MapMyNotes

Final Report for CS39440 Major Project

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

4th March 2016
Version: 1.0 (Draft)

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

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

Declaration of originality

In signing below, I confirm that:

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

Name .....

Date .....

Consent to share this work

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

Name .....

Date .....

Acknowledgements

I am grateful to...

I’d like to thank...

Abstract

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

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Design

As the application was developed in an iterative manner, over a series of sprints, class diagrams and design diagrams were not created at the very start of the project. Over a series of sprints designs were iteratively developed regarding the system overview. However, some design decisions were decided at the start of the project. The chapter will clearly explain rationale for the decisions and state whether the design decisions were a result of iterative processes or upfront design.

1.1 Implementation tools

The following discusses the design decisions regarding the implementation tools that were used during the project, providing rationale for the choice of tools.

1.1.1 Programming language

With the programming language choice unlikely to change per sprint, then an upfront design decision was made. From the analysis it was concluded that the application would be implemented as a web application. A consideration between different server side programming languages was considered in depth.

Typically with server side applications the traditional language choices are: PHP, Ruby, Python, C#, Java and JavaScript, which has increased in popularity recently [26].

Evaluating the analysis decomposition in the meetings in the beginning sprints it was determined that OpenCV, would be needed to be utilised in the project. OpenCV's original source code is written in C++, however Python and Java bindings to OpenCV 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 reducing the opportunity for support during the project, it was dismissed as viable choice. Java applications are predominately large commercial applications, using a range of enterprise software - often renowned for their performance abilities [17]. This approach felt too cumbersome for a proposed light-weight application.

By being constrained by design decision to use OpenCV coupled with the lack of reliable

wrappers for other languages and a reluctance to use Java then Python was selected as the most suitable language. Python offers a lightweight and quick to learn syntax that produces readable code whilst allowing a object-oriented paradigm to be followed. Additionally, its support for OpenCV is sufficient for the application.

1.1.2 Choice of framework

To recap, Python was chosen as the language of choice for the application. Frameworks aid in the development of a project, especially with web applications where routing is often required; often routing support is a key feature of frameworks. Exploratory work was completed in the early sprints to find a suitable tool. The frameworks Django [4], Flask [7] and Bottle [3] were evaluated.

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

Flask and Bottle are classified as "micro-frameworks", offering a lot less restrictive structure and allowing the developers to have more control on how the applications are structured. On face value, Flask and Bottle appear to be very similar: they are both lightweight with a similar syntax. Evaluating both of the frameworks identified that Flask had a larger support community than Flask, coupled with better documentation, compared to Bottle. As neither framework had been used before then a support community was an important design decision.

As a result, Flask was chosen as the framework which will be used throughout the application. Spike work was completed to see how simple it was to create a quick application, after discovering it was perfectly fine, Flask was agreed as the framework of choice.

1.1.3 Database management system

As the analysis suggested it should be a web application then data persistence should be an integral part of the application. Prior to choosing an appropriate tool, a rationalised decision had to be made on the different management systems available.

1.1.3.1 NoSQL or relational database management systems

The choice of database management system was considered at length. Firstly, the decision of whether to use NoSQL or a relational database management system (RDBMS) had to be decided. This was decided during the early sprints, whilst the design was beginning to emerge.

RDBMS's are suited to separating the data into logical relations which decouples through the process of normalisation. The data normally suited to a RDBMS is one which has a uniform structure and can be represented as a series of relations, with specific constraints.

NoSQL provides a more flexible structure, allowing data to have no specific structure. In MongoDB [15] documents are equivalent to relations - structuring the data as key-valued pairs over specific column structures. This offers more flexibility in the data passed to the database.

From previous design choices made in prior sprints regarding the database structure, both options were seriously considered. After the analysis of what the note meta data would consist of, it was decided that the structure for the data would not vary, each note will have an associated lecturer, title etc, then a relational database would be suitable for the application. In addition, the notes *must* follow a specific layout structure, therefore affirming that data would be linked and structured.

A specific choice of RDBMS had to be chosen as an appropriate tool to persist the data from a note; digital ocean offered a concise comparison aiding in the design [24].

1.1.3.2 SQLLite

SQLLite was considered as a viable option; Flask provides documentation for its ease of use with the micro-framework. Considering the larger picture: would it scale well with multiple users interacting with it? Digital ocean state it would be a perfectly fine testing and development tool, but for production it may not scale well with multiple reads and writes.

1.1.3.3 PostgreSQL

Digital ocean describe PostgreSQL as the most advanced of the RDBMS's available. The additional support of more types an attribute can have allows for a greater control when creating a database. For example, the JSON type support would allow the application to save returned JSON from 3rd party services directly to the database.

1.1.3.4 MySQL

MySQL and PostgreSQL for most applications would be interchangeable. It offers a wide range of support along with comprehensive documentation. There are concerns over performance, however.

After evaluating the possible RDBMS's it was concluded that MySQL and PostgresSQL would be more preferable for scalability. Although there is not a great difference between the two, PostgreSQL was chosen for its advanced ability allowing for the application to become more advanced. This goes against the YAGNI (you ain't gonna need it) approach, but infrastructure decisions are harder to change, than code implementations.

1.1.4 Continuous Integration tool

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

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

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

1.2 Overall Architecture

1.2.1 CRC cards

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

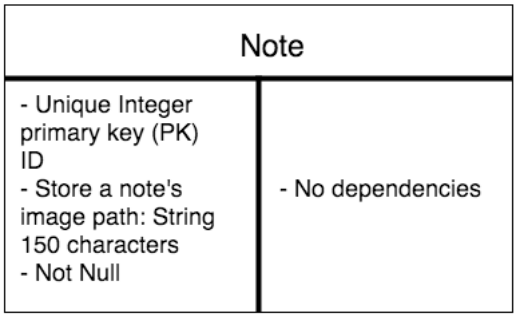

Figure 1.1: An example from Sprint 3, showing a CRC card at the very beginning of creation.

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

1.2.2 Class Diagram

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

1.2.3 Database diagram

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

Figure 1.2, shows the output from the result of continuous design on the database diagram through the iterations.

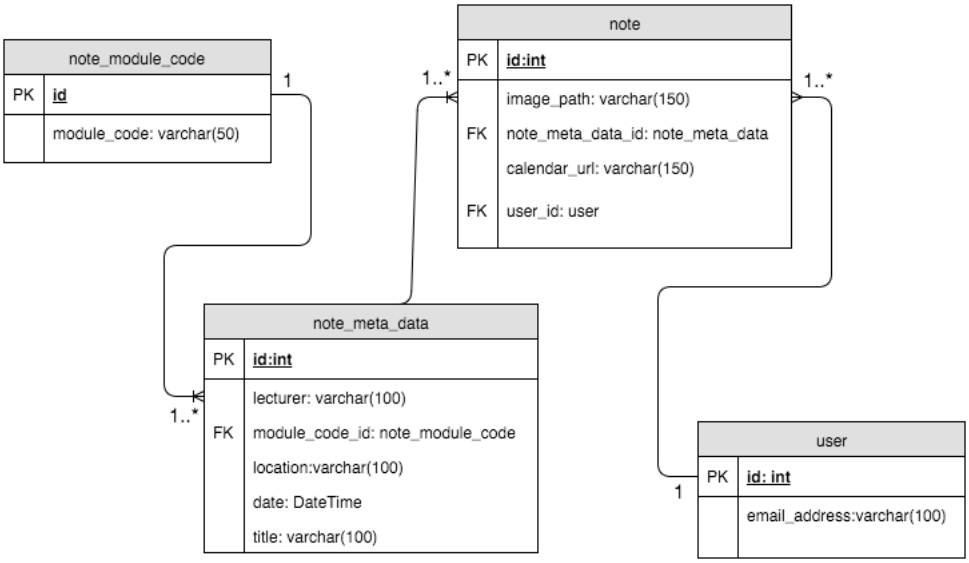

Figure 1.2: The final result of the database diagram. After a series of iterations.

1.2.3.1 Justification of design

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

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

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

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

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

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

Finally, a user relation was established after the application decided it would be expanded

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

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

1.2.4 MVC Structure

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

1.2.4.1 About MVC

MVC is a design pattern where you split the logic into a Model View and controller. As displayed in figure 1.3 the MVC approach helps to decouple logic from the view file. Therefore all the information passed to the view is renderable content. The controllers do not integrate with the database directly. The primary job for the controllers is to interact with the model, any services and ask to render the view files.

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

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

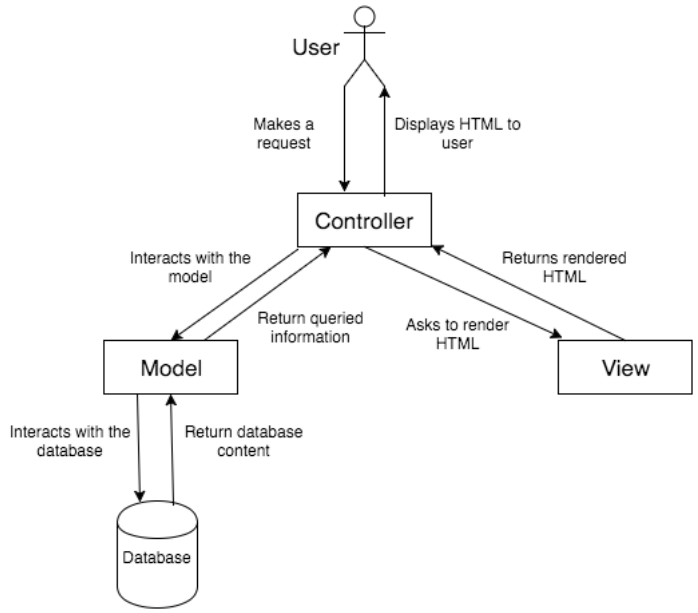

Figure 1.3: A example of how the model-view-controller(MVC) framework integrates.

1.2.4.2 Structuring the Web application

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

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

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

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

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

1.2.4.3 Extending views

It is worth mentioning that Flask uses the Jinga templating engine[CITE]. By default Jinga would expect the DOM to be represented again in every view file. A design decision was made to extend the view files. Figure 1.4 illustrates the view file design.

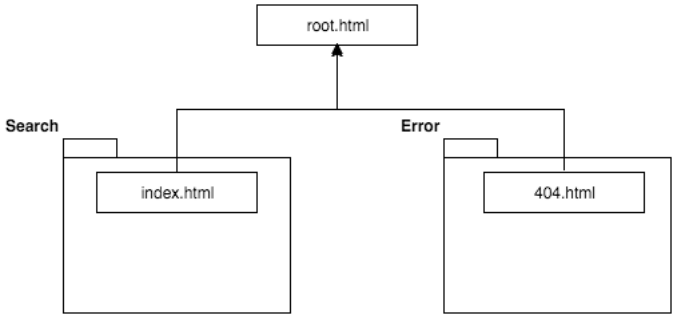

Figure 1.4: A diagram illustrating how extension in Jinga html template engine works.

Each of the view file templates are separated into their own directories. Inside the directories, the HTML files extend the index.html root file. This overrides the block “content”, dynamically

generating a new content for every new URL. The advantages of this design is that meta-data, headers, stylesheets and navigation can be placed in the index.html page and all subpages will inherit these automatically, sticking to the principle of DRY.

1.2.5 Application URLs

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

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

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

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

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

1.2.6 HTTP methods

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

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

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

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

1.2.7 Interaction with the application

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

1.3 User Interface

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

To begin with the user interface (UI) went through a few iterations of wiremocks to sketch out what it should look like. This iterative approach was applied when actually designing the website. Firstly it was important that the application felt as thought it was an application, rather than a traditional website. Colours from Google Style guide [CITE] was used to keep with nice, clean and bold colours. Bootstrap[CITE] was considered to be used on the application and it did give a default responsive theme built in - however, keeping with the idea of minimalist and not over-bloated, then personally I feel that that bootstrap comes with a lot of the additional material which would not be needed.

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

- Wiremock examples
- Media query code examples.

1.4 Other relevant sections

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

1.4.1 Tesseract

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

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

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

1.5 Optimising tesseract

1.5.1 ImageMagick

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

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

1.5.2 OpenCV

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

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

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

1.6 Version control

Appendices

Appendix A

Third-Party Code and Libraries

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

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

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

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

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

Appendix B

Ethics Submission

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

Appendix C

Code Examples

3.1 Random Number Generator

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

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

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

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

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

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

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

Annotated Bibliography

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

An article explaining how Evernote does character recognition on images

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

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

[10] 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.

[11] 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.

[12] 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

[13] ——, "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.

[14] ——, "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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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

[21] 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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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

[27] 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.

[28] 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.