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

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

By including my name 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 Stormzy for having an awesome playlist on spotify. And Katie I guess 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. [57] 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 vary in structure from note to 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 [69].

As an initial step in this project an informal survey of note-takers was conducted. It was concluded that 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

Figure 1.1 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

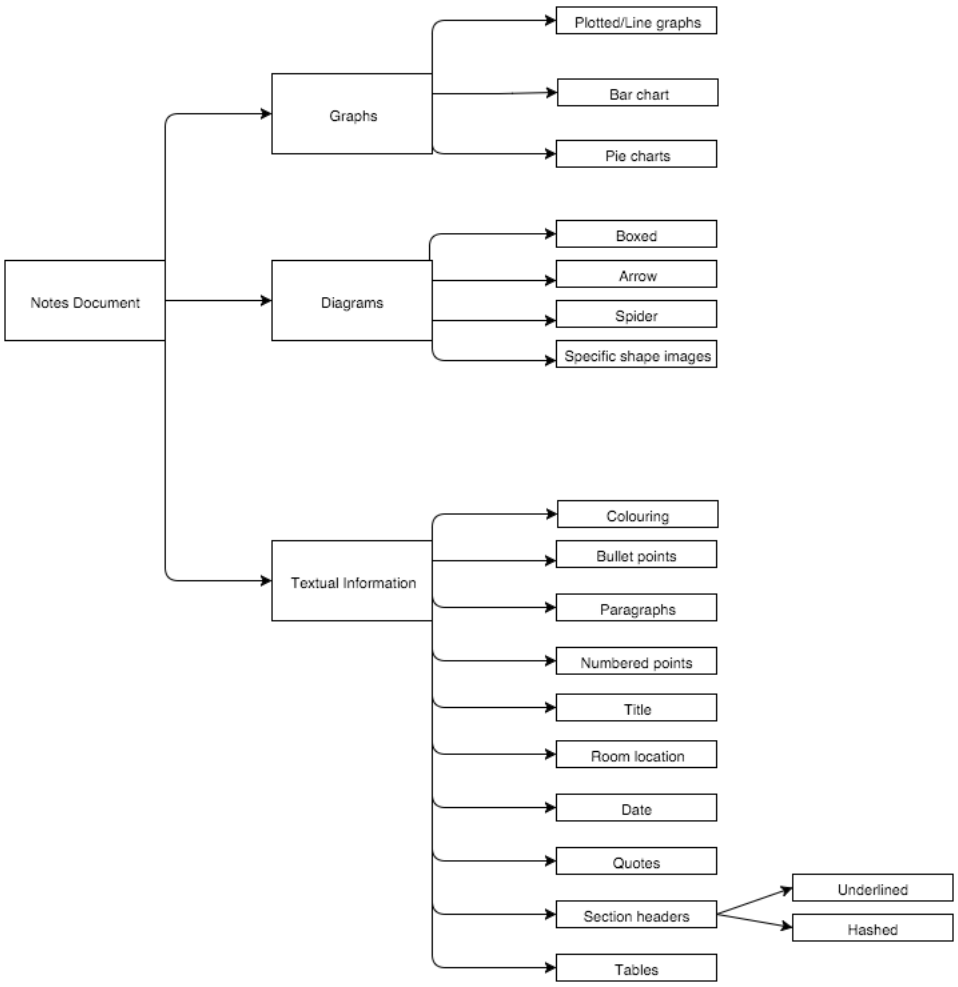


Figure 1.1: A taxonomy showing the structure and classification of different types of notes and what is contained in a note.

ways, for example the headings may be underlined or hashed - if they were adopting a mark-down style approach. This syntax helps to show that there's a break in the content, and it should be sub-sectioned. Text 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 [13]. Both congruent and incongruent coloured text helped to increase memory retention of post-graduate learners [44]. 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 textual 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, textual 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 [35]. 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. [53] used the open-source tool, Tesseract [60] 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 rate on this character alone.

The problem of handwriting recognition has not been solved - but tools such as Tesseract offer support for improvement in this field. As Rakshit et al., discussed 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 text identification rate of 83.5% experienced by Rakshit et al., it is a viable option to use for handwriting recognition.

1.1.2.1 How Tesseract works

Tesseract was initially developed by HP, but has now been made into an open-source tool. Smith [56] gives an excellent overview of the Tesseract engine. The following section summarises how Tesseract works. Tesseract uses connected components to identify the outline of characters and these are then collected 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 [55]. 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 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, 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:

- OneNote
- EverNote
- Google Keep

1.1.3.1 OneNote

OneNote [38] 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 [40]. 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 [39] 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 [14] 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 [5] 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 [27] 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 is been preferred, offering a more raw feel to the application. The user has 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 to-do lists and jotting down quick notes, rather than an archiving tool suitable for substantial note taking. Nevertheless, the tagging with labels is a useful feature and the filter by image is a smart tool; this only shows notes with specific images. The simplicity of the User Interface (UI) and the ease in which text can be extract provides a great reference.

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 contain. The

automatically cropping of an image is an important feature and should be considered for the application in the future. *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 these are often stored in notebooks, with no structure 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, instead of trawling through lots of paper and trying to find the content. 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.

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 the main component of the note the primary efforts of the application would be focussed on parsing the text. Diagrams, graphs and images would be future work - due to time constraints.

An OCR tool

As handwriting recognition is an active research project, a bespoke handwriting recognition tool using machine learning techniques could have been developed. However this would not have solved the issue of creating a 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.

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. There needed to a decision of 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 metadata: 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 metadata but not automatically tag the metadata.

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.

OCR for the authors handwriting

After research into OCR technologies, such as Tesseract [60], 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.

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.

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.

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, [50] 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.

1.2.1 Objectives

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

1. Investigate how to extract handwritten 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.
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 full list of notes based on the module code entered.
6. The backend of the application must conduct basic OCR recognition, analysing the first three lines of the notes.
7. The backend must integrate with a calendar to archive the notes into users events.

1.2.2 Compromising with objectives

Some additional compromises were made separate of the analysis due to the complexity of the tasks at hand.

- It would be desirable 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 so these projects lend themselves to an Agile approach. 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 reliably estimated 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 [54] 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 'Goldilocks' task [1].

For planning a sprint, the planning poker [58] technique was adopted; user-stories are estimated on a scale of 1, 2, 3, 5, 8 etc. When a task was estimated about 15 story points, it would be reflected upon to ensure the scope was fully understood - this would be broken down to sub-stories where appropriate.

At the end of a sprint, a review and retrospective was conducted in the form of a blog post [29], 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 week's 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 [62] 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. This was reciprocated from both sides where both Major Projects were discussed. The primary aim was to identify

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

the main tasks which needed to be completed that day to keep the project progressing.

1.3.3 Incorporated Extreme Programming

In tandem with Scrum, Extreme programming [7] 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 [37]. This was implemented throughout the project, with both unit and acceptance tests being written before the code implementation.

TDD follows three cycles: red, green and 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. This would result in ensuring that new features were created from all code committed into the repository. See Section 2.6.3 for further reference on how CI was used in the project.

1.3.3.3 CRC cards

Class, responsibilities and collaboration (CRC) cards [70] 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. See Section 2.1.2 for further reference on how CRC cards were utilised.

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. Instead adopting an iterative approach to design was preferred. Regardless of this, some important design decisions were decided at the start of the project. The chapter will clearly explain rationale for the decisions and state whether they were the result of an iterative processes or an upfront design.

2.1 Overall architecture

This section discusses the architecture of the web application. The design for the web application was developed over a series of sprints, iteratively increasing with each user story, therefore no upfront design was conducted at the start of the process.

2.1.1 Class Diagram

An overview of the resulting design of the class diagram is presented, with rationale for decisions made and how an iterative approach was used to reach the concluded design. The class diagram can be found in Appendix G, section 7.1.

2.1.1.1 Justification of design

The following section discusses the appropriateness of the design and any justifications needed. Overall, the design clearly shows the object oriented principle of low coupling high cohesion being used on the project.

Google Services
During early iterations, the Google Calendar API was only going to be utilised to parse the user's calendar events. As a result, the class `GoogleCalendarService` was created - this would ensure that the logic encapsulating the Google calendar was centralised into one class. With the version number and API unlikely to change, constants were chosen as the best identifier; if the URLs and version number were to change in the future it would be easy to change these. The initial purpose of the class was to perform key operations to extract the events, as shown with the

function `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 functions were added.

Initially users were not considered a core part of the system. However, the user story to incorporate users into the system was created. Upon creation it was clear that another class to integrate with the Google Plus API would be required. This class followed a similar structure to the calendar class, except for parsing a user's email, so that it can be persisted in the database.

Eventually, the design was evaluated and duplicated functionality amongst the methods was discovered. In both of the classes the `build` and `execute_request` functions were duplicated, without having class specific content. As a result, the extract class refactoring technique [20], was used to create a super class `BaseGoogleService`. This class encapsulates logic for building and executing queries. Extracting these functions to a super class ensured that pure logic for data and query manipulation can be moved to the `GoogleCalendarService` and `GooglePlusService` classes.

Helper classes
Helper classes, in the design, are independent classes that help to modularise the system - whilst grouping related functionality into a single class. As the system grew the duplication of code was beginning to become apparent, so helper classes aid in keeping a design simple.

For example the `SessionHelper` class was developed to initially store credentials after the oAuth log in, discussed in Section 3.4.1, had been completed and had successfully being appended to the session. The class' functions expanded as further duplication of the session handling was developed into the system. This level of abstraction gave a semantic meaning to the interactions with the session.

Most of the helper classes do not interact with the other classes in the system. There is an exception with the `GoogleServicesHelper` class. In this class majority of the functions are static. This design decision was induced due to the class not interacting with any specific class level attributes. Furthermore, prior to the implementation of editing a calendar event, the code was dispersed throughout the controllers. In an effort to reduce code duplication this helper class was created - but it was quickly decided that it would just be a proxy for calling specific functions in each of the services classes. Although ideally they should be class level functions, it is appropriate to use static functions.

Persistence classes
The relationships between the persistence classes will not be discussed in this section, see Section 2.4. 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 a reflection on the content of a note was conducted that the design changed making this field a required attribute.

There are a series of `save` functions in the application, these were added to the design when the controllers were constantly duplicating functionality when persisting object to the database. This improved the readability of the application, providing a succinct solution to persisting an instance to the database.

In parts of the application, information needed to be extracted from the database. To aid in readability, static methods such as `find_meta_data` were created to keep domain related functionality together, but without creating a specific instance.

Binarisation
The `BinariseImage` class is the model representation of the image segmentation script, as seen in section 3.1. The class is called from the controller when a user uploads their image. The output from the class methods is a binarised image. There are a series of functions which integrate with OpenCV API's [33], manipulating an image and performing morphological operations. The class has been constructed so most of the functions are modular.

2.1.2 CRC cards

To aid with the design, class collaboration cards (CRC) were drawn up for each feature. The user-story was decomposed into tasks and each of the tasks had associated CRC cards. This aided in thinking about the design for the class, as well as other classes it interacts with. The overall design discussed in section 2.1.1 is a result of the diligent planning with the CRC cards.

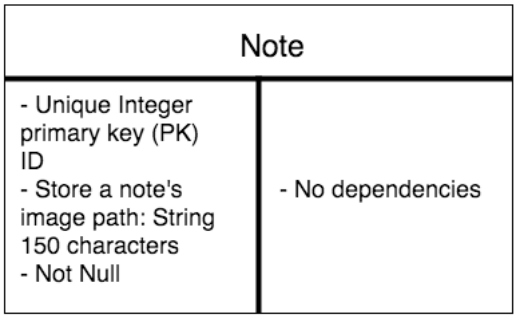


Figure 2.1: An example from Sprint 3, showing a CRC card at the very beginning of creation.

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.

Throughout each feature implemented into the system, the CRC cards were created, evaluated and refactored as "throw away designs". Although they were lightweight design tools, they helped to think about the system for the current feature being implemented - reducing the future design creep. For example, during the creation of the note into CRC cards, the image patch attribute was considered to be its own relation. After evaluating that this would be overcomplicating the design for the current feature this design decision was rejected and it would not add benefits to the existing system. The CRC cards were kept for each design and formulated into the class diagram at the end of the development process for formal documentation.

Overall CRC cards were at the forefront of the design during this project. They enabled a clear, concise and well considered design to evolve over a series of sprints. For a further example of an in-depth CRC card see, Appendix H section 8.1.

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, an activity diagram was constructed to consider the flow of the application.

Throughout the sprints, the design for the activity diagram expanded. The result is depicted in Figure 2.2. The user was not initially part of the application, so the activity did not include the first activity of logging into the system. This was included into diagram once the user story for users must be incorporated into the system was brought forward into the sprint.

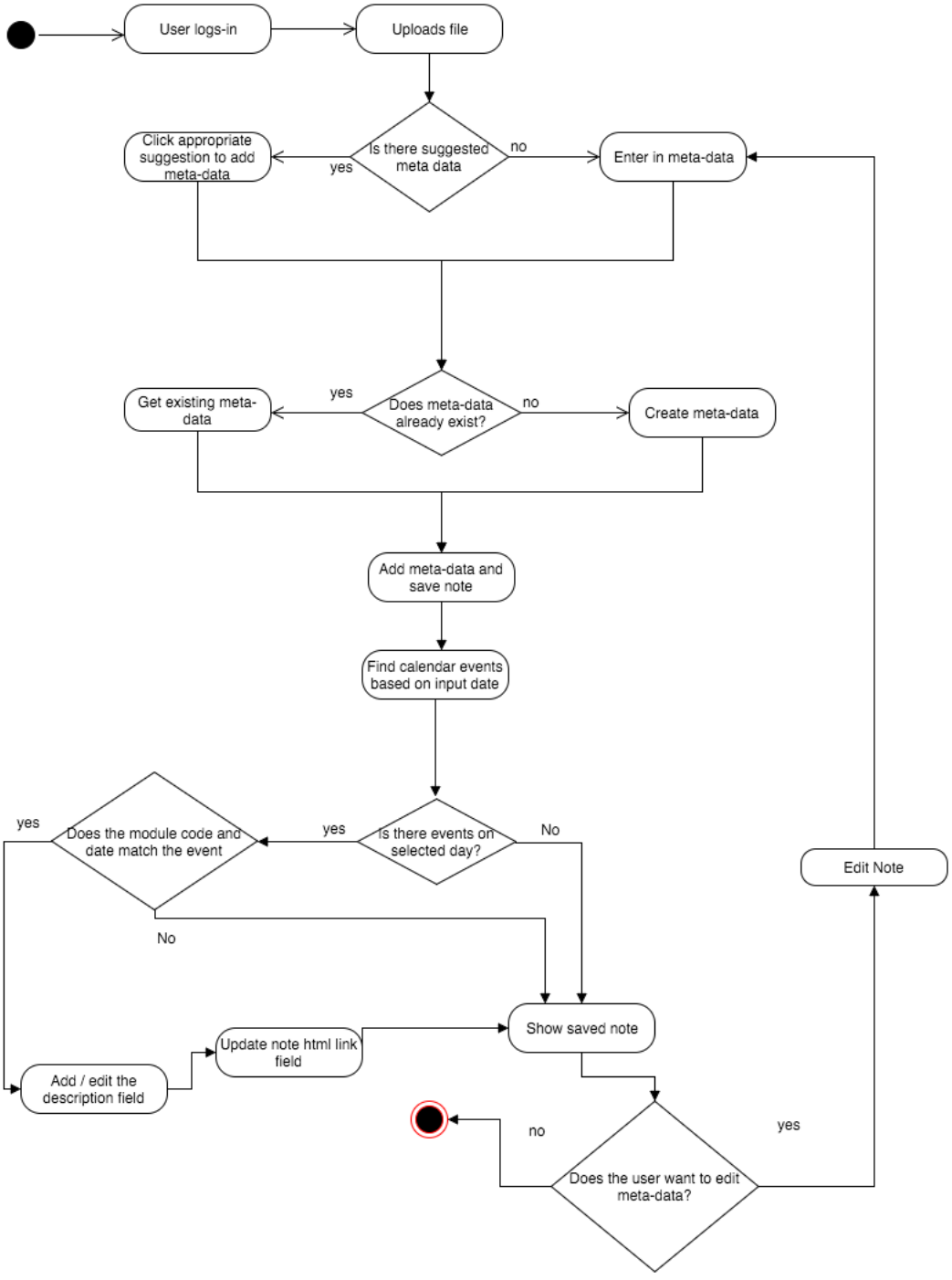


Figure 2.2: An activity diagram to show how to save a note and the integrations with the calendar.

The conditional checks to identify if there was meta-data outputted from the Tesseract output

could not be clicked upon to populate the form. This activity was included into the design during the planning of the feature for that sprint.

Overall, the activity diagram displayed shows the final output of how a note is added into the system. This design has been meticulously developed through a series of iterations to show the final output. It shows that a user can upload an image, they can select any associated meta-data from the suggestions, save the note and it will add it to the calendar item; there is also the option to edit the note.

2.1.4 Model-view-controller

The application would be designed in an Model-View-Controller (MVC) approach. Rationale for different aspects of the MVC structure will be discussed.

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 not to directly integrate with database and specific logic, instead to interact with a series of models and services. Finally, the controllers will aid in passing dynamic content to view files, returning rendered HTML.

The model in the MVC structure has no acknowledgement of the view file. Instead of rendering any form of HTML, the model 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 from the controller. There may be specific logic which impacts the HTML displayed, but no direct calls are made to the database layer or the controller. It uses the dynamic content passed in.

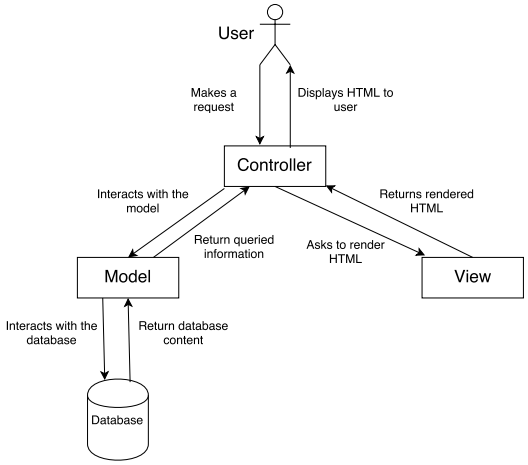


Figure 2.3: A example of how the model-view-controller (MVC) framework integrates.

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 objective when considering the design of the application would be reusability of the codebase, where applicable. 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 would become obfuscated, due to related files - such as views - not being grouped together. Due to this preference an MVC approach would be appropriate - as all the view files can be placed in the same directory.

The framework chosen, see Section 2.6.2, does not support an MVC structure 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 purpose. It also over-complicates the identification of interdependent classes, as it is not explicitly clear from the imports what class is used.

To overcome this, Blueprints [2] were used. Bluerpints are modularised routes allowing different routing options to be placed in different files. Annotations were used to define the blueprint route - each being its own separate controller.

Models were separated into their own directory, and a one class per file policy was adopted to keep the design clean and simple. This ensured that the related file only represented the one class in the system - this would remove any ambiguity when looking at the directory structure.

It is worth considering the view files. The view files were the only section of the web application structure which underwent an iterative process. Initially, the view files would represent the entire DOM tree in a singular file (duplicating headers, scripts etc). This is not the best design decision as there is core HTML which would not change between the different view files, so there was additional duplication that was redundant.

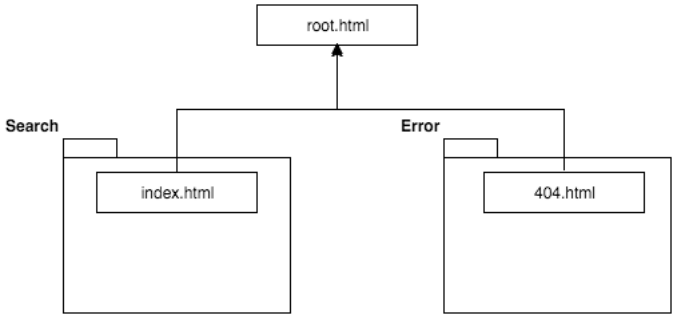


Figure 2.4: A diagram illustrating how extension in Jinga html template engine works.

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 interacting with the page can clearly see 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, query strings were decided to be used. 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.

RESTful URLs help to show the a hierarchy of content. Exposing a user to such a URL helps them to clearly identify their content. When the system evolved to displaying a note for a user `/show_note/1`, was chosen for the URL; it is easier to read than `/show_note?note_id=1`. This allows the user to not have additional query parameters to decipher before working out the context of the page.

For the story of viewing notes, it was worth noting that traditional RESTful URLs would be adapted for readability. For example `/view_notes/` was designed, when a proper RESTful URL may be `/notes/`. This offered more semantic meaning to the page's aim.

Overall the design considerations for the URL structure were an important design aspect that was considered to a great deal, to ensure that the user gets the best experience of interacting with the application as possible.

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 [32] was initially used by converting the image to greyscale - but this yielded poor results. After further design decisions were made to convert the image to monochrome this still returned too much noise. In the following iteration, the processing step would investigate whether the specific thresholding algorithms would be useful.

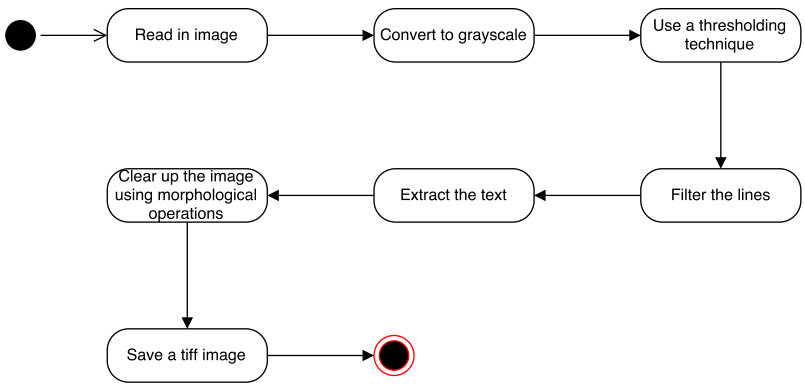


Figure 2.5: An activity diagram to depict the design of the algorithm for the image segmentation.

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 3.1.

This high-level activity diagram shows the design stages which were used as a high-level interpretation of the binarisation process. 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.

Blue lined paper was one way to overcome this issue. Filtering the lines from a thicker 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 lines to be important, but it was producing 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 should only extract the text. This was the final iteration of development on the binarisation script.

2.3 Tesseract

During the analysis phase Tesseract was identified as the OCR tool of choice. Patel et al. [48] performs a case study using Tesseract as the OCR tool to analyse printed text in an image. Patel et al., also discuss the comparison against a proprietary OCR tool, Transym [64].

Patel et al. concludes that Transym only yielded a 47% accuracy on 20 images compared to 70% accuracy using the Tesseract engine.

The first few iterations gave significant insight into how the document might me parsed. Due to the complexities with analysing the whole text on the image, it was limited to the first three lines, parsing the most useful information. It was decided that the first three lines were to be extracted forming the information for the metadata. Although design considerations for parsing the image and looking for key words was considered, it was ultimately rejected due to the complexity. Therefore, a structured approach was adopted. Below is an example of how the meta-data needs to be structured for the notes:

Listing 2.1: An example exert from a valid structured note

```
CS31130: This is a title
Date: 28th April 2016 14:00
By: A Lecturer's name
```

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. Pangram's, the "quick brown fox" is the most common example, is a good way to represent text as it contains all alphabetical characters [72]. 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

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

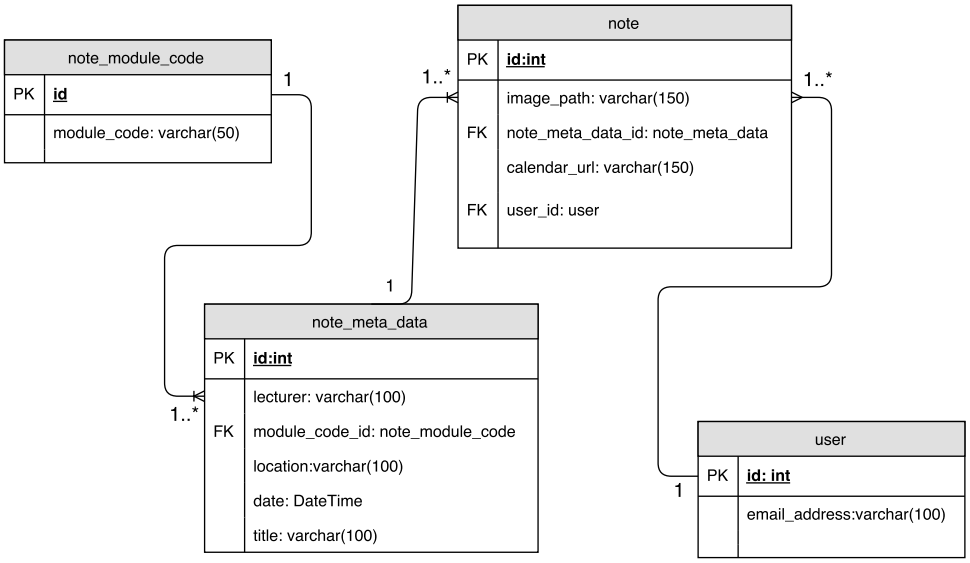


Figure 2.6: The final result of the entity-relation diagram - after a series of iterations.

Figure 2.6, shows the output from the result of final design of the entity-relation diagram. Each new user story added new design implication to the design. For example, when the very basic user-story for creating a note was established, the metadata needed to be added in the future, but the story involved just a note. So no foreign key was created, just the image path attribute.

2.4.1 Justification of design

Below is a justification of the designs through various stories which affected the entity-relation design.

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 easily located. 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 metadata, 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 metadata.

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 metadata to more than one note. As denoted from the relationships: a note will have a singular metadata item, but the metadata item could have many notes.

With attributes lecturer, location and datetime - these were the initial design decisions made to be included in the metadata. 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.

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 metadata and the module code is explicit: the metadata must contain one module code but the module code can have more than one metadata item.

User
This was not added to the application until around sprint five. However, the user will have an email address and that would be stored. It is in its own relation due to logic when creating a user: 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 data-redundancy, by providing solid rationale for the resulting design.

2.5 User Interface

With the web application being a core part, a series of User Interface (UI) designs were collated at the start of each breakdown of the story.

The UI had to 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 [25]. An alternative of Bootstrap [1] 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.

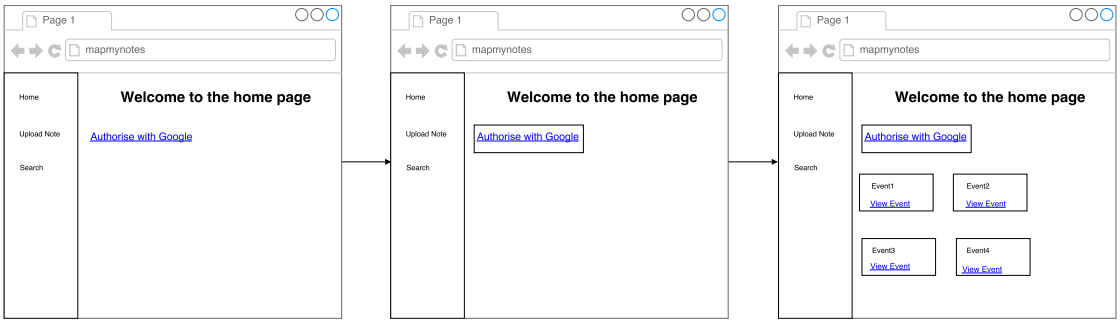


Figure 2.7: From left to right, the homepage wiremock through the different iterations and the change of requirements

Figure 2.7 shows the exploratory wireframe design completed prior to the UI. From the early iterations it was just an authorise button, then a requirement was added to show the user events from the last seven days it was mocked up to reflect this. This process was completed over the stories. If the story reflected a change in the content displayed on the screen a conceptual design was mocked up to ensure there was an idea of how it intended to look.

Further mockups available in Appendix H, Section 8.2.

2.6 Implementation tools

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

2.6.1 Programming language

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

As a web application was being developed investigatory work was completed into the suitability of several server-side languages. Traditionally server-side application languages are: PHP, Ruby, Python, C#, Java and JavaScript, which has increased in popularity [68].

Decomposition of 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 therefore 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 [47]. 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.

2.6.2 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 [12], Flask [16] and Bottle [8] were evaluated.

Some frameworks constrain the developers to specific implementations through abstracted classes whereas some offer more flexibility. Whilst evaluating Django, an extensive MVC framework, it was concluded that such a large framework was excessive for this application and it was 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 over the structure. On face value, Flask and Bottle appear to be very similar; they are both lightweight with a similar syntax. After evaluating both of the 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 quickly showing that it was a suitable tool to use.

2.6.3 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 [66] was an initial choice; it is a standalone Java application which a repository can be synced to.

Travis CI [65], 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.

Although there was not much difference between the two tools, Travis did have the advantage that the web interface could be used rather than a standalone application. A disadvantage of Jenkins would be that for each branch a built script would have to be developed; ideally, the CI tool would be a quick set up and go process, not to be lumbered with further changes. As a result, Travis was chosen as the CI of choice.

2.6.4 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 [63] repository on GitHub [22]. 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`. All branches were checked out from the development branch - ensuring that all features were from up to date commits. With each feature being developed in its own branch it ensured that any changes made would not affect the overall system. This provided a good platform to develop safely, whilst preserving working code.

Once the code was pushed to GitHub, Travis would automatically build the branch - inside the travis.yml file it would run a series of tests on the application. Once the tests had passed a pull request would be made on the branch into development. If this test successfully passes, and it is safe to merge then it was merged to development.

2.6.5 Development environment

The text editor, Atom [21], was used for the majority of the project. It is a lightweight text editor, which provides suitable syntax highlighting. However later in the project, when refactoring became more cumbersome due to the increase in code base - PyCharm community edition [34] was used as it offered better refactoring functionality.

Chapter 3

Implementation

This chapter discusses the implementation challenges of image processing, handwriting training and the creation of the web application. It will provide details on issues overcome, whilst identifying where issues may still persist.

3.1 Image processing

Image processing would prove to be an integral part of the application. Pre-processing of the image would be needed to improve the likelihood of Tesseract correctly identifying the characters. This went through several iterative prototypes prior to outputting a fully binarised image.

3.1.1 Optimising Tesseract

Prior to OpenCV being used as the image processing tool several attempts were made to binarise an image using ImageMagick. In sprint zero, converting the image to grey-scale was attempted but this returned poor results from Tesseract. The next iteration of the script was to convert the image to monochrome, this binarised the image, but left a lot of additional noise. It was then suggested by Dr Hannah Dee to use OpenCV for the binarisation process.

3.1.1.1 Otsu

Otsu [4] is a binarisation technique which essentially converts an image to black and white. Otsu is a global thresholding algorithm, where it uses the whole image for pixel comparison. This is unlike local thresholding algorithms where comparisons are made on smaller segments of the image [43].

Images of notes will often have non-uniform lighting; shadows will often be displayed as a user takes a photo. This is problematic for Otsu as shadows will affect the whole picture for a global thresholding algorithm.

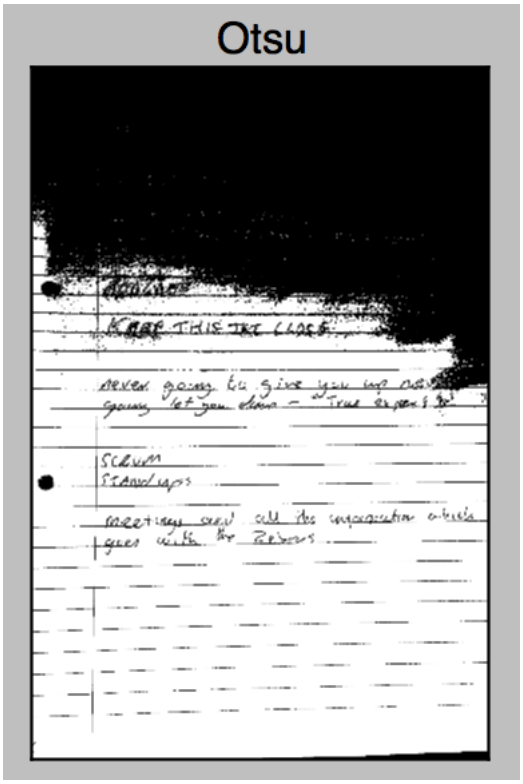


Figure 3.1: The use of Otsu binarisation technique on an image with a little shadow across the image

Figure 3.1 shows the Otsu binarisation method used on an image with the a slight shadow over the top right of the image. It can be clearly seen that the binarisation segments the image into two distinct regions: the bottom half is white whereas the top half is black. It can be concluded that this would not be a sufficient solution for identifying characters, when specific regions of the image are unreadable.

Otsu attempts to segment the grey-level from the image into a series of histograms. Otsu then determines the optimal threshold value by "maximising the discriminant measure" [4]. Essentially, Otsu attempts to maximise the margin between the histograms, this margin would then act as the threshold value as to whether a pixel is segmented into either a foreground or background pixel [30].

Hewlett-Packard, the creator of Tesseract, describe Otsu as its underlying pre-processing algorithm when converting the image prior to extracting textlines and characters [59]. Once the spike work was completed with Otsu, shown in Figure 3.1, it was clear to identify that Tesseract would find it difficult to identify the characters from the image when the output was so poorly binarised.

Overall Otsu, although it is a very reliable binarisation method, suffers from imposed shadows over images.

3.1.1.2 Adaptive threshold

As Tesseract uses Otsu as its pre-processing step, using an image which has been binarisated with Otsu already would not be advantageous in improving the accuracy of the characters detected. As a result, the next iteration evaluates an adaptive threshold technique.

Adaptive threshold calculates the threshold over a series of smaller segments in the image [15]. As a result shadows have a smaller impact over the whole image, due to adaptive threshold being a local threshold technique. This makes adaptive thresholding rewarding for non-uniform lighting situations, as it becomes more invariant to shadows.

Using the OpenCV library there was two options with adaptive threshold [46]:

1. Gaussian adaptive threshold: the weighted sum of the neighborhood.
2. Mean adaptive threshold: the mean of the neighborhood.

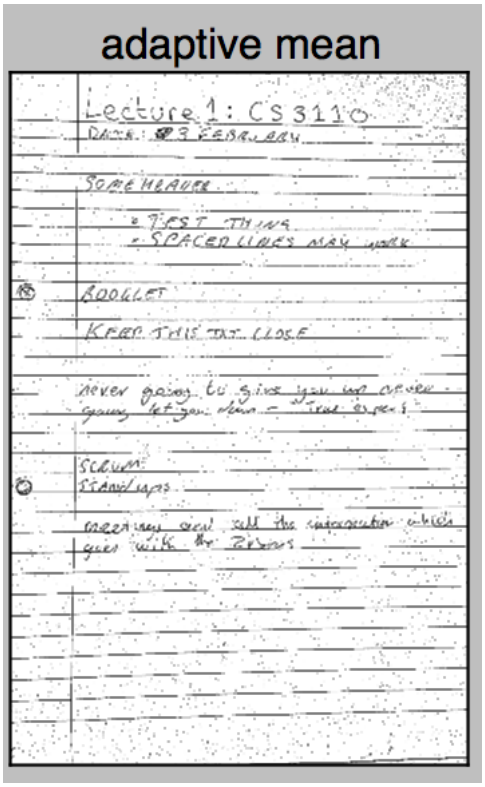


Figure 3.2: Adaptive mean threshold algorithm on a note, showing binarisation but there is still noise in the image.

Mean adaptive threshold uses a specific block size around a given pixel. If the block size was four, then the neighbourhood size would be four and calculations would be made to calculate the mean pixel value in that block. The mean value is selected as the thresholding value for the pixels inside the block; each of the pixels will then be identified as either foreground or background pixels [15]. Figure 3.2 shows mean adaptive threshold being used on an image. An additional issue present with the adaptive mean is the noise pixels; there are considerable amounts of noise polluting the image.

Gaussian adaptive threshold differs from the mean adaptive threshold, as instead of calculating the mean value over the block size, it first uses a Gaussian value over blocksize. A Gaussian weight is calculated dynamically depending on the blocksize used for the thresholding. Every pixel inside the block is then multiplied by the Gaussian, and an average value is then taken across the pixels which is used as a threshold. Like the mean adaptive threshold, this is then used to determine if the pixel is foreground or background [9] [15].

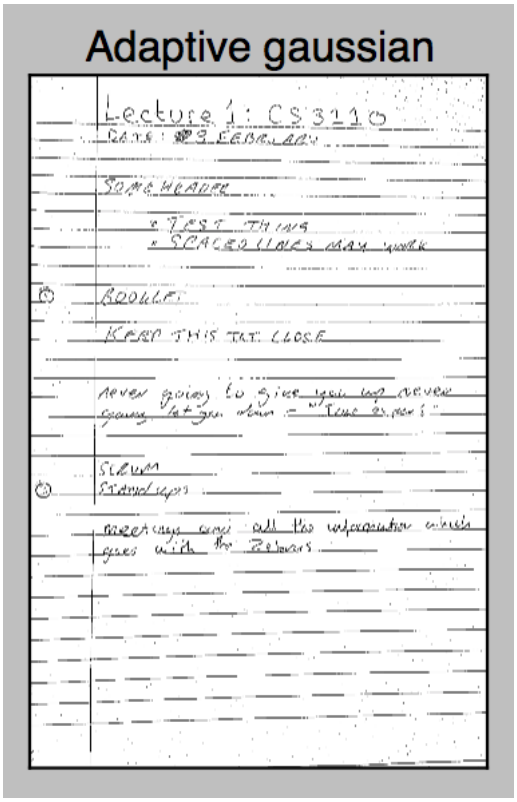


Figure 3.3: Adaptive Gaussian used over the image, showing a lot smoother of an image

Figure 3.3 shows the adaptive Gaussian being used to binarise an image. The output clearly does not have a shadow overlaying the image and there is less noise pixels than the mean adaptive thresholding.

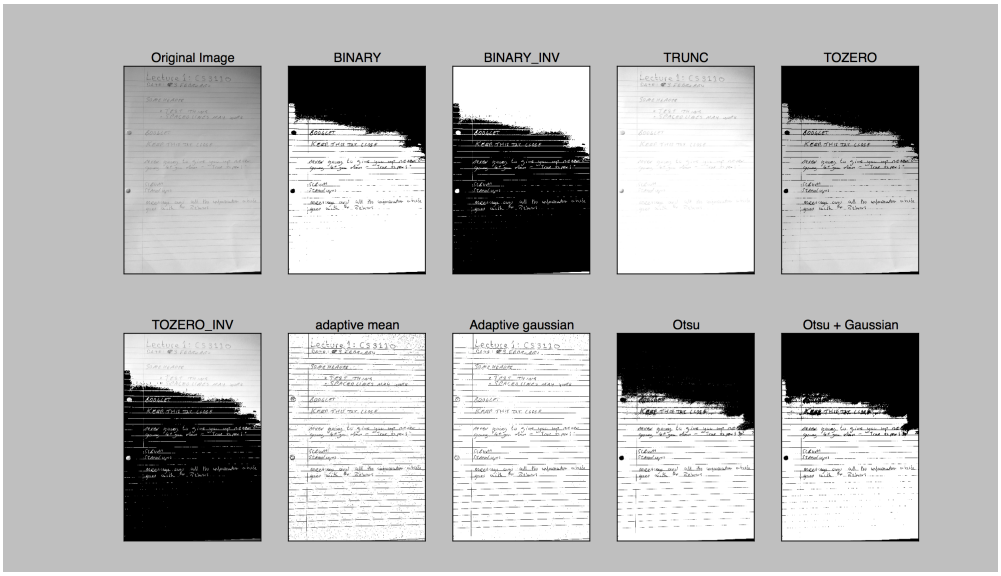


Figure 3.4: A variety of thresholding techniques used on the same note, showing adaptive threshold resulting in the best output.

Figure 3.4 displays additional types of the thresholding that were investigated throughout the iterative process. The Gaussian adaptive threshold provides the clearest results from visual inspection of the different thresholding techniques experimented. As a result, the Gaussian adaptive threshold will be continued to be improved upon throughout a series of iterations to reduce additional noise, via morphological operations to aid in smoothing.

3.2 Lined paper

Initially normal lined paper was used for notes in the project, but after the binarisation process it left too much noise. Further smoothing of the image did not remove the noise, so bespoke lined paper was created to aid in removing the lines but keeping the text uniformly straight. Refer to Appendix F, Section F.1.

3.2.1 Filtering the blue lines

Over a series of iterations, the primary objective was to remove the blue lines from the image. Examples of the lined paper can be found in Appendix F.

Algorithm 1 Initial removing the blue lines algorithm

```
1: function REMOVE_LINES
2:     image ← read_image_as_grayscale()
3:     lower_black ← np.array([0,0,0])
4:     upper_black ← np.array([175,20, 95])
5:     mask_black ← cv2.inRange(erode, lower_black, upper_black)
6:     mask[np.where(mask_black == 0)] ← 255
7: end function
```

Algorithm 2 attempted to filter all the colour values between a grey and a black range. By restricting it to a specific range it was intended to bypass the blue lines. However, the blue lines would still contain dark pixels - so only segments of the line would be removed.

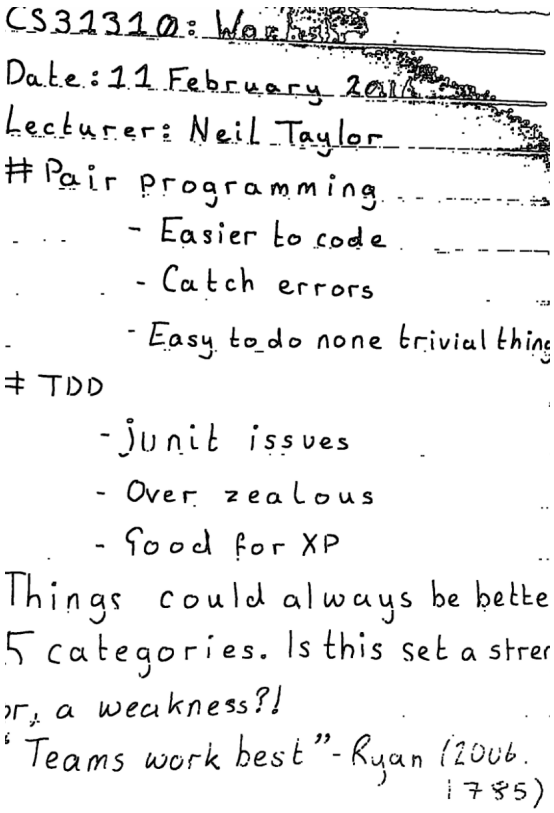


Figure 3.5: An example output from the algorithm 2. There is still significant amounts of noise in the image.

Different morphological operations were used on the image in an attempt to clear the noise pixels. Erosion operations were used, by passing a kernel over the image essentially removing small black noise pixels from the white background whilst expanding the darker pixels, enhancing the text on the page [45]. Dilation is essentially the opposite: a kernel is passed over the image, and the white background areas get larger and the black text gets thinner; this has the effect of removing the characters quality [45].

The result of the morphological operations ended up reducing the quality of the segmentation, as shown in Figure 3.5. This highlighted further iterations were needed for an improved output.

3.2.2 Only extracting the text

There was no simplistic way to identify and filter the lines, therefore it was decided that the text will just be extracted.

OpenCV has an example of line extraction and binarsation [6]. From the example, structuring elements were used to extract the text from the image. Further erosion and dilation were used to remove additional noise. Throughout the process, masks were used to transfer the state of the

image to another mask. An example is transferring the text to a mask, but had an unwanted side effect that line noise way transferred too.

Due to the text having connected pixels, unlike the eroded noise, then connected components were used to identify characters. As a result of morphological operations the lines were no longer connected. The identified characters were copied to a final mask.

Further smoothing cleared up the image. After a series of iterations the binarisation process was complete.

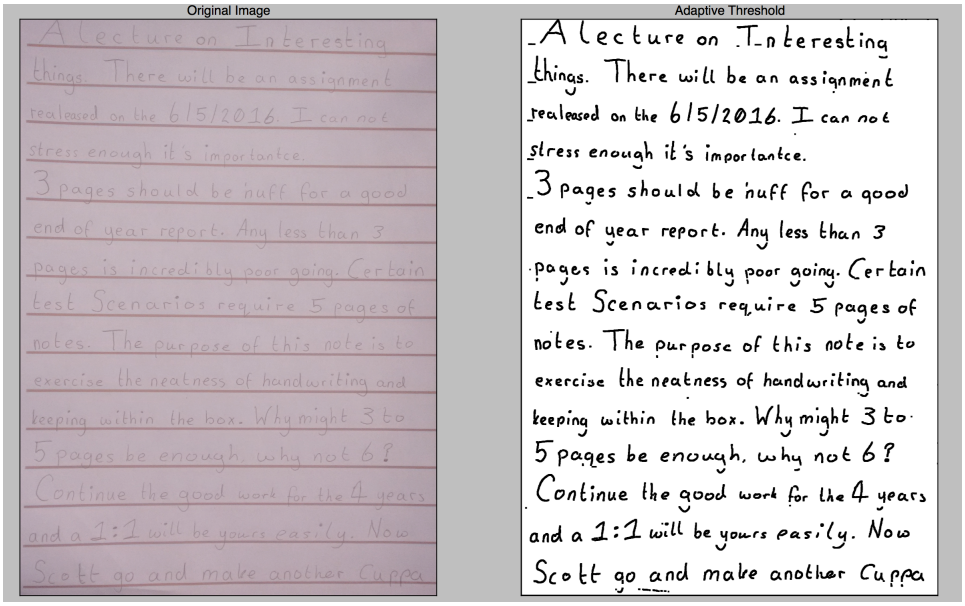


Figure 3.6: A poor quality image has been binarised successfully with little noise.

Figure 3.6 shows the result of the binarisation script. Adaptive threshold works well on the image, due to local thresholding not being affected by shadows. Images can be taken in non-uniform lighting and it will produce a fully binarised image. There were issues which affected the implementation such as changing to a bespoke lined paper. Overall the binarisation segmentation works well. Further examples can be found in Appendix F.

3.3 Handwriting training

The training of handwriting was a constant task through out the sprints. It was initially proving cumbersome in the early sprints. After the changes implemented from Section 3.1.1.2 the results from the handwriting recognition improved considerably.

3.3.1 Training process

Tesseract's training was a methodical process. Tesseract's GitHub wiki [71] and Gonzalez [23], provided great reference tools on how to train the data.

Firstly, as handwriting was being trained a new language would have to be created. Each train-

ing example has a specific format which must be adhered to: `lang.font.expNumber.tiff`. The file is then run through the Tesseract training process using `batch.nochop makebox` command, on the specific language `eng.ryan.exp2a`; this created a box file for the given training example. The box file contains the characters which Tesseract believes are in the image; each line is a new character as shown below:

```
S 155 2398 208 2487 0
3 242 2403 295 2485 0
9 320 2403 376 2476 0
1 405 2396 448 2467 0
1 467 2396 504 2462 0
0 520 2393 588 2455 0
: 604 2400 628 2451 0
```

The box files were too complex to analyse as it was not intuitive to see the identified characters without a graphical interface. Figure 3.7 shows the use of the jTessBoxEditor [67] tool to identify characters and their bounding boxes to overcome this issue.

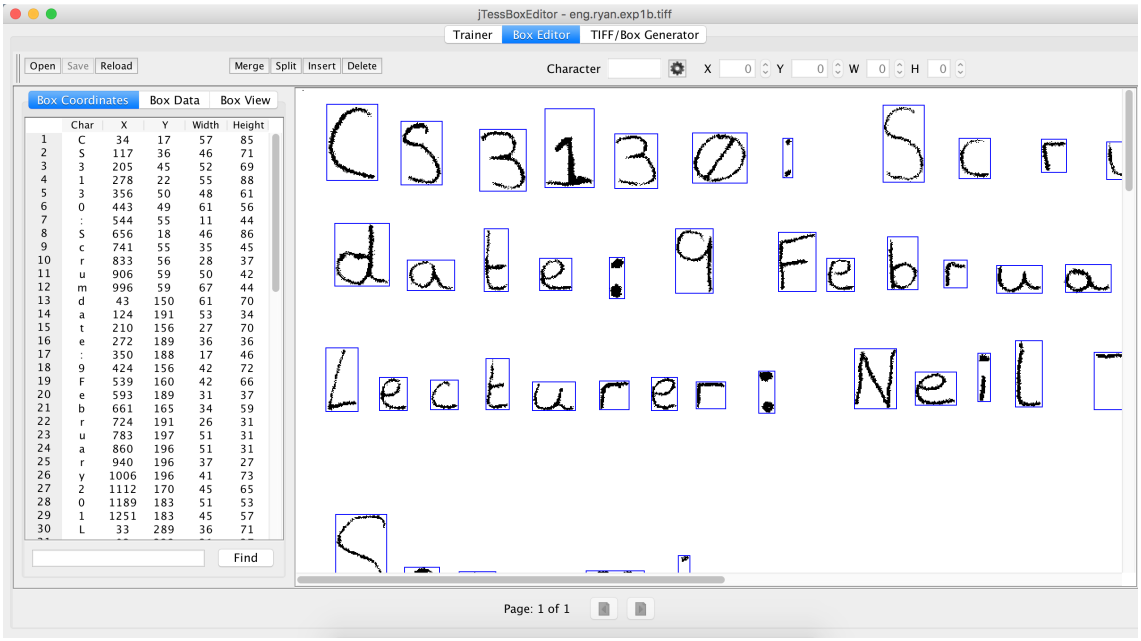


Figure 3.7: A example of the jTessBoxEditor being used identify characters in the tiff box file.

Once the characters had been manually changed, the box file was passed through Tesseract's training process. `tesseract <file> -l eng.ryan.exp2a box.train` would train the engine on the image's box file, for the language `eng.ryan.exp2a`. Often there were issues with being unable to identify tagged characters; these box file lines were deleted.

Following this process, Tesseract required an additional file (`unicharset`) to be able to extract all possible characters that is identified in the training examples.

Tesseract's GitHub repository states that clustering is an important process to extract "prototypes". The clustering commands aid in ensuring the shapes of the characters tagged and identified are known so they can be used as a reference again.

A frequent words file was created, from the `/usr/share/dict/words` directory, to help to identify common words. This would aid in improving the chances of detecting specific words. Common words could also be defined; "by" and "date" are examples of words in this file.

Finally, the `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 enabling new training data to use this language.

Reinforcement of the characters identified was needed, so further training examples were created. This was called "bootstrapping". Therefore when training on another example, if the language was set to `eng.ryan.exp2a` it would reinforce that specific language with new data.

Throughout the sprints issues were identified whilst training the data. Characters would often not be identified correctly on the image, with specific issues with the letter "g" identified as a 3. When a blob could not be identified at all, Tesseract would label it "∼"; these were ultimately removed when it was discovered that it would fail to identify them if edited. The training was conducted on twelve training examples, a selection can be found in Appendix 4.2.

3.4 Web application

The web application was the main part of the development and specific sections proved to be more complex than anticipated.

3.4.1 OAuth

The Google OAuth integration was more complex than first considered. Google suggested to use the Google client library [24] for all OAuth requests, to avoid security issues when making requests. Therefore, this library was utilised throughout the project.

The Google API client would, on occasion, raise peculiar errors. Whilst making a query to the calendar API it would raise the error, "rootURL", when using the build API call. This was mystifying as it was previously working. An issue was raised on the library's GitHub repository [28]. To confuse matters more, when querying the Google people plus API, it would work perfectly fine - however the Google calendar API would raise an exception. It eventually stopped throwing an exception, but the reason is unknown.

OAuth2 was implemented in the application so when the user clicks "authorise with Google" it will initialise the process for OAuth. Once the user accepts the use of selected services a secure JSON [1] credential file was returned containing specific tokens used when querying; these are appended to the user's session.

The credentials contain an expiration time, as shown in Appendix 5.3. When making a request this expiration time was checked, and if it was exceeded, then an error would be raised when querying the API's and displayed to the user. To overcome this, additional checks were made to ensure the credentials in the session were still valid. If they were not then it would redirect the user to the log-out route, clearing the session and asking the user to re-authenticate.

[1] JavaScript Object notation is stored in key-value pairs.

3.4.2 Reoccuring events

Reoccurring events within the Google Calendar integration, poised a lot of issues. It was identified in the pre-beta testing that if the user has a reoccurring event then it would not append the URL of the note to the event.

When a query was made for a list of events if there were reoccurring events then it would group the events by the first instance of these events. This proved to be problematic as it would display to the user that the note was taken on the 12th March, for example, but there were events from February being shown.

To account for this, the structure of the response was analysed and it was identified that grouped events had a `reoccurrenceID` key. After finding that the calendar API can fetch reoccurring items, a query was made using this ID key - filtering by the start and end date from the initial query. A check was included to ensure that when displaying to the user the event did not contain the `reoccurrenceID` key ensuring the singular events were displayed to the user.

Editing a reoccurring event produced further unexpected behaviour: when the event had been successfully modified and the note URL had been appended, it returned both the grouped event and the modified edited event in the initial query. Google must classify that changed reoccurring events as new instances. Instead of displaying more duplicated events to the user, a check for the `recurringEventId` key, which was present in the modified event, was conducted; if it was present the event was omitted.

Another issue identified were all-day events. All-day events do not have a datetime key in their start response field, returned from the Google Calendar API. This would cause the application to raise an exception and prevent the user from adding a note. A check was made to make sure that the datetime key was present.

3.4.3 Tesseract confidence

Displaying the Tesseract data went through a series of iterations over the latter sprints.

At a basic level integrating Tesseract into the web application was fairly simple and was implemented around sprint eight. The binarisation script, see Section 3.1, was incorporated to the application. This was added to the file upload route, as the user's image needs to be binarised when it is uploaded. A Tesseract wrapper could have been implemented but due to time constraints a 3rd party library, tesserocr [3], was used.

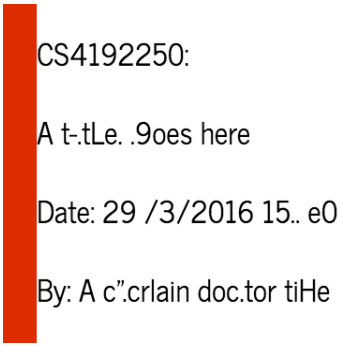
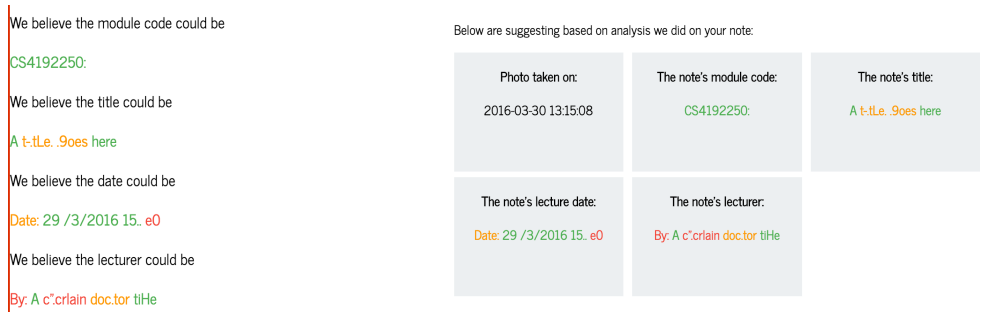


Figure 3.8: Tesseract being integrated into the application at a very basic level

Tesserocr is a Python implementation of Tesseract's C++ API. Tesserocr uses textlines to extract the text from the uploaded image. The first three lines were then iterated over, identifying the box for the lines so that text could be extracted. Each of these lines mapped the words identified and the confidence score for each word on a scale of 0 - 100 (0 being uncertain and 100 being certain).

The module code, lecturer, location and date were extracted via list-comprehensions, matching metadata structure defined in Section 2.3. Modifications to the confidence scores were attempted in the controller to replace with a class name for the colours - so that the view file could render the content easily. Problems were encountered when the API returned tuples, an immutable type, so modification was not as eloquent as originally envisaged, therefore numbers remained.

Conditional checks were made in the view files on the confidence score; 75 would be green text, less than 74 and greater than 70 would be orange text and below 65 would be red text. Figure 3.9 shows the resulting output from the conditional checks. There are anomalies, with "Date" for example, it is orange but in-fact it is extracted perfectly.



(a) (b)

Figure 3.9: Coloured representation of the confidence of the words from the handwriting: a) shows the initial steps with the image. b) shows the resulting output after styling of the web application

3.4.4 Parsing EXIF data

EXIF data parsing would be an important section of the application. EXIF data is essentially metadata about an image [49]. When a user uploads a note, it analyses the image for the EXIF metadata; it uses the date of the image taken to query Google Calendar for additional events.

Throughout the sprints the EXIF data parsing was extended to allow for a greater variation in images uploaded. Python's image library [36] was used to parse the data. Further additional checks were made to ensure that the images were either JPEG or TIFF as they only contain the metadata.

During user-testing issues arose where a participant could not upload their note successfully. The image was formatted a JPEG but the mobile phone photo did not contain the "dateTime" EXIF key. Checks were implemented to ensure that this key existed.

3.4.5 Displaying calendar events

Over several user stories displaying different events around the application were created. The first instance of displaying the events were incorporated into the homepage, displaying the last seven days worth of events from the user, shown in Figure 3.10. This was simple to implement but provided a strong foothold into the interactions with the Google Calendar API; this stretched from the application to the testing infrastructure.

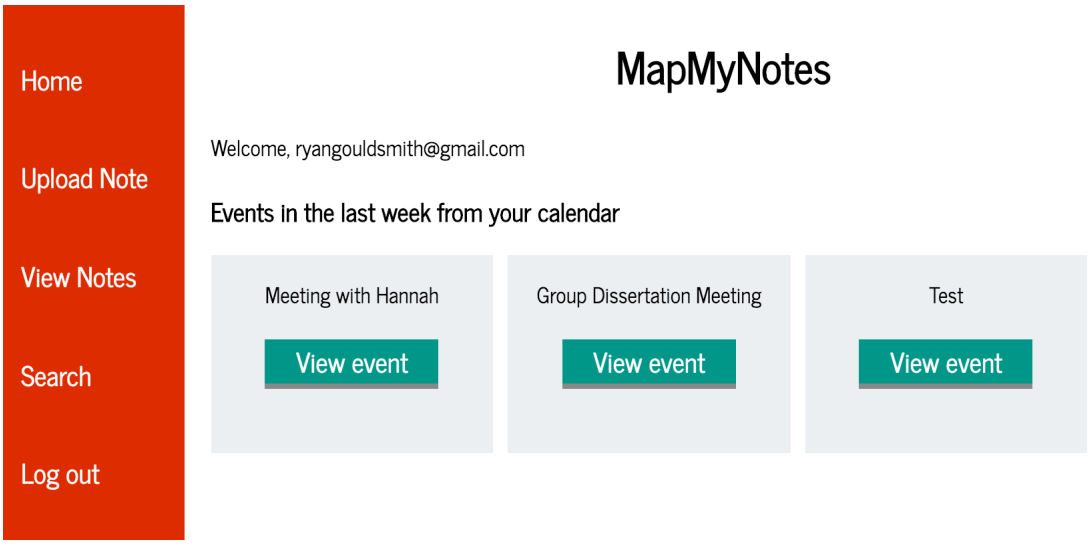


Figure 3.10: An example of displaying the events from the last week from the user's calendar

One issue which arose was the change in timezones to British summer time during the project. If there was an event starting at 14:00, and the user queried for events at 14:00, then it would not return any results from the Google Calendar. Upon closer inspection if the query was for 13:00 then it would return the correct event. This issue rendered the application to a halt whilst this could be fixed: eventually, the timezone was appended to the user's input and querying with the timezone included returned the correct event from Google.

3.4.6 Editing calendar events

The user story for adding a note was implemented and as part of the tasks for this story, the note URL must be saved in the calendar event. When the user enters the date into the form, a query would be made to the Google Calendar API to return all associated events from that given day. Checks were then performed to ensure that the module code and the summary of the event matched.

This poised the problem of being able to add 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 confusion as to which event to add to. Over the next iteration of development, the calendar events were additionally validated against the start date from the event and the user's input.

One issue which arose when adding the note's URL to the description field of the calendar was that it would replace the original content inside the description. This is a major concern for the

users of the application as it would overwrite any data. Another issue it created was that multiple notes could not be attached to a given event. This was changed so that it would append to the description field, rather than overwriting it.

The algorithm for adding to a calendar is stated below.

Algorithm 2 Adding a note URL to the calendar

```
1: Create a calendar service object
2: Prepare URL from saved note
3: Build the HTTP request
4: Find an event for that day from the start time as given
5: Parse the events
6: Check to see if the summary contains module code AND the start date time matches
7: if contains module code then
8:     check the response to see if description includes URL
9:     Save URL to the notes attribute in the database
10: end if
```

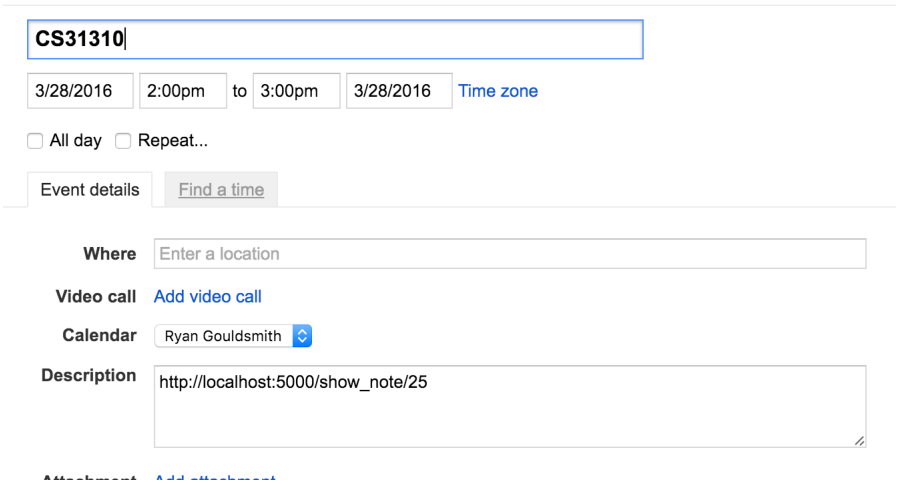


Figure 3.11: Saving a note correctly to a calendar event item

Figure 3.11 shows the output from saving a note to a specific date and appending the note's URL to the description field.

3.4.6.1 Editing a note

The user story "editing a note" was established midway through the project, an example task included when editing a note's start date it should be reflected in the user's calendar. If the user changes their date then it should remove the URL from the old calendar item and append to a new one, if the event exists. This proved to be a complicated implementation.

When a user edits a note it would query the API to return the event for the given note, based on the time start which was persisted for the note's relation. This event was then modified by replacing the description field with an empty string, replacing all the content. This caused issues

regarding the data stored by the user being deleted arbitrarily. As a result a find and replace was used on the description field to remove any strings which matched the URL.

It is worth acknowledging at this point in the development considerable aspects of the codebase was refactored. There was duplicated functionality spread across multiple blueprints, making the codebase obfuscated. As a result the design decision for the `GoogleServicesHelper` emerged to abstract the duplication.

3.4.6.2 Logging in and out

Enabling the system to have users was an emerging user story midway through the project. As the log in would be deferred to Google, then the user would need to connect to the service. When the log in process has been completed a user's email address is extracted from the Google Plus API response and created in the database.

Using the application it was noticed that multiple user's were being created for a single email address. To reduce this problem a helper function was implemented which would find a user from their email address, if the user could not be found then they were added to the relation.

The user's ID is then appended to the session. Every page on the application verifies that this key exists. Once a user had finished using the application, the log-out route would destroy the user's session.

Delegating the responsibility to Google was a good design decision; the security which Google have would have been better than what could be implemented by the author. Furthermore, ethically, the only person information the system uses from a user is their email address.

3.5 Travis

Although not strictly a coding implementation, Travis formed a core part of the application and issues were found whilst using Travis.

Firstly, at the start of the process extensive time was invested into trying to auto-deploy from Travis. Whilst giving detailed instructions on how to deploy to 3rd party systems the documentation for deployment to a virtual private server was sparse. Over several sprints the auto-deploy pipeline was desired but due to the lack of documentation there was no auto-deployment from Travis to a server.

When integrating Tesseract with the application, it became apparent of another issue with Travis: Tesseract and OpenCV had to be both built from source. TesserOCR uses Tesseract 3.04, and at the time of writing, Ubuntu's latest package is 3.02; this is the same for OpenCV 3.0.0. As a result, the build time for Travis was increased exponentially. The current build time was around thirty minutes; caching was investigated but no suitable solution was identified.

Chapter 4

Testing

This chapter discusses the testing strategy which has been implemented on the project. This includes unit, integration acceptance and user testing utilised throughout the application. Additional testing strategies for the image segmentation and Tesseract training will be discussed.

4.1 Overall approach to testing

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

4.1.1 Test-driven-development

TDD aims to ensure that tests are considered 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 there being no implementation code to pass the test. The following steps would be to ensure that the tests pass by adding the associated code. After the test passes refactoring is conducted to ensure that design is kept as simple and as clean as possible for the current implementation.

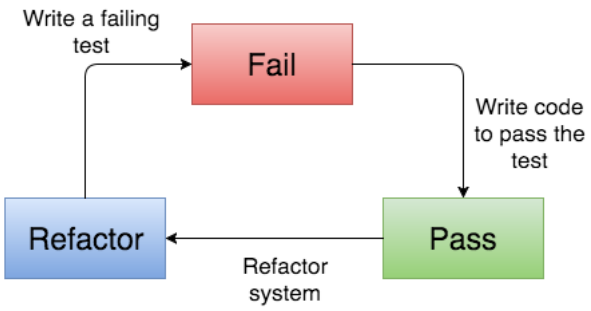


Figure 4.1: The cycle of TDD during the development stages of the application.

Whilst adopting the TDD process there were two methods which could have been adopted: a group of tests were created for a feature, or one test for one singular bit of functionality and tests are iteratively created. The latter approach was used to ensure that focus was only kept on the smallest aspect of the system.

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 repeated per feature.

4.2 Automated testing

It is worth noting that Flask's testing documentation is very sparse and not comprehensive on how to test a system fully.

During the first few sprints of testing, pytest [51] was originally being used to create test classes, with test classes sub-classing `unittest.TestCase`.

Flask tests were refactored midway through the sprints to use Flask-testing [31]. This offered better testing support for Flask applications, allowing the creation of a fake application and providing the functionality to run a live server for testing. This live server would be needed for the acceptance testing.

4.3 Mocking tests

The purpose of mocking is to change the output of a function to a value which is returned every time the test is called [19]. It was established that certain tests would need to be mocked, as the data returned would alter after every call. It was identified that *all* interactions with Google API's, any interactions with Tesseract and the Session would need to be mocked.

Dale [10] discusses writing tests and the need for mocks when external factors are out of the developer's control. As the Google API does not support specific environment API's, such as production or development, then all URLs are linking directly a production URL. Each test should be tested in isolation and each test should be independent of these external factors. For example, the test may query the API once and pass the test, however on the next query it may fail due to a different response; this case 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, Tesseract would output a different response for a test image. This meant that the functions which interact with the image would need to be mocked, to return consistent results for every test.

Whilst establishing how mocks worked in Python there was a lot of duplication when mocking different classes. The library mock [18] was used for annotations, proving an annotation style syntax above test functions.

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
```

Listings 4.1 shows the syntax which was initially used during the mocking tests. This would result in many of the tests becoming unreadable and obfuscated. Additionally the Do not Repeat Yourself (DRY) principle was violated by duplicating large amounts of the testing codebase.

Listing 4.2: Mocks using the patch and start. It stops in the dear downs

```
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()
```

Looking for a more succinct solution in the mock API uncovered the option to patch object calls.By refactoring the test codebase to use patch calls, instead of annotations, the duplication on-top of every test function was eradicated. Initially, it was not entirely clear how to implement the patch calls into the testing system. It was eventually discovered that they were included in the `setUp` [1] and `tearDown` [2]. An example is shown in 4.2.

As the development increased, so did the need for further mocking enhancements. During integration testing especially, multiple functions were called most than once. In order to mock a series of return values then the "side effects" were implemented from the mock library.

```
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()
```

Listings 4.3 displays an example use of the "side effect" API. In the example if first outputs a google response, then when `execute_request` is called for a second time a new event response is returned and so on.

Another example of where "side effects" were used was with the Google Calendar. During

[1] A function which is run before each test has been run.
[2] A function is run after each test has completed.

the integration tests, there were times which getting events from the calendar were called more than once; it would first get a list of events and a singular event. As the same function was called multiple times to execute the request the "side effects" were needed; multiple JSON files were returned from the functions in the test. Examples of mocking data used for the Google integration can be found in Appendix E.

Mocking was heavily featured in the testing. As it was not considered during the initial design phase it took longer than expected to overcome the issues. Most of the testing codebase was massively refactored in the light of the change of mocking style.

4.3.1 Unit testing

Unit testing aims to isolate other interactions with the class and focus the testing on a specific function to ensure that the correct data value or operation is being performed. During the testing stage this was one of the core testing strategies.

When testing the individual functions, there was often database transactions being tested. A live database would not be ideal to test against, therefore there needed to be a test database. In each of the tests `setUp` functions, the configuration was overwritten to use a test SQLite database. To fully ensure that previous tests were truly independently tested then the database was dropped and created before every test.

Descriptive test names were the aim, to help to show a high-level of documentation in the system. With a well tested system, and detailed tests then it is easy to see what the system does and does not do.

```
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 a selection of unit tests for the user class. Each of the tests considered one function in the user class to implement, whilst edge-case tests were considered too.

The testing as a whole could be considered part of the design of the application: after the story was broken down into a series of tasks, and the analysis of CRC cards was completed each function was written as a test. Whilst doing the unit tests, the core design decisions on what would be returned and what is expected of a function was decided - as well as descriptive function names. Each test would compare the output of the function with the expectation of the output.

Over the course of the project, the unit tests evolved and so did the model classes. The unit tests were the core foundations for the application to be built upon. With adopting this approach, there are times which the unit tests need to be refactored to reflect the new design. Therefore, by keeping up to date with the design it would act as a form documentation for the system.

An excerpt of unit tests can be found in Appendix C, Section 3.1.

4.3.2 Integration testing

As the application made use of routing then tests to ensure the correct response codes were being returned was important. When breaking down the story, it was identified if there needed to be new routes tested or if the controller would change its functionality. If so these were the first tests written for the routing sections, and implemented from the design considerations in Section 2.1.4.3.

The integration tests were important as it was testing the model components and data being brought together. The tests ensured that the different systems were compatible and performed the correct operations. Routing tests consisted of checking that the response codes were correct, any redirects were correctly redirected.

An example integration test from the activity diagram shown in Section 2.1.3, would be: once a POST request has been made to add note URL did the note correctly get saved with the associated metadata. This test would check for the persistence of the note, and that both data can be sent to the route and it had a specific outcome.

Appendix C Section 3.3 shows a selection of integration tests.

4.3.3 Handling sessions

Session handling was a lot more complex than first anticipated. Throughout the application sessions are used to handle server-side states of the system: i.e, is the user logged in.

When testing the session it would attempt to be tested in an isolated environment, like the unit tests. Integration tests threw a lot of errors when testing with sessions. For example, if a route was accessed but the session was not set then the test would error. To overcome this, sessions needed to be modified in the tests. As the integration tests were using Flask's `test_client` context, then modifying the sessions was a little easier. Flask had a solution to testing the session handling [17].

Listing 4.3: 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 4.3, 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. The acceptance tests would not acknowledge that the session had been modified, like in Figure 4.3, causing the tests to error. As there was a lack of documentation it was decided that the session helper would be mocked. By mocking the responses in the `create_app` function, it enabled the sessions to be modified, allowing the acceptance tests to be run.

4.4 Acceptance testing

Acceptance tests were created to check that the correct output was displayed for using the system. These tests ensured that the system integrated together, rendering the correct content and executing the correct operations.

Each user story was broken down and throughout the testing process the acceptance tests would be the final tests added to check that the functionality was completely integrated. Selenium for Python [42] was used as the acceptance tool for interacting with the web page. By interacting with the document object model (DOM) it was able to test any dynamic HTML rendered.

Before the tests were written a browser type was selected for Selenium; Chrome, Firefox, or a headless browser (via phantomJS) could be selected. The headless browser was selected as it can perform the same interactions but it does not display a graphical UI, and is a little quicker [52].

Acceptance tests were created to extend the `LiveServerClass` from the Flask-Testing package. This differs from the Integration and Unit tests as those tests subclass `TestClass`. Using the `LiveServerClass` creates a server instance so Selenium can access it easily. This removes the need for an external Selenium sever.

After the unit tests were written for a feature, the acceptance tests were added to ensure the systems combined together successfully. An example of an acceptance test from the user-story, search for a note is as follows:

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”

Acceptance are unlike any of the other testing techniques used in the application; rather than testing for underlying functionality the principle is to test the content to the user is displayed correctly. The tests are evaluated against the content displayed.

Another example of Selenium tests being beneficial was checking the output colour from Tesseract’s confidence. The logic in the view file determined the colour and Selenium was able to confirm the logic was correct.

```
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.

In Figure 4.3 it shows that the time to run to five 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 a selection of acceptance tests refer to Appendix C, Section 3.2.

Overall, the acceptance tests were incrementally developed helping to aid the design of the frond-end through a series of tests and they proved an important tool when testing.

4.5 User Testing

As the application was aiming to solve a problem, a set of potential users were asked to perform a user study of the application. Their responses was 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 informal 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 and changed from areas of the design which were potentially over looked.

Figure 4.4: A pie chart from the Google forms questionnaire [26] that the users conducted showing that they would not use the application for archiving their notes.

An interesting reflection from the user study was that participants 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, so the application would not benefit them.

There were positives to come out of the user testing where users agreed that it was simple to use and easy to navigate around and the websites presentation was well received. See Appendix **??** for user study results.

4.6 Tesseract testing

As Tesseract was a training process no additional code was written for this. An analysis of how well Tesseract learnt as it progressed through the training process can be represented.

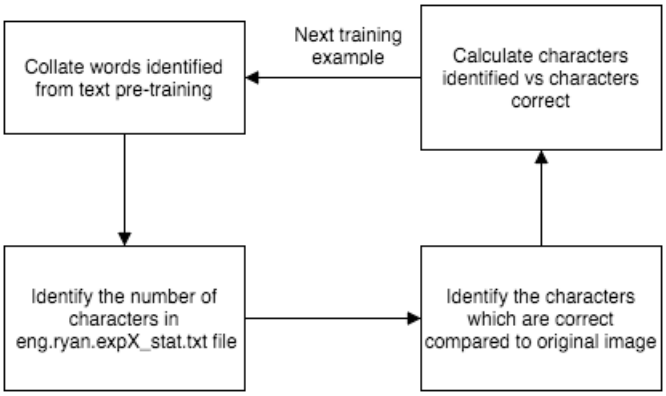


Figure 4.5: A simple framework showing the steps of analysing each of the training examples for a statistical measure for how successful the training process was.

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

Before the Tesseract training can be analysed it is worth acknowledging the test results from the spike work conducted at the start of the project which compares different thresholding algorithms on which one was suitable.

Image pre-processing Spike work - Correct Results vs Characters on the page			
Paper type	Greyscale	Original	Threshold
Blue-Lined	6.75%	13.5%	28.0%
Lined	0%	25%	13.5%
Plain	0%	0%	23%

Table 4.1: Table showing the results of non trained handwriting on different adaptive thresholding algorithms.

Table D.3 shows the results from the experiments conducted to analyse which thresholding algorithm works efficiently. The result clearly show that the thresholding image yielded the best accuracy. As a result, thresholding was used. The rest of the section discusses the Tesseract training performance on adaptive threshold with Gaussian. For further statistics on this test see Appendix 4.3.

Figure 4.6 shows the output analysed from the Tesseract training using the segmentation algorithm discussed in Section 3.1. 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 D, section 4.1 for the full statistics.

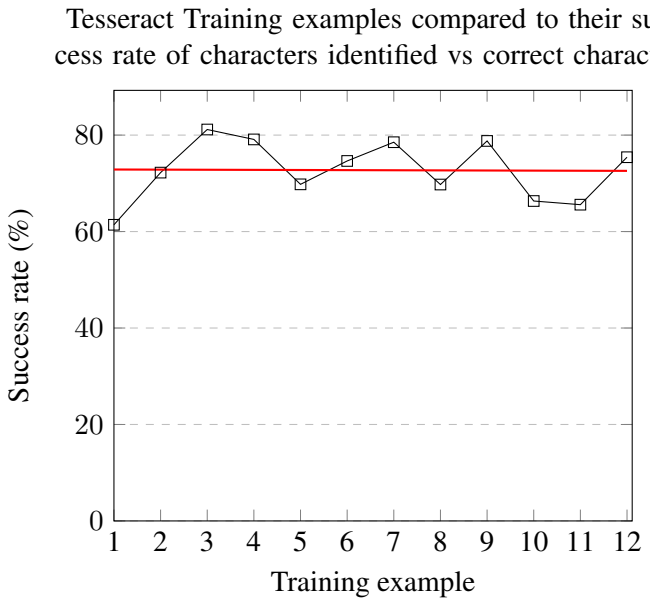


Figure 4.6: A line-graph showing the success rate of the Tesseract training results over 12 examples. The trend line shows an almost horizontal linear line.

4.7 Image threshold testing

A methodical testing approach would not appropriate for the segmentation script development, as it was predominantly spike work.

The testing was conducted at more of a visual level, checking the output of the image to see if the image has been binarised successfully. Once the script had been developed to a suitable level then the spike work would stop and testing would begin.

The code was re-written following a TDD approach, although a lot of the code was interacting with OpenCV it was assumed that these functions had been reliably tested. Nevertheless, testing for the script was produced and checks such as checking for black pixels in the image was written.

For examples of the test images used for the image threshold used as training examples for the Tesseract engine, see Appendix 4.2.

Chapter 5

Evaluation

This chapter will evaluate the project as a whole. It will evaluate the requirements, any design decisions made, the good features of the project and areas which could be improved.

5.1 Correctly identified requirements

To recap, the following high-level requirements and objectives were identified in Section 1.2.1:

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 full 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.

These were the classified as the minimum requirements needed to complete the application. At the start of the project it was made aware that completing these requirements would be a large challenge. Dr Hannah Dee had reservations about integrating the handwriting recognition, but it was added to make the application stand out from a simple web application.

The investigation and prototype work into extracting handwriting text took longer than expected. This was partly due to turning images to grey-scale did not improve the recognition rate.

The requirement took a diversion down a prototyping and almost research approach into the different binarisation techniques available. As this emerged as being an important part of the project, multiple sprints were dedicated to ensuring the image was correctly binarised.

The author's handwriting was successfully trained on twelve different notes. From Figure **??** is was clear that no further examples were improving the recognition rate. The training returned around a 72% recognition rate for the author's handwriting. As it was not defined what the ideal recognition rate would be at the start of the project, then this is a very rewarding return.

Although it was not ideal it was acknowledged that the notes must have some structure. Although a little restrictive for the end user, which could have affected user experience, it was essential that this was conducted.

The application was the main crux of the project, so a lot of effort was exerted in an attempt to implement the functionality from the requirements. Although an agile approach was adopted the high-level requirements did not change. There were a series of smaller requirements as the application grew - such as selecting the Tesseract data. Additionally, the metadata used in the application expanded on further investigation to include the title, location and date. The application was extended to include the searching and viewing all notes for the user.

At a basic level the handwriting recognition was integrated into the web application, parsing the first three lines. This was implemented on the requirements to train on the author's handwriting and the binarisation of the uploaded image. An aspect which was iteratively developed was the Tesseract confidence - where the colour was outputted. This was not initially in the requirements, but it was an emerging one throughout the iterations. This requirement ended up having design implications.

Potentially the most structurally complex requirement to implement was the calendar integration. Initially it was only intended to save the URL to the event. This requirement expanded and went into a variety of directions which included: showing the last seven days of events, and editing the note would remove the URL from the event.

As the application was developed in an agile approach not all of the requirements could be truly identified at the start - so the calendar integration task is an example of user stories being appended to the backlog as the project progresses. Therefore the initial requirements provided a good opportunity to discuss if there were any lower-level requirements. Nevertheless, as the project was developed all the initial requirements were completed throughout a diligent process coupled with extensive testing on each feature.

5.2 Design decisions

The design decisions, see Section **??**, were a core part of the application process. The design to use an MVC approach was one of the most important design decisions made on the project. It enabled the code to become decoupled, which produces a modular system, greatly enhancing the readability of the application.

The entity-relation diagram has been well considered where each of the relations aims to reduce data-redundancy by being normalised. Although the design is clear, the image path from the `Note` relation could be extracted to its own relation, and form a 1-1 dependency with the Note.

The overall architecture of the system was well considered at the start of each feature. The

resultant class diagram shows that there is low coupling but high cohesion within the models, obeying to good object oriented principles. Overall, the class diagram shows an overall good design of the system. Although the design is simple, there are sections of the classes which would be refactored slightly if given the chance. In the `BinariseImage` class there are several functions which would be better as static functions due to them not interacting with class properties. This would improve consistency amongst other classes where static methods have been included. Furthermore, the `OAuth` classes would be refactored to ensure that the `http_auth` variable was not used as a parameter; this was implemented to help with mocking for the tests. The use of helpers and services throughout the design have helped to improve the readability and using these as a proxy reduced the amount of code duplication, resulting in a cleaner implementation.

Strictly not related design, but the PEP8 standard [**?**] was adhered to from the start of the process. Partly due to the inexperience with Python the standardisation of the implementation was over looked. Identifying this at the beginning of the project would have not resulted in a large refactor of the application.

Overall the design throughout the project has been kept as simple as possible, which is a credit to the process, resulting in an intuitive design which has emerged. An upfront design would not have identified the need for the service helpers as this was implemented midway through the project.

5.3 Use of tools

This section will evaluate the key tools used in the application and justify their usefulness.

5.3.1 Flask

As the application increased in complexity there were times when the micro-framework, Flask, was thought to an incorrect design decision. On the surface Flask seemed to provide sufficient documentation, but beyond the simple applications Flask lacked in documentation. This would often increase the complexity of a task. An example of the over complexity is handling multiple configuration files. This was a lot harder than expected, as there was no support given from the framework for this basic functionality.

Additional libraries were often used to accommodate for the lack of functionality Flask offered. Cross site request forgery (CSRF) [1], is a big security concern but Flask did not offer any protection against this attack. The third party library, SeaSurf [CITE], had to be installed to offer protection in the application.

On reflection Flask did offer a flexible approach to the structuring of the application. Blueprints were useful when modularising the system as it enforced a modular design. For the most part Flask was a suitable choice, but the lack of testing documentation and support the application gave to developers was a little poor.

[1]

5.3.2 OpenCV

Using OpenCV for the pre-processing stage was an important design change through the project, instead of using ImageMagick. By changing to use OpenCV a superior segmentation algorithm was created for the image. Potentially the Tesseract training process would not have reached the success it has reached without the use of OpenCV. This did result in further work, but in doing so it helped to produce a better application overall.

5.3.3 PostgreSQL

PostgreSQL provided all the support that was needed in the application. In the design Section **??**, PostgreSQL was evaluated against MySQL and in hindsight the justification for using PostgreSQL instead of MySQL did not end up being used in the application. The more advanced queries and data types were not utilised, therefore using MySQL would also have been suitable in the application.

5.3.4 Google Calendar

Google Calendar was a sensible design decision. Although there were unexpected complexities, there was an abundance of support for the tool. Even when there were issues and support requests were made then the developers aided to help. Integrating the calendar into the application makes the application feel like a complete application. The only negative regarding Google Calendar be down to testing: it over complicated the design of the tests and a lot of time was lost testing this.

5.3.5 Continuous integration tool

Travis was a superb tool used throughout the development process. Testing in an isolated environment ensured that the application was working as intended. Chosen over Jenkins for the easy set up ended up was justified. However, there were disadvantages to using Travis the build times are *slow*. As it has to compile OpenCV and Tesseract from source for every build when code is committed, then build times are around 25 minutes. This stopped development for 25 minutes every time a feature had been completed, which slowed down the development time.

5.4 Meeting the users needs

A key feature of the application would be that the user can digitise their notes easily. A limitation in user testing was that there was not a wide user study, only two students tested the application. Regardless, the feedback which was received was that the system was intuitive and simple to use, but it did not fit with their process of note-taking - so they would not find it useful.

A true indication of meeting the users needs can not be established off two studies. However, the feedback which was received was that the application works well and does solve a problem, but due to the way different people take their notes it may not be for everyone.

5.5 Limitations of the project

Due to the time constraints of the project specific sections of the application could not be developed fully.

One limitation which impacts the quality of the characters identified is that the image has to be correctly rotated to 90 °, and cropped sufficiently. When a user uploads from their cameras they would have to manually ensure the image is correctly oriented and cropped sufficiently - this is not ideal for user experience.

The system only handles handwritten text from the author. Tesseract was not generic enough to analyse handwriting generally. This reduces the extensibility of the application as other users can not use this application without training their handwriting first. Additionally, it only analyses non-cursive handwriting not cursive and in general most people write in a cursively.

A known constraint is that the user has to write their notes in a specific format: the metadata has to be specified in a specific way to ensure that the metadata can be extracted reliably every time. Furthermore, the application only works with purely text notes not one which contains images - this reduces the type of notes which a user could work with.

5.6 Further enhancements

Although the application produced is to a high quality there are sections which could be improved further.

5.6.1 Handwriting training

A major improvement would be to improve the handwriting recognition for multiple users. Figure **??** shows that three training examples were user would be enough to train the application for a new user. This would enable more users to use the application. This could be incorporated into the web application as a setting section where the user uploads three handwritten notes.

Another feature which would be useful is extending Tesseract's popular words list with domain specific words when a user uploads an image. This would make common words from the user to be found more easily.

5.6.2 Image processing

When a user uploads their notes it would have been beneficial to auto-crop the image. This would remove the background from the note, leaving just the text. By enabling this feature the user would not have to crop prior to uploading creating a better user experience. It was identified in the analysis that existing systems implement this functionality.

Extracting images would have been the next logical step for the direction of the application. This would allow a user to draw an item on a note knowing that this would be extracted from the uploaded note and displayed on the canvas.

5.6.3 Web application

The application could be enhanced to display more recognised text. A WYSIWYG editor would eventually be the aim of the application. This would give the user full control over the content which can be added to the note. More users may be interested in using the application when a feature which gives the user more control over the note's content to add hyperlinks, for example.

5.7 Evaluating the process

A software methodology has been followed with diligence and precision through throughout the project. A Scrum approach has been fully adopted and which aided significantly during the design, development and testing. The weekly sprints helped to deconstruct the tasks ensuring that only the main priority tasks were completed. The burndown charts during the sprints were a useful representation of progress throughout the project, identifying problematic days.

The use of TDD was an imperative process followed throughout the project. The system has been developed with an extensive testing infrastructure behind the development. This was professionally and diligently followed even when implementation issues arose. As a result the system has 248 tests, which fully test all aspects of the system; this would not have been possible if it was not for TDD.

Overall, the process was stuck to thoroughly and meticulously. Scrum was fully adopted on the project and the structured guide to development made the development lifecycle a lot easier.

5.8 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 deciding if NoSQL would be useful. In hind-sight a NoSQL system would expand the application from not just a note-taking tool but one which could be used by the wider public. This would require a database which could accommodate for a variation in its structure.

Whilst training the handwriting data it would be imperative to keep a record of training statistics as each example was being trained. This would stop too many training examples being conducted which had marginal overall impact on the how well the characters were identified.

A different framework would be chosen at the start of the project. Flask lacked support, especially on testing, which slowed down the development time considerably. Therefore, a more mainstream Python framework, like Django, would be used at the framework of choice.

5.9 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.

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 segment an image 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.

5.10 Overall conclusions

Although there were some limitations and aspects of the application which could be changed it should not overlook the product produced. There are 3 core achievements in the project: the image processing, handwriting analysis and the web application.

Over the 15 week process a series of research and prototype work has been completed which can successfully binarise an image and remove any additional noise. The segmentation algorithm can take an image with non-uniform lighting conditions and still output a binarised image which clearly shows all the text on the page. This segmentation can now improve the Tesseract's handwriting recognition rate for handwritten text.

The analysis of handwritten text to return a 72% recognition rate is an achievement in itself. This was only achieved through the successes of the segmentation algorithm. Further analysis was conducted into the success rate which was achieved.

The web application was developed to a high standard, using solid software engineering processes. At a basic level the user can log-in and authorise via OAuth2 with Google's servers. Images can be uploaded, which are binarised using the segmentation algorithm and the first three lines of the handwritten note are extracted and presented as to the user. This step alone shows how the three different sections of the project integrate into one flow for the user to selection information from their note. After this further complexity was added to make additional HTTP requests to the user's calendar to add and edit specific events based on the date of the lecture.

At an in-depth level design patterns have been implemented to help to with the maintainability of the codebase. The application has succeeded in reducing the code duplication in the project by abstracting duplicated code into a series of helper classes. All of this is wrapped in a solid development methodology which has enabled the application to be iteratively developed gaining constant feedback from Dr Hannah Dee.

To conclude, a substantial project has been undertaken encompassing a variety of different aspects which all combined to produce an intuitive note-taking tool.

Appendices

Appendix A

Third-Party Code and Libraries

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

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

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

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

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

Appendix B

Ethics Submission

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

Appendix C

Testing Results

This appendix chapter shows the different sections of the application that has been tested and the test outcomes.

3.1 Unit tests

3.1.1 Binarise image

```
tests/test_acceptance_homepage.py::TestAcceptanceHomepage::test_once_authorised_it_displays_users_email_address PASSED
tests/test_acceptance_homepage.py::TestAcceptanceHomepage::test_should_display_the_correct_events_in_calendar PASSED
tests/test_acceptance_homepage.py::TestAcceptanceHomepage::test_signing_in_does_not_show_the_sign_in_button PASSED

============================================================================== 3 passed in 9.30 seconds ===============
```

Figure C.1: Acceptance test being conducted for the homepage, to ensure that the homepage displays the correct content.

3.2 Acceptance tests

The following section displays visual representation of the acceptance tests being executed, and their overall status.

3.2.1 Add meta-data

```
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_clicking_on_date_field_shows_datepicker PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_clicking_on_time_field_shows_timepicker PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_clicking_suggested_lecturer_from_tesseract_populates_lecture_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_clicking_suggested_module_code_from_tesseract_populates_module_code_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_clicking_suggested_title_from_tesseract_populates_title_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_ensure_the_fields_have_required_key PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_does_not_show_exif_data_if_image_is_a_png PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_exists PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_has_correct_url_action PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_has_date_of_lecturer_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_has_lecturer_name_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_has_location_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_has_module_field PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_has_title_exists PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_form_shows_exif_data_from_image PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_google_calendar_event_shows_when_exif_data_matches PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_google_calendar_response_without_a_date_time_field_ignores_the_response PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_module_field_label_content PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_module_field_label_exists PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_submit_button_exists PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_tesseract_data_is_coloured_correctly_for_confidence PASSED
tests/test_acceptance_meta_data_form.py::TestAcceptanceMetaDataForm::test_tesseract_data_shows_when_image_is_uploaded PASSED

========================================================================== 22 passed in 115.96 seconds ==========================================================
```

Figure C.2: Acceptance test being performed to ensure that meta-data can be added to the correct note.

3.2.2 Viewing all the notes

```
tests/test_acceptance_view_all_notes.py::TestAcceptanceShowNote::test_to_view_all_notes PASSED

============================================================================== 1 passed in 7.24 seconds ==
```

Figure C.3: Acceptance test being conducted to ensure that all the notes can be viewed.

3.3 Integration tests

3.3.1 Add and edit meta data

```
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_add_meta_data_route_get_request_not_allowed PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_add_meta_data_route_returns_302 PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_add_module_code_via_post_request_successfully PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_edit_route_upload_erroneous_date_format_returns_error PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_edit_route_upload_erroneous_time_format_returns_error PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_get_edit_note_information_returns_200_success PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_it_saves_a_note_object_once_the_meta_data_added PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_once_a_note_is_saved_it_redirects_to_show_note PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_post_to_edit_note_changes_the_foreign_key_association PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_post_to_edit_note_different_data_created_new_meta_data PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_post_with_already_existing_meta_data_should_return_instance PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_posting_exisiting_module_code_new_meta_data_new_instance PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_posting_redirects_back_to_show_note PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_uploading_empty_data_returns_error PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_uploading_erroneous_date_format_returns_error PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_uploading_erroneous_time_format_returns_error PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_using_the_different_module_code_should_save_new_code PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_using_the_same_module_code_as_before_if_one_exists PASSED
tests/test_integration_add_edit_meta_data.py::TestIntegrationAddEditMetaData::test_when_session_doesnt_contain_user_id_redirect_homepage PASSED

========================================================================== 19 passed in 6.17 seconds ==========================================================
```

Figure C.4: Integration tests carried on the add and edit meta url to ensure the system worked well together.

3.4 User study tests

Appendix D

Tesseract

This chapter shows the table outputting the results from the Tesseract training phase.

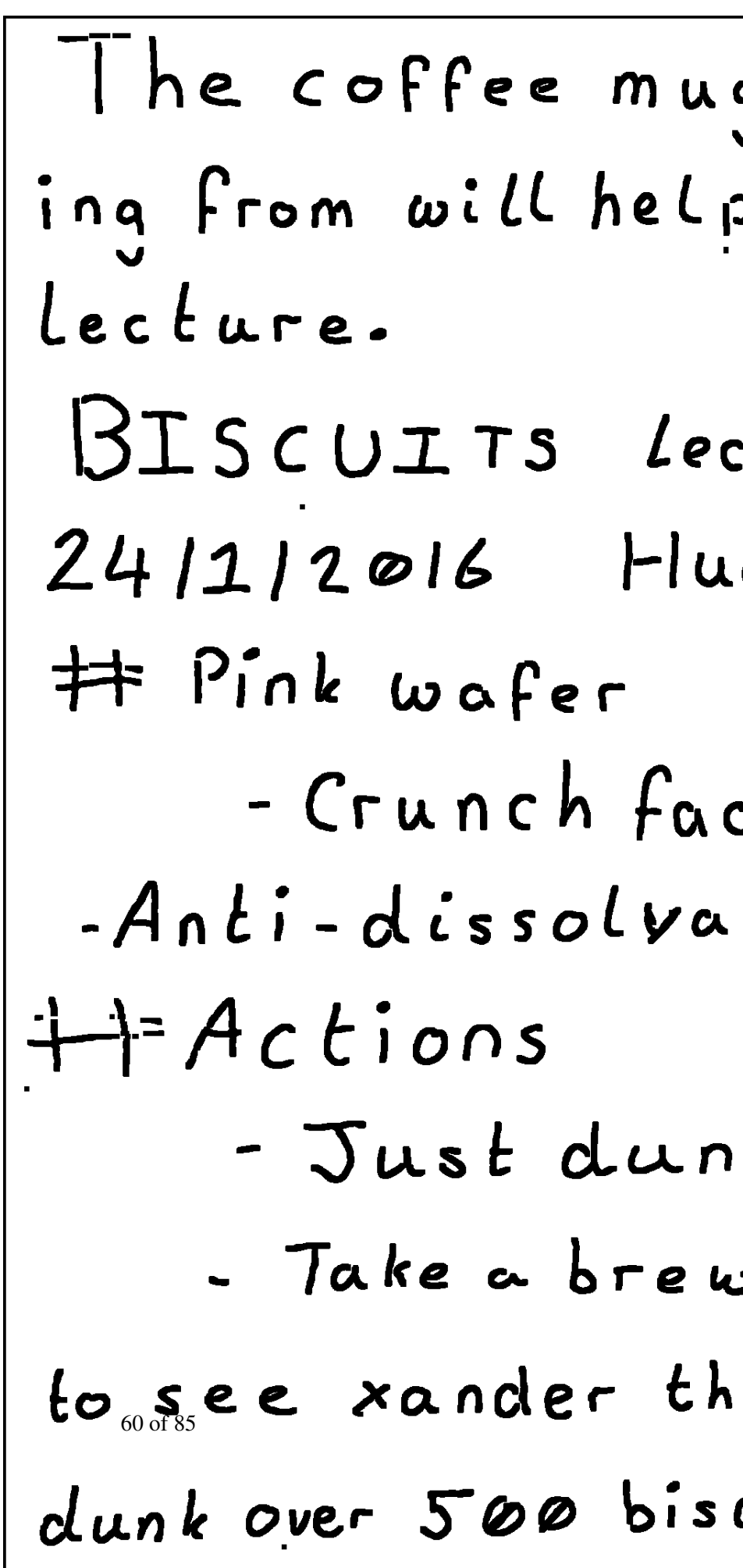
4.1 Tesseract data results

Experiment	Characters Identified	Characters Correct	Correct Percentage
1	114	70	61.40
2	252	182	72.22
3	345	280	81.15
4	335	265	79.10
5	288	201	69.79
6	276	206	74.63
7	326	256	78.52
8	400	279	69.75
9	462	364	78.78
10	401	266	66.33
11	366	240	65.57
12	362	273	75.41

Table D.1: A table which shows the statistics from the correctly identified characters during the training process.

4.2 Training examples

This section displays some of the training examples which are used in the image training process.
Further examples can be found under the `tesseract_training_data/adaptive_threshold_trainin`


The coffee mu
ing from will hel
lecture.
BISCUITS lec
24/1/2016 Hu
Pink wafer
- Crunch fac
-Anti-dissolva
Actions
- Just dun
- Take a brew
to see xander th
dunk over 500 bis

4.3 Pre-adaptive threshold results

Image pre-processing Spike work - Correctly identified characters			
Paper type	Greyscale	Original	Threshold
Blue-Lined	18	36	75
Lined	0	67	36
Plain	0	0	63

Table D.2: Table showing the results of correctly identified characters in an image over different paper styles and different image processing steps.

Image pre-processing Spike work - Detected Characters			
Paper type	Greyscale	Original	Threshold
Blue-Lined	85	119	157
Lined	19	261	1186
Plain	18	15	169

Table D.3: Table showing the results of the detected characters in an image over different paper styles and different image processing steps.

Appendix E

Example test data

5.1 Calendar week response mock


```
{
"accessRole": "owner",
"defaultReminders": [
  {
    "method": "email",
    "minutes": 30
  },
  {
    "method": "popup",
    "minutes": 30
  }
],
"etag": "\"1234567891012345\"",
"items": [
  {
    "kind": "calendar#event",
    "etag": "\"1234567891012345\"",
    "id": "ideventcalendaritem1",
    "status": "confirmed",
    "htmlLink": "https://www.google.com/calendar/event?testtest",
    "created": "2014-09-10T14:53:25.000Z",
    "updated": "2014-09-10T14:54:12.748Z",
    "summary": "Test Example",
    "creator": {
      "email": "test@gmail.com",
      "displayName": "Tester",
      "self": true
    },
    "organizer": {
      "email": "test@gmail.com",
      "displayName": "Test",
```

```
        "self": true
      },
      "start": {
        "dateTime": "2016-12-01T01:00:00+01:00"
      },
      "end": {
        "dateTime": "2016-12-01T02:30:00+01:00"
      },
      "transparency": "transparent",
      "visibility": "private",
      "iCalUID": "123456789@google.com",
      "sequence": 0,
      "guestsCanInviteOthers": false,
      "guestsCanSeeOtherGuests": false,
      "reminders": {
        "useDefault": true
      }
    }
  ],
  "kind": "calendar#events",
  "nextSyncToken": "synctokenasbebebe=",
  "summary": "test@gmail.com",
  "timeZone": "Europe/London",
  "updated": "2016-03-16T15:13:26.416Z"
}
```

5.2 Google plus response mock

```
{
  "tagline": "Some Dummy data taglone",
  "verified": "False",
  "circledByCount": 100,
  "objectType": "person",
  "emails": [
    {
      "type": "account",
      "value": "test@gmail.com"
    }
  ],
  "occupation": "A Test Occupation"
}
```

5.3 Google Oauth response

An excerpt from the oauth response

```
{
  "access_token": "foo",
```

```
  "expires_in": 10,
  "refresh_token": "refresh"
}
```

Appendix F

Image Processing

6.1 Pre-blue lined image

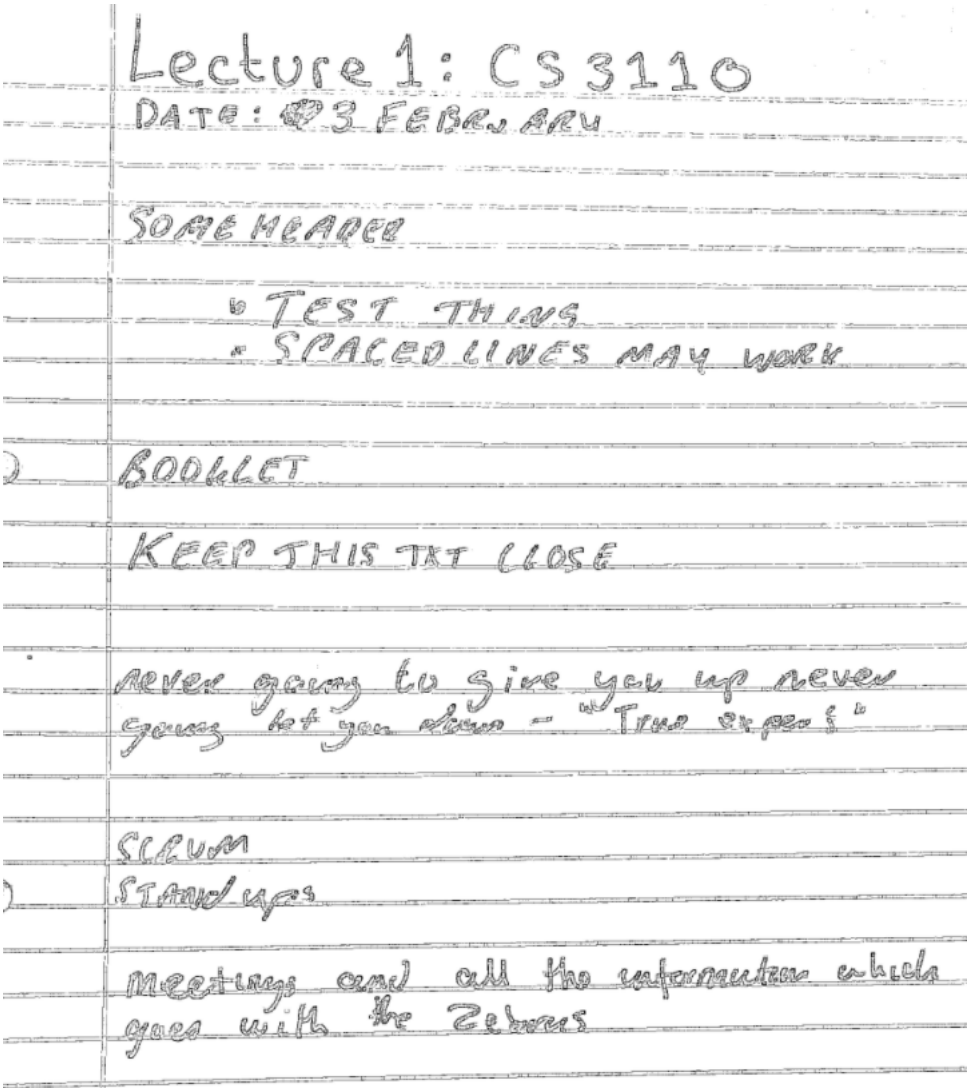


Figure F.1: The adaptive threshold on normal lined paper caused too much noise to be interfered with the Tesseract engine

6.2 Filtering the blue lines

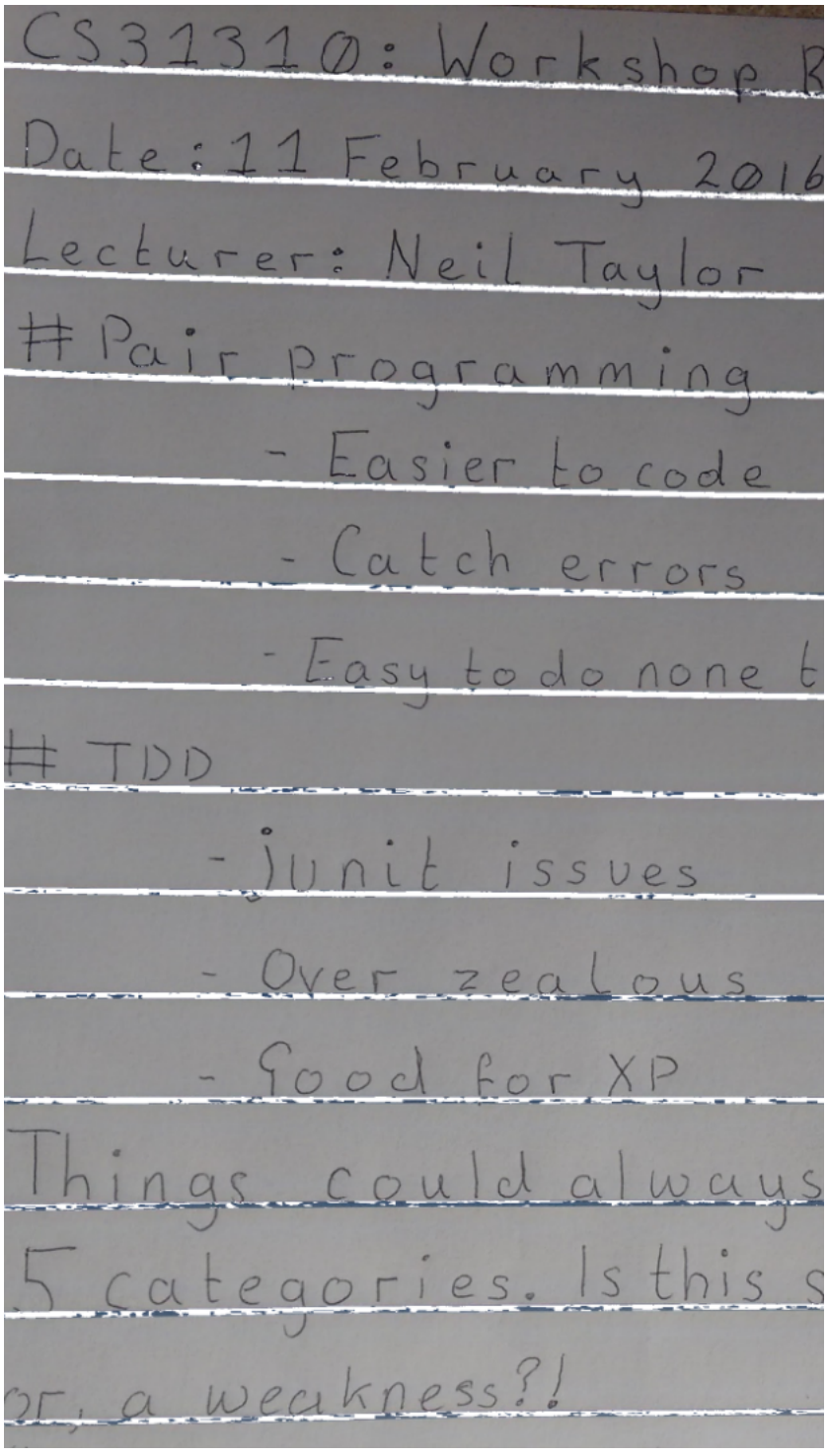
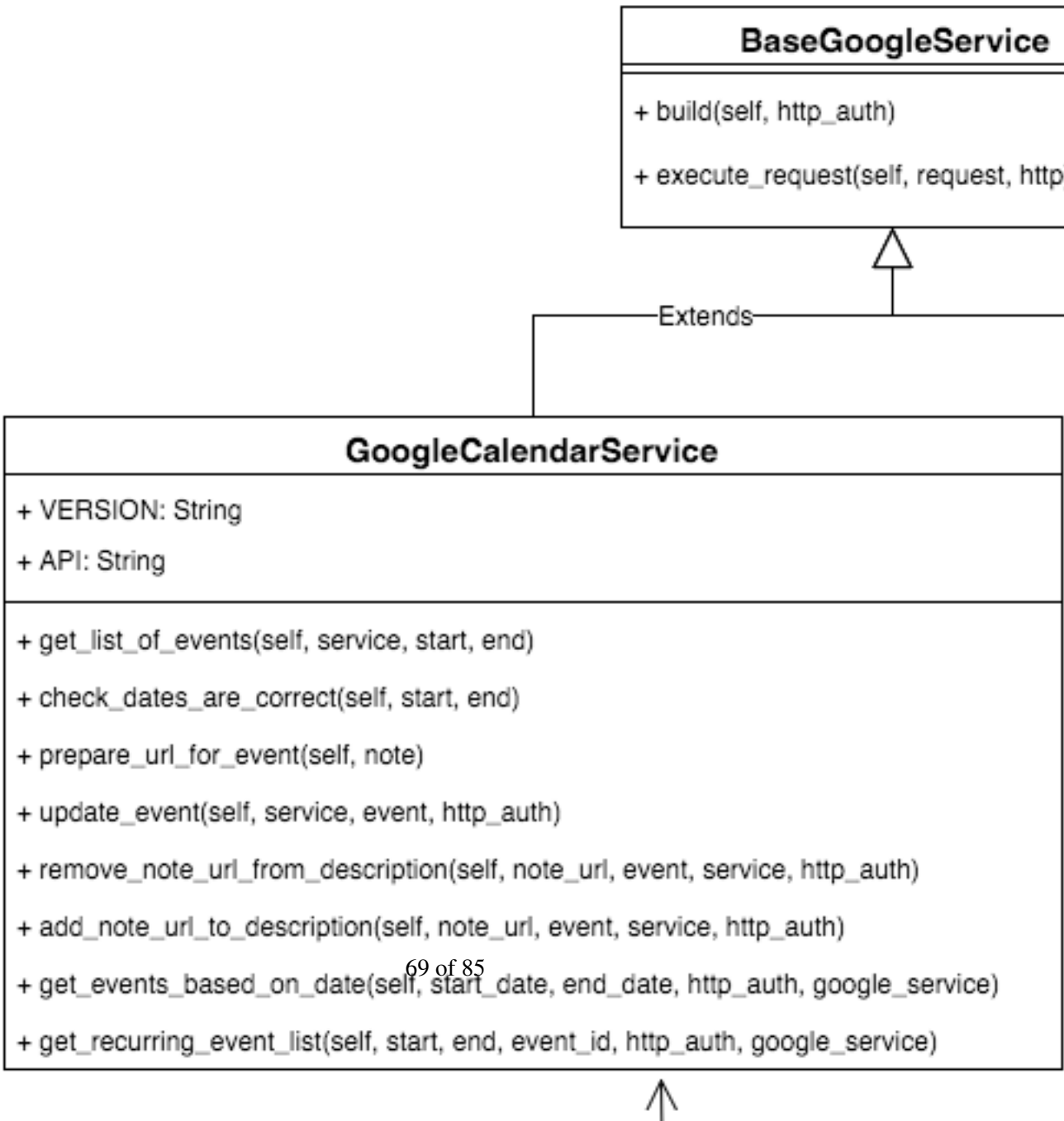


Figure F.2: Blue lines in the adaptive threshold have been identified and removed to be a white colour.

Appendix G

Design decisions

7.1 Class diagram

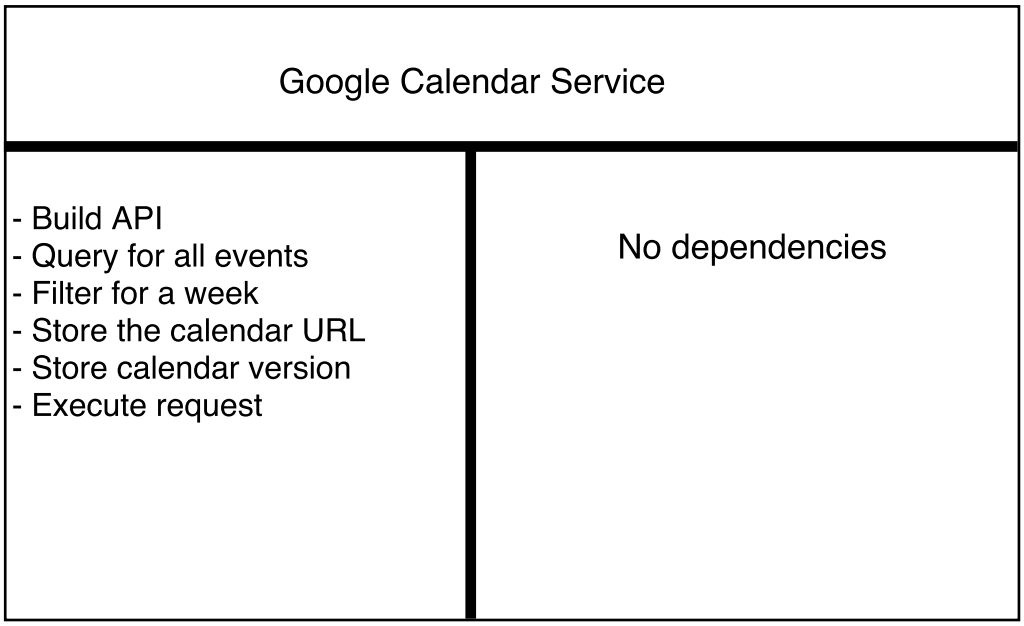


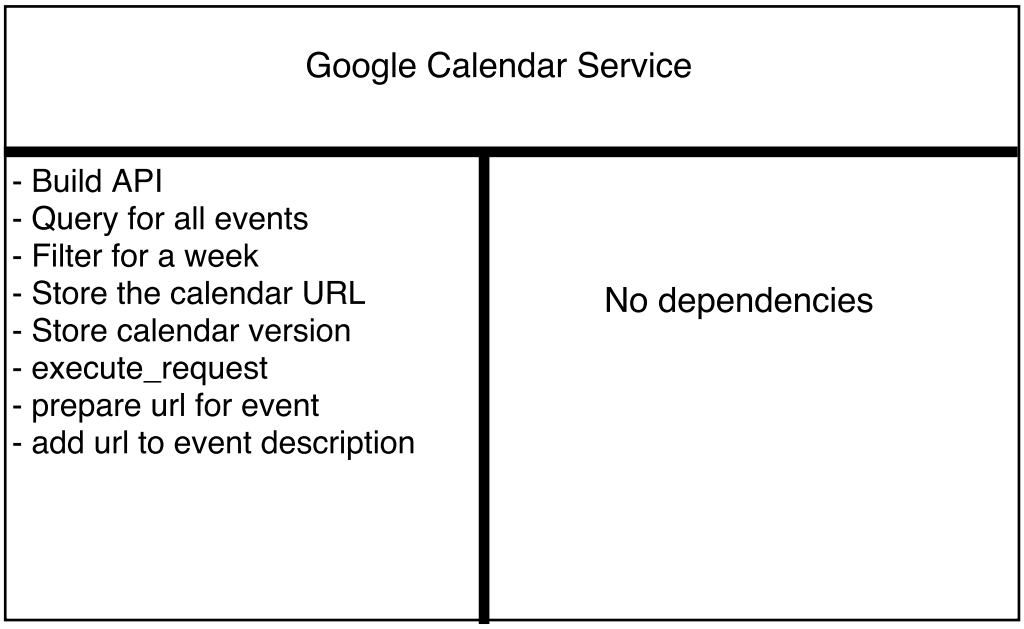
Appendix H

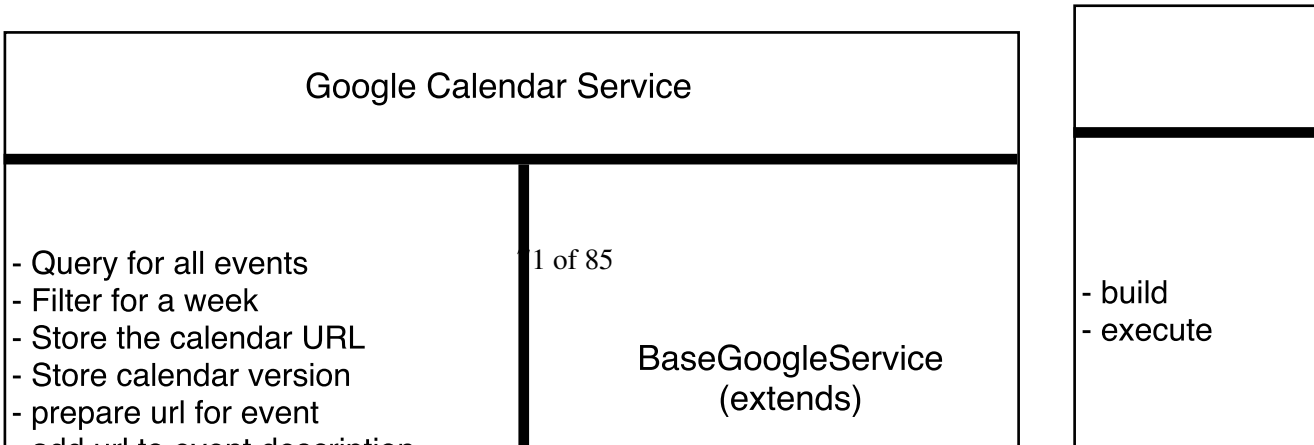
Design suppliments

8.1 CRC cards

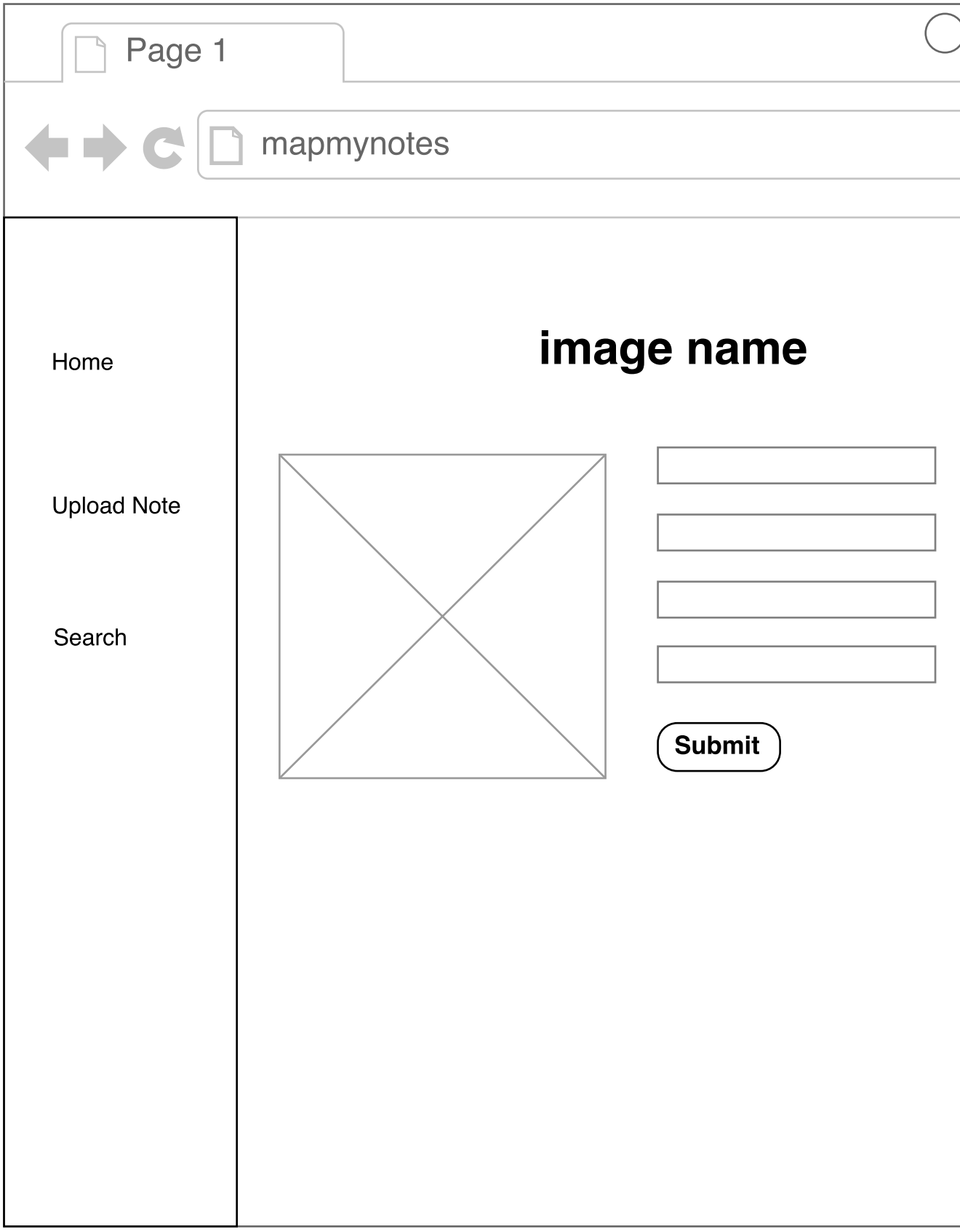
Below is an excerpt of the the examples of a more complex CRC card design in the system. Throughout the the project, each class went through an iterative process of using CRC cards. Therefore, a lot of them have been omitted to save space.


Google Calendar Service
- Build API
- Query for all events
- Filter for a week
- Store the calendar URL
- Store calendar version
- Execute request
No dependencies


Google Calendar Service
- Build API
- Query for all events
- Filter for a week
- Store the calendar URL
- Store calendar version
- execute_request
- prepare url for event
- add url to event description
No dependencies


Google Calendar Service
- Query for all events
- Filter for a week
- Store the calendar URL
- Store calendar version
- prepare url for event
BaseGoogleService
(extends)
- build
- execute

8.2 Wireframes



Appendix I

Scrum process supplementary materials

The appendix discusses some of the additional material to show the process of scrum used as the methodology of choice. Below is a collection of user-stories throughout the sprints.

9.1 Sprint burndown charts

Figure I.1: An example of the burndown chart for a sprint, showing areas where there may have been difficulty.

9.2 Overall burndown chart

Figure I.2: The overall burndown of the sprints during the development period. This clearly shows a consistent work flow up until more knowledge of the project was achieved, going below the expectation line.

Id	User story	Sprint	Story points
1	As a user I want to be able to upload an image of a set of notes so that I can see my note in the application	2	10
2	As a user I want to be able to tag my notes so that all my notes are under the correct module	4	5
3	As a user I want to be able to add information about the notes so that I can reference them in the future	4	15
4	As a user I want to be able to save a note, so that I can find it again later	3	10
5	As a user I want to be able to search for a given module, so that I can find all notes for that module	7	8
6	As a user I want to be able to sign in via google sign in	5	15
7	As a user I want to use Tesseract OCR so that I can identify characters	1	15
8	As a user I want to be able to view the application on a website	2	5
9	As the customer I want to see the image being binarised properly	2	10
10	As a developer I need to train my handwriting, so that Tesseract can recognise my handwriting	10	n/a
11	As a user I want to be able to edit the meta data, so that I can update it in light of a change	5	5
12	As a user I want to be able to remove a note incase I do not want it to appear	5	5
13	As a developer I want to the website to have good styling	4	8
14	As a developer I want to integrate tesseract into the application, so it can read information from a note	8	8
15	As a user I want to be able to view all the notes I have as a user so I can easily find all of them again	6	3
16	As a user I want to view a list of events on the homepage from my calendar, so I can see recent events	6	15
17	As a user I want to be able to save the URL in the calendars event	7	10
18	As a user I want to be able to tag the title of the lecture, so that I can know which one it is.	6	5
19	As a user, when I authorise I want to show my email address and remove the authorise button, so I know I have signed in	6	3
20	As a developer I want to be able to get the date taken from EXIF data, to show information about a note	7	8
21	As a user I want to be able to edit the date and update my calendar	9	8
22	As a developer I want to be able to associate a note with a user	7	5
23	As a user, I want to be able to have automated suggestion of meta data from the image, so that I can know what to tag.	8	5
24	As a user, I want to be able to logout, so that I can close my session	8	5
25	As a user I want to be able to click Tesseract Items, so that it's easier for me to put in the fields.	9	10
26	As a user I want to be able to edit and save to reoccurring events	10	10

Table I.1: A table showing the user stories identified throughout the project, along with the sprint in which they were implemented and associated story points

Annotated Bibliography

[1] "CSS Bootstrap," last checked 3rd April 2016. [Online]. Available: http://getbootstrap.com/css/

Bootstrap library was considered when thinking about the styling using CSS.

[2] "Modular Applications with Blueprints Flask Documentation (0.10)," last checked 28th April 2016. [Online]. Available: http://flask.pocoo.org/docs/0.10/blueprints/

Blueprints were used to modularise the code and expand it for a larger project. They were implemented to attempt to decouple specific routing.

[3] "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.

[4] "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.

[5] "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

[6] "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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] ——, "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.

[18] 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

[19] 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.

[20] ——, "Extract Class," Oct. 1999, last checked 28th April 2016. [Online]. Available: http://refactoring.com/catalog/extractClass.html

The description of what the extract class refactoring technique, which was used extensively on the project.

[21] GitHub, "Atom," 2016, last checked 28th April 2016. [Online]. Available: https://atom.io/

The text editor which was used for the majority of the project. It lacks refactoring tools, and the application grew too much for a find and search.

[22] ——, "GitHub," 2016, last checked TODO. [Online]. Available: http://www.github.com

The hosting service for the priavte git repository for the application.

[23] B. M. Gonzalez, "Iris : a solution for executing handwritten code," Master's thesis, University of Agder, 2012. [Online]. Available: http://brage.bibsys.no/xmlui/handle/11250/137557

A great reference material for creating an application which would parse the text on the page using Tesseract. Used prodominentely for understanding how to train Tesseract on handwritten text.s

[24] Google, "API Client Library for Python — Google Developers," 2016, last checked 30th April 2016. [Online]. Available: https://developers.google.com/api-client-library/python/

The client library which is used to interact with the oAuth services and the queries to the external API's.

[25] ——, "Color - Style - Google design guidelines," 2016, last checked 28th April 2016. [Online]. Available: https://www.google.com/design/spec/style/color.html

The colour guide was used for the CSS colours used throughout the application.

[26] ——, "Map My Notes Usability Questionnaire - Google Forms," 2016, last checked 30th April 2016. [Online]. Available: https://docs.google.com/forms/d/10noZA1Jrq0H_ffLeGd1q8cf2le-m6uJGVSKhqWvNWoY

The questionnaire which was created for part of a usability study which participants would attempt to use the application.

[27] ——, "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.

[28] 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.

[29] ——, "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.

[30] 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.

[31] 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.

[32] ImageMagick, "ImageMagick: Convert, Edit, Or Compose Bitmap Images," last checked 28th April 2016. [Online]. Available: http://www.imagemagick.org/script/index.php

ImageMagick is a library which was used for the image binarisation but was not used in the end, due to OpenCV providing better support.

[33] Itseez, "OpenCV — OpenCV," 2016, last checked 28th April 2016. [Online]. Available: http://opencv.org/

The image processing library used for the image binarisation and the various morphological tools. One of the best tools used on the project.

[34] JetBrains, "PyCharm :: Download Latest Version of PyCharm," 2016, last checked 28th April 2016. [Online]. Available: https://www.jetbrains.com/pycharm/download/

An IDE used later on in the project to aid in more comprehensive refactoring tools.

[35] 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.

[36] F. Lundh and Contributors, "Pillow - Pillow (PIL Fork) 3.2.0 documentation," 2016, last checked 30th April 2016. [Online]. Available: http://pillow.readthedocs.io/en/3.2.x/

The Python image library which was used for the EXIF data parsing.

[37] 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.

[38] 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

[39] ——, "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.

[40] ——, "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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] OpenCV., "Eroding and Dilating OpenCV 2.4.13.0 documentation," 2016, last checked 30th April 2016. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

A reference for how dilation and errosion works in the OpenCV, used for reference throughout the development and implementation.

[46] 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.

[47] 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.

[48] C. Patel, A. Patel, and D. Patel, "Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study," *International Journal of Computer Applications*, vol. 55, no. 10, pp. 50–56, Oct. 2012, last checked 28th January 2016. [Online]. Available: http://dx.doi.org/10.5120/8794-2784

A great paper on a Tesseract case study tool. It was used as a good comparsion for other OCR technologies as well as providing statistical results for the use of Tesseract.

[49] D. Peterson, "What is EXIF? :: Digital Photo Secrets," last checked 30th April 2016. [Online]. Available: http://www.digital-photo-secrets.com/tip/38/what-is-exif/

A good reference for explaining simple what EXIF data is and the purpose of it.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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

[55] R. Smith, "A simple and efficient skew detection algorithm via text row accumulation," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 2. IEEE, Aug. 1995, pp. 1145–1148 vol.2, last checked 29th April 2016. [Online]. Available: http://dx.doi.org/10.1109/icdar.1995.602124

An excellent paper which describes how text lines are extracted from an image in Tesseract.

[56] ——, "An overview of the tesseract ocr engine," in *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*, 2007, pp. 629–633, last checked 29th April 2016.

An excellent paper which discusses how Tesseract works with a comprehensive description of the details of Tesseract.

[57] 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

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] L. Torvalds, "Git," 2016, last checked 28th April 2016. [Online]. Available: https://git-scm.com/

The version control management system used on the project, to manage workflows

[64] Transym, "Transym - OCR software for Integrators — Transym Computer Services," 2016, last checked 28th April 2016. [Online]. Available: http://www.transym.com/

A comparison tool to the Tesseract OCR that is proprietary.

[65] Travis, "Travis CI - Test and Deploy Your Code with Confidence," 2016, last checked 28th April 2016. [Online]. Available: https://travis-ci.org/

The Travis CI tool which was used during the process. Would be reliably, used and a great tool to aid in the development process.

[66] Various, "Jenkins," 2016, last checked 28th April 2016. [Online]. Available: https://jenkins.io/index.html

The Jenkins CI tool was considered when analysing which CI tool to use and integrate. Eventually was not chosen because of it being a standalone application.

[67] 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.

[68] 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

[69] 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.

[70] 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.

[71] T.-O. Wiki, "TrainingTesseract tesseract-ocr/tesseract Wiki," last Accessed 6th February 2016. [Online]. Available: https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract

A reference for how to train the different user languages with Tesseract.

[72] F. Words, "Pangrams," last checked 28th April 2016. [Online]. Available: http://www.fun-with-words.com/pangrams.html

A tool which describes what panagrams are as well as using this tool as inspiration for some section of the training data to ensure that there was a good spread of data.