some_more_json
```

This style of the testing syntax causes the codebase to become far too messy and unreadable. Additionally, large chunks of functionality was being duplicated going against my DRY (do not repeat yourself) principle.

Eventually, looking through the API docs for the mock object a patch object call was discovered. This allowed for a much cleaner test code base. These patches could be located in the setUp and tearDown function, replacing the annotations at the top of test functions.

The following code makes the mocking a lot more succinct, keeping it in the setUp functions and reducing the duplication

```
def setUp(self):
  # some code
  authorise_patch = mock.patch()
  authorise_mock = authorise_patch.start()
  authorise_mock.return_value = some_json

def teatDown(self):
  mock.patch.stopAll()
```

Side effects....

4.3.1 Unit Tests

After refactoring the testing infrastructure to use Flask-testing fraamework instead of the default python unit test, then unit testing became event easier.

The tests would extend the libraries TestCase class. This allows the application access to the application context (the current application) to test models.

Unit tests were conducted for all the classes in the model directory. Model testing themselves was relatively easy and simple to implement.

4.3.2 Route Testing

As the application was using routes then tests were conducted to ensure that the routes could be hit. Whilst doing TDD these were often the first tests to be written. This allowed design considerations to be thought over regarding specific routes.

Other aspects of testing the routes ensured that the correct response codes were returned, ensuring that the routes were working as intended from the very core of the application. Additional tests were added to check that any redirects worked correctly, to ensure flow between controllers were being executes where applicable.

Finally, in some sections persistance testing occured where a test database was checked to make sure that code inside the controllers was being executed. For example, when adding a note the note route checks to make sure that a note was created in the test database.

4.3.3 Handling sessions

One of the tricker aspects regarding testing was the handling of sessions. In parts of the application sessions are used to handle specific states of the system, i.e user logged in.

During testing of routes then session handling would not be too complex. The flask documentation gives a clear breakdown of how to modify a session. To begin with you have to open a test client context - this mirrors what a normal client would look like. From there a session transaction context needs to be opened. Once this has been opened then it would allow for session modification.

Issues arose during the acceptance tests where session modification would need to happen. Issues arose due to the fact that selenium would run on a different process to the application. As a result session modification could not occur during using selenium and acceptance test. This caused a lot of problems regarding testing. As a result, more mocking had to be used. This time the session helper would be then ended up mocking.

Overall, session handling was one of the hardest aspects with testing to get around and find a suitable, yet clean solution.

4.3.4 User Interface Testing

4.3.5 Stress Testing

4.4 Integration Testing

Integration tests would aid in thinking about not just the backend models, but how the information would be presented to the end user.

These tests were implemented in the form of using Python's selenium tool. As previously used before, selenium is a testing tool which allows you to integrate with the DOM and perform automated browser testing. These tests are important to ensure that the user interface is tested for input and ensuring values are correct.

An example of a selenium test to ensure that when the user goes to the file upload page and they're entering information into the meta-data form that associated calendar events are correctlt displayed.

Another useful test where selenium works well is when checking the background colour of the Tesseract output. By mocking the tesseract output the tests were able to test logic such as identifying the confidence score to the correct class name - and therefor the classname being identified with the correct background colour of the text.

4.5 User Testing

Due to the application aiming to solve a problem with a set of students then user studies were conducted and teh responses were analysed and evaluated.

4.6 Tesseract Testing

Due to there being no code written for the Tesseract training process there were no formal tests conducted for this section. However, what could be tested was how well the Tesseract learned as it was going through the learning process.

A test framework was produced which ran the training process on the image. As mentioned, it outputs a text file which is associated to the words identified. This file was then renamed to $eng.ryan.expx_stat.txt$ so that when the final part of the training process is ran it did not overwrite the file.

A statistical analysis was collected on the outputted characters from each of the training examples. Each of the characters were identified on each line of the text file, where each character represents a box which Tesseract has identified as a character.

4.7 Image thresholding Testing

When writing my script for the image thresholding research it was soon discovered that the script would be very trial and error and very much like a protoyping.

When testing the output from the image, then it would be a visual check rather than a specific image that was looking to be achieved. Looking at each itteration through the scripts improvement it would be evaluated as whether it was a worse or better binarisation script. Once a sufficient level of satisfaction from eye-judgement was achieved then the testing from that script was stopped.

Once the spike work for the script had been completed, TDD performed on the image to turn the prototyped script into clean testable code. Using the outputted numpy arrays from open cv, they were compared against what was believed to be outputted.

When the image converts the white image mask to black it performs checks for the image for any black value in the image.

Chapter 5

Evaluation

5.1 Correctly identified requirements

To recap, the following requirements and objectives were identified at the beginning of the project:

1. Investigate how to extract handwriting text from an image - this will involve looking into ways OCR tools can interpret handwriting.
2. Train the OCR to recognise text of the author's handwriting.
3. Produce a set of a rules which a note must comply to.
4. Produce a web application to form the core part of the product. This includes allowing a user to upload an image, display the image. Add appropriate tagging to a note such as module code.
5. The user must be able to search for a given module code, showing the fill list of notes based on the module code entered.
6. The backend of the application must conduct basic OCR recognition, analysing the first 3 lines of the notes.
7. The backend must integrate with a calendar to archive the notes away later to be found again.

The afformentioned requirements were part of the core system requirements; these were clas-sufied as the minimum which the application must do. Personally, these targets, partly self-imposed, were extremeley challenging. The amount of work that was taken on, with lots of different components required that work was produced at a steady pace.

Investigation and research work was extensively conducted to identify how handwriting can be extract from an image. This was further enhanced by looking to optimise the performance of the Tesseract engine. Not only was the image binarised - there were studies and iterations of improvemet which would help to benefit the end user.

After twelve different training examples, from hand, the Tesseract enging is not yielding any greater of a success. Therefore it can be concluded that a sufficient job has been achieved with handwriting training. Probably one of the most frustrating experiences with labelling individual

characters. Nevertheless, the training could not return a greater success than currently therefore good work has been completed to get it to that stage.

With the web application being the core part of the problem, then a lot of effort was centered in on ensuring the correct functionality was added. Personally, the application reflects that of the work carried out on it. A user can upload an image to the application and tag all associated meta-data, including the title, module code, date, and location.

The application creates an easy to use interface that allows the user to search for a given module code with ease. The case in which they enter the text is not important, as the application handles the case-sensitivity allowing for a better user experience.

One of the more impressing features of the application is the handwriting recognition integration. It parses the first three lines of the image and shows it to the user in an intuititve manner. This was built ontop of the work already implemented in the project with the handwriting recognition. With the final steps included the application has its true purpose, and that extra bit of complexity which personally the application needed.

Finally, from the original requirements the calendar integration was succesfully implemented; it was in retrospect one of the more challenging tasks to overcome. Ranging from the testing the services to implementing reocurring events, Google calendar always threw up a lot of issues. The reocurring events and all day events were particially frustrating, however there has been a wide interaction with the calendar. It can list all events in the last 7 days, find specific events by day and is able to find reocurring events. Ontop of this is can add to an event description and remove a specific string from the description field.

Overall, the application produced meets all the core requirements and goes beyond, in some aspects, the original features. The app has been well tested with a strong testing infrastructure backing the application. There have been complex design decisions to ensure that the design of the application is simple and intuitive to a reader. Personally, the app has been completed to a good standard.

5.2 Design decisions

5.3 Use of tools

5.3.1 Flask

During the project there were times in which there were feelings that the toolset should be different. Flask, the micro-framework for the web application, was pehaps a bad decision in hindsight. There was a lack of documentation and out-the-box support made even the most basic of functionality difficult and uncessarily complex. The configuration files for example: there was no default support for multiple support for default configuration files.

Another example is default security concerns. As stressed the importance of in *Internet based applications* cross-site request forjery (CSRF) is a big issue among internet security. By default Flask does not support CSRF checking. An additional 3rd party library, SeaSurf, had to be installed to handle the basic functionality of this.

That said, Flask did offer a great deal of flexibility regarding the project. Directories were

customisable easily, and the use of blueprints enabled that MVC feel to the project. There were times which just the basic support for a framework would have aided the developer and helped to improve the productivity, rather than being subjected to more external code.

5.3.2 OpenCV

The choice of using OpenCV compared to ImageMagick proved to be an extremely important design change. This helped the Tesseract training to increase substaintially, without the use of OpenCV it cpuld be argued that the Tesseract training would never have gotten off the ground. The research conducted into looking into different binarisation techniques was imperitive; there could have been a long chase going down the greyscale image route, trying to train the data and not getting anywhere.

5.3.3 PostgreSQL

The use of PostgreSQL was overall a good choice. Potentially all the decisions regarding using that over MySQL were not fully met, restrospecitively analysising the decision. However, the decision to use that over SQLlite, especially as more than one person would use it would be a wise decision that was right to be made.

However, one aspect which should be noted was that if the project was to start again then a more serious look into the NoSQL database should be considered. During the application development, it was acknowledged by Dr Hannah Dee that it would be good to use the application for conferences too. The conferences do not follow the same structure as that of a lecture, prodominetly lecture name and module code.

Due to the variability in the differing data returned for different event then prehaps a NoSQL approach would have been beneficial.

5.3.4 Tesseract

Tesseract itself has been a great tool during the creation of the product. The training process was a little tedious but the tool would not be swapped out for another form of OCR tool.

5.3.5 Google Calendar

Although Google Calendar threw up a lot of issues along the way, the choice was a good decision in the end. After analysing the calendars choosing the most popular one appealed to more of the market.

Additionally, all the good support that Google gave [CITE Issue] was beneficial in the end, due to a large community of developers available.

5.4 Meeting the users needs

5.5 Additional project aims

Due to the nature of the project, all the features and functionality could not be implented into one project spanning just 15 weeks. Due to this a few requirements were cut back from the initial specification.

There are tasks such as: auto-skewing and cropping the image on upload, search by date, share link with other peers, extracting images from the text and training it on generic handwriting. These features would have made the application a lot more useable for the user. ADD SOME MORE

5.6 Starting again

If this project was to start again, there would be a stronger emphasis on the design considerations for the bigger picture. As previously stated, a mongo database would have been chosen over a RDMS, offering more flexibilty and scope for the application to become a viable success.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the clients existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Appendix B

Ethics Submission

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

Appendix C

Code Examples

3.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [**?**].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*-------------------------------------------------*/
  /* Minimum Standard Random Number Generator        */
  /* Taken from Numerical recipies in C              */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity   */
  /* Always call with negative number to initialise  */
  /*-------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
```

```
static long iy=0;
static long iv[NTAB];
double temp;

if (*idum <=0)
{
  if (-(*idum) < 1)
  {
    *idum = 1;
  }else
  {
    *idum = -(*idum);
  }
  idum2=(*idum);
  for (j=NTAB+7;j>=0;j--)
  {
    k = (*idum)/IQ1;
    *idum = IA1 *(*idum-k*IQ1) - IR1*k;
    if (*idum < 0)
    {
      *idum += IM1;
    }
    if (j < NTAB)
    {
      iv[j] = *idum;
    }
  }
  iy = iv[0];
}
k = (*idum)/IQ1;
*idum = IA1*(*idum-k*IQ1) - IR1*k;
if (*idum < 0)
{
  *idum += IM1;
}
k = (idum2)/IQ2;
idum2 = IA2*(idum2-k*IQ2) - IR2*k;
if (idum2 < 0)
{
  idum2 += IM2;
}
j = iy/NDIV;
iy=iv[j] - idum2;
iv[j] = *idum;
if (iy < 1)
{
  iy += IMM1;
}
```

```
   if ((temp=AM*iy) > RNMX)
   {
     return RNMX;
   }else
   {
     return temp;
   }
 }
```

Annotated Bibliography

[1] "Evernote Tech Blog — The Care and Feeding of Elephants," https://blog.evernote.com/tech/2013/07/18/how-evernotes-image-recognition-works/, 2013, last checked 25th March 2016.

An article explaining how Evernote does character recognition on images

[2] R. Agarwal and D. Umphress, "Extreme Programming for a Single Person Team," in *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ser. ACM-SE 46. New York, NY, USA: ACM, 2008, pp. 82–87. [Online]. Available: http://dx.doi.org/10.1145/1593105.1593127

This paper was useful on how Extreme Programming can be modified to a single person project. It provided thought on the methodology which should be undertaken on the project and how different aspects of Extreme Programming can be used.

[3] M. A. A. Dzulkifli and M. F. F. Mustafar, "The influence of colour on memory performance: a review." *The Malaysian journal of medical sciences : MJMS*, vol. 20, no. 2, pp. 3–9, Mar. 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3743993/

A paper reviewing whether colour helps with memory retention. Used for the analysis and further confirmation in the taxonomy of notes section.

[4] Evernote, "The note-taking space for your life's work — Evernote," https://evernote.com/?var=c, 2016, last checked 17th April 2016.

The Evernote application is an example of the organisational and note-taking application that this project is looking at as a similar system.

[5] Google, "Meet Google Keep, Save your thoughts, wherever you are - Keep Google," https://www.google.com/keep/, 2016, last checked 17th April 2016.

Google keep is an organisational and note-taking application, it is used as part of the evaluation and background analysis. It was compared against what the application could do.

[6] R. Gouldsmith, "Ryan Gouldsmith's Blog," https://ryangouldsmith.uk/, 2016, last checked TODO.

A collection of blog posts which explain the progress every week through a review and reflection post.

[7] S. Knerr, L. Personnaz, and G. Dreyfus, "Handwritten digit recognition by neural networks with single-layer training," *IEEE Transactions on Neural Networks*, vol. 3, no. 6, pp. 962–968, Nov. 1992. [Online]. Available: http://dx.doi.org/10.1109/72.165597

A paper describing how a Neural network was build to identify handwritten characters on the European database and the U.S. postal service database.

[8] C. Maiden, "An Introduction to Test Driven Development — Code Enigma," https://www.codeenigma.com/community/blog/introduction-test-driven-development, 2013, last checked 17th April 2016.

A blog post giving a detailed description of what Test-driven development includes. Gives supportive detail to discussing that tests can be viewed as documentation.

[9] Microsoft, "Microsoft OneNote — The digital note-taking app for your devices," https://www.onenote.com/, 2016, last checked 13 April 2016.

Used to look at and compare how similar note taking applications structure their application. Used the applicatation to test the user interface and what functionality OneNote offered that may be usedful for the application

[10] ——, "Office LensWindows Apps on Microsoft Store," https://www.microsoft.com/en-gb/store/apps/office-lens/9wzdncrfj3t8, 2016, last checked 17th April 2016.

The Microsoft Lens application which would automatically crop, resize and correctly orientate an image taken at an angle.

[11] ——, "Take handwritten notes in OneNote 2016 for Windows - OneNote," https://support.office.com/en-us/article/Take-handwritten-notes-in-OneNote-2016-for-Windows-0ec88c54-05f3-4cac-b452-9ee62cebbd4c, 2016, last checked 17th April 2016.

An article on OneNote's use of handwriting extraction from an image. Shows simply how to extract text from a given image.

[12] O. Olurinola and O. Tayo, "Colour in learning: Its effect on the retention rate of graduate students," *Journal of Education and Practice*, vol. 6, no. 14, p. 15, 2015.

Discusses a study which shows that coloured text is better for the memory retention rates, than that of non-coloured text. Used during the taxonomy of notes section.

[13] A. Pilon, "Calendar Apps Stats: Google Calendar Named Most Popular — AYTM," https://aytm.com/blog/daily-survey-results/calendar-apps-survey/, 2015, last checked 13th April 2016.

A survery showing that Google calendar was ranked the most used calendar people use. Added to the analysis stage to justfy why Google calendar was chosen instead of other calendars available.

[14] S. Rakshit and S. Basu, "Recognition of Handwritten Roman Script Using Tesseract Open source OCR Engine," Mar. 2010. [Online]. Available: http://arxiv.org/abs/1003.5891

The paper presents a case-study into the use of the Tesseract OCR engine. It analyses how to use train the data on handwriting based recognition, drawing conclusions on where it's useful - as well as it's downfalls.

[15] Scrum.org, "Resources — Scrum.org - The home of Scrum," https://www.scrum.org/Resources, 2016, last checked 17th April 2016.

The website for the scrum methodology principles. The website was used to reference the process and methodology which was adapted in the project

[16] T. J. Smoker, C. E. Murphy, and A. K. Rockwell, "Comparing Memory for Handwriting versus Typing," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 53, no. 22, pp. 1744–1747, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1177/154193120905302218

Used to show that there handwriting is still an important part of memory rentention with note taking, compared to digital text

[17] M. G. Software, "Planning Poker: Agile Estimating Made Easy," https://www.mountaingoatsoftware.com/tools/planning-poker, 2016, last checked 17th April 2016.

Showing the use of planning poker with exactly how it was implemented in the application using the scrum based approach.

[18] Tesseract, "Tesseract Open Source OCR Engine," https://github.com/tesseract-ocr/tesseract, 2016, last checked 17th April 2016.

The open source optical character recognition engine which will be used in the application to analyse characters on a page.

[19] Tiaga, "Taiga.io," https://taiga.io/, 2016, last checked TODO.

The project management toold which was utilised to help to keep track of the project's progress throughout the process. Utilised the Scrum tools available that the application gives.

[20] M. Webster, "Taxonomy — Definition of Taxonomy by Merriam-Webster," http://www.merriam-webster.com/dictionary/taxonomy, 2016, last checked 17th April 2016.

A definition of exactly what a teaxonomy is. Clearly labelling it as a classification of a problem.

[21] D. Wells, "CRC Cards," http://www.extremeprogramming.org/rules/crccards.html, 1999, last checked 17th April 2016.

A description of what CRC cards are and why they're useful when considering the design of an application. Used as a reference material throughout the process, as well as during the chapter discussing the process.