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. [39] 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 [47].

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

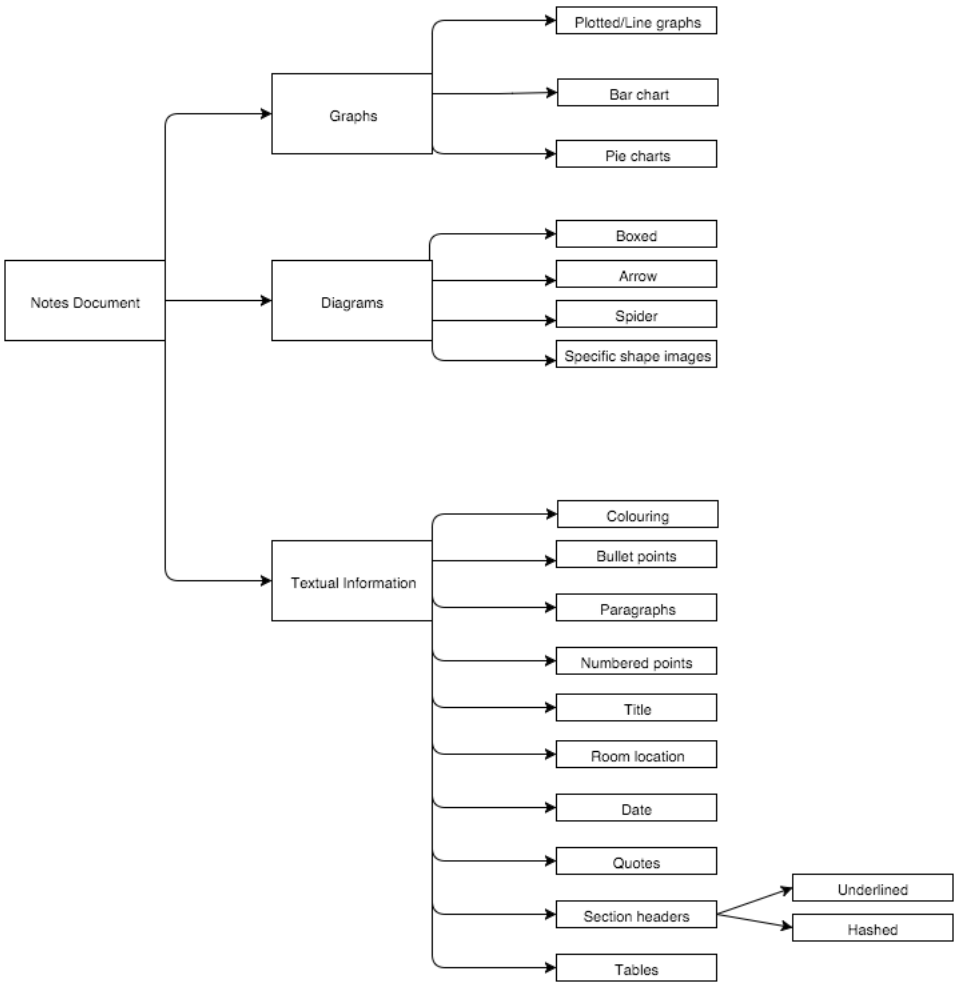

Figure 1.1: A taxonomy showing the structure and classification of different types of notes and what is contained in a note.

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 [11]. Both congruent and incongruent coloured text helped to increase memory retention of post-graduate learners [31]. 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 [23]. 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, showing that handwriting recognition is still very much an active research area, where solutions are still being developed to optimise the correct classifications of text.

Another approach is to analyse handwriting via an OCR (optical character recognition) tool. Rakshit et al. [37] used the open-source tool, Tesseract [42] 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. As 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 recognition rate of 83.5% experienced by Rakshit et al., it is a viable option to use for handwriting recognition.

1.1.2.1 Tesseract operations

Tesseract was initially developed by HP, but has now been made into an open-source tool. Ray Smith gives an excellent overview of the Tesseract engine, the following section summaries how Tesseract works. Tesseract uses connected components to identify the outline of characters, these are then collated into blobs. These blobs are then collected into text lines, which are deconstructed into individual words.

The process then goes through a two stage process, of identifying the words first - and the

second identifies words which are not well known. Tesseract has the ability to identify textlines from skewed images. Therefore, as long as the image is text, a slight skewing of the image would not affect the ability to identify text. Textlines are found from the blobs, filtered and then sorted aided for tracking. Blobs are then processed in order checking for overlapping coordinates - from which they're either added to a new line, or appended to an existing line.

Smith discusses that the words are then split into characters. Due to handwritten text being varied, as a user would not write uniformly, then chopping is complex.

After the word has been chopped Tesseract needs to segment the character to identify the character. Smith describes that chopping may leave parts of the characters unattached, so an A* algorithm is used to compare different fragments from a graph of chopped characters. These broken characters are then checked against previously trained examples and similar patterns are attempted to be extracted.

The classification is described by Smith as a "two step process". Firstly, the character is evaluated and matched against a potential list of characters from previous examples. The second step involves calculating how well that character matches those in the list, the highest match is then selected as the character.

It is important to acknowledge that Smith discusses the implementations of Tesseract as a whole, with results comparable for printed text, not handwritten text - where there is more variation. However, the paper gives a detailed explanation of the complex process of how Tesseract analyses the image, as well as the text.

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 [25] 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 [27]. 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 [26] 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 [12] 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 [3] stating that OCR recognition on images is possible. This would allow the user to upload an image outputting a list of potential words for each word found in the image. 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 [18] 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 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.

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 implausible to be completed within the timeframe.

1.2.5 OCR for the authors handwriting

After research into OCR technologies, such as Tesseract [42], 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 a web application version of their system. 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 of 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, [34] 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 [38] 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 valued 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 [40] 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.

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

At the end of a sprint a review and retrospective was conducted in the form of a blog post [20], 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 20 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 [44] which was invaluable, and provided built in functionality such as burndown charts per sprint. This shows how well story points are being completed, in the form of velocity, 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 [5] 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 [24]. This was implemented throughout the project, with both unit and acceptance 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 [48] 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, over a series of sprints, class diagrams and design diagrams were not created at the very start of the project. Over a series of sprints designs were iteratively developed regarding the system overview. However, some design decisions were decided at the start of the project. The chapter will clearly explain rationale for the decisions and state whether the design decisions were a result of iterative processes or upfront design.

2.1 Overall Architecture

This section discusses the architecture of the web application as a whole. The process of the design with the web application was developed over a series of sprints, rather than an upfront design.

2.1.1 Class Diagram

Presented is an overview of the class diagram presented, with rationale for the decisions and how an iterative development was reached to conclude the design.

Refer to appendix **??**, section **??** for the class diagram.

The design clearly shows the object oriented principle of low coupling high cohesion.

2.1.1.1 Justification of design

The following section discussing the appropriateness of the design and justification, as well as exploring the evolutionary design.

Google Services During the first iterations, the Google Calendar API was only going to be used to parse the calendar events. Therefore, the class `GoogleCalendarService` was created, to extract the code away from the controllers. The version number and API was unlikely to change, but they were formed as constants for the ease of changing if in the future they did. This class needed to perform key operations to extract the events, such as `get_events_based_on_date`. The functions themselves were iteratively developed, initially only using the `execute_request` and `get_list_of_events`. Due to the scope changing with complexity, in the latter sprints

further methods were added.

Initially user's were not a part of the system. However, after implementing the associated user story it was clear that another class to integrate with the Google Plus API would be needed. This followed the similar structure as the calendar class, except for parsing a user's email to persist in the database.

Eventually, it was seen that this design was repeating functionality in places. In both of the classes the `build` and `execute_request` were duplicated, without having class specific content. As a result, the extract class refactoring technique was used to create a super class `BaseGoogleService`. This class encapsulates the logic for building queries and executing them. With both the google plus and calendar classes sub-classing this, it extracts the final aspect of querying to the super class leaving logic and query manipulation to the sub-classes.

Helper classes Helper classes, in the design, are independent classes that help to modularise the system - whilst collecting related functionality into a single class. Helper classes were developed iteratively, encapsulating similar functionality into one place.

For example the `SessionHelper` class was developed in such a way, that during the first few iterations of design it was only used to ensure that the credentials were appended correctly to the session. This functionality was duplicated, so an additional class was extracted. Over the course of adding user management and storing errors the method's grew.

As noticed, most of the helper classes do not interact with the other classes in any class. There is an exception with the `GoogleServicesHelper` class. In this class majority of the function are static. They're static because they do not interact with any class variables. Prior to the implementation of editing a calendar event, the code was dispersed throughout the controllers. In an effort to reduce the code this helper class was created - but it was quickly decided that it would just be a proxy for calling specific function in each of the services classes. Therefore the functions were converted to static as true OO principles were not needed with these.

Persistence classes The relationships between the persistence classes are not discussed in this section, see section **??**. It is worth noting that designing the persistence classes was again an iterative process through the sprints. For example, for majority of the sprints the title attribute in the `NoteMetaData` class was not added. It wasn't until the design changed and this field needed to be added a reflection was added.

There are a series of `save` functions in the application, these were added to the design when the controllers were constantly adding it to the database session. There needed to be a succinct way to save that specific instance. Therefore, database session code was extracted to this method, allowing each instance object to be saved.

In parts of the application, there needed to be a way to extract information from the database. To aid in readability, static methods such as `find_meta_data` was created to keep domain related functionality together, but without having the need to create a specific instance.

Binarisation The `BinariseImage` class is the model version of the image segmentation script see section **??**. The class would interact with the controllers and be called when a user uploads their file. It was important to have this as a model, due to the core functionality it offers to the system.

2.1.2 CRC cards

To aid with the design, class collaboration cards (CRC) was drawn up for each feature. The user-story was decomposed into tasks and each of the tasks had associated CRC cards. This helped to think about the design. The overall design shown in section 2.1.1 is a result of the diligent planning with the CRC cards.

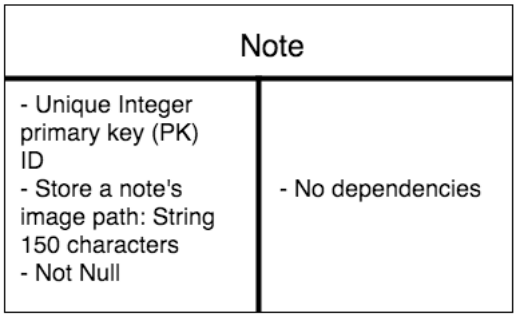

Figure 2.1: An example from Sprint 3, showing a CRC card at the very beginning of creation.

Figure 2.1 shows an example of a CRC card at the very beginning of the note class creation. The left hand side helped to think about methods and attributes for the class. The right hand side shows the responsibilities, where the note may interact with other classes.

These classes were evaluated and considered throughout each feature from a user-story. There were times during the design that these helped to keep a simple design, adopting YAGNI. For example, the note has an image path attribute, this was going to be extracted into its own relation, but after evaluating at the time a one-to-one relationship would not add any benefits to a system. Therefore the CRC card enabled a concise clear design, which allowed reflection and evaluation.

Appendix **??** section **??** shows a range of CRC cards used iteratively on the application.

2.1.3 User interaction

After decomposing the problem that a user would need to be able to add a note, edit and save to a calendar. A activity diagram was thought about to consider the flow of the application.

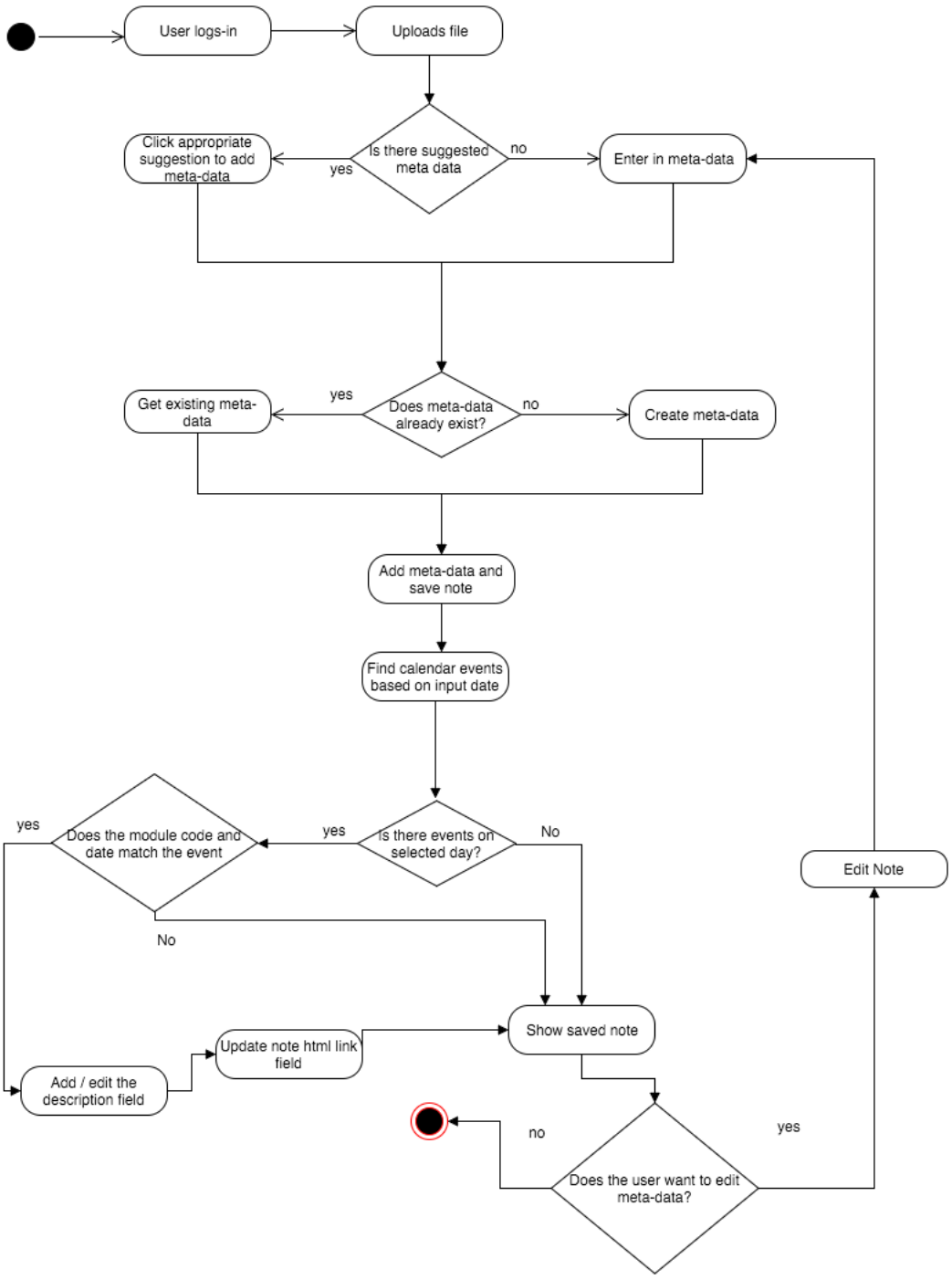

Figure 2.2: An activity diagram to show how to save a note and the integrations with the calendar too.

Over the series of sprints, the design for the activity diagram expanded and grew. The resultant is depicted in Figure 2.2. To begin with the user activity where the user logs in was not incorporated until the thought of expanding the application for further users was considered.

Additionally the conditional check to see if there was meta-data suggested, from which the user could click on the meta-data to auto-populate the field content, has been developed into the

final design. Instead, of the conditional check just an activity was selected to enter in the meta-data.

Overall, the activity diagram shown shows the final output of how a note is added into the system. This design has been meticulously developed incrementally to show the final output. Each of the iterations would increase the functionality, and as a result show the final output.

2.1.4 Model-view-controller

The application would be designed in an model-view-controller (MVC) approach, so it would be beneficial to show the designs which would show how the system interacted with different components.

2.1.4.1 About MVC

MVC is a design pattern where logic is differentiated from presentation layers, as shown in Figure 2.3.

The controllers aim is to not directly integrate with database and business logic, instead it interacts with a series of models and services. Finally, the controllers will help to request to render specific view files with dynamic content.

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.

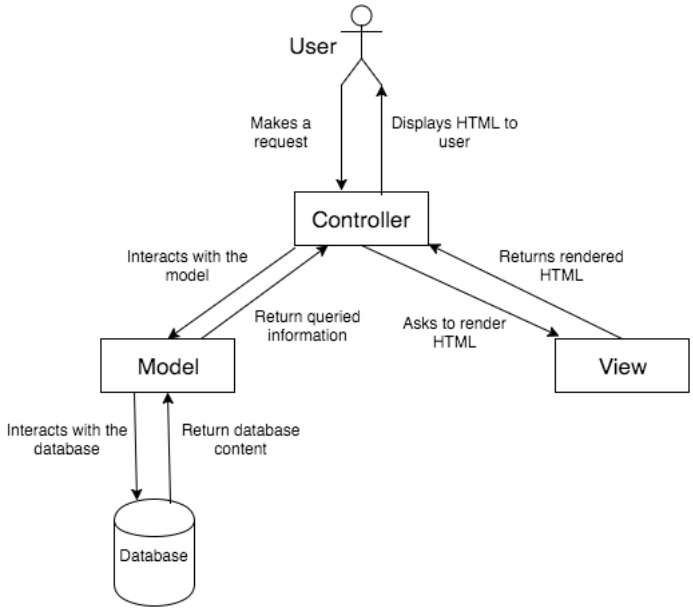

Figure 2.3: A example of how the model-view-controller(MVC) framework integrates.

2.1.4.2 Structuring the web application

Although all the files could not be identified in the design section, the overall structure of the application has been considered.

The primary aim for considering the design of the application would allow the application to reuse aspects of the codebase where appropriate. A module based design was considered, where each section of functionality was its own module - but this was rejected as it felt like the codebase became obfuscated. Therefore the MVC approach was adopted.

The framework chosen, see section **??**, does not support MVC structures out of the box. Routes are expected to be placed in a singular file; this philosophy is carried through to the models. This was not chosen as the structure of the application as it reduces the clarity of what the code was supposed to do by over-complexing where dependencies between different classes are supported.

To overcome this, Blueprints were used. These are essentially controllers, like those found in Ruby on Rails. Annotations were used to define a route and a blueprint was associated to one file.

Models were separated into their own directory, and a one-class per file was adopted to keep the design clean and simple. This ensured the design of the classes was considered before creating such files.

It is worth considering the view files. The view files were the only section of the web application structure which underwent and iterative process. Initially, the view files would represent the entire DOM tree in a singular file (duplicating headers, scripts etc). However, this is not optimal.

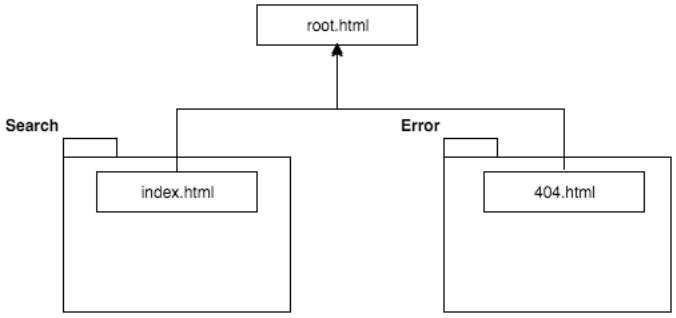

Figure 2.4: A diagram illustrating how extension in Jinga html template engine works.

Figure 2.4 shows the result after the sprint which the design was improved upon. All template files now extend "root.html", overriding the "content" block. This ensures that the do not repeat yourself (DRY) principle is adhered to and HTML, such as the navigation, are only declared once.

2.1.4.3 Constructing URLs

Often overlooked when considering a design is the URL structure. The design not only aids the developer, but the user using the page to clearly know the intention of that page. Typically there are two types of URLs RESTful-like and query strings.

During the iterations, especially when new functionality was being considered, specific routes were thought about carefully. In the search user-story, standard procedure was followed. Query strings create URLs such as: `/search?module_code=cs31310`; representing the query string as key-value pairs. During the search feature, it was decided that this approach would be adopted so that the user can easily bookmark the page - as well as creating a common URL structure.

RESTful urls help to show the a hierarchy of content. Exposing a user to such URL helps them to clearly identify their content. In iteration for displaying a note `/show_note/1`, was decided for the URL; it is easier to read than $/show\_note?note_id = 1$ - instantly showing to the user the point of the page.

For the user task viewing notes, it was worth noting that traditional RESTful URL's would be changed for readability. $/view\_notes/$ was used, when a proper RESTful url may be `/notes/`. This offered more semantic meaning to the URL structure.

It was worth considering the URL structure and way in which the application displays the content to a user, without it a user may be left confused at what the page is trying to convey.

2.2 Image processing

In the very early sprints, the image processing design went through several substantial iterations. Each of the tasks relating to the user story to binarise an image, had design implications.

Early work was conducted to investigate how to prepare images for the Tesseract engine. Imagemagick was initially used by converting the image to grayscale - but this yielded poor results. After further design decisions were made to convert the image to Monochrome it was decided that the next iteration produced a better image segmentation pipeline.

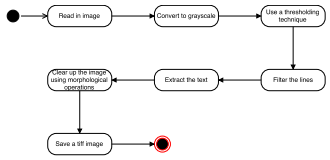

Figure 2.5: An activity diagram to depict the design of the algorithm for the image segmentation.

Figure 2.5 shows the overall activity of how the image processing will be intended to be implemented, after early design work showed binarisation was more complex. Further descriptions of specific implementation can be found in the implementation section.

This high-level activity diagram was the result of continuous design in the first few sprints. Initially a design was drawn up to just binarise the whole image, but due to implementation issues, this caused too much noise. Therefore, a new algorithm had to be established.

The blue-lined paper was one way to overcome that. Filtering the lines from a solid lined paper, would ensure less noise was on the image, creating a better binarised image. Overall, the activity diagram depicts the algorithm of taking a mobile phone photo, filtering the lines, binarising the image and extracting a tiff image. The tiff was selected as a design decision, as Tesseract input requires a tiff file.

This design initially considered blue to be important, but it was produced too much noise in the implementation. As a result, instead of trying to extract the lines, it was decided that the lines should be filtered and we should only extract the text. This was the final iteration of development on the binarisation script.

2.3 Tesseract

Early on in the sprints Tesseract was identified as the OCR tool of choice. Evaluations into the different tools available: [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.

The first few iterations gave a concrete overview that the first three lines of the text would be parsed, forming the information for the meta-data. Appendix **??** shows the layout of the three lines.

It is worth acknowledging that the test-data used for the Tesseract training had a design element attached to it. When considering what the test data should consist of, there had to be a variety in the data. Panagram's, the quick brown fox is the most common, is a good way to represent text as it contains all alphabetical characters. [CITE website used]. This would give Tesseract the best possible chance at learning different characters - due to there being an abundance of each letter.

2.4 Entity-relation design

Whilst creating the CRC cards, it allowed considerations to be made about the relations and how they are connected. Through each user-story analysed it was reflected upon if that would affect the entity-relation design.

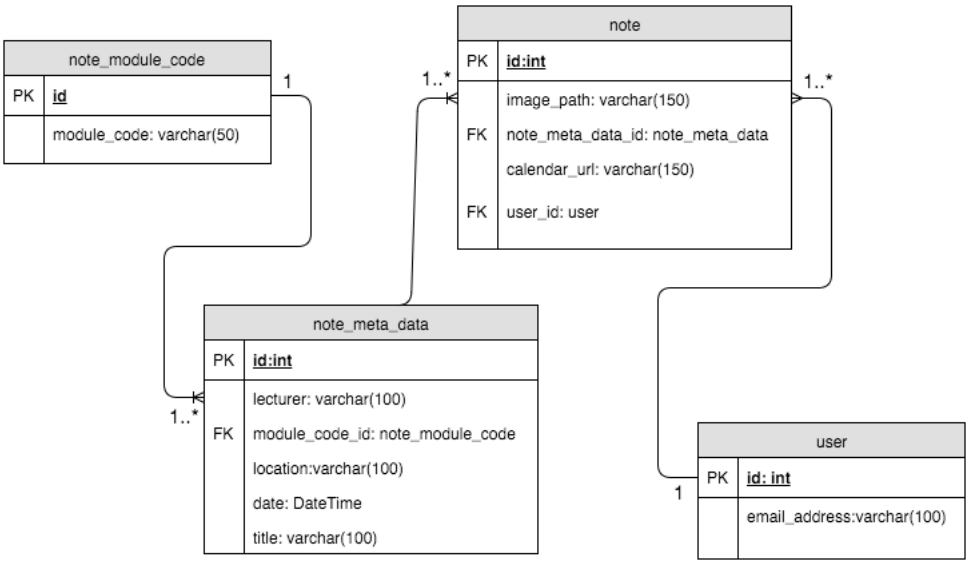

Figure 2.6: The final result of the entity-relation diagram. After a series of iterations.

Figure 2.6, shows the output from the result of continuous design on the database diagram through the iterations. The design per iteration matched that of the story. For example, although Meta-Data was known to be added into the system, this was not added until that story was brought into the sprint - instead just a Note relation was created.

2.4.1 Justification of design

Below is a justification of the designs through the iterations.

2.4.1.1 Note

During design persisting the note was one of the first entity-relation design decisions which was made. The attributes selected for the `Note` relation best justify what a note consists of. Firstly, the note contains an image link, which is a relative path to the image. This was persisted to ensure that it could be found easily. When the story for implementing user's was actioned, an additional field containing the user's id was added to the relation.

During the implementation of adding a URL to a calendar event, the calendar URL was persisted to the database of the associated note. The event ID could have be saved, but the URL was decided to stored so additional queries were not made to the external service. Furthermore, a note will only have one URL.

When implementing the note's meta-data, a relation was created and the foreign key was added to the note relation. This was created to ensure that a note must have associated meta-data.

2.4.1.2 Note_meta_data

The `note_meta_data` relation was created in its own relation to reduce data-redundancy, following the principle of normalisation in relational databases. The content could be duplicated for multiple notes, if a user tags the same meta-data to more than one note. As denoted from the relationships: a note will have a singular meta-data item, but the meta-data item could have many notes.

With attributes lecturer location and datetime - these were the initial design decisions made to be included in the meta-data. However, in a later iteration it was decided a title would be preferable; this was added to the relation. The date field is a date-time instead of a string due to integration with the calendar requires specific date-time strings, making it easier to parse.

Initially developed with the module code in this relation, in subsequent iterations the module code was extracted and a foreign key was used.

2.4.1.3 Module code

The module code was developed into its own relation to prevent data-redundancy. A user may enter multiple notes for the same module code - as a result the database would only need to include one reference of that module code. The relationship between the meta data and the module code is explicit: the meta data must contain one module code but the module code can have more than one meta-data item.

2.4.1.4 User

This was not added to the application until around sprint 5. Details regarding oAuth can be found in **??**. However, the user will have an email address and that would be stored. It is in its own relation from a logic perspective: every time a user signs up to the system they are not creating a note instantly, therefore a relation was created to separate this logic. The foreign key was added to a note, so that a note can only have one user - and a user can have multiple notes.

Overall, a succinct collection of relations have been developed which aim to solve the issues of the data-redundancy, by providing solid rationale for the resulting design.

2.5 Sprints

2.6 User interface

With the web application being a core part a series of user interfaces (UI) was collated at the start of each breakdown of the story - if there was a significant change in the UI.

The user interface had make the web application feel like an application, rather than a traditional website. This was identified from the background analysis where many systems felt like an application. The colour scheme was aiming to be simplistic, using the Google colour style guide. An alternative of bootstrap was considered, instead of designing bespoke CSS. Although it has a built-in responsive theme, due to the over-kill of the additional files a simpler approach was adopted.

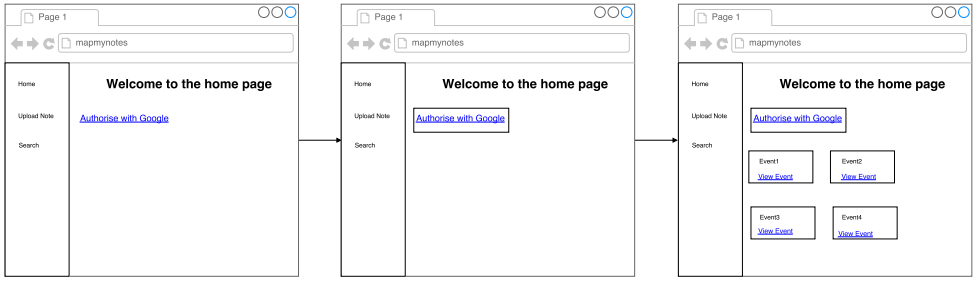

Figure 2.7: From left to right, the homepage wiremock through the different iterations and the change of requirements

Figure 2.6 shows the exploratory wiremock design completed prior to the UI interface. From the early iterations it was just an authorise button, then as a requirement was added to show the user events from the last seven days it was mocked up to reflect this.

Further mockups available in Appendix **??**.

2.7 Implementation tools

The following sections discuss the implementation tools and their purpose within the application.

2.7.1 Programming language

The programming language would not change per sprint or over an iterative process - as a result this was identified in sprint 0, where additional spike work was completed.

As a web application was being developed investigatory work was completed into the analysis of suitable server-side languages. Typically traditional server-side applications language are: PHP, Ruby, Python, C#, Java and JavaScript, which has increased in popularity [46].

Decomposing the analysis in the early sprints determined that OpenCV would be utilised on the project. OpenCV's source code is written in C++, however Python and Java bindings are available. Additional research was conducted to see if a reliable wrapper for either PHP or Ruby was available, and after a lot of investigation it was concluded there was not.

C++ is not considered a standard web application development language removing it as a viable option for the web application. Java applications are predominately large commercial applications, using a range of enterprise software - often renowned for their performance abilities [33]. This approach felt too cumbersome for a proposed light-weight application.

By being constrained by design decisions to use OpenCV and a reluctance to use Java, then

Python was selected as the most suitable language. Python offers a lightweight and an easy to learn syntax that produces readable code, allowing a object-oriented paradigm to be followed. Additionally, its support for OpenCV is sufficient for the application.

2.7.2 Framework

As Python was being used as the language of choice, this narrowed down the frameworks available. Frameworks are useful for handling more complex features like routing and session handling - leaving the developer to focus on more domain specific issues. Exploratory work was completed in the early sprints to find a suitable tool. The frameworks Django [10], Flask [14] and Bottle [6] were evaluated.

Some frameworks constrain the developer's to specific implementations through abstracted classes whereas some offer more flexibility. Whilst evaluating Django, a full framework, it was concluded that such a large framework was excessive for this application and rejected as a choice for the framework.

Flask and Bottle are classified as "micro-frameworks", offering a lightweight structure, allowing developers to have more control on the structure. On face value, Flask and Bottle appear to be very similar; they are both lightweight with a similar syntax. After evaluation both frameworks it was concluded that Flask has a larger support community compared to Bottle - along with more reliable documentation.

As a result, Flask was chosen as the framework which will be used throughout the application. Spike work was completed into evaluating Flask's viability for the application and was decided to be a good choice.

2.7.3 Continuous integration tools

Continuous Integration (CI) is normally used in development teams to ensure that all code is checked into the repository. As it was changed for a single person project, so did the point of using it; it was used to ensure every commit passed all tests when pushing to the repository.

After identifying CI would be used in the analysis stage, an appropriate tool would have to be chosen. Jenkins was an initial choice; it is a standalone Java application which a repository can be synced to.

Travis CI, is a CI tool in the cloud which can be synced to a GitHub repository. Tests can be run during every commit of the application and details regarding if it errors, passes or fails is available.

A key design decision was to be able to check the build process quickly. With Travis it has a web interface clearly showing the output. Jenkins would require additional set up of specific build scripts for every branch in the application. Travis would be easier to set up and would easily integrate with the application.

2.7.4 Version control

Version control was used on the project to ensure that code was under specific versions. The project was created on a private Git repository on GitHub. Git was chosen for its familiarity and GitHub is a well known place for handling Git based solutions; Travis CI integrated well with GitHub.

It is worth making a mention on the Git flow which was used. As each story was implemented a branch would be created in the form of: `feature/<summary_of_story>`, such as `feature/logout`. Branches were created from the development branch. All code was pushed to its own separate branches, and only when Travis CI passed a pull request was created on GitHub. Once Travis had successfully passed all the pull request tests, and it was safe to merge then the branches were merges into development.

2.7.5 Development environment

The text-editor, Atom, was used for the majority of the project. It was a lightweight tool which accommodated for Python syntax. However later in the project, when refactoring became more cumbersome due to the increase in code base - PyCharm community edition was used as it offered better refactoring functionality.

Chapter 3

Implementation

3.1 Image processing

The image processing would prove to be an integral part of the application, enhancing the ability to learn handwriting more effectively and efficiently. The end script was implemented over a series of sprints.

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.

3.1.1.1 OTSU

OTSU, created by Nobuyuki Otsu, is a binarisation technique which essentially converts an image to black and white. Otsu is a global thresholding algorithm, using the whole image as a comparsion. This is unlike local thresholding algorithms where comparisons are made pixel by pixel [30].

Due to notes having non-uniform lighting by the intrusion of shadows on an image, this makes OTSU difficult to reliably binarise the image.

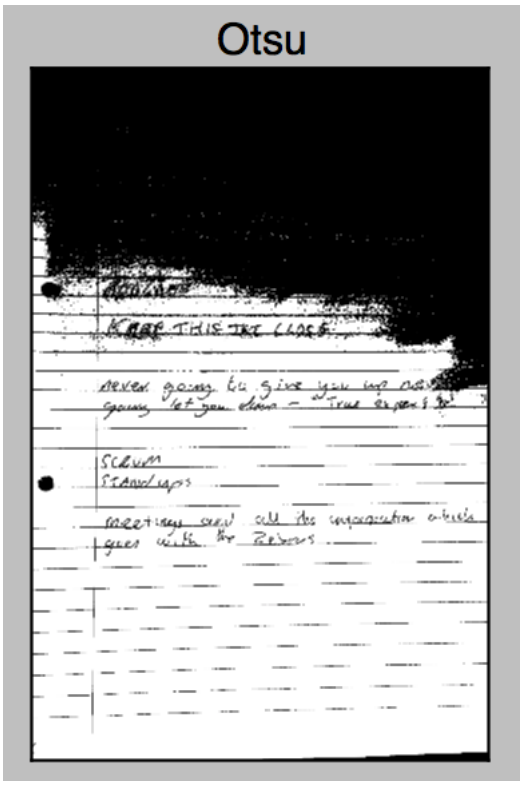

Figure 3.1: The use of OTSU binarisation technique on an image with a little shadow across the image

Figure 3.1.1.1 shows the binarisation technique on an image with the a slight shadow imposed on the image. Clearly it can be seen that it binarises the wrong sections of the image, this would be due to global thresholding.

The basic premise of OTSU is the image's grey-level values are segmented into a series of histograms. OTSU determines the optimial threshold value by "maximising the discriminant measure" [2]. In other words, OTSU attempts to maximise the margin between the histograms. From the maximising of the histograms, pixels can be segmented into their background or foreground pixels [21].

HP, who created Tesseract, describe OTSU as its underlying pre-processing algorithm when it tries to identify characters [41]. From the iterative spike work with OTSU it is clear to see, just from Figure 3.1.1.1, how Tesseract would be unable to clearly identify characters from that image.

Overall, OTSU, although it is a very solid binarisation method, it suffers from imposed shadows over images. This would not be the best option to choose when choosing an appropriate binarisation technique.

3.1.1.2 Adaptive threshold

From the enlighting analysis of OTSU it was realised that using an OTSU thresholding ontop of an OTSU threshold from the Tesseract engine would not be beneficial. Therefore, in the next iteration of the script, an adaptive threshold approach was chosen.

Adaptive threshold will calculate the threshold over a series of smaller segments of the image [13]. This reduces the impact of shadows over an image, and does not consider global illumination as the key.

Using the OpenCV library there was two options [**?**]:

1. Gaussian adaptive threshold which is the weighted sum of the neighborhood
2. Mean adaptive threshold, which is the mean of neighborhood.

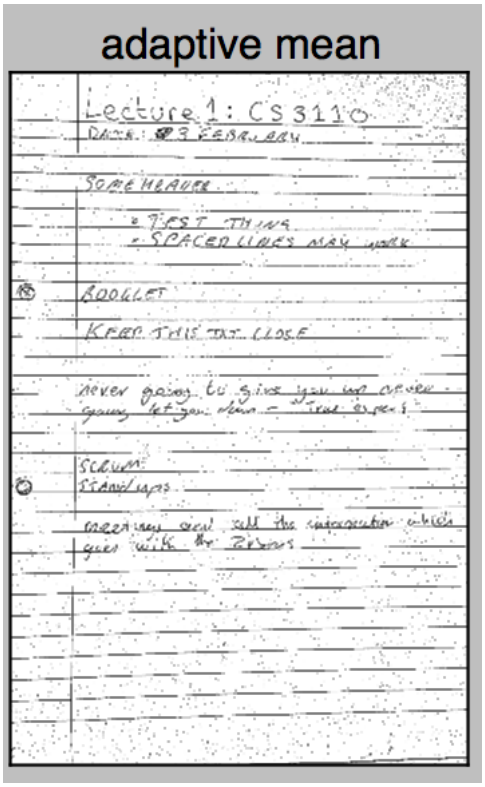

Figure 3.2: Adaptive mean threshold algorithm on a note, showing binarisation but there is still noise in the image.

The mean neighboured takes a block size around the pixel, say 4, and will work out the mean pixel value from that block. The mean value selected will then be selected as the thresholding value, which will determine whether pixels are background or foreground [13]. Figure 3.2 shows a mean adaptive threshold. Due to smoothing issues there is still noise on the image.

The gaussian operation differs from the mean as it uses a gaussian value over the sub-image. Firstly each "blocksize" is a value which surround the pixel. A default gaussian weight is then calculated [ADD calculation] based on the blocksize. For every pixel in the block, multiply it by the gaussian, an average weight is then taken and used as a threshold [7] [**?**].

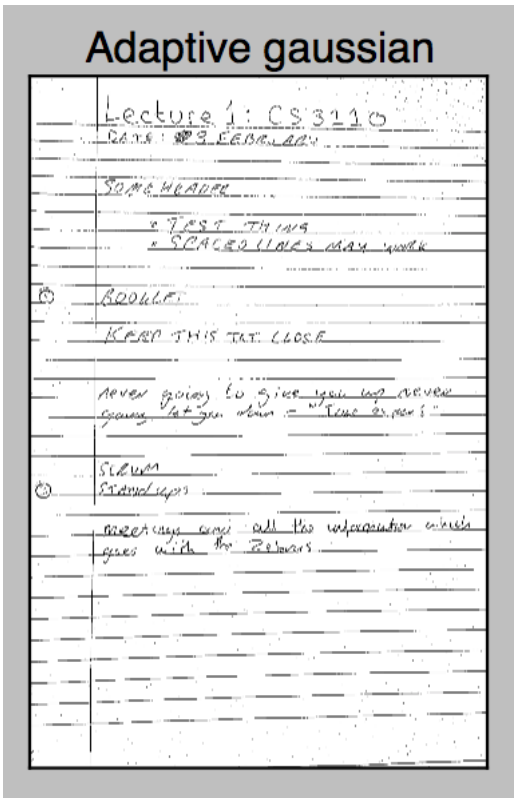

Figure 3.3: Adaptive Gaussian used over the image, showing a lot smoother of an image

Figure 3.3 shows the adaptive gaussian shows an image which is working it's way to binarisation. It does not have a shadow overlaying the image and the text, lines and little noise have been extracted.

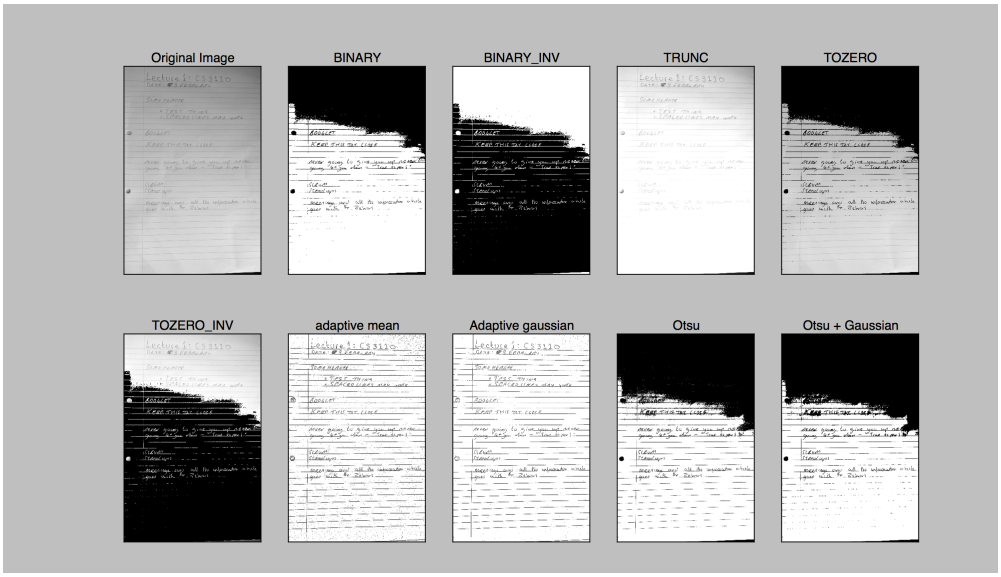

Figure 3.4: A variety of thresholding techniques used on the same note, showing adaptive threshold resulting in the best

Figure **??** displays the other types of thresholding options iteratively tried. It was decided that the gaussian adaptive thesholding would be implemented and improved with different morphological operations.

3.2 Lined paper

Intially standard lined paper was used for the notes, but the noise produced from the lines was too much to ensure reliable readings from, Tesseract. Refer to appendix [**?**] section [**?**].

3.2.1 Filtering the blue lines

Custom lined paper, with equal spacing was constructed to overcome this as discussed in section **??**.

Over a few iterations the main aim was to remove the blue lines from the image.

Algorithm 1 Initial removing the blue lines algorithm

1: **function** ((r)emove_lines)
2: $image \leftarrow read\_image\_as\_grayscale()$
3: $lower\_black \leftarrow np.array([0, 0, 0])$
4: $upper\_black \leftarrow np.array([175, 20, 95])$
5: $mask\_black \leftarrow cv2.inRange(erode, lower\_black, upper\_black)$
6: $mask[np.where(mask\_black == 0)] \leftarrow 255$
7: **end function**

Algorith 2 has its obviousy flaws. It attempts to get the values between a grey-black color range. The blue lined tex would obviously have some black content in it, so not all the lines would be removed.

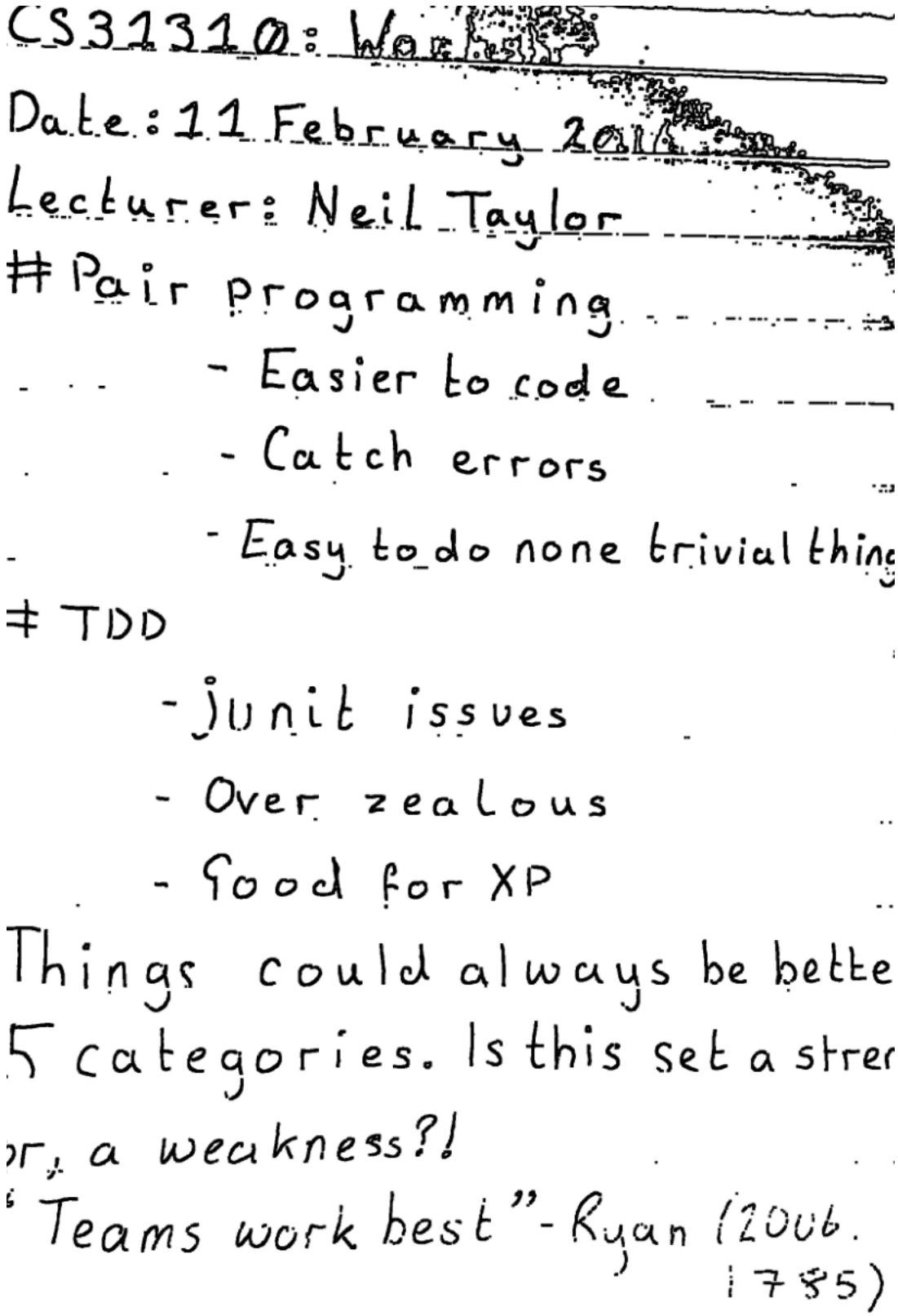

Figure 3.5: An example of the above algorithm. There is still significant amounts of noise in the image.

After morphological operations such as erroding and dilation, the line noise was no longer being removed and insteadit began to affect the quality of the segmentation, as shown in 3.5. Although these early iterations were not perfect, it was on the write tracks.

3.2.2 Only extracting the text

Due to their being no easy way to identify the lines, but ignore them, then it was decided to just extract the text - bypassing the lines.

OpenCV had a very good example for line extraction and binarisation [4]. After binarising the image, using aforementioned implementation, the horizontal lines wre extracted using OpenCV's structuring element `Morph_Rect`.

Morphologial operations, errosion and dilation, were performed to remove the lines and any additional noise. There was a sereis of blank image masks used an intermediatery step to transfer the text from one image to another. The text was transferred over to the blank mask, but line noise was still being transferred.

Connected components via identifying where there were contours on the image was utilised to extract pixels which were connected to one another. Due to different morphological operations, the horizontal lines were not connected completely. As a result the connected components identified all the text in the image - these were then transferred to a new mask.

Finally, errosion was used to remove additional noise and dilation used to fill in and make the characters clearer on the image. Eventually, the binarisation script was complete after a few iterations.

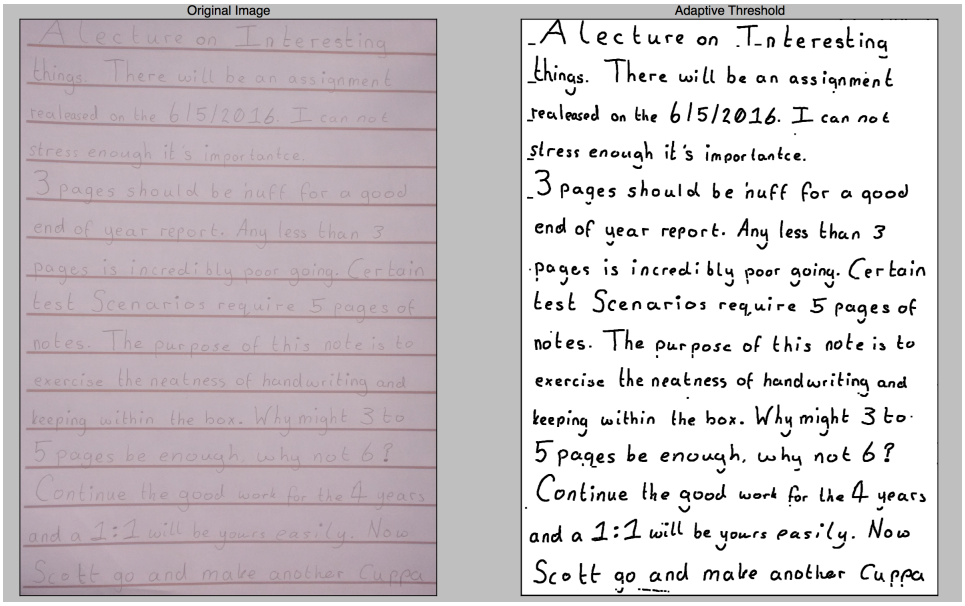

Figure 3.6: A poor quality image has been binarised successfully with little noise.

As shown in Figure 3.6, the image has been successfully binarised. This shows where adaptive threshold works well - the image is a poor quality image, but it is locally thresheld. Eventually the output of the image is a clean binarisation script, with little noise. There were naturally technical

difficulties along the way, and this influenced the implementation decision. Eventually it was agreed the image quality was good enough to stop the binarisation script. Further images of the iterations can be found in Appendix **??**.

3.3 Handwriting training

Initially the handwriting training was not yielding a good success rate. After the changes implemented from section 3.1.1.2 the handwriting training was able to identify more characters successfully.

3.3.1 Training process

Prior to sprint 4, the training process was conducted using greyscale images. As this was not yielding much of a good recognition rate, then the binarisation script was implemented due to the problems faced.

Specific files formats were required during the training process with Tesseract; `<lang>.<font>.exp<num` was the layout required for training.

When training on the test images, a selection can be found in Appendix **??**, Tesseract outputted a box file which contained all the characters on different lines; each lines consists of coordinates for the box file and the associated character. Initially looking at this file it was hard to identify where it was recognising characters.

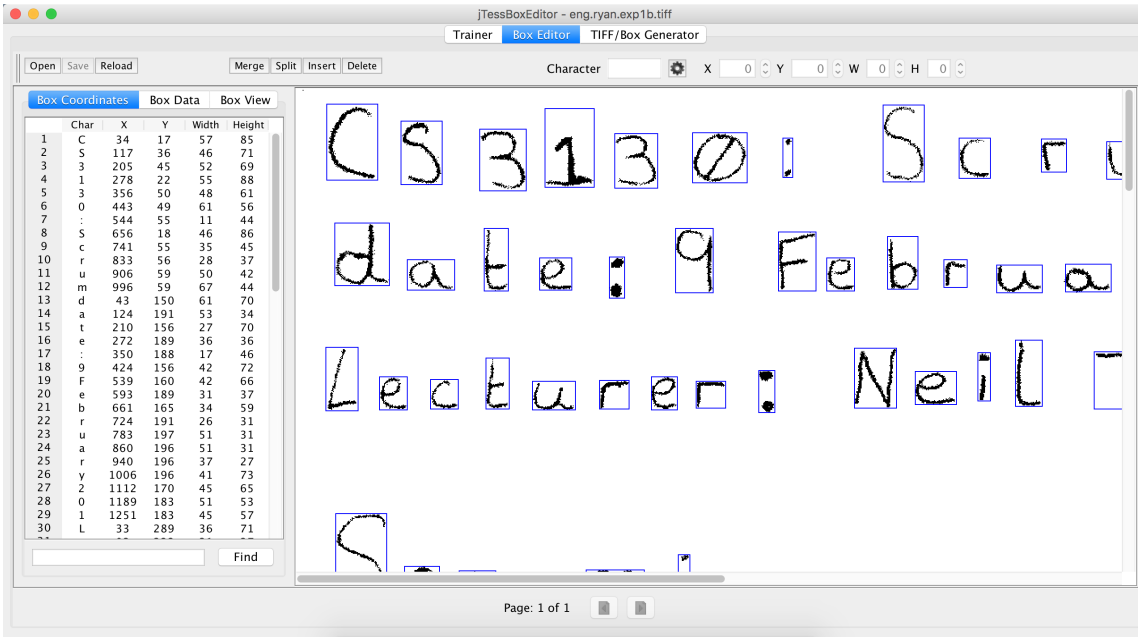

Figure 3.7: A example of the jTessBoxEditor being used identify characters in the tiff box file.

Figure 3.3.1 shows the use of the tool jTessBoxEditor [45] when re-arranging the boxes and

tagging the correct characters.

Issues were identified through the handwriting training section: sometimes characters would not be identified from the image correctly. Often when Tesseract would fail to identify a character correctly it would depict it as “ ”. This would be split and tried to be characterised until Tesseract produced an error saying it failed to identify the characters. After a few iterations of this happening, it was decided that the best option was to remove this line from the box file.

After the tagging of characters had been completed the training process occurred. This process was repeated for each of the 12 training examples.

Following the guide from Github and this guys thesis: When starting the training process, the image was trained on a new language, `eng.ryan.exp2a` along with the command `batch.nochop makebox`. This allows Tesseract to attempt to add box files, around characters which it thinks it has identified from previous examples.

After the identification of the characters was complete, the box file was trained using Tesseracts but in command. `tesseract <file> -l eng.ryan.exp2a box.train`. This process would attempt to train the characters for the given language. At this point there was often some errors where it can not identify the character tagged, it was learned over the itterations it was best to delete these. A trained file for the specific.

After training the characters, Tesseract training requires a few other files to be able to learn from the new examples. To aid in the identification of the possible characters `unicharset_extractor` is run over all the box files. A series of characters which Tesseract can output is collated.

A series of clustering to ensure that characters detected could be identified again after the training process had been completed. Directed Acyclic Word Graph files were created to help to identify words and frequent word files were created to aid in improving Tesseract’s chances of identifying specific characters. `/usr/share/dict/words` was piped into the words file, to ensure it had all the words in the dictionary, and additional common words such as “by” and “date” was added to the frequent words list.

Finally, `combine_data` command was used to combine all the data together and output a trained data file in the form of “eng.ryan.exp2a.trainedddata”. This was then copied to the shared Tesseract data directory resulting in a language being created for the specific user.

It is worth noting that this was done in the process a few times but it was quite lengthy, so a script was created to speed up the process. Additionally, once the language has been created once it needed to be reinforced with additional learning. This was called “bootstrapping”. If the language was specified when learning it would reinforce that langauge and the process could start again.

3.4 Web application

The web application was the core part of the development and certain aspects proved to be more complex than others.

3.4.1 OAuth

The Google OAuth integration was a lot trickier in places than anticipated. Using the Google client library, to handle the Oauth connection reliably, was imperitive, but there was side effects from using this.

There were occassions which the Google API client raised peculiar errors; during one query it would work and then in the next query a "rootURL" error would be thrown. An issue was raised on the GitHub repository [19]. What made this issue more confusing was that it would work for querying the people plus API for the user's email address, but would constantly fail with the calendar API requests. The problem was potentially a caching issue with the application - although there was no fix for this.

During the exhanging of tokens the credentials were appended to a secure session; every page request which needed to use the credentials would select them from the session. Inside the credentials there is an expiration token, as shown in Appendix **??**. If this token was used to make a query then the user would be dealt with an error message, informing them they have an outdated token. As the user did not need to know about this, checks were added to ensure that the token is still valid. If it was not then it would log the user out , clearing the session, and ask them to re-authenticate.

3.4.2 Reoccuring events

Reoccurring events were to be a *big* problems. Identified in the pre-beta testing, if a user had a reoccurring event then it would not append the URL to the calendar item. This design decision was not picked upon and therefore was not considered as a possibility.

All-day events were another issue. By the nature of them being all day, they do not have a date-time key in the start response from the Google Calendar API. The application would therefore error when it attempted to access a key that did not exist; this issue was resolved quickly.

The reoccuring events issues was more complex than that. When intially querying for an event, if the event was reoccurring, it would gorup the reoccurring events by their first occasion which the event was created. This proved to be problematic as it would display to the user that the note was taken on the 12th March, for example, but show an event from February.

The grouped event had a reoccurrence ID key, which was used to create another query which would return all the instances of the recocurring event; this was filtered to the start and end date of the initial query. The correct event could now be displayed.

However, when editing an reoccurring event further unexpected behaviour was experienced. After modifications to the reocurring event have been performed, and a query has been performed for all the events again, it returns both a grouped event and the instance of the edited event. So in theory it must create a new instance of an event that is reoccurring. A succinct solution was not found and instead, when parsing the events, those with the key `recurringEventId` were omitted.

3.4.3 Tesseract confidence

The Tesseract confidence went through a few iterations during the latter sprints, to achieve a better user experience.

Firstly, before any pre-processing could be implemented, the binarisation script needed to be included into the web application. It was not included until around sprint 8, where it was added as a series of function calls to the upload route. Afterwards, Tesseract was integrated into the application using tesserocr library [1]. A wrapper could have been written for the Tesseract's C++ API's, but due to time constraints it would not be beneficial. Figure **??** shows Tesseract originally implemented into the system.

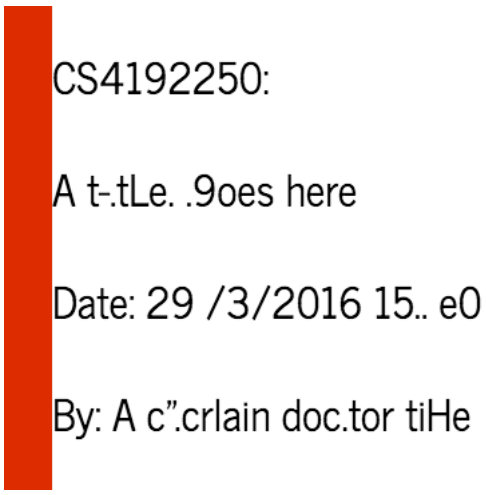

Figure 3.8: Tesseract being integrated into the application at a very basic level

To extract the first three lines of text from an image, the tessocr library uses textlines (an underlying feature of Tesseract), to extract all the lines which contain text in the image. Each of these lines was iterated over and all words were maped to their overall confidence.

Lines are then extracted via list-comphrension and slicing for each of the different words. A technique was tried to modify the confidence scores in the controllers, to replace the integers with the specific classname for the colours. However due to the API returning tuples, which is an immuable object, modifications could not be made easily. The eloquent envisiged would be more work than necessarily.

In the view file, checks were made on the tuple; a confidence of 75 would equate to green, less than 75 but greater than 70 would be orange and below 65 would be red. Figure 3.4.3, shows the final coloured output. It's worth noting the anomoloy with "Date:", it labels it orange but it is perfectly extracted.

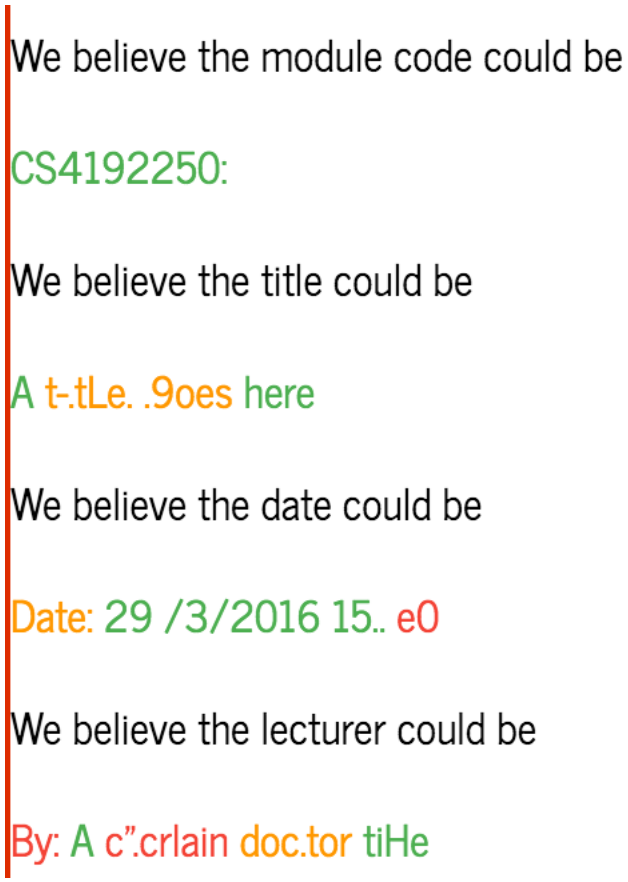

Figure 3.9: Coloured representation of the confidence of the words from the handwriting.

3.4.4 Parsing exif data

One aspect of the application which was important was the parsing of exif data. Without the consideration of exif data, the initial suggestion text - which shows all the events based on a note upload - would not have been achieved.

Through the sprints, the exif data was refined and improved to allow for a greater deal of variation. Initially, Python's image library was used to get the exif data items; checks were conducted to ensure that the images were either JPEG or Tiff.

Issues arose during user-testing, when one of the participants could not upload their note probably because there was an issue with parsing EXIF data. Their mobile phone did not contain the EXIF "dateTime" key needed to be parsed; the key way checked to exist and the user could upload their image.

3.4.5 Displaying calendar events

Displaying the various events was adopted over several user-stories. It was first suggested by the supervisor that when a user authorises with Google it displays the last seven days on the homepage. This was fairly simple to implement, querying the Google Calendar API. What is worthy to note

is that this was imperitive for integrating the other Google Services into the application - not only was it the implementation structure, but also the testing infrastructure behind it.

After parsing the exif data a story was raised to find all events based on any times found. This involved querying the external API for the events - this task was one of the first which testing began to be problematic, with mutliple requests to same service being conducted. Eventually the date was formatted from the exif time start until the end of that day at "23:59".

3.4.6 Editing calendar events

As discussed in the design the adding a note the user must be able to add the note URL into their calendar event. Implementing this threw a few issues. Firstly, the date was parsed by the user's input - this was then used as the query to return associated events from the Google Calendar API. Checks were then performed to ensure that the module code and the summary of the event matched.

This created the problem of being able to add to the note's URL to the correct event. If there was more than one event with the same module code that day then there could be a confusion as to which one to add to. The next itteration of the calendar events ensured that checks were made to event to validate the correct day.

An issue which was discovered was that when adding to the description field it would just replace the original content inside the description field. This is obviously a major concern as it would not allow for multiple notes per events, and it would over-write any data the user had in their description. This was changed so that it would append to the description field.

The algorithm for additng to a calendar is stated below.

Algorithm 2 Adding a note URL to the calendar

1: $Createacalendarserviceobject$
2: $Prepareurlfromsavednote$
3: $BuildtheHTTPrequest$
4: $Findaneventforthatdayfromthestarttimeasgiven$
5: $Parsetheevents$
6: $ChecktoseeifthesummarycontainsmodulecodeANDthestartdatetimematches$
7: **if** $containsmodulecode$ **then**
8: $\quad checktheresponsetoseeifdescriptionincludesurl$
9: $\quad SaveURLtothenotesattributeinthedatabase$
10: **end if**

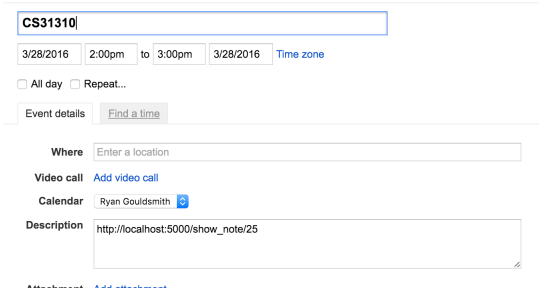

Figure 3.10: Saving a note correctly to a calendar event item

Figure 3.4.6 shows the output from saving a note to a specific date and it adds it to the correct calendar item.

3.4.6.1 Editing a note

During the interations it was established that when editing a note, it should reflect the changes in the user's calendar; if a user changes a date, the calendar should be updated. This design decision would prove to be a complicated implementation.

When the user edits their note, it would query the API to return the event, based on the time start which was persisted for the note. The event was intially updated in such a way it replaced the description with an empty string. This causes the problem, the same as add, where it would replace all the content. Instead, a find-and-replace on the description field, to remove the URL easily. The add was performed on the new date.

A note worth mentioning is at this point, with the editing a note, a lot of the code-base was refactored. The application was duplicating a lot of the functionality and the codebase became obfuscated. This was where the refactoring into the `GoogleServicesHelper` class was created from.

3.4.6.2 Logging in and out

Although not an original analysis decision it was decided in the design that a user's table would be implemented. When the user clicks the authorise with Google button, it uses the oAuth connection, afforementioned, and connects to the Google+ API. When connecting with the API it extracts the meta-data from the user and parses the email address, and adds it to the database.

During the sprints an issue which arose was that multiple user's were being created for a singular email address. Therefore, to reduce the problem a helper function was implemented to find existing users based on the email address. Using the SQLALchemy API's this was simplistic.

Afterwards, the user's id is appended to the user's session. This is then checked and required for every page which the user visits. When the user had finished their session, the logout route destroys everything which is in the user' sessions - forcing them to log back in again.

The decision not to use a custom-built log-in was worth it. As OAuth was being used, it was better to delegate the responsibility to a well tested system. Additionally, ethically, only an email address is being stored regarding personal information.

3.5 Travis

Although not strightly a coding implementation, Travis formed a core part of the application and was built around its successes. There were a few problems running throughout the duration of the process.

Firstly, at the start of the process time was invested to try and get the auto-deployment from Travis working correctly. Travis is able to deploy to PaaS (platform as a service) tools such as Heroku, however when trying to deploy to a custom VPS (virtual private server) issues arose. For many different attempts were conducted over several sprints to get the pipeline set up, but to no avail. Alas, Travis was used for better functionality.

Another issue, which arose when Tesseract was brought forward as a user-story, was trying to build Tesseract from source. TesserOCR, uses Tesseract 3.04 - but the latest stable package on Ubuntu is 3.02 (at the time of writing). Not only this, but Leptonia, had to be compiled from source due to Tesseract 3.04 requiring Leptonia 1.71 - and Ubuntu does not have these packages.

Additionally, this increased the build time exponentially, with building OpenCV from source as well. It now stands at around 30 minutes; caching was looked into but no solution could be found. This remains an outstanding issues.

Chapter 4

Testing

This chapter discusses the testing strategy which has been implemented on the project. This includes unit, acceptance and user testing utilised throughout various parts of the application.

4.1 Overall approach to testing

To recap, an agile approach was adopted throughout the project. From the process, test-driven-development was used throughout the application for almost all aspects of testing.

4.1.1 Test-driven-development

Test-driven-development (TDD) aims to produce tests prior to the implementation of features. As a result, all implementation code is supported by a series of tests. Figure 4.1 shows the TDD cycle.

Initially a test is created, this would then fail due to no implementation code. 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 possible for the current implementation.

With TDD it could have gone one of two ways: a group of tests were created for a feature, or

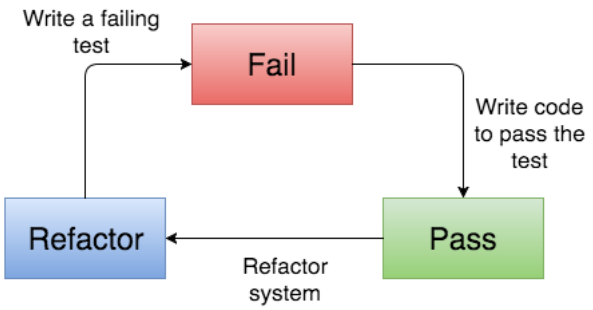

Figure 4.1: The cycle of TDD during the development stages of the application

one test for one bit of functionality. The latter approach was chosen, ensuring that the design was carefully considered.

Using TDD allows for the domain of the problem to be considered before implementing code to solve the issue. The user-stories were deconstructed into a series of tasks. There tasks were then formulated into different tests (unit, integration and acceptance). The cycle was then repeated.

4.2 Automated testing

One thing to note is that Flask's testing documentation is very sparse and is of low quality.

During the first few sprints of testing, pytest [35] was originally being used to create test classes, with test classes sub-classing `unittest.TestCase`.

Flask tests were refactored mid-way through a series of sprints to use Flask-testing [22]. This offers better testing support for Flask application, allowing the creation of a fake application and providing the functionality to run a live server for testing, which would be needed for acceptance tests.

4.3 Mocking tests

The purpose of mocking is to change the output of a function to a value which is returned every time [17]. It was established that certain tests would need to be mocked, because some data would change from test to test. It was identified that *all* interactions with Google API's, any interactions with Tesseract and the Session would need to be mocked.

As discussed by Mathew Dale [8] when writing tests, mocks are needed because of external factors out of a developer's control. As the Google API does not support specific environment API's, such as production or development, then all URLs would be to a production URL. The issue this raises when testing is ensuring that the tests are isolated and pass every time. For example, the test could query the API once and pass the test, however on the next query it may fail due to a different return value; this requires mocks to be used. The mock would return a specific value every time, ensuring consistency among tests.

The principle is the same for testing Tesseract in the web application. If more training was conducted then the results from the test on the image would change, therefore the data was mocked from the returned text functions to provide consistency.

During the first few sprints, whilst understanding how mocks work with Python, there was a lot of duplication with the mocking services. The library mock [16] uses annotations above test functions to signal a mock value.

Listing 4.1: An example of using mocks, following the annotation pattern

```
@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
```

In code example 4.1 shows the syntax which was initially used in the tests, for mocking. This would result in many of the tests becoming unreadable obfuscated. Additionally the do not repeat yourself (DRY) principle was violated, by duplicating much of the codebase.

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


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

Investigating the mock API documentation, patching object calls was discovered. Implementing this solution reduced the amount of code for mocking specific functions. The annotations were removed from the top of function tests. Initially it was not entirely clear how to implement these patch functions. Eventually the patch was included in the `setUp` and `tearDown` functions, as shown in example **??**.

Often when testing there needed to be way to alternate the output of a function. In the Python mock library side effects allowed the mock object to return different values.

```
def setUp(self):
    # some code
    self.google_patch = mock.patch.object(
        GoogleCalendarService, "execute_request")
    self.google_mock = self.google_patch.start()
    self.google_mock.side_effect = [self.google_response, self
        .new_event, self.google_response, self.updated_response
        ]

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

Figure 4.3 shows the use of the "side effect" API. From the above example, it outputs Google response first, then when `execute_request` is called for a second time `new_event` response is fired and so on. Examples of mocking data can be found in Appendix **??**. An example usage of this would be for when the Google Calendar needed to get a user's events. Firstly, in the controller it would get a list of events and then it would get a singular event. Since the same function call was used, a test needed to be added to check the return values were indeed correct.

Overall, mocking produced a substantial amount of the testing. It was not considered when designing the system that mocking would be needed, it was only when testing happened that it was realised it was needed. The tests were refactored to incorporate the mocking facilities.

4.3.1 Unit testing

A unit test is used for models where functions can be tested in isolation to ensure that they perform the correct operations. In the application unit tests were created for every model.

Sometimes there was a cross over between database transactions and testing specific functions. In these cases, using the actual database would not be used for testing as this would interfere with the actual data in the system.

The config was overwritten to include a test SQLite database. This ensured that a test database was used, which was cleared and created for every test. This allowed each test to be tested independently of one another.

Very much like the function names of the classes, the test cases were as descriptive as possible. This formed part of the documentation of the application.

```
test_user.py::TestUser::test_creating_a_new_user_should_return_1_as_id PASSED
test_user.py::TestUser::test_creating_a_user_should_return_the_correct_email PASSED
test_user.py::TestUser::test_find_a_user_by_email_address_should_return_user PASSED
test_user.py::TestUser::test_finding_a_user_by_email_address_which_doesnt_exist PASSED
test_user.py::TestUser::test_user_function_to_save_a_user_successfull PASSED

========================================================================== 5 passed in 1.83 seconds ===
```

Figure 4.2: Example Unit test for the user class. Each of the tests pass

Figure 4.2 shows an example of the unit tests for the user class. The functions were decomposed and tests were associated for each function. In cases both edge-cases and normal expected results were selected.

Refer to appendix x for full unit test cases.

4.3.2 Integration testing

Due to the application having external routes, then testing them to ensure the correct response codes was important. These were the first tests written for the routing sections, and implemented from the design considerations in section **??**.

The tests consisted of checking that the response codes were correct, any redirects were correctly redirected; this tested to ensure the application flow was implemented correctly.

Where there was interactions with the database in the routes, then tests were conducted to ensure the routes performed the correct persistence. For example, when posting to the meta-data URL, then checks were created to ensure that the module code was being persisted correctly.

Appendix **??** section **??** depicts the integration tests in the system.

4.3.3 Handling sessions

One of the trickier aspects of testing was the handling of sessions. In parts of the application sessions are used to handle specific states of the system, i.e when a user's logged in.

During testing, sections of the system are tested in isolated environments. If the route to show all the notes was tested, it would require the user to be logged in - therefore storing data in the session. However, when testing this there will be no session set when running the test. Flask had a solution to testing the session handling [15].

Listing 4.2: An example of how sessions were handled and modified in the tests.

```
with self.client.session_transaction() as session:
```

```
session['user_id'] = self.user_id
```

Figure 4.2, displays an example on how the session had to be modified in the integration tests. After the session transaction context has exited then the session has been modified for that test.

Acceptance testing initially proved to be problematic. When testing with the sessions, the selenium tests would not acknowledge that a session had been set, like in Figure 4.2, causing the tests to fail. After a lot of problems, it was confirmed that the session helper would have to be mocked, in the `create_app` function, which is initialised when a test is created.

Overall, session handling was one of the hardest aspects with testing to overcome to provide working tests.

4.4 Acceptance testing

Acceptance tests evaluate the system as a whole. This includes testing to ensure that the front-end functionality is what is expected when the user views the web-page.

To implement these tests a tool would be needed to check what the application looked like on a web-page, the tool selected was selenium for Python [29]. It can integrate with the DOM (document object model), and perform automated browser testing.

When testing with selenium there is the option to test with a standard browser, Chrome, Firefox etc, or it can be tested on a headless browser (via PhantomJS). A headless browser can access a webpage the same as Chrome, but it does not display a graphical user-interface [36].

An example of an acceptance test may follow a pattern like:

1. Go to page /search.
2. Find the search field.
3. Enter the text “CS31310”
4. Click submit
5. Find “searched-item” from the DOM.
6. Return whether it equals “CS31310”

The above example shows that unlike other tests, which test the specific feature or functionality, this test checks what is displayed to the user via interactions with the web browser. The test then passes or fails based on assertions. Another example of the selenium tests being useful was checking the output colour from Tesseract. The logic in the view file determined the colour and Selenium was able to confirm the logic was correct.

In Figure 4.3 it shows that the time to run to run 5 tests increased to 16.09 seconds; one of the disadvantages is the time taken to run the test-suite. Due to the complexity with loading data correctly to the view file, then these tests are imperative to ensure the user expects to see the correct content. For full acceptance tests refer to Appendix **??**, section **??**.

With acceptance tests the classes all subclass the `LiveServerClass` from the Flask-Testing package. This is different than the Integration and Unit tests as they use `TestClass`. The

```
tests/test_acceptance_edit_meta_data.py::TestAcceptanceEditMetaData::test_edit_form_is_displayed_on_the_page PASSED
tests/test_acceptance_edit_meta_data.py::TestAcceptanceEditMetaData::test_edit_form_populates_existing_information_correctly PASSED
tests/test_acceptance_edit_meta_data.py::TestAcceptanceEditMetaData::test_ensure_the_fields_have_required_key PASSED
tests/test_acceptance_edit_meta_data.py::TestAcceptanceEditMetaData::test_when_editing_the_date_it_shows_unable_to_save_to_calendar_if_no_event_was_found PASSED
tests/test_acceptance_edit_meta_data.py::TestAcceptanceEditMetaData::test_when_editing_the_date_updates_event_link_should_be_new_html PASSED

============================================================================== 5 passed in 16.09 seconds ===============================================================================
```

Figure 4.3: An example of the acceptance tests running. It shows that the time to run the tests have increased considerably.

LiveServerClass allows an instance of the application to be loaded alongside selenium, removing the need for an external selenium server or for the tests to be ran with the server running. It was at this time in the sprints which a large refactor of the system occurred, accommodating for the automatic application launch.

4.5 User Testing

Due to the application aiming to solve a problem, a set of potential user's were asked to perform a user study of the application. Their responses were analysed and their opinions on whether the software met their aims was collated.

Prior to the actual scheduled user-testing, feedback was given regarding the displaying of the Tesseract output confidence. These "over the shoulder" comments were along the lines of: "It would be great if you could click the identified text and it would automatically populate the text boxes". This was then implemented as a result from pre-user testing.

Further issues which were identified during the user testing were:

- Uploading a JPG image off a phone, which does not have the correct date-time exif-data key causes the application to fail.
- Uploading an image with a previously uploaded filename caused the application to display the old file.

These issues were caught and modified thanks to extensive user-testing of the application.

Figure 4.4: A pie chart from the Google forms questionnaire that the users conducted showing that they would not use the application for archiving their notes.

One interesting reflection of the user-case study was that people would not use the application, as shown in Figure 4.4. They were quick to defend the applications quality, but the use-case for them taking notes was not present. They much preferred to write up their notes from the lecture for memory retention.

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 learnt as it progressed through the training process.

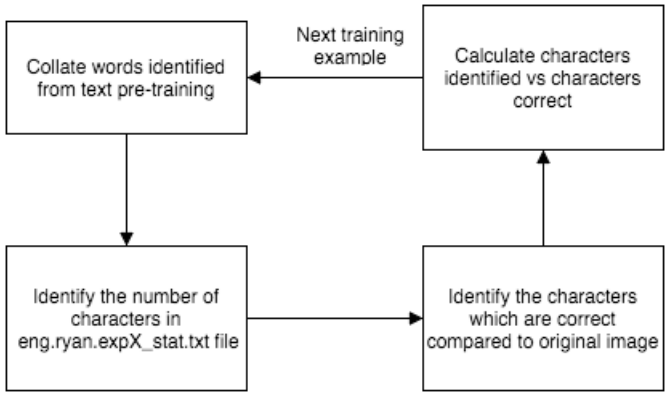

Figure 4.5: A simple framework showing the steps of analysing each of the training examples for a statistical measure for how successful the training process was.

Figure 4.5 shows a simple framework for analysing how well Tesseract trained the data. After the statistics has been collated then a graph was constructed to show the trends.

Figure 4.6 shows the output analysed from the Tesseract training. It shows each training example with an associated success rate for the characters identified. The conclusions clearly show that there is no improvement from the Tesseract output after around the 3rd example. A horizontal linear regression line shows that it has peaked at around 72% correct recognition rate. Refer to appendix **??**, section **??** for the full statistics.

4.7 Image threshold Testing

Due to the nature of the image processing script it was quickly realised a methodical approach to testing would not be beneficial, due to almost constant spike work.

TDD would not be performed before considering the design for the image bininarsation script. Instead the testing would consist of a more visual check of the outputted image to see if the result was successfully binarised. Once the script had been developed to a reasonable level of satisfaction the spike work would cease, and testing would ensue.

The code was re-written following a TDD approach. It involved analysing numpy arrays returned from OpenCV and checking, for instance, if any of the array contained black values.

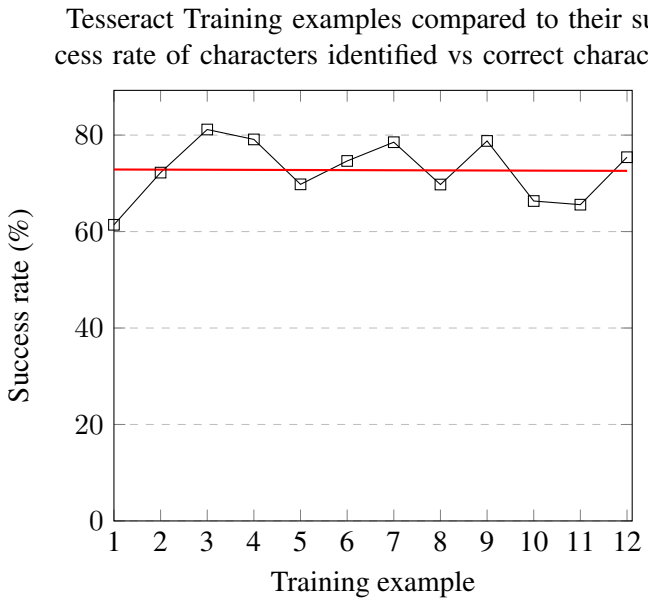

Figure 4.6: A line-graph showing the success rate of the Tesseract training results over 12 examples.

ADD reference to appendix to show examples of output

Chapter 5

Evaluation

This chapter will evaluate how well the project as a whole has gone. It will look at different issues relating from correctly identified requirements, design decisions, what was good about the project and areas which could be improved.

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 aforementioned requirements were part of the core system requirements; these were classified as the minimum functionality of the application. Personally these targets, which was partly self-imposed, was extremely challenging. Due to the size of the project, with lots of different components, work was produced at a steady rate over the sprints.

Investigation and research work was extensively conducted to identify how handwriting can be extracted from an image. This was important in aiming to optimisie the performance from the Tesseract engine. This was achieved through the binarisation of the image.

After twelve different training examples, created from handwritten notes, the Tesseract engine was not yielding any greater of a success. Therefore it can be concluded that a sufficient job has been achieved with the handwriting training. One of the most disheartening experiences was when characters could not be identified by the Tesseract engine, during the training phase. Nevertheless, Tesseract has reached a return rate on characters which can not be improved.

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, with the finished product showing the diligent effort constructed on the application. 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 has an intuitive interface allowing the user to navigate around the web application easily. Implementation support has been added so the user does not have to concern themselves over case-sensitivity during the use of the application, creating 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 text from the image, displaying the content to the user easily. This was built on-top of the work already implemented in the project with the handwriting recognition, allowing it to be incorporated into the application to complete the archiving tool.

Finally, from the original requirements the calendar integration was successfully implemented; it was in retrospect one of the more challenging tasks to overcome. Ranging from the testing the services to implementing reoccurring events, Google calendar always threw up a lot of issues. The reoccurring events and all day events were particularly challenges specifically down to all the possible use-cases. It can list all events in the last 7 days, find specific events by day and is able to find reoccurring events. On-top 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 some additional features such as clicking the suggested data. The application has been well tested, mainly from a strong testing infrastructure offering a range of unit, acceptance and route tests. There have been complex design decisions to ensure that the design of the application is as simple as possible. Overall, the application has been completed to a good standard.

5.2 Design decisions

Reflecting on the design decisions which were evaluated in section x. The use of the MVC framework structure was extremely beneficial. It encouraged that the code was decoupled at all times, leaving the system extremely modular.

The overall database diagram is well designed, and each of the relations is correctly identified and justified. One aspect which would be nice to change would be the note relation, so that the image path was represented in its own relation.

In the binarisation class a couple of the methods could be static due to not interacting with the classes properties.

Overall, the design of the classes show a good level of understanding regarding the object oriented paradigm. The use of helpers and service objects help to extend the system, reducing the amount of duplicated code, whilst semantically collating similarly related code together.

Although not strictly related to the core design of the system, but it was wished that the PEP8 [**?**] standard was followed and implemented as the application was being developed. Due to an inexperience using Python, this was not realised until later on in the project - this resulted in a larger refactor; this should have been used from the beginning.

5.3 Use of tools

This section will evaluate the key tools used in the application and justify whether they were the right or wrong tools.

5.3.1 Flask

During the project there were times in which there were feelings that the Flask, the micro-framework, was the wrong design decision. There was a lack of documentation and out-the-box support made even the most basic of functionality difficult and unnecessarily complex. For example the configuration file documentation for multiple configuration files was very poor and by default no support was given.

Another example is default security concerns. As stressed the importance of in *Internet based applications* cross-site request forgery (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, and the use of blueprints, and routing, enabled the MVC design pattern to be utilised. It would have been nice to see the framework giving support for multiple configurations.

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 substantially, without the use of OpenCV it could be argued that the Tesseract training would never have gotten off the ground. The research conducted into looking into different binarisation techniques was imperative; 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, retrospectively analysing the decision. However, the decision to use that over SQLlite, especially as more than one person would interact with the application was a correct design choice.

In the end the choice between PostgreSQL and MySQL would not have mattered a great deal for the application at its current stage, as both would be suitable and interchangeable.

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. Since Tesseract gave around a 75% success rate, that was still rather high for an OCR.

5.3.5 Google calendar

Although Google Calendar exposed a lot of complexities 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. Furthermore, the support from Google was beneficial as information regarding API calls could easily be found.

5.3.6 Continuous integration tool

Travis was a superb tool used throughout the development of the application. It was imperative with catching testing errors in an isolated environment. The only disadvantage is that the build times are *slow*. Having to compile OpenCV and Tesseract from source every time a push is committed meant that build times were around 25 minutes. This was far too long, but until the repositories are updated to the correct version this will be the same build time.

5.4 Meeting the users needs

One of the key premises that the application was worthwhile was that the user would need to digitise their notes easily. However, from the analysis given back from the user testing it seems as though that selection of users did not find the application useful for their note-taking style.

Primarily, this was due to people taking notes in different ways. They often prefer to write up their notes. One way in which this could be overcome would be to provide a WYSIWYG editor, so the user can have full control over what is entered into the note application.

So technically, the application did not meet the use-case of the users wholly. Certain aspects of the application would have to be developed further, to ensure that the users needs were fully met.

5.5 Additional project aims

Due to the nature of the project, all the features and functionality could not be implemented into one project spanning just 15 weeks. Due to this a few requirements were cut back from the initial specification, some of which are critical for improving the system.

5.5.1 Auto-correcting of image

A big problem with the handwriting recognition is that the image has to be correctly rotated to 90 °, and cropped sufficiently. This would have been a nice feature to look to implement, but due to

the time constraints this was not plausible. Sadly, the resultant of not including this will impact the user-experience, as it would have to be cropped prior to uploading.

On reflection this would have been preferable and should have been a higher priority in the backlog. However, issues arose with the calendar taking longer than expected, resulting in being unable to start this task.

5.5.2 Extracting images

Extracting images would have been the next logical step for the application to be developed. This would mean that the notes would have to have a massive restructure with the way it parses text. However, been able to select any image in a note and then change the size on a canvas would have been a great feature to implement.

5.5.3 Generic handwriting recognition

A core acknowledgement of the limitation that the system holds is that the handwriting recognition only works for the author's handwriting. To overcome this, it would have been preferable to develop a tool to dynamically train the user's handwriting in the application.

From Figure **??** it shows that all that would be needed would be three training examples to help the user to have a good recognition rate. The improvement by making it more generic would result in the application been accessible for a wide range of users.

In conjunction with this, extending the popular words list that Tesseract has would have been a nice way to get a domain specific set of text on a note. For example, if the user had three notes of CS31310, then it would add that module code to the popular words lists, making it more likely to find it again.

5.6 Evaluating the process

The process followed on this application has been done with diligence and precision. The Scrum approach has been fully adopted and has aided significantly in the design, development and testing. The weekly sprints helped to decompose the tasks that were at hand, enforcing that the tasks for that sprint was the main priority. What was most useful was the burndown charts every sprint, this helped to analyse exactly what went wrong, or right, on specific days - as well as providing an easy way to view the velocity.

The retrospectives were valuable in ensuring that the process could always looked to be improved upon.

What was most valuable though was the use of TDD. The system has been fully tested and there are a wide range of tests which show a great precision in an attempt to develop a good bit of software. There were frustrating times, as described in the testing section, but overall the approach of TDD was one of the better decisions.

Overall, the process was stuck to thoroughly and meticulously even when the times were tough. In the end the process aided with many aspects of the design and keeping the implementation simple. Therefore, Scrum was deservedly the right process to use.

5.7 Starting again

Although this project has been completed to a high standard, there are a few aspects which would be changed if this was to be started again.

When considering the use of the database management systems, a stronger analysis should have been considered when making the decision when choosing between MySQL and NoSQL. A version of a NoSQL system would have been implemented instead of a relational model, in the hind-sight of the application. This would allow the application to become a generic note-taking application which is not constrained to pure lecture notes for students.

Whilst training the handwriting data it would be imperative to keep a record of how well the training is doing as it was being performed. This iterative approach at creating a graph would have identified earlier where the training had stagnated so further examples did not have to be trained.

In-light of the Flask application potentially being the wrong framework for the solution, another framework has been considered. Still adopting the MVC structure, Django should have been considered more comprehensively during the design phase. Although it is larger, the complexity of the application grew and further out of the box support was needed.

5.8 Relevance to degree scheme

The author's degree scheme is *Computer Science*. The project has shown a full range of capabilities which satisfy that this project has enhanced and furthered knowledge relating to the subject of Computer Science, as well as enhancing skills already learnt throughout the duration of the author's degree.

The project incorporates many different engineering aspects:

- It is developed in an agile methodology process, enforcing good software engineering practices throughout the entire project.
- Design patterns were considered and used throughout the project, predominantly MVC.
- Research work to identify how to binarise a script using computer vision techniques.
- Programming was conducted to implement a fully functioning web application, following a code re-usability ethos.
- Evaluations and experiments conducted to analyse the accuracy of successful characters identified by an OCR engine.

Overall, there are many aspects of this project which encompass the field of computer science from the research elements, to the analysis and right down to the process followed.

5.9 Overall conclusions

Although alternative implementations and design decisions could have been made it should not be deterred from the application that has been produced.

An application which allows a user to upload their note, tag it with identified meta-data which has been extracted from the user's handwriting and then integrate with this in the calendar is a substantial application. Comprehensive research work was conducted to analyse how Tesseract could have been optimised, and a binarisation script, which works on a variety of images.

This has been conducted with a solid process backing the design, testing and implementation iterative process to show a high quality engineered project.

After substantial work has been completed on this application, the author is delighted with the outcome and believes that the application can be further enhanced with additional features aforementioned, to make it a solid use-case for helping students with their University lecture notes.

Appendices

Annotated Bibliography

[1] "sirfz/tesserocr: A Python wrapper for the tesseract-ocr API," last checked 25th April 2016. [Online]. Available: https://github.com/sirfz/tesserocr

The Tesseract wrapper which was used to extract the data from the image. It gives the confidences and all the words associated to the lines.

[2] "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan. 1979. [Online]. Available: http://dx.doi.org/10.1109/tsmc.1979.4310076

The original paper which OTSU is represented. Although a bit mathematical, some bits were good for reference material on how the algorithm works.

[3] "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

[4] "OpenCV: Extract horizontal and vertical lines by using morphological operations," 2015, last checked 25th April 2016. [Online]. Available: http://docs.opencv.org/3.1.0/d1/dee/tutorial_moprh_lines_detection.html\#gsc.tab=0

A great reference on how to use different morphological operations and adaptive threshold techniques to extract and binarise an image. Used extensively with the image segmentation script.

[5] 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.

[6] Bottle, "Bottle: Python Web Framework Bottle 0.13-dev documentation," http://bottlepy.org/docs/dev/index.html, last checked 22nd April 2016.

The Python framework was used as a case-study of potential frameworks to use for the application. Discussed in the design section, but rejected as a choice.

[7] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008, pp. 138–139.

A book which explains how the Gaussian function for the adaptive threshold with OpenCV works. It gives a simple description, one which is easy to follow.

[8] M. Daly, "Mocking External Apis in Python - Matthew Daly's Blog," http://matthewdaly.co.uk/blog/2016/01/26/mocking-external-apis-in-python/, Jan. 2015, last checked 25th April 2016.

A nice simple blog post explaining why hitting an external API is bad, and there should mocking objects instead.

[9] P. Developers, "PEP 8 – Style Guide for Python Code — Python.org," https://www.python.org/dev/peps/pep-0008/, last checked 23rd April 2016.

The PEP8 standard was used throughout the codebase as an implementation style guide. It is referenced in the evaluation to discuss the design decision that should have been implemented from the start of the project.

[10] Django, "The Web framework for perfectionists with deadlines — Django," https://www.djangoproject.com/, last checked 22nd April 2016.

The Python framework was used as a case study, looking at the different frameworks available. It was rejected for it being too large for the project.

[11] 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.

[12] 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.

[13] Fisher, "Point Operations - Adaptive Thresholding," 2003, last checked 25th April 2016. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm

An article explaining clearly and simply how the adaptive thresholding algorithm works. It gives a good level of detail and is concise in its points.

[14] Flask, "Welcome — Flask (A Python Microframework)," http://flask.pocoo.org/, last checked 22nd April 2016.

The python framework used as an option. Was used in the design section evaluating the decisions that were made. It was used as the choice of framework.

[15] ——, "Testing Flask Applications Flask Documentation (0.10)," http://flask.pocoo.org/docs/0.10/testing/\#accessing-and-modifying-sessions, 2016, last checked 24th April 2016.

The testing documentation for Flask which discusses how session modifications should be handled. Used in the implementation and the testing discussion.

[16] T. P. S. Foundation, "26.5. unittest.mock mock object library," https://docs.python.org/3/library/unittest.mock.html, 2016, last checked 24th April 2016.

The mocking library used throughout the application. Although the documentation is for python 3, it works for python 2.7

[17] M. Fowler, "Mocks Aren't Stubs," http://martinfowler.com/articles/mocksArentStubs.html, last checked 25th April 2016.

When deciding whether mocks or stubs were used, Martin Fowler gave a nice concise answer. It turns out all the tests are mocking the behaviour from the external API.

[18] 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.

[19] R. Gouldsmith, "build throws KeyError: 'rootUrl' error on Google Calendar API Issue #208 google/google-api-python-client," https://github.com/google/google-api-python-client/issues/208, 2016, last checked 25th April 2016.

A issue which was raised by the author, regarding an issue experienced with a 3rd party library.

[20] ——, "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.

[21] A. Greensted, "Otsu Thresholding - The Lab Book Pages," http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html, June 2010, last checked 25th April 2016.

A great reference tutorial aiding to identify what OTSU threshold is and how it works in a simple to understand manner, with plenty of example.

[22] C. Heer, "Flask-Testing Flask-Testing 0.3 documentation," http://pythonhosted.org/Flask-Testing/, last checked 25th April 2016.

The documentation page for the testing library Flask-Testing. It was used throughout the project after a refactor realising it offered better support for testing Flask applications.

[23] 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.

[24] 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.

[25] 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

[26] ——, "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.

[27] ——, "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.

[28] MongoDB, "MongoDB for GIANT Ideas — MongoDB," https://www.mongodb.com/, last checked 22nd April 2016.

The Mongo DB tool used as a comparison for relational database systems and NoSQL ones.

[29] B. Muthukadan, "Selenium with Python - Selenium Python Bindings 2 documentation," https://selenium-python.readthedocs.org/, 2014, last checked 24th April 2016.

The selenium library used for the acceptance tests. It gives good documentation on how to access elements and how to get specific values from the text.

[30] H.-F. Ng, "Automatic thresholding for defect detection," *Pattern Recognition Letters*, vol. 27, no. 14, pp. 1644–1649, Oct. 2006, last checked 25th April 2016.

This paper was interesting as it aided in the dicussion of the different thresholding algorithms. It was good to reaffirm some knowledge gained during the development process.

[31] 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.

[32] Opencv, "Miscellaneous Image Transformations OpenCV 2.4.13.0 documentation," 2016, last Checked 25th April 2016. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html

A description of the adaptive threshold function, which shows that there are two different functionc can be used.

[33] Oracle, "Overview - The Java EE 6 Tutorial," https://docs.oracle.com/javaee/6/tutorial/doc/bnaaw.html, last checked 22nd April 2016.

An article which discusses the use of Java as a web application language. It reaffirms the point raised that it is good for performance.

[34] 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.

[35] pytest-dev team, "pytest: helps you write better programs," http://pytest.org/latest/, last checked 24th April 2016.

The library was used throughout the development for reference on testing. It was especially useful for mocking test data.

[36] R. Python, "Headless Selenium Testing with Python and PhantomJS - Real Python," https://realpython.com/blog/python/headless-selenium-testing-with-python-and-phantomjs/, Aug. 2014, last checked 24th April 2016.

A demonstration on how to use Selenium with the Python examples. Additionally references the fact what phantomjs is, and it is a headless browser.

[37] 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.

[38] 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

[39] 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

[40] 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.

[41] M. Sturgill and S. J. Simske, "An Optical Character Recognition Approach to Qualifying Thresholding Algorithms," in *Proceedings of the Eighth ACM Symposium on Document Engineering*, ser. DocEng '08. New York, NY, USA: ACM, 2008, pp. 263–266. [Online]. Available: http://dx.doi.org/10.1145/1410140.1410197

A great paper which discusses the Tesseract engine by HP researchers. It is used to discuss the idea that OTSU is used as its pre-processing step.

[42] 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.

[43] O. Tezer, "SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems — DigitalOcean," https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems, last checked 22nd April 2016.

Used as a comparison between what relational management system should be used. Used in the design section for a comparision between the different systems presented and evaluated.

[44] 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.

[45] R. Viet OC, "jTessBoxEditor - Tesseract box editor & trainer," last Accessed 6th February 2016. [Online]. Available: http://vietocr.sourceforge.net/training.html

An excellent software package which allows the user to train the box files with a great graphical user-interface.

[46] w3Techs, "Usage Statistics and Market Share of JavaScript for Websites, April 2016," http://w3techs.com/technologies/details/pl-js/all/all, last checked 22nd April 2016.

The website shows a graph of how Javascript has increased its market share on recent web applications. Used as part of the design consideration regarding the use of programming language

[47] 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.

[48] 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.