



Final Report

My Important Things Project

CM3203 – Individual Project

40 Credits

Alfred Thomas Rowett: C1732088

Supervised by: Catherine Teehan

Moderated by: Michael Daley

Abstract

Looked after children are often declared 'wards of state', with their parental responsibility falling upon that of the local authorities. This is the case when their home environment is declared 'unsafe' and they are removed from the situation for their safety, the result of which is often that these children are moved between foster homes, group homes and relatives, whilst also bouncing between social workers and postcodes. Consequently, managing each child's documents and personal belongings is a complicated task when moving home and changing hands which is the stem of problem I aim to resolve during this project.

This project will detail the background, research, design, implementation, testing and user feedback of building the 'My Important Things App'. In collaboration with Cardiff Council Children's Services the project aims to find a digital solution to the problem of lost or damaged documents and memories of each child within the care system, bringing this all together into a single centralised and secure storage platform.

The following report will assess the strengths and weaknesses of the project in achieving the original requirements as well as the development process.

Acknowledgements

I would like to thank my supervisor Catherine Teehan for her dedication to the project and providing me with all the relevant information which allowed me to work so smoothly. As well as Caitlyn Powell, who undertook the other half of the Children's Services project to develop an application containing information on being in foster care and leaving care. Both myself and Caitlyn attended meetings and research groups together, and later shared design and implementation ideas to ensure both projects were kept similar for the same client.

I would also like to thank Thomas Pughsley and Samantha Anderson from the Child Friendly City Programme and National Youth Advocacy Service respectively, who were incredibly helpful in providing an insight to the care system and getting me access to potential users of my app through test groups which was invaluable for research and testing.

Table of Contents

| | |
|---|----|
| Abstract..... | 2 |
| Acknowledgements..... | 3 |
| Table of Figures..... | 6 |
| 1. Introduction | 7 |
| 2. Background | 7 |
| 2.1 Existing Solutions | 7 |
| 2.1.1 Document Storage Solutions | 8 |
| 2.1.2 Note/Journaling Apps | 12 |
| 2.1.3 Summary of Existing Solutions..... | 14 |
| 2.2 User Research and Feedback | 15 |
| 2.2.1 User Research Data Storage | 16 |
| 2.2.2 User Research Mobile accessibility | 16 |
| 2.2.3 Web Accessibility Research..... | 17 |
| 2.3 User Personas | 17 |
| 2.4 Project Justification..... | 19 |
| 2.5 Project Constraints..... | 20 |
| 2.5.1 Financial Constraints..... | 20 |
| 2.5.2 Timescale Constraints | 20 |
| 2.5.3 Coronavirus Constraints..... | 21 |
| 3. Methods of Development and Planning | 22 |
| 3.1. Agile Development Methodology | 22 |
| 3.2 Learning..... | 22 |
| 3.2.1 Web development | 22 |
| 3.2.2 Data Storage..... | 24 |
| 3.2.3 Children’s Services and the Care System | 25 |
| 4. Specification..... | 25 |
| 4.1 Functional Requirements..... | 26 |
| 4.1.1 Must Have | 26 |
| 4.1.2 Should Have | 27 |
| 4.1.3 Could Have | 27 |
| 4.2 Non-Functional Requirements..... | 28 |
| 4.2.1 Must Have | 28 |
| 4.2.2 Should Have | 28 |
| 4.3. Use Cases | 29 |

| | |
|---|----|
| 4.3.1 Use Case Diagram | 29 |
| 4.3.2 Use Cases | 30 |
| 5. Design..... | 35 |
| 5.1 UML Class Diagram | 35 |
| 5.1.1 UML Overview..... | 36 |
| 5.1.2 Use Case Mapping..... | 37 |
| 5.2 User Interface Designs and Wireframing..... | 38 |
| 5.2.1 Mobile First Development and Intuitive Design | 38 |
| 5.3 Prototype Designs..... | 41 |
| 5.3.1 Wireframes | 41 |
| 5.3.2 Colour Scheme | 45 |
| 5.3.3 Wireframe Strengths and Weaknesses..... | 45 |
| 6. Implementation | 46 |
| 6.1 Database Design..... | 46 |
| 6.1.1 User Profiles..... | 47 |
| 6.1.2 Document Storage | 47 |
| 6.2 Web Development Best Practices..... | 50 |
| 6.3 Django Implementation | 51 |
| 6.3.1 Static Page Implementation..... | 53 |
| 6.3.2 Dynamic Page Implementation..... | 53 |
| 7. Results..... | 63 |
| 7.1 Test Cases..... | 63 |
| 7.1.1 Functional Requirements..... | 64 |
| 7.1.2 Non-Functional Requirements..... | 64 |
| 8. User Feedback..... | 65 |
| 9. Conclusion..... | 65 |
| 10. Future Work..... | 66 |
| 11. Project Evaluation | 67 |
| References | 69 |

Table of Figures

| | |
|---|----|
| Figure 1 - Amazon S3 Monthly Estimates for Data storage | 20 |
| Figure 2 - Gantt Chart containing my project schedule, deliverables and milestones | 21 |
| Figure 3 - Use Case Diagram displaying how a user would interact with the main app functions (must haves)..... | 30 |
| Figure 4 - Upload/Delete image flowchart | 31 |
| Figure 5 - Upload/Delete Document Flowchart..... | 31 |
| Figure 6 - Add/Delete Calendar Entries Flowchart | 32 |
| Figure 7 - Add/Delete Journal Entry Flowchart..... | 33 |
| Figure 8 - Accessing Useful Links Flowchart..... | 33 |
| Figure 9 - Login and Logout Flowchart..... | 33 |
| Figure 10 - Download/Print Document Flowchart..... | 34 |
| Figure 11 - My Important Things App UML Class Diagram | 35 |
| Figure 12 - Example of Iconography | 39 |
| Figure 13 - Example of Navigation Bar | 39 |
| Figure 14 - Example of User feedback through button colour change..... | 40 |
| Figure 15 - Navy Blue #051938 | 45 |
| Figure 16 - Primary Blue #0275d8..... | 45 |
| Figure 17 - Orange #ff5b00 | 45 |
| Figure 18 - Database User Model | 47 |
| Figure 19 - Database User Group Model | 47 |
| Figure 20 - Database Memories Model | 48 |
| Figure 21 - Database Memory Category Model..... | 49 |
| Figure 22 - Memories Upload Screen | 49 |
| Figure 23 - New image added to database model..... | 49 |
| Figure 24 - User database interaction model | 50 |
| Figure 25 - Base project folder and extending apps. | 50 |
| Figure 26 - Project Installed Apps | 51 |
| Figure 27 - Django Password package | 52 |
| Figure 28 - Settings.py Static File paths | 52 |
| Figure 29 - Home app folder | 53 |
| Figure 30 - Document Wallet app folder | 53 |
| Figure 31 - Document Wallet models.py | 54 |
| Figure 32 - Admin model registration..... | 54 |
| Figure 33 - Page Render View function..... | 55 |
| Figure 34 - Document Wallet as seen by users..... | 55 |
| Figure 35 - Viewing Page View function | 56 |
| Figure 36 - Document View as seen by users | 56 |
| Figure 37 - Xframe Origin fix | 56 |
| Figure 38 - Document ordering View function | 57 |
| Figure 39 - New Document View function | 58 |
| Figure 40 - Upload Document View as seen by users..... | 58 |
| Figure 41 - File Deletion View function..... | 59 |
| Figure 42 - Navigation HTML Code | 60 |
| Figure 43 - Navigation as seen by the user | 60 |
| Figure 44 - Dynamic Loops | 61 |
| Figure 45 - Document Wallet as seen by the user | 61 |

1. Introduction

This project was proposed to me by Catherine Teehan as a collaboration with NYAS, Child Friendly Cardiff and Cardiff Councils Children's Services with the aim to create a digital solution for the document storage problems faced by children in care. The hope was to build an application that allowed for children within the care system to store important documentation and personal memories securely and easily in the form of photos and journal entries. With the current infrastructure being very limited and with the young people often changing homes and supervisors it is very easy for papers or boxes of photos to become misplaced or completely lost, with important documentation taking weeks to get replaced and personal effects simply being lost forever.

Being a young person in care is already a daunting experience with many people involved and the possibility of moving homes numerous times along with changing social workers. This was the source of my motivation to complete this project, the real-world impact it could have on so many people, both providing a better system for organising their lives, but also to aid in preserving the memories they already have.

As was originally laid out in the project initiation document the two main objectives for the project were to:

- Create a secure document 'wallet' for care experienced children.
- Create a 'memory box' associated with the document wallet.

This was later extended following research collection with groups of young people from the foster care system to include:

- Create a digital journal feature to allow users to make entries about their lives or notes.
- Create a calendar that would aid in organisation of users lives and help manage documents further.

These four main functions are what would therefore make up most of the application in a format that would be easy to use across all devices and in a secure method. All functions would easily offer filtering and categorisation for improved functionality and ease of use, whilst being stored securely online using the latest in web-hosting technologies.

The application would be both user and device friendly allowing for access from any device with internet access, with the hope to later be able to port into a mobile app rather than browser-based format. Designs would be run past test users to gain feedback and research into Web accessibility from online resources and webinars would aid in the design of a sleek and user friendly front-end.

The rest of the report will cover in further detail the design, research, and implementation aspects of the project.

2. Background

2.1 Existing Solutions

Before undertaking the design and production phases of this project, it was important to have a strong understanding of the existing solutions that are available on the market that aim to resolve the issues I am trying to fix. Although there are not any specific applications aimed at those in the

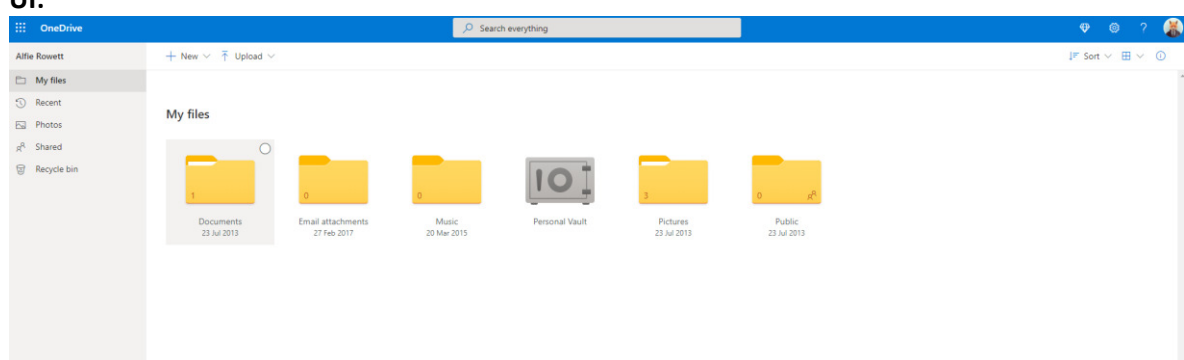
care system, there are numerous document storage and general life manager programs available. From my own research I was able to find that the most popular and widely used included Microsoft's OneDrive and OneNote, as well as Dropbox, Google Drive and Evernote, all of which fulfilled some of the objectives that I hoped to achieve with my own project.

However, as my objectives aim to include features from all the existing solutions, there is no other application currently in widespread circulation that meets every requirement set out by myself and the test group I spoke to.

Outside of the technological solutions in use, many of the research group we spoke to used physical storage of documents and memories, which although simple and always available, can easily be damaged, lost or simply disorganised which was a common response during our initial conversation with care experienced users. Many of those spoken to said they keep all documents in folders or cupboards which brings with it many negatives that could easily be avoided using a well-built digital solution.

Having taken the time to investigate each of the most popular available existing solutions, I have summarised the results below, looking at the pros and cons to each as well as the aspects I aimed to replicate for my own project due to their intuitive design or ease of use for the user.

2.1.1 Document Storage Solutions

| | |
|---|--|
| Application: Microsoft OneDrive | |
| Description: Popular cloud storage service offered by Microsoft in competition with Apple iCloud and Google Drive. Widely used thanks to the availability across all devices and integrated services including that of the very popular Microsoft Office Suite that allows for seamless use of Word, PowerPoint, Excel and more through the application. | |
| UI:  | |
| Pros: <ul style="list-style-type: none"> - Offers some level of free storage. - OneDrive mobile is free to use and allows for file access from anywhere at any time. - Integration with the powerful Microsoft Office Suite which is unchallenged in its document usage. - Clear UI design with obvious search and filter options - Documents and photos displayed in easy to view layout with clear locations and dates. | Cons: <ul style="list-style-type: none"> - No protection against human error or malware attacks. Meaning no version control for stored documents. - No obvious options to change the default style (dark mode, text size) for improved Web accessibility. |

Price Plans:

Free – OneDrive Basic offers 5GB of storage for no cost.

Cheap - OneDrive Standalone offers 100GB of storage for £1.99 a month.

Premium – Microsoft 365 Personal offers 1TB (1000GB) of storage alongside Skype and the Microsoft Office Suite for £59.99 for a year.

Features to use:

- Layout for documents and photos in clear grid format which is easy to search through and view for all users.
- Mobile accessibility allowing users access from any internet connected device.

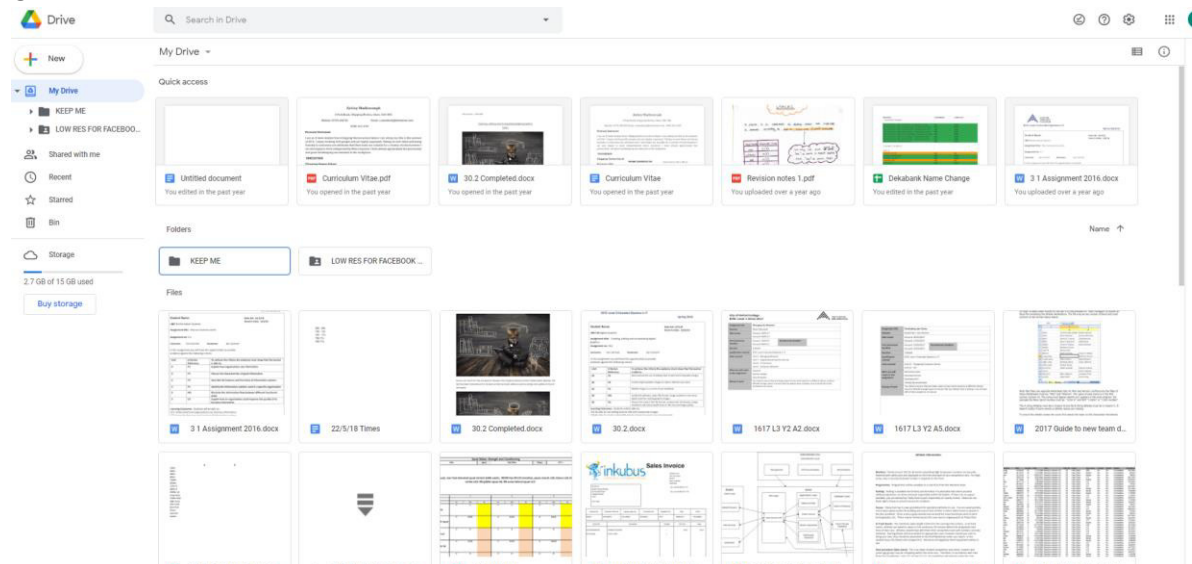
Source: <https://www.microsoft.com/en-gb/microsoft-365/onedrive/online-cloud-storage>

Costs: <https://www.microsoft.com/en-gb/microsoft-365/onedrive/compare-onedrive-plans?activetab=tab%3aprimar1>

Application: Google Drive

Description: Very popular cloud storage thanks to the generous 15GB of free storage compared to other free tiers from competitors. Being owned by Google allows for seamless integration in the online environment with many great features like collaboration.

UI:



Pros:

- Clear folder structure and filtering options as well as options to categorise and group contents.
- Easily sharable with sharing buttons available at all stages (folders and files).
- Supports version control which is perfect for making changes to existing documents.

Cons:

- White UI is hard to look at and read the text due to consistent brightness on screen.
- Contents displayed very uniformly and in an uninteresting manner (although is clear).
- No automated sorting for file types, unless organised everything is thrown in together.

Price Plans:

Free – Google Drive Free offers 15GB of free storage.

Cheap – Google Drive Basic offers 100GB of storage for £1.59 a month.

Standard – Google Drive Standard offers 200GB of storage for £2.49 a month

Premium – Google Drive Premium offers 2TB (2000GB) of storage for £79.99 a year.

Features to use:

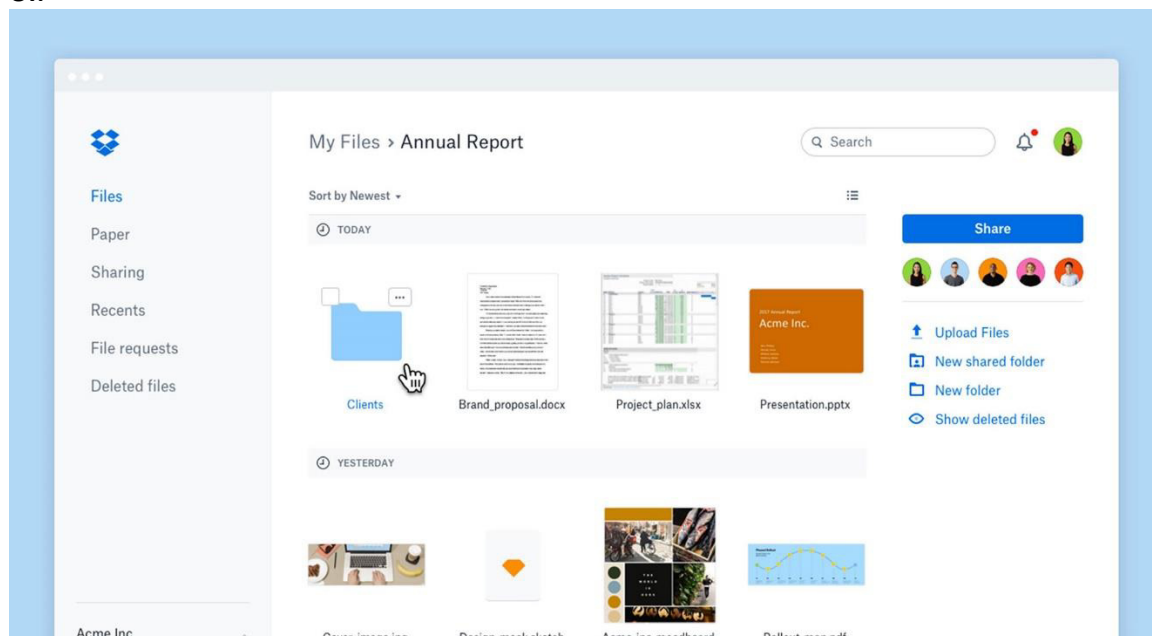
- Option to share files or photos at the click of a button is great addition that offers ease of use to users.
- Clear and idiot-proof UI layout that is hard to confuse.

Source: <https://www.google.com/intl/en-GB/drive/>

Costs: <https://one.google.com/storage>

Application: Dropbox

Description: Dropbox is a cloud storage services that allows for synced online and device file storage, whilst also offering logical sharing functionality to allow for smoother working and a reduction in complicated file storage.

UI:**Pros:**

- Great UI with basic but functional design which works well across all devices.
- Version control for all files allowing recovery of mistakes and lost work.
- Automatic file backup giving users peace of mind in document safety.

Cons:

- Does not offer the same level of search functionality as its competitors.
- Does not offer the same elite-level security as competitors (has resulted in a number of high-profile hacking stories).

Price Plans:

Free – Dropbox Basic offers 2GB of storage for free.

Standard – Dropbox Plus offers 2TB (2000GB) of storage for £9.99 a month.

Family – Dropbox Family offers 2TB (2000GB) of storage for up to 6 users for £16.99 a month.

Features to use:

- Simple but great UI design with layout working flawlessly for all device sizes.
- Automatic backup of important files for improved security/safety of high importance documents.

Source: https://www.dropbox.com/en_GB/

Costs:

https://www.dropbox.com/plans?_ad=496929751805%7C%7Cc&_camp=1033325279&_kw=dropbox+uk+pricing%7Ce&_tk=paid_sem_goog_biz_b&gclid=Cj0KCQjw9_mDBhCGARIsAN3PaFOCg8yrpoq6N9uMvyO9zkNukyNrd7HVgwNCl-4YLhE5A_x-L2HYyZEaAn3REALw_wcB&tab=personal

Application: Physical Storage

Description: Unlike the other solutions, physical storage of documents and printed photographs within folders, cupboards or draws is still widely used thanks to the lack of reliance on technology or an internet connection.

UI:



Pros:

- Always available, without requirements for digital platform or an internet connection.
- Safe from hacking risks or data leaks.
- Zero cost for storage allowing you to store as much as you can fit into your storage cabinet (storage location).

Cons:

- No protection against loss or damaged files, whether that be due to water, fire, or other damage.
- Documents can age and lose integrity.
- Can only be accessed from storage location and will require physical moving if changing location.
- No version control, and deletion of documents is permanent.

Price Plans:

Cost of printing photos and documents varies, but storage is free as it is not a digital solution.

Features to use:

- Being free to store large quantities of documents.

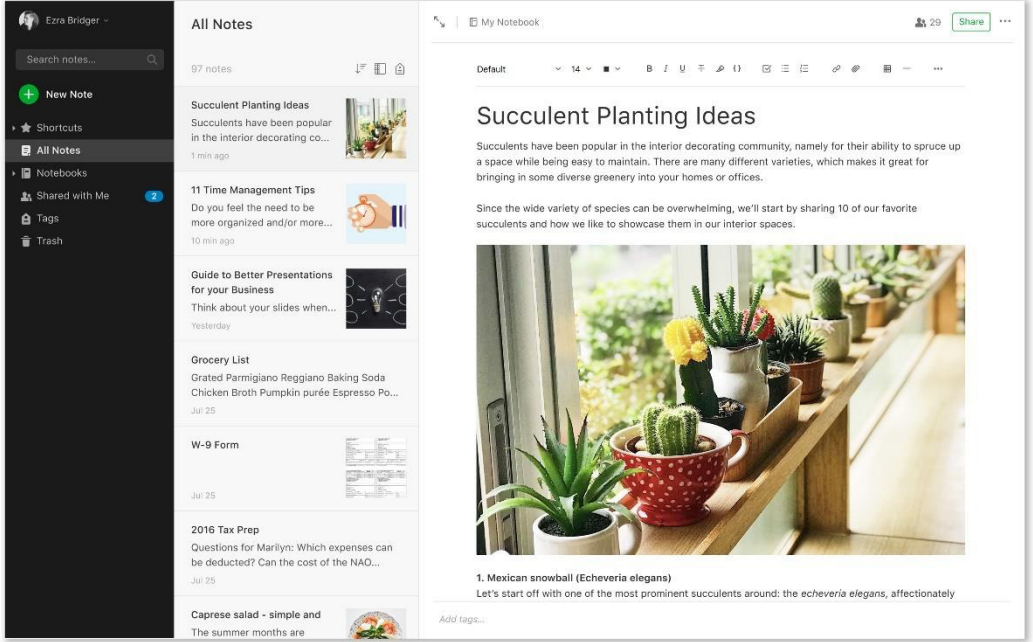
Being the existing non-digital solution, I aim to avoid being similar to physical storage due to the cons being the main driving force behind the entire project.

Source: N/A

Costs: N/A

2.1.2 Note/Journaling Apps

| | |
|--|--|
| Application: OneNote | |
| Description: Microsoft OneNote is a free note taking application that allows users to easily create and share notes, lists and other short form documentation. It is very popular thanks to its ease of use and integrated mobile app that works seamlessly alongside the website. | |
| UI: | |
| Pros: <ul style="list-style-type: none"> - Seamless integration with mobile app that works perfectly alongside the web application. - Links to OneDrive for easy connection and linked content. - Offers support for collaborative work. | Cons: <ul style="list-style-type: none"> - Limited functionality being solely a note taking app. |
| Price Plans: Free – OneNote is a free application offered by Microsoft. | |
| Features to use: <ul style="list-style-type: none"> - Completely free and unrestricted access to all features. - Ease of use and ability to create new content on the fly from your phone. | |
| Source: https://www.microsoft.com/en-gb/microsoft-365/onenote/digital-note-taking-app?ms.url=onenotecom&rtc=1 Costs: N/A | |
| Application: EverNote | |
| Description: Evernote is a modern note taking tool aimed at the young entrepreneurs and creative people and offers a smooth and professional looking interface. However, with limited free functionality it struggles to have the same widespread use as other options such as OneNote. | |
| UI: | |

| | |
|--|---|
|  | |
| Pros: <ul style="list-style-type: none"> - Effortless notetaking and syncing. - Excellent features provides a premium feel. - Easy and flexible access to all your content from both the web and mobile app. | Cons: <ul style="list-style-type: none"> - Limited storage and usability without paying a premium for the storage space. - Quite expensive premium plan for what is offered. |
| Price Plans: Free – Evernote Basic offers a 60MB monthly upload limit for zero cost. Standard – Evernote Premium offers 10GB of monthly uploads for £4.99. Business – Evernote Business offers 20GB of monthly uploads for £10.99. | |
| Features to use: <ul style="list-style-type: none"> - Well designed interface that improves the web accessibility of the application. - Dark mode and strong layout is clear and concise. | |
| Source: https://evernote.com/ Costs: https://evernote.com/compare-plans | |

| |
|--|
| Application: Physical Journal/Notepad |
| Description: Similarly to physical storage, this is the physical solution to note taking, through a journal/diary or notepad. Widely used thanks to being so basic, only requiring paper and a pencil to use. |
| UI: |


Pros:

- Can be taken anywhere and used without internet access.
- Does not require premium membership to use.
- Cannot be hacked or leaked due to lack of digital presence.

Cons:

- Can be lost or damaged through accidents or the passing of time.
- Can be filled up, not unlimited storage and will require purchasing new journals to fill.
- Does not offer any integrated sorting or search filters.

Price Plans:

Cost of buying pen/pencil and the journal itself varies. No price to simply write down your thoughts.

Features to use:

- Being accessible anywhere and small enough to carry around with.

Source: N/A

Costs: N/A

2.1.3 Summary of Existing Solutions

Whilst researching the existing digital and physical solutions to the problems; I was shown the popularity and widespread use of the Apple iCloud which I did not mention in any of my research above. The reason for this was simply due to it relying on Apple specific devices and the requisite Apple account which I feel limits the overall accessibility of the application, even though it offers a premium file storage and note taking functionality. Unfortunately, however I do not own an Apple device or account and felt it would be unfair to compare a product which I cannot personally access for research.

As a result, the limitation of being solely Apple products supported meant that I felt it was not a competitor within the existing solutions as I aim to provide an application that can be used by anyone on any device. Apple being a premium brand, brings with it a premium cost in both device ownership and all attached accounts, which is unfair and limits access to those who can afford to pay entry. I aimed to create a platform which would be free to use for all who qualified and Apples reliance on their own devices hinders this option.

Coming back to the existing solutions I was able to access, I have gathered together the main strengths and weaknesses across all of the applications to gain a better understanding of areas I can learn from and things to try and avoid within my own design.

Strengths:

- All the applications reviewed have uninterrupted transition between mobile and desktop use meaning that all functionality is mobile responsive and mobile first designed, resulting in a smooth and highly functional layout across all devices.
- Layouts and designs were practical and clear, comprising of a main body alongside searching and filtering options via categorisation and other filters. This is effective easy to use meaning there is little to no learning curve when first using the applications in question.
- Accessibility to stored content is painless and simple to operate, with all applications allowing for a download to local device or viewing within the application.
- Creation of notes is also incredibly simple and does not overcomplicate what is a very basic task when done using pen and paper in the real world.

Weaknesses:

- Storage size is limited by pay walls that dictate to what extent you can use the applications. With all offering some basic level for free that will inevitably fill up quickly, especially when holding photos and other large document formats.
- The obvious weakness to all the physical solutions is the lack of safety and how easy it is to lose documents.
- Colour schemes although changeable in some applications, others do not even offer dark mode, which greatly decreases the web accessibility for those who need it. Options to change text size and colours would all be greatly beneficial for both personalisation of applications but also to improve platform visibility.

It was clear after this evaluation that I wanted to use aspects from all the reviewed applications that currently meet my requirements. With Microsoft's OneDrive and OneNote being the simplest to use and offering the most at their free tier (OneNote being entirely free), as well as the importance in mobile responsive design implemented by all, particularly Evernote which offered a premium feel to note taking. I wish to take these ideas going forward with my own designs, ensuring that my methods keep mobile and cross-device usability at the forefront to improve users quality of life across many devices. I plan to take this further, designing the entire application in a mobile stylised format with large buttons and iconography to clearly label and display all the on-screen information. Not only this but offering the entire application with relatively unlimited storage for free is of great importance so that no pay wall limits the applications ability to meet user's needs.

2.2 User Research and Feedback

Within the initial stages of research and design, it was important to get a better understanding of what real-world users want and the driving force behind the project. As a result, a couple of separate meetings were organised to allow for conversations to be had regarding the struggles currently being faced and features they would like to see come out of my project. The two main aspects that became clear as requirements from talking to the young people from NYAS (National

Youth Advocacy Service), were the need for more cloud storage than what is currently available for free on the market, closely followed by the idea that the application should be easily accessible, not overcomplicating the basic functionality.

2.2.1 User Research Data Storage

User storage capacity was the most common response when collecting information from the groups we spoke to. With the most popular cloud storage services currently available offering between 2GB-15GB within their free/basic tier, it is clear how this could easily become filled and therefore a limiting factor for functionality. Nobody wants to be faced with the full-storage message and be required to delete memories to make space for new ones or be forced into a paid subscription service to keep your photos and documents safe for another month. Speaking with one of the attendees, she spoke about how she had multiple accounts across all existing free services, having filled her OneDrive and Dropbox accounts to the brim with thousands of photos and documents from her time growing up within the foster care system, pictures of friends, pets and school reports.

The paid tiers of the competitor services ranged in price from £5 to £15 upwards for varying amounts of storage capacity and users. Although this may seem reasonable, in the case of foster care I would like to ensure that the free storage offered is more than enough to fill their needs. I estimate that the average user over several years in foster care could acquire dozens of important documents from social workers, family members or school boards. On top of thousands of personal photos and other memories that need to be stored, all of this would add up to hundreds of gigabytes of data storage which is the aim of this project.

Of course, without any financial backing, this project is simply prototyping how the application would look and function on its most basic level and therefore offering any substantial storage options is not going to be possible without paying for the server space via one of many potential services. Therefore, whilst in the real world I would take this project further and increase storage capabilities, the prototype will only offer limited data storage due to the lack of funding involved (I will discuss this later within the constraints).

2.2.2 User Research Mobile accessibility

The second most requested feature/functionality brought up during the research collection stage, was accessibility to the application. With many of the existing apps offering widely supported programs across both web and mobile platforms, it was obvious that in order to suit the users' needs and compete on the same level, the project would need to be highly web accessible and functional across all devices.

With one member of the research group requesting that the application supported all in-app functionality on both mobile and desktop versions of the program, explaining that it would benefit her if she was able to manage and organise her documents/photos on desktop, but have quick and simple access to her most common files on the go, thus removing the annoyance of having to carry around a laptop.

Whilst many of the existing solutions offer a mobile app alongside their desktop counterparts, it was unfortunately a limitation of the project in terms of development time that I would only be able to develop the application as a web-based system. However, in order to offer the functionality requested by the test group, I was keen to ensure that the application was completely mobile-responsive and functioning on every level which would later lead me to my mobile-first design.

2.2.3 Web Accessibility Research

From the user research, storage capacity and mobile accessibility were the two main features to be aimed for. However, after having spoken to one user who had dyslexia, the importance of web accessibility and ease of use for those who have difficulty accessing digital content became apparent. Most notably, the potential for personalisation of the UI for the application was mentioned, ideally allowing the user to choose specific colours and combinations of colours and font sizes that would allow users to both better read the content on the screen but improve attention capacity to ensure that details were not overlooked.

The user in question continued to illustrate her point by discussing the common option on software and other applications to switch to dark mode to improve readability and comfort on the eyes. Expressing that she would be keen to see a similar feature, going further with colour options and font sizes. This idea was great, and would allow the users to personalise the application to their aesthetic preference simply for the sake of it, making each account a little more personal and special rather than a uniform and corporate feeling application.

This direction of conversation also discussed the importance not overloading users with text and overcomplicated words, especially bearing in mind that the application was aimed at children and young adults, the project needed to come across as welcoming and not too complicated. On top of this, going back to supportability for dyslexic users, by ensuring concise information and stressing text of importance can massively improve application usability for those with learning difficulties.

On the 25th of February, myself and Caitlyn attended an online seminar titled “Demystifying Digital Accessibility with GDS” (Thomson 2020), in which the speaker talked at great length about areas of web accessibility that had been brought up with our user discussions and through our own research. The speakers discussed many aspects of design I had yet to consider, including the importance of colour and highlighting in text, to ensure key words are made clear enough for all users and nothing is misleading in anyway, further that the type of language used is simplified to increase overall user comprehension. Whilst aspects of accessibility such as colour and font sizes may seem obvious, ensuring that the word count and language used is kept precise and accessible to screen readers are not areas which I would have previously considered, and they definitely impacted my designs and wireframes later on in the project.

2.3 User Personas

Using the information gained during the user research discussion groups as well as looking at existing solutions and other third-party research into web-accessibility I have created a couple of user personas that I feel allow me to better target my features and applications during the design and implementation stages of the project.

The two personas below are realistic portrayals of the type of user I am aiming my application at (within the care system). The personas include the user’s needs, wants and requirements for such an application for them to use it in the real-world.

Name: Cai Hughes

Age: 14

Technical Competency (out of 10): 9/10

Disabilities: None



Description: Cai is super active in the digital world; he loves to play video games with his friends and keep everyone up to date on his activities through regular social media posts consisting of photos and messages. He spends much of his time on his phone or laptop when at home and knows his way around the devices well. Whilst still at school he has a lot of contact with his teachers and social workers which leads to a lot of paperwork that he has been storing online but keeps running into storage capacity problems. He also wants somewhere to back up all of his regular photos to free up space on his devices to allow him to keep taking more photos of his daily activities including trips with friends and pictures of his dog Rex.

Aims of Use:

- Back up all of his photos to free space on his devices.
- Manage and organise his documents more easily whilst young and receiving paperwork on a regular basis from numerous different sources.
- Is very digitally minded and wants to be able to access everything from his phone so he is not stuck on his computer.
- Wants a large enough storage capacity that will keep him going for a number of years and will not regularly bother him with notifications.

Name: Sara Evans

Age: 17

Technical Competency (out of 10): 5/10

Disabilities: Dyslexia



Description: Sara is a rather quiet teenager who has been diagnosed with dyslexia, as a result she struggles to access many digital tools offered to her and wishes that everything had improved web-accessibility for dyslexic users. Sara is also quite forgetful and likes to daydream so has filled a notebook with important notes and general thoughts which she carries everywhere with her. She does not take too many photos, but the ones she does have are very precious to her and wants to know they are safe and far away from danger at all times. Spending most of her time with friends or in the great outdoors, she has not had as much experience with technology as many other people her own age, therefore can easily get confused when digital tools aren't simple to use or offer too much functionality without any explanation.

Aims of Use:

- Wants a secure and safe place to store her photos of loved ones.

- A simple UI that allows for ease of use and that does not require any extended training to use.
- Customisation options to improve the accessibility for dyslexic users including colour and font changes.
- A note taking functionality to replace her growing collection of notepads/journals.

2.4 Project Justification

From end to end of the research process into existing applications, speaking with my target users, and attending other research sessions I was made very aware that the project was indeed necessary. Having been brought on by Children's Services and Cardiff Council for this project, I was initially met with the feeling that there must already be an existing application that suits the users' needs and would therefore not justify the need for my own attempt to meet the brief provided. However, having completed all of my research I proved myself wrong by finding that the existing options were held back by numerous constraints that I felt I would be able to overcome.

With roughly 65,000 children within the UK living with foster families as of March 2020 (The Fostering Network 2020), and more specifically 4,990 children in the foster care system in Wales there is a clear and large portion of the population that could benefit from the project I was hoping to build. On top of this, there are only 3,700 foster families within Wales, meaning that many families were caring for multiple children simultaneously which further solidified the clear need for a system that would better allow foster children to organise and store aspects of their lives. Caring for multiple foster children becomes even more complicated when you must take into consideration the need to keep all paperwork and memories separate to avoid confusion and lost information, adding even more demand for my project from foster families as well as foster children themselves.

Having completed all the research, I knew this was a project I could see myself doing for all of the right reasons, not only could I easily see how the project would benefit the lives of thousands of people including children and adults; but I could also see where existing solutions had fallen short and could be improved to meet the very specific user requirements given to me.

For the project to be deemed a success by both myself future users, the application would need to offer:

- A mobile first design that will allow all users to seamlessly switch devices and have a fully functioning application wherever they are. A user must be able to complete all functions from both a mobile phone or a desktop computer.
- A large storage capacity that will ensure safe storage for all documents, photos, and thoughts for all users. Providing free storage without any tiers or paid elements to ensure that any barrier to entry is completely nullified.
- Options to personalise and customise the user interface in order to maximise web accessibility for all users. As well as to make each account more personal and therefore private.
- An intuitive design that is both simple but functional and enjoyable to use. Thinking mobile first with clear navigation and user feedback.
- Functionality that allows for users to store photos, documents (PDFs), journal entries and calendar alerts. Acting as a centralised application for organising the lives of foster children that will better their lifestyles and organisation of important elements of their daily lives.

- A secure and safe place for all personal affects to take the pressure off safe keeping all documents. Encrypted user accounts with databases of data stored in secure locations with numerous levels of protection.
- Coherent and consistent feedback to ensure the user is fully aware of all functions of the application and the processes being completed. As it must always be obvious what is happening on screen to avoid confusion to the less technically minded users.

2.5 Project Constraints

When planning for this project, there are several constraints that will likely impact the finished article. I have taken the time to discuss the constraints and their implications further, in order to minimise their effect on the project.

2.5.1 Financial Constraints

As a student project, the most frequently experienced hurdle in a project of this scale and using the technologies in question is that of finance. Whilst it is possible to design and create an application of this kind simply using code and a free development software, the question of hosting and in particular the hosting of the database is all down to money. Much like the existing solutions I had looked at during my research, the option to host my website and related database was offered by numerous different services with free tiers, however none offering more than 15GB of free storage (Amazon Web Services S3 Bucket). Any more than this or using a different server host would consequently incur substantial expense to store the levels of data that I would ideally want to offer if this project left prototype stages and went into production. If I were to attempt to offer unlimited storage to thousands of users, the result could be many thousands of pounds in expenses for the database alone. Talking specifically about Amazons Web Services and using their own estimate calculator, I would be having to spend \$100 USD minimum to host just 5000 GB of data for a month in London, this is excluding any extra costs for backup storage which may be desirable when storing personal documents.

Tiered price for: 5000 GB

5000 GB x 0.0240000000 USD = 120.00 USD

Total tier cost = 120.0000 USD (S3 Standard storage cost)

S3 Standard cost (monthly): 120.00 USD

Figure 1 - Amazon S3 Monthly Estimates for Data storage

Clearly this is not affordable, and I aim to produce to the entire project with no budget. As a result, the main constraint for the prototype is the storage capacity I will be able to demonstrate in the finished version. Whilst ideally, I would promise unlimited storage to my users, in the case of testing this project, storage will be very limited to ensure no issues are caused and all test users have the capacity to test the functionality even in a finite case such as this. Further expense could be incurred if I were to properly host the project on my own domain, however for my prototype there were enough free options available to allow me to sidestep this financial hurdle.

2.5.2 Timescale Constraints

The next major constraint to the project was that of the given timescale. Whilst we began in February of 2021 and had until May to having a working prototype and final report, there was a

great deal to try and fit into this timeframe. Not only was there a great deal of research required on the subject and deliverables, but I also needed to fully design and implement a working front-end and backend to the project to ensure a usable and functioning prototype by the final deadline.

Prior to starting this process, I had never undertaken a project of this scale, starting from nothing, and having to build up upon every stage until I had a finished product. On top of this I would be working independently when it came to actual design and eventual implementation which was a daunting task for one so large. The entire timeline was only fourteen weeks and as a result it was key for me to be able to break this down into manageable chunks to avoid looking at the big picture and instead at each step that I needed to cover. I managed this by producing a Gantt chart within my initial report and design phase, that allowed me to visualise the aspects I would have to cover in a timely manner and keep me on top of the overall schedule. The Gantt chart was produced simply within Microsoft word, but worked well enough to allow me to tick off tasks as and when I completed them. However, at the end of the day I also knew that given the time available I would still need to be careful not to set my sights too high and fall short with a broken prototype. I really wanted to finish this project with a basic but working prototype of a high standard, therefore planning my time efficiently would help me achieve this.

| Date | Deliverables | Documentation | Milestones |
|---------------------|---|--|--|
| Feb 1 - Feb 2 | Complete Research Ethics training program. | Receive my training certificate for ethics. | Submit my certificate to supervisor. |
| Feb 8 – Feb 21 | Application/Methodology Research for project. | Set-up of chosen language/program. Write up of approach. | Decision made for application methodology. |
| Feb 22 | Initial Research collection from Children's Services meeting. | Notes from discussion regarding key elements and functions required. | A final list of key functions and project targets in mind. |
| Feb 19 – Feb 27 | Wireframe design for project UI and flow | Wireframe notes and prototypes | Discuss and finalise a single wireframe from user input. |
| Feb 27 – Mar 19 | Front-end design with final wireframe design | Write up of the front-end code for visual aspects of project. | Completion of initial front-end design for review with test group. |
| March 20 – April 20 | Back-end coding with database setup. | Write up of back-end code and database design. | Database tests working as expected. And website secure and functional with |

Figure 2 - Gantt Chart containing my project schedule, deliverables and milestones

2.5.3 Coronavirus Constraints

Finally, there was the obvious constraint that came from the ongoing covid-19 pandemic. Whilst it might not have directly impacted my ability to code or produce the project, the impact that it did have was on my background research and contact with support. Unfortunately, the coronavirus had limited my access to university resources, meaning all work was produced on my own personal computer which required a great deal of initial set-up, installing the correct software and packages (which did hinder my timely progress in the early stages of development). Not only this, but I was also kept from meeting directly with my supervisor or any members of the client party, which resulted in all communications being made via email or Microsoft Teams. Although this solution was perfectly adequate for my needs, it did slow down response times in some situations, and towards the end of the project the user testing was no longer able to be completed in person. As it would be difficult to complete in person user testing without breaking covid restrictions, specifically social distancing. All the testing needed to be completed remotely which further added complications at such a late stage, which resulted in testing being carried out on a smaller less intimate scale.

3. Methods of Development and Planning

3.1. Agile Development Methodology

In order to compensate for the lack of a development team (as this was an independent project), as well as the timescale I was faced with I chose to follow an Agile approach in my development methodology. I felt this was the safest and easiest method for my timeline, providing me with increased flexibility, better control, and clearer metrics for success in the form of targets/goals along my schedule. On top of this, due to my lack of experience of similar sized projects, the iterative approach within the agile methodology would give me time to both implement and learn throughout the process, whilst also providing extra time for research and bug fixes which were inevitable.

Using the previously mentioned Gantt chart, I was able to split the large scale of the project down into bitesize and considerably more manageable portions, with individual milestones to reach including expected completion dates for each. By giving myself this plan, I was allowing myself to better organise and efficiently arrange my time to ensure the development was at maximum efficiency and the project was completed on time with the requirements I aimed for completed and working. The degree to which I felt my project would be complete would change throughout the process, as and when I met my required targets for the main functions, allowing me to consider adding improvements or further functionality to the application.

Throughout the entire timeline, I gave myself time to allow for regular progress reviews with my supervisor to ensure I was on target; meanwhile keeping up with my own milestones. By having this time set aside I was able to keep my supervisor up to date whilst also addressing any issues or concerns either of us felt had arisen since the previous catch-up.

3.2 Learning

3.2.1 Web development

Early in the research and initial briefing of the project I felt it would be best to produce the application as a web-based application. Whilst it would most likely go on to become an android and IOS compatible mobile app, the initial prototype with functionality and design would need to start from a web-based application. Not only did I have no experience in developing mobile apps before, but realistically a project like this would always initially be developed as a web application first, before being ported into a mobile specific format and paired to the existing solution. Now that I knew I was going to develop the project as a web-based application, I needed to look at the methods of doing this available to me. I eventually decided the best option would be to develop the entire project from the ground-up using the Django Web development framework, this was down to my coding strengths lying largely within Python programming in which Django is written, as well as the endless documentation and support the framework offers from online resources (both official and third party). Not only this, but Django is a very powerful framework that comes with dozens of handy plugins and predefined packages that encourage rapid development and clean design.

When considering the project was being developed for Children's Services, I had to bear in mind that my program would likely need to be accessed later on by other developers. Django would allow me to develop using a popular and widely understood framework, creating clear structure and pragmatic design that could easily be picked up and worked upon by someone else. This would have been an issue if I had chosen to develop the project solely using HTML and CSS, without the aids of a

framework. Whilst there were other possible frameworks and languages I could have used for this project, Django offered me the best support given the requirements I was hoping to complete.

There were numerous frameworks I considered using, however as seen in this simple breakdown of the most popular, Django was a clear winner for my own needs:

| Framework/Library | Advantages | Disadvantages |
|-------------------|---|--|
| Django | <ul style="list-style-type: none"> - Written in python, offering little required learning to support the very powerful language. - Fast processing offered by Djangos architecture ensures smooth and efficient transmission over the internet. - Rapid development is easy thanks to being able to work parallelly on coupled components and using pre-existing configs. - Django offers great scalability, which is very important when creating a project with potential growth. The framework supports high traffic and increased number of tools and functions in new apps. - Due to popularity, Django offers high level of security with few loopholes all inbuilt and easy to use. | <ul style="list-style-type: none"> - With a wide variety of prebuilt packages and tools available, the initial learning curve can be a lot for newcomers. - Does not easily support smaller projects, due to the file structure and processing time. However, this will not be a problem for my own project. |
| Flask | <ul style="list-style-type: none"> - Flask is very simplistic offering fewer features than competitors which makes it very easy to learn and master. - Flexible framework that does not force the user down certain methodologies. Majority of the framework can be changed and moulded to the users' specific needs. - Testing is very easy thanks to unit testing and a powerful debugging tool. | <ul style="list-style-type: none"> - Being based around a single source, all requests must be handled one by one, meaning scalability is very limited and not recommended. - Flask does not offer great support for third party modules due to the risk of security threats. |
| Drupal 8 | <ul style="list-style-type: none"> - I do have some past experience with Drupal 7/8 which would help when beginning to relearn the framework. - Drupal is an easy to use open-source framework which brings with it a long of documentation and online support from a robust user community. - Provides many prebuilt themes and styles that would massively shorten development times for front-end and design processes. | <ul style="list-style-type: none"> - Poorly written plugins are badly supported and can cause security liabilities which I aim to avoid. - Use of modules can become redundant when they are abandoned by developers and lose updates. - Not as widely used as Flask or Django so does not offer the same level of scalability and maintainability if the project were to go on past the prototyping stage. |
| React JS | <ul style="list-style-type: none"> - Creation of dynamic web applications is easy thanks to in-built components that handle syntax and other irritants during development. | <ul style="list-style-type: none"> - Poor quality documentation due to the rapid development of the library meaning that methods change regularly, and the environment needs to be constantly relearnt. |

| | | |
|--|--|--|
| | <ul style="list-style-type: none"> - Components are often reusable which lends to faster development and scalability/maintainability. - Huge amounts of documentation and user support is offered (although must bear in mind that changes to the library are frequent). | <ul style="list-style-type: none"> -Rapid development is required due to the constant environment changes. Developers must reach production before a major update is released and breaks everything previously produced. - I have no experience with ReactJS so would require significant learning time at the beginning of the project which is not viable given the timeframe. |
|--|--|--|

After breaking down the advantages and disadvantages of all the potential frameworks/libraries at my disposal, I concluded that Django was the best tool for me. Django offered the high-level programming and scalability that I desired, whilst including numerous packages and in-built libraries that I knew I could use to rapidly and efficiently develop my project. Not only this but having already previously had experience with the framework within my first year module “Web Applications”, as well as the huge supply of documentation and community support thanks to its popularity, Django offered a stable and realistic entry point for me to learn and develop from.

As I already had some existing knowledge of using the framework, I was able to start the new project with little problem and did not find the need to specifically complete any online courses before I began. I did however utilize both the official Django documentation (Django documentation 2021) as well as the documentation available on the Mozilla Developer Network, which offered guides on all aspects of the framework and its uses. The use of these guides was useful throughout the project and gave me a better grasp of some of the fundamentals I would be using a lot, whilst providing insightful and useful examples that related well to what I was hoping to achieve. I felt that the explanations were for the most part very clear and ensured that I was continually learning during the implementation process. I most used the documentations when faced with an error as I found they usually offered the most accessible response, in one case when trying to store PDF files and display them to the user, Django was blocking the file source as a security risk. As a result, I looked at the guides provided by Django and found that it was required for me to state that all file sources from the same location as the existing page were to be accepted. I later also used the documentation a great deal to implement user groups, users and the login/logout features which can be accessed through prebuilt Django modules that improve security and general usability and maintainability for future development.

For further guide I did also use several YouTube series from various creators that substantially improved my understanding of various aspects to Django. The series produced on User Registration and Login Authentication (Ivy 2020) as well as Full-Featured Web Apps (Schafer 2018), offered quick and easy tutorials on these topics which I had previously little to no experiences on. I came across these series through searching and finding that these creators were very popular members of the Django community.

3.2.2 Data Storage

Throughout the project I was aware that I would need to host all the user data somewhere for the application to be safe and functional. With so many options available to me it was overwhelming trying to decide what was the best path to go down. During my research I looked at the possibility of using AWS S3 and RDS services to store all the relational data and user documents (whether that be files or photos), however having had a deeper look I felt this was not going to be a plausible approach due to the learning curve required to properly implement Amazon Web Services into the project. Although there is substantial AWS documentation available online, I felt that I would not be

able to fully utilize the technology correctly or have a proper understanding, which I felt was key to working on the project as I wanted to know I had learnt the technology I was using rather than copying examples from digital resources until my own worked. Other options also included creating a MySQL database for all my own data storage, however due to the time constraints and the constant changes that would need to be made throughout the development process, this was not a very viable option. I do not have a great deal of experience using MySQL for building databases from scratch and felt that I did not have enough time within my timeline to learn the required level of knowledge to correctly utilise the technology.

SQLite is an already supported format for data storage within the Django environment, even being prebuilt when you start a new project. As a result, I felt that this was the best approach for realistically meeting the requirements I had set out. SQLite offered a portable and simple approach to data storage, although it did not offer the security and scalability of MySQL, for the purposes of building my functional prototype it met my needs and did not require any external work or setup which massively helped me with my time management and progression for the project.

On top of this I also chose to host the files such as images and PDF documents in a static local folder for my prototype build. Whereas, in a real-world launch I would aim to store this data within an Amazon AWS S3 bucket, this was not an option at this stage due to the financial constraint for the service as well as the steep learning curve required to use the service properly. The combination of SQLite with the static folders has worked well for this small-scale project and has allowed me to implement private accounts, user data and storing all the files with little need for further instruction.

With Django offering preinstalled support for SQLite, it was an easy decision to use this database management system.

3.2.3 Children's Services and the Care System

When faced with a project for real-world application it was important for me to feel that I knew the industry and people I was developing for. Having had no personal experience with the care system or Children's Services (other than some contact with friends who grew up in the system), I felt very out of my depth in the beginning. Although this would not necessarily impact the final result of the project, I feel that from a developer point of view, it is always best to have the best understanding of the world of the user and what it is they experience and need from the application I hope to build.

Whilst initial research and learning was completed through reading of official government documentation and guidelines publicly available on the internet (StatsWales 2021), this mostly only offered a statistical view of the care system and did not give me a personal interaction with those my project would impact. However, later discussions with groups of young adults from the care system gave me a better understanding of the circumstances they were in and how their lives have been impacted by the problems faced in document storage. On top of this, regular communication with Thomas Phugsley from the Child Friendly City Programme and Samantha Anderson from the National Youth Advocacy Service continued to provide information regarding the world of foster care and guardianship which helped solidify my own understanding of the target user.

4. Specification

4.1 Functional Requirements

4.1.1 Must Have

-1 Requirement: The platform must allow users to upload PDF documents and images in any common format to be stored within their Document Wallet and Memory Box, respectively.

Acceptance Criteria:

- User can upload any PDF file from their local device and for it to be viewable from the platform after.
- User can upload an image in any common format (whether that be JPG, PNG or other) and for it to be viewable in app after.
- User must not be limited or blocked from adding any new files to the platform due to lack of storage capacity or other reason.

-2 Requirement: The application must allow users to submit journal entries containing diary entries or simply notes from the day.

Acceptance Criteria:

- A user must be able to submit a journal entry, and have it displayed on screen in the order than they wish.
- Journal Entries are submitted with an automated timestamp to aid in organisation and date the entry for later use.

-3 Requirement: The application must allow for organisation of all stored files.

Acceptance Criteria:

- Organisation can be used via adding categories, allowing users to create and use custom categorisation models to group sets of documents and images for easier access.
- The user must be able to filter all of their stored files, preferably in terms of date of upload (ascending and descending).

-4 Requirement: Users must have a private space through personal accounts in order to separate their data from that of other users and protect their files from being viewed by others.

Acceptance Criteria:

- User can login to the system via a secure login page with their username and password.
- If a user has not got an account, the system will block access to content through shared links or entry of the URL to the application.
- Accounts must be separate, with all data related to individual users having an owner class within the model that specifies which is to be authorised for which users.
- When logged in, no user can access the files from another account.

-5 Requirement: The platform must allow users to delete any stored data or other account content.

- All documents submitted and stored in application must allow be able to be removed through a delete functionality, removing it from the database.
- All other content must also be able to be deleted from the account.

-6 Requirement: All uploaded documents must have the option to redownload to the local device.

Acceptance Criteria:

- All files uploaded to the “Document Wallet” will be downloadable from a download button accessed from the document view screen.
- All images stored in the “Memory Box” can be saved to your local device through download and save in image viewing page.

4.1.2 Should Have

-7 Requirement: Users will have the option to share the content of their account with others.

Acceptance Criteria:

- All content from the “Document Wallet” and “Memory Box” features can be shared with others via email by clicking a Share button within the relevant pages.
- Email received by the recipient containing the content of the document shared by the user.

-8 Requirement: Account usernames and passwords can be reset for existing accounts.

Acceptance Criteria:

- “Forgot my password” option available at the login screen to allow users to reset their password in the case that they forgot the current password.
- Within the account, profile settings can be changed to alter the current username and password of the logged in user.

-9 Requirement: All stored files within the “Document Wallet” can be printed from app to any connected printer device.

Acceptance Criteria:

- From the view document screen, users can click an icon that will allow them to print straight to any connected printer.

4.1.3 Could Have

-10 Requirement: Users are able to delete their accounts from the system.

Acceptance Criteria:

- Any user should have the option to delete their account along with all the related account information and data within the database.

-11 Requirement: Users can edit their profiles to contain personal information and a profile picture.

Acceptance Criteria:

- User profiles can be edited to include profile pictures of the user’s choice.
- User profile pages can be edited to include important information related to the user, including contact details for their social workers, closest GP, and school name etc.

-12 Requirement: Calendar app can send alerts of upcoming events.

Acceptance Criteria:

- Upcoming events on the user’s calendar sends an alert when within 24hours.
- User receives an email or alert ping on their mobile device.

- When an event begins, a second alert is pinged to the user via email or mobile push notification.

4.2 Non-Functional Requirements

4.2.1 Must Have

-1 Requirement: All features of the application must be functional and accessible from both mobile and desktop devices.

Acceptance Criteria:

- Users can access their accounts from both desktop and mobile devices.
- Users can access all functionality within the “Document Wallet” and “Memory Box” as well as the other features from both a desktop and mobile device.
- Downloading any stored documents will download to your current device, whether that be mobile or desktop

-2 Requirement: The entire platform will be designed in a web-accessible format.

Acceptance Criteria:

- The colour schemes and font sizes in use will be web-accessible for the majority of users and improve visibility and readability of all on-screen content.
- Navigation and use of iconography and buttons will be implemented in such a way as to benefit those who struggle with accessing digital content. Creating clear paths of navigation and making distinct impressions for each feature and function available.
- Word count as well as the language used in the application will be kept at a manageable level and suitable for the majority of users.

-3 Requirement: The entire platform must be easily navigable, with as few actions as possible required for all processes.

Acceptance Criteria:

- Users can simply and easily get around the entire application without any direction.
- All processes in app can be achieved within just a few actions.

4.2.2 Should Have

-4 Requirement: Platform should provide support for users through other resources.

Acceptance Criteria:

- Platform provides quick links to other resources available for those in the care system.
- Application should offer support through access to support lines and other important information for those in the care system.

-5 Requirement: User accounts can be personalised/customised for better web accessibility and personalisation preferences.

Acceptance Criteria:

- Account customisation page allows for personalisation of the user’s accounts, including options to change the colour scheme, overall theme of their account and other smaller details including font sizes and font styles.

-6 Requirement: The project should be implemented in a maintainable fashion to allow for future developments.

Acceptance Criteria:

- All code will be written in a functional and readable way to improve maintainability.
- Code is formatted to improve readability and flow of functions within each module.
- Comments will be implemented within all areas of the code to ensure future developers understand the existing functionality and use of functions and classes.
- Existing codebase will be implemented with space to develop and grow the current platform. Project will be capable of scaling with growth of the user base and functions through simple appending of new features to current code.

4.3. Use Cases

4.3.1 Use Case Diagram

The following Use Case Diagram is used to demonstrate the way in which a user would interact with the application. The use cases included in the diagram are largely from the “Must Have” category of functional requirements as these are the features that need to be working for the project to meet the brief and to satisfy the majority of the functions I aim to complete.

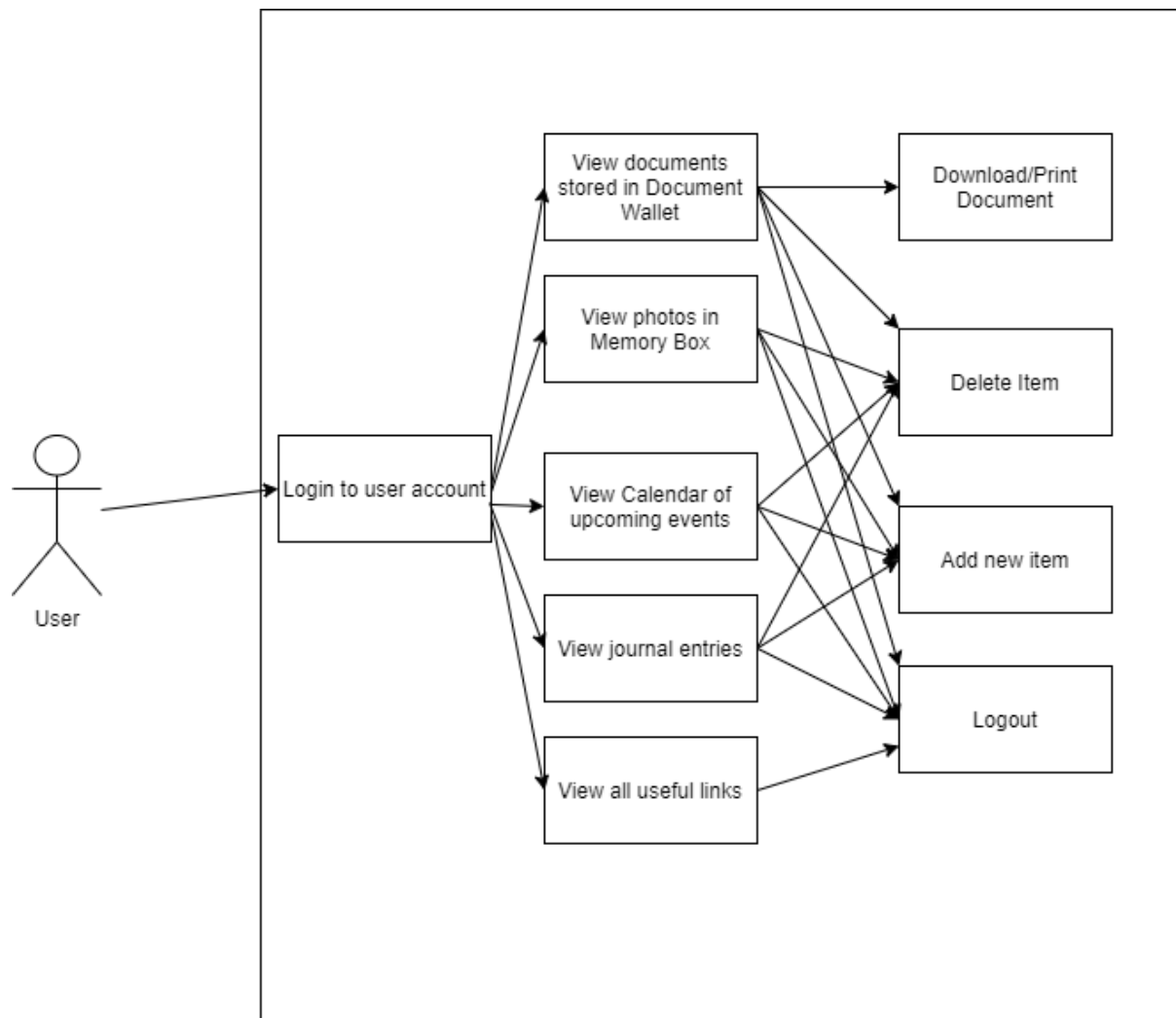


Figure 3 - Use Case Diagram displaying how a user would interact with the main app functions (must haves).

4.3.2 Use Cases

The following section will go into further detail of each of the use cases shown in the above Use Case Diagram. In this section I aim to demonstrate how a user will navigate the application as well as any alternative routes available to them, through Use Cases as well as flowcharts for each functionality path.

4.3.2.1 Use Case 1

| | |
|--------------------|--|
| Use Case: | User must be able to upload photos or delete existing photos. |
| Goal: | User wants to upload a selection of new photos to the "Memory Box" for safe storage. It must allow them to create a new category or choose an existing one, before prompting them to upload as many files as they like from the local device. |
| Basic Flow: | <ol style="list-style-type: none"> 1. User Logs in and is directed to the homepage. 2. User selects "Memories" from homepage. 3. User then selects "Upload new photos". 4. User completes digital form and uploads attached photos. 5. Fill in file(s) name(s). |

| | |
|--------------------------|---|
| | 6. Select a category from existing list or create a new one. 7. Attach all desired images using “Choose files”. 8. Upload images by pressing “Upload Images”. 9. User is redirected back to the “Memories” page and can view the newly added images. |
| Alternative Flow: | 3a. User chooses an existing image they want to delete. 4a. User is shown the chosen photo in the view image page. 5a. User clicks the dustbin icon to delete the image from the database. 6a. User is returned to the “Memories” page. |

Use Case 1 Flowchart:

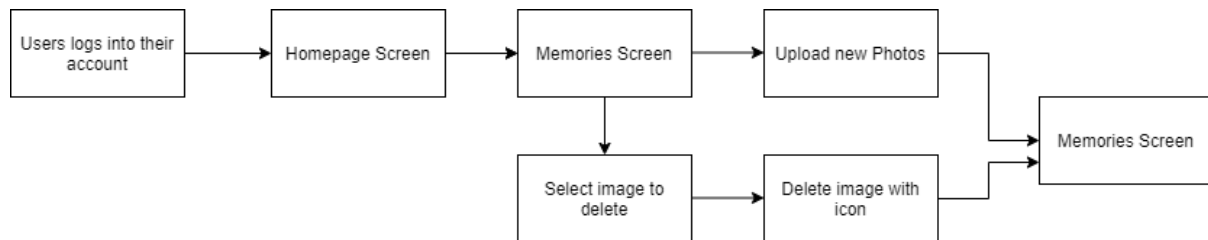


Figure 4 - Upload/Delete image flowchart

4.3.2.2 Use Case 2

| | |
|--------------------------|---|
| Use Case: | User must be able to upload PDF files or delete existing files. |
| Goal: | User wants to upload a selection of new PDF documents to the “Document Wallet”. It must allow them to create a new category or choose an existing one, before prompting them to upload as many files as they like from the local device. |
| Basic Flow: | 1. User Logs in and is directed to the homepage. 2. User selects “Document Wallet” from homepage. 3. User then selects “Upload new documents”. 4. User completes digital form and uploads attached documents. 5. Fill in file title. 6. Select a category from existing list or create a new one. 7. Attach all desired files using “Choose files”. 8. Upload documents by pressing “Upload Files”. 9. User is redirected back to the “Document Wallet” page and can view the newly attached files. |
| Alternative Flow: | 3a. User chooses an existing document they wish to delete. 4a. User is redirected to document viewer screen. 5a. User clicks the dustbin icon to delete the document. 6a. User is returned to the “Document Wallet” page. |

Use Case 2 Flowchart:

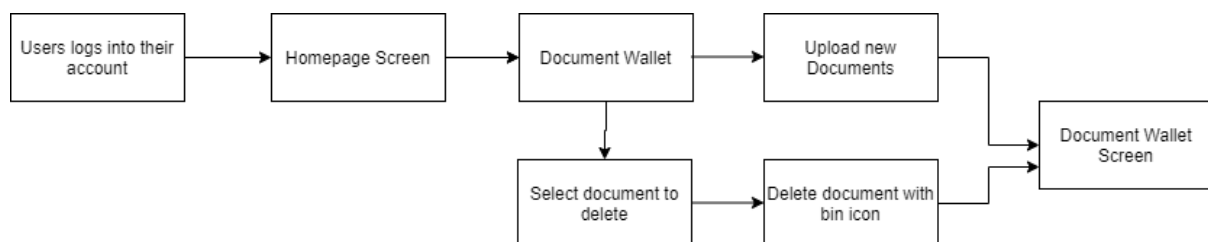


Figure 5 - Upload/Delete Document Flowchart

4.3.2.3 Use Case 3

| | |
|--------------------------|---|
| Use Case: | User must be able to append new calendar events. |
| Goal: | User wants to add a new upcoming date/event to their calendar. Calendar must allow for the user to create a new event with date, title and description. |
| Basic Flow: | <ol style="list-style-type: none"> 1. User Logs in and is directed to the homepage. 2. User selects "Calendar" from homepage. 3. User completes digital form. 4. Fill in the event name. 5. Write a description of the event in question. 6. Select a date for the event from built-in calendar. 7. Add entry to calendar by clicking "Add Entry". 8. New event is displayed beneath the form in ascending order of date. |
| Alternative Flow: | 3a. User chooses clicks bin icon on event they wish to remove from the calendar. |

Use Case 3 Flowchart:

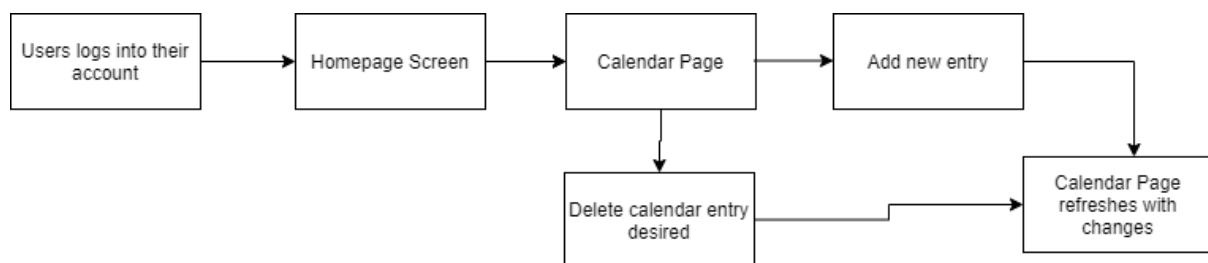


Figure 6 - Add/Delete Calendar Entries Flowchart

4.3.2.4 Use Case 4

| | |
|--------------------------|--|
| Use Case: | User must be able to add new journal entries and delete existing entries. |
| Goal: | User wants to add a journal entry or set of notes from a meeting to the "Journal". It must allow them to create a entry to the journal and choose from private or public categories before being uploaded. |
| Basic Flow: | <ol style="list-style-type: none"> 1. User Logs in and is directed to the homepage. 2. User selects "Journal" from homepage. 3. User then selects "Add Journal Entry". 4. User completes digital form to create new entry. 5. Fill in file title. 6. Select a category from dropdown. 7. Enter journal entry. 8. Add entry to journal by clicking "Add Entry". 9. User is redirected back to the "Journal" page and can view the newly entered entry. |
| Alternative Flow: | <ol style="list-style-type: none"> 3a. User chooses an existing entry they wish to delete. 4a. User is redirected to entry viewing screen. 5a. User clicks the dustbin icon to delete the entry from the journal. 20a. User is returned to the "Journal" page. |

Use Case 4 Flowchart:

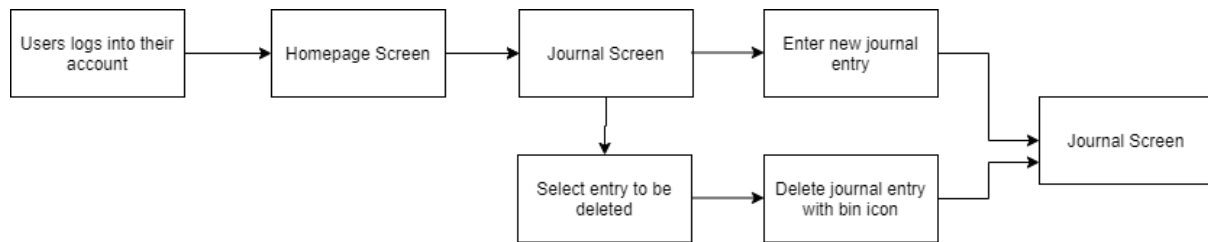


Figure 7 - Add/Delete Journal Entry Flowchart

4.3.2.5 Use Case 5

| | |
|--------------------------|--|
| Use Case: | User should be able to access other useful resources from other sites. |
| Goal: | The user should be able to find links quickly and easily to other resources available for those in the care system. |
| Basic Flow: | <ol style="list-style-type: none"> 1. User Logs in and is directed to the homepage. 2. User selects "Useful Links" from homepage. 3. User can click on any provided links from this screen and are redirected to other sites. |
| Alternative Flow: | No alternative route available for this use case. |

Use Case 5 Flowchart:

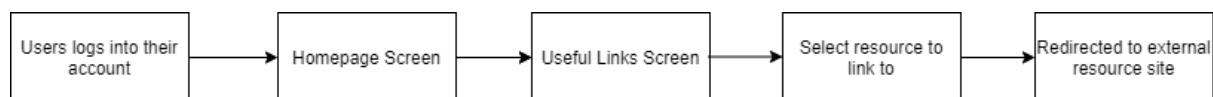


Figure 8 - Accessing Useful Links Flowchart

4.3.2.6 Use Case 6

| | |
|--------------------------|--|
| Use Case: | User must have their own private account. |
| Goal: | The user must have their own private account which only displays their personal information and not the data of other users. |
| Basic Flow: | <ol style="list-style-type: none"> 1. User uses their personal account information to login through account portal. 2. User can only see content related to their own account and no other user information. 3. User can logout of the account at any time. |
| Alternative Flow: | No alternative route available for this use case. |

Use Case 6 Flowchart:

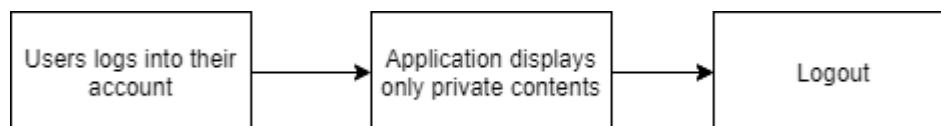


Figure 9 - Login and Logout Flowchart

4.3.2.7 Use Case 7

| | |
|------------------|---|
| Use Case: | User must be able to print or download any stored documents from the Document Wallet. |
|------------------|---|

| | |
|--------------------------|--|
| Goal: | If a user needs to have offline access or regular access to a specific file, they have the ability to download or print the chosen file to a local printer or have it saved onto their current device. |
| Basic Flow: | <ol style="list-style-type: none"> 1. User Logs in and is directed to the homepage. 2. User selects “Document Wallet” from homepage. 3. User selects the document they wish to download/print 4. User can choose the icons to download or print the current document and following any further printer instructions. |
| Alternative Flow: | No alternative route. |

Use Case 7 Flowchart:

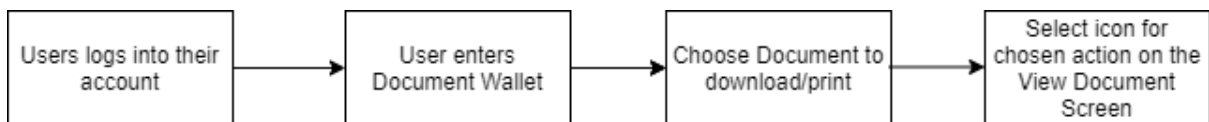


Figure 10 - Download/Print Document Flowchart

5. Design

5.1 UML Class Diagram

The following UML Class Diagram is used to portray the basic structure of the application, including all of the main functional classes.

I have however avoided adding every variable and method for the sake of clean design and ease of reading (for example all pages can navigate to all other features).

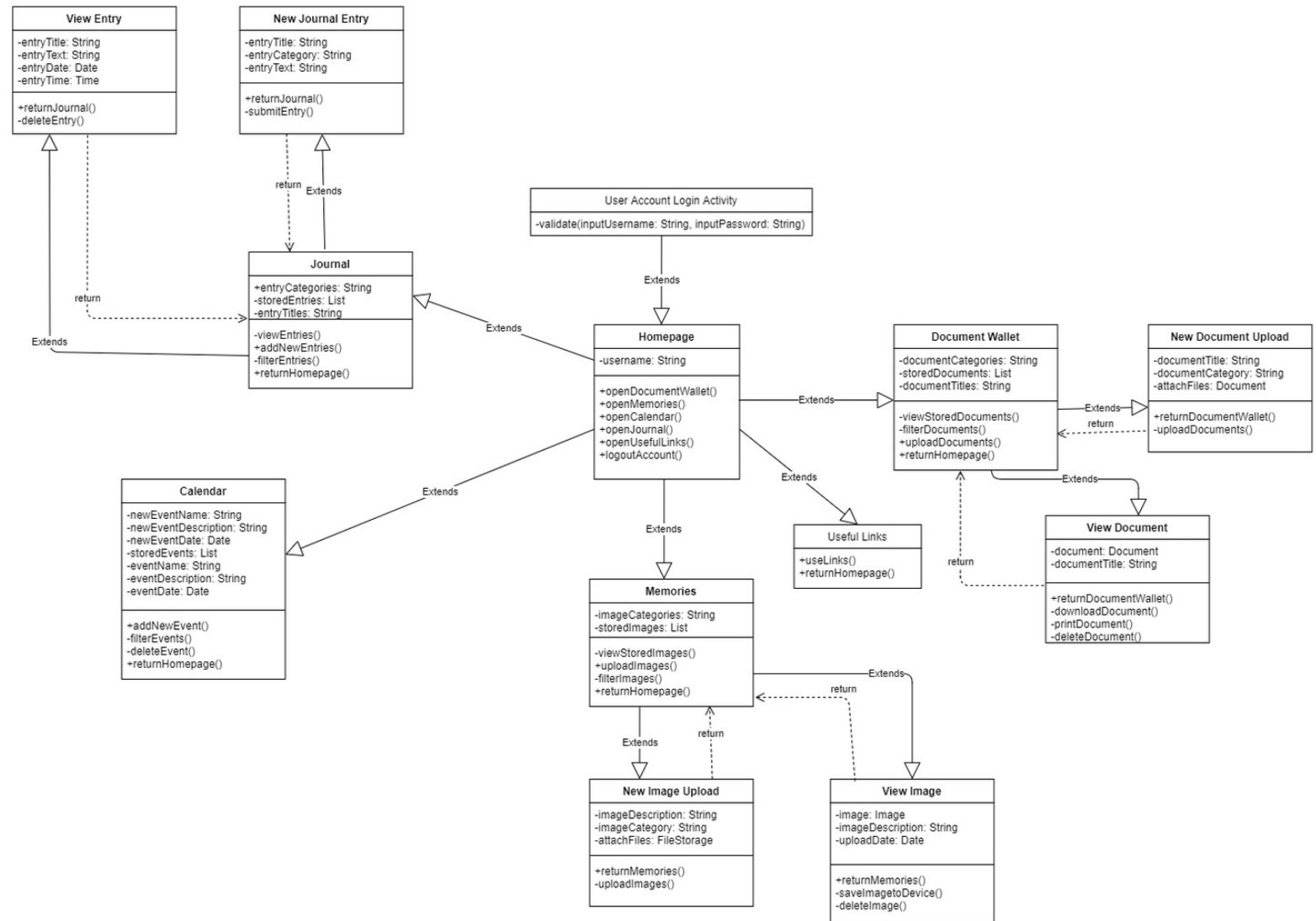


Figure 11 - My Important Things App UML Class Diagram

5.1.1 UML Overview

The primary class for the “My Important Things App” is **Homepage**. From here the user is able to access the rest of the classes and the additional content within each. In order to initially reach the Homepage, the user must Login to their account through the portal, from here the diagram then extends to Document Wallet, Memories, Calendar, Journal and Useful Links.

Within the **Document Wallet** the user can see a list of all the currently stored files they have on their account, this consists of buttons containing the document title given by the user upon creation. From this page, the user can also filter through the documents, either via date or by category (School, Medical, Social Services etc). Document Wallet extends again to **New Document Upload** in which the user is able to upload a new document, containing variables such as document title, related category as well as attaching the desired file. Returning to the Document Wallet, this can also extend to **View Document** which allows the user to inspect the file chosen by selecting it from the previous screen. Within View Document the user has the option to delete the file from their account, as well as print or download the file to the local device. This whole path of the UML Diagram will only handle PDF files due to the complexity of hosting other formats online without access to Microsoft services.

Memories is the next biggest feature within the diagram, again extending from the Homepage, users are faced with a grid of all their currently stored images. Similarly, to other classes within the UML, users can again filter their images by category or ascending/descending age from upload date. Memories extends to **View Image**, which allows the user to view the image as well as image title and postdate. The user can save the image to their local device or delete the image from this page also. An alternative route Memories will take the user to **Upload New Images** which will take user inputs of image title and category, before uploading attached images to the database to be seen within the **Memories** page.

The **Journal** class has the same functionality as the Document Wallet, allowing users to initially see a list of all their currently stored journal entries, this is just another button containing the title, date and timestamp of the day it was originally submitted. From here the class extends to **New Journal Entry** and **View Entry**, which allows the user to submit a title and entry text, before choosing whether they want this entry to be private. View Entry, allows the user to view the full text of the entry they chose at Journal as well as the title, date and timestamp; they can also delete the entry from here but don't have the option to update the entry as I didn't want users to be able to edit past journal entries.

Calendar is kept to a single class and does not offer any extension, keeping the functionality to a single page. The user can submit a new calendar event, with event name, description, and date. Or view the existing calendar events below the form, which displays a list of details for all current events. The user can filter by date and delete any events from this page as well.

Finally, the most basic class simply extends from the Homepage and this is **Useful Links**, which offers links to other digital resources for the user to access as they please.

There are a number of functions that are consistently available to the user and as a result were not included here or within the UML diagram. Most notably this would be the available navigation bar which allows a user to access all other functions on the application, as well as **Logout** of their account once they have finished. From all functions there is also access to the SQLite database which I have avoided linking to simplify the UML, however I have referenced specific variables when used within each class.

5.1.2 Use Case Mapping

In order to make certain that I was meeting the Must Have use cases, I have mapped these cases within the UML to display how each requirement will be met.

Memories:

- **Use Case 1:** To allow for a user to upload as many image files as they want and delete existing images.
- Through the extensions of Upload New Images and View Image the user can manage their memories easily.
- Upload New Images offers a digital form for attaching and detailing any new uploads.
- View Image provides a page to view and delete any unwanted content.

Document Wallet:

- **Use Case 2 and Use Case 7:** To allow the user to upload any PDF documents and delete currently stored documents. Or print/download documents from the application.
- New Document Upload offers an easy digital form method to attach and detail any new files a user wishes to store.
- View Document will allow the user to delete, print or download the file in question.

Calendar:

- **Use Case 3:** User must be able to add new events to their calendar.
- Within the calendar function, user can add new events to the calendar through the provided form.

Journal:

- **Use Case 4:** User can add and delete entries from their personal journal.
- The journal function can allow the user to visit View Entry to read and delete any current journal entries.
- The extension to New Journal Entry will allow users to submit new entries through the provided form.

Useful Links:

- **Use Case 5:** Extra resources are available to the user via attached links.
- Useful links is very simple and offers a list of useful resources via URL links to other sites within the care system.

User Account Login Activity:

- **Use Case 6:** To allow for each user to have a private account to securely store their content, I need a login screen, this is achieved using my Login Portal.
- Username authentication selects which account the user is trying to access.
- Password authentication ensures user has permissions to access this account.

Having now been able to cover the requirements I need to achieve; I can begin to think about potential User Interface designs.

5.2 User Interface Designs and Wireframing

During my research both online and with the test user groups, the clearest priorities from a design standpoint were that the application needed to be easily navigable on any device, meaning simple and clear layout for any device whether that be desktop or mobile/tablet. Next was the need to make the platform as web accessible as possible, aiming to offer high contrast colours, clear icons and understandable text. With this in mind I delved deeper into these areas of mobile first design and web-accessibility.

To guarantee the usability of the platform I also aimed to incorporate design principles such as Nielsen's Heuristics (Nielsen 1994), which covered aspects such as recognition, user freedom and minimalist design. Other ideas such as those noted by Adobe (Babich 2019) include the importance of giving the user control, providing a forgiving environment that makes the user feel more comfortable. When applied correctly to an application, these principles can massively improve the usability and positive feedback from users.

For the wireframes (or mock-ups) produced I used Balsamiq Cloud (Balsamiq, 2021) which is a powerful, web-based software for user-interface design. Having been previously exposed to the software during my Human Computer Interaction module in first year, I felt this gave me a familiar yet suitable tool to begin my designing on. I also felt that Balsamiq offered a strong range in icons and features that I knew I would need to have the desired effect on interface usability.

5.2.1 Mobile First Development and Intuitive Design

The UI (User Interface) needed to be designed with mobile users in mind. Whilst "My Important Things" is being aimed at all devices, in order to offer the flexibility of access and ease of use I aim to achieve, designing the application for mobile users and then porting the application to a desktop format was the way to go. Not only is it easier to move from a mobile to desktop design (rather than visa versa), but with 52.2% of all global web traffic being from mobile devices (Angle Studios 2020) in 2018 and this percentage only growing since, it was not hard to imagine that my target market was also going to likely fit into this bracket. Furthermore, with my audience being within the care system, the need to have easy access to important documentation regarding their school results, social workers, and social services paperwork, offering this from a mobile device is much better than expecting my users to always have access to a computer or print their documents. Whilst I was not specifically designing for Apple devices, I did refer to the Apple iOS developer guidelines on iOS design (Apple 2021). This guide offered general advice for developing for a mobile device and solidified principles that would benefit user comprehension of the platform. An example of this would be use of well-known icons and terminology, which benefit the user who most likely have already seen these terms used in other UI and the real-world. This guide along with the principles laid out by Nielsen reminded me to keep the design sleek and minimal to keep clarity high.

Example: Use of iconography throughout the application

By implementing the use of icons that users may have already seen and used before, gives the user a starting knowledge base the moment they open the application for the first time. This use of recognisable symbols can settle a new user and make sure they feel more comfortable in a new platform.

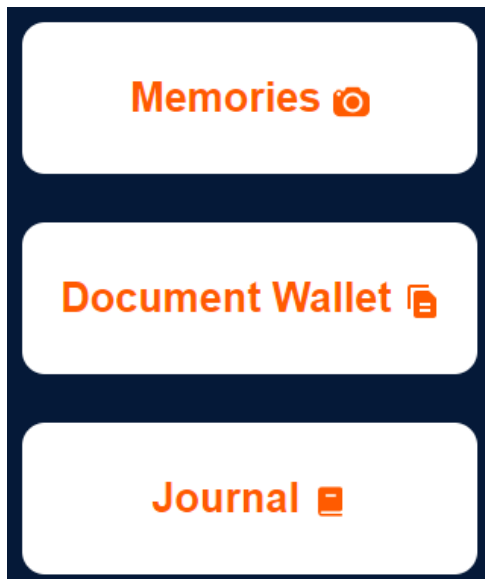


Figure 12 - Example of Iconography

Example: Navigation

Offering a navigation bar throughout the platform, can keep the user connected and make the features feel more flexible and user led. At no point should the user feel like they are being led down a linear route without the option to change direction, on top of this adding back buttons and navigation allows the user to undo mistakes they may have made using a feature, giving control back to the user which is an important theme highlighted within the iOS developer documentation.

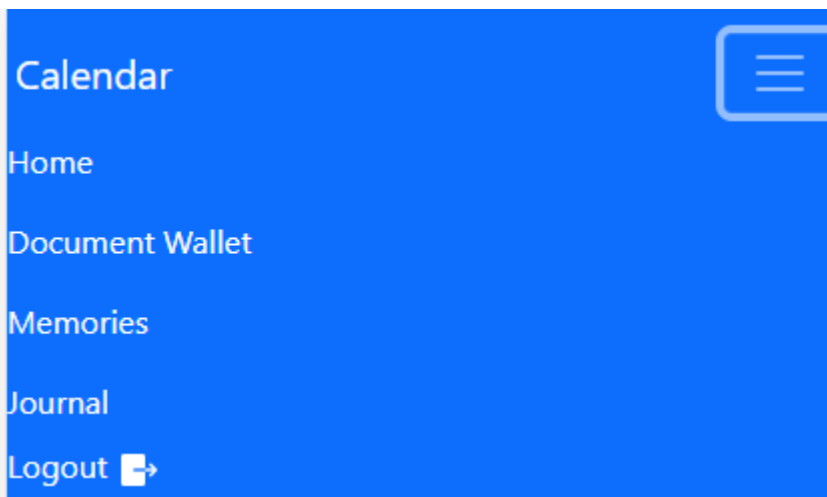


Figure 13 - Example of Navigation Bar

Example: User Feedback

Going back to Nielsen's heuristic design principles, a key idea is that of the "visibility of system status" and user feedback. It should be clear to the user at all stages what the system is doing, whether that be loading a new screen or highlighting a key point. The example below shows how when a user hovers over an icon within the main menu, it highlights by changing in appearance to clarify what the user is looking at. This is also a great benefit to those with learning difficulties such

as dyslexia, as it provides reinforced feedback to user actions which can help with concentration and user understanding where a plain button may be unclear.

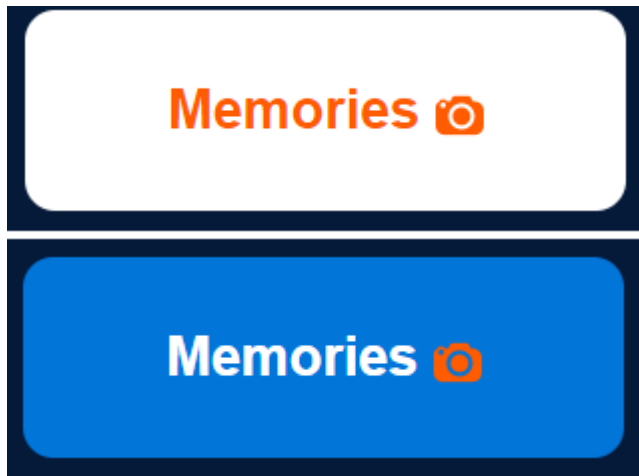



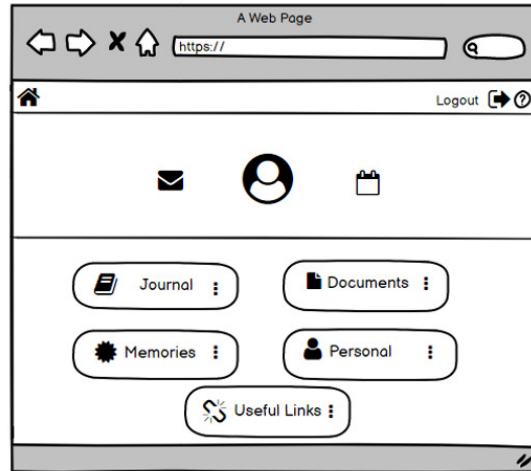
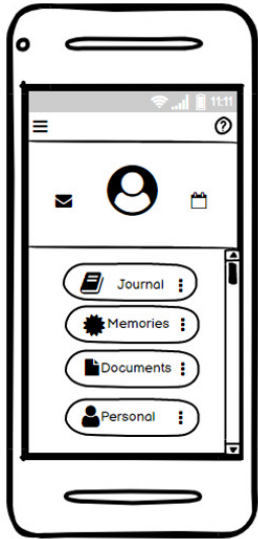
Figure 14 - Example of User feedback through button colour change

5.3 Prototype Designs

5.3.1 Wireframes

The following prototype designs were built before any final use case changes were made; therefore, wireframes may include features that did not make it into the final version of the platform. The designs however were created with the user personas in mind, specifically that of Sara Evans who had dyslexia and would require specific design elements to create a functional platform.

| Wireframes | Description |
|---|---|
|  | <p>Login Screen: This is the login screen for the entire application. It requires a username and password to access, which can be submitted into the relevant text fields before clicking the “Login” button.</p> <p>The create account link was later removed from designs. This was due to the service being for Children’s Services and therefore any users will be provided with an account rather than being able to create new ones as they wish.</p> <p>The profile photo was initially put in place to suggest the platform inside was a social media, centralised hub type of application which I hoped would appeal to the younger audiences. I later decided this was misleading as the accounts were meant to be private and not shared.</p> |

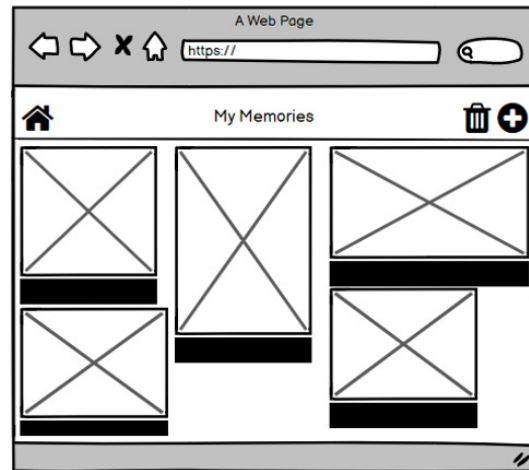
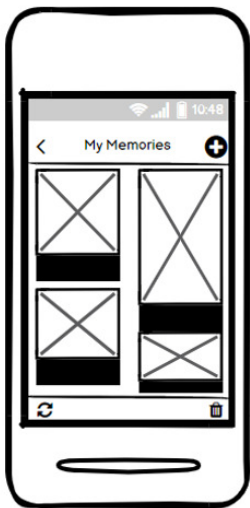


Homepage/Main Menu: The homepage is the hub for the entire application, therefore access to all the other features was available here.

To keep inline with principles of minimalism laid out by Nielsen and the iOS guide, I chose to keep the desktop version feeling very mobile oriented, with larger-than-life buttons for user interactions. I felt that this made the purpose of the app clearer whilst not overcomplicating or cluttering the screen with numerous panels and excessive text.

Further use of iconography on the menu buttons continued to confirm user ideas of each feature and better clarified each function without the need for written descriptions.

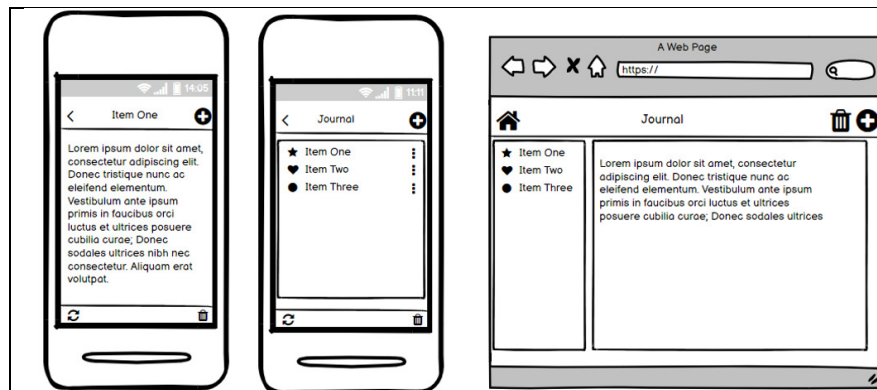
As well as this, the layout provided a seamless transition between mobile and desktop which I felt was important to ensure users did not feel lost if they accessed the platform from a new device. By creating a very similar layout across all devices, you solidify the platforms' theme and keep all variations unified.



Memory Box: The purpose of this page is displaying all of the user stored images. I aimed to keep the wireframe here minimalist to allow the user to not be distracted by any other on-screen elements. When a user wants to look at their photos, they should not have to work around other areas of the screen.

Inspiration for this page came from Pinterest (Pinterest 2021) with its feature board layout which appears fun and inviting. Images on mobile stack to allow sizes to stay at a viewable scale.

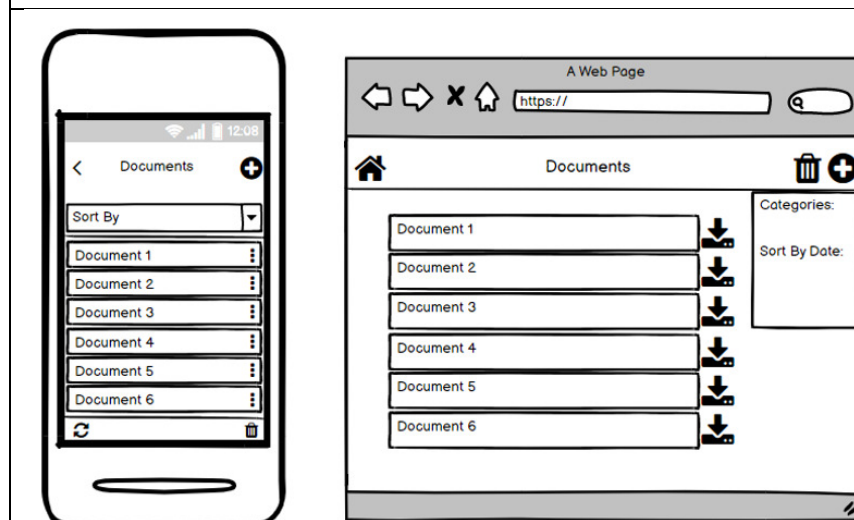
A description of the images is added beneath each to separate and provide any extra details the user felt necessary.



Journal: The journal feature is there to allow users to make notes of events or jot down quick diary entries quickly and easily for each day.

In order to keep the user from thinking too much about the process, I wanted to keep the functionality as natural as possible, therefore a user can create a new entry within just a few actions of the mouse and keyboard. In order to replace the notepad and pen, it needed to be as easy to use.

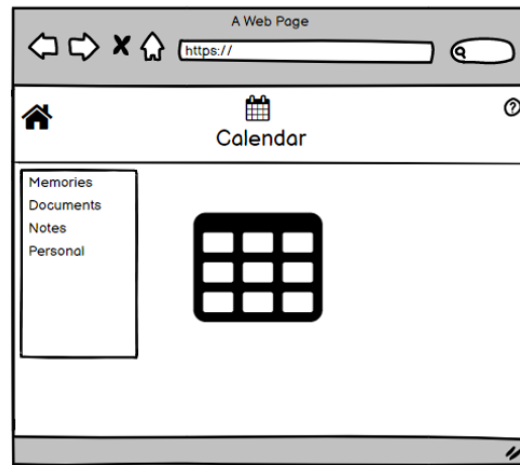
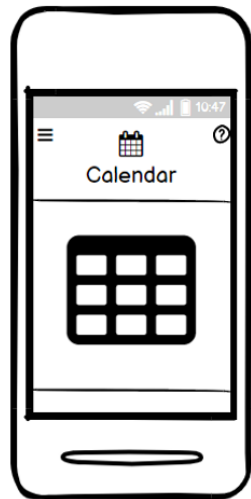
The journal screen is kept to just entry titles to avoid cluttering of the screen. To read each entry the user can simply select them and be redirected to a separate viewing page. The layout keeps all onscreen actions visible and within reach at all times.



Document Wallet: This screen allows users to add and view documents stored in the database.

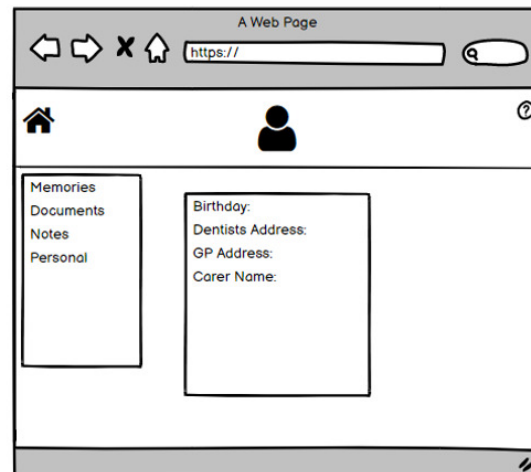
List format is used to suggest organisation and easy navigation through the stored documents. Also, with many potential users keeping their current documents in a physical format, I felt that creating the appearance of folders and grouping documents similar to that of a filing cabinet.

Consistent use of icons through download and bin icons keeps the platform theme consistent whilst making the features accessible for all (specifically making it clearer for users with dyslexia).



Calendar: The calendar is for users to keep on top of their events.

Again, to try and stay related to the real-world version, the design for the calendar is that of the physical paper version we all know. Users should be able to select any day of their choice and input any event.



Personal Profile: This is the personal profile page that later became Useful Links.

Users can access stored information written on the screen easily. Basic functionality led to basic design to keep use of use high. Being the only element on the screen, I centralised this to make it the sole target of the user's attention.

5.3.2 Colour Scheme

Having completed the UI designs, I did not include any colour scheme as I wanted to be sure the layout was working before I added any further aesthetic. In the end I decided on blue to be my primary colour, with a darker navy blue as the secondary colour; the use of these blues works well as they maintain an inviting colour palette whilst offering enough colour contrast to clearly identify the areas of change on screen. In order to avoid feeling too corporate and uniformly designed, I later decided to add orange as the feature colour. The use of orange was strategic, to draw the user's attention to the vastly different colour and improve human interface interaction.

The colour blue is widely perceived as being the colour of freedom and imagination, with representation in both the sky and sea (Bourn, 2011). By using two very different shades of blue, I aimed to create a platform in which users also felt free and capable of doing whatever they wished, on top of this blue is known to be a colour of trust, confidence, and stability which I hoped would install some increased faith in the platform and its ability to safely store users' personal affects.

I used the colour orange as the main contrast with the blue, adding a bright colour to cut through as the main features of the project. Orange colours promote creativity and emotional energy, which I aim to give the users to guide them through using each feature. I hope that by using a warmer colour such as orange, that the users will also feel more welcome and at ease when navigating the platform.



Figure 17 - Orange #ff5b00



Figure 16 - Primary Blue #0275d8



Figure 15 - Navy Blue #051938

5.3.3 Wireframe Strengths and Weaknesses

The strengths of "My Important Things" lies in its consistency of the design and the use of iconography throughout to develop a base understanding of each feature early on. By offering a consistent layout across all pages and devices, I hope to keep the user in control at all stages, they should always know how to navigate back from a mistake, find a feature on a different device or know where their photos are located.

Iconography and symbolism throughout the designs allows for the navigation of the application without the need to read any text. This is to increase web-accessibility and improve user comprehension and usability of the platform. The idea is to provide the user with a set of icons they have most likely already been exposed to in the real-world and will already understand their

meaning. By using well-known icons such as clouds for cloud storage, bins for deletion and plus for adding new elements, I can reduce the required learning curve for new users and lower the barrier to entry.

Jakob's Law of user experience (Jakob's Law, 1994) explains that most users will spend more time on other websites than your own, and therefore by pulling from features and designs they might have already spent time with can improve the usability of your own website. It is to this effect that I designed my Memories board to be like that of Pinterest or OneDrive, whilst creating a layout similar to that of Outlook for the other features.

I feel the weaknesses of my platform begins with the lack of error prevention on delete functions, without the implementation of confirmation screens, a user can quite easily accidentally delete items from their account with a single button click. To prevent this, I should implement confirmation pop-ups to confirm if the action was deliberate. In its current state this is a minor usability problem on Nielsons impact of usability scale.

Similarly, to the deletion, creating a new element is also unclear as once the user form is submitted to create a new item the user is instantly redirected back to the original screen without any on screen confirmation of their actions, instead relying on spotting the new item within their account. Again, this is not a major usability problem as it does not break the functionality, therefore will be given low priority to correct, however fixing this would improve user's quality of life.

For the duration of the designing, I tried to regularly update Samantha (at NYAS) and her group of test users. This allowed me to gain valuable feedback on the designs I was developing, this provided me with a space to bounce ideas of real users and gauge whether aspects of my design needed changing or completely removing. For instance, the introduction of image filtering by date was suggested within one of these discussions as well as the change from Diary to Journal in the app phrasing. This was altered as the group felt the term diary suggested the need for regular updates, where as a journal provides more flexibility and simply acts as a space for note taking. The users were particularly keen on the lack of on screen clutter, which made the available features more obvious when navigating.

6. Implementation

The next section of this report will cover the implementation, detailing how the platform was built and the techniques used. I will show how I have implemented the technologies I previously chose, including extracts of my code and database diagrams to demonstrate how I utilized the technology to meet my requirements.

6.1 Database Design

Using the Django framework, you are provided with SQLite by default and as a result you also get a number of pre-configured models that you can use to develop your database design, most notably this includes the User model which worked perfectly for my own project as I did not require any unique variables which would require me to create a custom User model. SQLite offered a very efficient and flexible design given the models I needed to store and on the small scale used for my prototype, the database handles all requests very responsively.

6.1.1 User Profiles

Using the Django User profile, I was easily able to add users to the pre-existing model, however without the need for some variables which were left blank for this project.

All users are generated by an admin to avoid members of the public registering for the service without authority. From here the User is given an ID to uniquely identify them, as well as the username and password which Django hashes before being stored in the database, the use of the hashing configuration is another great reason to use Django as it simplifies the security measures to keep each account safe. The user model also continues to store user permissions as well as general account information such as the initial date joined to the platform as well as the last login.

To avoid security risks and to simplify user testing, I have removed the need for names and email addresses from the accounts, however ideally in the future these could be added to increase functionality. Potential uses for the email field, could include sharing documents via email or receiving alerts to your email inbox regarding upcoming events on the calendar.

```
SELECT * FROM 'auth_user' LIMIT 0,30
```

| id | password | last_login | is_superuser | username | last_name | email | is_staff | is_active | date_joined |
|----|------------------------------|----------------------------|--------------|------------|-----------|-------|----------|-----------|----------------------------|
| 1 | pbkdf2_sha256\$216000\$L9... | 2021-05-10 17:16:55.420332 | 1 | alfie | | | 1 | 1 | 2021-04-25 13:26:59.882287 |
| 2 | pbkdf2_sha256\$216000\$JK... | 2021-04-25 13:38:12.082995 | 1 | alfie2 | | | 1 | 1 | 2021-04-25 13:27:10.643701 |
| 3 | pbkdf2_sha256\$216000\$8n... | 2021-05-09 13:29:07.125747 | 1 | Supervisor | | | 1 | 1 | 2021-05-09 13:28:13.343890 |
| 4 | pbkdf2_sha256\$216000\$L... | null | 0 | TestUser1 | | | 0 | 1 | 2021-05-11 11:25:34 |
| 5 | pbkdf2_sha256\$216000\$ml... | null | 0 | TestUser2 | | | 0 | 1 | 2021-05-11 11:25:59 |

Figure 18 - Database User Model

In order to allow for users and admins to have different privileges within the system, I also implemented user groups to better control the actions available to the different user types. I wanted to make sure that admins had the capacity to do anything and everything on the platform, whilst users could only access their own accounts and not the administration panel offered within the Django framework. The user group table is very simple and just highlights the ID of the group as well as the group name, which is later linked to each user account.

```
SELECT * FROM 'auth_group' LIMIT 0,30
```

| id | name |
|----|-------------|
| 1 | Admin |
| 2 | userProfile |

Figure 19 - Database User Group Model

6.1.2 Document Storage

Where I used the pre-defined model for users, I was not able to do the same for all the contents of the accounts. In this section I will be using the Memories feature as an example, as the database design is the same for the rest of the stored content.

Within my Memories app I built out the models that would become the database table for the relevant content, I chose to add categories to allow the users to group and organise all their images, as well as the following:

Collection: Memories_photo

Fields:

Id: int
Image: image
ImageDate: date
descriptionImage: string
category_id: int
owner_id: int

The IDs for each image is a integer and allows for the unique identification of that particular image, this is then followed by the relevant image as an attached file. Next comes the imageDate in a date format, this is automatically generated by the system at the time the user uploads the image to the database, this is in order to then allow the user to sort by date in terms of ascending or descending time order.

Image description is captured as a string and simply holds any description the user wished to add to the image, this could be the location name of the image, or just some general thoughts regarding the image. Finally, there are category ID and owner ID, both of which are used to identify the category to which the image variable belongs to, as well as the owner of the image to whom the image can be seen by. Users on another account are not able to access the content for a different user thanks to the use of the ownership variables which I have implemented across the entire platform.

| SELECT * FROM 'memories_photo' LIMIT 0,30 | | | | | Execute |
|---|--|----------------------------|------------------|-------------|----------|
| id | image | imageDate | descriptionImage | category_id | owner_id |
| 1 | cat1_FUOkOk0.jpg | 2021-04-25 13:32:32.904997 | Cats | 15 | 1 |
| 4 | husqy_iFq099E.png | 2021-04-25 13:34:12.848380 | Bikes | 14 | 1 |
| 5 | kawi_ushNKvk.png | 2021-04-25 13:34:13.595869 | Bikes | 14 | 1 |
| 6 | running_vGH1jKL.jpg | 2021-04-25 13:36:54.064985 | Run | 16 | 1 |
| 9 | 126817496_1050857598675328_4781348814... | 2021-04-25 13:53:21.567732 | Test | 16 | 1 |

Figure 20 - Database Memories Model

Using this database design enabled me to understand each variable and the ownership of all the stored images quickly and easily. With each image ID being related to a category_ID and owner_ID which can be seen above in the User Model table. Category ID extended to a separate table in which I stored all the user categories along with their relevant owners that would allow for the grouping of images within accounts (this table can be seen below).


```
SELECT * FROM 'memories_category' LIMIT 0,30
```

| id | name | ownerCategory_id |
|----|---------------|------------------|
| 14 | Motorbikes | 1 |
| 15 | Cats | 1 |
| 16 | Miscellaneous | 4 |

Figure 21 - Database Memory Category Model

The process of adding a new category from the front-end is very simple and joins together seamlessly with the database to create a very efficient and smooth program. By using the upload feature within the Memories function and entering in all the details requested the user can quickly update their database to include the new image.

Description

A picture of my old laptop

Select a Category

Please select a category

Create a new category

Laptop

Choose Images

Choose files 162195452_...30596_n.jpg

Upload Images +

Figure 22 - Memories Upload Screen

Which instantly displays within the database as a new row.

| | | | | | |
|----|---------------------------------------|----------------------------|----------------------------|----|---|
| 10 | 162195452_274017480801444_76110941... | 2021-05-11 12:10:04.525654 | A picture of my old laptop | 17 | 1 |
|----|---------------------------------------|----------------------------|----------------------------|----|---|

Figure 23 - New image added to database model

The following diagram displays how the database responds to a user request, in this case the user send a POST request to the system to login to the account, this is then verified using the Django authentication configuration. Once in to the platform, if the user goes to access the Memories function they send the request to view the html render for this page; in order to actually see any contents of this page the method first checks who the request user is, before matching the request user id from the user model linked with the account, to that of the owner_id of the elements to be displayed.

Once it has been clarified which images belong to the requesting user from the database, the database begins to return the files as a list of on-screen elements. This includes the image Id (which appears in the page URL when viewing an image), the image file itself, category Id and description of the image. The same process is done when also using the Document Wallet, Journal and Calendar features, checking the request user before returning the requested information from the relevant tables within the SQLite database.

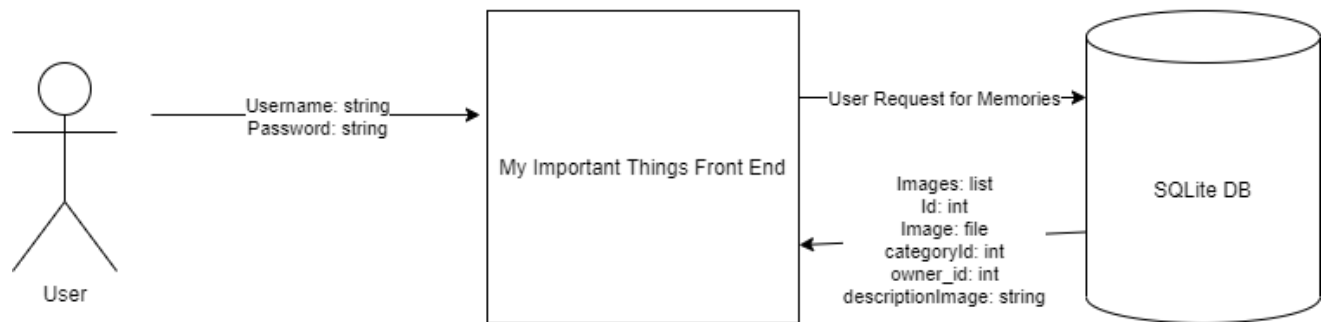


Figure 24 - User database interaction model

6.2 Web Development Best Practices

Given that the project had a very limited timeline from initial research all the way through to final prototype build, I aimed to follow the industry standard best practices for web development. Not only did I think this would make my workflow more time efficient, but knowing that I was developing this for Children's Services and Cardiff Council, I need to be sure that the code I was producing was maintainable and scalable for future developments.

Following the best practices laid out by Thinkful (Thinkful, 2021) I started the implementation process with vigorous planning. I knew which use cases I need to meet to satisfy the requirements set for myself, from here I was able to plan the routes down which the Django apps would work. As I wanted each function to be a separate independent module, I knew this meant they would all need to be contained within their own app, apart from any cases where the functionality was basic, in which case I felt it was best practice to keep these pages limited to paths within the Homepage app.

Connecting these apps together also would require a navigation bar, which I planned to have positioned in the same location on every page to maintain consistency and coherency between pages.

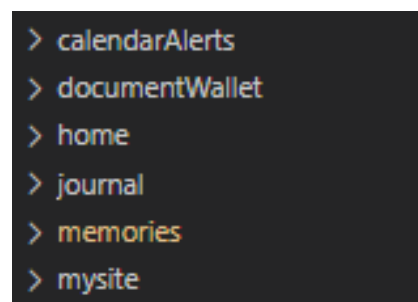


Figure 25 - Base project folder and extending apps.

Through the use of my agile development plan, the next element of industry best practice I tried to keep to was the idea of constantly making small improvements. In an attempt to avoid making large-scale errors and project breaking problems, I avoided deploying major changes and instead chose to implement smaller elements at a time to ensure the code was functional. The use of Git for version control was of great use here, as well as backing up local versions in zipped folders. By using version control to my advantage, on the few occasions where I deployed larger portions of codebase that

caused problems, I was easily able to revert my changes and come back to a working version of the project. Mentally I also benefited by using agile development and my Gantt chart, by cutting the overall workload down into manageable chunks which I could go through as a checklist. Not only did this improve my moral when working independently on such a large task, but it also kept me up to date and avoided mistakes or forgetting things when simply looking at the big picture.

Coming back to the prospect of handing this project over to future developers, I also needed to be sure that all the code made sense and served its purpose efficiently and without over dramatization. Not knowing who will pick this project up going forward, I wanted to develop the code in a way that was reusable and therefore scalable where necessary. With many of my features sharing similar functionality, it was easy to move my functions around and reuse a great deal of my code, not only saving me time in the development process, but making the entire project more unified and relatable. If a developer could understand one feature, they would most likely understand the rest. This is what is referred to as thinking smarter and not harder within the industry, reusing code is not always lazy and benefited myself during this project, and will benefit anyone who works on it in the future.

6.3 Django Implementation

In this section I will details exactly how I implemented the Django framework to create the platform as it is now. As there is a great deal of repetition within the views.py and functionality, for the examples in this section I will be referring to the Homepage and Document Wallet features from the project.

When a new Django project is created, the folder structure and required files are all created for you. Most importantly this includes the paths to each app directory within the project, user validation, database settings and a list of all the installed apps.

It is important to register any new apps within the project settings to allow the directory to be activated within the system, this is accomplished by taking the AppConfig name from the apps.py file and inserting it within the settings.py list (this can be seen here). This list not only includes any app made by myself, but also any Django contrib packages which will come in handy when creating accounts and administrator permissions.

```
# Application definition
#List of all installed/built apps being used in the project
INSTALLED_APPS = [
    'home.apps.HomepageConfig',
    'memories.apps.MemoriesConfig',
    'calendarAlerts.apps.CalendaralertsConfig',
    'journal.apps.JournalConfig',
    'documentWallet.apps.DocumentwalletConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Figure 26 - Project Installed Apps

The password validation package provided with Django is a very powerful tool and meant I wasn't required to build out my own validation for user profiles. This is yet another reason why I chose to

use this framework, as I knew the packages available would both save me time, but also be recognised as a industry standard.

```
# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

Figure 27 - Django Password package

The most important change that I made to the settings.py file is that of the Static file directories. With the prototype storing image and documents within this folder, providing root access for the platform was necessary to ensure documents were properly located and rendered.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
MEDIA_URL = 'images/'
#Set X_FRAME option to except files with source of same origin as project
X_FRAME_OPTIONS = 'SAMEORIGIN'

#Sets directory for all content to be stored on local device (creates a folder within django project to store in SQLite DB)
STATICFILES_DIRS = [
    BASE_DIR / 'static'
]

MEDIA_ROOT = BASE_DIR / 'static/images'

#Login redirect route
LOGIN_REDIRECT_URL = '/homepage'
```

Figure 28 - Settings.py Static File paths

Similarly to the urls.py that can be found within each app, the project as a whole contains a main urls.py file that is used as a switch board, directing the user to each individual folder as requested. In this list of paths is included the URL path which then includes any paths extending on within the individual apps. As a result if I were to access “homepage/” path, I’d be redirected to the urls.py within the home folder to be further rerouted depending on my actions.

```
#Configuration of all paths to all installed apps on the project
urlpatterns = [
    path(r'admin/', admin.site.urls),
    path(r'homepage/', include('home.urls')),
    path(r'memories/', include('memories.urls')),
    path(r'calendar/', include('calendarAlerts.urls')),
    path(r'documentWallet/', include('documentWallet.urls')),
    path(r'journal/', include('journal.urls')),
    path(r'accounts/', include('django.contrib.auth.urls')),
]
```

6.3.1 Static Page Implementation

Within the application there are a number of static pages, these pages tend to be basic text-based pages without any dynamic content such as the Homepage and required considerably less work to produce. In this case, these pages were all built within the Home app folder. As you can see from the file structure here, it contained the same files as for dynamic content, however this was largely unused.

For the implementation of the login/logout pages, the Django preconfigured methods were used and therefore will not be shown here.

With the templates folder storing all related html files and then all the .py files containing the backend code that rendered and structured the app. I will go into more detail about these files within the dynamic content examples.

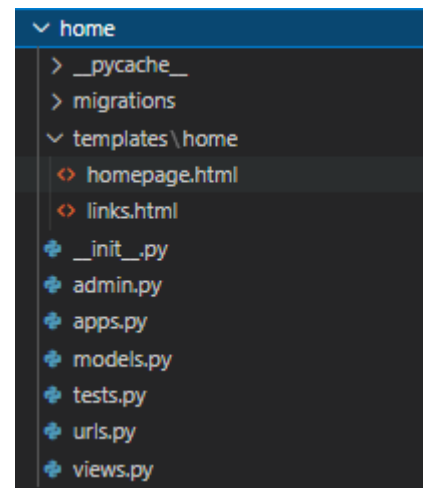


Figure 29 - Home app folder

6.3.2 Dynamic Page Implementation

For the following examples I will be referring to the Document Wallet app and its relevant files (the functionality of which is shared across all apps). On the right you can see the identical file structure used when working with dynamic apps compared to the static apps mentioned above. Within this I will not be covering the _init_.py file or the tests.py files as neither were altered during this project. _init_.py is the file which identifies the folder as a module within the python directory and allows the user to execute any code at the beginning of the module as it is the first file accessed. tests.py is a file to allow for quick and easy unit testing offered by Django, however for this project was unnecessary.

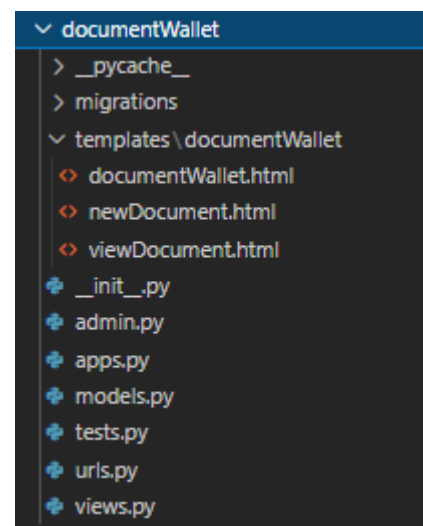


Figure 30 - Document Wallet app folder

To begin with in order to register the app as a working module within the project directory and ensure the app is accessible from the platform once hosted, the apps.py lets you name and register the app for appending within the project settings file. Now that the app is visible to the whole project it is key to properly structure the module using urls.py to create various paths that the user can go down when using different features. The purpose of the paths is to update the page URL as well as to call the relevant views from views.py which will ultimately render the new page, in the example below (right), the paths include routes to add new documents, view documents or sort documents.

```

from django.apps import AppConfig

#Configure the app/feature name for connection to the main project

class DocumentWalletConfig(AppConfig):
    name = 'documentWallet'

from django.urls import path
from . import views

#Set URL paths within the Document Wallet app

urlpatterns = [
    path('', views.documentWallet, name='documentWallet'),
    path('add/', views.newDocument, name='newDocument'),
    path('document/<str:pk>/', views.viewDocument, name='document'),
    path('deleteDocument/<str:pk>/', views.deleteDocument, name='deleteDocument'),
    path('AscWallet/', views.AscWallet, name='AscWallet'),
    path('DescWallet/', views.DescWallet, name='DescWallet'),
]

```

The next stage to making the app was adding the models that would be needed to populate the SQLite database and allow users to store documents. In order to achieve this, I first needed to create custom models for categories as well as the documents. The models contained unique identifiers as well as variables such as date, title and owner within the classes that would become columns in the database tables later on. It is important to create unique and clear variable names at this stage to avoid confusion later on, by doing so improves the developer user experience with strong naming conventions which save time in the long run.

Note that I imported the User model from the Django auth.models which was a pre-existing user model which I am using for my profiles. Therefore, I was not required to create a bespoke user model.

```

from django.db import models
from django.contrib.auth.models import User

# Create your models here.

class Category(models.Model):
    name = models.CharField(max_length=100, null=False, blank=False)
    ownerCategory = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True, related_name='ownerDocWallet')
    def __str__(self):
        return self.name

class Document(models.Model):
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True, blank=True)
    file = models.FileField(null=False, blank=False)
    fileDate = models.DateTimeField(auto_now_add=True)
    title = models.TextField()
    owner = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)

    def __str__(self):
        return self.title

```

Figure 31 - Document Wallet models.py

To then allow these new models to be used and recognised by the database, I needed to register them within admin.py which also moved the newly made models to the administrator pages.

```

from django.contrib import admin
from . models import Category, Document
# Register your models here for admin access

admin.site.register(Category)
admin.site.register(Document)

```

Figure 32 - Admin model registration

Finally in order to implement all of the above code and provide the dynamic features to the pages created, I had to create the views within views.py that would enable the options to create, delete sort and other features of the platform.

6.3.2.1 Page render

The initial view is used to render the Document Wallet page for users, this view is activated from the Homepage when a user clicks through to the feature using the CTA (call to action). The following view is used to initially collect the categories using a GET request to the database, before also collecting all Document objects filtered by the current account. It then returns the found categories and documents for this particular user and appends them into a dictionary where they can be referenced from the template. Finally the view renders the new template HTML file along with the context dictionary containing all the account variables needed.

```
#View renders the document wallet, displaying user specific elements and loading content from models
@login_required
def documentWallet(request):
    category = request.GET.get('category')
    if category == None:
        files = Document.objects.filter(owner=request.user)
    else:
        files = Document.objects.filter(category__name=category).filter(owner=request.user)

    categories = Category.objects.filter(ownerCategory=request.user)

    context = {'categories': categories, 'files': files}
    return render(request, 'documentWallet/documentWallet.html', context)
```

Figure 33 - Page Render View function



Figure 34 - Document Wallet as seen by users

6.3.2.2 View Document

Once on the Document Wallet page it is possible to then choose to view any of the documents being stored, this is obviously key to making the feature useful and also provides the options for the user to later delete, download or print the document they view.

This view works by collecting the file Id passed with the request to view the document, the system finds the document object with the matching Id private key, before running some user permissions. The if statement double checks whether the owner of the document is the same person as the user who requested to view it, if so then the document view is displayed, however if not then the user is faced with a 403 Permission Denied error. In the case that a malicious user has the private key Id for a file that is not their own, then this user authentication catches the permissions and blocks them.

All user accounts needed to be secure and not viewable by the public, therefore adding in this increased level of security is crucial to manage account permissions and keep content private.

```
#Function to display file view page, renders single chosen file
@login_required
def viewDocument(request, pk):
    file = Document.objects.get(id=pk)
    if file.owner == request.user:
        return render(request, 'documentWallet/viewDocument.html', {'file': file})
    else:
        raise PermissionDenied()
```

Figure 35 - Viewing Page View function

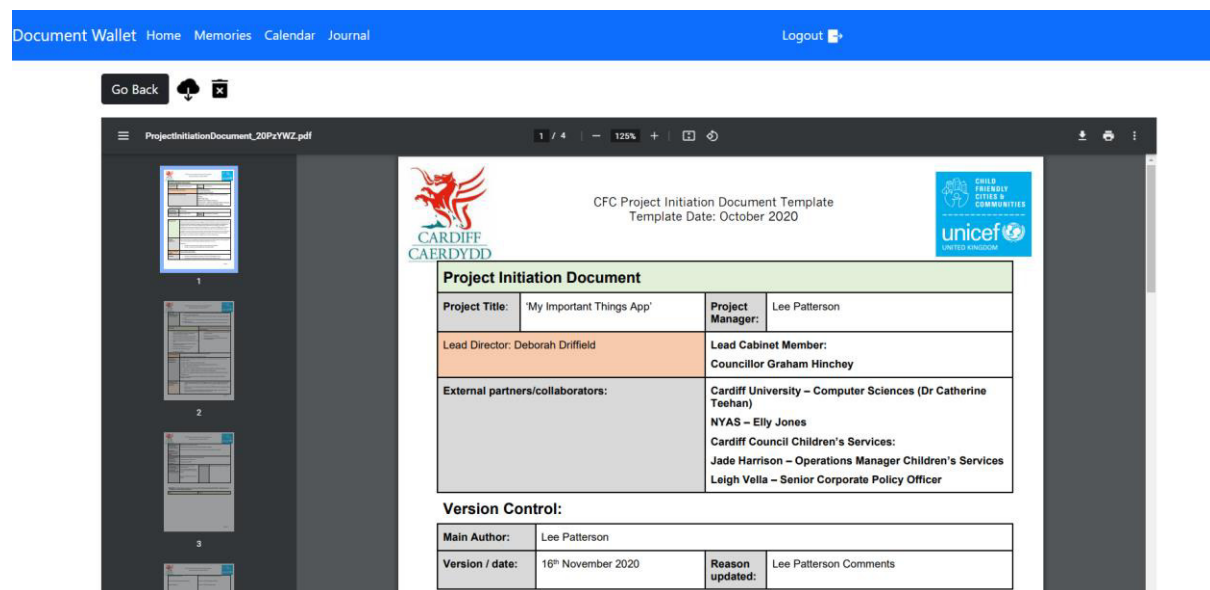


Figure 36 - Document View as seen by users

In order to be able to render the pdf files, it was also required that I add a section of code to manage error handling. Unfortunately due to the default settings, Django doesn't support the access of files hosted at the same origin as the page, therefore the addition of the code seen here as well as changes to the project settings was required to allow these files to be rendered.

```
#HttpResponse to handle error of pdf files source being from the same origin as page
@xframe_options_sameorigin
def view_two(request):
    return HttpResponse("Display in a frame if it's from the same origin as me.")
```

Figure 37 - Xframe Origin fix

6.3.2.3 Document Ordering

As is with the basic page render view function, the ascending and descending functions render the page in the date order selected by the user. The only difference here is that the files when collected from the database are ordered by the fileDate variable, before being appended to the dictionary and rendered.


```

#Filter stored files by ascending order
@login_required
def AscWallet(request):
    category = request.GET.get('category')
    if category == None:
        files = Document.objects.order_by('fileDate').filter(owner=request.user)
    else:
        files = Document.objects.filter(category__name=category).order_by('fileDate').filter(owner=request.user)

    categories = Category.objects.all()
    context = {'categories':categories, 'files':files}

    return render(request, 'documentWallet/documentWallet.html', context)

#Filter stored files by descending order
@login_required
def DescWallet(request):
    category = request.GET.get('category')
    if category == None:
        files = Document.objects.order_by('-fileDate').filter(owner=request.user)
    else:
        files = Document.objects.filter(category__name=category).order_by('-fileDate').filter(owner=request.user)

    categories = Category.objects.all()
    context = {'categories':categories, 'files':files}

    return render(request, 'documentWallet/documentWallet.html', context)

```

Figure 38 - Document ordering View function

6.3.2.4 Add New Documents

Within this function, the user is creating a new document within the database. The initial request is received as a POST method from the form within the HTML template. The system then collects all of the attached files that came with the form, checking if a category was selected or a new category created, before adding the new file with all the related information. When this process is finished, the function renders the Document Wallet page again, with the addition of the newly uploaded files.

```

#function receives POST request to create a new file within wallet
@login_required
def newDocument(request):
    categories = Category.objects.all()

    if request.method == 'POST':
        data = request.POST
        files = request.FILES.getlist('files')

        if data['category'] != 'none':
            category = Category.objects.get(id=data['category'])
        elif data['categoryNew'] != '':
            category, created = Category.objects.get_or_create(
                name=data['categoryNew'],
                ownerCategory=request.user)
        else:
            category = None

        for file in files:
            file = Document.objects.create(
                category=category,
                title=data['title'],
                file=file,
                owner=request.user,
            )

        return redirect('documentWallet')

    context = {'categories':categories}
    return render(request, 'documentWallet/newDocument.html', context)

```

Figure 39 - New Document View function

Figure 40 - Upload Document View as seen by users

6.3.2.5 Delete Documents

Lastly, is the function used for deleting files that already exist within the database. This function works by receiving the deletion request along with the specific document Id (private key), which is then again double checked to be received from the user with the correct account permissions before finally deleting the file and returning the user to the Document Wallet screen. The user authentication was again added here to ensure that URL splicing was not possible for a method of deleting other users files.

```
#Private key passed for element to be deleted from database
@login_required
def deleteDocument(request, pk):
    file = Document.objects.get(id=pk)
    if file.owner == request.user:
        file.delete()
        return redirect('documentWallet')
        return render(request, 'documentWallet/viewDocument.html')
    else:
        raise PermissionDenied()
```

Figure 41 - File Deletion View function

6.3.2.6 HTML Templates

From the perspective of the front-end the view functions are all accessed via user actions using CTAs or other interactions. The functions are all processed before a rendering of the changes is displayed within the template for the user to view.

Across the board the navigation bar is available on almost every page, this is rendered using HTML along with some basic inline CSS and use of the Bootstrap libraries for functionality and layout. The general layout for this navigation bar was taken from the Bootstrap documentation (Bootstrap, 2021) which permitted me to easily provide user flexibility as well as mobile optimisation through the conversion into a burger menu on smaller devices.

For all of my templates I also took the time to correctly format them to ensure readability and functionality was clear to developers. Comments were also added to notify the reader of particular aspects that needed attention such as functions.

```

<!--display navigation bar with hamburger menu for mobile-->
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <a class="navbar-brand m-1">Document Wallet</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarText" aria-controls="navbarText" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarText">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link navText" href="http://127.0.0.1:8000/homepage">Home</a>
      </li>
      <li class="nav-item">
        <a class="nav-link navText" href="{% url 'memories' %}">Memories</a>
      </li>
      <li class="nav-item">
        <a class="nav-link navText" href="{% url 'calendar' %}">Calendar</a>
      </li>
      <li class="nav-item">
        <a class="nav-link navText" href="{% url 'journal' %}">Journal</a>
      </li>
    </ul>
  </div>
  <div class="collapse navbar-collapse" id="navbarText">
    <ul class="navbar-nav mr-auto">
      <a href="{% url 'logout' %}" style="text-decoration: none;" >
        <li class="nav-item" style="color: white;" >Logout </li>
      </a>
    </ul>
  </div>
</nav>

```

Figure 42 - Navigation HTML Code



Figure 43 - Navigation as seen by the user

When rendering the individual elements of a user account, it was important for me to keep this code as clean and optimised as possible, this resulted in the use of many “if in list” loops which kept the necessary code to a minimum whilst providing the same functionality. This can be seen clearly when loading the categories and documents on the Document Wallet screen, by doing this the loading of database elements is also kept dynamic and will not require user actions to reload any updates.

In the case that the user hasn’t uploaded any documents yet, I also displayed a short message to the user to keep them in the loop, following Nielsen’s principle of system status visibility.

```

<div class="row">
  <div class="col-md-3" style="padding-right: 0px;">
    <div>
      <!--Categories created by user to allow for further filtering-->
      <div class="card-header">
        Categories
      </div>
      <ul class="list-group list-group-flush">
        <li class="list-group-item categoryStyle">
          <a href="{% url 'documentWallet' %}" class="categoryStyle">All</a>
        </li>
        {% for category in categories %}
        <li class="list-group-item categoryStyle">
          <a href="{% url 'documentWallet' %}?category={{category.name}}" class="categoryStyle">
            {{category.name}}
          </a>
        </li>
        {% endfor %}
        <a href="{% url 'newDocument' %}" class="btn btn-dark btn-sm m-1">Upload new documents </a>
      </ul>
    </div>
  </div>
  <div class="col-md-9">
    <div class="row">
      <!--Display all files within the accounts files DB column-->
      {% for file in files %}
      <div class="col-md-9">
        <div class="my-2">
          <a href="{% url 'document' file.id %}">
            <p class="btn btn-block btn-primary" style="width: 100%;">{{file.title}}</p>
          </a>
        </div>
      </div>
      {% empty %}
      <h3>Nothing has been saved yet!</h3>
      {% endfor %}
    </div>
  </div>
</div>

```

Figure 44 - Dynamic Loops

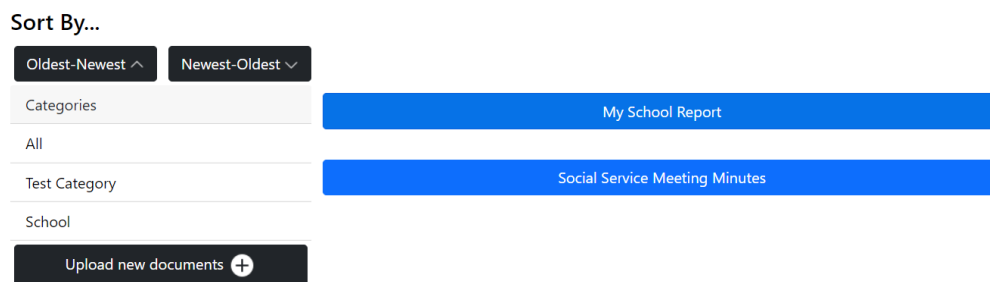


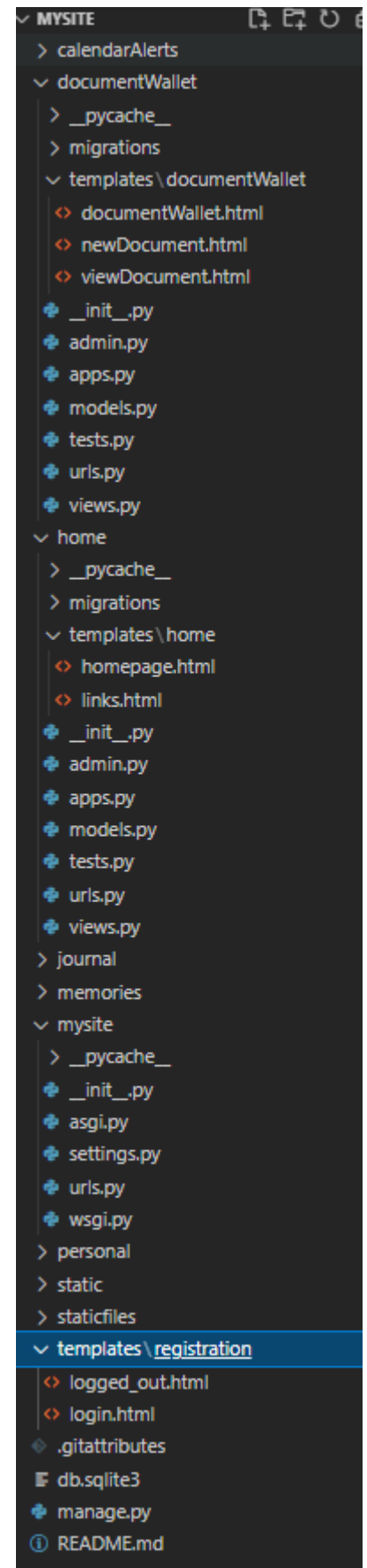
Figure 45 - Document Wallet as seen by the user

I have chosen to only show examples from the Document Wallet, this is due to the same functions and features being widely used across all other apps within the project and I wanted to avoid repetition. The examples shown above are implemented throughout my platform using different database models and information to offer the user with all the features I laid out within my requirements in the initial project plan.

6.3.2.7 Project Structure

Having now finished the entire implementation of all apps and functions within the prototype version I can view the project structure as a whole. Django does a great job of providing a comprehensive and usable structure that improves workflow, but also comes with a great deal of documentation and support.

In the example of creating the Login and Logout pages, by referring to the documentation (Django, 2021) I was able to create the route recognised by the framework as well as template names that would be instantly incorporated using the packages available to me. Naming conventions for files and folders was highly relevant for this project as the entire framework is based on using understood pathways, in doing so however the project was very similar to that of other Django projects and followed the best practice principles for this level of development.



6.4 Platform Hosting

Once the platform was completed, and in order to facilitate user testing remotely, I was required to get the system hosted. In the end I used a popular free web-hosting service called PythonAnywhere (PythonAnywhere, 2021), which offered Django support and a free tier with enough storage to allow users to test general functionality.

Following the detailed guides provided by PythonAnywhere I was able to quickly and easily deploy “My Important Things” to their platform using a virtual environment and the provided bash consoles. With the provided 512mb of storage and 3 months of free hosting, this was a very simple and affordable method for me to get the platform operational outside of my local server. Not to mention the fact that having the project hosted did help in providing closure to the project as this was the last step required before testing could happen.

PythonAnywhere is a very popular and powerful tool that offered insight into user activity on the platform as well as being able to let me change and track everything from the dashboard (see below).

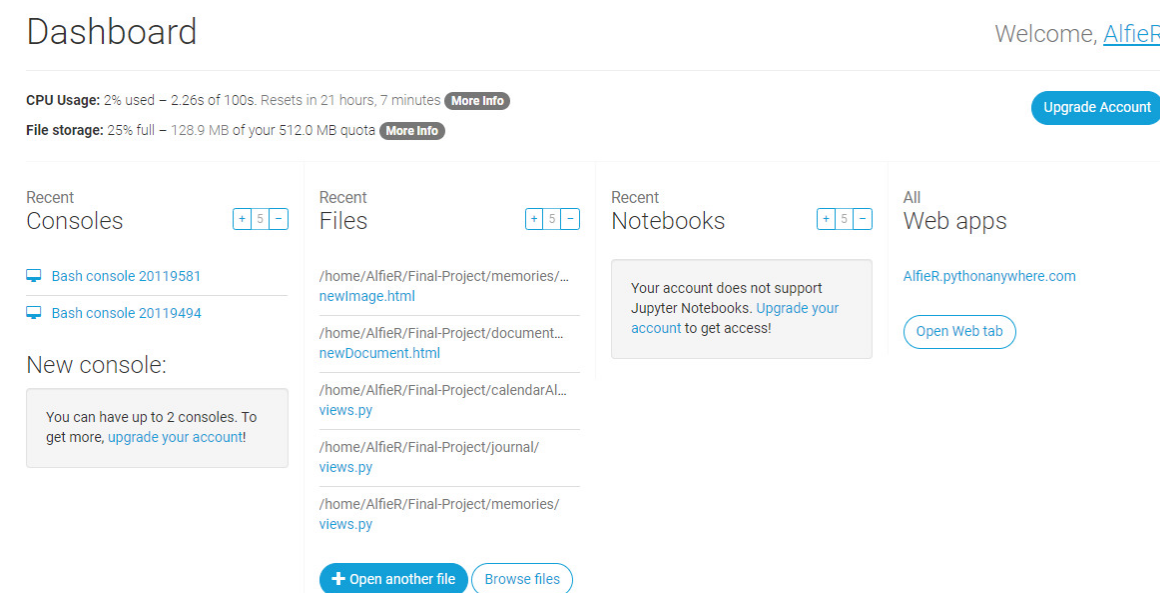


Figure 46 - PythonAnywhere Dashboard

7. Results

This portion of the report will detail the test cases implemented to check that requirements were met, in particular the Must Have functional requirements described in part 4.1 of the report.

7.1 Test Cases

In order to objectively see whether I was able to meet the requirements for this project, I have set up a number of test cases which I will run through to validate the success of the platform. I have done tests for every functional and non-functional requirement with a pass or fail as well as any changes that were made as a result.

Of the total 18 test cases, 11 passed giving me a 61% success rate. To view the complete test case results please refer to the Appendix Test Cases, detailing the test procedure, prerequisites, and further comments where applicable.

7.1.1 Functional Requirements

- 1-Requirement: The platform must allow users to upload PDF documents and images. **Pass**
- 2-Requirement: The application must allow users to submit journal entries. **Pass**
- 3-Requirement: The application must allow for organisation of all stored files. **Pass**
- 4-Requirement: Users must have a private space through personal accounts with private content. **Pass**
- 5-Requirement: The platform must allow users to delete any stored content. **Pass**
- 6-Requirement: All uploaded documents must downloadable. **Pass**
- 7-Requirement: Users will have the option to share the content of their account with others. **Fail**
- 8-Requirement: Account usernames and passwords can be reset for existing accounts. **Fail**
- 9-Requirement: All stored files within the “Document Wallet” can be printed from the platform. **Pass**
- 10-Requirement: Users are able to delete their accounts from the system. **Fail**
- 11-Requirement: Users can edit their profiles to contain personal information and a profile picture. **Fail**
- 12-Requirement: Calendar app can send alerts of upcoming events. **Fail**

7.1.2 Non-Functional Requirements

- 1-Requirement: All features of the application must be functional and accessible from both mobile. **Pass**
- 2-Requirement: The entire platform will be designed in a web-accessible format. **Pass**
- 3-Requirement: The entire platform must be easily navigable. **Pass**
- 4-Requirement: Platform should provide support for users through other resources. **Pass**
- 5-Requirement: User accounts can be personalised/customised. **Fail**
- 6-Requirement: The project should be implemented in a maintainable fashion. **Pass**

All testing was carried out on both a Honor 10 Lite smartphone on chrome as well as a desktop tower computer running windows 10 and Chrome browser. User testing was also carried out on a range of devices using differing browsers.

Regarding the failures, the problem with these features was largely due to time constraints that resulted in the functionality not being implemented or not fully being finished. The only initial failure that I was able to resolve was the security and private space for Test case number 4 of the functional requirements. Where user content was able to be accessed via the URL from a different account, this was quickly patched to check user authentication of the request user.

Test case 1 of the functional requirements was to check the correct functionality of uploading documents and pdf files to the system, although this passed within the test case as it did allow the user to achieve the goal, I did notice there was no validation for file type which therefore allowed users to upload images to the Document Wallet and documents to the Memories board. Again, this

problem has been quickly resolved by adding accepted formats to the forms for uploading within these apps.

8. User Feedback

Upon completing the project and carrying out my own functional testing, I also hosted the project live using PythonAnywhere. By doing so I was able to collect user feedback through hands-on testing. The completed feedback forms can be found attached within the appendix, however the consensus regarding the overall quality of the finished prototype was positive.

Whilst the question of functionality could be tested by myself, my ability to fairly judge the design was limited. Luckily, through questioning the test group who had access to the platform I was able to find that they had no issues navigating the platform. Test user 1 stating that “Having just a limited number of tabs to not overwhelm someone” was a strong point, also saying that the platform was “more than easy enough to navigate”. Given that usability and navigation were key aspects that I wanted to achieve with this project, I am very happy to receive this feedback from real users and acts as evidence that my planning and design phases were worth the time. Test user 2 further backed up the statements made by user 1, saying that “I found the app very easy to use and it was straight forward”.

From the four user responses I received, all four of them found the application easy to navigate and operate the majority of the functions, with the Document Wallet and Journal features being the most popular aspects of the platform.

As for the areas that users felt required some improvement, these largely revolved around the accessibility of the calendar feature and the lack of colours used on some pages. Whilst the colour scheme was implemented on some screens, I left the design basic for what I had felt was improved visibility; however this sentiment was not found in some of my test users. Within the feedback from User 1, they suggested an increase in colour merely due to personal preference. On the other hand User 2 found that the calendar and journal features were hard to use for dyslexic users, stating that an option for custom styling and colour would have been an improvement to this fact.

This feedback was much appreciated and has allowed me to better understand how the platform has met the requirements and where there are areas that can stand to improve. I’ve used the information given to me by the care experienced users and applied it within the section on future work.

9. Conclusion

This project began as a relatively simple idea to digitise and organise the often-scattered documentation possessed by those in the care system (in Wales specifically). With the two main features briefed as a secure document storage location and an image cloud storage solution, this was a solid foundation for me to build upon with new features and ideas that came from research with the young adults of NYAS (National Youth Advocacy Service). Speaking with the target audience, allowed me to develop a personal relationship with the people I was aiming to help and provided me with the drive to complete the project.

Designing the project around my discussions with the target group, I was able to create my two personas to whom which I aimed the platform specifically. Having now finished the project, I can confirm that I was able to accomplish most of the aims for each persona; only missing the options for platform personalisation from Sara and the storage capacity capabilities of Cai. These were either

down to the lack of time or available money for the project which were both constraints I was aware of going into this process. The rest of the persona goals have been achieved however:

- Offering a secure and private account to store photos of loved ones and documentation.
- A simple UI that promotes usability and functionality.
- A journaling tool to offer a replacement for the physical notepads.
- Organisational tools to better manage the large quantities of content held by each user.
- Providing a smooth and functional platform across all devices.
- Cloud storage to free up space on device hard drives.

During design and implementation phases of the project I was focused on making the platform as intuitive and relatable as possible, pulling from applications I knew the user group already had experience with and understood well. Following Nielsen's principles of design along with industry standard practices for web applications I was able to apply beneficial elements such as consistent navigation bars, iconography and colour schemes that improved usability and user comprehension, which in turn led to a more natural and comfortable experience for the test users.

Whilst the constraints previously spoken about did hinder the final product, they did not impact my ability to deliver upon the bulk of the Must Have requirements within the timeframe originally given to me. Regrettably, there were aspects which I wish I had been able to improve upon throughout the process, however I was always sure to keep my eye on the end goal to provide a working prototype to Children's Services to demonstrate that such a platform was possible and could in fact benefit the lives of all of those who used it.

In the end, the finished system surpassed the initial expectations of brief, offering a structured clean manner. Although additional features could be added, the finished product accomplished all that I set out to do and I am delighted that I was able to deliver the project to Children's Services in a high-quality state.

10. Future Work

The project has been finished to a high standard with which I am very pleased. Most of my requirements have been met and as a result I would call the prototype a success however I know if I were to continue working on the platform there are many changes and new additions I would make.

To start with in terms of the technology used I would aim to migrate all the data storage hosting over to an AWS system, whilst this was researched earlier on in the project timeline the financial constraint as well as the limited time available to learn the technology resulted in my avoiding this option for the prototype. If I were to take the project on however, I would definitely make the effort to move all document storage into an S3 bucket for improved security and accessibility among other features. The current database whilst functional is very limited in terms of scalability and wouldn't work for a project containing thousands of users and their content.

To further improve upon the service provided to the users, I would also plan on improving existing features such as the search filtering for content, to include a more precise date searching function as well as specific search bar for document titles for example. Using the feedback I received from my user feedback forms, I can see that the calendar feature is the biggest flaw of the current features and would most likely require a full rework. Whilst some users did find the calendar useful, I believe the chaotic state of the page leaves something to be desired in terms of the design and visual appeal.

In terms of new features, the addition of requirements that I was unable to implement for the current version would be incredible. Functionality such as the platform personalisation was unfortunately missed in prototype stages and being a request from test user 2 for the future, I would hope to include this in any future versions. Smaller inclusions would be the calendar alerts which could apply an API to carry out email alerts or mobile push messages, as well as increased account control for the user (whether that be account deletion or password and username changes). Whilst my test cases did for the majority return as passes, I feel that the inclusion of these missing features and functions would vastly improve the overall usability and user experience. Providing a more rounded and complete package is always going to be the end goal for a project such as this, as a result constant changes and updates would inevitably happen to keep users in control and provide tools that will improve their lives within the care system.

Finally for future work I would love to be able to migrate the platform over to a fully supported mobile app. Whilst being functional on mobile browsers, offering a separate app from the Apple/App store would be great for user flexibility and allow me to develop more bespoke features not limited by the frame of browsers.

11. Project Evaluation

I have thoroughly enjoyed my time working on this project and felt that it has provided me with the opportunity to learn a great deal from all aspects of the development lifecycle. The project has been invaluable at teaching me the importance of proper planning and research before commencing on any implementation and has also given me the time to focus on industry standards and principles that I may have previously ignored in past modules.

Throughout this project I have been constantly learning and progressing my knowledge of Python and in particular the use of the Django framework. Providing me a platform from which to experiment, but also learn from a very popular community of developers. Although I did have some Django and general web-development experience from the previously mentioned Web Applications module (CM1102), the process of designing and building an entire platform from scratch was fascinating with plenty of chances to learn. Looking back on the project as a whole, I have wider appreciation for those within the industry and a stronger understanding of the huge amounts of work that go into the platforms that I use on a daily basis and yet take for granted. This sense of appreciation was only further developed by my use of Django, which has been used to create some of the world's most popular digital platforms including Youtube, Spotify, Instagram, Dropbox and Pinterest, some of which are websites I aimed to improve upon or took inspiration from as a result of their success and popularity.

My ability to find resources has also been refined, through constant reference to official documentation and guidelines, YouTube tutorials and other online forums. I found that I was better equipped to resolve problems I encountered towards the end of the timeline as opposed to the beginning where I felt quite lost in the wealth of support offered online.

Regarding the success of my organisation, I would argue that I did a remarkably good job and kept the standard of organisation high throughout the timeline. Not only did I manage to stick to my development schedule as initially shown in the Gantt chart, but I was also in regular contact with both my supervisor, as well as Samantha and Thomas from NYAS and Children's Services. This consistency in communication was more than enough to satisfy all parties involved which was a key concern of mine when I first started as I felt the responsibility and weight of the project may keep me from doing my university work and updating the client.

As is the case with all independent projects, the last couple of weeks of the project were the most stressful due to finding several bugs during the testing stages. Thankfully however with the use of GitHub for version control, this was not a major problem during the larger portions of coding. I managed to maintain regular commits to my git repository which reduced the risk of any lost work from accident or buggy deployments. On top of personal backups within zip folders and cloud storage, the process of developing using industry practices was a contemporary concept for me and involved regular commenting, commits and testing which I had typically never done on past work. Even though I was never faced with any large-scale problems, having this security in my work bestowed me with confidence that helped alleviate stress whilst working. Please refer to my git repository found here: <https://github.com/alfierowett/Final-Project> .

To conclude, this project has helped me in my determination to learn and join the web-development industry upon finishing university. With high expectations from myself, my supervisor, and the client, I am proud of the work I achieved and feel grateful for the opportunity to work on a project that can benefit so many people in the care system. I found the entire experience very rewarding from both a moral and an educational standpoint and I hope to take all the fundamentals I learnt from this independent project and apply them to all of my future work and career in computer science.

References

Thomson, D. 2020. *Demystifying Digital Accessibility with GDS*. [Online Webinar hosted by texthelp]. 25th February 2020. Available at: https://mautic.texthelp.com/digital-accessibility-gds-webinar-resources?utm_content=Accessibility%20and%20inclusion%20advice%20from%20GDS&utm_medium=Email&utm_source=Mautic&utm_campaign=Government%7CUK-WP-RD-GDS-Webinar-WBN-2021-FEB&fbclid=IwAR088a1SebAxFf_RVMtp83WWYim4dzy3-OOYTxpqmJLmGWjDNWQ7EkKefs

The Fostering Network. 2020. *Fostering Statistics*. Available at: <https://www.thefosteringnetwork.org.uk/advice-information/all-about-fostering/fostering-statistics?fbclid=IwAR0n5koLMuY9JHfNhtQqtFMEYI7dnnYIM-8IoD5b0zxUFp-ckuKDCFb5KOg>

Django, 2021. *Django Documentation*. Available at: <https://docs.djangoproject.com/en/3.2/> [Accessed: Feb-April 2021]

Mozilla Developer, 2021. *Django Web Framework (Python)*. Available at: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django> [Accessed: Feb-April 2021]

Ivy, 2020. *Django (3.0) Crash Course Tutorials*. Available at: https://www.youtube.com/watch?v=tUqUdu0Sjyc&t=210s&ab_channel=DennisIvy

Schafer, 2018. *Python Django Tutorial: Full-Featured Web App*. Available at: https://www.youtube.com/watch?v=a48xeeo5Vnk&t=866s&ab_channel=CoreySchafer

StatsWales, 2021. *Children's Services*. Available at: <https://statswales.gov.wales/Catalogue/Health-and-Social-Care/Social-Services/Childrens-Services>

Nielsen, J. 1994. 10 Usability Heuristics for User Interface Design. Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/>

Babich, N. 2019. The 4 Golden Rules of UI Design. Available at: <https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/>

Balsamiq. 2021. Introduction to Balsamiq Mockups 3. Available at: <https://balsamiq.com/assets/wireframes/mockups3fordesktop/balsamiq-mockups-3-for-desktop-documentation.pdf> [Accessed: February, March 2021]

Angle Studios. 2021. Why Mobile-First Web Design Is Becoming More Important. Available at: <https://anglestudios.co.uk/blog/why-mobile-first-web-design-is-becoming-more-important/> [Accessed: February 2021]

Apple. 2021. iOS Design Themes. Available at: <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/> [Accessed: February, March 2021]

Pinterest. 2021. Pinterest. Available at: <https://www.pinterest.co.uk/>

Nielsen, J. 2017. Jakob's Law of Internet User Experience. Available at: <https://nngroup.com/videos/jakobs-law-internet-ux/>

Bourn, J. 2011. Color Meaning. Jennifer Bourns Color Meaning Blog Series. Available at: <https://www.bourncreative.com/meaning-of-the-color-blue/> [Accessed: March 2021]

Thinkful. 2021. Web Development Best Practices. Available at: <https://thinkful.com/blog/web-development-best-practices/>

Bootstrap. 2021. Navs. Available at: <https://getbootstrap.com/docs/4.4/components/navs/>

PythonAnywhere, 2021. Deploying an existing Django project on PythonAnywhere. Available at: <https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/>