Development of an OCR Web Application with Tesseract.js and MERN

Clemente Gonzales

N00193107

Supervisor: Cyril Connolly

Second Reader: Catherine Noonan

Year 4 2022-23

DL836 BSc (Hons) in Creative Computing

Abstract

The overall aim of the project was to study OCR (Optical Character Recognition) and learn how to develop a web application that contained this functionality. This involved an application that allowed users to take notes from various platforms and transfer those notes into one central application.

Various steps were taken to accomplish this goal, which includes requirements gathering, design, implementation, testing, and project management. These steps are explained in greater detail and documented in the report.

Various tests were carried out after the implementation of the application, which resulted in finding underling issues and allowing these issues to be fixed before deployment.

Further work that could potentially be done would be to allow the application to work with React Native and have a mobile version available.

Acknowledgements

I would like to thank my supervisor Cyril Connolly for providing great assistance and guidance and for pushing me towards creating a polished product, as well as being very encouraging and understanding throughout the duration of this project.

I would like to thank my peers for participating in surveys and testing whenever possible, giving needed criticisms on the usability, and finding underlying problems within the application.

Finally, I would like to thank my family for giving me the opportunity to study and complete this degree.

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Clemente Gonzales

Signed

Clemente Gonzales

Failure to complete and submit this form may lead to an investigation into your work.

Table of Contents

1 Introduction 1

2 Researching Optical Character Recognition 2

2.1 Introduction to Text Recognition 2

2.2 Optical Character Recognition 2

2.2.1 A Brief Introduction to Optical Character Recognition 2

2.2.2 Overview of The General OCR Algorithm 3

2.2.3 A Brief Overview of OCR Libraries 11

2.3 User Experience / Interfaces 14

2.3.1 Introduction to User Experience 14

2.3.2 UX / UI Design Principals 14

2.3.3 Responsive Design 16

2.3.4 Design Systems 17

2.4 A Comparison of EasyOCR and Tesseract 18

2.4.1 Introduction 18

2.4.2 Comparison of the two 18

2.5 Conclusion 23

2.6 Summary 24

3 Requirements 25

3.1 Introduction 25

3.2 Requirements gathering 25

3.2.1 Similar applications 25

3.2.2 Interviews 31

3.2.3 Survey 32

3.3 Requirements modelling 41

3.3.1 Personas 41

3.3.2 Functional requirements 45

3.3.3 Non-functional requirements 46

3.3.4 Use Case Diagrams 47

3.4 Feasibility 49

3.5 Conclusion 49

4 Design 51

4.1 Introduction 51

4.2 Program Design 51

4.2.1 Technologies 51
4.2.2 Structure of React 52
4.2.3 Model View Controller Design Pattern 56
4.2.4 Application architecture 57
4.2.5 Database design 58
4.2.6 API design 60
4.2.7 Process design 61
4.3 User interface design 62
4.3.1 Wireframe 62
4.3.2 User Flow Diagram 64
4.3.3 Style guide 64
4.4 Conclusion 69
5 Implementation 70
5.1 Introduction 70
5.2 Technologies 70
5.3 Implementation Roles 71
5.4 Scrum Methodology 71
5.5 Development environment 74
5.5.1 Visual Studio Code 74
5.5.2 GitHub and Git 74
5.6 Sprint 1 - Research and Requirements 75
5.6.1 Goals 75
5.6.2 Item 1 – Researching Project Ideas 75
5.6.3 Item 2 – Researching OCR Technologies in React 75
5.6.4 Item 3 – Gather User Requirements 76
5.6.5 Item 4 – Creating User Personas 77
5.6.6 Item 5 – List of Functional and Non-Functional Requirements of the application 77
5.6.7 Item 6 – Creating Use-Case diagrams. 77
5.6.8 Conclusion of Sprint 1 77
5.7 Sprint 2 – Design 78
5.7.1 Goal 78
5.7.2 Item 1 – Researching the Technologies. 78
5.7.3 Item 2 – Database Design 78
5.7.4 Item 3 – Process Design 79
5.7.5 Item 4 – User Interface Design 79
5.8 Sprint 3 – Implementation 1 82

5.8.1 Goal 82
5.8.2 Item 1 – OCR Functionality Prototype 82
5.8.3 Item 2 – Notes Taking Prototype 84
5.8.4 Item 3 – User Register and Login Prototype 93
5.9 Sprint 4 – Design 2 97
5.9.1 Goal 97
5.10 Sprint 5 – Implementation 2 and Testing 107
5.10.1 Goal 107
5.11 Sprint 6 – Testing 134
5.11.1 Goal 134
5.12 Sprint 7 – Thesis 156
5.12.1 Goal 156
5.13 Conclusion 167
6 Testing 168
6.1 Introduction 168
6.2 Functional Testing 168
6.2.1 Navigation 168
6.2.2 CRUD 172
6.2.3 Discussion of Functional Testing Results 174
6.3 OCR Reader Accuracy Testing 175
6.3.1 No Pre-Processing 176
6.3.2 Gaussian Blur 178
6.3.3 Image Dilation 179
6.3.4 Colour Inversion 180
6.3.5 Threshold Filter 181
6.3.6 Mixing Filters 182
6.3.7 Conclusion 184
6.4 User Testing 185
6.4.1 Application Overview 185
6.4.2 User Tasks 185
6.4.3 Usability Testing Participants 187
6.4.4 Usability Factors 187
6.5 Conclusion 189
7 Project Management 190
7.1 Introduction 190
7.2 Project Phases 190

7.2.1 Proposal 190
7.2.2 Requirements 190
7.2.3 Design 191
7.2.4 Implementation 191
7.2.5 Testing 191
7.3 Teamwork 192
7.3.1 Roles 192
7.3.2 Communication 192
7.3.3 Difficulties 192
7.3.4 Resolving Difficulties 192
7.4 SCRUM Methodology 193
7.5 Project Management Tools 194
7.5.1 Pen and Paper Notes 194
7.5.2 GitHub 195
7.5.3 Journal 195
7.6 Reflection 196
7.6.1 Your views on the project 196
7.6.2 Completing a large software development project 196
7.6.3 Working with a supervisor 196
7.6.4 Technical skills 197
7.6.5 Further competencies and skills 197
7.7 Conclusion 197
8 Conclusion 198
8.1 Final Words 199
9 References 200

1 Introduction

The overall aim of this project is to study OCR and discover methods in allowing developers to implement this functionality in a web-based application.

This document will discuss and display the process shown in developing this system, as well as studying OCR as a technology, with the various ways to use it.

The final application will be web-based, which will act as a note taking app, with functionalities that allow users to extract text from images.

The technologies that will be used in the application consist mainly of the MERN Stack, which will mainly be used to develop the full stack application, TesseractOCR.js which is a library for the OCR functionality, and Quill.js which will be used for the notes taking section. Various other technologies will also be used but will be discussed in their respective chapters.

Management of this project was done with various tools such as Git and GitHub. These tools were used for version control and show the progress that will be made and what will be completed at each stage of development.

A to-do list will also be used to take care of the different tasks that need to be completed on that day.

The project is separated into different stages, which consist of Researching OCR and UX/UI, Requirements Gathering, Design, Implementation, and Testing.

Requirements gathering will gather any relevant information from users / other applications, etc. and gather any relevant functionalities that should be included in the project.

Design will be covering both the program design and the user interface design.

The Implementation will be displaying how the application will be developed and will cover in detail how functionalities were created.

Once this is completed, the next step will be to perform both user testing and functional testing, to test the application and see if everything works as intended.

We will now discuss the requirements gathering.

2 Researching Optical Character Recognition

2.1 Introduction to Text Recognition

A system with the ability to recognize text within natural images have many practical applications that can be useful for society. These systems, for example, can be essential for users who are visually impaired to allow them to go through different environments, such as a small shop, or simple navigation through different roads and streets within the city.

Text recognition can allow for this kind of usage, as text within natural images provide a large amount of information within it, however despite this usage, it is difficult to process text within natural images.

While there are methods to read different characters using machine learning. There are many problems when reading images from text. Font variation, backgrounds, textures, and lighting can provide challenges to the system, and are present within every image (Wang, (2012, November)).

The goal of this research report is to investigate potential methods to help develop a system that can read text from different images, and the methods used to develop them. One of which, is Optical Character Recognition.

2.2 Optical Character Recognition

2.2.1 A Brief Introduction to Optical Character Recognition

Optical Character Recognition (OCR) is a heavily researched area in the field of machine learning, artificial intelligence, and computer vision (Ranjan, (2021)). The way OCR works is that a piece of software takes an image as an input, and reads the text and sentences within that image, and converts it into digital text, such as ASCII or Unicode (Matei, 2013). This allows the computer to understand the text being presented to it.

There are many applications that can be endearing for the further study of OCR. Companies within a paper-intensive industry can benefit greatly from the use of this kind of software, where a large collection of forms and documents can be found. This can include the legal industry, banking, healthcare, captcha, optical music recognition, and automatic number recognition (Singh, 2012), and can provide great benefits to these industries.

Despite these benefits, there are still limitations. Such examples consist of the speed and accuracy of an OCR program. Many OCR programs will suffer from either being a fast but inaccurate program, or slow but highly accurate one (Matei, 2013). Two examples of this will be looked at later.

This is because OCR software uses Neural Network models to recognize new characters and texts based on previous training. These models are trained from large datasets, consisting of thousands of samples to “learn” how a specific character will look (Ranjan, (2021)).

Due to this, there is a large amount of research in finding ways to improve OCR algorithms to create better accuracy, with lower load times (Matei, 2013).

2.2.2 Overview of The General OCR Algorithm

Figure 1 displays the general overview of the algorithm. Each of these steps have their own tasks required for the algorithm to predict what the image is representing to text:

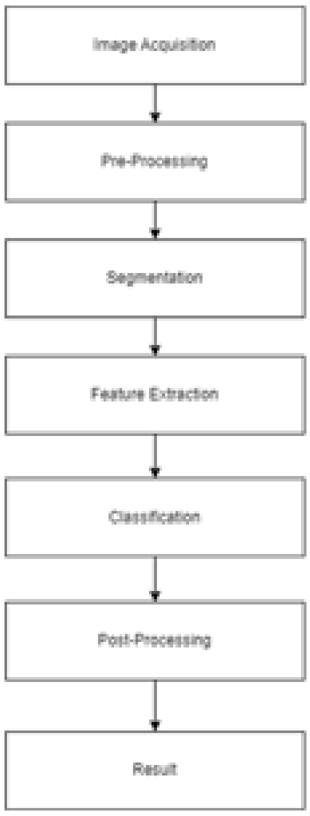


Figure 1 General Overview of the Algorithm

Image Acquisition: Is the process of retrieving the image using a hardware system such as a camera from a phone (Ong, 2016). This can consist of a photograph being uploaded to the system and retrieving the image through there, and the image is fed to the system as an input (Ranjan, (2021)). The image should also contain the text that the user wishes to extract, and display on the computer (Matei, 2013).

Pre-Processing: Is the steps that are required to enhance the image for segmentation (Ranjan, (2021)). This can consist of the image being changed to a monochrome colour to allow for easier readability (Matei, 2013).

Segmentation: This is where the document is segmented into rows, and columns, to extract the words that are within the document (Ong, 2016).

The angles within the segments will be inputted to the neural network, and for the algorithm to confirm it (Matei, 2013).

Feature Extraction: Feature Extraction is when the features of different symbols are extracted (Matei, 2013). Symbols are characterized and meaningless features are left out of the algorithm. These extracted features are used to train the system (Mithe, 2013)**.**

Classification: The tested image is added to the program for classifying. There are several techniques that can be applied to classification. These methods can include artificial neural networks, support vector machines. During classification, the tested image's features are compared with the pattern that is in the training dataset (Ong, 2016).

Post-Processing: Is the process that helps to improve the accuracy of the recognition. This can consist of the analysis of the syntax and the semantics (Ong, 2016).

Grouping, error detection and correction normally occur within this phase of the process (Eikvil, 1993).

2.2.2.1 Image Acquisition

The OCR program requires an input of an image, and within that image containing text, that can be extracted. The image quality and resolution are essential for a clear reading, and an accurate answer (Ranjan, (2021)). Parameters such as colour, fuzziness, lighting, and clarity can affect the reading and answer.

Acquiring the images can come from a variety of sources. These can range from:

1.) Photographs: These can come from mobile phones, that contain the required text that the user wishes to extract (Ranjan, (2021)).
2.) Scanned Documents: Documents that have already been scanned and stored as an image can be inputted to the system (Ranjan, (2021)).
3.) Screenshots and digital images: Mobile phone or computer applications allow users to take screenshots of their screen. This can be used to inserted into the OCR system and be processed the same way as other photographs can be too. Digital images are the same in the sense that they can also be processed through the system to extract text (Ranjan, (2021)).

Through the scanning process, an image of the original document is captured, and optical scanners are used. These optical scanners normally contain a transport mechanism and a sensing device that can allow the system to convert light intensity into grey levels (Eikvil, 1993). This allows the system to read the text in a monochromatic way, ensuring that it remains more accurate than with colour.

It is important to have a clear reading of the image, including the light levels and readability of the text, as the text gets converted into black and white for the program to read. Like less noise, a focus on the text, and lighting should be taken into consideration (Ranjan, (2021)). The result of the reading can be heavily affected if these aren't taken into consideration.

This process of converting an image to a bilevel image of black and white is often referred to as thresholding. This process is essential as the quality accuracy of the result is dependent on the quality of the bilevel image. The best forms of thresholding are generally ones that are adaptive and can take into consideration the properties of the brightness and the contrast of the image to result in a better and clearer image. These methods however generally are dependent on multilevel scanning and requires much ore computational capacity, resulting in less usage (Eikvil, 1993).

These papers allude to fact that the image should be clear and readable for the best possible effect, as image processing applications are done using grey scale images, to make the processing efficient (Ranjan, (2021)).

2.2.2.2 Pre-Processing

The result of the image that has been scanned through may contain a certain amount of noise, depending on how clear and readable the image is, as the characters may be smudged or broken. Some of these defects can be eliminated using pre-processing, to smooth out these digitized characters (Eikvil, 1993). This process can help optimize the reading and allow for a clearer answer.

During this phase, smoothing and normalization occurs. Smoothing allows rules to be applied to the image, with the help of filling and thinning techniques. Filling removes the small breaks, holes and gaps in the digitized characters, and thinning reduces the width of the line. A technique that is commonly used in smoothing is to move a window across a binary image of that specific character, and while this is there, it applies certain rules to what is inside of the window (Mithe, 2013)**.**

Normalization handles the size, slant, and rotation of the character. For example, if the letter "J" is off centred, and leaning more towards the right, normalization will fix this issue to allow for easier readability and centre it (Mithe, 2013)**.**

The image is also sharpened to enhance the high frequency details. This is to emphasize the edges in the image, to make it easier for the machine to pick out the patterns and letters (Matei, 2013).

The reason for pre-processing is to allow for easier classification for the program to detect the different strokes and classify which belongs to which letter in the alphabet (Ong, 2016).

Another step-in pre-processing is applying adaptive thresholding, which is used to segment the image. This is done by setting the pixels whose values are above a certain threshold to the front value, and all remaining pixels, who do not meet this threshold to the background value (Matei, 2013).

Once pre-processing is complete, it is now ready for the next stage.

2.2.2.3 Segmentation

Segmentation is the process of locating sections of printed or handwritten text. Segmentation distinguishes text from figures and the image. When segmentation is applied to text, it isolates characters or words that are within that text (Mithe, 2013)**.**Text lines are first segmented, then within these lines of text, the words are then further segmented, and finally from the words, the characters are segmented (Rao, 2016). It is essential to locate the sections of the document where text have been written and identify them from figures and graphics. For example, in the process of automatic mail-sorting, the address must be located and separated from other print on the envelope like the stamps or company logos, before we get to recognition (Eikvil, 1993).

Most OCR algorithms will segment words to isolated characters that gets recognized individually. In general, this form of segmentation is done by isolating each connected component, that is each connected black area. It is easy to implement, however there are some problems that occur if the characters are touching or are broken and consist of multiple parts (Eikvil, 1993). These can lead to the system recognizing a completely different word or leading to inaccuracies.

According to (Eikvil, 1993), there are a few problems within segmentation, and are as follows:

• Extraction of characters that are touching or fragmented. These distortions can lead to multiple joined together characters being understood as one single character, or that a piece of a character is thought to be an entirely different symbol. These joints can occur if the photograph or document is a dark photocopy or if it is scanned at a low threshold. Joints are also common if the fonts are serifed. If the document has a light photocopy or is scanned at a high threshold, it is also possible for a split to occur due to the circumstances of the scan (Eikvil, 1993).

• Distinguishing noise from text can also be an issue. Dots and accents might be misinterpreted as noise, and vice versa, causing inaccuracies to the finished product (Eikvil, 1993).

• Misinterpreting an image or geometry and seeing them as text. This can lead to non-text being sent to recognition and being unable to read it (Eikvil, 1993).

• Misinterpreting text as an image or geometry. In this scenario, it is possible that the text will not be passed to the recognition stage. This often occurs if characters are attached to graphics.

2.2.2.3.1 Segmentation methods

Document segmentation is a key pre-processing stage in applying an OCR system. It is the process of classifying a document image into homogeneous zones, i.e., that each zone contains only one kind of information, such as text, a figure, a table, or a halftone image. In many cases, the accuracy rate of systems related to the OCR heavily depends on the accuracy of the page segmentation algorithm used (Rao, 2016).

There are three categories of Algorithms of document segmentation according to (Rao, 2016). These are as follows:

- Top-down methods, which is when a document is segmented from large regions into smaller regions recursively. The segmentation process will stop when it reaches a stage that meets a criterion, i.e., it reaches the final range of segmentation (Rao, 2016).

- Bottom-up methods will search for interest pixels, and groups these interest pixels. They manage the interest pixels into associated elements that represent characters that are combined into words, or lines, or blocks of text afterwards (Rao, 2016).
- Hybrid methods is an integration of both approaches seen from above (Rao, 2016).

2.2.2.4 Feature Extraction

Feature extraction is the process of extracting relevant features from objects or alphabets to build a feature vector. The objective of this step is to capture vital traits and symbols (Eikvil, 1993). These vectors are used by classifiers to identify the input unit with the objective output unit. The easier the features are to determine; the more effortless classification becomes (Rao, 2016).

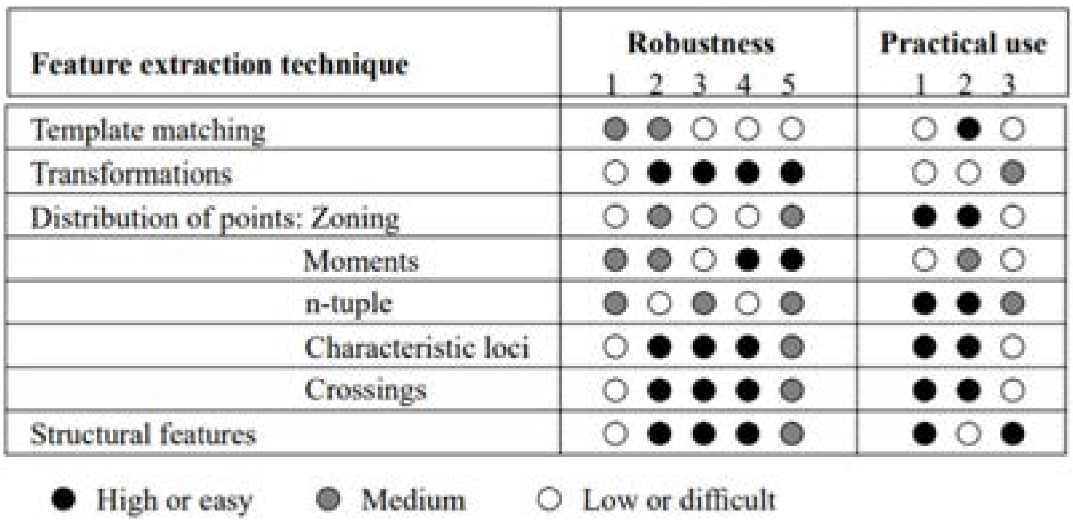
According to (Rao, 2016), there are many methods in which feature extraction can be accomplished. For example, one such method consists of a directional chain code feature, and zoning for handwritten numerical recognition. It consists of a feature vector of length 100 and have a high level of accuracy, however it proves to be time consuming and complex (Rao, 2016).

Another potential method that was seen in (Rao, 2016) is that the end points are the potential features towards recognition. These features use horizontal/vertical strokes and for handwritten numerals obtained a recognition accuracy of 90.50%. Despite this, the method uses the thinning process which can result in a loss of features.

However, according to (Eikvil, 1993), the techniques of feature extraction can be categorized into three different groups, where the features can be extracted. This can consist of:

- The distribution points.
- Transformations and series expansions.
- Structural Analysis

In both (Rao, 2016) and (Eikvil, 1993)**,** we see that noise and distortion can skew the results that are given. (Eikvil, 1993), however, goes into more detail, we can see that noise, distortions, style variation, translation, and rotation can all effect the speed of the recognition, the complexity of implementation and whether the system will need further support to recognize the text. The following table can display an example of feature extraction.



Feature extraction technique	Robustness					Practical use		
	1	2	3	4	5	1	2	3
Template matching	◍	◍	○	○	○	○	●	○
Transformations	○	●	●	●	●	○	○	◍
Distribution of points: Zoning	○	◍	○	○	◍	●	●	○
Moments	◍	◍	○	●	●	○	◍	○
n-tuple	◍	○	◍	○	◍	●	●	◍
Characteristic loci	○	●	●	●	◍	●	●	○
Crossings	○	●	●	●	◍	●	●	○
Structural features	○	●	●	●	◍	●	○	●

● High or easy ◍ Medium ○ Low or difficult

Figure 2 Feature and Extraction techniques from (Eikvil, 1993)

2.2.2.4.1 Distribution Points

As seen in article (Eikvil, 1993) and (Rao, 2016), the distribution points are key features in identifying the essential components of the character. These different techniques can be listed below as:

- Zoning – The rectangle covering the character is divided into different regions and densities of black points (Eikvil, 1993).
- Moments – The moments of black points being chosen as a centre (Eikvil, 1993).
- Crossing - The features are found through the number of times the character shape and the vectors along certain directions are crossed (Eikvil, 1993).
- N-Tuples – The joint occurrence of the foreground and background (black and white) points in a certain order being identified as features (Eikvil, 1993).
- Characteristic Loci – Vertical and Horizontal vectors are generated. The segments of a character that intersects with the vector is used as a feature to describe the character (Eikvil, 1993).
- Transformation and Series expansions - reduce the dimensionality of the feature vector and the extracted features can be made invariant to global deformations (Eikvil, 1993).

2.2.2.4.2 Structural analysis

Structural analysis allows features that describe the geometric structure of a symbol to be identified and extracted. These can include the physical characteristics of the character, commonly identifying loops, strokes, intersections, lines, bays, and endpoints. Structural analysis is a technique with a large tolerance to style of the character and noise variations but are not very tolerant to rotation and translation (Eikvil, 1993).

2.2.2.5 Classification

The classification state is when the features are extracted to identify the text segment that is according to the rules. Classification is generally accomplished by the comparison of feature vectors that correspond to the input characters with the representative of each character class (Lehal, 1999).

Before doing this however, the classifier should possess a range of training patterns (Verma, 2012). There are many classification methods, that have been proposed by different researchers, and some of them will be looked at in a later section. These techniques can consist of statistical methods, template matching, syntactic methods, artificial neural networks, and kernel methods (Verma, 2012).

According to (Lehal, 1999), the nearest neighbour classifier (statistical method as mentioned in (Verma, 2012)) and binary classifier trees have been the two most common classifiers.

2.2.2.5.1 Statistical methods (K-Nearest Neighbour)

The reason behind using the statistical methods is to determine which category the pattern belongs to. This is done through the observation and measurement process, where it prepares a set of numbers that is used to prepare a measurement vector (Lehal, 1999).

The K-NN rule is a non-parametric recognition technique. The method compares unknown patterns received from the feature extraction stage to a set of patterns that have been labelled with class identities during the training stage. A pattern is then recognized to be of the class of pattern, to which it has the closest distance (Verma, 2012).

This technique is effective for classification problems where the patterns consist of a limited number of variations. For clear and specific machine-printed text, the patterns of each class tend to be clustered tightly towards the patterns that represent that class. The nearest neighbour approach can be effective method of classification; however, it can suffer from memory and size issues if more fonts are added (Lehal, 1999).

Other common statistical methods are Bayesian classification, which assigns a pattern to a class with max posteriori probability, Quadratic Discriminant Function (QDF), Linear Discriminant Function (LDF), Cross Correlation, Regularized Discriminant Analysis (RDA) and Euclidean Distance (Verma, 2012).

2.2.2.5.2 Artificial neural networks

A neural network is an architecture that contains a massive interconnection of flexible node processors (Rao, 2016). The output from one node reinforces the next one in the network, and a result is garnered from the complex collaboration of all nodes. Feed-forward and feedback neural networks can be considered as a categorization of a neural network architecture (Rao, 2016).

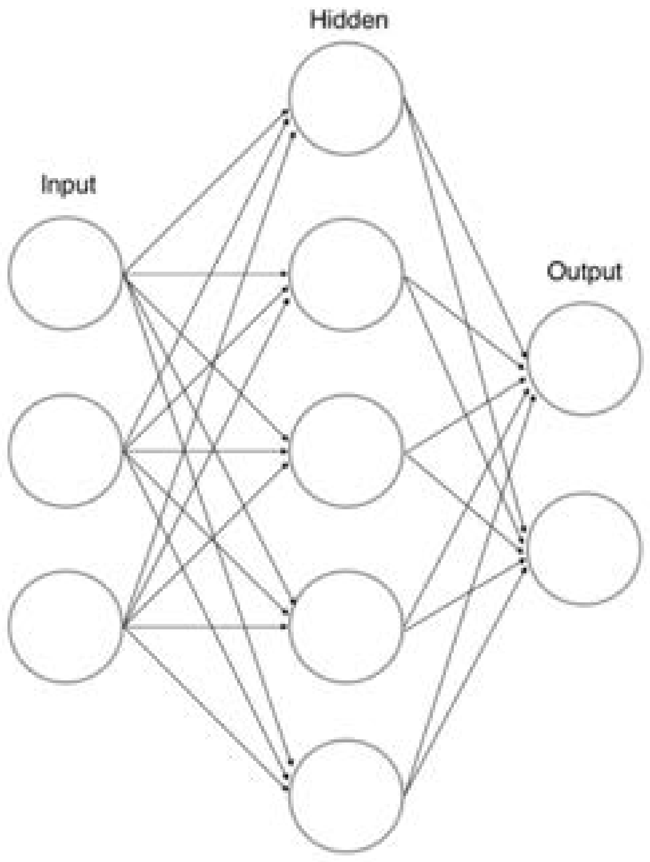


Figure 3 (Lopez, 2017), Basic ANN Structure

In (Lopez, 2017), we can see the basic structure of an artificial neural network. The network contains of an input, hidden, and an output layer. Training is done using a method known as back-propagation.

In (Eikvil, 1993), we see the use of back propagation, where it mentions that within this kind of network, which consist of many layers of interconnected elements, when a feature vector enters as an input through the input layer, each element of the layer calculates the weighted sum of the input and transforms it into an output done using a non-linear function. While the neural network is training, the weights are adjusted until the required output is achieved. The problem with a neural network in OCR is their limited predictability due to training, however they are very adaptive which provides to be a general advantage. Therefore, libraries such as OpenCV, who uses CNN for OCR is widely used, because of this adaptability.

According to (Verma, 2012), the most used neural networks for OCR and the pattern classification task is the feed-forward network, the Radial-Basis Function (RBF) networks, Convolutional Neural Networks (CNN), Vector Quantization (VQ), Learning Vector Quantization (LVQ), and auto-association networks.

An interesting study that was found in (Matei, 2013), shows that combining the two methods of using K-NN and Neural networks can provide staggering results. Their study uses Artificial Neural Network and K-NN as a confirmation algorithm, combining the two, and bases the vectors on the angles of digits rather than pixels. Some advantages found within their system consist of the ability to work in different light levels and exposure conditions, insensitivity to rotation and being able to deduct and use exploratory character recognition which provided great success with moderate levels of training.

2.2.2.5.3 Template matching

One of the simpler methods towards pattern recognition, template matching is the approach where a prototype of the pattern that must be recognized is available. The pattern that is to be recognized is compared with patterns that are already stored and ignores the size and style of these patterns (Verma, 2012).

2.2.2.6 Post-Processing

Post-Processing is the final stage of the program, where it is used to make corrections towards spelling errors, and the grouping of the words together (Eikvil, 1993).

- **Grouping:** The result of the symbol recognition on a document will result in individual symbols. Grouping will take these symbols and group them together where they each belong with each other, making words, numbers, and sentences. This process is known as grouping, where symbols that are found to be adequately close are grouped together (Eikvil, 1993).
- **Error-detection and Correction:** In advanced OCR problems, a system consisting only of single-character recognition will not be sufficient. Errors will often occur, and a correction must be provided. One of these approaches is done using syntax, where it follows the rules of grammar, for example a capital after a full stop. Another approach is done using dictionaries, where it can provide to be the most efficient method for error detection and correction (Eikvil, 1993).

The next section will discuss how the general algorithm is used in OCR libraries.

2.2.3 A Brief Overview of OCR Libraries

2.2.3.1 Tesseract

Tesseract is an open-source OCR engine developed in 1984 – 1994 at HP. Tesseract uses the Line Finding algorithm. This algorithm is designed so that a skewed image has the capability to be recognized without having to de-skew the image, which saves the image quality. Blob filtering and line construction is some of the key parts of the process. More details of the process can be found at (Smith, 2007, September). However, the process is like what is described in the general overview of the algorithm section above.

In short however, the program first reads the text and converts it into black and white outlines, which are then converted into Blobs. These Blobs are arranged into text lines, where the lines and regions are then analysed for some fixed area or are of equal text size. Text is divided into two words, with these being definite space and fuzzy spaces. Recognition is also started as a two-pass (Patel, 2012).

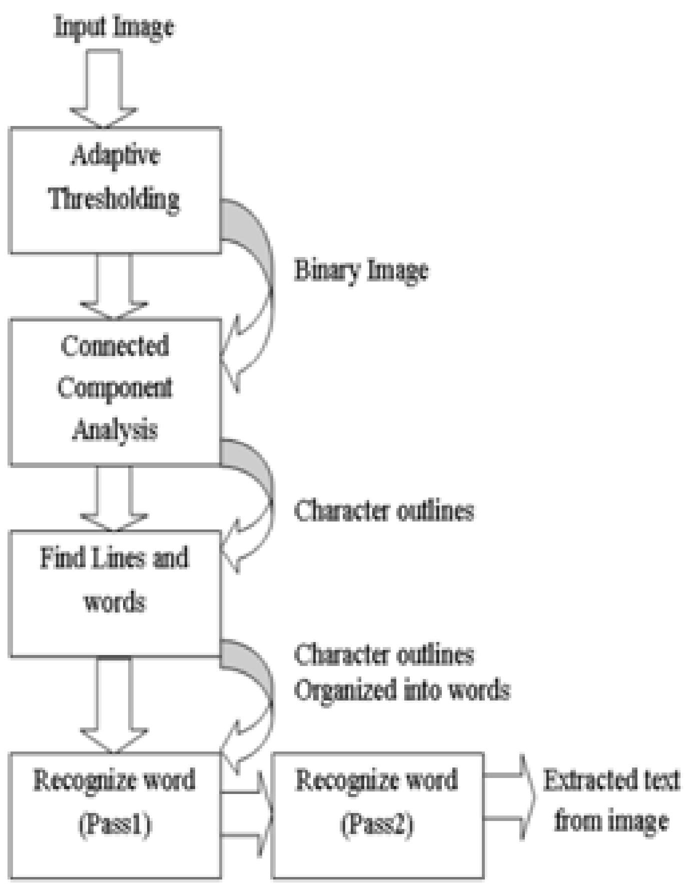


Figure 4 Tesseract Architecture (Patel, 2012)

Tesseract can provide better accuracy to grey-scale images rather than coloured ones, seen in the experiment conducted in (Patel, 2012). Tesseract works best when it comes to grey-scale images, and as a result works best when working with these kinds of images, with very high accuracy, however, falls short when it comes to images with colour.

This method can be effective in capturing hand-written documents, where it is a simple monochrome colour, and there is not much on the written paper being presented, using tesseract can provide to be an effective solution.

2.2.3.2 OpenCV

Open-Source Computer Vision (OpenCV) is a library with programming functions for real-time computer vision. The library has over 2000 algorithms and has been widely used around the world. Programmers can implement many digital image-processing algorithms for mobile phones (Ma, 2000).

The use of OpenCV can be beneficial in OCR. This is because OpenCV can detect and factor in the different noise levels, and colours that occur when using OCR is regular day-to-day life.

A study in (Goel, 2019), shows a process where it uses Convolutional Neural Networks (CNN) and the OpenCV Library to develop an accurate OCR system. It extracts natural scenes from images, where there can be a wide range of colours, fonts, textures, lighting conditions, etc. and can use the two methods effectively to accurately capture the text from an image (Goel, 2019). From this, it can be safely assumed that it is possible to use OpenCV in open areas, for example a street road, in attempts to examine a sign to see where a user is, and is more accurate than using something like Tesseract, despite taking longer to process.

2.2.3.3 A Comparison of The Libraries

The reason for discussing these libraries is to see their different benefits and advantages. From researching the Tesseract, KerasOCR, EasyOCR, and OpenCV, a conclusion can be drawn that there are many benefits for the use of each library and can be highly effective depending on the goal of the software a person is developing needs to accomplish.

Tesseract is highly effective when used in document reading, where the image and font presented is in a monochromatic colour and noticeably clear to read. This can work best with handwritten documents, where the ink and the paper are highly contrasted. Pairing this with the high accuracy, and lower loading times, it can prove to be the most effective method of OCR in a system, where the main goal is to read notes on paper rather than in the outdoors.

OpenCV however, can be used for recognition in the wild instead, with the drawback of slower loading times is much more optimized for OCR in the wild. The capability of this is mentioned above, where it uses CNN to look at many images and interpret the text.

Overall, the two libraries both provide benefits for different goals, and can compensate for each other’s weaknesses in OCR.

2.3 User Experience / Interfaces

2.3.1 Introduction to User Experience

The concept of User Experience (UX) and User Interface (UI) is a term that is used within many platforms. Specifically in this dissertation, the UX and UI for the web will be reviewed to understand and implement the best user experience possible.

Poorly designed UI can spoil a user's experience with a website, and to avoid this, research will be done to implement standard design practices and foundations that can be globally applied to many websites.

The goal is to give the user the best potential experience when interacting with the website, as well as be an aesthetically pleasing website to look at, and to retain their attention and consistently keep returning.

The next section will be focused on discussing the UX and UI Design Foundations, to help us understand the reasoning and psychology behind these design principals.

2.3.2 UX / UI Design Principals

In this section, the essential design principals will be discussed and researched. The steps and goals for each design principal will be looked at here, starting with the look of the website.

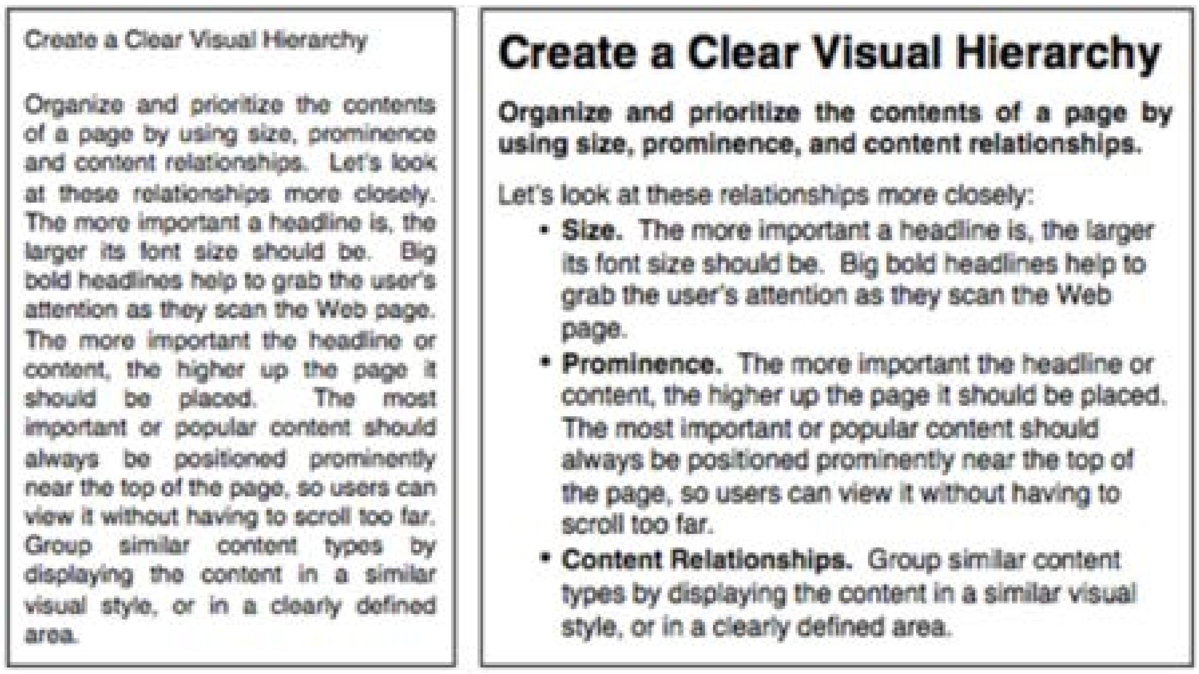
2.3.2.1 Aesthetics and Clarity

As design and aesthetics and visual composition is attractive to the eye, it is possible to convey the idea of your website quickly and clearly through the design alone. Graphics design principles that include contrast, hierarchy, spacing, alignment, and using colour effectively provides the overall look and feel of an interface (Bhaskar, 2011). These elements are essential when creating a web application, as these concepts can also be applied to web-apps.

Based on previous experience, it is effective to build a design system, where the designer chooses a fitting colour palette that suits the feel of the website, choses the typography which complement the website, and each other, and the style of the icons and buttons to be used to help keep the look of the website consistent.

2.3.2.2 Visual Structure / Hierarchy

Visual Structure and Hierarchy is an essential part of designing an interface. A visual hierarchy allows users to organize information into clear categories that can be repeated throughout the website (Fleming, 1998). This allows users to better understand what they are looking at, providing clarity when analysing the body for information. It allows users to separate their relevant goals from irrelevant information that is provided within a body of text (Johnson, 2020).


Create a Clear Visual Hierarchy

Organize and prioritize the contents of a page by using size, prominence and content relationships. Let's look at these relationships more closely. The more important a headline is, the larger its font size should be. Big bold headlines help to grab the user's attention as they scan the Web page. The more important the headline or content, the higher up the page it should be placed. The most important or popular content should always be positioned prominently near the top of the page, so users can view it without having to scroll too far. Group similar content types by displaying the content in a similar visual style, or in a clearly defined area.

Create a Clear Visual Hierarchy

Organize and prioritize the contents of a page by using size, prominence, and content relationships.

Let's look at these relationships more closely:

- **Size.** The more important a headline is, the larger its font size should be. Big bold headlines help to grab the user's attention as they scan the Web page.
- **Prominence.** The more important the headline or content, the higher up the page it should be placed. The most important or popular content should always be positioned prominently near the top of the page, so users can view it without having to scroll too far.
- **Content Relationships.** Group similar content types by displaying the content in a similar visual style, or in a clearly defined area.

Figure 5 (Fleming, 1998) Visual Hierarchy

Looking at fig. 5, it is easy to better distinguish the relevant information on the text on the right. The use of bold and large font for the title, while using smaller text however still in bold for the subheading shows the main ideas that the article provides.

Relevant size can help communicate the relevance of the information provided, as seen in the previous figure. Large items, position of element, and colour and contrast, can help draw a user's attention (Fleming, 1998). This can allow for the communication of information to be more easily done and is not simply font size, but all the other elements listed above can help draw a user's attention.

2.3.2.3 Requirements

The requirements gather phase is an essential part of the design progress. It finds the needs and the subject matter of the application that is being developed. It is essential to understand what is being developed and the plan the functionalities that the system will have.

There is a plethora of different methods to accomplish the requirements gather phase. Those which can be listed below:

2.3.2.3.1 User Observations

Observing user behaviour while interacting with the website can provide to be beneficial when planning the design of the application. Through observing their behaviour, it is possible to see what the users do, and what they like and dislike about the application (Stone, 2005). This can allow the developers to discover flaws that is found within the design and allow the developer to make changes where necessary to provide a better end-product overall.

Direct observation is when a developer is watching a user interact with their system and taking notes on any issues that the user is facing. This is normally done on the user's system

(Stone, 2005).It is however easy for a developer to overlook an important issue when doing direct observation, as the process is not recorded, and cannot be studied after it is over. This is where it is possible to instead record a user's actions for further study.

Indirect observation allows just that, where the user can provide a video recording that allows the developers to watch and analyse the recording to find issues, they might deem necessary to fix (Stone, 2005).

Both methods have their own benefits and are excellent ways to find issues within the design to fix for a much better product.

2.3.2.3.2 Interviews

Interviews can also be conducted when gathering requirements. The interview process is done to find the different features that a user would like to see in an application that the developer is creating. There are generally two types of interviews; a structured interview, where questions are generally pre-written and contains little scope in exploring different topics that arise in conversation, and a flexible interview, where the questions are more broad and can explore and dive deeper into a certain topic within that interview (Stone, 2005). Both provide different benefits. The structured interview can allow developers to gather data on different functionalities on the system of something that has a clearer goal, while the general interview can allow developers to gather more ideas on what kinds of functions a system can have, as well as what the different users would like to see.

2.3.2.3.3 Surveys and Questionnaires

Surveys and questionnaires are like interviews, where they can be used to gather information and is flexible and friendly in gathering more precise information (Stone, 2005). The use of surveys can be used to mass gather information as tens to hundreds of people can take part in a short survey. This is good when gathering requirements, as it allows users to show developers what they wish to see within their program.

There are two kinds of questions when creating a questionnaire or a survey. One of which are known as "closed questions". These consist of closed-ended answers, for example "yes" or "no" (Stone, 2005) and can be used for specific topics that can be answered easily.

The second set of questions are "open questions" where they are open to what the user has to say. These questions can provide rich data from what a user has to say due to the openness of the question (Stone, 2005). For example, this can be used to gather feedback on a certain feature and see how a user would like that feature to be improved upon.

2.3.3 Responsive Design

As technology continues to evolve, responsive design becomes more and more crucial in the role of web development. Many people will own a wide range of screens, where websites might not be optimized for. This problem can be resolved with responsive design.

Responsive web design attempts to mix HTML5 and CSS3 features with new design methodology to website architecture, where it the website will adapt to a browser with varying screen sizes (Gardner, 2011). This allows websites to be flexible and compatible with almost any device and provides to be a great tool for accessibility to a website.

According to (Gardner, 2011), responsive design contains three parts:

- A fluid layout is the use of a flexible grid, that allows a website to scale to the width of a screen / browser. These layouts are responsive and can change to whatever screen it fits. Popular practice can consist of using grid systems which consist of columns, gutters, and rows (Gardner, 2011). A fluid layout can allow for a website to be seen and accessed using many different devices and can potentially lead to more people viewing it
- Images and media being flexible and adapt with the website to the browser that it is being viewed through is also essential. Images should be sized accordingly, depending on the screen size. This can be done using CSS, and it is also possible to keep the resolution by using the "max width" property in CSS (Gardner, 2011). Doing this however can lead to the website loading slower, as the images are mostly kept in high resolution and can provide potential performance issues.
- Media Queries can be used to address usability issues. A common problem that can occur when attempting to develop for all devices is that components are not as optimized as they could be in the location that they are in. For example, if we take a side nav and translate that to a mobile device from a desktop, the side-nav will shrink in width in the mobile version. To combat this, the use of media queries is available, where it can alter the viewing experience and style depending on the device that it is being viewed upon (Gardner, 2011).

All these concepts regarding design can be streamlined using design systems, which we will look at in the next section.

2.3.4 Design Systems

A design system is a group of requirements that manages design through the maintenance of consistency and reduction of redundancy (Kumar, 2022). Design systems aim to keep the look of a website to be consistent and follow rules of good practice in graphic design.

These design systems are commonly very consistent in look and can help when developing a web application. They can also prove to be useful, as they are very user friendly and clear to read.

Examples of design systems are Material UI design, Fluent Design Systems, Atlassian, and Polaris. These design systems are well respected and commonly used throughout the web. Material Ui Design for example is used by Google.

2.4 A Comparison of EasyOCR and Tesseract

2.4.1 Introduction

Before beginning the development of the application to be built, a direct comparison was done to test the accuracy of both EasyOCR and Tesseract. The code for EasyOCR was written in Python, and developed by using Anaconda, and the code for Tesseract was written in JavaScript, more specifically React.js and developed using Visual Studio Code.

This comparison was done to determine the accuracy of both libraries, as both provide great benefits and help to decide on which library is preferable to use. The test will be done with a list of the same images to provide a fair test.

2.4.2 Comparison of the two

The first image to be examined was found in Google Images, with a quick search of a random image to use of a bank sign. In figure 6, the words detected are accurate, however the spacing is slightly off. Nonetheless EasyOCR provides an accurate reading of the sign shown in the image.

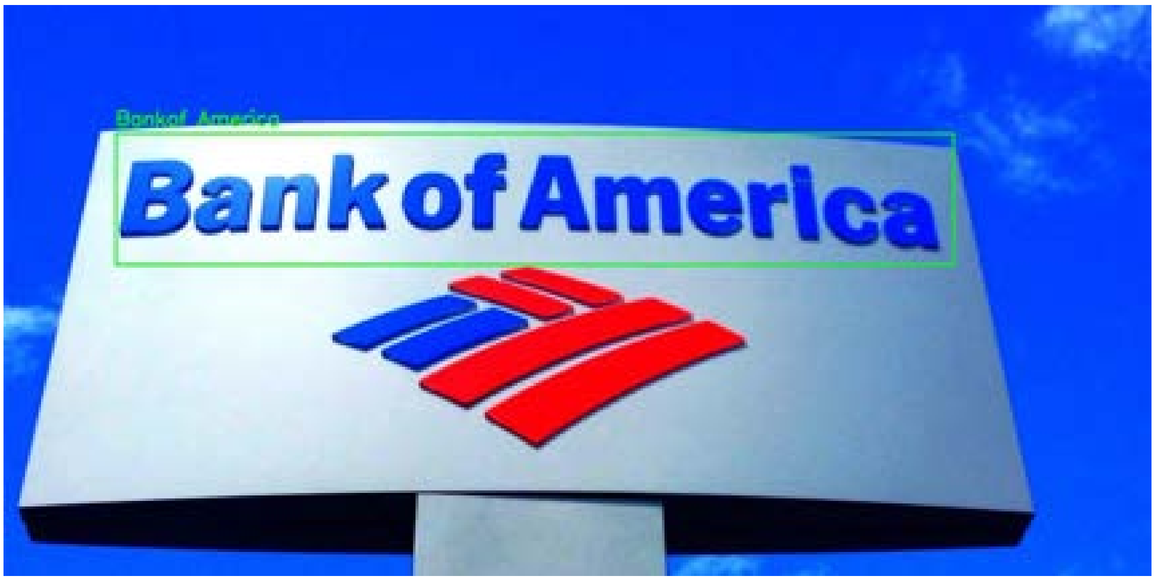
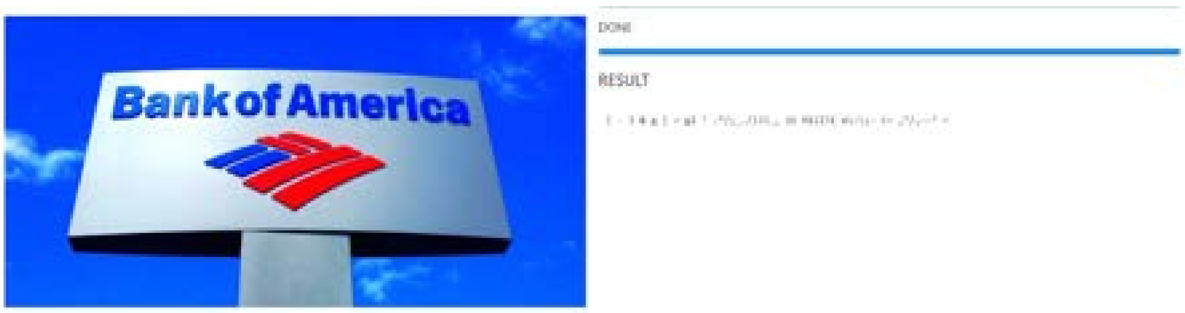


Figure 6 Bank of America (EasyOCR Reading)

The same cannot be said however for Tesseract's reading. As expected, due to the different colour, the Tesseract library struggles with the reading of the letters and displays a very inaccurate result as seen in figure 7.



[illegible]

Figure 7 Bank of America (TesseractOCR Reading)

The next image that was then examined was a simple paragraph, again found with a quick Google search. In figure 8, the letters and words detected are highly accurate, as the letters are clear to read and are not distorted in any way. EasyOCR results are seen in figure 8.

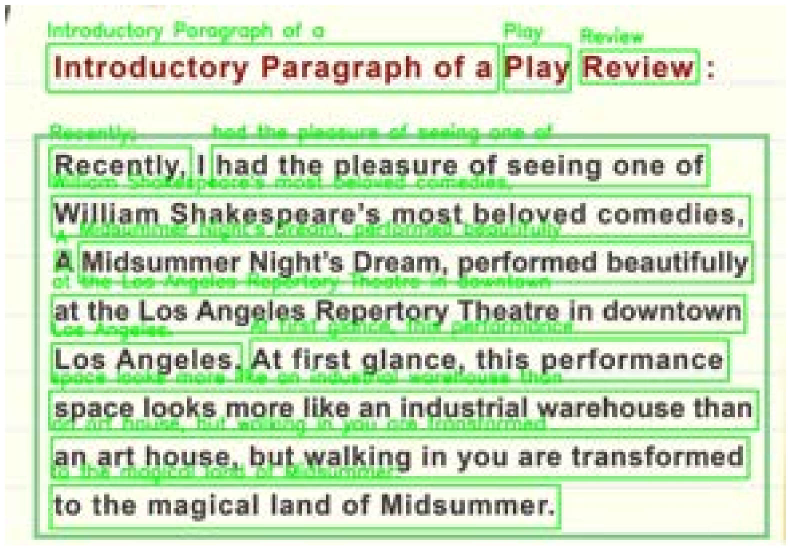
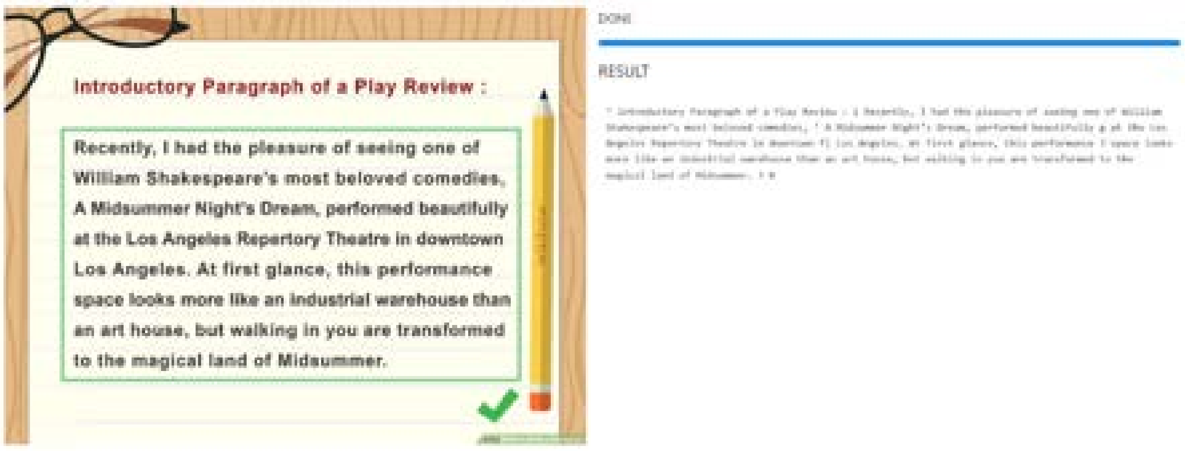

Introductory Paragraph of a
Play
Review
Recently,
had the pleasure of seeing one of
William Shakespeare's most beloved comedies,
A
Midsummer Night's Dream, performed beautifully
at the Los Angeles Repertory Theatre in downtown
Los Angeles.
At first glance, this performance
space looks more like an industrial warehouse than
an art house, but walking in you are transformed
to the magical land of Midsummer.
Introductory Paragraph of a Play Review :
Recently, I had the pleasure of seeing one of
William Shakespeare's most beloved comedies,
A Midsummer Night's Dream, performed beautifully
at the Los Angeles Repertory Theatre in downtown
Los Angeles. At first glance, this performance
space looks more like an industrial warehouse than
an art house, but walking in you are transformed
to the magical land of Midsummer.

Figure 8 Paragraph EasyOCR result

The same can also be seen in figure 9, where Tesseract is then used. It is important to note that the use of Tesseract proved to be much faster than using EasyOCR, and provides highly accurate results in this format, as the words are easy to read and very clear to see.


Introductory Paragraph of a Play Review :
Recently, I had the pleasure of seeing one of
William Shakespeare's most beloved comedies,
A Midsummer Night's Dream, performed beautifully
at the Los Angeles Repertory Theatre in downtown
Los Angeles. At first glance, this performance
space looks more like an industrial warehouse than
an art house, but walking in you are transformed
to the magical land of Midsummer.
DONE
RESULT

" Introductory Paragraph of a Play Review : i Recently, I had the pleasure of seeing one of William Shakespeare's most beloved comedies, ' A Midsummer Night's Dream, performed beautifully g at the Los Angeles Repertory Theatre in downtown fl Los Angeles. At first glance, this performance f space looks more like an industrial warehouse than an art house, but walking in you are transformed to the magical land of Midsummer. | a

Figure 9 Paragraph Tesseract Result

The next test was to test the accuracy of the application when reading letters and paragraphs through common household objects, in this case from a swimming pool box. In figure 10, EasyOCR again proves to be highly accurate when reading letters, as it has very little inaccuracies as seen in the figure below.

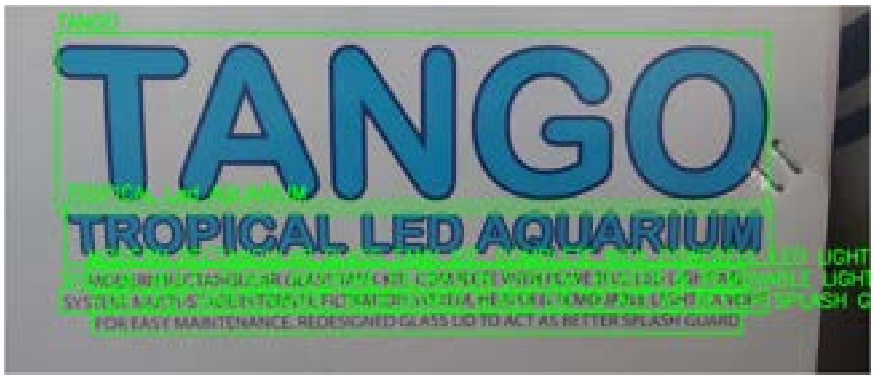
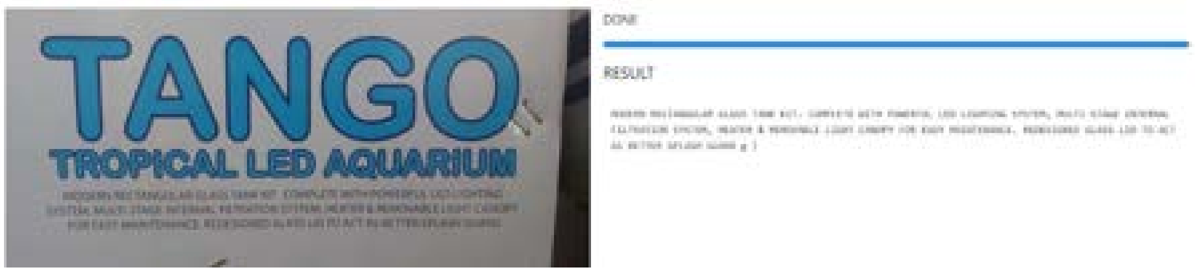


Figure 10 EasyOCR Reading of Swimming Pool Box

TesseractOCR also proves a very accurate reading of the paragraph seen in the bottom of the box. Although it cannot read “Tango”, as it is in colour and struggles to read it, the paragraph at the bottom of the pages shows its accuracy as it is a clear, black font and easily seen against a contrasting background.



DONE

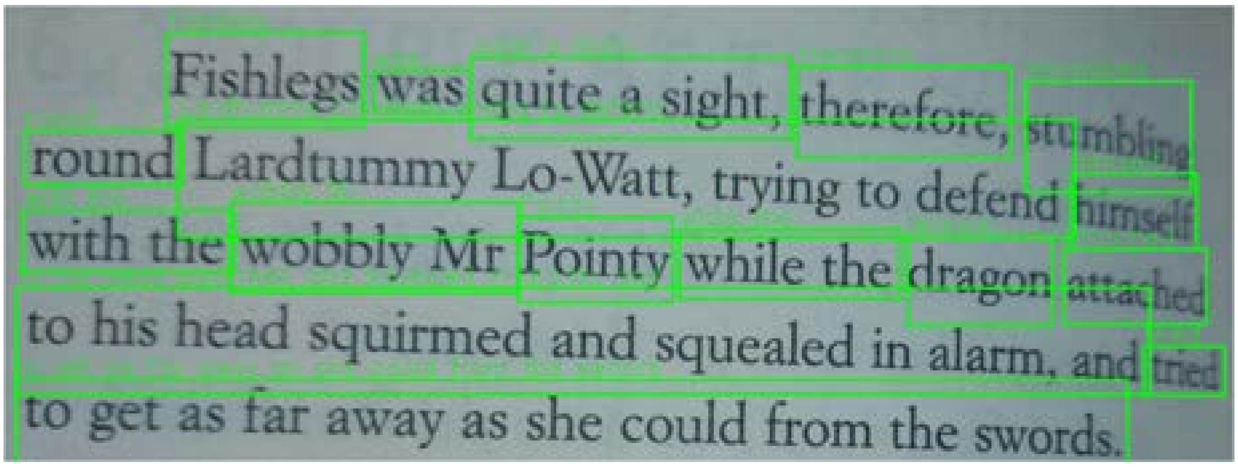
RESULT

```
MODERN RECTANGULAR GLASS TANK KIT, COMPLETE WITH POWERFUL LED LIGHTING SYSTEM, MULTI STAGE INTERNAL
FILTRATION SYSTEM, HEATER & REMOVABLE LIGHT CANOPY FOR EASY MAINTENANCE, REDESIGNED GLASS LID TO ACT
AS BETTER SPLASH GUARD g |
```

Figure 11 TesseractOCR Reading of Swimming Pool Box

Although TesseractOCR struggles with different colours and typography, a test was done with both libraries to see how the two would compare against regular fonts from a novel.

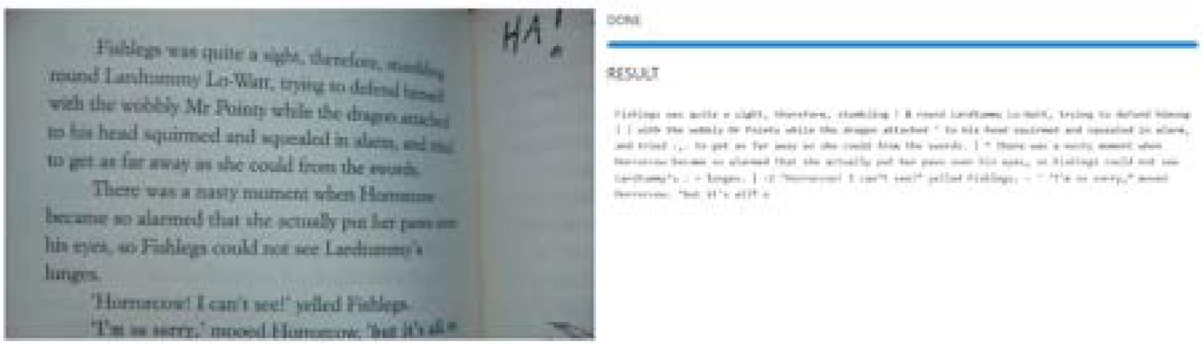
EasyOCR was tested first and can be seen in figure 12. Although it is quite difficult to see, the reading was highly accurate with the use of EasyOCR. This process however took around one to two minutes to load, showing that it takes a very long time to return results.



Fishlegs was quite a sight, therefore, stumbling
round Lardtummy Lo-Watt, trying to defend himself
with the wobbly Mr Pointy while the dragon attached
to his head squirmed and squealed in alarm, and tried
to get as far away as she could from the swords.

Figure 12 EasyOCR Book Reading

TesseractOCR on the other hand only took around 20 – 30 seconds to read the following paragraph and provided highly accurate results. These results can be seen in figure 13.



Fishlegs was quite a sight, therefore, stumbling
round Lardtummy Lo-Watt, trying to defend himself
with the wobbly Mr Pointy while the dragon attached
to his head squirmed and squealed in alarm, and tried
to get as far away as she could from the swords.
There was a nasty moment when Horrorcow
became so alarmed that she actually put her paws over
his eyes, so Fishlegs could not see Lardtummy's
lunges.
'Horrorcow! I can't see!' yelled Fishlegs.
'I'm so sorry,' mooed Horrorcow, 'but it's all

HA!

DONE

RESULT

```
Fishlegs was quite a sight, therefore, stumbling ! & round Lardtummy Lo-Watt, trying to defend himself
| | with the wobbly Mr Pointy while the dragon attached ' to his head squirmed and squealed in alarm,
and tried ;,. to get as far away as she could from the swords. | " There was a nasty moment when
Horrorcow became so alarmed that she actually put her paws over his eyes, so Fishlegs could not see
Lardtummy's . - lunges. | :I "Horrorcow! I can't see!" yelled Fishlegs. - ' "I'm so sorry," mooed
Horrorcow. "but it's all! o
```

Figure 13 TesseractOCR Book Reading

Both results provided clearer readings when using a black and white background with a clearer to use font. TesseractOCR showed the accuracy it can perform when used in this specific scenario, however from previous examples can struggle when reading things used in the outside world such as signposts and is where EasyOCR then shines.

A final set of tests were then performed, this time testing different handwritten texts with the use of pens, markers, and pencils.

The first library to be tested was EasyOCR, and it once again proves to be highly accurate in the use of an outside environment, and the results can be seen in figure 14.

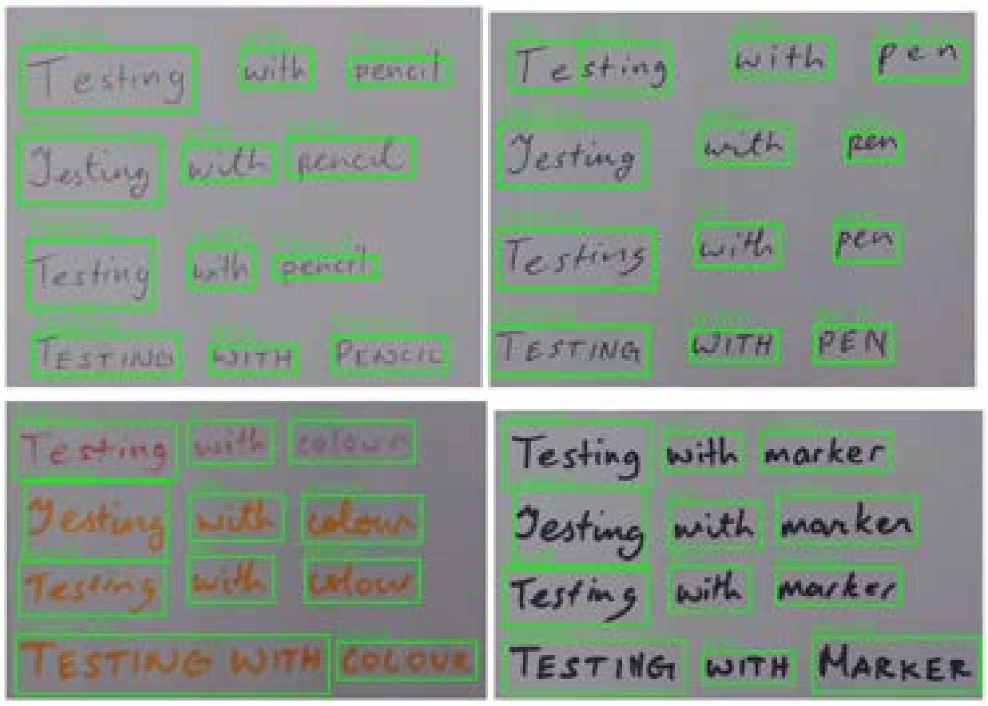
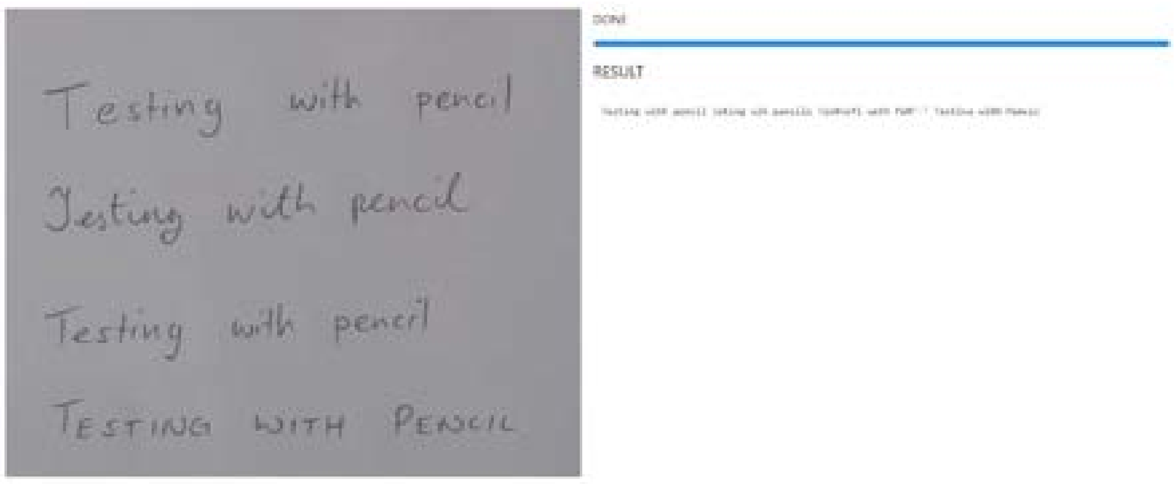


Figure 14 Handwritten EasyOCR Results

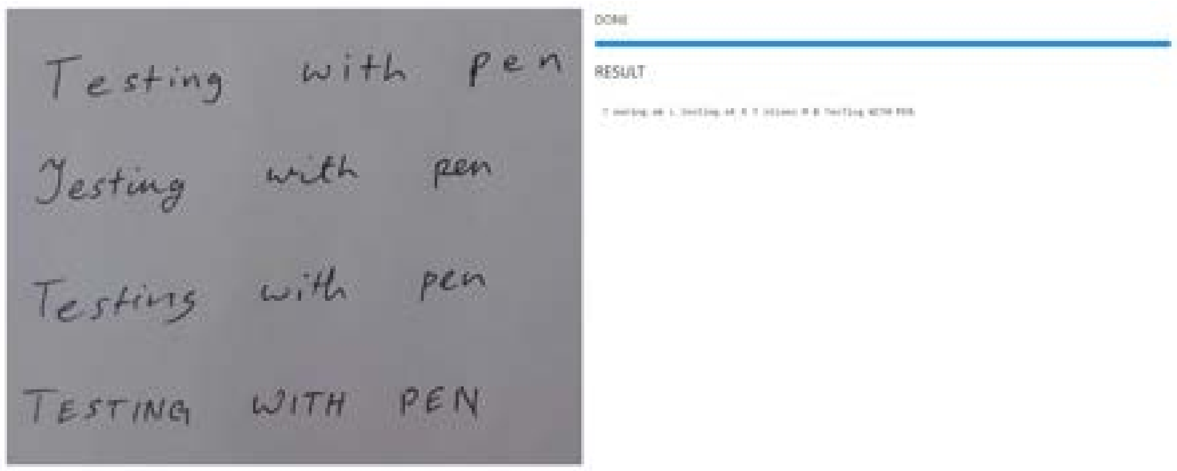
These results are to be expected with EasyOCR as it has shown throughout the testing phase that it is highly accurate, with some mistakes here and there.

TesseractOCR is then used, and again struggles to read some of the results, due to the font being hard to read, as it is handwritten text. Although it can occasionally read some words, it struggles to read handwritten texts and lettering, as it is not incredibly accurate, but can still read some of the presented letters. These results can be seen in figure 15, 16, 17 and 18.



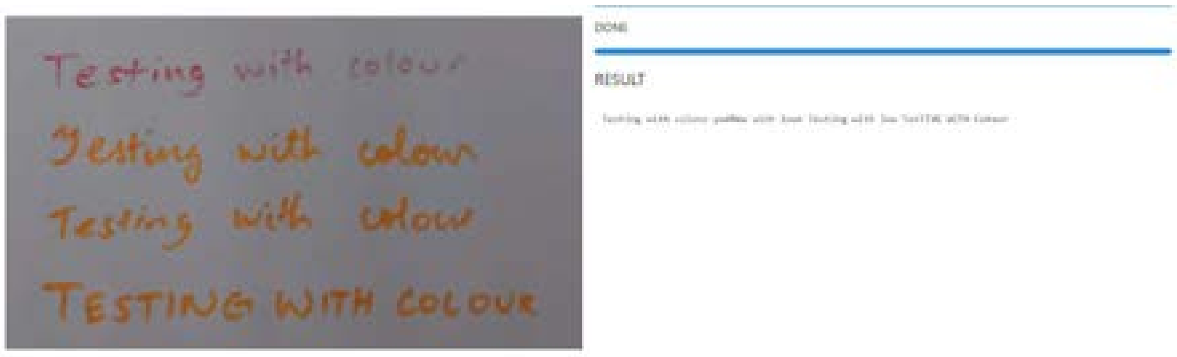
```
[illegible]
```

Figure 15 TesseractOCR Results for Handwritten Text with a Pencil



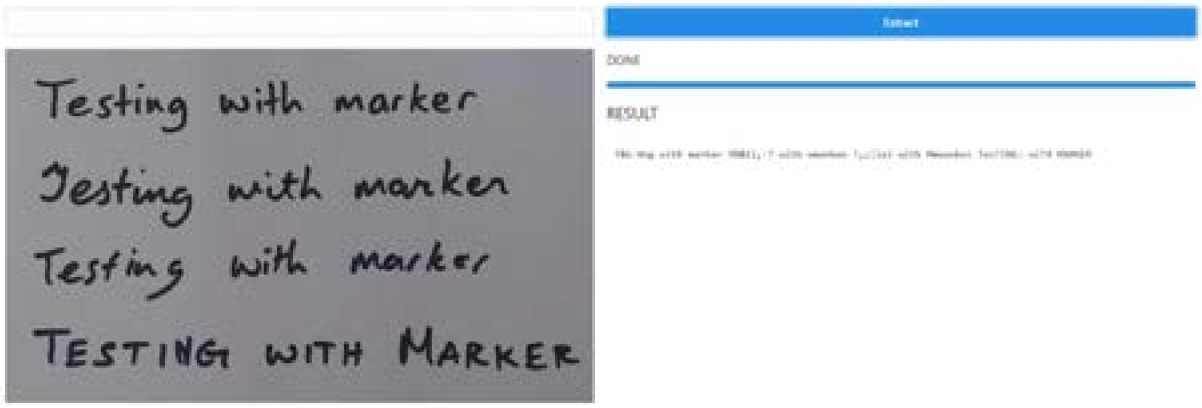
[illegible]

Figure 16 TesseractOCR Results for Handwritten Text with a Pen



[illegible]

Figure 17 TesseractOCR Results for Handwritten Text with a Coloured Marker



[illegible]

Figure 18 TesseractOCR Results for Handwritten Text with a Black Marker

2.5 Conclusion

From these tests, it’s clear that EasyOCR is the optimal library to use if factoring in accuracy, despite this the benefit of speed provided by TesseractOCR can be show in these tests, as it is far faster than EasyOCR and accurate when presented in the correct circumstances. Tesseract can also be used with React and JavaScript and can provide an easier development process than using EasyOCR.

2.6 Summary

In short, OCR is a character recognition system that can be used for computers to recognize characters from an image. There is a general algorithm that requires to go through the steps of image acquisition, pre-processing, segmentation, feature extraction, classification, and post-processing, where the general algorithm of OCR is discussed.

Different libraries can be used to make this process easier and follows most of the steps with the general algorithms, although would contain some differences. These libraries can work well with certain functionalities, for example Tesseract OCR works well with documents, where the image is generally black and white, while KerasOCR and EasyOCR works well with images with many parameters although slightly slower.

A comparison of Tesseract.js and EasyOCR was conducted to see the benefits of both libraries and concluded that it would be for the best to use Tesseract.js for the project.

In the UI and UX section of the report, the basic design principals were discussed. These consist of how a developer can design a website and good practices to follow when doing so, including contrast, hierarchy, colour, and structure, following a user-friendly design.

Requirements gathering was also discussed to help developers understand flaws in their design or discover the user’s needs and the different functionalities that the website will contain.

Responsive design was also covered for more compatibility options. Media queries, grid systems and fluid layouts are ways to allow for a website to be more compatible with more than one system, for example accessibility for mobile users.

Finally design systems were briefly covered, with their benefits and examples were given of good design systems.

Overall, the research project allowed for a better understanding and insight in the topic of OCR and discover methods of developing an OCR system for the web, with a brief insight to web-design and compatibility with different devices.

3 Requirements

3.1 Introduction

The purpose of the requirements phase is to allow for developers to work out what the application should be able to do. It is important to understand what the users would like the application to do rather than the developer deciding what is required.

The project area that will be looked at will be a progressive web application, with OCR functionality, and an adaptive UI that can allow for mobile compatibility.

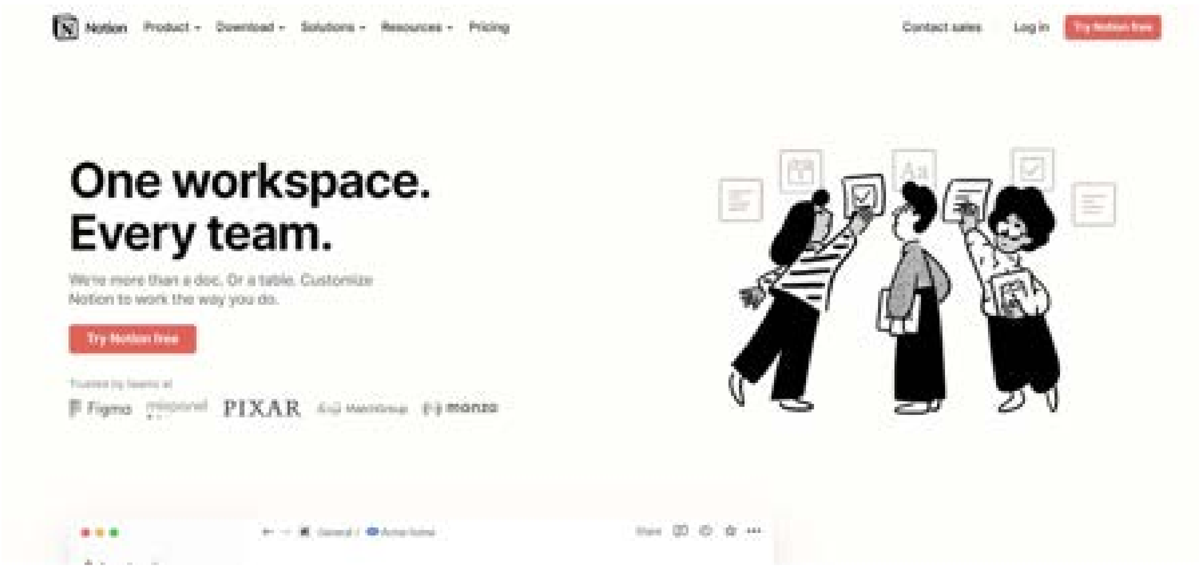
Several methods were used to gather information on how the product will be presented. This includes looking at similar applications, conducting interviews, surveys, and questionnaires to find the functional and non-functional requirements that the application will have, as well as gathering opinions on the design of the application.

3.2 Requirements gathering

3.2.1 Similar applications

The research of similar applications is essential in finding the features that the web-app will be using. When looking at similar applications, the advantages, disadvantages, and descriptions of each website will be examined. The first website that will be looked at is Notion.

3.2.1.1 Notion



We're more than a doc. Or a table. Customize Notion to work the way you do.

Figure 19 Notion Homepage

Fig 19 displays Notion, which is a productivity application that is designed to boost productivity. The application contains a list of many features that include: the organization of notes into different folders and sub-folders to organize their workspace, to-do lists to display different tasks, a note taking feature, where users can take notes of what they need. This note taking feature also allows for users to customize the font size, colour, background colour, etc. It is also possible to share notes with other users. It is also possible to add widgets to further customize and personalize the look and feel of their workspace.

Advantages of Notion:

- The application contains a detailed notes taking feature that allows users to edit notes and customize the look, such as the font size, the background colour, etc.
- The application has a clean look to it and is designed to be user friendly.
- A user can fully customize the folders and the folder structure, allowing for very detailed customization.
- The user can personalize these folders by adding images, icons, and names to them.
- Users can sort, filter, and add tags to the notes and folders that they are using.
- It is possible for a user to share their notes with other people.
- Users can add bullet points, headings, to-do lists, and even tables to their notes.
- Users can also upload images, links, files such as CVs, and widgets to the notes page that they are using.
- Users can search for a note or folder that they are looking for.
- Creating a new page is easy to do and can be immediately accessed at the bottom of the side-nav bar.
- You can import pages to use the structure of other users. This can have new widgets and features such as a clock in the page dashboard.
- When sharing a notion folder, it is possible for other users to comment on the page you are viewing.
- Users can collaborate with each other and create folders where an entire group can use them.
- Users can choose to have a certain folder to be private.
- When deleting a page, you can view the deleted pages in the bin folder.
- A help button is always displayed in the bottom right corner as a "?" sign.
- Notion has a mobile version that functions the exact same as the desktop version.

Disadvantages of Notion:

- The user interface is quite intimidating to learn when just starting to use the application, as it is filled with features that can put off new users.
- The images a user chose for their folder picture is only limited to images from Unsplash.com.
- There is no way to directly message other users in the team.

From an examination of this web app, the advantages of notion far outweigh its disadvantages and has many features that can prove to be beneficial to the web app that will be worked on. One major benefit of Notion is that it can also be used on a mobile application, and functions the exact same as the desktop version, although slightly slower. Similarly, we can see such benefits in the next application, which is going to be Google Keep.

3.2.1.2 Google Keep

Google Keep
Save your thoughts, wherever you are
Try Google Keep
Surprise party for Kristin!
Capture what's on your mind

Figure 20 Google Keep Homepage

Figure 20 shows the homepage for Google Keep. This is Google’s notes taking application, and to-do list, although focuses more on being a simple to-do list application, rather than a notes taking app. Google Keep also has a mobile version that functions similar to the desktop application. This is an easy and highly effective web app, that is user friendly and very easy to use.

Advantages of Google Keep:

- Google keep has an incredibly simple UI, and as a result is very easy to use.
- Users can use Google Keep as a notes taking application, or use it’s to-do lists featues.
- Users can organize and sort their notes by using tags to show which folder they belong to.
- The web app also uses a side-nav for easy access to all notes.
- It is easy to create a new note.
- A user can indent their to-do checkbox and have a multi-layered to-do list.
- Users can change the background of a note, and can also upload an image of their choice to use a background.
- Users can pin their notes, and functions similarly to a favourites feature. These pinned notes will be immideately brought to the top of the page.
- Users can archive, or delete notes.
- A users is also able to use a draw feature, where they can draw in the application.
- Users can also add a collaborator and share their notes with other users.

Disadvantages of Google Keep:

- Google Keep is primarily a to-do list app, and as a result does not have many features for the notes taking section of the application.

- Due to the simplicity of the app, the features are not as detailed as other applications.
- When writing long paragraphs while taking notes, it can be difficult to see them as it is hard to separate into headings and general paragraphs.
- The notes and does not have a way to make the screen larger when reading them.
- You cannot add sub-folders to further organize them.

Despite the disadvantages, Google Keep is an effective web app that acts as an effective to-do list tool. The indentation of checkboxes to further organize their tasks is incredibly effective and can prove to be a very useful feature when using it. It is primarily a to-do list app, hence why the notes taking features are somewhat lacking, but can prove to still be functional and effective.

The UI of Google Keep is also incredibly easy to use. The simplicity can entice users to use the web app over others that are more complex and confusing to use. A good balance of both notes taking and to-do features are also seen in Evernote, which will be the next application that we will be discussing.

3.2.1.3 Evernote

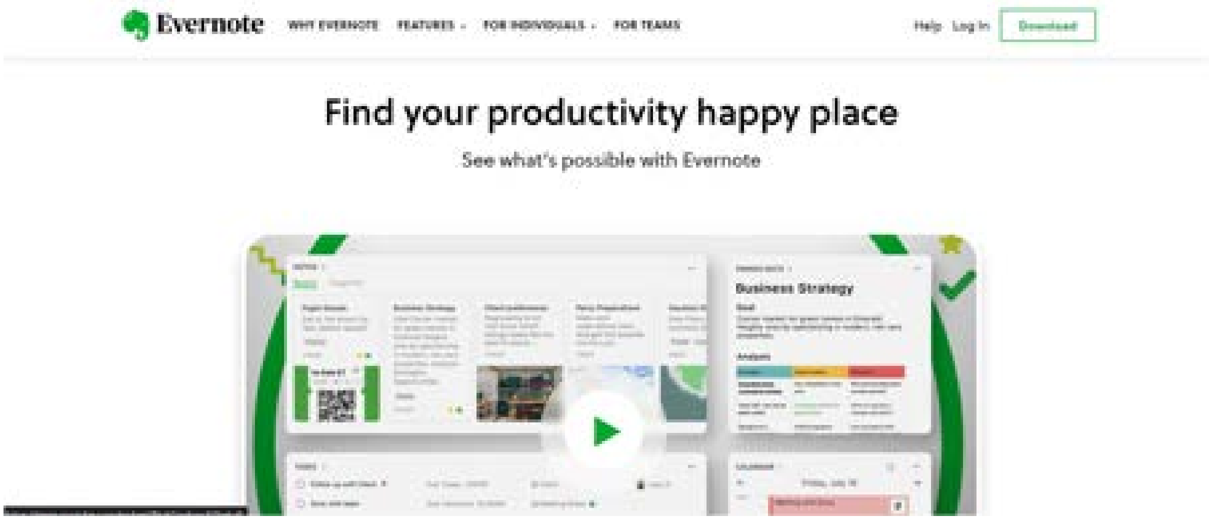


Figure 21 Evernote Homepage

In figure 21, evernote is another productivity app that is commonly used. It proves to be a highly effective app to take notes, and contains many useful featues that can allow for easy of use to the users.

<u>**Advantages of Evernote:**</u>

- Contains a draw feature, where users can draw an image, or for hand-written tasks.
- Evernote has a to-do list feature, where users can set custom due-dates, set the task to be re-occuring or set the due date to “today” or “tomorrow”.
- A user is able to add shortcuts to allow for quick access to their most important notes.
- A user can also organize them into folders, called “notebooks”.

- It is easy to navigate through the website, and all folders and notes are accessible through the side-nav.
- There is a “recently viewed” tab that can show the recently viewed.
- The notes display the time where it was last edited and viewed.
- A user can share their notes with other members as a shareable link or an invite through email.
- Users can add reminders of tasks they have to do that day.
- Users can duplicate a note that they have.
- Users can search for something they need in a specific note.
- Evernote has a version history, where a user can view each version of their notes.

Disadvantages of Evernote:

- Again, the notes taking features are not as elaborate and deailed when compared to something like Notion.
- Not every feature is availble in the free plan, for example the version history of the notes.

Some of the features listed above are locked by a premium plan, however despite this Evernote proves to be a very good balance of both applications, with complex enough features and an easy to use UI.

3.2.1.4 Text Scanner OCR

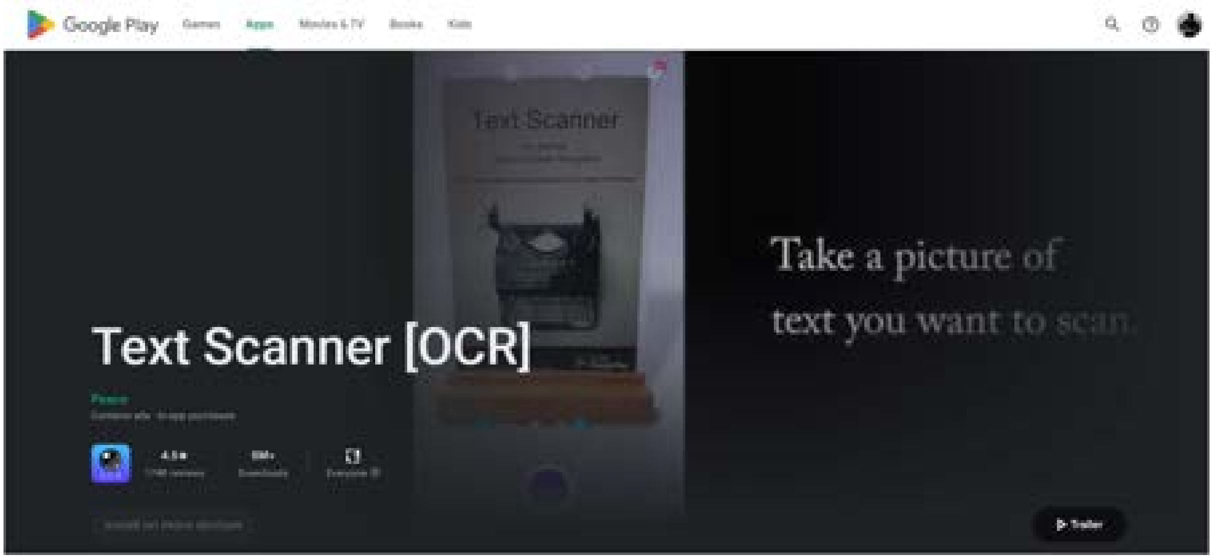


Figure 22 Text Scanner [OCR]

Figure 22 shows Text Scanner OCR, which is a mobile app used to extract text from images. It is a simple app and focuses on the OCR functionality.

Advantages of Text Scanner [OCR]

- Easy to use interface.
- Text extraction is fast.
- The returned results are accurate to the text presented.
- The app can access your gallery or take a picture with the camera.
- Settings can be configured to increase brightness or zoom in and out.

Disadvantages of Text Scanner [OCR]

- Contains too many ads.
- Cannot save results returned.
- No web version.
- Limited number of uses, as it can only be used ten times before needing to refresh tokens.

3.2.2 Interviews

Interviews were conducted towards a variety of users to gather information and data on the kind of people who would be using this application. It was also done to see their notes-taking habits and how frequently they would use notes taking applications, and similar tools. The interviews provided great insight into the different user's habits and provided some interesting features that they would have wished to see in the application.

The first interview conducted was with a 22-year-old student, who is currently studying in their final year of college. This interviewee normally has multiple ways of notes taking, ranging from flashcards to general computer notes, to notes on their phone. The interviewee also finds organizing of notes into folders to be slightly tedious and would prefer if they were more in-depth to allow for further organization.

When asked if the user enjoys making their notes look nice, the user says that they find it beneficial, although they don't normally spend too much time on it. They would use the basics such as bolding, italics, changing the font size to represent a title, etc.

The interviewee uses a flashcard app known as Anki, and in this app, the interviewee has several folders setup, one for college, one for their own personal development, one for language learning, and one for an archive. The most important folders are set to the top of the list, and in each folder, contains a set of folders including college material, modules, lectures, etc, with each one containing an individual card. A feature that Anki has is to allow for users to add tags to a card and organize those cards into tags to allow for easy sorting of these cards, as all you need to find them is to search.

In this application, the user mentions that it does not have an archive feature built in with the app and mentions that although it is not the most necessary, it would be nice to have that kind of feature in an application for a better ease of use.

When asked what their favourite features were from this application, they mentioned Anki, Notes on IOS, and Trello. In Anki, their loved how there was no paywall, and that it was feature-heavy while remaining free to use, and the gamification of the application, allowing them to constantly return to the application, as there is a daily streak feature that it has.

In the "Notes" app on IOS, they liked the straightforward UI that it has, the autoformat, the checklist, the ability to add checklists, draw, the fact that it automatically arranges by date, and that a user can share notes with other users, and lastly has a global search feature, which allows users to search for a certain thing, and the search feature will look through all the notes to find it.

Finally, the features that the interviewee loves in Trello consists of the checklists, and the priority sorting found in the application.

When asked if an OCR feature would be handy in a notes-taking application, the interviewee replies with "yes, but I would be quite concerned about its ability to read handwritten texts". They find that it would be handy for other users but note so much themselves as they tend to simply take notes on other devices.

Features they would like to see on a similar application being built would be the gamification of a system like this, spaced repetition, add folders / subfolders, and easily customizable, an archive feature and a global search feature, like the one in the Notes app, and finally a daily notes section.

The second interviewee is a 23-year-old graduate. The interviewee generally didn't really take notes, even during their time in college, they would often only listen to lectures and rarely take down notes, however when they do, they would always organize it, although it can be mentioned that they would potentially use a notes application in daily life outside of work reasons to bullet journal / consistently journal their tasks.

The interviewee used multiple notes taking applications, however Notion and CollaNote proved to be their personal favourite.

Initially, they originally found Notion to be confusing, with overwhelming UI as there are a lot of features that Notion provides its users and great variety in customization. CollaNote on the other hand was very easy to use and provided many features such as the possibility of exporting as a PDF file, a collection of different templates and designs. CollaNote also contains features such as public rooms, where anyone in the world can write things down, the possibility to insert pictures, draw and even add stickers too.

When asked about their favourite feature, the interviewee quickly brought up CollaNote again. This time discussing a feature that allowed documents to be turned into a PDF, and in this PDF the user is allowed to edit things and highlight things that are written in the PDF too, which the user found to be very helpful.

The next question the interviewee was prompted with was to see if they would make their notes look nice and decorate their notes. The interviewee only uses the simple tools, such as highlights, bold and italics, and bullet points.

The interviewee also thought that an OCR feature would be helpful in an application, as they were very fond of the PDF feature found in CollaNote.

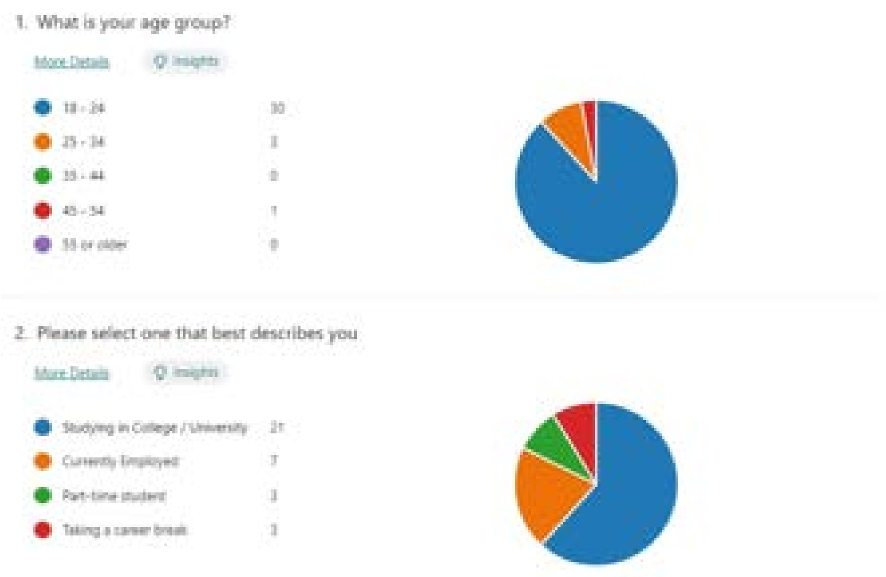
The interviewee is then asked for recommendations on features that they might like to see in a similar application being built, and they replied with a calendar, a section that allows a user to focus on certain tasks for the day, a schedule to make organization easier, that is like a timetable, and a to-do list widget or section in the application, to help with the productivity.

3.2.3 Survey

A questionnaire was carried out to determine the functional and non-functional requirements of the application. It was conducted to find out how users would take notes, and the methods they require to do so. This questionnaire also establishes what features and functionalities that user would like to see in the application, and ways to figure out how to improve upon the flaws found in similar applications already being used.

Microsoft Forms was used to create the questionnaire and was shared on various social media platforms such as discord, Instagram, and Facebook. The survey managed to reach up to 34 participants, with 23 out of the 30 participants consisting of 18 – 24 years old, and 3 participants being 25 – 34 years old. These participants were mostly in college / university, with 64% of them being in a fulltime course.

These results can be seen in the figure below:



1. What is your age group?

More Details   Insights

🔵	18 - 24	30
🟠	25 - 34	3
🟢	35 - 44	0
🔴	45 - 54	1
🟣	55 or older	0

2. Please select one that best describes you

More Details   Insights

🔵	Studying in College / University	21
🟠	Currently Employed	7
🟢	Part-time student	3
🔴	Taking a career break	3

Figure 23 Question 1 and 2 Results

In the next question, the user was asked to state how they feel about the following opinions and can be seen in the figure below:

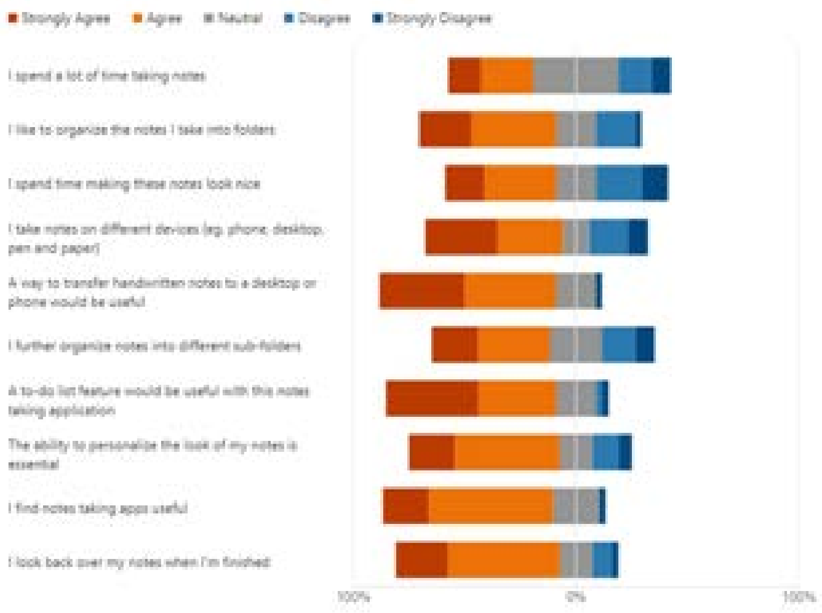
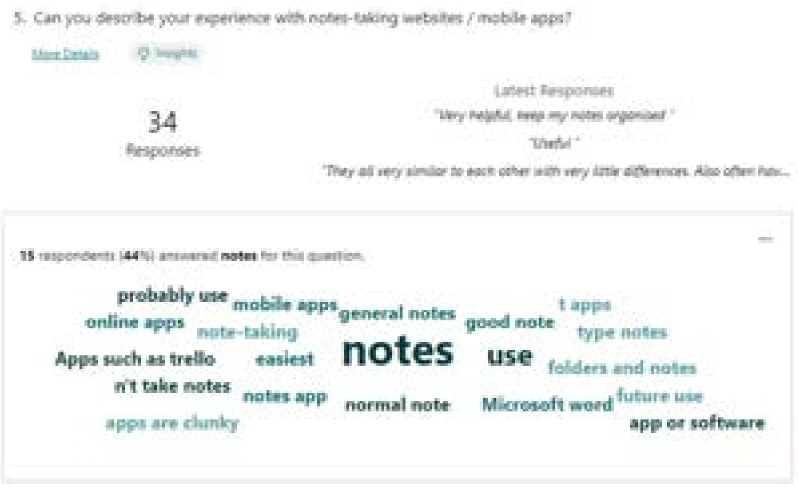


Figure 24 Question 4 results

From these results, we can gather information that most users answering the questionnaire find that organisation of their notes would be a great benefit towards the application. We can also see that most users would like to see a to-do list feature, personalization and giving them control over how the notes will look when looking through them.

With the user results being collected, it can be agreed upon that these features should be included upon the development of the application.

The next question then asks the user to describe their experiences with notes-taking websites / mobile apps and can be seen in the figure below.


5. Can you describe your experience with notes-taking websites / mobile apps?

More Details   Insights

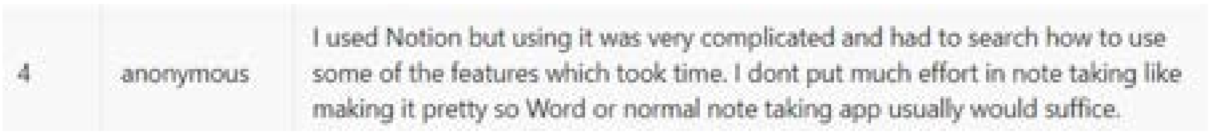
34
Responses

Latest Responses
"Very helpful, keep my notes organized "
"Useful "
"They all very similar to each other with very little differences. Also often hav...

15 respondents (44%) answered **notes** for this question.

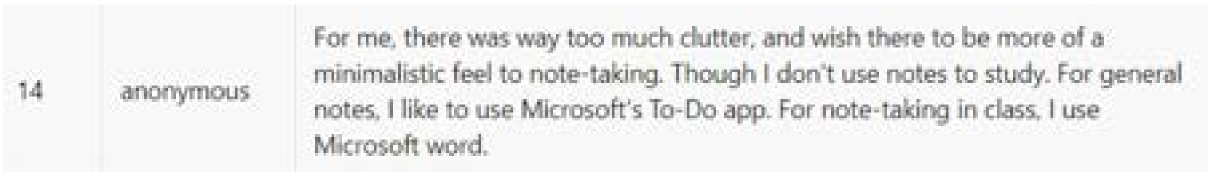
Figure 25 Question 5 results

From the results gathered, and reading through the answers, a lot of respondents would often organize notes into different folders to allow for ease of accessibility. Some also found that the UI of certain notes-taking websites were also a little complicated. E.g., a user found notion to be complicated to use, seen in the figures below.



4	anonymous	I used Notion but using it was very complicated and had to search how to use some of the features which took time. I dont put much effort in note taking like making it pretty so Word or normal note taking app usually would suffice.

Figure 26 User Results of Notion (1)

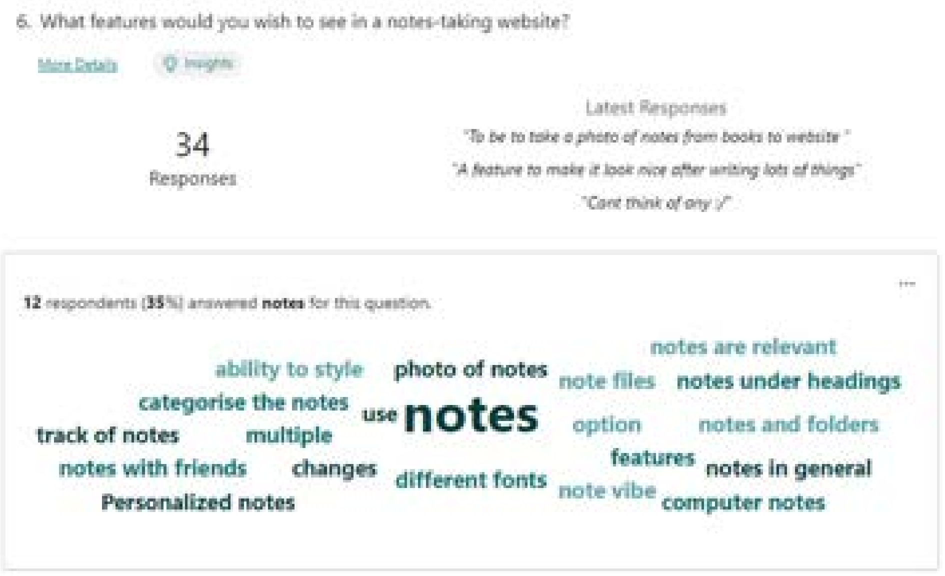


14	anonymous	For me, there was way too much clutter, and wish there to be more of a minimalistic feel to note-taking. Though I don't use notes to study. For general notes, I like to use Microsoft's To-Do app. For note-taking in class, I use Microsoft word.

Figure 27 User Results of Notion (2)

From these results, it should be noted that the app being developed should be easy to use and user friendly, as that was a common flaw found in most of the answers. It should also have an ability to save a user's notes so that they won't be lost.

The next question then asks what kinds of features a user would like to see in a notes-taking application, seen in fig 28.



6. What features would you wish to see in a notes-taking website?

More Details Insights

34
Responses

Latest Responses
"To be to take a photo of notes from books to website "
"A feature to make it look nice after writing lots of things"
"Cant think of any :/"

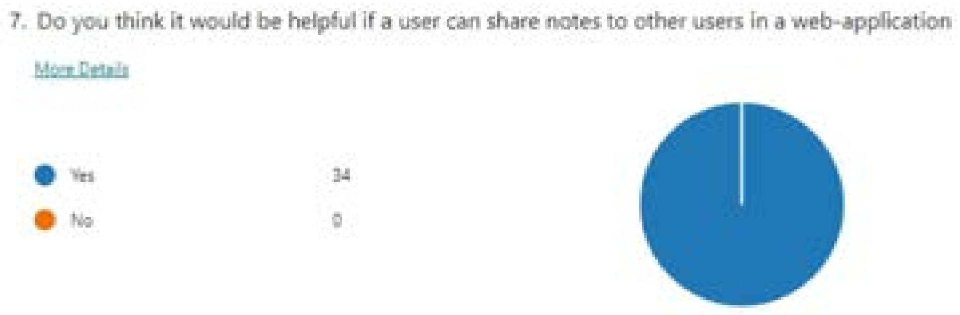
12 respondents (35%) answered **notes** for this question.

Figure 28 Potential Features

This question gave a great insight on different features that could be applied to the application. One of which included the ability to customize the look, adjusting different font sizes, bolding and italics, colours, inclusion of images into the file etc. to allow for customization.

Another set of ideas suggested is the ability to link different notes sections to each other, a to-do list feature, a "sort" feature, bullet points and numbers option, and a clean UI.

The user was then asked whether a “share” feature would be useful when using the application, and as seen in fig 29, 100% of users agree that it would be a great addition to the application.

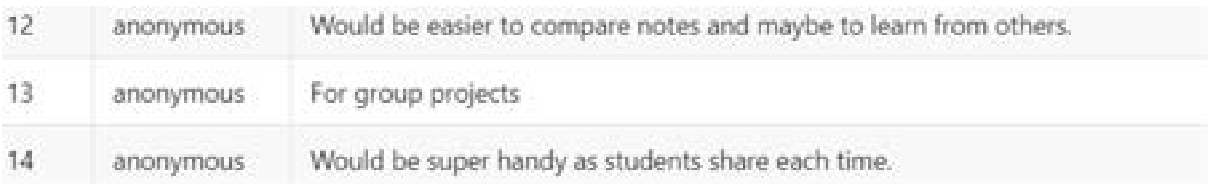

7. Do you think it would be helpful if a user can share notes to other users in a web-application

More Details

● Yes	34
● No	0

Figure 29 Question 7

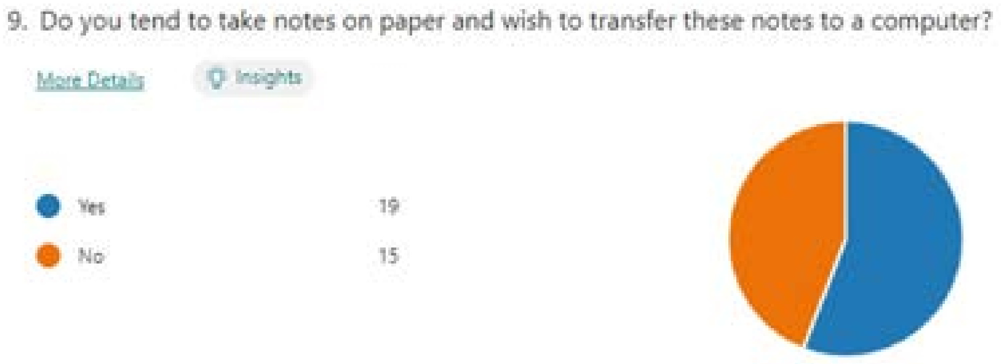
Followed up by this question is question 8, where the user is asked why they think it would be a useful feature, to allow for teamwork and co-operation with other users. Potentially friends and classmates. These can be seen in fig 30.



12	anonymous	Would be easier to compare notes and maybe to learn from others.
13	anonymous	For group projects
14	anonymous	Would be super handy as students share each time.

Figure 30 Question 8

Question 9 then asks whether a user would take notes on paper and wishes to transfer them to the desktop device, seen in the figure below.

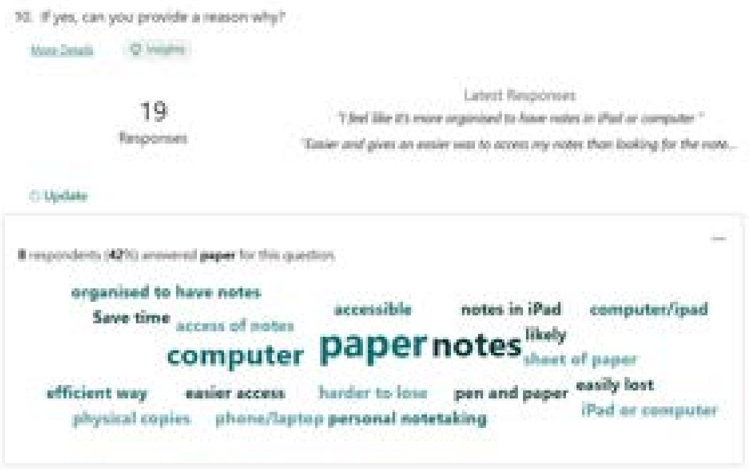

9. Do you tend to take notes on paper and wish to transfer these notes to a computer?

● Yes	19
● No	15

Figure 31 Question 9

Around 56% of users answered “yes”, showing that potentially developing OCR functionality could be beneficial to the application, as more than half the users could potentially use this functionality.

Question 10 then asks why the user would find it to be a helpful feature, seen in figure 32.



10. If yes, can you provide a reason why?

More Details Insights

19 Responses

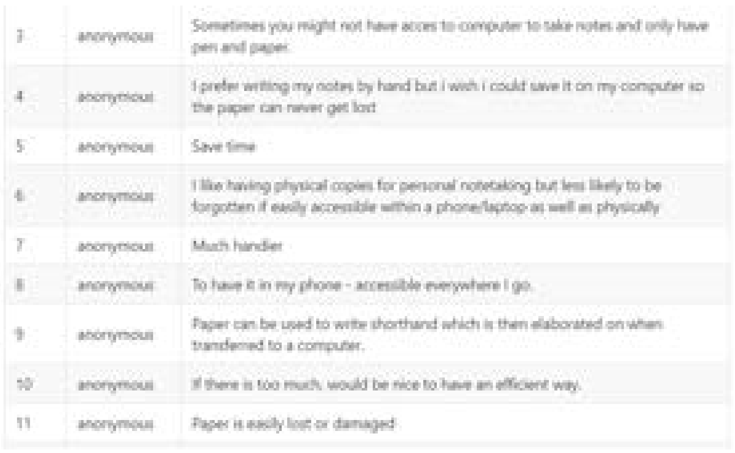
Latest Responses
"I feel like it's more organised to have notes in iPad or computer"
"Easier and gives an easier was to access my notes than looking for the note..."

Update

8 respondents (42%) answered paper for this question.

Figure 32 Question 10

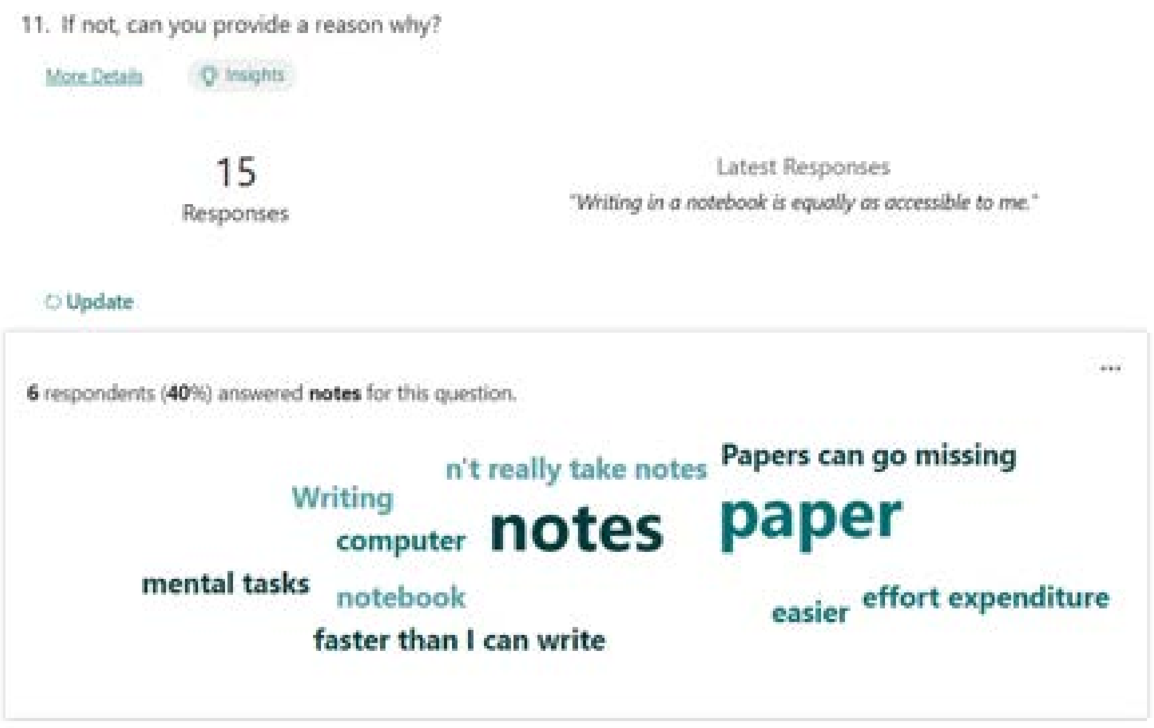
From the answers seen, those who answered “yes” state that paper can be easily destroyed, and that it would be more convient to have it saved on a device such as a mobile phone, to have easier access to those notes, as well as for organization purpses. This can be seen in figure 33.



3	anonymous	Sometimes you might not have acces to computer to take notes and only have pen and paper.
4	anonymous	I prefer writing my notes by hand but i wish i could save it on my computer so the paper can never get lost
5	anonymous	Save time
6	anonymous	I like having physical copies for personal notetaking but less likely to be forgotten if easily accessible within a phone/laptop as well as physically
7	anonymous	Much handier
8	anonymous	To have it in my phone - accessible everywhere I go.
9	anonymous	Paper can be used to write shorthand which is then elaborated on when transferred to a computer.
10	anonymous	If there is too much, would be nice to have an efficient way.
11	anonymous	Paper is easily lost or damaged

Figure 33 Respondents of Question 9

The respondents who answered “no” to question 9 were also asked for their reasoning behind their answer in question 11. Seen in figure 34.


11. If not, can you provide a reason why?

More Details Insights

15 Responses

Latest Responses
"Writing in a notebook is equally as accessible to me."

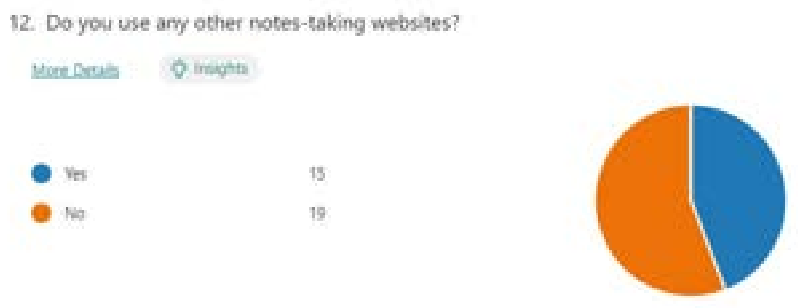
Update

6 respondents (40%) answered notes for this question.

Figure 34 Question 11

Where most users state that they already take notes on their computer, and don't necessarily need to transfer, however it is also important to note that there are users who simply prefer notes-taking on paper, as they find it easier to retain information that way rather than through desktop.

Question 12 asks if the participant uses any notes taking websites / mobile applications, seen in figure 35. We can see that there are more users who answered "no", with 44% answering "yes".

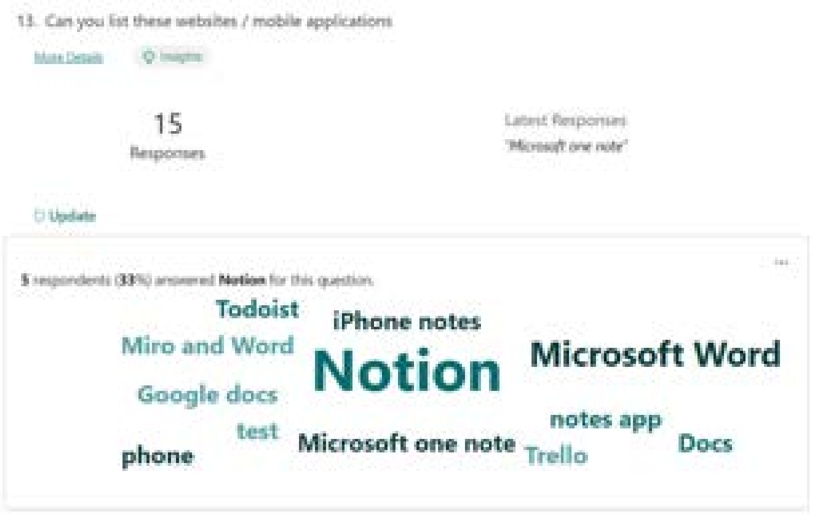

12. Do you use any other notes-taking websites?

More Details Insights

● Yes	15
● No	19

Figure 35 Question 12

Question 13 then branches, and first asks the users who answered "yes" to list off the applications that they use, seen in figure 36.


13. Can you list these websites / mobile applications

More Details · Insights

15
Responses

Latest Responses
"Microsoft one note"

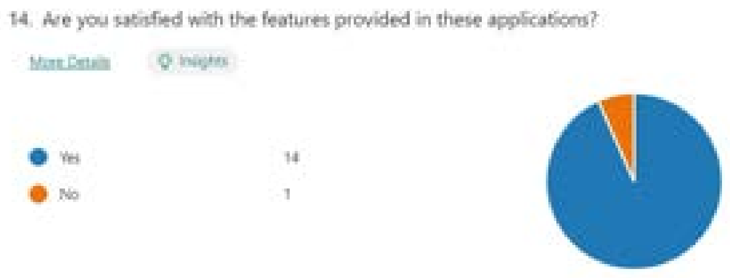
Update

5 respondents (33%) answered Notion for this question.

Figure 36 Question 13

The users answered mostly with Notion, Microsoft Word and One Note, Google Docs, and Trello. These applications are very commonly used and contain functionality that can provide to be useful when taking notes.

Question 14 asks users if they are satisfied with the functionalities that are in these applications, with most users answering with "yes". This can be seen in the figure below.

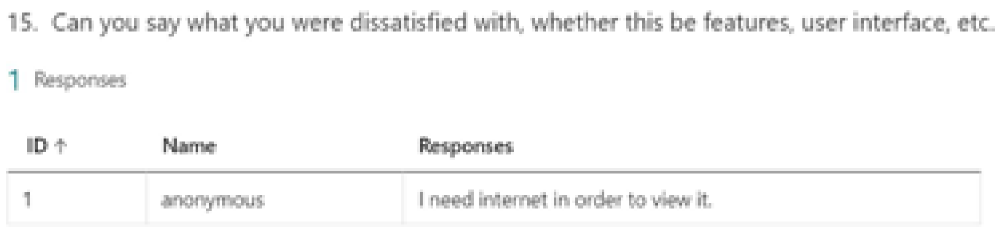

14. Are you satisfied with the features provided in these applications?

More Details · Insights

●	Yes	14
●	No	1

Figure 37 Question 14

Question 15 asks users what they were dissatisfied with, only retrieving one answer where the users "needs internet to use them". Seen in the figure 38.

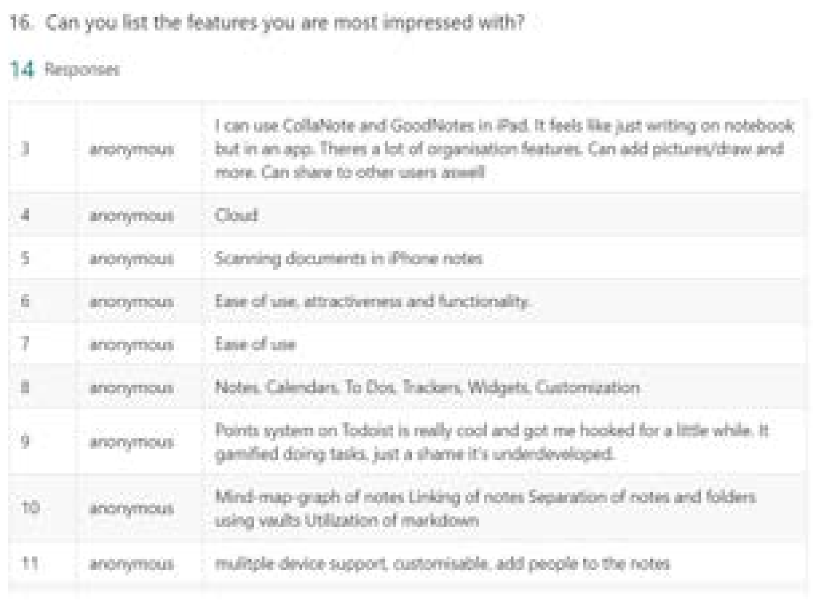

15. Can you say what you were dissatisfied with, whether this be features, user interface, etc.

1 Responses

ID ↑	Name	Responses
1	anonymous	I need internet in order to view it.

Figure 38 Question 15

Question 16 then asks users what they were most satisfied with, and the answers range from cloud accessibility, support for multiple devices, ease of use, and allowing for the notes to look nice. Some of these responses can be seen in figure 39.

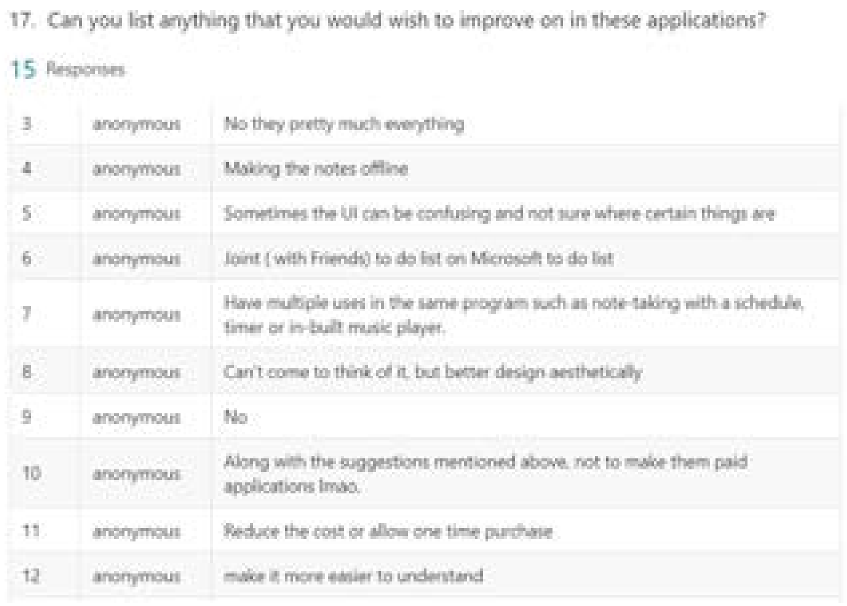

16. Can you list the features you are most impressed with?

14 Responses

3	anonymous	I can use CollaNote and GoodNotes in iPad. It feels like just writing on notebook but in an app. Theres a lot of organisation features. Can add pictures/draw and more. Can share to other users aswell
4	anonymous	Cloud
5	anonymous	Scanning documents in iPhone notes
6	anonymous	Ease of use, attractiveness and functionality.
7	anonymous	Ease of use
8	anonymous	Notes, Calendars, To Dos, Trackers, Widgets, Customization
9	anonymous	Points system on Todoist is really cool and got me hooked for a little while. It gamified doing tasks, just a shame it's underdeveloped.
10	anonymous	Mind-map-graph of notes Linking of notes Separation of notes and folders using vaults Utilization of markdown
11	anonymous	mulitple device support, customisable, add people to the notes

Figure 39 Question 16

Question 17 then asks if there are any features that the user wishes to be improved upon, and a common answer was to make the UI simpler to understand and less cluttered. Some also mention to have more functionality such as a built-in timer and music player, the ability to make notes offline and allowing users to attach files, seen in figure 40.


17. Can you list anything that you would wish to improve on in these applications?

15 Responses

3	anonymous	No they pretty much everything
4	anonymous	Making the notes offline
5	anonymous	Sometimes the UI can be confusing and not sure where certain things are
6	anonymous	Joint (with Friends) to do list on Microsoft to do list
7	anonymous	Have multiple uses in the same program such as note-taking with a schedule, timer or in-built music player.
8	anonymous	Can't come to think of it, but better design aesthetically
9	anonymous	No
10	anonymous	Along with the suggestions mentioned above, not to make them paid applications lmao.
11	anonymous	Reduce the cost or allow one time purchase
12	anonymous	make it more easier to understand

Figure 40 Question 17

From the results gathered in the surveys, we have a better understanding of the types of functionalities that the user can be expecting from an application such as this one, and the UI should also be easy to use and easy to understand.

3.3 Requirements modelling

3.3.1 Personas

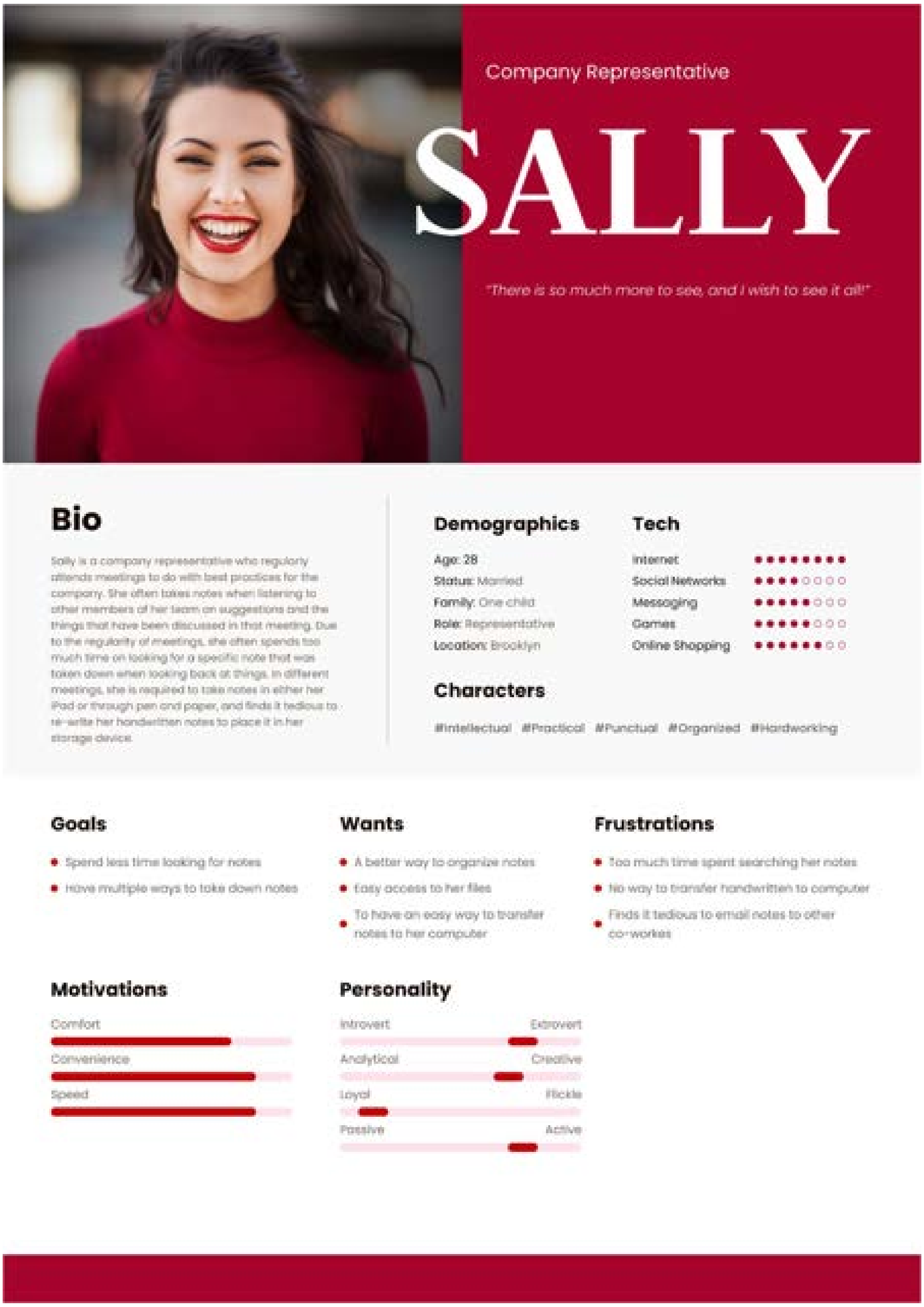
Personas are fictional characters that are made to help developers understand what the user’s needs are. These characters also help in identifying the target audience that the developers are aiming for.

The data collected from the survey and interviews help with the creation of these user personas. These personas contain the goals of that character, and their frustrations in what they are currently working on.

The personas created are:

- Sally
- Anya
- Denji

The personas can be seen in more detail in figure 41, 42, and 43.



Company Representative

SALLY

"There is so much more to see, and I wish to see it all!"

Bio

Sally is a company representative who regularly attends meetings to do with best practices for the company. She often takes notes when listening to other members of her team on suggestions and the things that have been discussed in that meeting. Due to the regularity of meetings, she often spends too much time on looking for a specific note that was taken down when looking back at things. In different meetings, she is required to take notes in either her iPad or through pen and paper, and finds it tedious to re-write her handwritten notes to place it in her storage device.

Demographics

Age: 28
Status: Married
Family: One child
Role: Representative
Location: Brooklyn

Tech

Internet	●●●●●●●●
Social Networks	●●●●○○○○
Messaging	●●●●●○○○
Games	●●●●●○○○
Online Shopping	●●●●●●○○

Characters

#Intellectual #Practical #Punctual #Organized #Hardworking

Goals

- Spend less time looking for notes
- Have multiple ways to take down notes

Wants

- A better way to organize notes
- Easy access to her files
- To have an easy way to transfer notes to her computer

Frustrations

- Too much time spent searching her notes
- No way to transfer handwritten to computer
- Finds it tedious to email notes to other co-workes

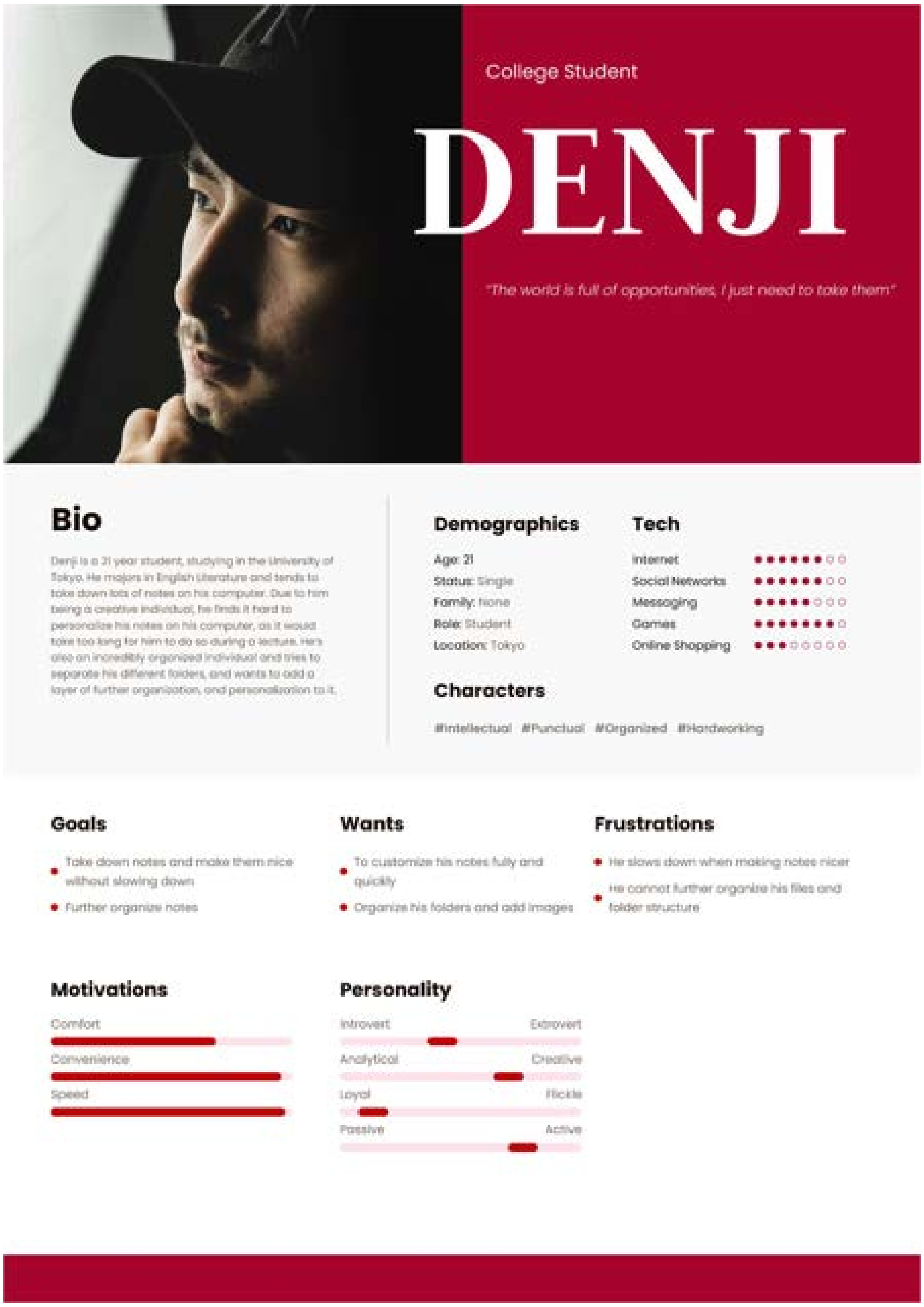
Motivations

Comfort
Convenience
Speed

Personality

Introvert — Extrovert
Analytical — Creative
Loyal — Fickle
Passive — Active

Figure 41 Persona 1



College Student

DENJI

"The world is full of opportunities, I just need to take them"

Bio

Denji is a 21 year student, studying in the University of Tokyo. He majors in English Literature and tends to take down lots of notes on his computer. Due to him being a creative individual, he finds it hard to personalize his notes on his computer, as it would take too long for him to do so during a lecture. He's also an incredibly organized individual and tries to separate his different folders, and wants to add a layer of further organization, and personalization to it.

Demographics

Age: 21
Status: Single
Family: None
Role: Student
Location: Tokyo

Tech

Internet	●●●●●●○○
Social Networks	●●●●●●○○
Messaging	●●●●●○○○
Games	●●●●●●●○
Online Shopping	●●●○○○○○

Characters

#Intellectual #Punctual #Organized #Hardworking

Goals

- Take down notes and make them nice without slowing down
- Further organize notes

Wants

- To customize his notes fully and quickly
- Organize his folders and add images

Frustrations

- He slows down when making notes nicer
- He cannot further organize his files and folder structure

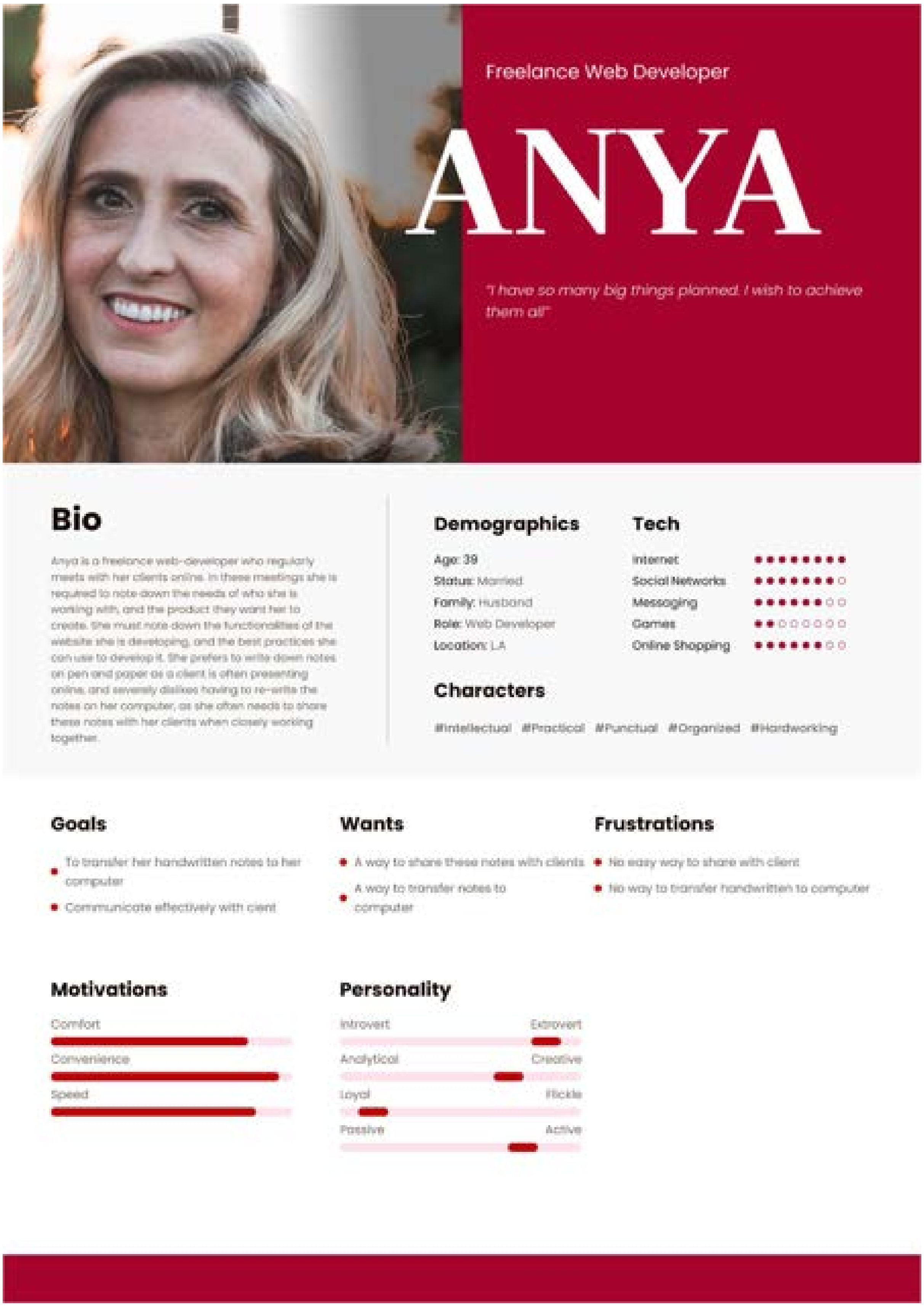
Motivations

Comfort
Convenience
Speed

Personality

Introvert — Extrovert
Analytical — Creative
Loyal — Fickle
Passive — Active

Figure 42 Persona 2



Freelance Web Developer

ANYA

"I have so many big things planned. I wish to achieve them all"

Bio

Anya is a freelance web-developer who regularly meets with her clients online. In these meetings she is required to note down the needs of who she is working with, and the product they want her to create. She must note down the functionalities of the website she is developing, and the best practices she can use to develop it. She prefers to write down notes on pen and paper as a client is often presenting online, and severely dislikes having to re-write the notes on her computer, as she often needs to share these notes with her clients when closely working together.

Demographics

Age:	39
Status:	Married
Family:	Husband
Role:	Web Developer
Location:	LA

Tech

Internet	●●●●●●●●●
Social Networks	●●●●●●●●○
Messaging	●●●●●●●○○
Games	●●○○○○○○○
Online Shopping	●●●●●●●○○

Characters

#Intellectual #Practical #Punctual #Organized #Hardworking

Goals

- To transfer her handwritten notes to her computer
- Communicate effectively with client

Wants

- A way to share these notes with clients
- A way to transfer notes to computer

Frustrations

- No easy way to share with client
- No way to transfer handwritten to computer

Motivations

Comfort

Convenience

Speed

Personality

Introvert — Extrovert

Analytical — Creative

Loyal — Fickle

Passive — Active

Figure 43 Persona 3

3.3.2 Functional requirements

The functional requirements contain a list of functions that users should be able to do in the application. The list contains a set of requirements that range from high priority to low priority.

#	Functional Requirements	Priority
1	Users will be able to sign up or log into the application	High
2	Users should be able to create a notes folder	High
3	Users should be able to edit a notes folder	High
4	Users should be able to delete a notes folder	High
5	Users should be able view notes folders	High
6	Users should be able to create a new note	High
7	Users should be able to update notes	High
8	Users should be able to view all notes in a specific folder	High
9	Users should be able to delete notes in a specific folder	High
10	Users should be able to open notes	High
11	Users should be able to access OCR functionality	High
12	Users should be able to upload an image and read using OCR	High
13	Notes can auto-save	High
14	Users should be able to save an image to a specific note	Medium
15	Users should be able to change their email	Medium
16	Users should be able to change their password	Medium
17	Users should be able to change their name	Medium
18	An information icon to show how to use the website should be available	Low
19	Users should be able to verify password	Low

3.3.3 Non-functional requirements

These are requirements which if not met do not stop the application from working, but which mean that the application is not working as well as it should. They are usually based on issues such as:

- Usability
- Performance
- Security

#	Non-functionality Requirements	Type
1	The application should be compatible to mobile screen sizes	Usability
2	The application should work in multiple browsers	Usability
3	The application should have information of how to use some parts of the website	Usability
4	Users should be able to navigate the website easily	Usability
5	The application should be user friendly and accessible	Usability
6	Users can access all their notes through a side-nav	Usability
7	Users will not be able to view another user’s account	Security
8	User password will require at least 8 characters and contain a number	Security
9	User password will be encrypted	Security

3.3.4 Use Case Diagrams

There are two main users of the web application. These consists of visitors and registered users. The use-case diagram is meant to represent what the users can do once they access the website. The use-case diagram for the application and its potential users can be seen in figure 44, and fig 45.

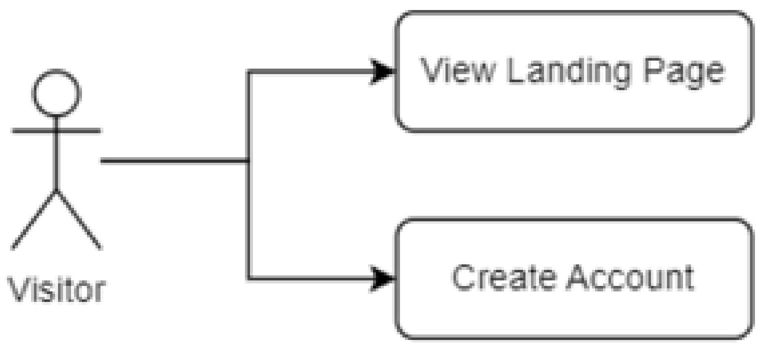


Figure 44 Visitor Use Case Diagram

Visitors:

- A visitor can create an account.
- A visitor can view the homepage and sign-in / login page.

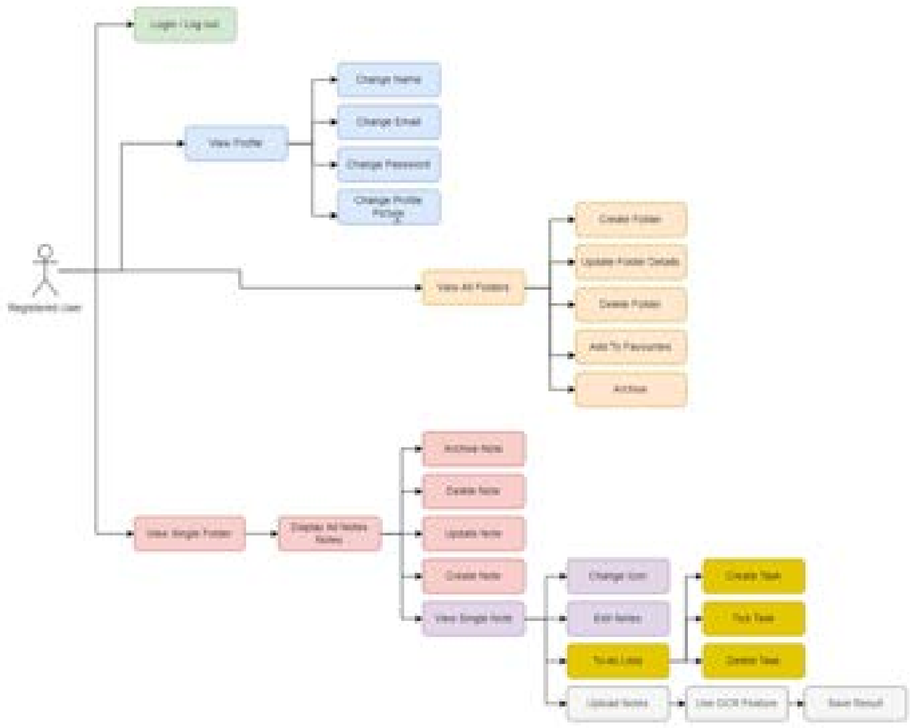


Figure 45 User Use Case Diagram

Registered Users:

- A user can log in and log out of the application.
- A user can view all their notes folders.
- A user can create a new notes folder.
- A user can edit a notes folder.
- A user can delete a notes folder.
- A user can open each notes folder.
- A user can view all notes in that folder.
- A user can create a new note.
- A user can edit a note.
- A user can delete a note.
- A user can access the note and edit its contents.
- A user can change the notes font size.
- A user can change the colour of the note font.
- A user can add italics and bold to a note font.
- A user can add a checklist in the note.
- A user can access the OCR feature.
- A user can use the OCR feature to upload an image and read the results from that image.
- A user can discard or save the results to another file.
- A user can add a note to favourites.
- A user can access all notes from the sidebar.
- A user can access notes from their mobile device.

- A user can save their notes through autosave.
- A user can change their email address.
- A user can change their password.
- A user can change their profile picture.

3.4 Feasibility

The feasibility section looks over the different technologies that are to be used together in the development of the application and looks at any technical issues that can arise during development, e.g., compatibility issues.

The application will be developed using the MERN stack, and because of this, the technologies that are to be used are very commonly used together.

The technologies used to develop this application consists of:

- React.js which will provide the frontend of the application.
- Express.js which will be the node.js framework.
- MongoDB which will provide the database.
- Node.js will act as the web server.

These technologies should be very compatible with each other, and technical issues shouldn't occur when working with this stack because of this.

Due to mobile compatibility being an important feature from the data gathered in the interviews, the application should contain this feature. Mobile compatibility should be accomplishable with the use of React Native to allow for this feature to be implemented. Although due to time factors, it is possible that only a barebones version of this feature will be accomplished, as other more important features are being developed.

Tesseract.js will allow the project to contain OCR functionalities. This library was studied and compared with Easy OCR to see which library can be most effective in a web-application in a previous section of this document. Due to Tesseract.js being a JavaScript library, implementation of this library will be feasible and compatible with the other technologies used in this project.

3.5 Conclusion

In this chapter, the developers are given a better understanding of what features are to be seen in the application, as well as examining user's needs before starting development and design of the app.

This was done through several methods, including surveys and questionnaires, interviews, and researching similar applications to gather the functional and non-functional features that the application should have.

The survey conducted provided information on what users find useful about notes-taking apps, and to gather general information about a user’s opinion of these apps. This ranged from what they found to be advantageous, what they found to be a disadvantage, etc.

Interviews were then conducted to gather a better insight of a user’s opinions of these applications and gather further information on what they would like to see more of.

The list of functional and non-functional requirements was then listed out, showing the functionalities that the application should contain, ranging from high to low priority requirements, where non-functional requirements showed what extra features the web application can have that will improve the application.

A use-case diagram was also created to show how different users were supposed to interact with the web application, and what they can access as a visitor, and as a user.

4 Design

4.1 Introduction

The chapter is to show the development of the design of the application. The design section covers the UI of how the app will look, covering the style guides, typography, wireframes etc, allowing for the developer to view the finish product and where the functionalities will be placed. It also covers the program design, which covers the technologies used and the structure of the code, as well as the database design of the application.

The application that is going to be developed is going to be a web app, that will be developed using the MERN stack. It will act as a text editing application where a user will be able to store notes into folders and take down and save notes. Users will also be able to upload screenshots of notes with text on them and read the text on that note using OCR.

4.2 Program Design

The program design section is essential to make the programming and coding of the application to be better planned and more straightforward, following potential patterns when structuring the code and API.

4.2.1 Technologies

The technologies that will be used to develop this application will be:

- MongoDB
- Mongoose
- Express.js
- React
- Node.js
- HTML
- CSS
- Axios
- AWS

These technologies were chosen because they generally work well with each other. This functionality helps with the compatibility of the application and can help minimize issues when developing the app.

React

React.js is a JavaScript framework that is used to develop frontend web-applications. It is commonly used to quickly develop the frontend of a website and would take much less code than it would when using vanilla JavaScript. (David, 2022)

React was chosen for the reasons listed above, as you can create components that can be reused throughout the website, providing a potentially quick development of the UI of the website.

<u>MongoDB</u>

MongoDB will provide the database of the application. The database will be in a JSON-like format and as it is a NoSQL database, the fields and data structures of the database can be changed and provide scalability (What Is MongoDB?, 2022). MongoDB is also easy to work with and is highly compatible with the other technologies that will be used to develop this application, one of which is Axios, which is what will be used to call data from the API.

<u>Express.js</u>

Express.js will be covering the backend of the application. The backend API will be written in Express.js, as it is a web-application framework that allows developers to build RESTful API's (Express - Node.js web application framework, 2017). It is flexible and lightweight allowing for quick development and is compatible with the technologies available in this application.

<u>Mongoose</u>

Mongoose is an ODM (Object Data Modelling) library that is used for MongoDB and Node.js. It Is used to manage the relationships between data, provides validation for the schema, and can act as a translator of code for MongoDB. (Karnik, 2018)

It will act as a translator between React.js, Express and the MongoDB, to translate the code and its representation in MongoDB.

<u>Node.js</u>

Node is an asynchronous JavaScript runtime engine and is used to develop scalable network apps (Node.js, 2023). It will act as the server of the application.

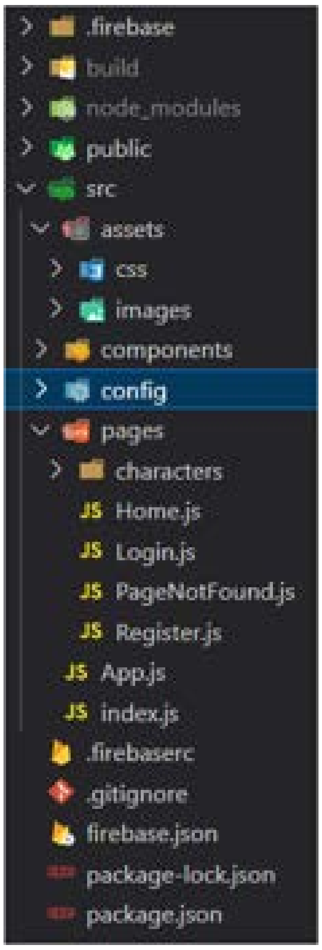
<u>AWS</u>

AWS will allow developers to store the images, with the use of AWS S3 Buckets. This is because MongoDB cannot store images and will need to be uploaded to a different method of storage. In this case, it is S3 buckets.

Another set of technologies that could be used is the Django Framework, although this would have been ideal when developing the OCR application, the time it would have taken to become comfortable with learning Python and the framework would have been too long, and setback progress on the application.

4.2.2 Structure of React

The structure of React consist of different folders, used for different purposes to keep it organized. This section will be using the folder structure of a previous project and can be seen in fig 46.



```
> .firebase
> build
> node_modules
> public
v src
  v assets
    > css
    > images
  > components
  > config
  v pages
    > characters
      Home.js
      Login.js
      PageNotFound.js
      Register.js
    App.js
    index.js
  .firebaserc
  .gitignore
  firebase.json
  package-lock.json
  package.json
```

Figure 46 React Folder Structure

<u>Root Directory:</u>

The root directory consists of all the folders that was used in the previous application. This included the deployment, the build, node modules, public folder, packages, etc.

<u>SRC Directory:</u>

The SRC directory contains the many folders used to develop the application. It is where the code is held, and contains the code and where the different components, assets, pages, etc. are held.

<u>Assets Directory:</u>

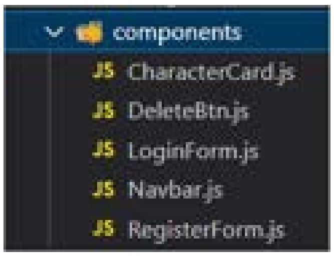
The assets directory contains the images that are used throughout the program, and general assets that can also be considered. It also contains the styling files, where the global styles can be stored.

<u>Layouts Folder:</u>

Although not available in Fig 46. the layouts folder is where it contains layouts for the entire project. For example, it stores headers, footers, nav-bars, etc.

Components Folder:

The components folder is where the components used throughout the entire application is stored. Using a pervious project as an example, figure 47 shows what is inside the components folder.



```
components
    CharacterCard.js
    DeleteBtn.js
    LoginForm.js
    Navbar.js
    RegisterForm.js
```

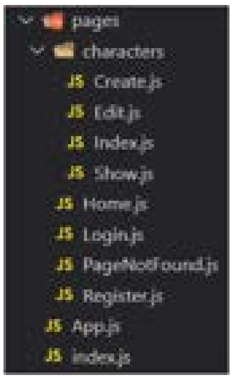
Figure 47 React Components

As seen in the fig 47, the different functionalities are stored in this folder, and contains components such as cards, buttons, and forms. These components are globally available throughout the entire application and can be called and used in the different pages available.

Config Folder:

The config folder contains a set of configuration files, that can be used to store environment variables. These variables are accessed throughout the entire application.

Pages Folder:



```
pages
  characters
    Create.js
    Edit.js
    Index.js
    Show.js
  Home.js
  Login.js
  PageNotFound.js
  Register.js
App.js
index.js
```

Figure 48 React Pages

Figure 48. shows the pages folder, which contains all the pages that the application has. Within this folder is a sub-folder that organizes the characters page towards their components. These pages normally contain components grouped into one.

App.js:

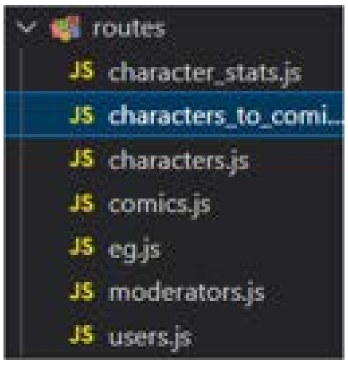
App.js is used to display the application. It contains all the folders, routes, pages, and components within the application.

Index.js:

Index.js displays the app.js file.

Routes Folder:

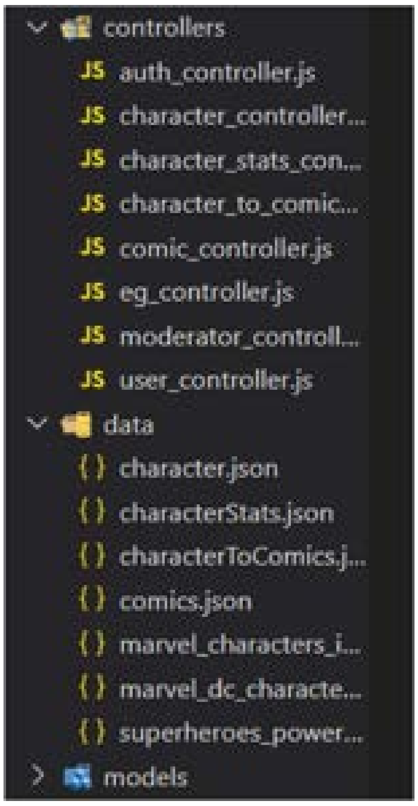
This folder contains the many routes of the application, ranging from private and protected routes, and all other types. It can be seen in the figure below.



```
routes
  character_stats.js
  characters_to_comi...
  characters.js
  comics.js
  eg.js
  moderators.js
  users.js
```

Figure 49 Routes Folder

The backend of the application also contains the “controller”, the “data”, and the “models” folder, which is used to store the schema, controllers and data of the application.



```
controllers
  auth_controller.js
  character_controller...
  character_stats_con...
  character_to_comic...
  comic_controller.js
  eg_controller.js
  moderator_controll...
  user_controller.js
data
  character.json
  characterStats.json
  characterToComics.j...
  comics.json
  marvel_characters_i...
  marvel_dc_characte...
  superheroes_power...
models
```

Figure 50 Controllers Folder

Controllers Folder:

The Controllers folder holds the CRUD functionalities of a specific object in the application, e.g., CRUD of a specific character in the case of the previous application.

Models Folder:

The Models folder holds the schema of the application, and how this object will be stored in the database, containing the attributes of e.g., a character for example.

Utils Folder:

Finally, the Utils folder contains the utilities used throughout the application. E.g., the database

4.2.3 Model View Controller Design Pattern

The Model-View-Controller is a design pattern that is used to split an application into 3 different components: the Model, the View, the Controller. These components handle specific requests throughout the application. It is efficient, and scalable, and is an industry standard.

- The Controller receives the requests from the user and sends it to the model.
- The Model responds to actions that are requested and returns it back to the controller. Once this is done the Controller sends it to the View.
- The View then responds with the data back to the user.

The diagram seen in Fig 51 displays the Model-View-Controller design pattern.

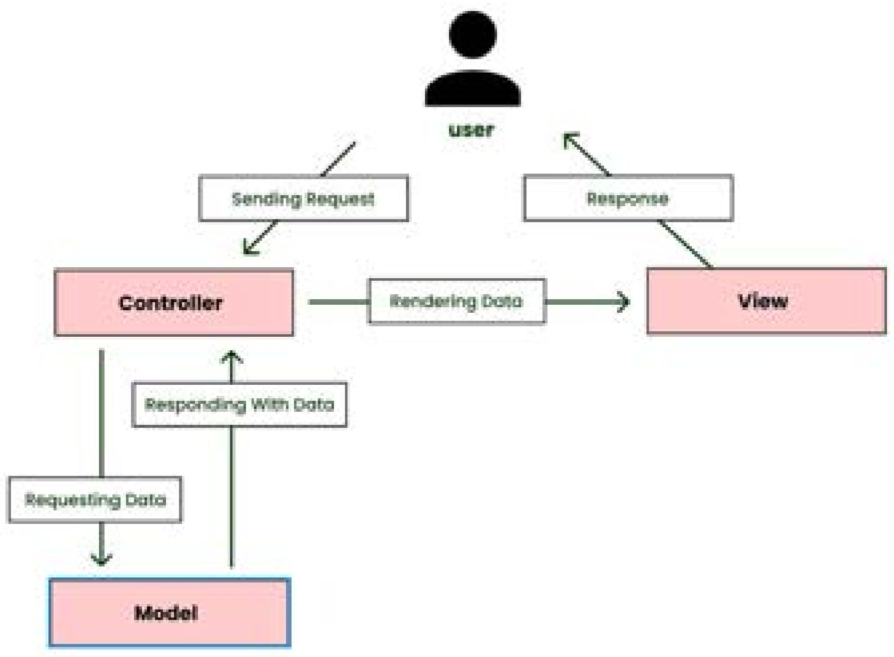


Figure 51 Model-View-Controller Design Pattern

4.2.4 Application architecture

A web application architecture is a guide on how an application's software are used together, and how that application reacts with each other. This can allow for scalability and reliability as they display the interactions are between things like middleware, databases, and the frontend (Ferguson, 2021).

In this application, the application is using the MERN stack, which consists of MongoDB, Express.js, React.js and Node.js. In this stack, React.js acts as the frontend of the application, it is used to quickly build UI and allow for the components that the website will be using. MongoDB will then act as the database for the application, taking in the requests sent by the user and sending back a response to the user with the requested data. It is used with components consisting of JSON and uses JavaScript. Express and Node.js will then act as the backend and server of the application, taking in API calls and requests and placing the data requested to the webpage.

The diagram below (figure 52) displays the application architecture for this project, displaying the frontend, the server, and the database, as well as showing the requests and responses that is returned from the database.

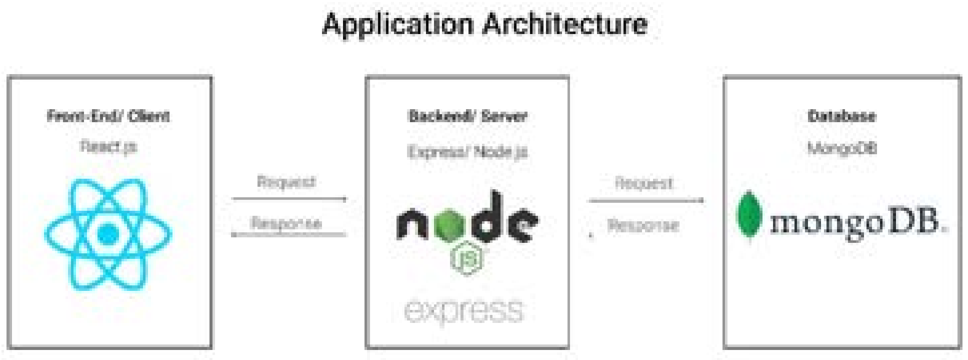
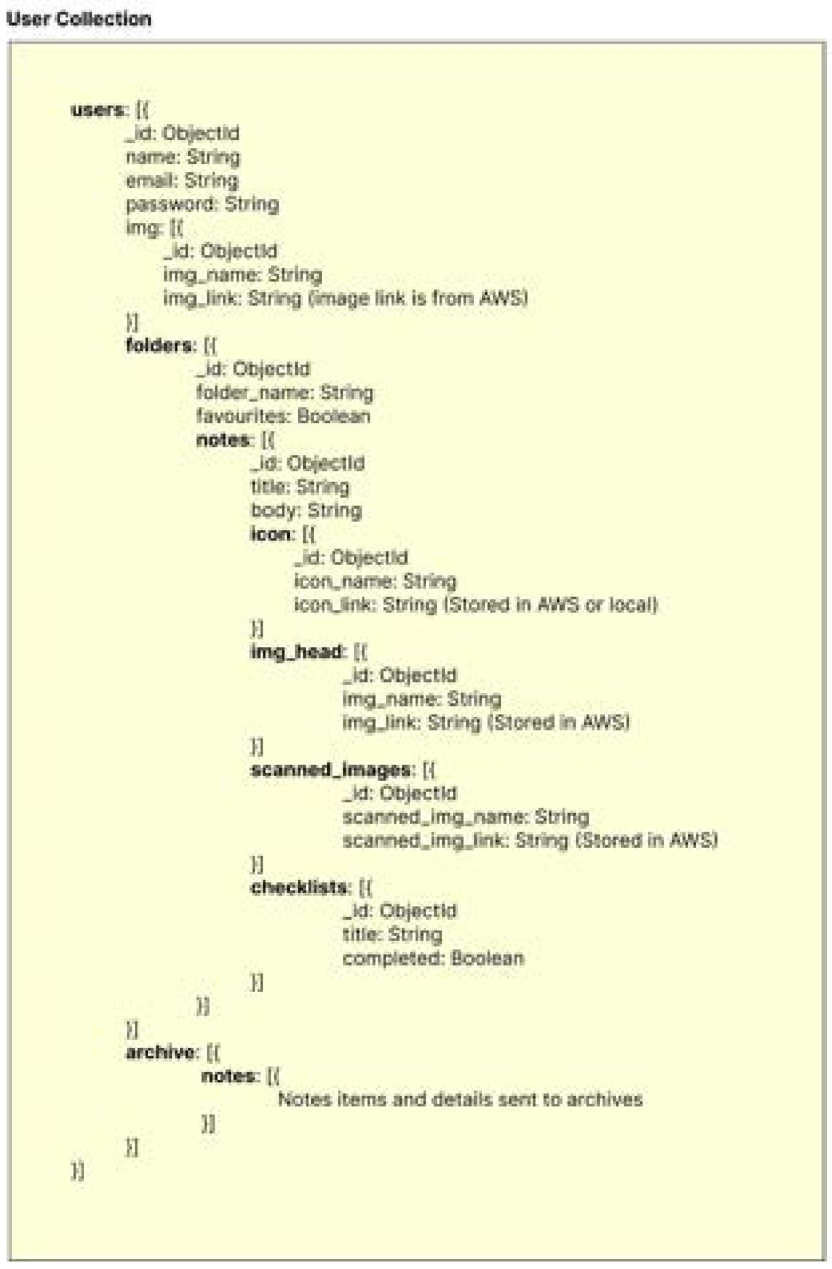


Figure 52 Application Architecture

4.2.5 Database design

The figure below (figure 53) shows the embedded data models. These data models are seen in the JSON file in MongoDB and are shown to represent how it looks in the database. These embedded data models are required to store a set of related data in a singular database record.



User Collection

```
users: [{
        _id: ObjectId
        name: String
        email: String
        password: String
        img: [{
            _id: ObjectId
            img_name: String
            img_link: String (image link is from AWS)
        }]
        folders: [{
                _id: ObjectId
                folder_name: String
                favourites: Boolean
                notes: [{
                        _id: ObjectId
                        title: String
                        body: String
                        icon: [{
                            _id: ObjectId
                            icon_name: String
                            icon_link: String (Stored in AWS or local)
                        }]
                        img_head: [{
                                _id: ObjectId
                                img_name: String
                                img_link: String (Stored in AWS)
                        }]
                        scanned_images: [{
                                _id: ObjectId
                                scanned_img_name: String
                                scanned_img_link: String (Stored in AWS)
                        }]
                        checklists: [{
                                _id: ObjectId
                                title: String
                                completed: Boolean
                        }]
                }]
        }]
        archive: [{
                notes: [{
                        Notes items and details sent to archives
                }]
        }]
}]
```

Figure 53 Database Design

ERD’s (Entity relationship diagrams) were also created to better display the relationships between each item in a diagram. The ERD of the project is displayed in the next image (figure 54) and can be better seen in Figma: https://www.figma.com/file/8wushDYxUM3YNyoGlC2RUW/Final-Project?node-id=0%3A1&t=vzfT2FTUSNURtNsh-1

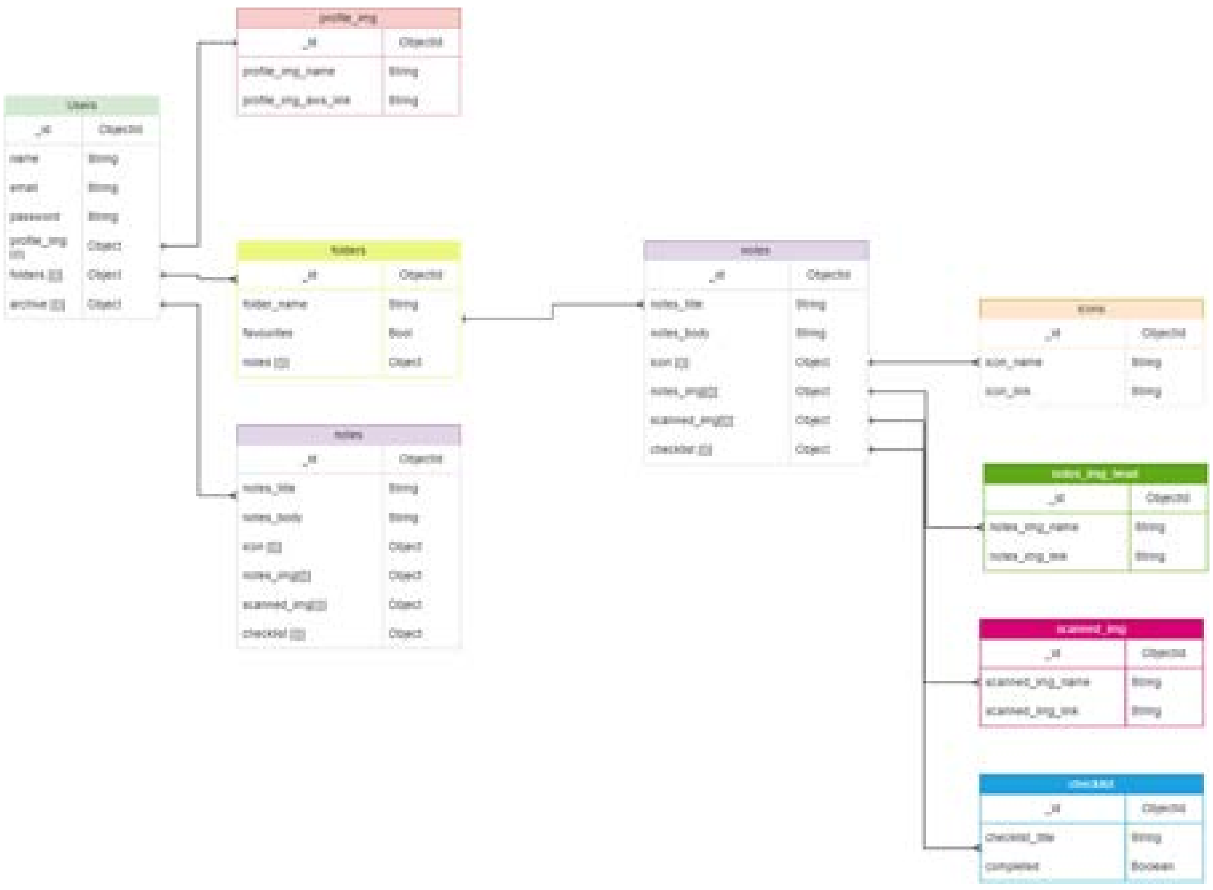


Figure 54 Entity Relationship Diagram

4.2.6 API design

REST Method	Endpoint	Description
GET	/user/ :id	Get User
POST	/register	Sign Up
POST	/login	Log In
PUT	/edit/user/ :id	Update user details
POST	/folder	Create Folder
GET	/folders	Get All Folders
GET	/folder / :id	Get Single Folder
PUT	/edit/folder/ :id	Update user details
DELETE	/folder/ :id	Delete Single Folder
POST	/note	Create Note
GET	/notes	Get All Notes
GET	/note/ :id	Get Single Note
PUT	/edit/note/ :id	Update note details
DELETE	/note/ :id	Delete Single Note
POST	/ocr_img	Create OCR Image
GET	/ ocr_img	Get All OCR Images
GET	/ ocr_img/ :id	Get Single OCR Image
DELETE	/ ocr_img /:id	Delete Single OCR Image
POST	/checklist	Create Checklist
GET	/ checklist	Get All Checklist
GET	/ checklist / :id	Get Single Checklist
PUT	/edit/ checklist / :id	Update Checklist details
DELETE	/ checklist / :id	Delete Single Checklist

4.2.7 Process design

To show different sequences that the user might be going through while using the application, a sequence diagram was created to display the process of what can be done while using the application. The sequence diagram can be seen in figure 55.

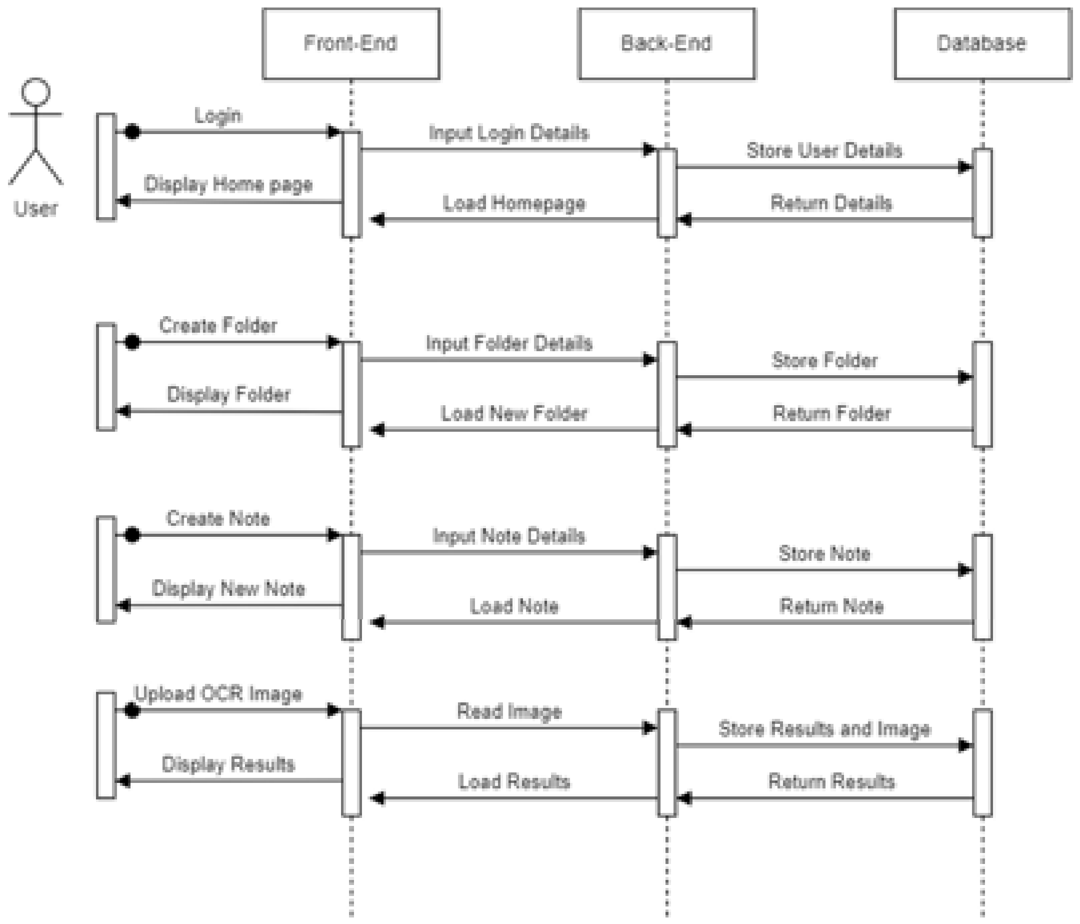


Figure 55 Sequence Diagram

4.3 User interface design

This section discusses the design of the user interface. This is done through a set of steps that look at the different requirements that were initially found in the previous section and implemented with the functionalities and design in mind.
The app will first look at paper prototypes, wireframes, user-flow diagrams, and style guides.

4.3.1 Wireframe

A wireframe is used to display the content and functionality of the layout of a page. These wireframes usually do not consist of colour, or typography or much detail to begin with. We start the development of the wireframes by initially creating a set of paper-prototypes that are used to develop ideas on where to put different pieces of functionality and how to use the system.

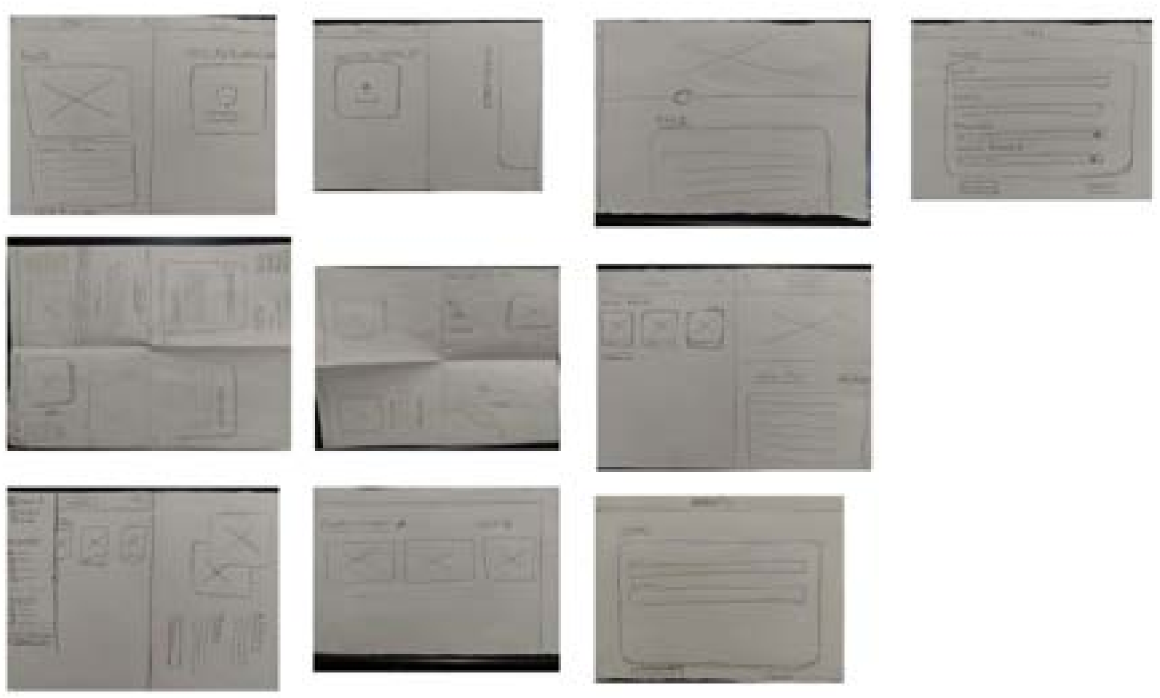


Figure 56 Paper Prototypes

Paper prototypes are generally done before building the wireframe, and are seen in the figure above (fig 56)
One the development of these paper prototypes is complete, having a slightly more developed version (without detail) is then created. The lo-fidelity (lo-fi) wireframes are then made with the use of paper-prototypes as a guideline. The lo-fi wireframes show a better look of how the application will look and provide a closer look into how the final application will look.
The wireframes are provided in figure 57 below and was designed in a prototyping tool known as “Figma.”

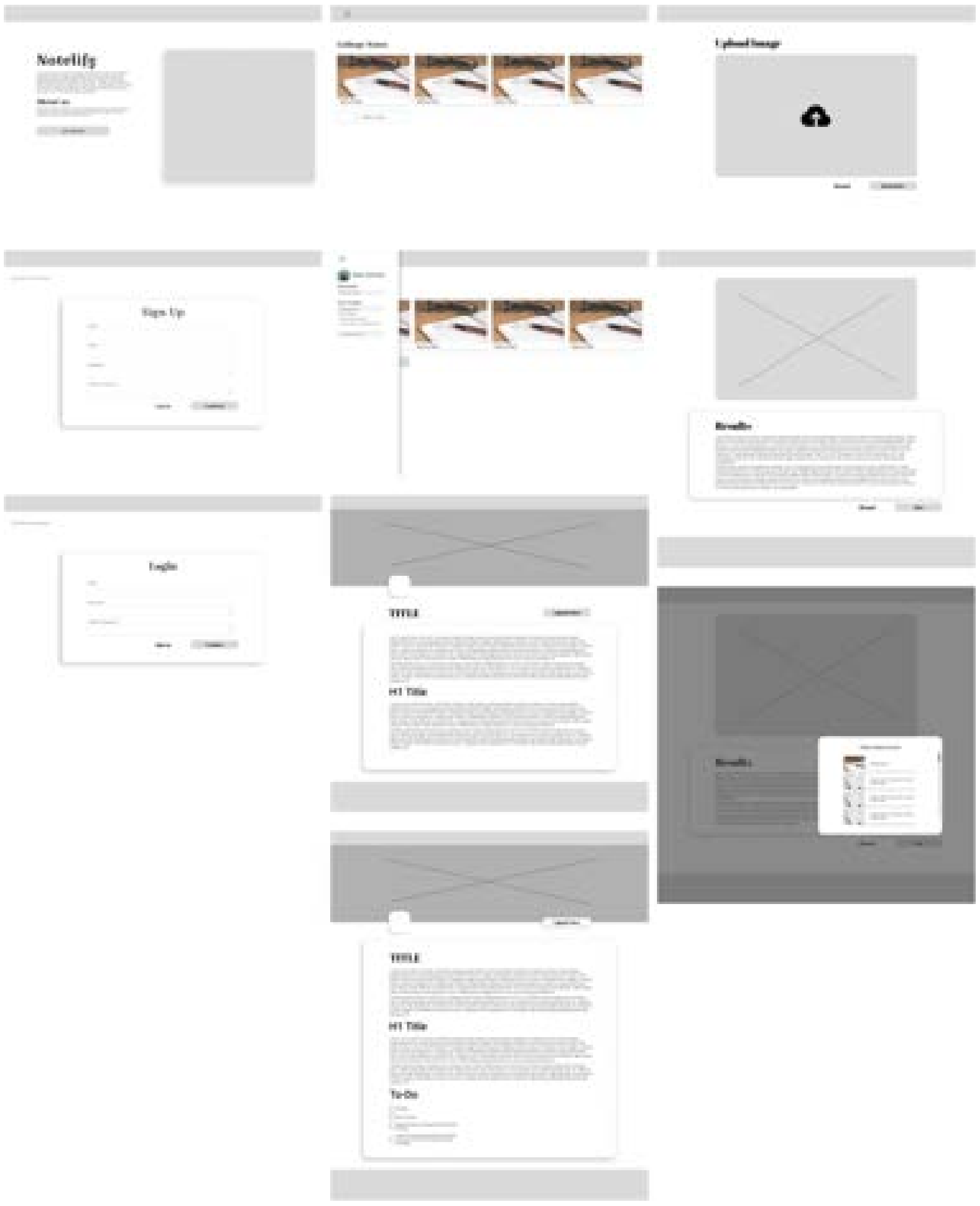


Figure 57 Wireframes

4.3.2 User Flow Diagram

The user flow diagram shows how users will navigate from one page to another within the application. It can be seen in figure 58.

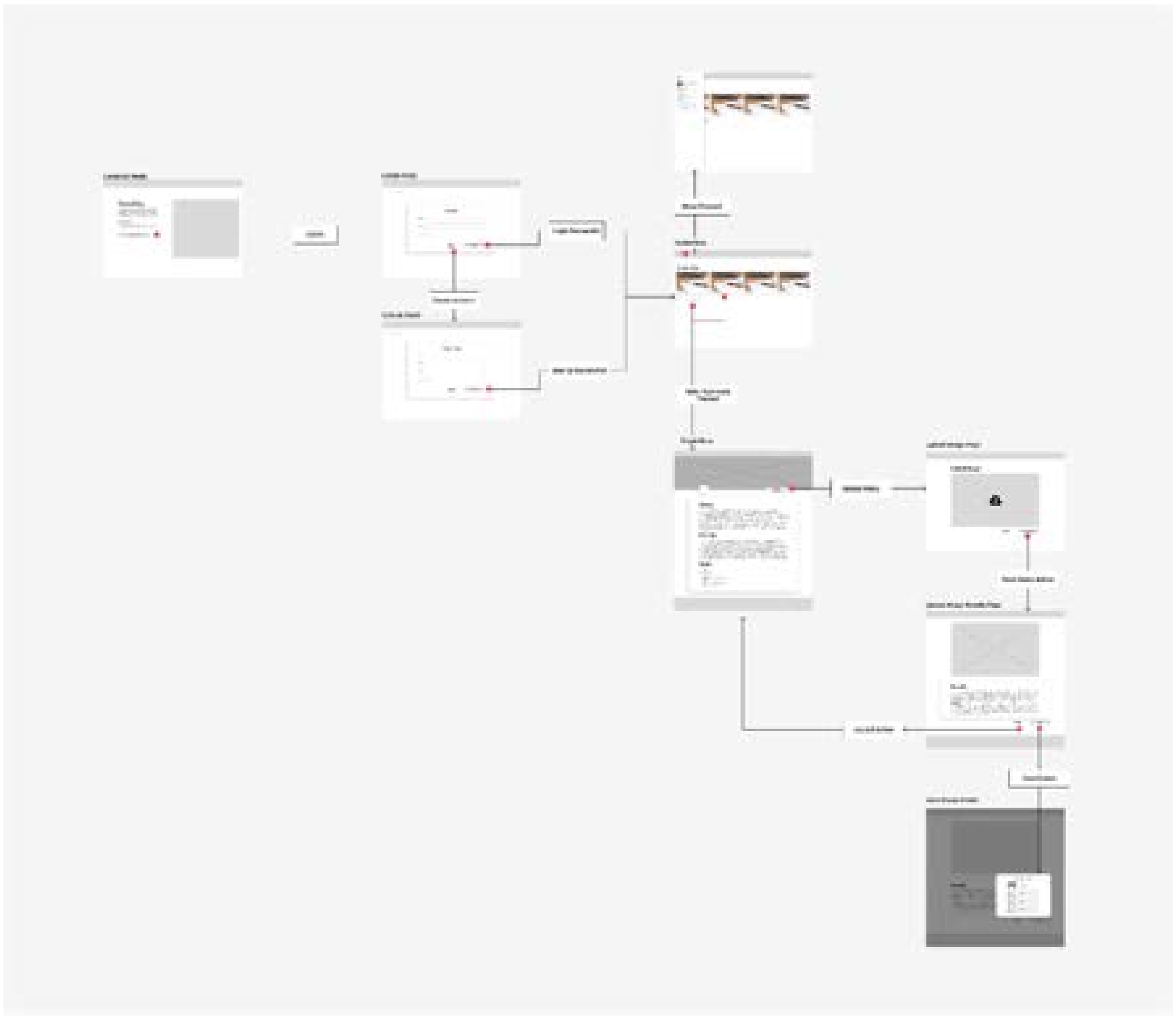


Figure 58 User Flow Diagram

The full diagram can be seen in:
https://www.figma.com/file/8wushDYxUM3YNyoGlC2RUW/Final-Project?node-id=0%3A1&t=Cc2mS5tSz8jgz3fE-1

4.3.3 Style guide

The style guide shows the colours, typography, layout, form elements, buttons, tabs, and cards for how the website is going to look. Style guides are often used to keep the website looking consistent. It also covers the grids and spacing of the application and how they are used throughout the website. The full style guide can be seen below in figure 59.

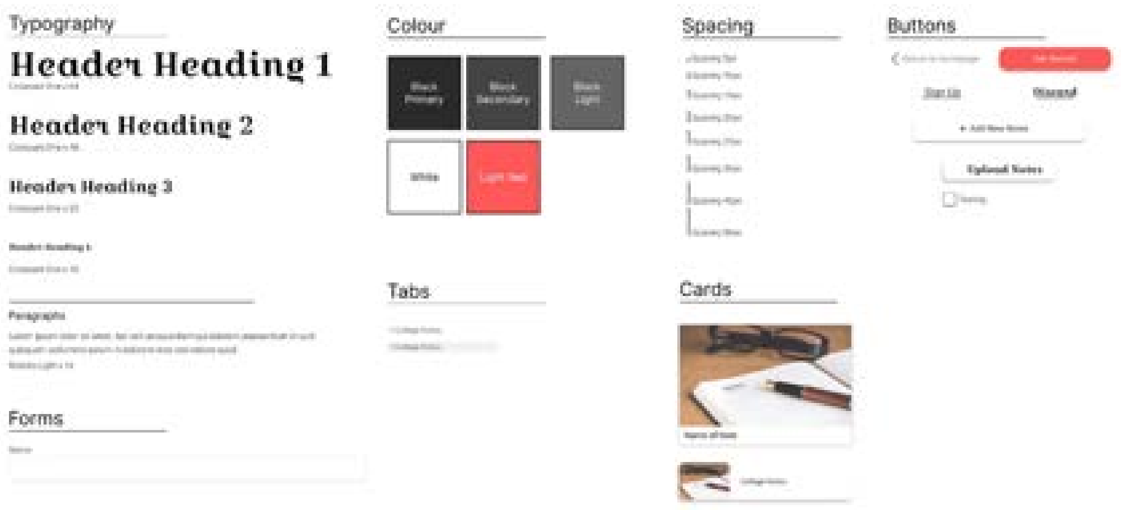
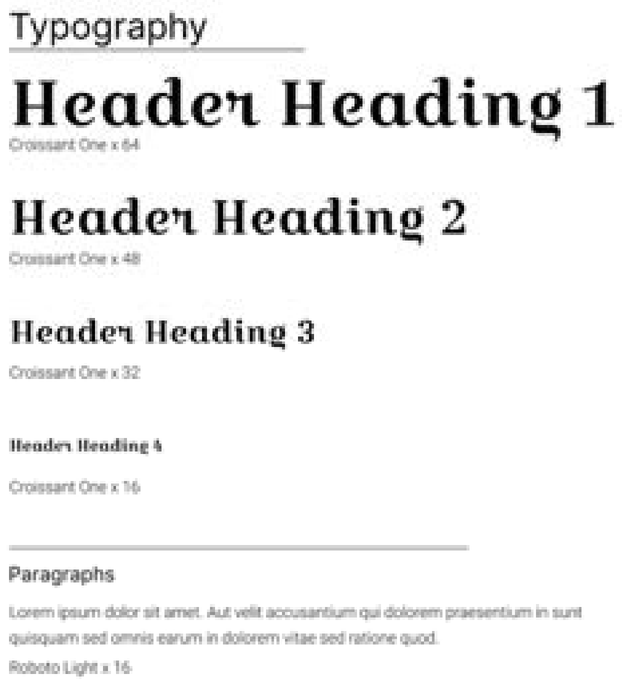


Figure 59 Style Guide

Typography:
The font that will be used for heading will be Croissant One in several different sizes for the headings to indicate importance. These headers will not be used often and will be reserved for headings and areas of importance. The reason behind using this font is because it is easy to read and suits the look and feel of the website. The paragraphs will consist of Roboto Light as it contrasts well with the heading font and is also clear and easy to read, making it a great choice for the body of the website. A screenshot of the typography can be seen in figure 60.


Typography

Header Heading 1
Croissant One x 64

Header Heading 2
Croissant One x 48

Header Heading 3
Croissant One x 32

Header Heading 4
Croissant One x 16

Paragraphs
Lorem ipsum dolor sit amet. Aut velit accusantium qui dolorem praesentium in sunt quisquam sed omnis earum in dolorem vitae sed ratione quod.
Roboto Light x 16

Figure 60 Typography

Colour:

Colour will be used sparingly throughout the website. It will be used to highlight important objects that are seen throughout the website, such as buttons. The colour pallet will remain simple, as the main use of colour throughout the application will be from images. Figure 61 shows the colour pallet of the application.

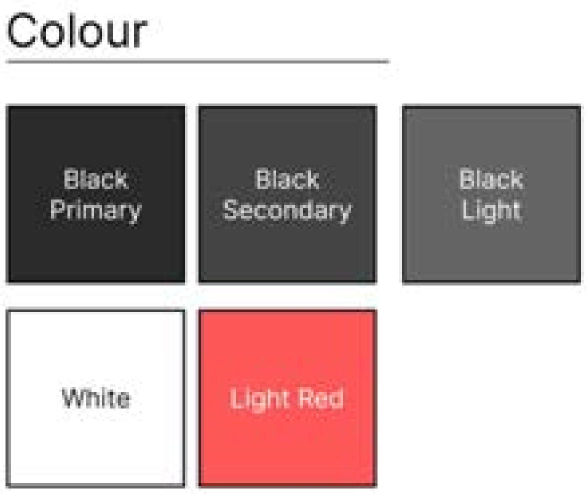


Figure 61 Colour Palette

<u>Spacing</u>:
Spacing is used to keep the application consistent with the size and how far each element is away from each other. The consistency of the spacing is essential to allow for a more professional-looking application. The spacing is seen in figure 62.

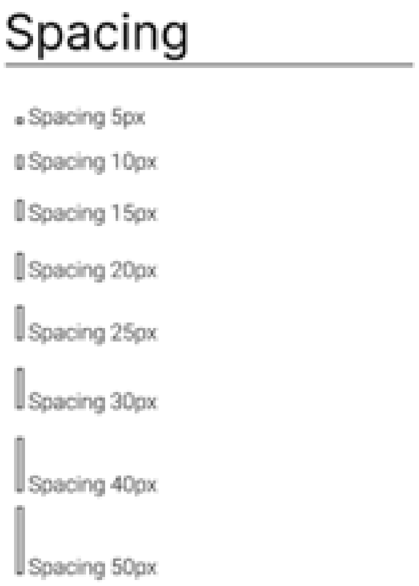


Figure 62 Spacing Size

<u>Buttons</u>:
Buttons are used throughout the entire website. The different buttons are for different occasions, for example “return to homepage” will only be used in the sign up and log in form. These buttons can be seen in figure 63.

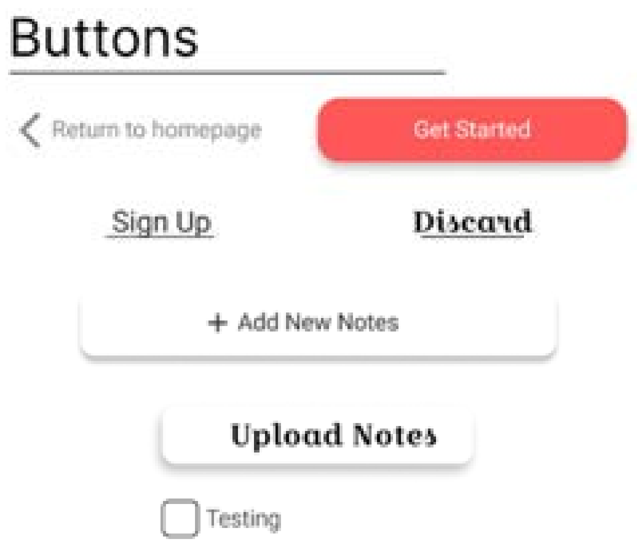


Figure 63 Button Variation

<u>Forms</u>:
The form will consist of the text area and the name, it will also contain the buttons that are needed. This can be the “sign up” button, the “confirm” button and the “return to homepage” button.

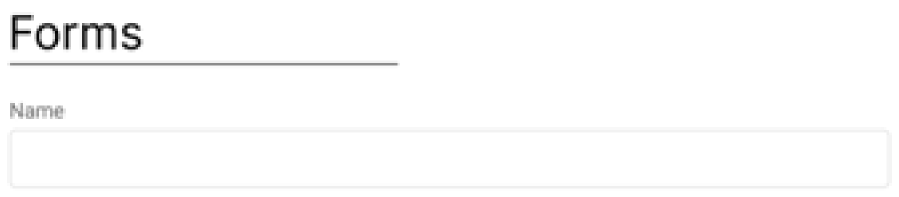


Figure 64 Forms

Tabs:
Tabs are used to display where the user is on the website. It is used for easy navigation and is accessible through the side navigation. The tabs are kept simple and clean to allow for as little visual clutter as possible when navigating the website. Figure 65 shows the tabs when navigating the website.

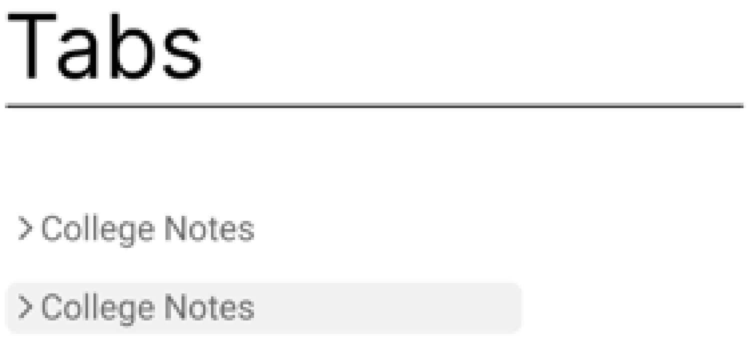


Figure 65 Tabs

Cards:
Cards are used to help distinguish the different notes that the user creates. The cards available will contain an image and a name of the note to allow users to differentiate it from others. It remains simple to prevent visual clutter, with a drop shadow to highlight the card itself. The second card shows when the user needs to save a note after using the OCR functionality. The two cards can be seen in figure 66.

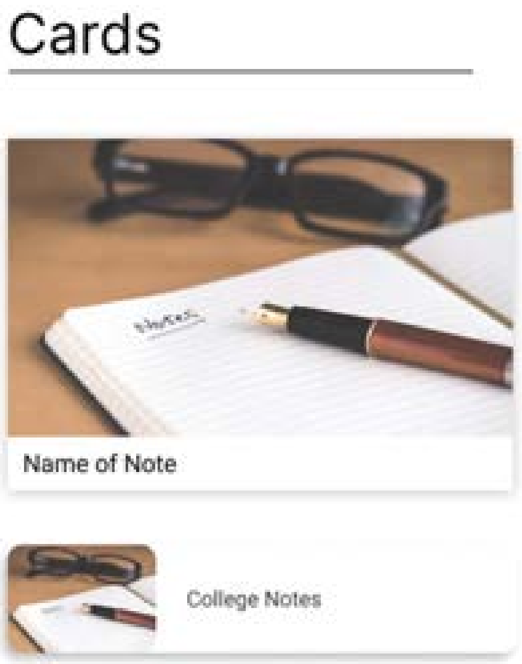


Figure 66 Card Variations

4.4 Conclusion

In conclusion, this chapter discussed the program design, and how it will be developed, as well as the user interface design, which discussed the different UI elements that will be seen in the application.

5 Implementation

5.1 Introduction

The implementation chapter discusses how the application was developed, as well as the technologies used to build it. This chapter will be discussing the different technologies used within the project, and the sprints that occurred throughout the project, and the items that were completed within each sprint.

5.2 Technologies

The technologies that were used to develop the application are seen as follows:

- **React.js**
  React is a JavaScript framework that is primarily used to develop the front-end of an application. React is component-based and can be used to quickly develop an effective and intuitive, web-based UI (David, 2022). The reason for choosing React.js was primarily for its components, vast number of libraries and resources, and it's React Native feature, to allow for mobile-compatibility, and its fast development.

- **Express.js**
  Express.js is a fast, and unopinionated web framework for Node.js that is used to build RESTful APIs and will provide the backend for the application. (Express - Node.js web application framework, 2017)

- **MongoDB**
  MongoDB will act as the database for the application. It is a very flexible database and scales very well as the application develops.

- **Mongoose**
  Mongoose is an Object Data Modelling (ODM) and is used for MongoDB and Node.js. It can handle the relationships between the data and can translate objects within code and how it is represented in the database, as well as provide schema validation. (Introduction to Mongoose for MongoDB, 2018)

- **AWS S3 Bucket**
  AWS S3 is an Amazon object storage service that can retrieve and store data. It will be used in the application to store images. (Getting Started – Amazon Simple Storage Service (S3) – AWS, 2023)

- **Socket.io**
  Socket.io is a library that allows for cross communication between a client and server and is used in our text editor to share documents with other users. (Introduction | Socket.IO, 2023)

- **Tesseract.js**
  Tesseract.js is a JavaScript library port of the Tesseract OCR engine. The port will allow developers to use OCR functionality in a web-based application. The library is used to provide the OCR functionalities seen in the application. (Tesseract.js | Pure Javascript OCR for 100 Languages!, 2023)
- **Quill**
  Quill.js is a powerful text editor library, that will be used to provide for the notes-taking feature that the application will have. It is powerful, easy to use and customizable (Quill - Your powerful rich text editor, 2023). The reason behind using Quill is because of the Delta feature, which allows developers to store the position and style of that specific document in a backend.

- **Figma**
  Figma is a free online web service that is used for designing UX / UI applications. Figma was used to design the wireframes, and UI of the project, and to create the design systems of the application.

The application that will be developed for this project is a note taking application that contains OCR functionalities. The user will be able to create new documents, sort their documents into folders and use OCR functionalities alongside the notes taking features to help users take notes on different mediums and transfer it to the application.

5.3 Implementation Roles

The project consisted of a single developer, and a project supervisor and secondary reader. The project will be developed by Clemente, including all the design, functionalities, and testing that is seen throughout the project, while being supervised by Cyril Connolly. Weekly meetings were held, where updates for the project progress and criticisms were exchanged between the student and supervisor and were consistent during the development process.

5.4 Scrum Methodology

Scrum Methodology was used for the implementation of this project. The Scrum methodology follows the Agile Development methodology, and divides the project into smaller, manageable goals that are achievable throughout development. These goals are generally accomplished within one to two weeks, called sprints, depending on the plan that the developers have set out. (What Is Scrum Methodology? & Scrum Project Management, 2022)

The project is divided into different roles, which consists of:

- Scrum Master – The Scrum Master is to guide the developers and manage the project. They will oversee the project, keeping the Scrum up to date and will provide training and mentoring in case a team needs it (What Is Scrum Methodology? & Scrum Project Management, 2022)

- Software development team – The team that will be developing the project and following the Scrum goals.
- The product owner – Will be the clients that will use the software that is being developed.

In this project, Cyril will provide the role of the Scrum Master, while Clemente will be the software development team that will accomplish the goals set out to him.

Each sprint in the project will be completed every two weeks, and in total consisting of 7 sprints, and a set of goals will be accomplished within those sprints. The figure below shows how the scrum methodology was used:

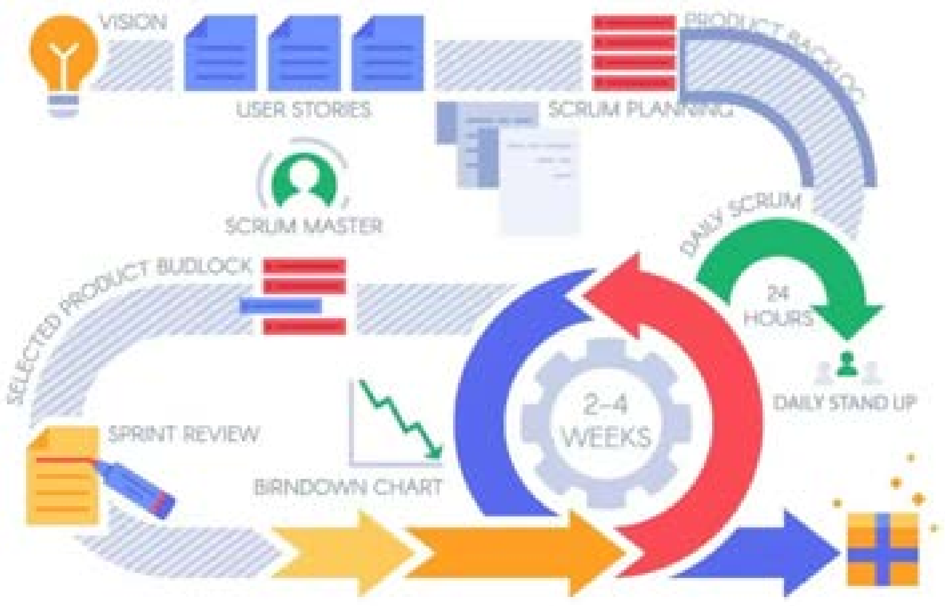
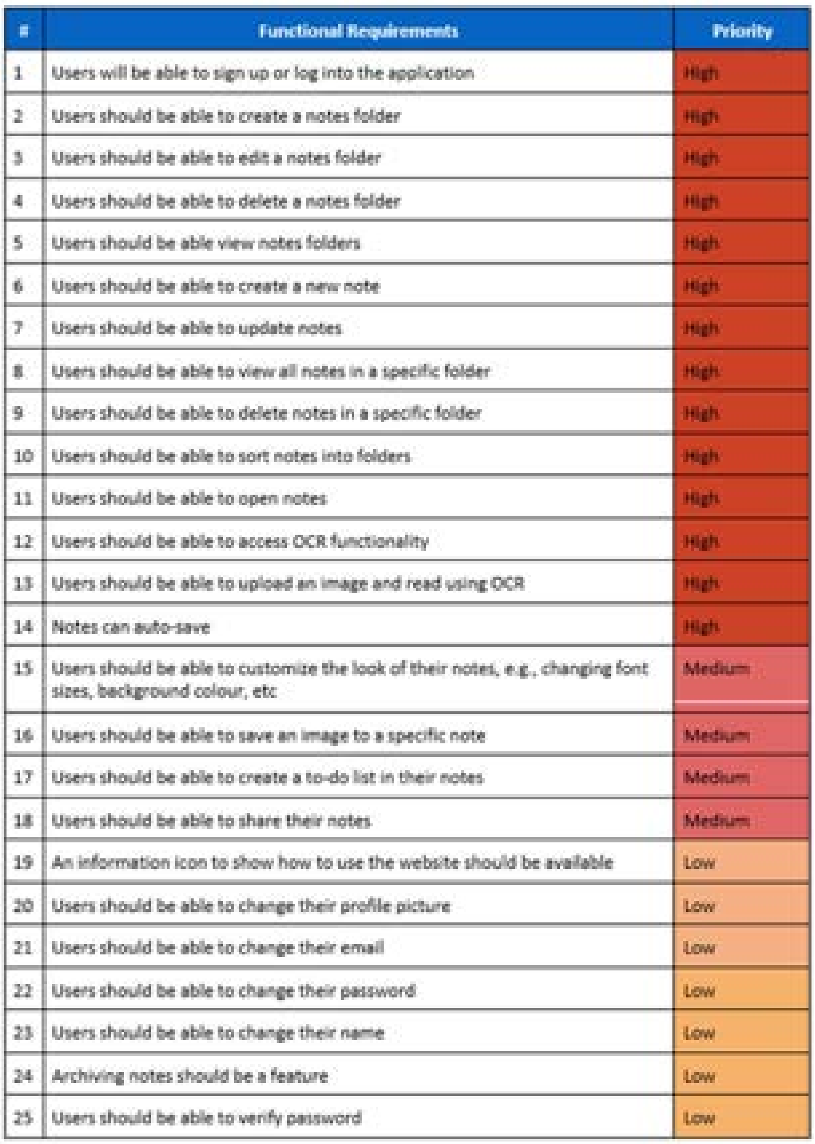


Figure 67 Scrum Diagram (What Is Scrum Methodology? & Scrum Project Management, 2022)

The requirements for the application were broken into a product backlog, and each item on the product backlog was broken down to smaller functionalities, which created the sprints. The product backlog can be seen in the figure below:



#	Functional Requirements	Priority
1	Users will be able to sign up or log into the application	High
2	Users should be able to create a notes folder	High
3	Users should be able to edit a notes folder	High
4	Users should be able to delete a notes folder	High
5	Users should be able view notes folders	High
6	Users should be able to create a new note	High
7	Users should be able to update notes	High
8	Users should be able to view all notes in a specific folder	High
9	Users should be able to delete notes in a specific folder	High
10	Users should be able to sort notes into folders	High
11	Users should be able to open notes	High
12	Users should be able to access OCR functionality	High
13	Users should be able to upload an image and read using OCR	High
14	Notes can auto-save	High
15	Users should be able to customize the look of their notes, e.g., changing font sizes, background colour, etc	Medium
16	Users should be able to save an image to a specific note	Medium
17	Users should be able to create a to-do list in their notes	Medium
18	Users should be able to share their notes	Medium
19	An information icon to show how to use the website should be available	Low
20	Users should be able to change their profile picture	Low
21	Users should be able to change their email	Low
22	Users should be able to change their password	Low
23	Users should be able to change their name	Low
24	Archiving notes should be a feature	Low
25	Users should be able to verify password	Low

Figure 68 Backlog

These sprints were accomplished over the course of two weeks and encouraged developers to be productive to keep up to date with their work.

5.5 Development environment

5.5.1 Visual Studio Code

Visual Studio Code is a fast code editor that supports developers by syntax highlighting, bracket-matching, auto indentation and many more. It allows developers to inspect and debug their code and add many other plugins that is created by the community. (Visual Studio Code, 2021)

The reason Visual Studio Code is being used is because it runs well with web development and JavaScript and can highlight different errors that can occur during the development process. It is fast, easy to use and highly customizable to run with different plugins.

Visual Studio Code is where most of the code will be written. It is compatible with React, JavaScript, Mongo, Mongoose and works very well when also using Node to install the packages that will be used to develop the application.

5.5.2 GitHub and Git

GitHub provides a method of version control. It will be used to create the many commits that will be created and tracks the different changes that a developer makes within those commits. It can also provide for different branches and merging. (What Is GitHub? A Beginner's Introduction to GitHub, 2022)

The version control system was essential during development to help save progress or revert to a previous version in case the current version is too broken to salvage.

The way these versions are uploaded to GitHub is using Git commands, which allows for the different features and commits to be saved.

5.6 Sprint 1 - Research and Requirements

5.6.1 Goals

The goal of Sprint 1 is to research project ideas, view similar applications, and conduct surveys and interviews to gather information on a user's needs.

- **Item 1:** Research the ideas for project in a similar area.
- **Item 2:** Researching OCR Technologies in React.
- **Item 3:** Gather User Requirements
- **Item 4:** Create User Personas
- **Item 5:** List Functional and Non-Functional Requirements of application
- **Item 6:** Create Use-Case Diagrams

5.6.2 Item 1 – Researching Project Ideas

The goal of this item was to perform research for the project area and find out what will be built towards the end of the application. Due to the nature of OCR being heavily committed with writing, developing a notes-taking application with OCR functionality was decided early into the project. The idea of the project was to allow users to take notes on pen and paper and transfer those notes to a desktop application to give them flexibility in where they can store their notes.

5.6.3 Item 2 – Researching OCR Technologies in React

Technologies were then researched to see if this application was feasible to develop. As it turns out, there are available libraries for React.js to that will allow developers to build an OCR application. One of these being Tesseract.js, a port of Tesseract OCR to a react library, usable within JavaScript.

A prototype was then developed in React.js to see if it is possible to add this feature.

Prototypes for the notes-taking side of the application was also developed during this time. A library called "Quill.js" was used to develop the notes-taking part of the application, as it is a very powerful notes taking library with the ability to store the properties of what is in the documents.

Tutorials were used and altered for both the OCR prototype and the Notes taking prototype. The notes taking tutorial can be found here: (Simplified, 2021) and the OCR Reader prototype can be found here: (basarat, 2022)

5.6.4 Item 3 – Gather User Requirements

Gathering the user requirements was to find the different functionalities and features that a user would be able to use within the application. The requirement gathering process was done in three ways to maximize the usable functionalities within the application.

These steps consisted of:

- Researching similar applications
- Surveys
- Interviews

5.6.4.1 Item 3.1 – Researching Similar Applications

Similar applications were looked and examined to find the advantages and disadvantages of that specific application. High quality applications were looked at to ensure that the functionalities of the program that is being built will contain the essential features of a notes taking application.

The applications that were researched were:

- Notion
- Evernote
- Google Keep
- Text Scanner [OCR]

The advantages and disadvantages of these applications can be found in Research Section of this report.

5.6.4.2 Item 3.2 – Surveys

Surveys were conducted to gather mass data on the application. The surveys were sent to gather data on whether the common person uses notes taking applications, what they like about these applications, and the different features that they would like to see on an application like this.

These surveys returned a variety of responses and helped further determine the functionalities that the application will possess.

The results can be seen in the research section of the report.

5.6.4.3 Item 3.3 – Interviews

Users were also interviewed to find what features suit them, what they would like to see and how they interact with these kinds of notes taking applications.

In these interviews, the users were asked what their favourite features were from notes taking application that they have previously used, and the results were listed in the research section of the report.

5.6.5 Item 4 – Creating User Personas

A set of user personas were created to showcase the potential users who will be using the application, and to display their needs and wants that the application should provide. These personas can be seen in figure 69 and is shown in greater detail in the research section.

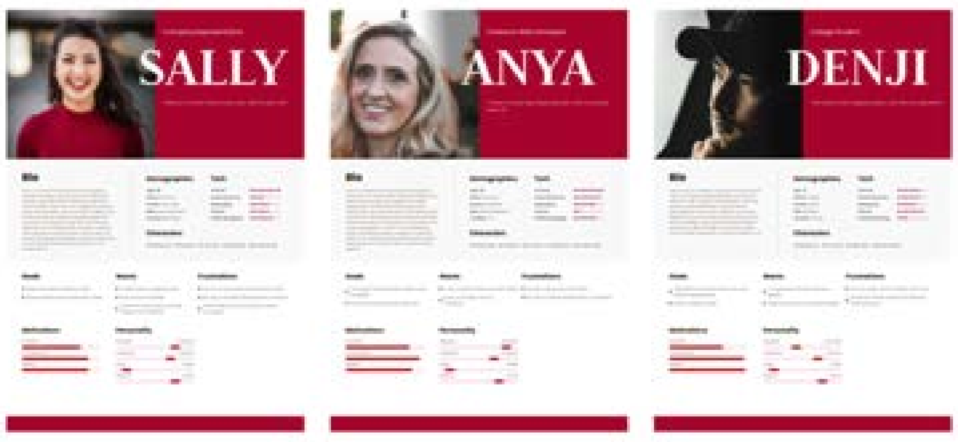


Figure 69 Personas

5.6.6 Item 5 – List of Functional and Non-Functional Requirements of the application

Functional and Non-Functional requirements were then listed. These requirements were gathered from the needs that has been provided from the information that was gathered from the surveys, interviews, and personas. The requirements were listed from high to low priority to show the developers what to prioritise during the development of the application. The list can be seen in the research section of the report.

5.6.7 Item 6 – Creating Use-Case diagrams.

A use-case diagram was then created to showcase what certain users can do. These diagrams show the registered and un-registered users and what the application allows them to do.

5.6.8 Conclusion of Sprint 1

The overall goals of this sprint were accomplished and gave a clear end-product on what should be built towards the end of the project, showing the functionalities and needs that should be implemented towards the end of the project.

5.7 Sprint 2 – Design

5.7.1 Goal

The goal of Sprint 2 was the design the different parts of the application. These included both the Program Design, and the User Interface Design, showcasing how the project will be developed in its design systems, as well as the look of the application.

- **Item 1:** Research the Technologies
- **Item 2:** Database Design
- **Item 3:**Process Design
- **Item 4:** Wireframes
- **Item 5:**User Flow Diagram
- **Item 6:**Style Guide

5.7.2 Item 1 – Researching the Technologies.

This item consists of performing research on the technologies that will be used in the application, the design patterns that will be used and the architecture of the overall application.

- Technologies and Structure

  The technologies that will be used for this application was displayed in this section, showcasing the stack that will be used and explaining the reasoning behind using them.

  The structure of how React.js is used was also discussed, showing the different folder structures, and explaining the reasoning behind that structure.

  The Model View Controller (M.V.C) design pattern was also discussed in this section, showing how the stack uses M.V.C and a diagram to compliment it too.

  Application Architecture was also in this section, it is to show the connections between the technologies and how they all worked together to create the finished product.

These can all be seen in further detail in the design section of the application.

5.7.3 Item 2 – Database Design

The database design section then showcases the structure of the database. Diagrams were created to show the relationships between data, and how they will connect with each other. These diagrams include a Database Record and Entity Relationship Diagram and can be seen in the figure below (figure 70), and in greater detail in the Design section of the document.

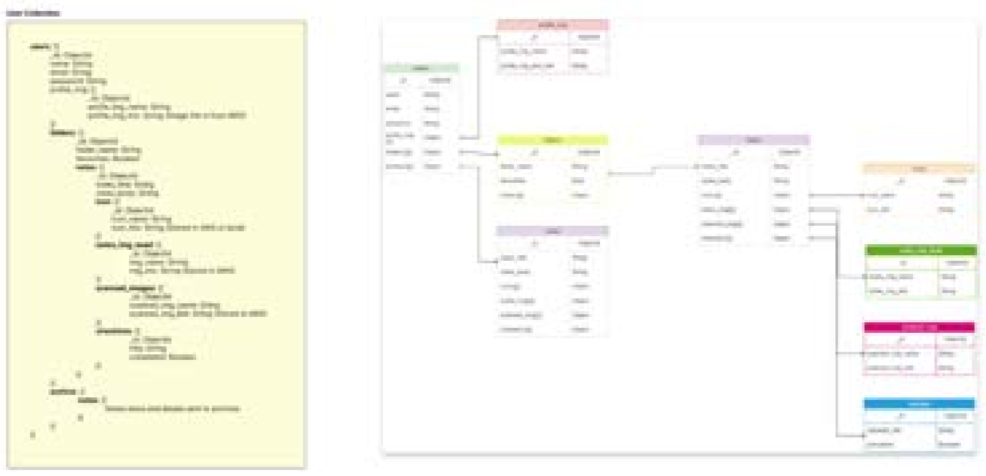


Figure 70 Database Design

5.7.4 Item 3 – Process Design

The API and Process Design were created in item 3.

The API design shows how the calls should look in the backend of the application, and what calls a specific entity should be able to perform.

The process design then shows a sequence diagram that show the process of what is being performed.

These diagrams can be seen in the design section of the report.

5.7.5 Item 4 – User Interface Design

Item 4 then shows the creation of the User Interface Design of the application. This is done through a series of steps that allows for a final design to be created, which includes:

- Paper Prototypes and Wireframes
- User Flow Diagram
- Style Guides

5.7.5.1 Item 4.1 – Paper Prototypes and Wireframes

Paper prototypes were produced to quickly lay down ideas for a potential UI design. This was done to allow for mistakes and fast creation, and to give ideas on how the webpage will look before wireframes are produced.

Once a paper prototype was completed, a wireframe is then created. The design of the wireframe is based on the design of the paper prototype; however, the wireframe is built in a design tool. In this case, the wireframe was built using Figma.

These paper prototypes and wireframes can be seen in the figure below (Figure 71) and can be seen in greater details in the Figma board, or the design section.

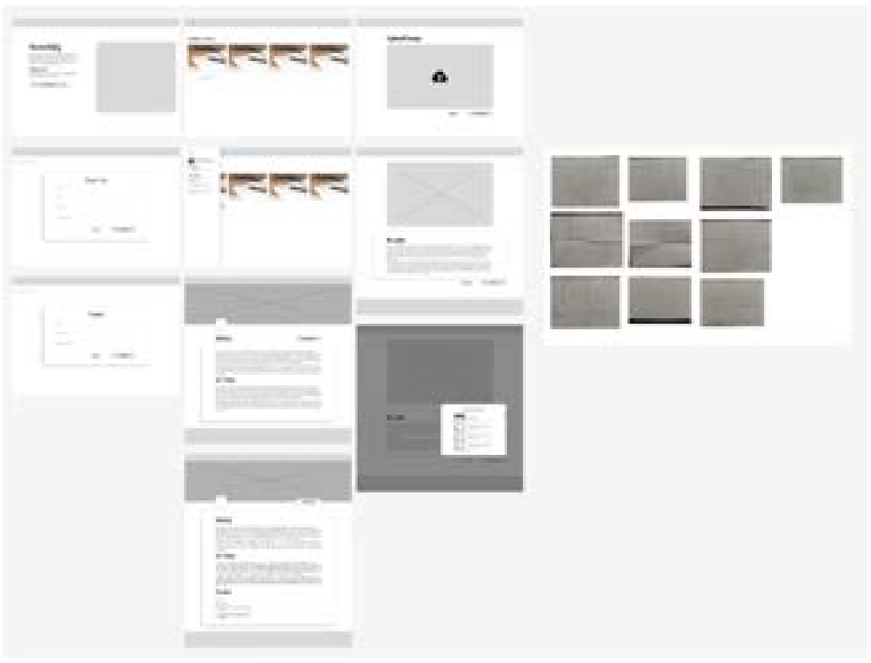


Figure 71 Wireframe

5.7.5.2 Item 4.2 – User Flow Diagram

A user-flow diagram was then created to show how a user will interact with the website, and the pages that they are to be led to, after clicking a specific button, or nav component. This was done to show how they will interact with the design and for navigation purposes. The user-flow diagram can be seen in figure 72.

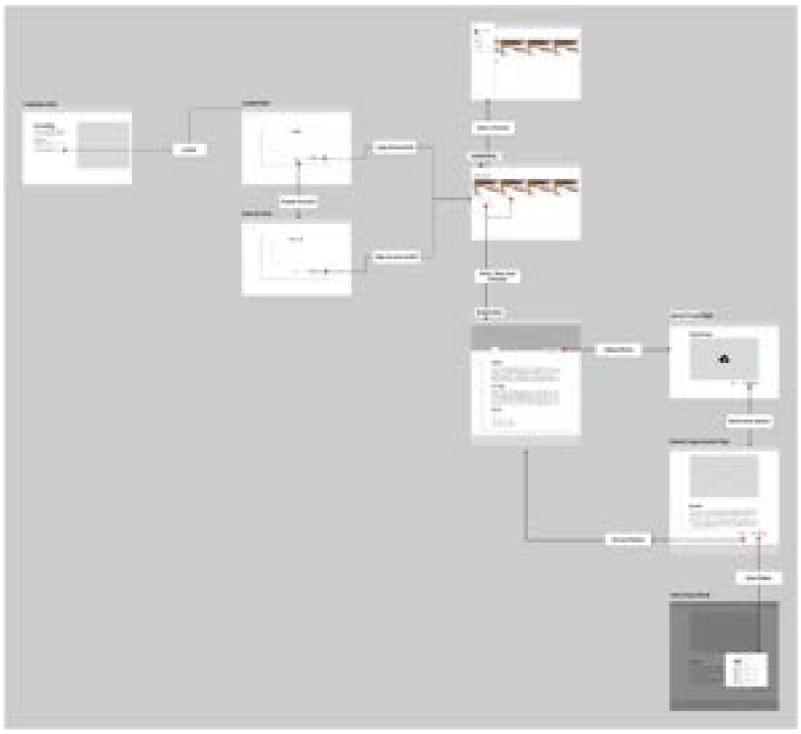


Figure 72 User Flow Diagram

5.7.5.3 Item 4.3 – Style Guide and Finished Design

Finally, a style guide and the finished design was created. The style guide was built to give the UI of the application a consistent look, and to choose the colour scheme, iconography, font and font sizes, etc.

With the style guide completed, the finished design was then constructed, and built with the use of the style guide and the wireframe.

The finished design and style guide can be seen in figure 73.

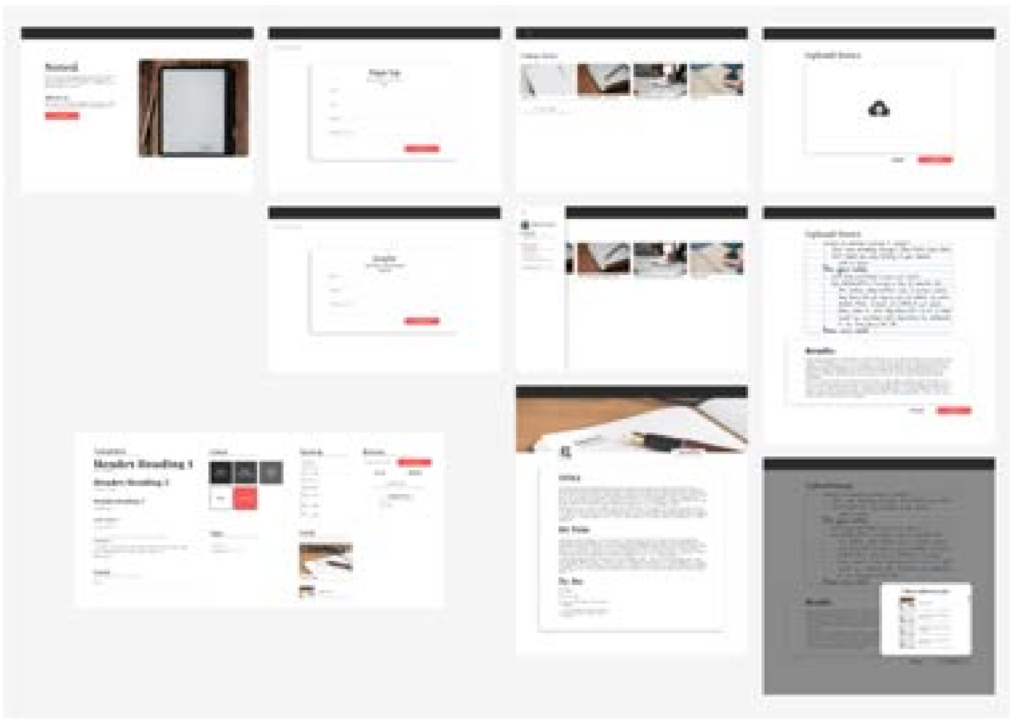


Figure 73 Final Design

5.8 Sprint 3 – Implementation 1

5.8.1 Goal

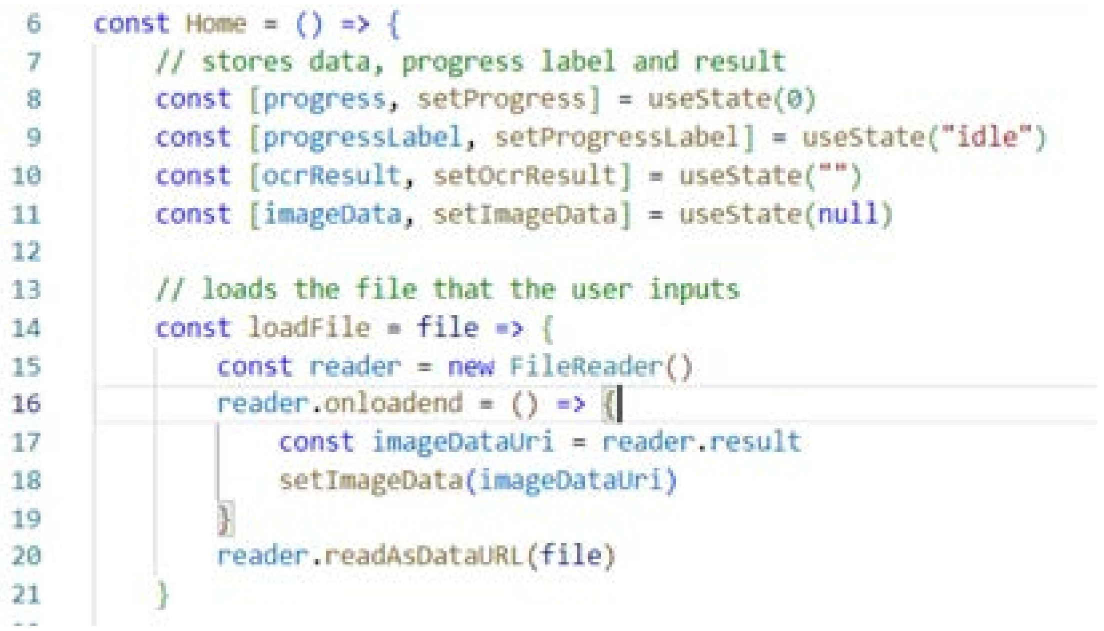
The goal of sprint 3 was to begin the implementation of the application. During this sprint, prototypes were further developed separately to see if the different parts of the application can work together. The items that were developed during this phase of the project were:

- **Item 1:** OCR Functionality prototype.
- **Item 2:** Notes taking prototype.
- **Item 3:** User profile.

5.8.2 Item 1 – OCR Functionality Prototype

A prototype for OCR functionality was developed in React. The tutorial used to develop this prototype was found on YouTube and uses a library for React known as “Tesseract OCR”, a library that was ported from Python to JavaScript.

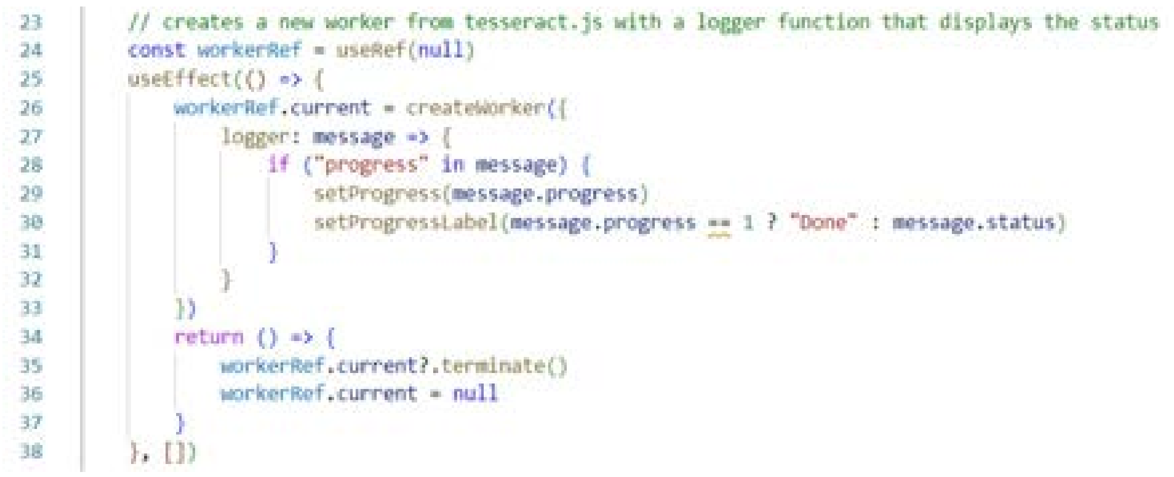
The prototype that was developed took in an input from the user and uses Tesseract to read the text and return it back to the user.



```
const Home = () => {
    // stores data, progress label and result
    const [progress, setProgress] = useState(0)
    const [progressLabel, setProgressLabel] = useState("idle")
    const [ocrResult, setOcrResult] = useState("")
    const [imageData, setImageData] = useState(null)

    // loads the file that the user inputs
    const loadFile = file => {
        const reader = new FileReader()
        reader.onloadend = () => {
            const imageDataUri = reader.result
            setImageData(imageDataUri)
        }
        reader.readAsDataURL(file)
    }
```

Figure 74 OCR Prototype (1)

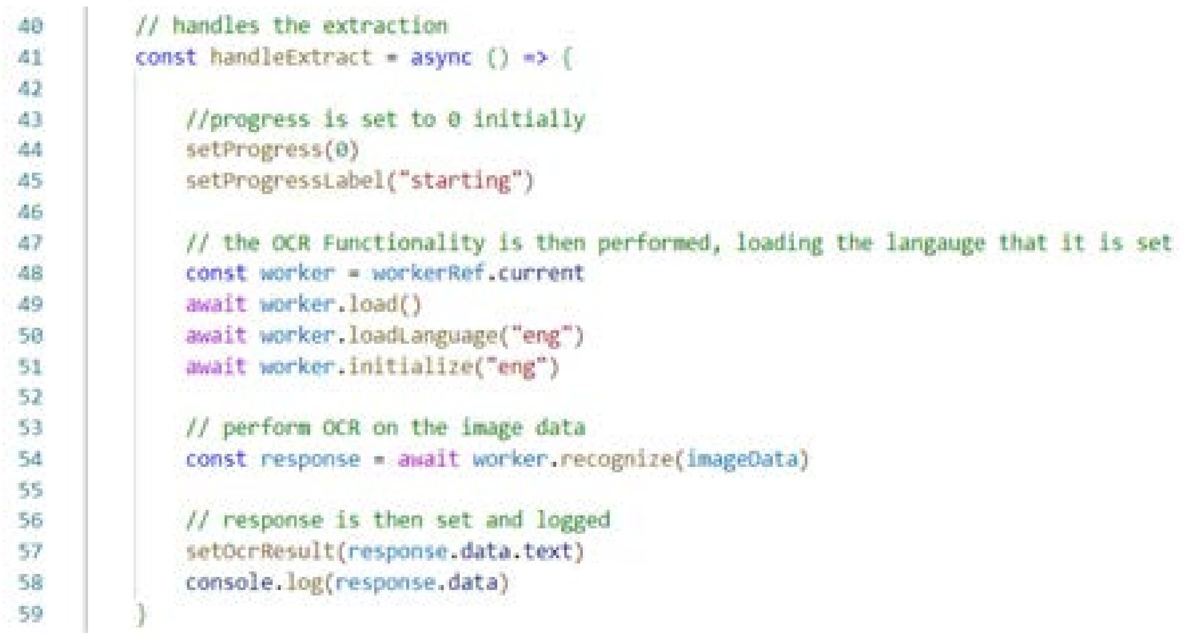
In figure 74, the data is first stored using useState, setting it to its initial value. The file that the user inputs is then loaded with the use of the “loadFile” method. When the file is then read, the result is set to the image data using the “setImageData” function.



```
// creates a new worker from tesseract.js with a logger function that displays the status
const workerRef = useRef(null)
useEffect(() => {
    workerRef.current = createWorker({
        logger: message => {
            if ("progress" in message) {
                setProgress(message.progress)
                setProgressLabel(message.progress == 1 ? "Done" : message.status)
            }
        }
    })
    return () => {
        workerRef.current?.terminate()
        workerRef.current = null
    }
}, [])
```

Figure 75 OCR Prototype (2)

Figure 75 then shows a new worker being created, coming from Tesseract.js. The worker ref then logs the progress message, setting the status until it is complete. Once it has been completed, it then terminates the workerRef.



```
40  // handles the extraction
41  const handleExtract = async () => {
42
43      //progress is set to 0 initially
44      setProgress(0)
45      setProgressLabel("starting")
46
47      // the OCR Functionality is then performed, loading the langauge that it is set
48      const worker = workerRef.current
49      await worker.load()
50      await worker.loadLanguage("eng")
51      await worker.initialize("eng")
52
53      // perform OCR on the image data
54      const response = await worker.recognize(imageData)
55
56      // response is then set and logged
57      setOcrResult(response.data.text)
58      console.log(response.data)
59  }
```

Figure 76 OCR Prototype (3)

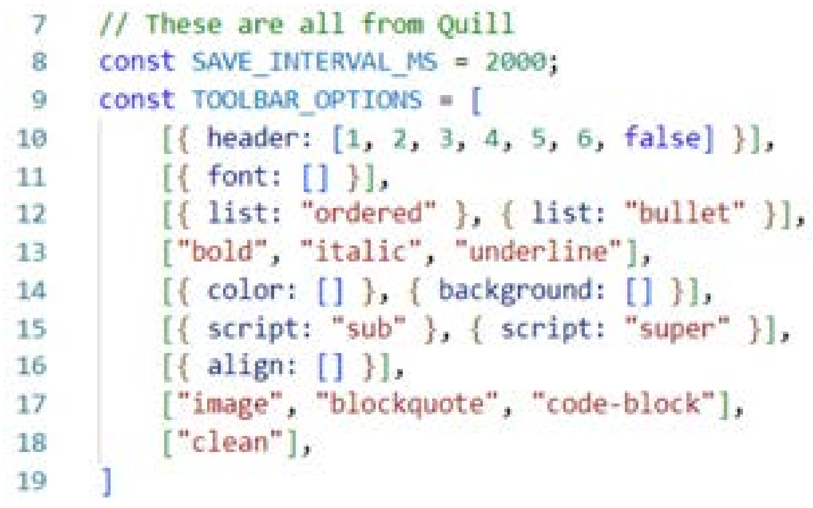
Figure 76 then shows how the extraction of the text is handled with the “handleExtract” method. The progress label is initially set to 0, then “starting” when it begins extracting text. Line 48 shows the workerRef being received in its current state, then loading the worker and setting the worker’s language to English in line 50 and 51.

Line 54 then shows the response after the worker recognizes the imageData variable. Once this is complete the result is set in line 57 and console logged in line 58.

5.8.3 Item 2 – Notes Taking Prototype

The prototype for the notes taking section of the application uses a library called “Quill.js”. The tutorial for this prototype was found on YouTube, and can be seen here: (Simplified, 2021)

The goal of this item was to develop a notes-taking prototype with a working backend that connects to MongoDB. This database uses a prototype database and was not used in the final product, when all the prototypes are connected to create one application.

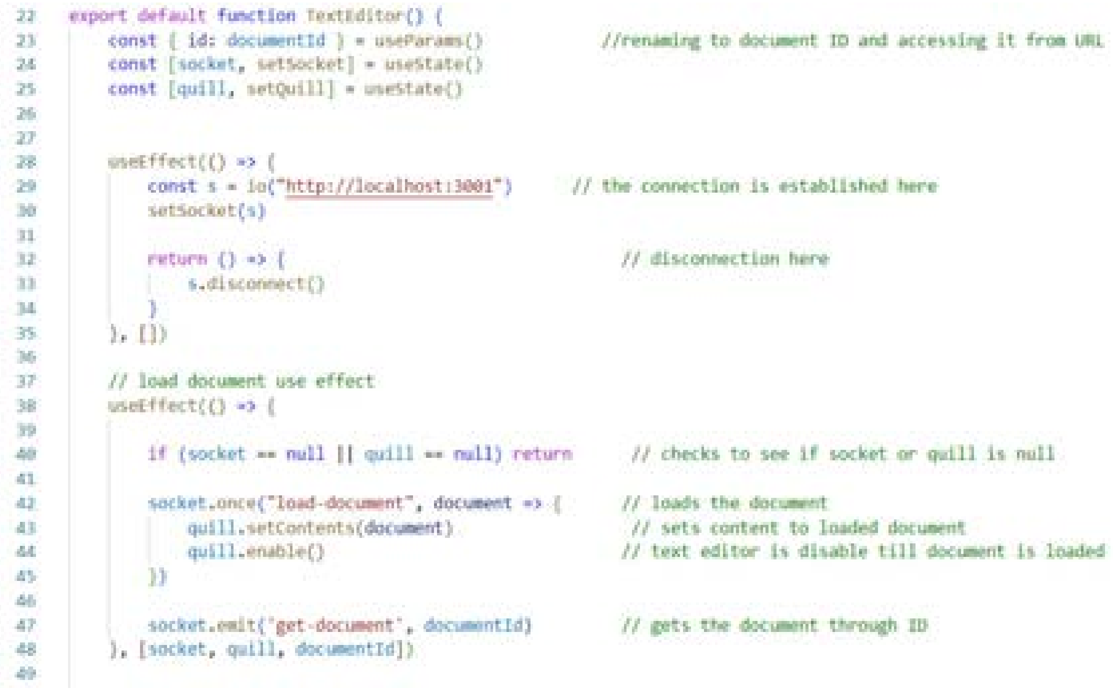


```
 7  // These are all from Quill
 8  const SAVE_INTERVAL_MS = 2000;
 9  const TOOLBAR_OPTIONS = [
10      [{ header: [1, 2, 3, 4, 5, 6, false] }],
11      [{ font: [] }],
12      [{ list: "ordered" }, { list: "bullet" }],
13      ["bold", "italic", "underline"],
14      [{ color: [] }, { background: [] }],
15      [{ script: "sub" }, { script: "super" }],
16      [{ align: [] }],
17      ["image", "blockquote", "code-block"],
18      ["clean"],
19  ]
```

Figure 77 Notes Taking Prototype (1)

Figure 77 displays the toolbar from Quill. Line 9 to 19 shows the different options that the user can use, and all come from Quill.js, and stores it in a variable called “Toolbar Options”.

Line 8 just stores the save interval, saving every 2 seconds.



```
22  export default function TextEditor() {
23      const { id: documentId } = useParams()          //renaming to document ID and accessing it from URL
24      const [socket, setSocket] = useState()
25      const [quill, setQuill] = useState()
26
27
28      useEffect(() => {
29          const s = io("http://localhost:3001")      // the connection is established here
30          setSocket(s)
31
32          return () => {                              // disconnection here
33              s.disconnect()
34          }
35      }, [])
36
37      // load document use effect
38      useEffect(() => {
39
40          if (socket == null || quill == null) return      // checks to see if socket or quill is null
41
42          socket.once("load-document", document => {      // loads the document
43              quill.setContents(document)                  // sets content to loaded document
44              quill.enable()                               // text editor is disable till document is loaded
45          })
46
47          socket.emit('get-document', documentId)         // gets the document through ID
48      }, [socket, quill, documentId])
49
```

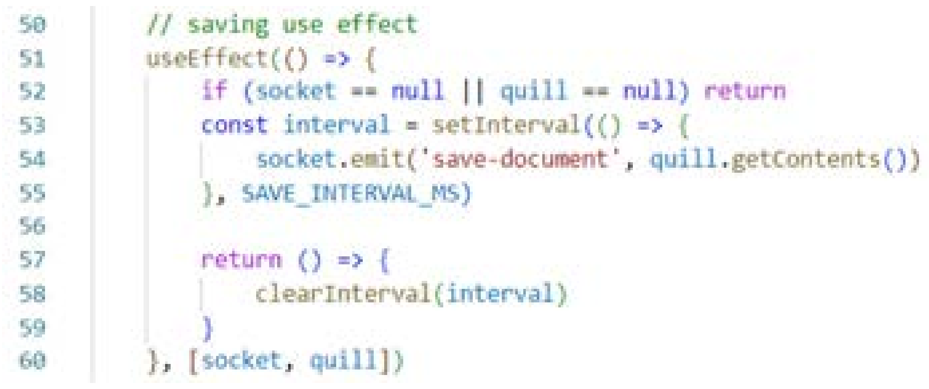
Figure 78 Notes Taking Prototype (2)

Figure 78 then shows the text Editor function, with the different use effects required to work.

Line 23 to 25 first creates variables. The Line 23 takes the document ID from the parameter of the URL. A socket and Quill variable is then set with a useState.

The useEffect in line 28 shows the connection being established, with the use of socket io to localhost 3001. The socket is set to “s” and is disconnected when it’s finished.

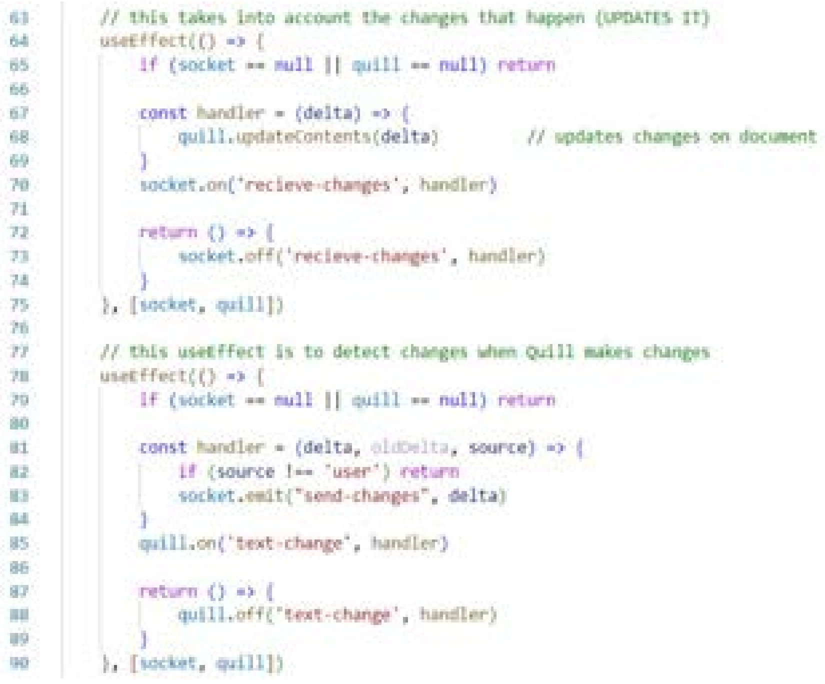
The useEffect from line 38 to 48 loads the document. It first check to see if the socket is null, and will return if it is, seen in line 40. If it is not null, the socket then loads the document and sets the contents to the document loaded. Once that is completed quill is then enabled in line 44. The socket then gets the document with the use of the document ID.



```
// saving use effect
useEffect(() => {
    if (socket == null || quill == null) return
    const interval = setInterval(() => {
        socket.emit('save-document', quill.getContents())
    }, SAVE_INTERVAL_MS)

    return () => {
        clearInterval(interval)
    }
}, [socket, quill])
```

Figure 79 Notes Taking Prototype (3)

Figure 79 shows the use effect that saves the document. It first checks to see if the socket is null and return if it is. The setInterval function then runs the save-document function every two seconds, coming from the “SAVE_INTERVAL_MS” variable set earlier in the page.

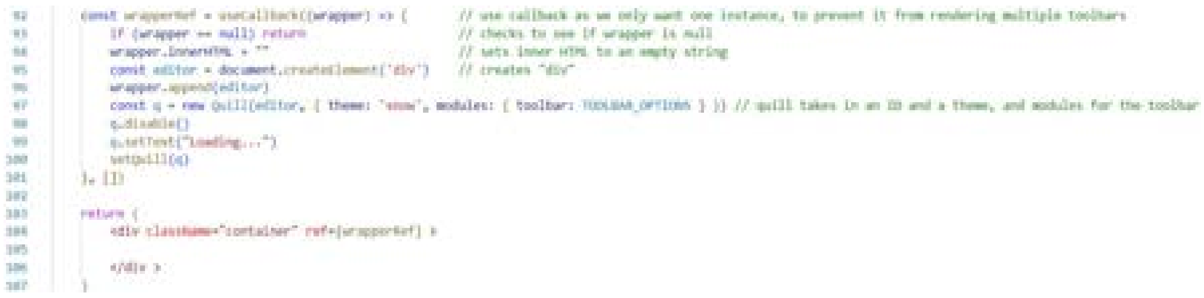


```
// this takes into account the changes that happen (UPDATES IT)
useEffect(() => {
    if (socket == null || quill == null) return

    const handler = (delta) => {
        quill.updateContents(delta)         // updates changes on document
    }
    socket.on('recieve-changes', handler)

    return () => {
        socket.off('recieve-changes', handler)
    }
}, [socket, quill])

// this useEffect is to detect changes when Quill makes changes
useEffect(() => {
    if (socket == null || quill == null) return

    const handler = (delta, oldDelta, source) => {
        if (source !== 'user') return
        socket.emit("send-changes", delta)
    }
    quill.on('text-change', handler)

    return () => {
        quill.off('text-change', handler)
    }
}, [socket, quill])
```

Figure 80 Notes Taking Prototype (4)

Figure 80 then shows the updates for changes being made on the document. The first useEffect will listen for changes being made by other users and presents the updates that those users perform. The delta is then applied to quill using the updateContents method. Delta simply stores the position and styling of the typography being used in the document.

The second useEffect listens for the changes being made by the current user. The handler function is called whenever a change in the document is made, and the delta variables are to show the changes made, the old version and who made the changes.



```
92      const wrapperRef = useCallback((wrapper) => {       // use callback as we only want one instance, to prevent it from rendering multiple toolbars
93          if (wrapper == null) return                     // checks to see if wrapper is null
94          wrapper.innerHTML = ""                          // sets inner HTML to an empty string
95          const editor = document.createElement('div')    // creates "div"
96          wrapper.append(editor)
97          const q = new Quill(editor, { theme: 'snow', modules: { toolbar: TOOLBAR_OPTIONS } }) // quill takes in an ID and a theme, and modules for the toolbar
98          q.disable()
99          q.setText("Loading...")
100         setQuill(q)
101     }, [])
102
103     return (
104         <div className="container" ref={wrapperRef} >
105
106         </div >
107     )
```

Figure 81 Notes Taking Prototype (5)

Figure 81 creates a function called "wrapperRef", to render out one toolbar. The wrapper first checks to see if it is null, then sets the inner HTML as an empty string. A div is then created and appends a new quill editor. The new Quill editor takes in the theme, and the toolbar.

The function will initially be disabled, seen in line 98, but will enable once it finishes loading.

The server side of the application was also built during this prototype. It is stored in MongoDB and is written in express.js. The structure of the code consists of the controller, the models, the routes, and the server. This can be seen in figure 82, the figure below.

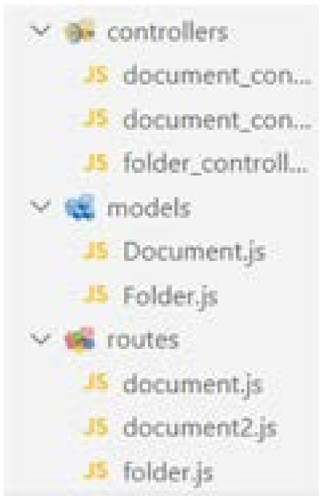
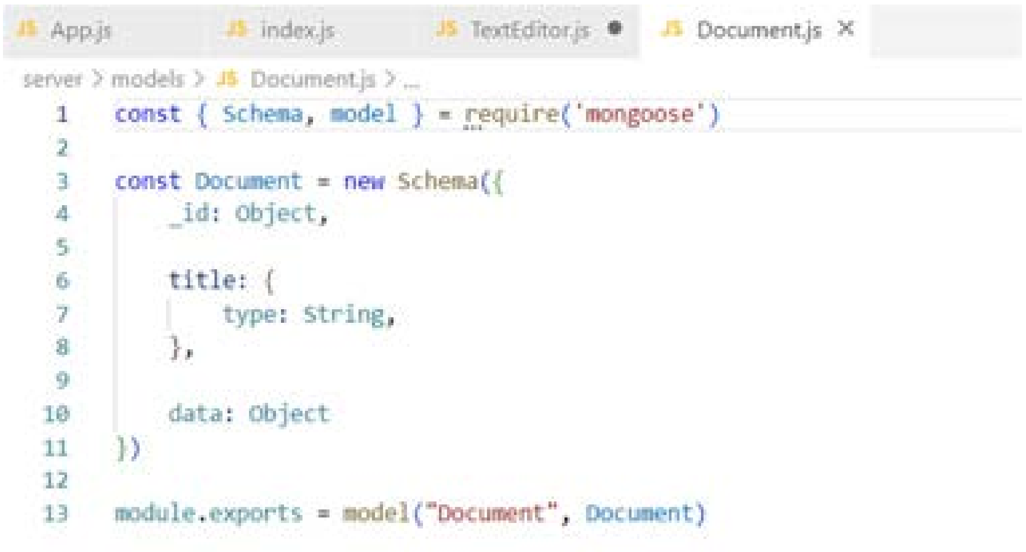


Figure 82 Backend of Notes Taking Prototype

This kind structure will be found throughout the entire backend of the application and will be explained in detail in this section.

We will first start by discussing the document model.

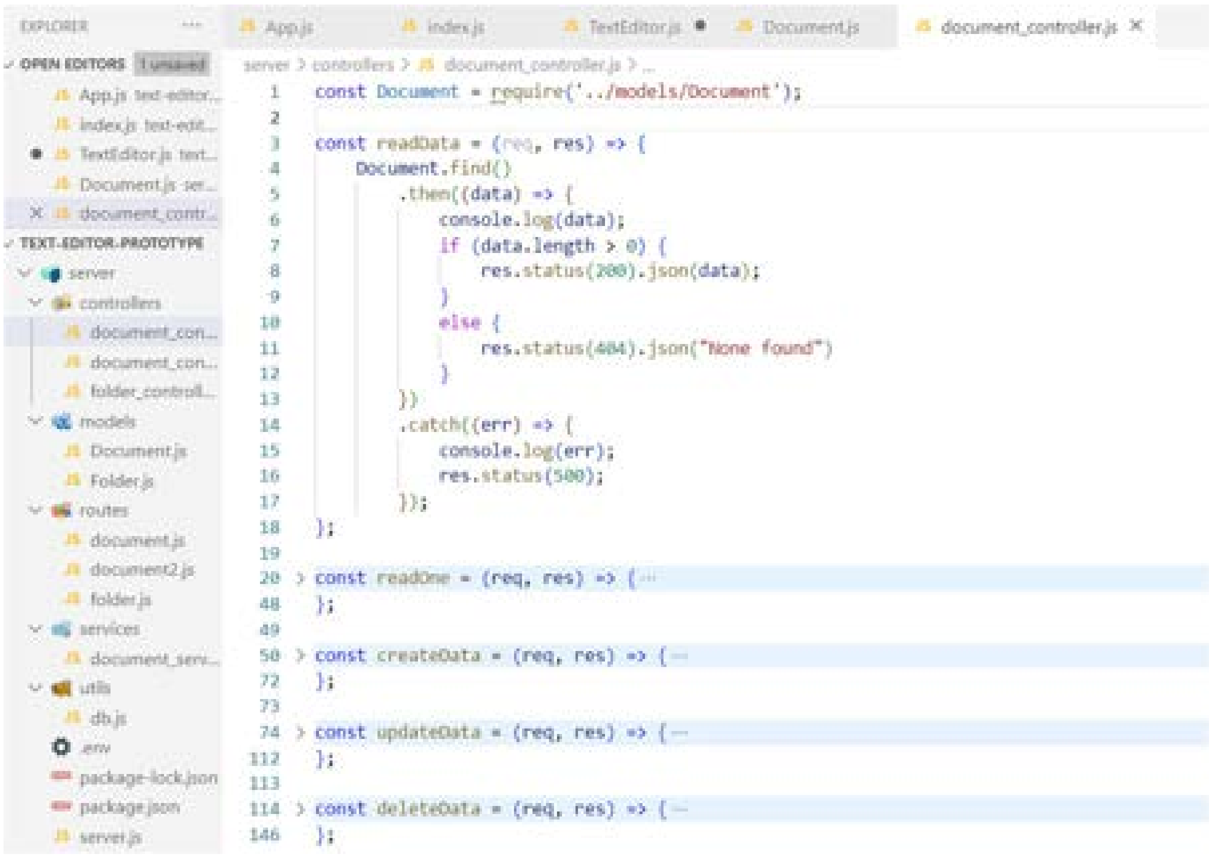


```javascript
const { Schema, model } = require('mongoose')

const Document = new Schema({
    _id: Object,

    title: {
        type: String,
    },

    data: Object
})

module.exports = model("Document", Document)
```

Figure 83 Document Schema

Figure 83 shows the document model and shows how it is supposed to be stored in the database, consisting of an ID, a title, and the data, which is the quill.js document files being stored.

It then exports the model as “Document” and is used in the document_controller.js file.

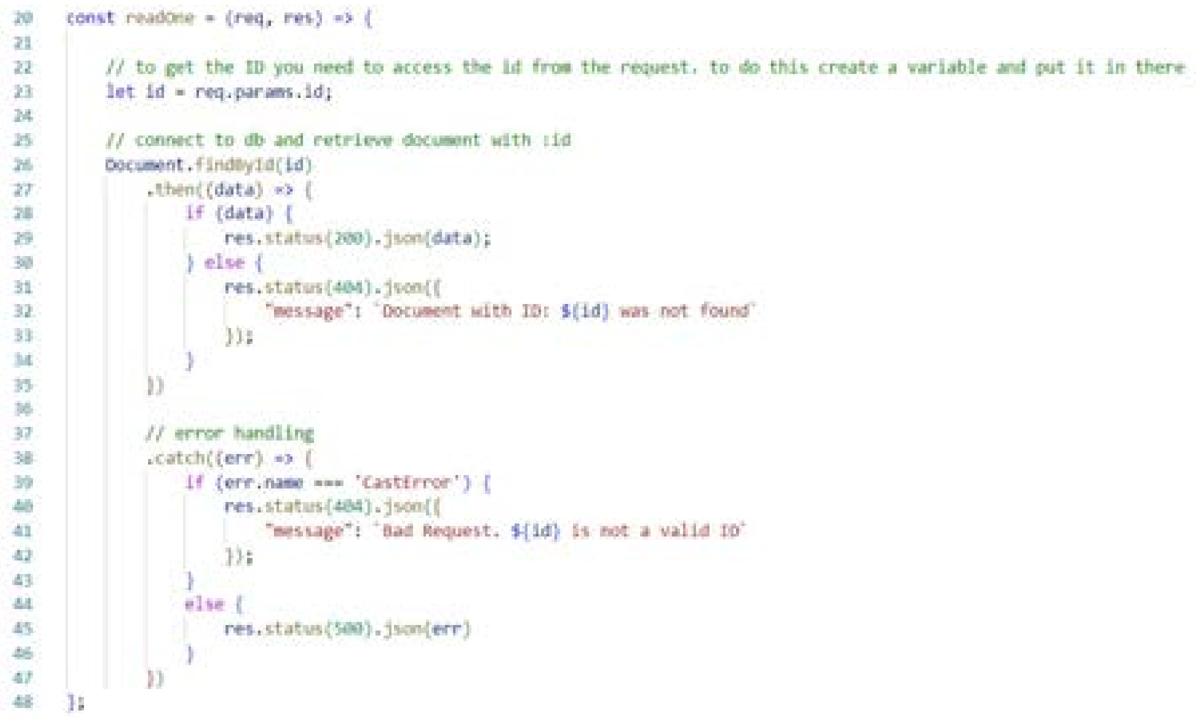


```javascript
const Document = require('../models/Document');

const readData = (req, res) => {
    Document.find()
        .then((data) => {
            console.log(data);
            if (data.length > 0) {
                res.status(200).json(data);
            }
            else {
                res.status(404).json("None found")
            }
        })
        .catch((err) => {
            console.log(err);
            res.status(500);
        });
};

const readOne = (req, res) => {···
};

const createData = (req, res) => {···
};

const updateData = (req, res) => {···
};

const deleteData = (req, res) => {···
};
```

Figure 84 Document Controller

Figure 84 shows the document controller and shows the different CRUD functionalities that is being used, although this will be changed in a later section to suit the folders feature of the application.

The readData function seen in figure 84 will return all the documents that is in the database. The find() method is from MongoDB and will find all the data and return it to the user. The if statement provided is for error handling in case no data was available.



```
const readOne = (req, res) => {

    // to get the ID you need to access the id from the request. to do this create a variable and put it in there
    let id = req.params.id;

    // connect to db and retrieve document with :id
    Document.findById(id)
        .then((data) => {
            if (data) {
                res.status(200).json(data);
            } else {
                res.status(404).json({
                    "message": `Document with ID: ${id} was not found`
                });
            }
        })

        // error handling
        .catch((err) => {
            if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
            else {
                res.status(500).json(err)
            }
        })
};
```

Figure 85 Document Read One Method

Figure 85 then shows the readOne method. It first requests for the ID of the document from the params, then it finds the id using the findByID method, passing the id as a parameter. It then checks to see if the data is available, where it returns “Document with ID: ${id} was not found” if that document doesn’t exist, and simply returns the document if it finds it.

A catch is then put in place as a method of error handling, to check if the ID is valid or not.

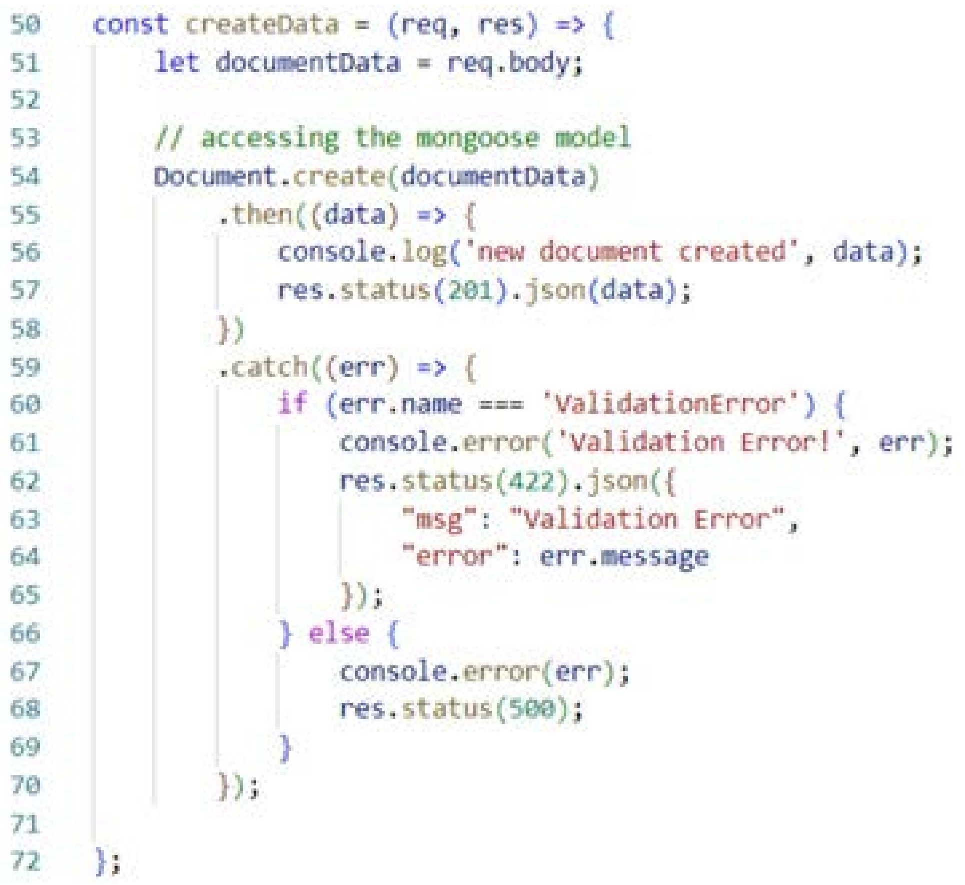
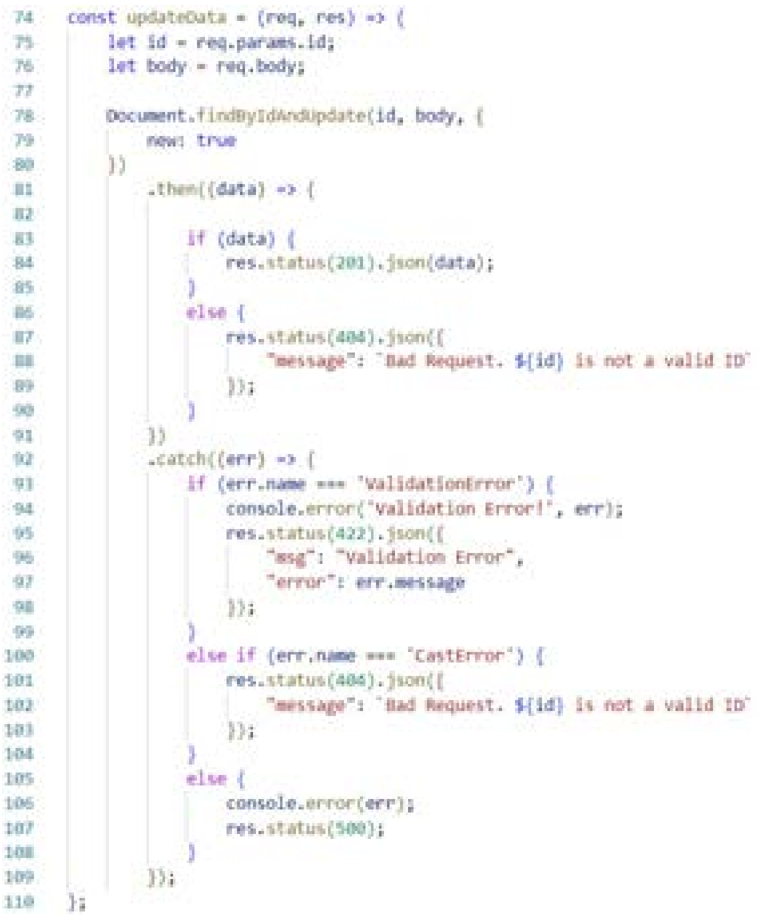

```
const createData = (req, res) => {
    let documentData = req.body;

    // accessing the mongoose model
    Document.create(documentData)
        .then((data) => {
            console.log('new document created', data);
            res.status(201).json(data);
        })
        .catch((err) => {
            if (err.name === 'ValidationError') {
                console.error('Validation Error!', err);
                res.status(422).json({
                    "msg": "Validation Error",
                    "error": err.message
                });
            } else {
                console.error(err);
                res.status(500);
            }
        });

};
```

Figure 86 Document Create Data Method

Figure 86 shows the create data method. This method allows users to create new documents. It first requests for the body, then accesses the mongoose model and uses the create() method from MongoDB and passes the data as a parameter.

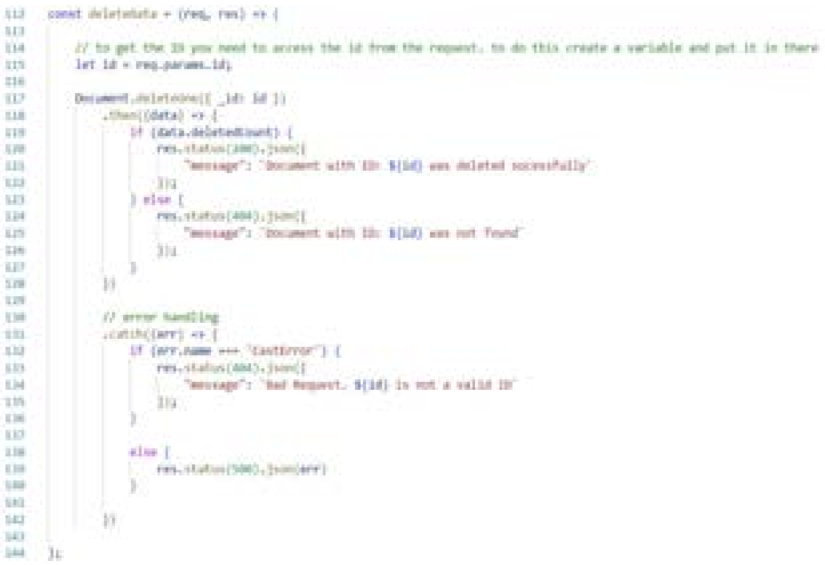
It then checks to see if it is successful and returns the data with a status of 201. If it is not successful, an error is thrown to the user.



```
const updateData = (req, res) => {
    let id = req.params.id;
    let body = req.body;

    Document.findByIdAndUpdate(id, body, {
        new: true
    })
        .then((data) => {

            if (data) {
                res.status(201).json(data);
            }
            else {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
        })
        .catch((err) => {
            if (err.name === 'ValidationError') {
                console.error('Validation Error!', err);
                res.status(422).json({
                    "msg": "Validation Error",
                    "error": err.message
                });
            }
            else if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
            else {
                console.error(err);
                res.status(500);
            }
        });
};
```

Figure 87 Update Data

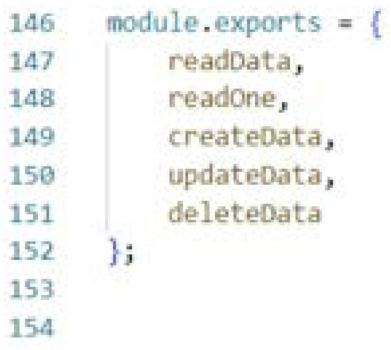
Figure 87 the shows the updateData method, which updates the document data. It first takes in the ID and the body that the user decides to put in. It then uses the findByIdAndUpdate() method to update the data, through an error if it is unsuccessful or invalid data was provided or returning it if it was successfully updated.



```
const deleteData = (req, res) => {

    // to get the ID you need to access the id from the request. to do this create a variable and put it in there
    let id = req.params.id;

    Document.deleteOne({ _id: id })
        .then((data) => {
            if (data.deletedCount) {
                res.status(200).json({
                    "message": `Document with ID: ${id} was deleted successfully`
                });
            } else {
                res.status(404).json({
                    "message": `Document with ID: ${id} was not found`
                });
            }
        })

        // error handling
        .catch((err) => {
            if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }

            else {
                res.status(500).json(err)
            }

        })

};
```

Figure 88 Delete Data

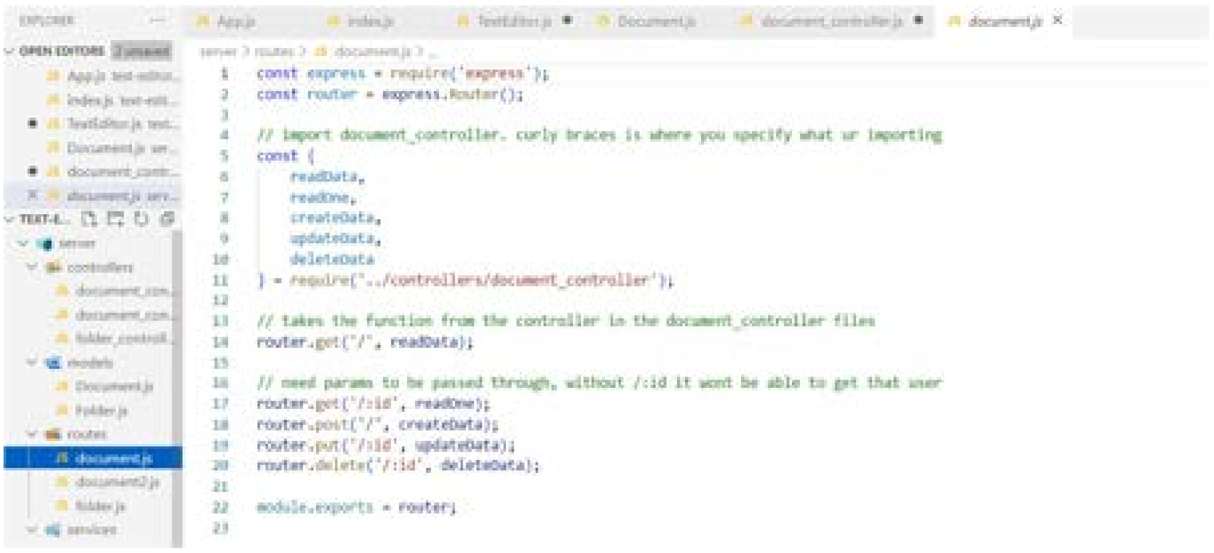
Finally, figure 88 shows the delete data method, where it takes in the ID that the user wants to delete, and uses the deleteOne() method to delete it from the database, again throwing an error if it is unsuccessful, and showing a message “Document with ID: ${id} was not deleted”, and if it is successfully deleted it returns with a status 200 and a message saying it was successfully deleted.



```
module.exports = {
    readData,
    readOne,
    createData,
    updateData,
    deleteData
};
```

Figure 89 Exports of Functions

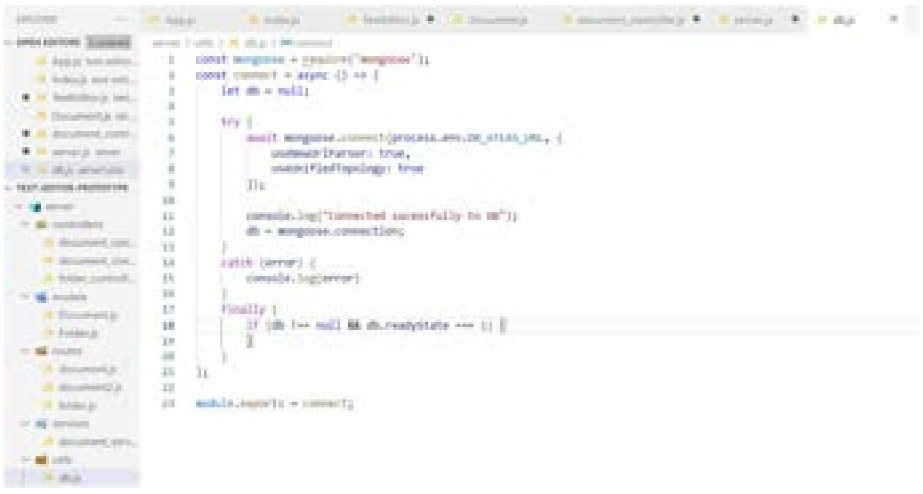
Figure 89 then shows the functions being exported, and it is then use in the routes folder.



```
const express = require('express');
const router = express.Router();

// import document_controller. curly braces is where you specify what ur importing
const {
    readData,
    readOne,
    createData,
    updateData,
    deleteData
} = require('../controllers/document_controller');

// takes the function from the controller in the document_controller files
router.get('/', readData);

// need params to be passed through, without /:id it wont be able to get that user
router.get('/:id', readOne);
router.post('/', createData);
router.put('/:id', updateData);
router.delete('/:id', deleteData);

module.exports = router;
```

Figure 90 Routes

Figure 90 shows the routes file and uses the functions that were exported from the models file and matches the function with the route of the request, for example router.get(‘/:id’, readOne) gets the id of the URL and uses the readOne method. The router is then exported. To the server to be used.

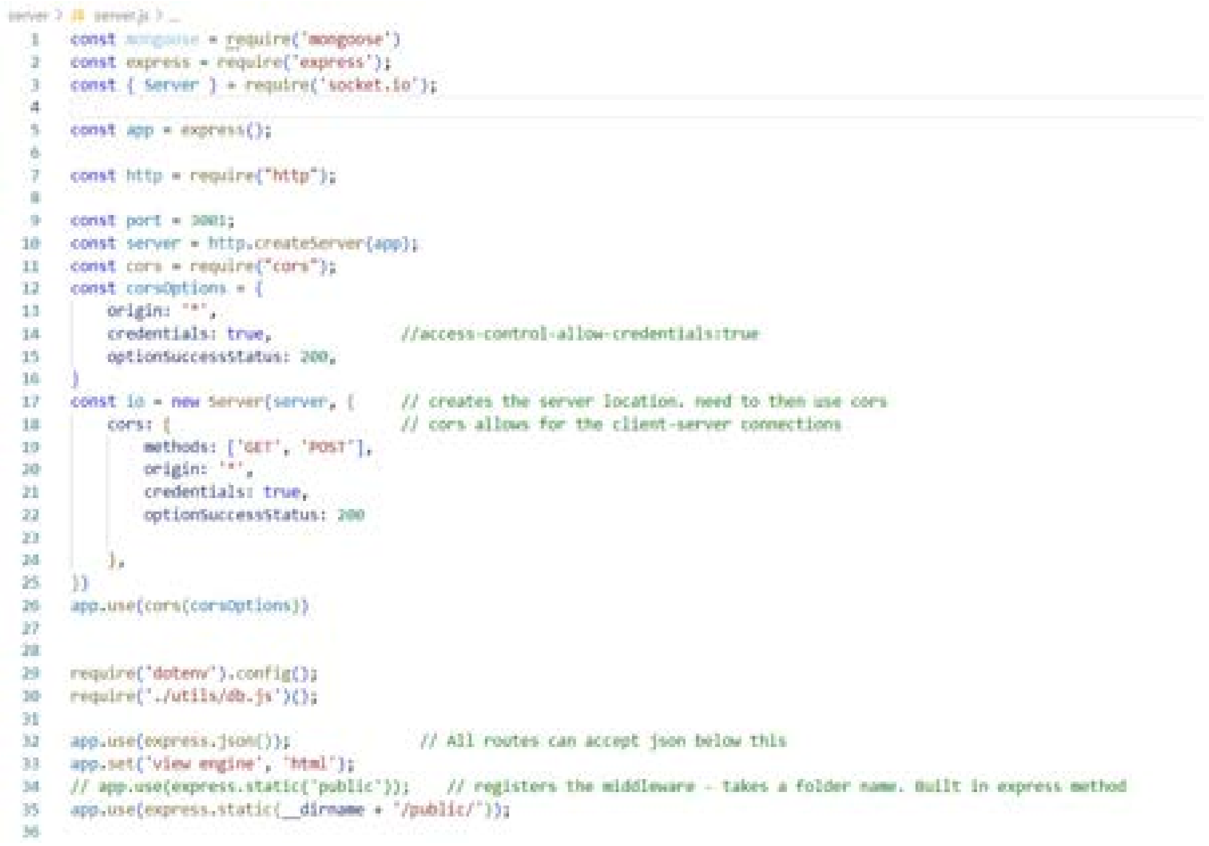


```
const mongoose = require('mongoose');
const connect = async () => {
    let db = null;

    try {
        await mongoose.connect(process.env.DB_ATLAS_URL, {
            useNewUrlParser: true,
            useUnifiedTopology: true
        });

        console.log("Connected successfully to DB");
        db = mongoose.connection;
    }
    catch (error) {
        console.log(error)
    }
    finally {
        if (db !== null && db.readyState === 1) {
        }
    }
};

module.exports = connect;
```

Figure 91 db.js

In the figure above, it shows the Utils folder, where the db.js file is stored. The connection is established to the MongoDB database here using an environment variable that stores the URL called "DB_ATLAS_URL".

The server.js file then uses all the components together to run the server for the application.

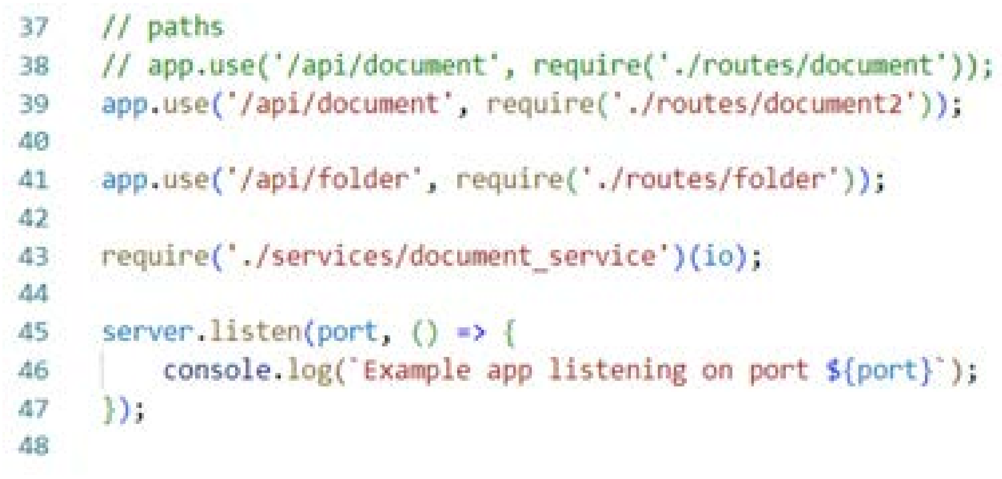


```
const mongoose = require('mongoose')
const express = require('express');
const { Server } = require('socket.io');

const app = express();

const http = require("http");

const port = 3001;
const server = http.createServer(app);
const cors = require("cors");
const corsOptions = {
    origin: "*",
    credentials: true,            //access-control-allow-credentials:true
    optionSuccessStatus: 200,
}
const io = new Server(server, {     // creates the server location. need to then use cors
    cors: {                         // cors allows for the client-server connections
        methods: ['GET', 'POST'],
        origin: '*',
        credentials: true,
        optionSuccessStatus: 200

    },
})
app.use(cors(corsOptions))


require('dotenv').config();
require('./utils/db.js')();

app.use(express.json());              // All routes can accept json below this
app.set('view engine', 'html');
// app.use(express.static('public'));    // registers the middleware - takes a folder name. Built in express method
app.use(express.static(__dirname + '/public/'));

```

Figure 92 Server.js

The figure above shows the connections being created and requiring all the different utilities needed for the server to run.

We first require mongoose, express, and socket io. Then the server is then created with the port of 3001, as well as the socket.io server. It then requires the dotenv file and the db.js file for the database. The server then is changed to accept json and sets the view to html.



```
// paths
// app.use('/api/document', require('./routes/document'));
app.use('/api/document', require('./routes/document2'));

app.use('/api/folder', require('./routes/folder'));

require('./services/document_service')(io);

server.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
});

```

Figure 93 Server Port

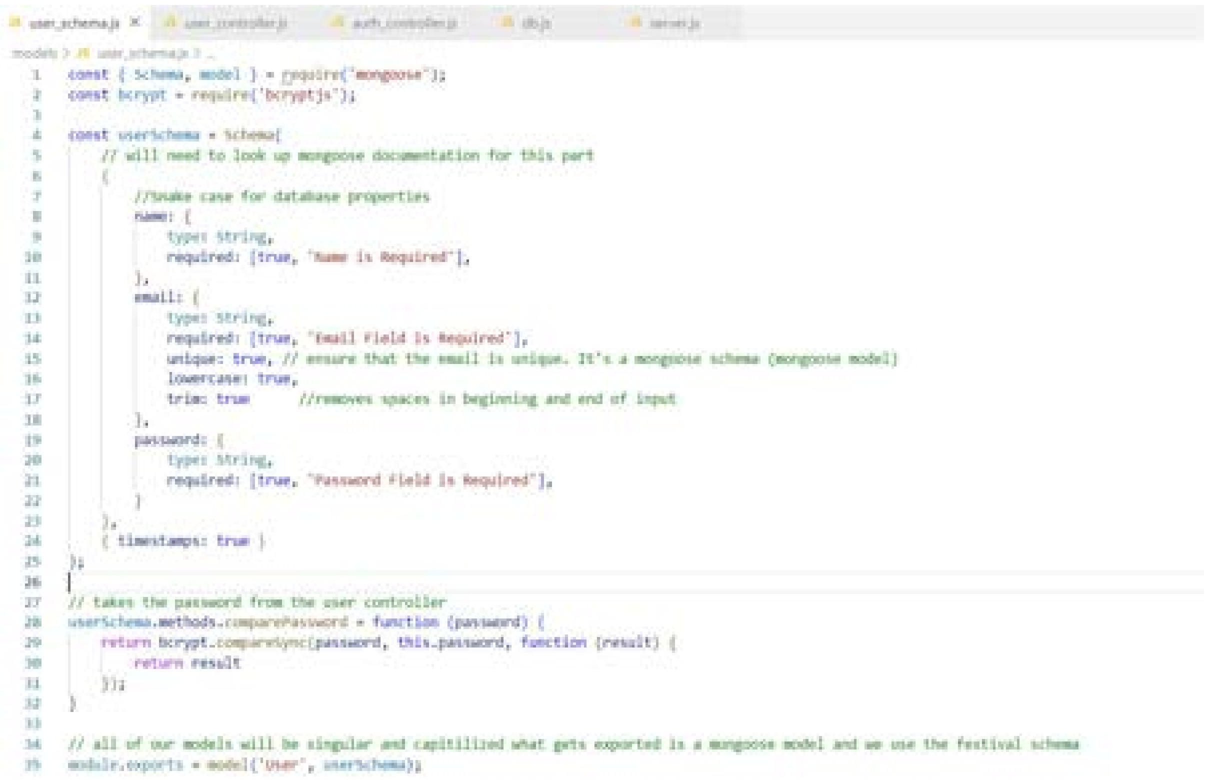
Figure 93 finally shows the paths that is being used and console logs the server port. The services folder will be discussed in a later section.

5.8.4 Item 3 – User Register and Login Prototype

Item 3 was to develop a prototype that allowed users to register and login to the application.

The development of the user's register and login were very similar to the document CRUD functionalities, with some changes.

Like the document, the user contained a schema, a controller, a route, and was all placed into the server.js.

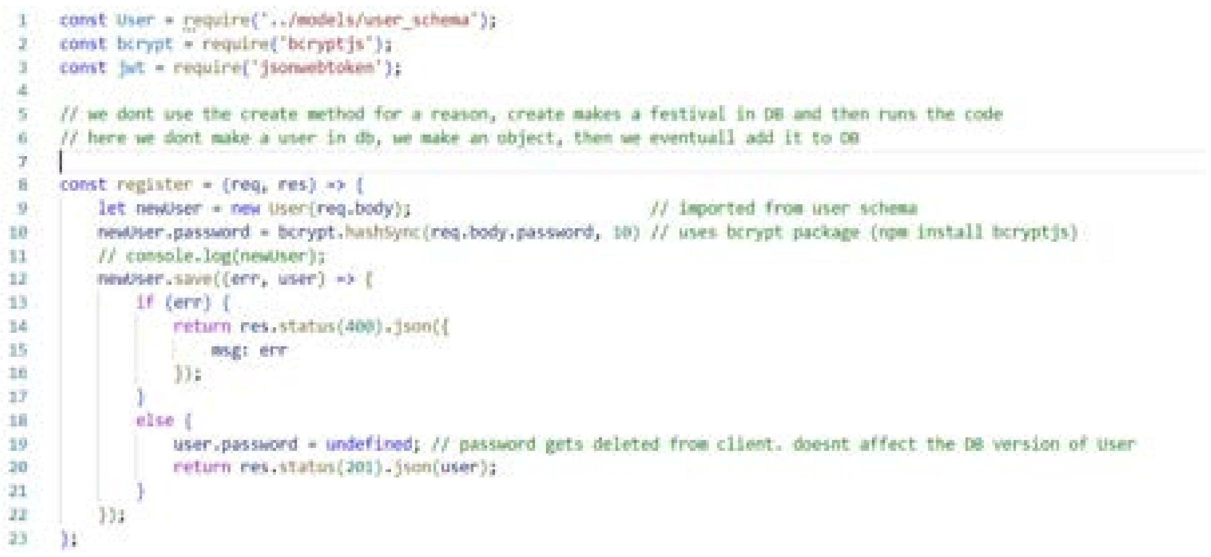


```
1   const { Schema, model } = require('mongoose');
2   const bcrypt = require('bcryptjs');
3
4   const userSchema = Schema(
5       // will need to look up mongoose documentation for this part
6       {
7           //Snake case for database properties
8           name: {
9               type: String,
10              required: [true, 'Name is Required'],
11          },
12          email: {
13              type: String,
14              required: [true, "Email Field Is Required"],
15              unique: true, // ensure that the email is unique. It's a mongoose schema (mongoose model)
16              lowercase: true,
17              trim: true      //removes spaces in beginning and end of input
18          },
19          password: {
20              type: String,
21              required: [true, "Password Field Is Required"],
22          }
23      },
24      { timestamps: true }
25  );
26
27  // takes the password from the user controller
28  userSchema.methods.comparePassword = function (password) {
29      return bcrypt.compareSync(password, this.password, function (result) {
30          return result
31      });
32  }
33
34  // all of our models will be singular and capitilized what gets exported is a mongoose model and we use the festival schema
35  module.exports = model('User', userSchema);
```

Figure 94 User Schema

Figure 94 first shows the user schema, the user will need to input their name, email and password in the schema, and the password becomes encrypted with the use of the function seen in line 28 to 32.

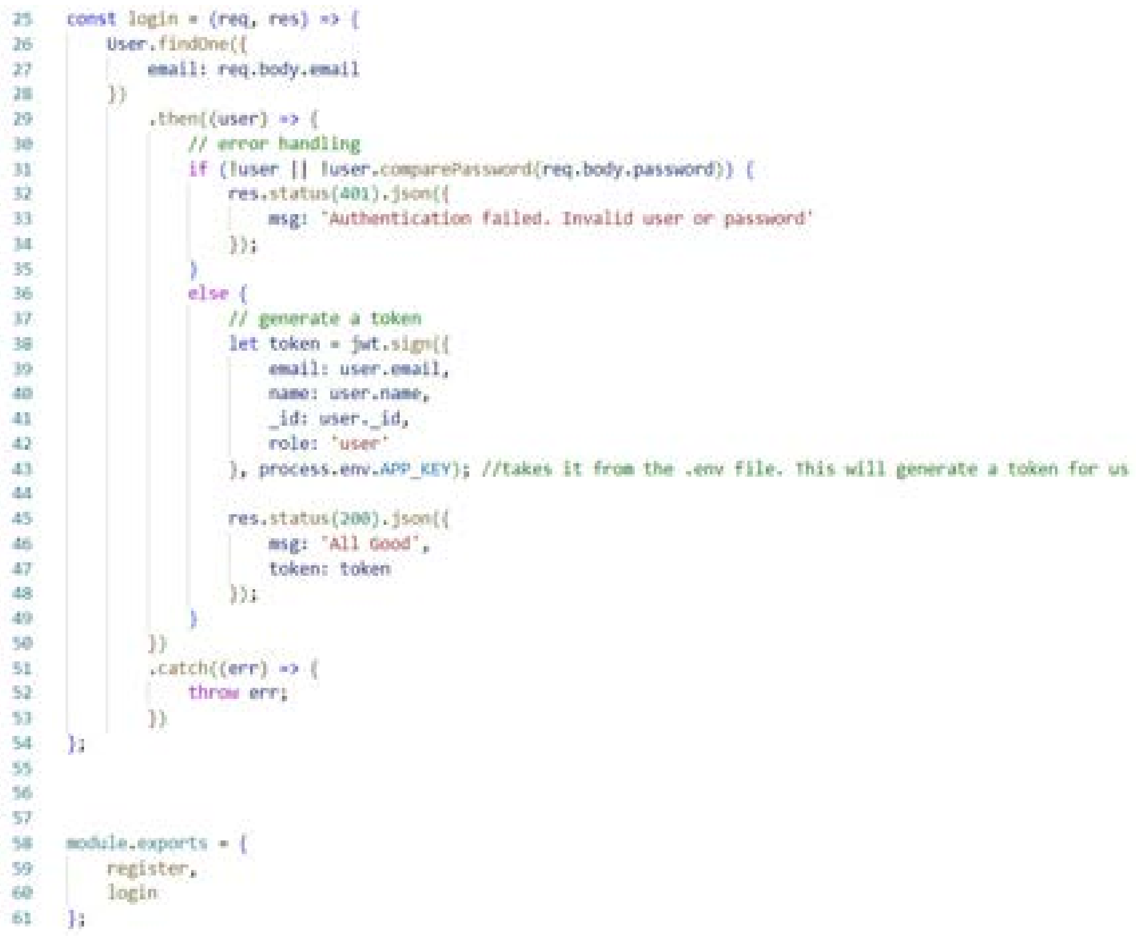
The schema is then exported as “User”.



```
const User = require('../models/user_schema');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

// we dont use the create method for a reason, create makes a festival in DB and then runs the code
// here we dont make a user in db, we make an object, then we eventuall add it to DB

const register = (req, res) => {
    let newUser = new User(req.body);                       // imported from user schema
    newUser.password = bcrypt.hashSync(req.body.password, 10) // uses bcrypt package (npm install bcryptjs)
    // console.log(newUser);
    newUser.save((err, user) => {
        if (err) {
            return res.status(400).json({
                msg: err
            });
        }
        else {
            user.password = undefined; // password gets deleted from client. doesnt affect the DB version of User
            return res.status(201).json(user);
        }
    });
};
```

Figure 95 User Controller Register Function

The user controller is then seen in figure 95, where it displays the register method. It is like the create method, however it encrypts the user password with the use of a library called Bcrypt.js and saves the user into the database.

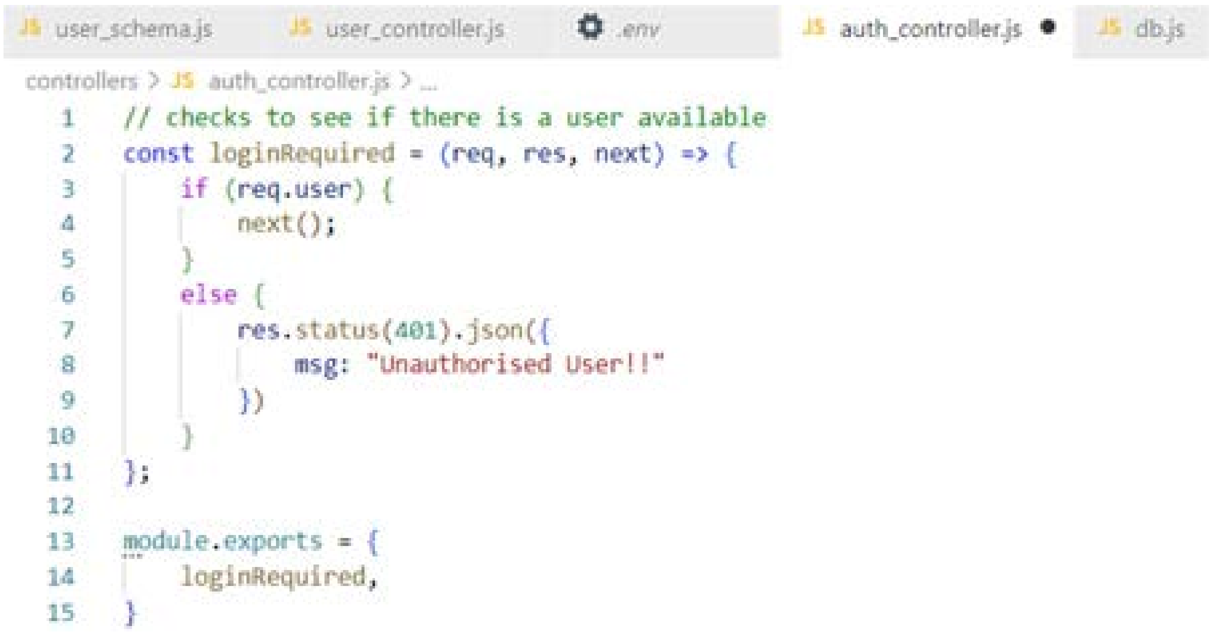


```
const login = (req, res) => {
    User.findOne({
        email: req.body.email
    })
        .then((user) => {
            // error handling
            if (!user || !user.comparePassword(req.body.password)) {
                res.status(401).json({
                    msg: 'Authentication failed. Invalid user or password'
                });
            }
            else {
                // generate a token
                let token = jwt.sign({
                    email: user.email,
                    name: user.name,
                    _id: user._id,
                    role: 'user'
                }, process.env.APP_KEY); //takes it from the .env file. This will generate a token for us

                res.status(200).json({
                    msg: 'All Good',
                    token: token
                });
            }
        })
        .catch((err) => {
            throw err;
        })
};



module.exports = {
    register,
    login
};
```

Figure 96 Login Function

Figure 96 then shows the login method and what is being exported.

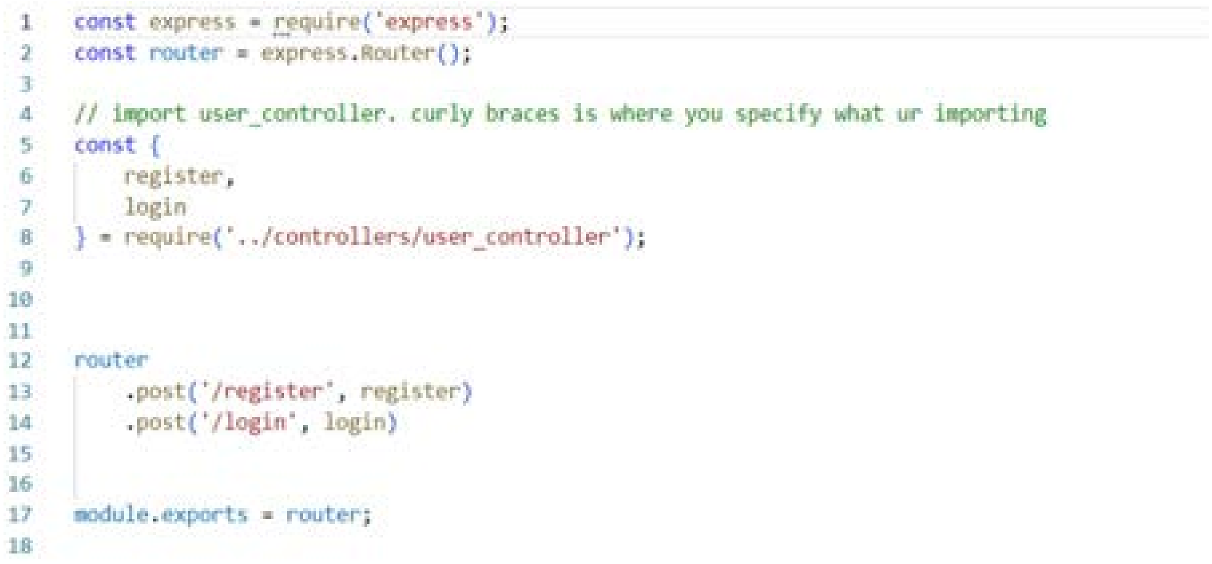
The login will first check to see the user's email address. It will then check to see if the user password is the same as the password that was stored in the database, or if that user exists. If it doesn't it returns an error, otherwise it will generate the token with the use of JSON Web Token (JWT), and sign the user in. The secret key from JWT is the process.env.APP_KEY. It will then return a status 200 if it works.



```javascript
// checks to see if there is a user available
const loginRequired = (req, res, next) => {
    if (req.user) {
        next();
    }
    else {
        res.status(401).json({
            msg: "Unauthorised User!!"
        })
    }
};

module.exports = {
    loginRequired,
}
```

Figure 97 Login Required Function

Figure 97 then shows the authorization of the application. The login required is to ensure that a user is logged in before proceeding with the rest of the application.

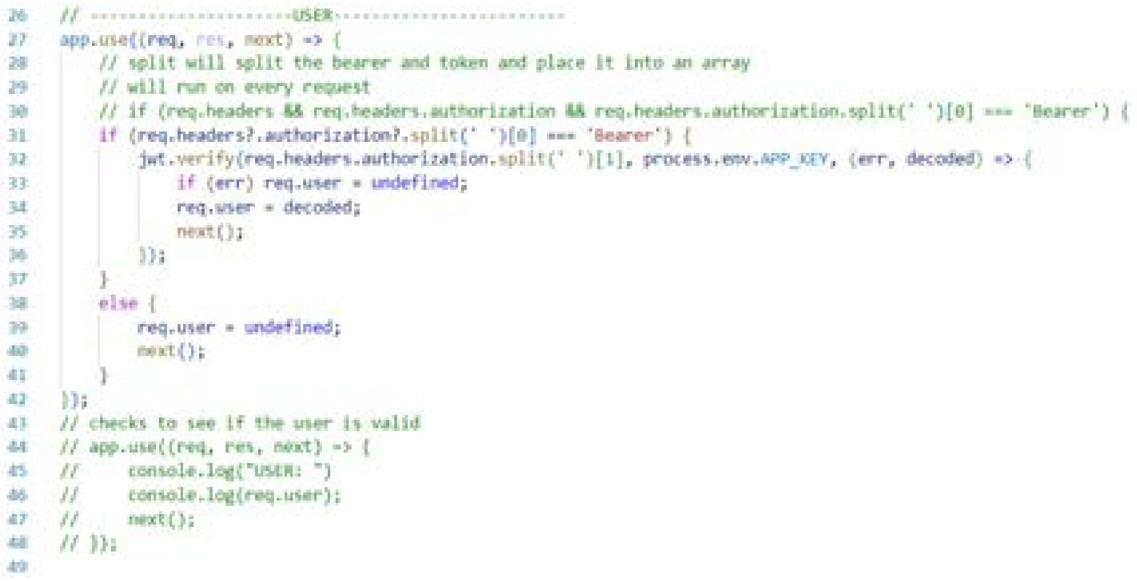


```javascript
const express = require('express');
const router = express.Router();

// import user_controller. curly braces is where you specify what ur importing
const {
    register,
    login
} = require('../controllers/user_controller');



router
    .post('/register', register)
    .post('/login', login)


module.exports = router;

```

Figure 98 User Routes

Figure 98 then shows the route of the user, and what it can do. Here it takes in a register and login.

Finally, the server.js is mostly the same, however it adds a middleware that can be seen in figure 99.



```
// ----------------------USER-----------------------
app.use((req, res, next) => {
    // split will split the bearer and token and place it into an array
    // will run on every request
    // if (req.headers && req.headers.authorization && req.headers.authorization.split(' ')[0] === 'Bearer') {
    if (req.headers?.authorization?.split(' ')[0] === 'Bearer') {
        jwt.verify(req.headers.authorization.split(' ')[1], process.env.APP_KEY, (err, decoded) => {
            if (err) req.user = undefined;
            req.user = decoded;
            next();
        });
    }
    else {
        req.user = undefined;
        next();
    }
});
// checks to see if the user is valid
// app.use((req, res, next) => {
//     console.log("USER: ")
//     console.log(req.user);
//     next();
// });
```

Figure 99 Middleware

This middleware checks for the JWT token and verifies it.

This item of the sprint allows users to login to view their profile and will be altered to show users their own specific files and documents.

5.9 Sprint 4 – Design 2

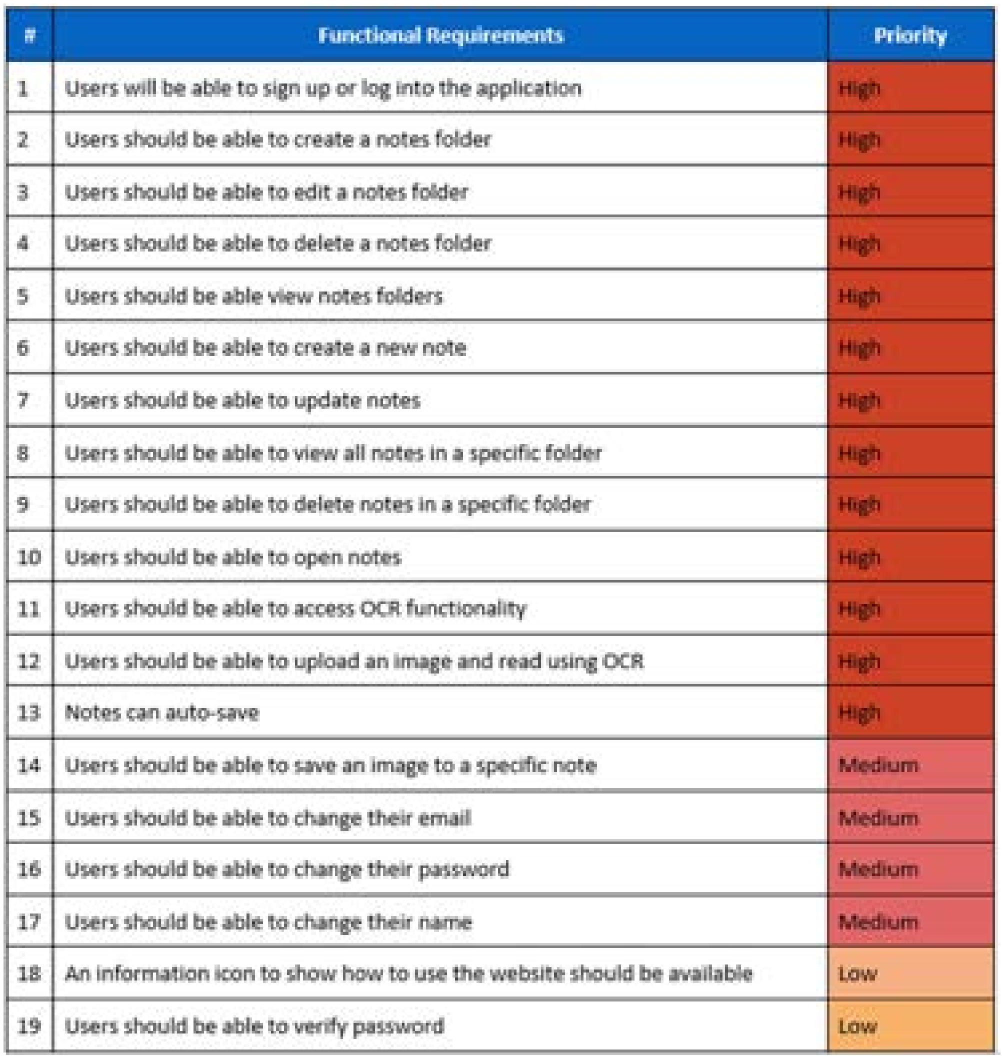
5.9.1 Goal

The goal of this sprint was to update the design section, to cater for changes that were made during the development of the application in Implementation 1. The backlog and design of the application was updated to create a more user-friendly design of the application. The items that were included in this sprint were:

- **Item 1:** Update Backlog.
- **Item 2:** Update user interface design.
- **Item 3:** Connect Prototypes (including backend).
- **Item 4:** Bug Fixes and problems.

5.9.1.1 Item 1 – Updated Backlog

The functional requirements backlog was updated slightly. Some features were removed or placed in a lower priority to focus more on the development of the more core-features of the application. The updated backlog can be seen in figure 100:



#	Functional Requirements	Priority
1	Users will be able to sign up or log into the application	High
2	Users should be able to create a notes folder	High
3	Users should be able to edit a notes folder	High
4	Users should be able to delete a notes folder	High
5	Users should be able view notes folders	High
6	Users should be able to create a new note	High
7	Users should be able to update notes	High
8	Users should be able to view all notes in a specific folder	High
9	Users should be able to delete notes in a specific folder	High
10	Users should be able to open notes	High
11	Users should be able to access OCR functionality	High
12	Users should be able to upload an image and read using OCR	High
13	Notes can auto-save	High
14	Users should be able to save an image to a specific note	Medium
15	Users should be able to change their email	Medium
16	Users should be able to change their password	Medium
17	Users should be able to change their name	Medium
18	An information icon to show how to use the website should be available	Low
19	Users should be able to verify password	Low

Figure 100 Updated Backlog

Some parts included in the backlog were already included in other features, such as "The ability to style the notes page" being included in the Quill.js library already being done. The to-do-list and share notes feature were removed to focus on other parts of the project.

Some parts of the project were also increased in priority, such as a user being able to edit their own profile, compared it initially being set to low.

5.9.1.2 Item 2 – Updated User Interface Design

The user interface of the application was also updated.

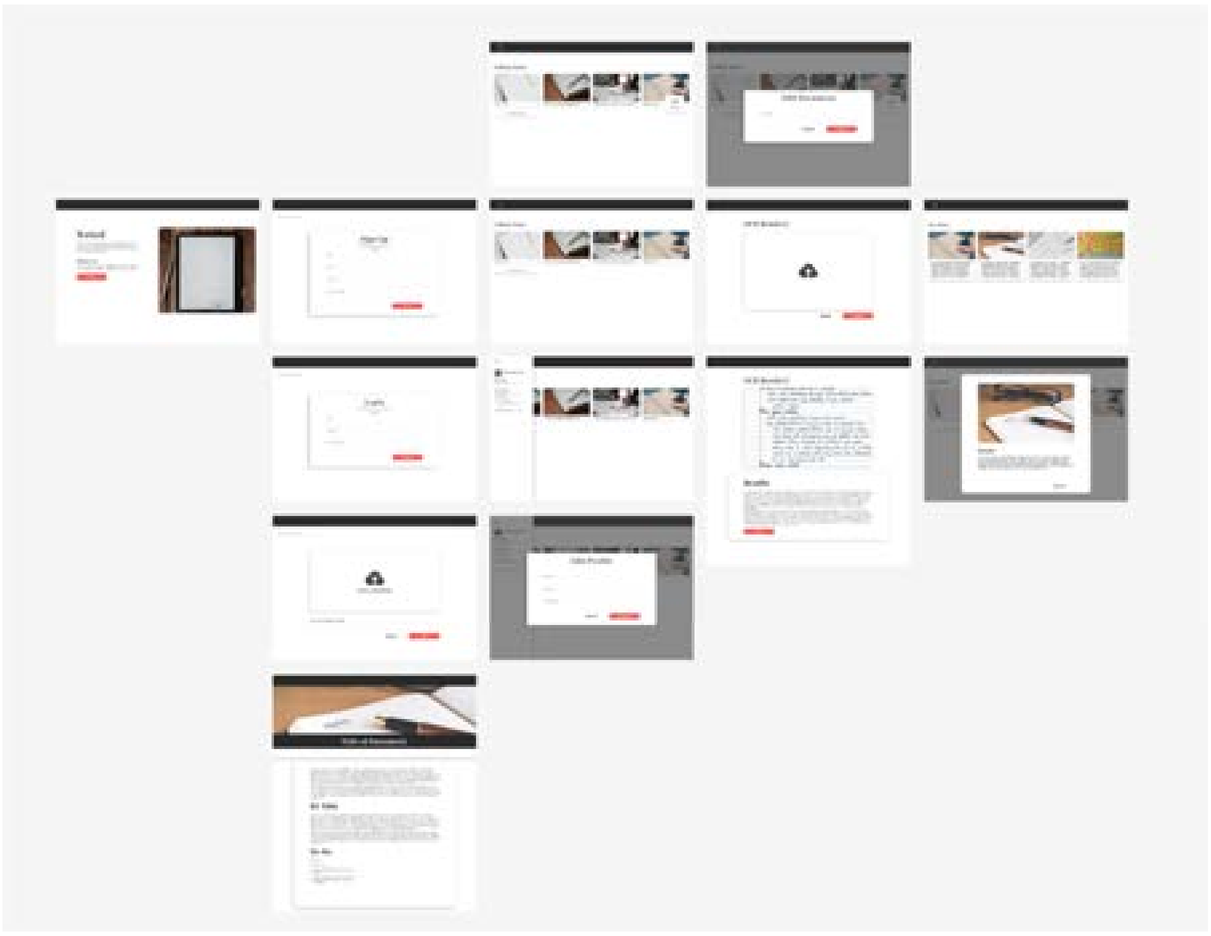


Figure 101 Updated Design

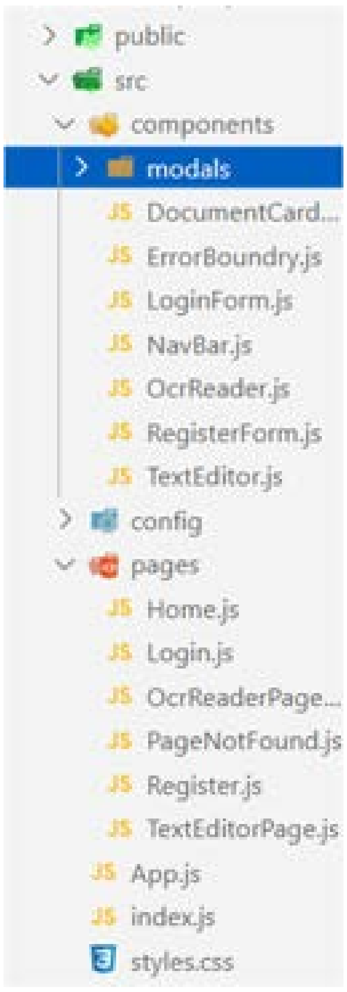
The updated design is seen in figure 101, this design includes the modals and pages that were created while developing the application. The new design contains the modals for when the user wishes to edit their profile or document.

The user can also view the files that was saved from the OCR reader in a modal and is seen in its own separate page.

The title of the document or folder can also be seen when viewing either the document or the folder.

5.9.1.3 Item 3 – Connecting All Prototypes Together

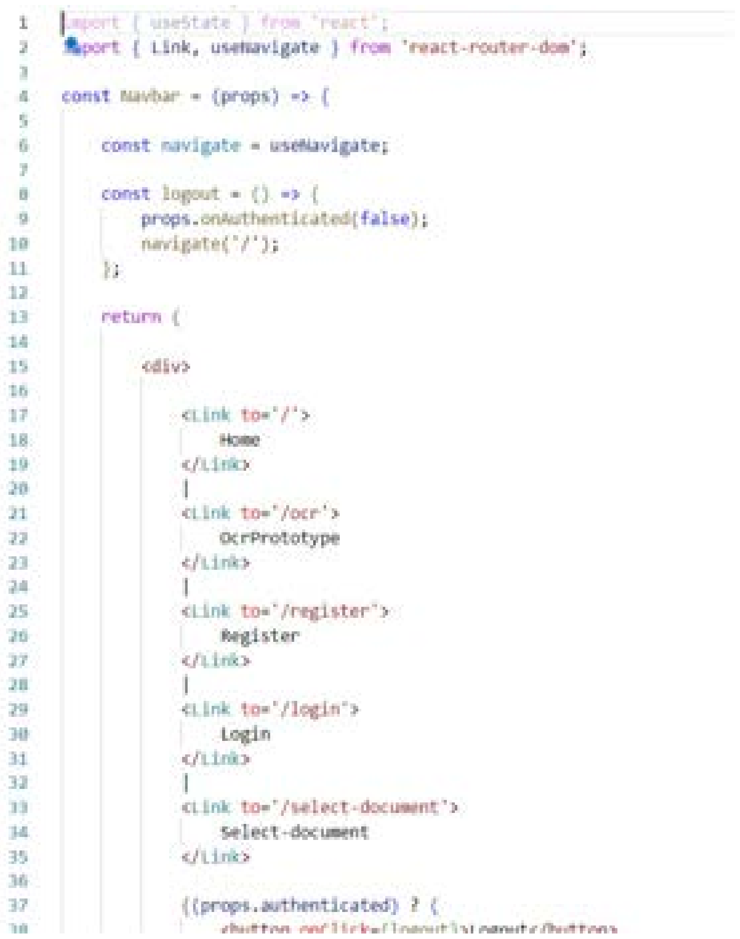
The goal of this item was to connect all the existing prototypes together to form a more unified application. This was accomplished by combining the prototypes one-by-one, starting with the login and register functionalities, then including the OCR Reader page, and finally including the document editor.



```
> public
v src
  v components
    > modals
      DocumentCard...
      ErrorBoundry.js
      LoginForm.js
      NavBar.js
      OcrReader.js
      RegisterForm.js
      TextEditor.js
  > config
  v pages
      Home.js
      Login.js
      OcrReaderPage...
      PageNotFound.js
      Register.js
      TextEditorPage.js
    App.js
    index.js
    styles.css
```

Figure 102 Combined Frontend

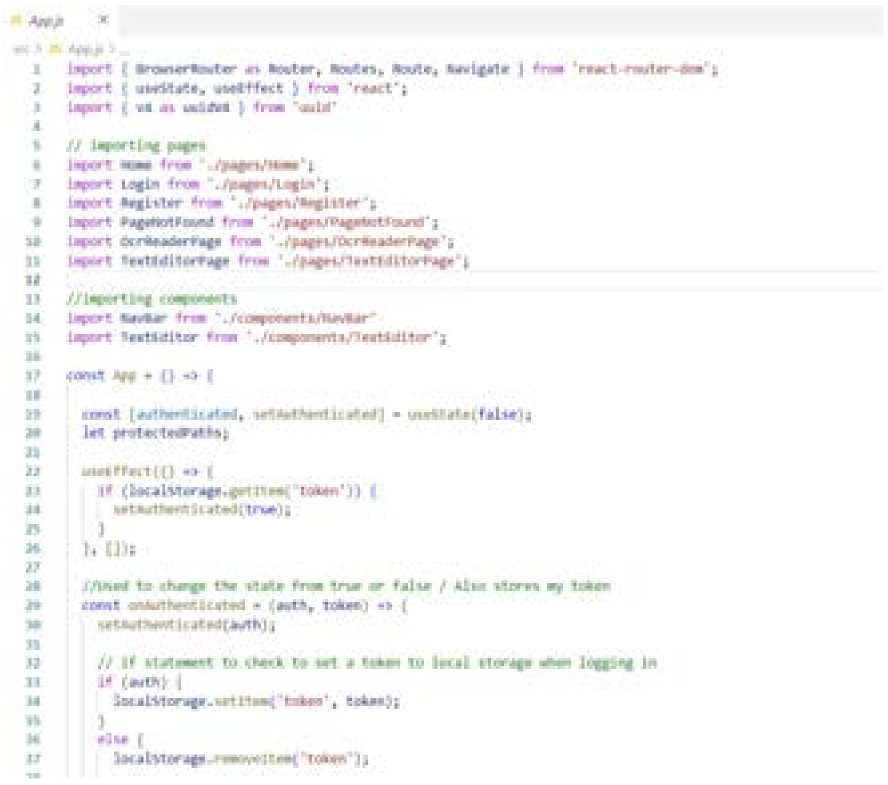
Figure 102 shows the structure of the combined project, including pages and components, showing all the prototypes combined.



```
import { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';

const Navbar = (props) => {

    const navigate = useNavigate;

    const logout = () => {
        props.onAuthenticated(false);
        navigate('/');
    };

    return (

        <div>

            <Link to='/'>
                Home
            </Link>
            |
            <Link to='/ocr'>
                OcrPrototype
            </Link>
            |
            <Link to='/register'>
                Register
            </Link>
            |
            <Link to='/login'>
                Login
            </Link>
            |
            <Link to='/select-document'>
                Select-document
            </Link>

            {(props.authenticated) ? (
                <button onClick={logout}>Logout</button>
```

Figure 103 Navbar Component

Figure 103 shows the navbar. The inclusion of a Navbar was also included in this phase, to allow users to navigate to the different pages that were created.

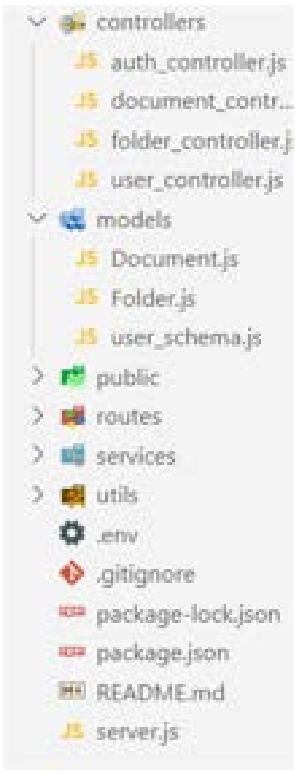


```
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { useState, useEffect } from 'react';
import { v4 as uuidv4 } from 'uuid'

// importing pages
import Home from './pages/Home';
import Login from './pages/Login';
import Register from './pages/Register';
import PageNotFound from './pages/PageNotFound';
import OcrReaderPage from './pages/OcrReaderPage';
import TextEditorPage from './pages/TextEditorPage';

//importing components
import NavBar from './components/NavBar'
import TextEditor from './components/TextEditor';

const App = () => {

  const [authenticated, setAuthenticated] = useState(false);
  let protectedPaths;

  useEffect(() => {
    if (localStorage.getItem('token')) {
      setAuthenticated(true);
    }
  }, []);

  //Used to change the state from true or false / Also stores my token
  const onAuthenticated = (auth, token) => {
    setAuthenticated(auth);

    // If statement to check to set a token to local storage when logging in
    if (auth) {
      localStorage.setItem('token', token);
    }
    else {
      localStorage.removeItem('token');
```

Figure 104 App.js file

Figure 104 shows the app.js file, which was also modified to include all the new pages for prototypes that were combined, allowing for the baseline of the application to be created.

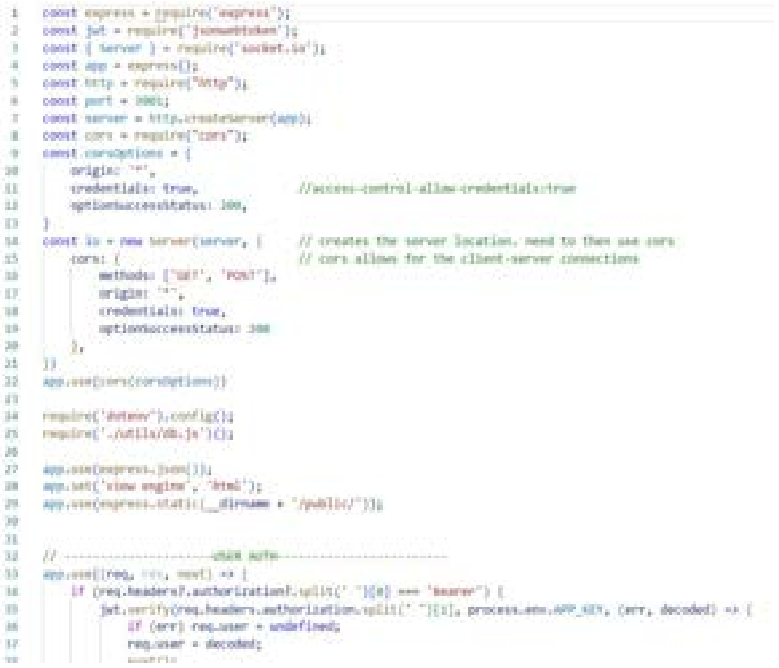
A similar procedure was also performed in the backend of the application, combining all of them into one.



```
controllers
    auth_controller.js
    document_contr...
    folder_controller.j
    user_controller.js
models
    Document.js
    Folder.js
    user_schema.js
public
routes
services
utils
.env
.gitignore
package-lock.json
package.json
README.md
server.js
```

Figure 105 Combined Backend

Figure 105 shows the controllers, schemas, and routes, creating one central backend for the application. A new MongoDB database was also created for the new data being made.



```
const express = require('express');
const jwt = require('jsonwebtoken');
const { Server } = require('socket.io');
const app = express();
const http = require("http");
const port = 3001;
const server = http.createServer(app);
const cors = require("cors");
const corsOptions = {
    origin: "*",
    credentials: true,            //access-control-allow-credentials:true
    optionSuccessStatus: 200,
}
const io = new Server(server, {      // creates the server location, need to then use cors
    cors: {                          // cors allows for the client-server connections
        methods: ['GET', 'POST'],
        origin: "*",
        credentials: true,
        optionSuccessStatus: 200
    },
})
app.use(cors(corsOptions))

require('dotenv').config();
require('./utils/db.js')();

app.use(express.json());
app.set('view engine', 'html');
app.use(express.static(__dirname + "/public/"));


// ----------------------check auth----------------------------
app.use((req, res, next) => {
    if (req.headers?.authorization?.split(' ')[0] === 'Bearer') {
        jwt.verify(req.headers.authorization.split(' ')[1], process.env.APP_KEY, (err, decoded) => {
            if (err) req.user = undefined;
            req.user = decoded;
```

Figure 106 Server.js

Figure 106 shows the server.js file. This file now contains all the routes that were required to combine the prototypes, although there were some slight modifications made to allow for the server.js file to work alongside the document's functionalities, which will be covered in item 4.

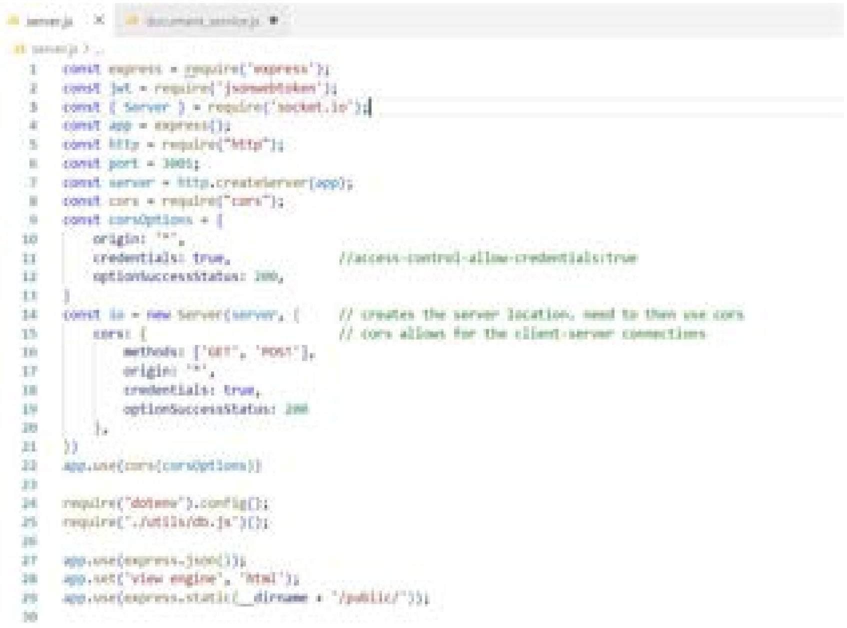
5.9.1.4 Item 4 – Fixing the problem with the document's backend.

The goal of this item was to fix a certain problem that was occurring with the backend for the documents file.

As socket.io was running its own server, the actual server for the backend needed to be restructured and altered to cater for this cause, as there were problems occurring when both servers were running.

Unfortunately, a screenshot of the error was not taken during this process, however, to allow this to work the socket.io server must be using the same server as express and use the functions the document to save.

Firstly, the socket.io server was first initialized to use the same server as express.

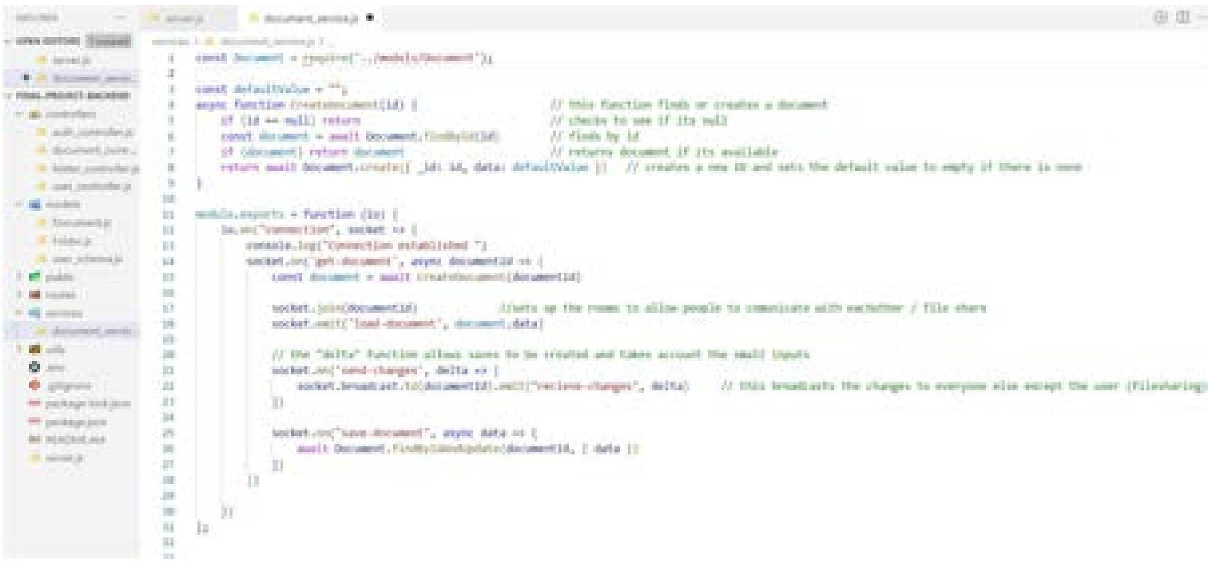


```
1   const express = require('express');
2   const jwt = require('jsonwebtoken');
3   const { Server } = require('socket.io');
4   const app = express();
5   const http = require("http");
6   const port = 3001;
7   const server = http.createServer(app);
8   const cors = require("cors");
9   const corsOptions = {
10      origin: "*",
11      credentials: true,            //access-control-allow-credentials:true
12      optionSuccessStatus: 200,
13  }
14  const io = new Server(server, {     // creates the server location, need to then use cors
15      cors: {                         // cors allows for the client-server connections
16          methods: ['GET', 'POST'],
17          origin: '*',
18          credentials: true,
19          optionSuccessStatus: 200
20      },
21  })
22  app.use(cors(corsOptions))
23
24  require("dotenv").config();
25  require("./utils/db.js")();
26
27  app.use(express.json());
28  app.set('view engine', 'html');
29  app.use(express.static(__dirname + '/public/'));
30
```

Figure 107 Initialization

Figure 107 shows the initialization of this, with the CORS option being set in the process. Line 14 – 21 then creates the server location.

The next step to solving this problem is to use the required functions to allow the document file to run. This is first done by separating these, and placing it into a services folder, which held the document_service.js file.

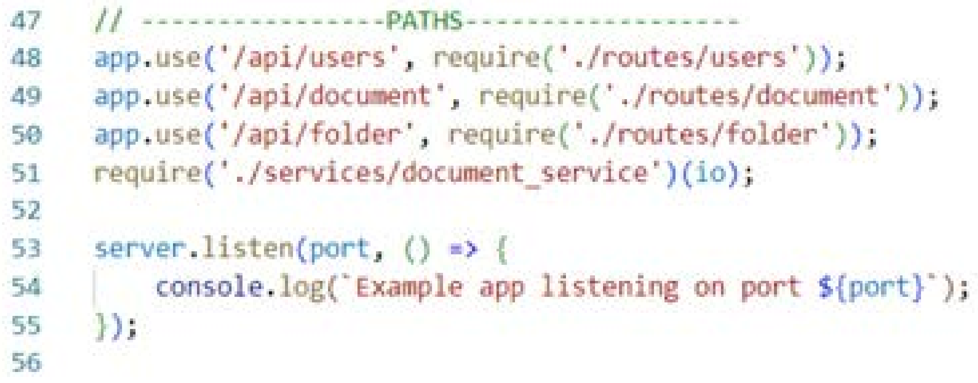


```
const Document = require("../models/Document");

const defaultValue = "";
async function createDocument(id) {                    // this function finds or creates a document
    if (id == null) return                             // checks to see if its null
    const document = await Document.findById(id)       // finds by id
    if (document) return document                      // returns document if its available
    return await Document.create({ _id: id, data: defaultValue })   // creates a new ID and sets the default value to empty if there is none
}

module.exports = function (io) {
    io.on("connection", socket => {
        console.log("Connection established ")
        socket.on("get-document", async documentId => {
            const document = await createDocument(documentId)

            socket.join(documentId)            //sets up the rooms to allow people to comunicate with eachother / file share
            socket.emit("load-document", document.data)

            // the "delta" function allows saves to be created and takes account the small inputs
            socket.on("send-changes", delta => {
                socket.broadcast.to(documentId).emit("recieve-changes", delta)     // this broadcasts the changes to everyone else except the user (Filesharing)
            })

            socket.on("save-document", async data => {
                await Document.findByIdAndUpdate(documentId, { data })
            })
        })

    })
};
```

Figure 108 Methods

This can be seen in figure 108, where the methods using socket.io are being used.

Finally, it is required to run these functions and get used in the server.js file.



```
47  // ----------------PATHS------------------
48  app.use('/api/users', require('./routes/users'));
49  app.use('/api/document', require('./routes/document'));
50  app.use('/api/folder', require('./routes/folder'));
51  require('./services/document_service')(io);
52
53  server.listen(port, () => {
54      console.log(`Example app listening on port ${port}`);
55  });
56
```

Figure 109 Paths

The figure above shows the document_service.js file being run, in line 51.

The result of this restructuring of the code allowed the document file to work again when it has been incorporated with the rest of the prototypes, allowing for the application to work.

5.9.1.5 Item 5 - Bug Fixes and problems

Aside from the major problem shown in the previous item, there was only one minor problem that occurred during this phase of implementation that was notable.

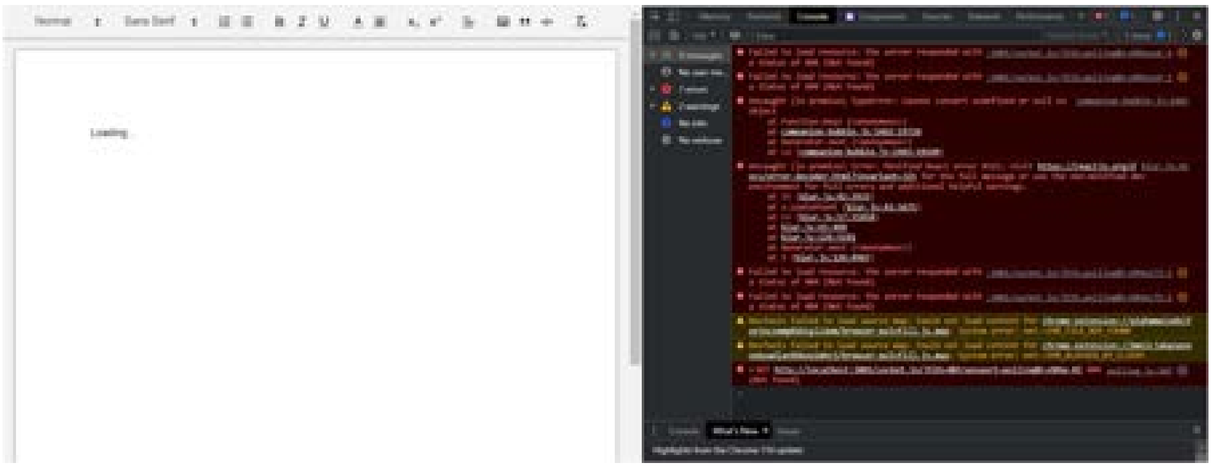
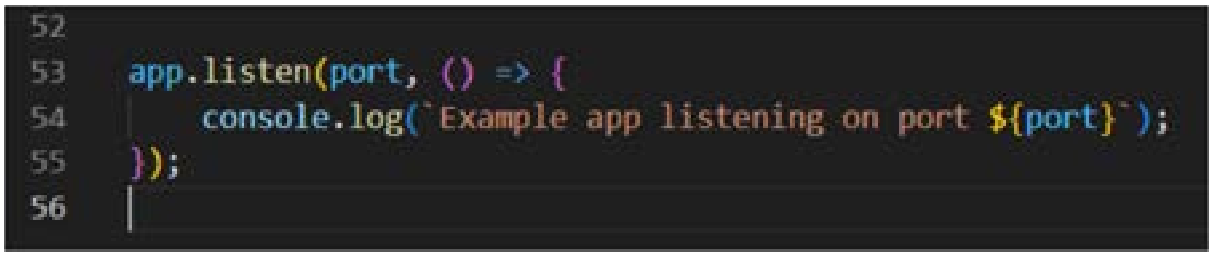


Figure 110 Problem 1

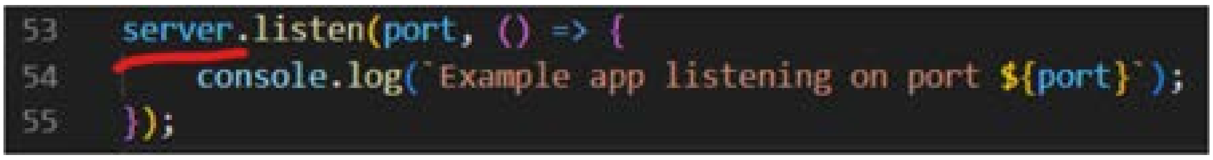
Figure 110 shows an infinite loading screen occurring, which didn't allow users to use the document editor, as it was disabled until the data was returned.



```
52
53    app.listen(port, () => {
54        console.log(`Example app listening on port ${port}`);
55    });
56
```

Figure 111 Result

Figure 111 shows the problem. The reason for this not working was because "app.listen" in line 53 was not using app anymore, and needed to be updated to "server.listen".



```
53    server.listen(port, () => {
54        console.log(`Example app listening on port ${port}`);
55    });
```

Figure 112 Result

Figure 112 shows the correct code, and the infinite loading for the documents page has been fixed.

5.10 Sprint 5 – Implementation 2 and Testing

5.10.1 Goal

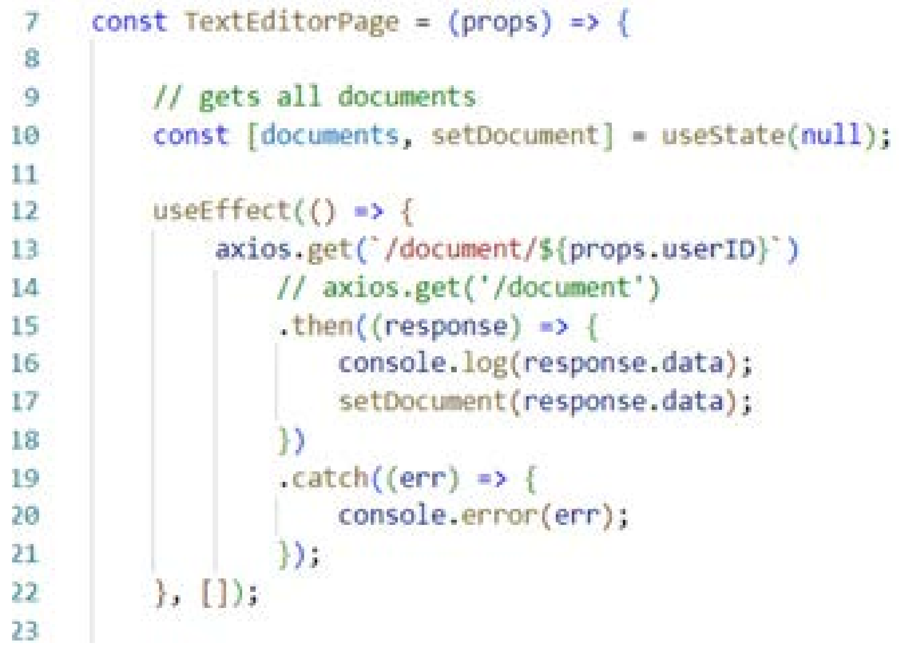
The goal of this sprint was to implement more features of the application, as well as perform minor testing to help eliminate errors that were occurring during the development of the application. The items for this sprint include:

- **Item 1:** Including CRUD in the Frontend for Document Editor.
- **Item 2:** Displaying the documents that is specific for each user.
- **Item 3:** Adding basic styling to project and pages.
- **Item 4:** Including an image upload feature.
- **Item 5:** Bug Fixes and problems

5.10.1.1 Item 1 – Including CRUD in the Frontend for Document Editor.

The goal of this sprint was to include the CRUD functionalities in the frontend of the documents. Currently a user is unable to create, read, update, or delete a document unless it is viewing a single page with the id already available or using the backend to perform these methods.

To start, TextEditorPage.js was created to display all documents. This was done with the use of axios, retrieving all the documents from the database. This can be seen in figure 113.

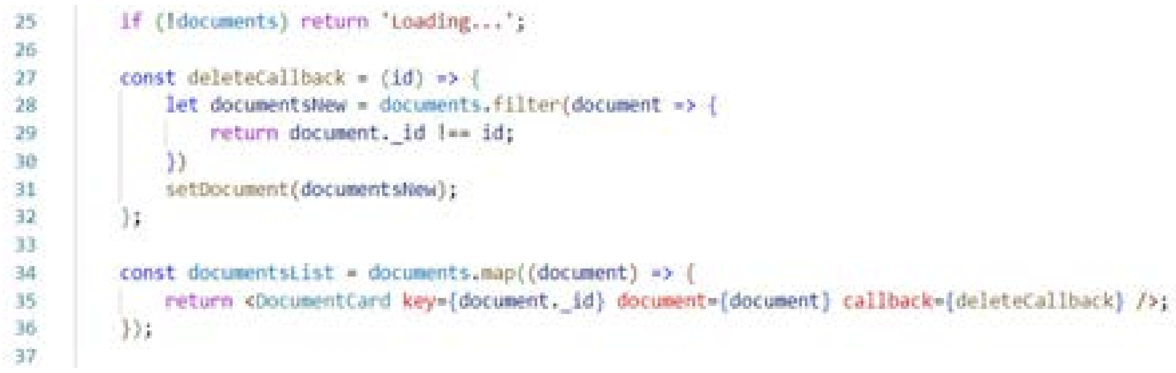


```
const TextEditorPage = (props) => {

    // gets all documents
    const [documents, setDocument] = useState(null);

    useEffect(() => {
        axios.get(`/document/${props.userID}`)
            // axios.get('/document')
            .then((response) => {
                console.log(response.data);
                setDocument(response.data);
            })
            .catch((err) => {
                console.error(err);
            });
    }, []);

```

Figure 113 Data Removal

Figure 114 shows the data being retrieved.

The data is then placed into a document card, which will receive the document id as a key.



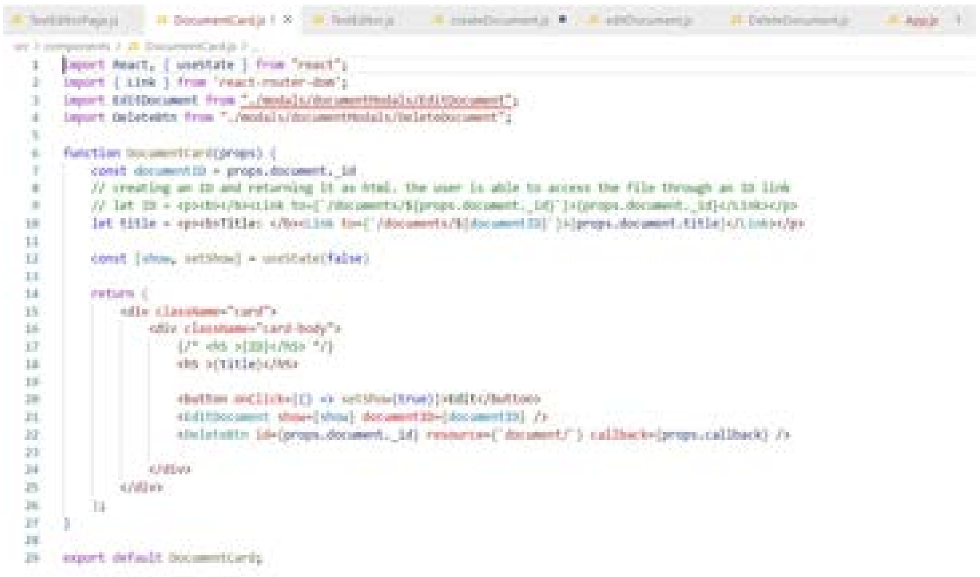
```
25      if (!documents) return 'Loading...';
26
27      const deleteCallback = (id) => {
28          let documentsNew = documents.filter(document => {
29              return document._id !== id;
30          })
31          setDocument(documentsNew);
32      };
33
34      const documentsList = documents.map((document) => {
35          return <DocumentCard key={document._id} document={document} callback={deleteCallback} />;
36      });
37
```

Figure 114 Error handling

Figure 114 shows the that if there is no documents, it returns as "loading".

Line 34 – 37 shows the documents list returning a card for each document that is within the database.

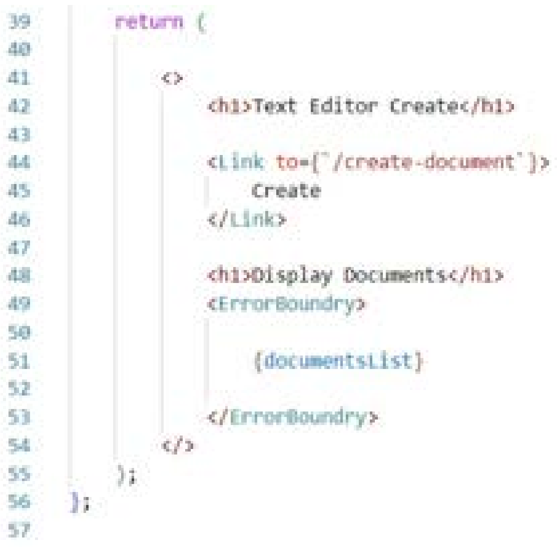
The document card is a separate file, which contains its own properties, that can link users to that specific document.



```
1   import React, { useState } from "react";
2   import { Link } from 'react-router-dom';
3   import EditDocument from "./modals/documentModals/EditDocument";
4   import DeleteBtn from "./modals/documentModals/Deletedocument";
5
6   function DocumentCard(props) {
7       const documentID = props.document._id
8       // creating an ID and returning it as html. the user is able to access the file through an ID link
9       // let ID = <p><b></b><Link to={`/documents/${props.document._id}`}>{props.document._id}</Link></p>
10      let title = <p><b>Title: </b><Link to={`/documents/${documentID}`}>{props.document.title}</Link></p>
11
12      const [show, setShow] = useState(false)
13
14      return (
15          <div className="card">
16              <div className="card-body">
17                  {/* <h5 >{ID}</h5> */}
18                  <h5 >{title}</h5>
19
20                  <button onClick={() => setShow(true)}>Edit</button>
21                  <EditDocument show={show} documentID={documentID} />
22                  <DeleteBtn id={props.document._id} resource={`document/`} callback={props.callback} />
23
24              </div>
25          </div>
26      );
27  }
28
29  export default DocumentCard;
```

Figure 115 Document Card.js

Figure 115 shows DocumentCard.js. It uses the title as a link to the specific document, line 10 shows the route, using the document id to redirect the user to that specific document. It also contains the edit and delete document buttons.

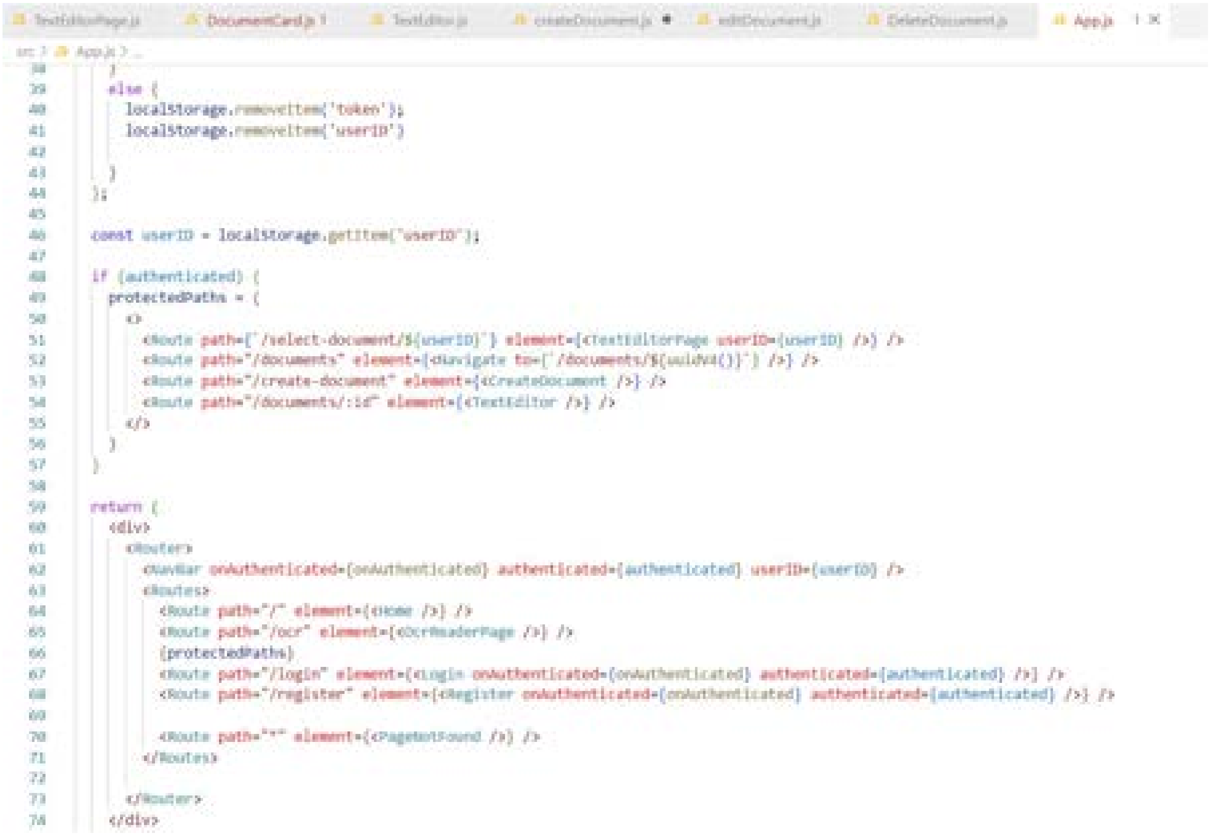


```
    return (

        <>
            <h1>Text Editor Create</h1>

            <Link to={`/create-document`}>
                Create
            </Link>

            <h1>Display Documents</h1>
            <ErrorBoundry>

                {documentsList}

            </ErrorBoundry>
        </>
    );
};
```

Figure 116 Document List

The document list is then placed on the page to display all the documents. This can be seen in the figure above.

If the user clicks on one of the links, it brings the user to the single document page.



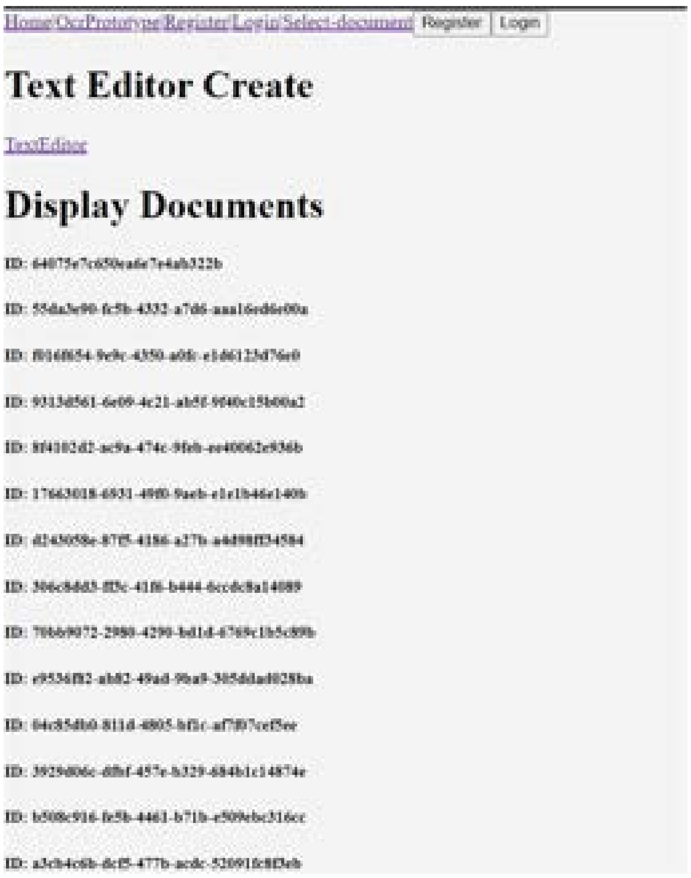
TextEditorPage.js DocumentCard.js 1 TextEditor.js createDocument.js editDocument.js DeleteDocument.js App.js 1

src > App.js > ...

```
38      }
39      else {
40        localStorage.removeItem('token');
41        localStorage.removeItem('userID')
42
43      }
44    };
45
46    const userID = localStorage.getItem('userID');
47
48    if (authenticated) {
49      protectedPaths = (
50        <>
51          <Route path={`/select-document/${userID}`} element={<TextEditorPage userID={userID} />} />
52          <Route path="/documents" element={<Navigate to={`/documents/${uuidV4()}`} />} />
53          <Route path="/create-document" element={<CreateDocument />} />
54          <Route path="/documents/:id" element={<TextEditor />} />
55        </>
56      )
57    }
58
59    return (
60      <div>
61        <Router>
62          <NavBar onAuthenticated={onAuthenticated} authenticated={authenticated} userID={userID} />
63          <Routes>
64            <Route path="/" element={<Home />} />
65            <Route path="/ocr" element={<OcrReaderPage />} />
66            {protectedPaths}
67            <Route path="/login" element={<Login onAuthenticated={onAuthenticated} authenticated={authenticated} />} />
68            <Route path="/register" element={<Register onAuthenticated={onAuthenticated} authenticated={authenticated} />} />
69
70            <Route path="*" element={<PageNotFound />} />
71          </Routes>
72
73        </Router>
74      </div>
```

Figure 117 Protected Paths

Figure 117 shows the protected paths. On line 54, the route will take a document ID and redirect it to that text editor page with that same id.


Home|OcrPrototype|Register|Login|Select-document Register Login

Text Editor Create

TextEditor

Display Documents

ID: 64075e7c650eade7e4ab322b

ID: 55da3e90-fc5b-4332-a7d6-aaa16ed6e00a

ID: f016f654-9e9c-4350-a0fc-e1d6123d76e0

ID: 9313d561-6e09-4c21-ab5f-9f40c15b00a2

ID: 8f4102d2-ac9a-474c-9feb-ee40062e936b

ID: 17663018-6931-49f0-9aeb-e1e1b46e140b

ID: d245058e-87f5-4186-a27b-a4d98ff34584

ID: 306c8dd3-ff3c-41f6-b444-6ccdc8a14089

ID: 70bb9072-2980-4290-bd1d-6769c1b5c89b

ID: e9536f82-ab82-49ad-9ba9-305ddad028ba

ID: 04c85db0-811d-4805-bf1c-af7f07cef5ee

ID: 3929d06c-dfbf-457e-b329-684b1c14874e

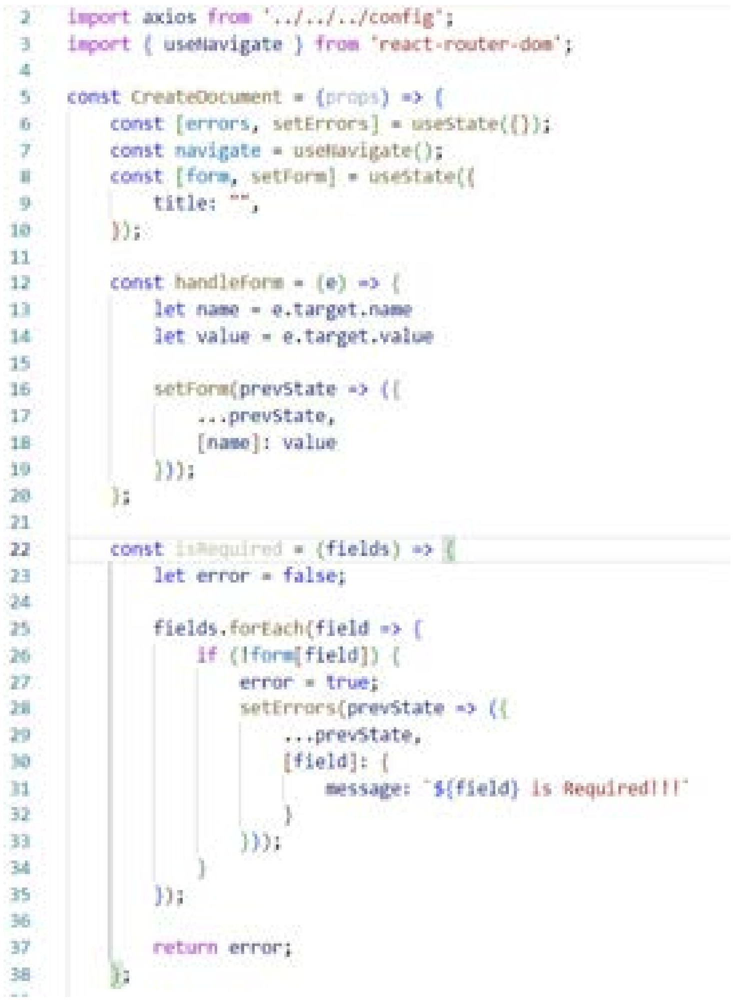
ID: b508c916-fe5b-4461-b71b-e509ebc316cc

ID: a3cb4c6b-dcf5-477b-acdc-52091fc8f3eb

Figure 118 Displaying All Documents

Figure 118 shows the documents all being displayed, simply showing the ID of each document.

Next, a create method was built in the frontend. A page was first created for the create documents page, this contained a form for the title of the document, and later will add an image.

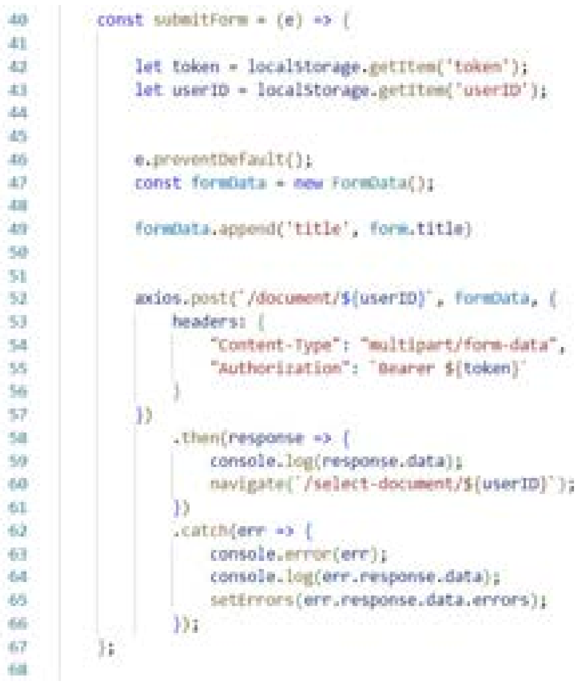


```
import axios from '../../../config';
import { useNavigate } from 'react-router-dom';

const CreateDocument = (props) => {
    const [errors, setErrors] = useState({});
    const navigate = useNavigate();
    const [form, setForm] = useState({
        title: "",
    });

    const handleForm = (e) => {
        let name = e.target.name
        let value = e.target.value

        setForm(prevState => ({
            ...prevState,
            [name]: value
        }));
    };

    const isRequired = (fields) => {
        let error = false;

        fields.forEach(field => {
            if (!form[field]) {
                error = true;
                setErrors(prevState => ({
                    ...prevState,
                    [field]: {
                        message: `${field} is Required!!!`
                    }
                }));
            }
        });

        return error;
    };
```

Figure 119 Form

Figure 119 shows the setup for the form of the application. The “handleForm” method is meant to handle the different input fields in the form and is useful if there were more than one input field.

The “isRequired” function is to state whether that specific field is required when creating a new document.



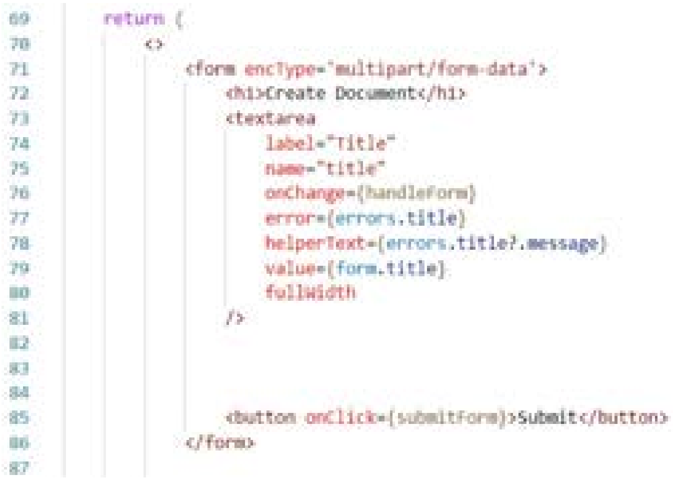
```
const submitForm = (e) => {

    let token = localStorage.getItem('token');
    let userID = localStorage.getItem('userID');


    e.preventDefault();
    const formData = new FormData();

    formData.append('title', form.title)


    axios.post(`/document/${userID}`, formData, {
        headers: {
            "Content-Type": "multipart/form-data",
            "Authorization": `Bearer ${token}`
        }
    })
        .then(response => {
            console.log(response.data);
            navigate(`/select-document/${userID}`);
        })
        .catch(err => {
            console.error(err);
            console.log(err.response.data);
            setErrors(err.response.data.errors);
        });
};
```

Figure 120 Handler

The submit form is then used to handle when a user is submitting the form. It first takes the token and user id from the local storage of the app. It then creates a form data and appends the title from the form, meaning the title is now a part of the formData variable. Once this is done an axios.post request is done, uploading the formData to the userId.

The content is specified to be a multipart/form-data as an image upload will be implemented in a later sprint.

The data is then uploaded, and the user is navigated back to the select-document page.

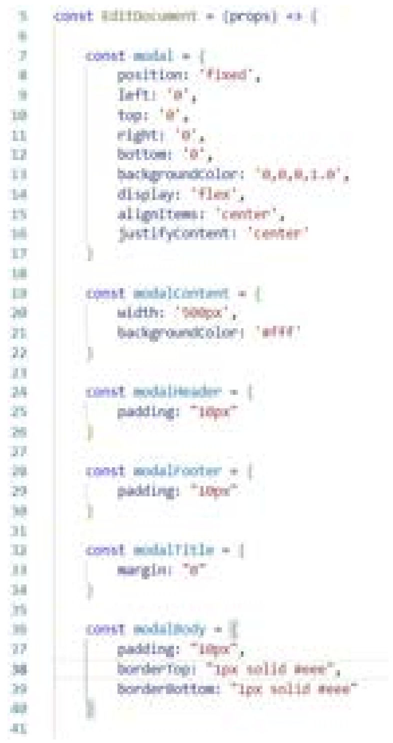


```
return (
    <>
        <form encType='multipart/form-data'>
            <h1>Create Document</h1>
            <textarea
                label="Title"
                name="title"
                onChange={handleForm}
                error={errors.title}
                helperText={errors.title?.message}
                value={form.title}
                fullWidth
            />



            <button onClick={submitForm}>Submit</button>
        </form>
```

Figure 121 Submission

Figure 121 shows the form and the submit button using the submitForm function.

The edit documents modal is like the create document function. The code will first contain the modal styling, seen in figure 122.

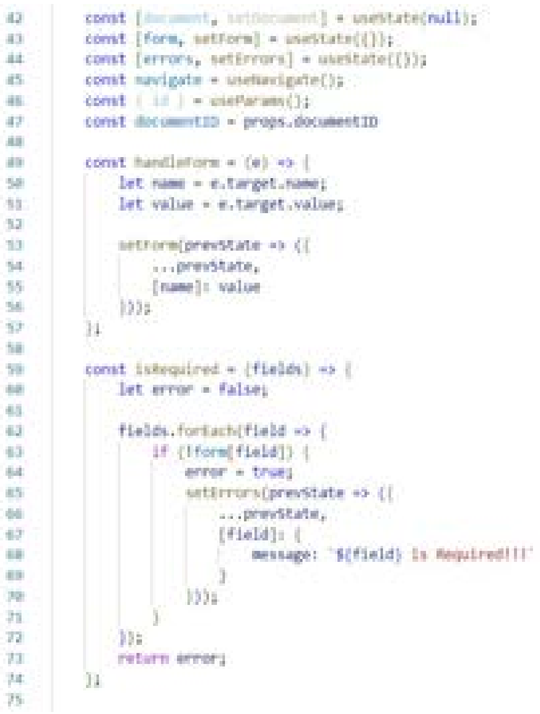


```
const EditDocument = (props) => {

    const modal = {
        position: 'fixed',
        left: '0',
        top: '0',
        right: '0',
        bottom: '0',
        backgroundColor: '0,0,0,1.0',
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center'
    }

    const modalContent = {
        width: '500px',
        backgroundColor: '#fff'
    }

    const modalHeader = {
        padding: "10px"
    }

    const modalFooter = {
        padding: "10px"
    }

    const modalTitle = {
        margin: "0"
    }

    const modalBody = {
        padding: "10px",
        borderTop: "1px solid #eee",
        borderBottom: "1px solid #eee"
    }

```

Figure 122 Edit Document Modal

Figure 122 is a simple modal that will only be the styling for it temporarily.

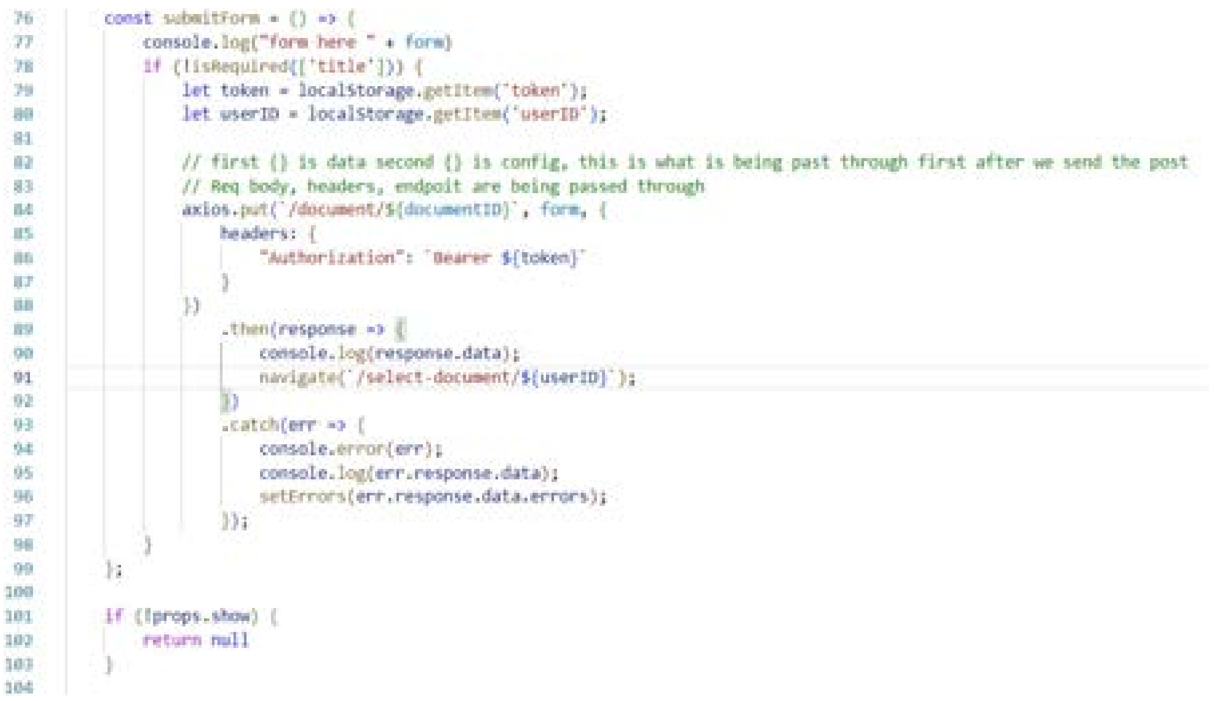
The code then sets up the variables that will be used, and the isRequired and handleForm functions, which can be seen in figure 123.



```
    const [document, setDocument] = useState(null);
    const [form, setForm] = useState({});
    const [errors, setErrors] = useState({});
    const navigate = useNavigate();
    const { id } = useParams();
    const documentID = props.documentID

    const handleForm = (e) => {
        let name = e.target.name;
        let value = e.target.value;

        setForm(prevState => ({
            ...prevState,
            [name]: value
        }));
    };

    const isRequired = (fields) => {
        let error = false;

        fields.forEach(field => {
            if (!form[field]) {
                error = true;
                setErrors(prevState => ({
                    ...prevState,
                    [field]: {
                        message: `${field} Is Required!!!`
                    }
                }));
            }
        });
        return error;
    };

```

Figure 123 Is Required and Handle Form Functions

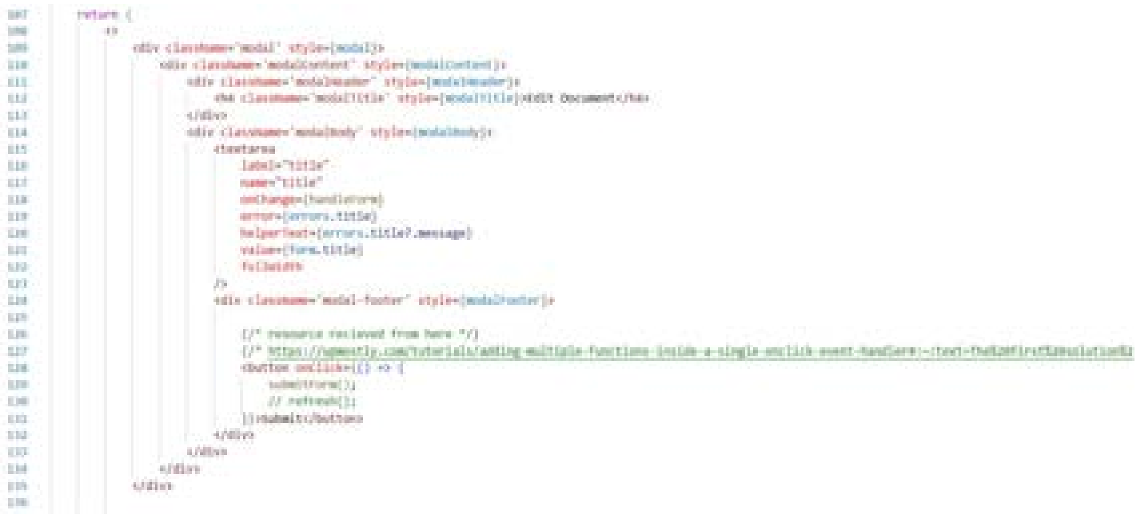
The axios.put method is then used in the submit form. It uses the documentID to update it. This can be seen in figure 124.



```
const submitForm = () => {
    console.log("form here " + form)
    if (!isRequired(['title'])) {
        let token = localStorage.getItem('token');
        let userID = localStorage.getItem('userID');

        // first {} is data second {} is config, this is what is being past through first after we send the post
        // Req body, headers, endpoit are being passed through
        axios.put(`/document/${documentID}`, form, {
            headers: {
                "Authorization": `Bearer ${token}`
            }
        })
            .then(response => {
                console.log(response.data);
                navigate(`/select-document/${userID}`);
            })
            .catch(err => {
                console.error(err);
                console.log(err.response.data);
                setErrors(err.response.data.errors);
            });
    }
};

if (!props.show) {
    return null
}
```

Figure 124 Axios Call

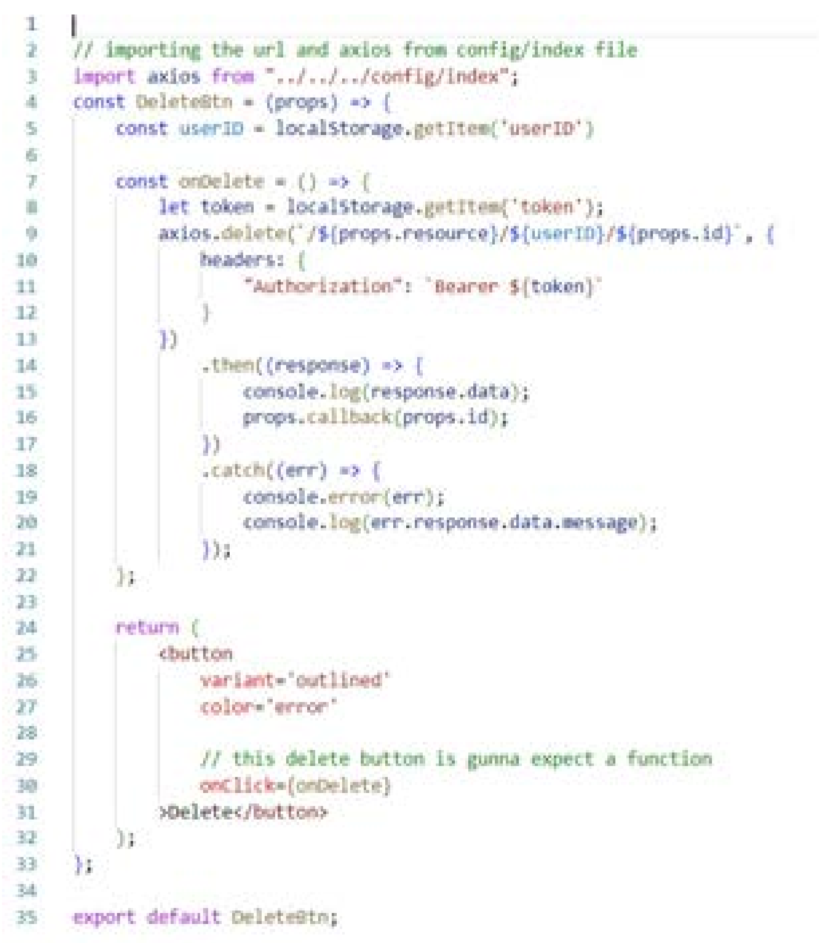
The form itself is seen in figure 125.



```
return (
    <>
        <div className='modal' style={modal}>
            <div className='modalContent' style={modalContent}>
                <div className='modalHeader' style={modalHeader}>
                    <h4 className='modalTitle' style={modalTitle}>Edit Document</h4>
                </div>
                <div className='modalBody' style={modalBody}>
                    <textarea
                        label="title"
                        name="title"
                        onChange={handleForm}
                        error={errors.title}
                        helperText={errors.title?.message}
                        value={form.title}
                        fullWidth
                    />
                    <div className='modal-footer' style={modalFooter}>

                        {/* resource recieved from here */}
                        {/* https://upmostly.com/tutorials/adding-multiple-functions-inside-a-single-onclick-event-handler#:~:text=The%20first%20solution%2
                        <button onClick={() => {
                            submitForm();
                            // refresh();
                        }}>Submit</button>
                    </div>
                </div>
            </div>
        </div>
```

Figure 125 Form

Finally, the implementation of the delete document will be done. Figure 126 shows the code for the delete document function:



```
// importing the url and axios from config/index file
import axios from "../../../config/index";
const DeleteBtn = (props) => {
    const userID = localStorage.getItem('userID')

    const onDelete = () => {
        let token = localStorage.getItem('token');
        axios.delete(`/${props.resource}/${userID}/${props.id}`, {
            headers: {
                "Authorization": `Bearer ${token}`
            }
        })
            .then((response) => {
                console.log(response.data);
                props.callback(props.id);
            })
            .catch((err) => {
                console.error(err);
                console.log(err.response.data.message);
            });
    };

    return (
        <button
            variant='outlined'
            color='error'

            // this delete button is gunna expect a function
            onClick={onDelete}
        >Delete</button>
    );
};

export default DeleteBtn;
```

Figure 126 Delete Document Function

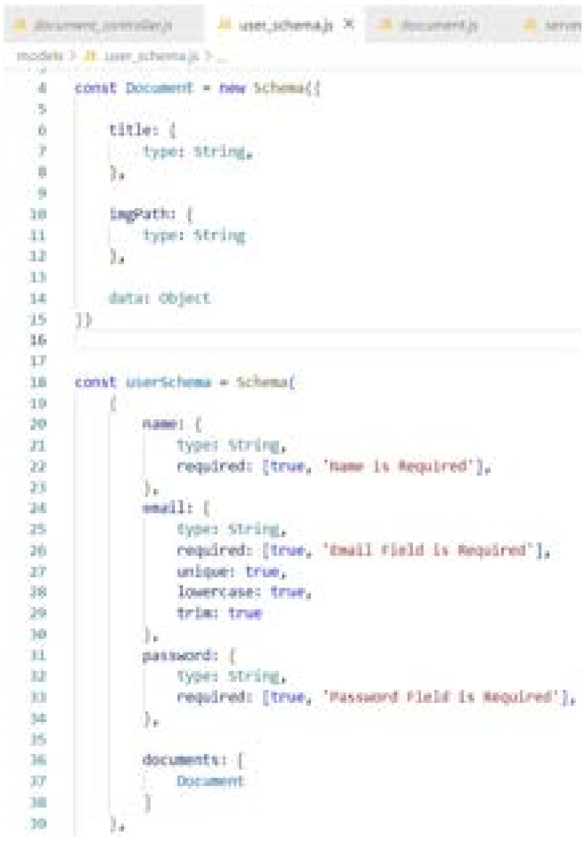
It first accesses the userId and the token and deletes the resource that is passed through. In this case resource is "document".

It is then returned as a button that can is used in the DocumentCard mentioned when discussing the DocumentCard.

5.10.1.2 Item 2 – Displaying the documents that is specific for each user.

The goal of this item was to alter the code to allow users to see their own specific documents. As of now, when a user logs in, they can see the documents for all users, and the documents are not specific to their own.

The code was updated to cater for this problem and begins with updating the document schema to be included with the user schema.

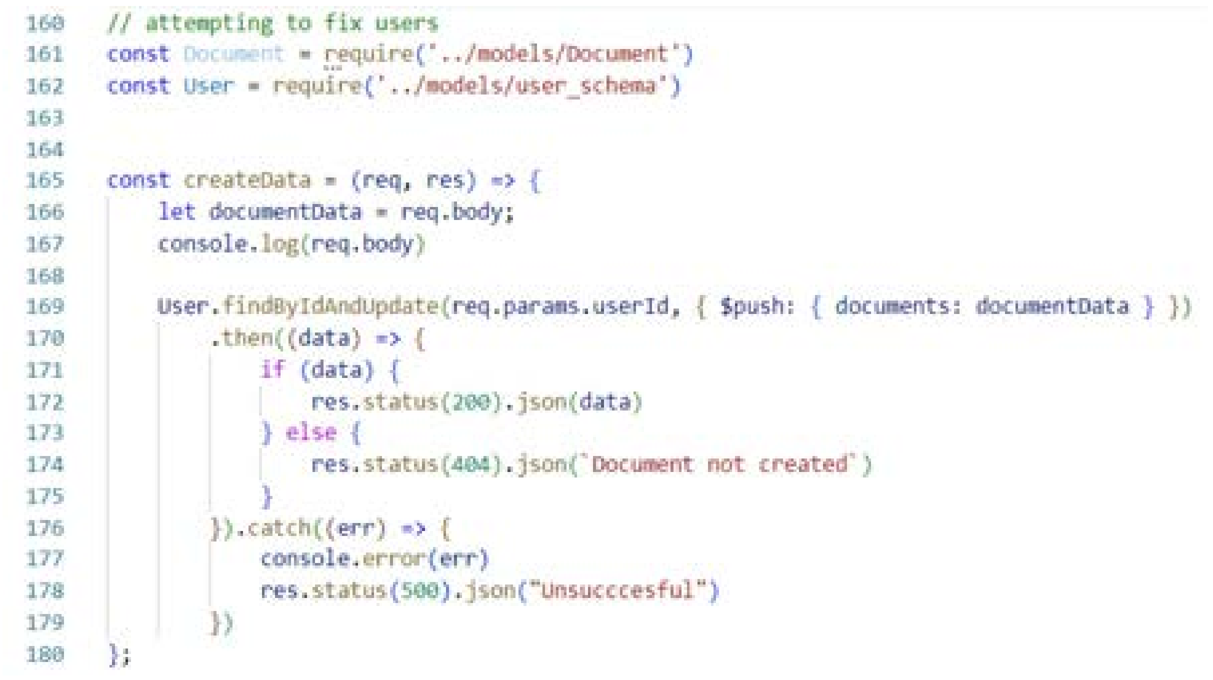


```
const Document = new Schema({

    title: {
        type: String,
    },

    imgPath: {
        type: String
    },

    data: Object
})


const userSchema = Schema(
    {
        name: {
            type: String,
            required: [true, 'Name is Required'],
        },
        email: {
            type: String,
            required: [true, 'Email Field is Required'],
            unique: true,
            lowercase: true,
            trim: true
        },
        password: {
            type: String,
            required: [true, 'Password Field is Required'],
        },

        documents: [
            Document
        ]
    },
```

Figure 127 Document Schema in User Schema

Figure 127 shows the document schema being included in the user schema. This change was performed to documents to be included in a user's profile, rather than being a separate entity.

Next, changes were made to the document_controller.js file. The CRUD functionalities were updated in the controller. It no longer uses the previous version, and now uses slightly altered functions to cater for the changes made when finding a document in a user's profile.



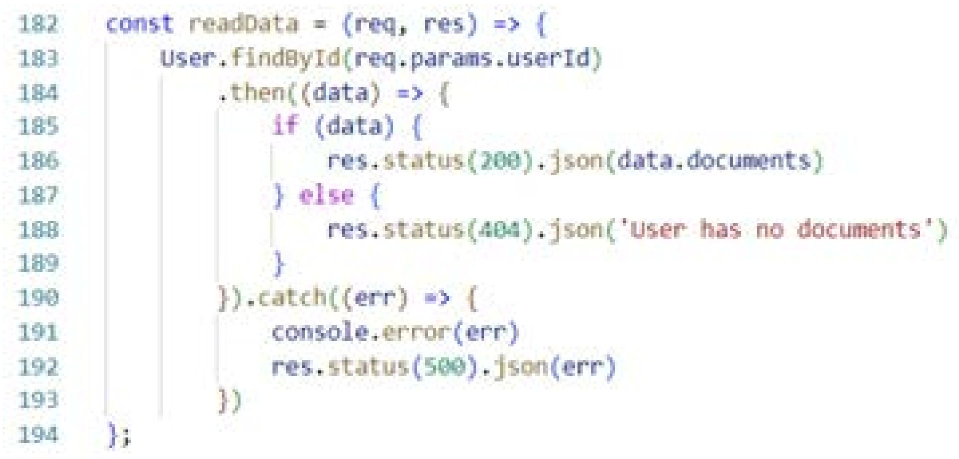
```
// attempting to fix users
const Document = require('../models/Document')
const User = require('../models/user_schema')


const createData = (req, res) => {
    let documentData = req.body;
    console.log(req.body)

    User.findByIdAndUpdate(req.params.userId, { $push: { documents: documentData } })
        .then((data) => {
            if (data) {
                res.status(200).json(data)
            } else {
                res.status(404).json(`Document not created`)
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json("Unsucccesful")
        })
};
```

Figure 128 Using User Schema

In figure 128, it is shown that instead of using the “document” schema, it is instead using the “user” schema to find the documents that are stored within the users.

In the “createData” function, it was altered to use the user instead of the document, alongside the “findByIdAndUpdate” function found within the express.js documentation. The code will first request the userId and push the data to the documents inside of the user. It will return with a status of 200 when successful, and a 404 or status 500 when it is unsuccessful.

The “readData” function was also updated.

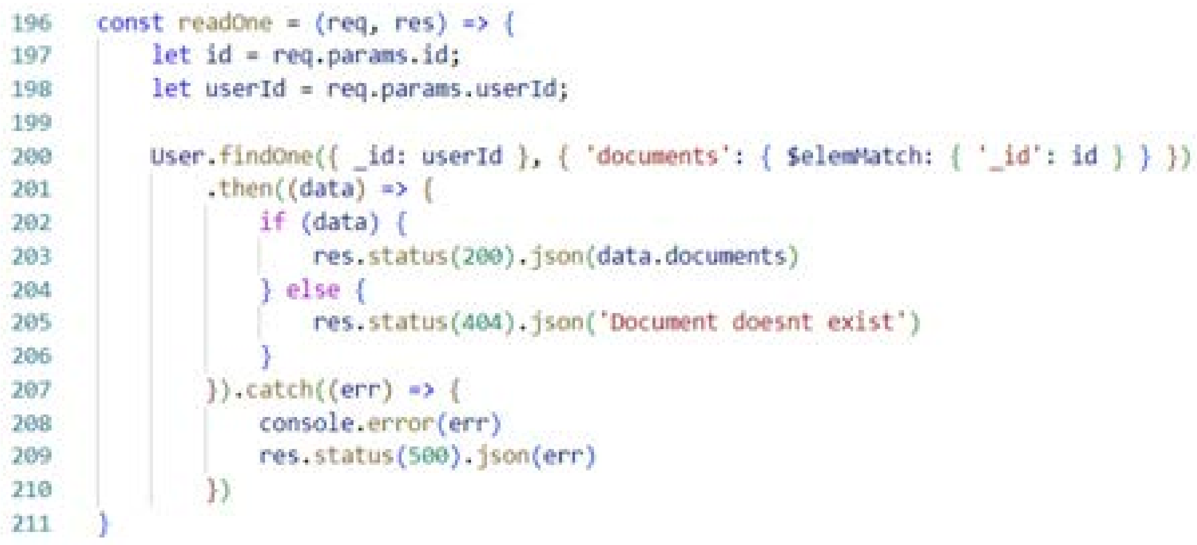


```
const readData = (req, res) => {
    User.findById(req.params.userId)
        .then((data) => {
            if (data) {
                res.status(200).json(data.documents)
            } else {
                res.status(404).json('User has no documents')
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json(err)
        })
};
```

Figure 129 Read Data Updated

Figure 129 shows the function requesting for the user first, then returning the documents. This means that when a user is found, it will return the documents of that user with a status 200 when successful, and status 404 or status 500 when unsuccessful.

The “readOne” function is like the above “readData” function.

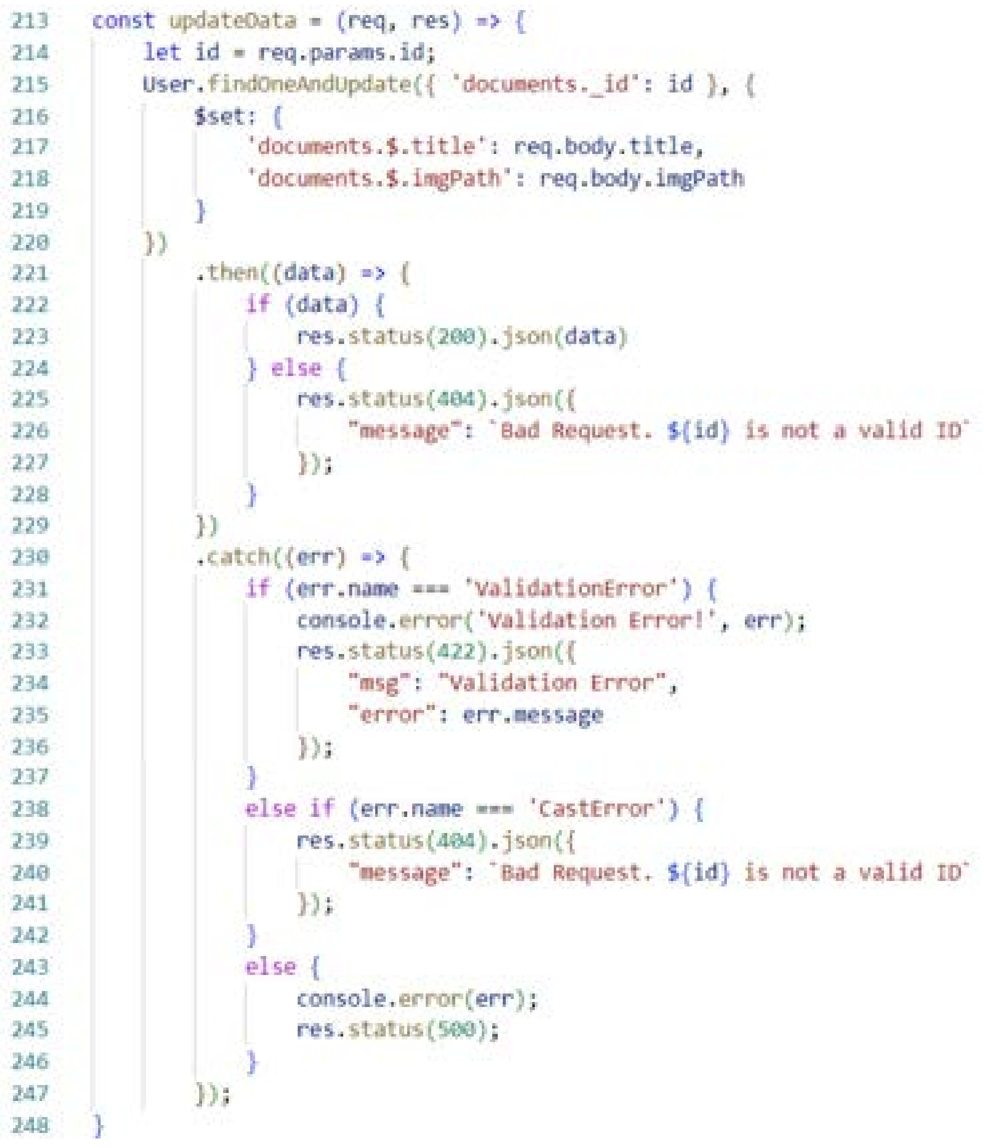


```
const readOne = (req, res) => {
    let id = req.params.id;
    let userId = req.params.userId;

    User.findOne({ _id: userId }, { 'documents': { $elemMatch: { '_id': id } } })
        .then((data) => {
            if (data) {
                res.status(200).json(data.documents)
            } else {
                res.status(404).json('Document doesnt exist')
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json(err)
        })
}
```

Figure 130 User Id

Figure 130 shows that it will first find the userId, then search through the documents and find the document that matches the id that was requested. It returns if successful, and status 404 / 500 when it is unsuccessful.

The “updateData” function contains the most significant changes made.

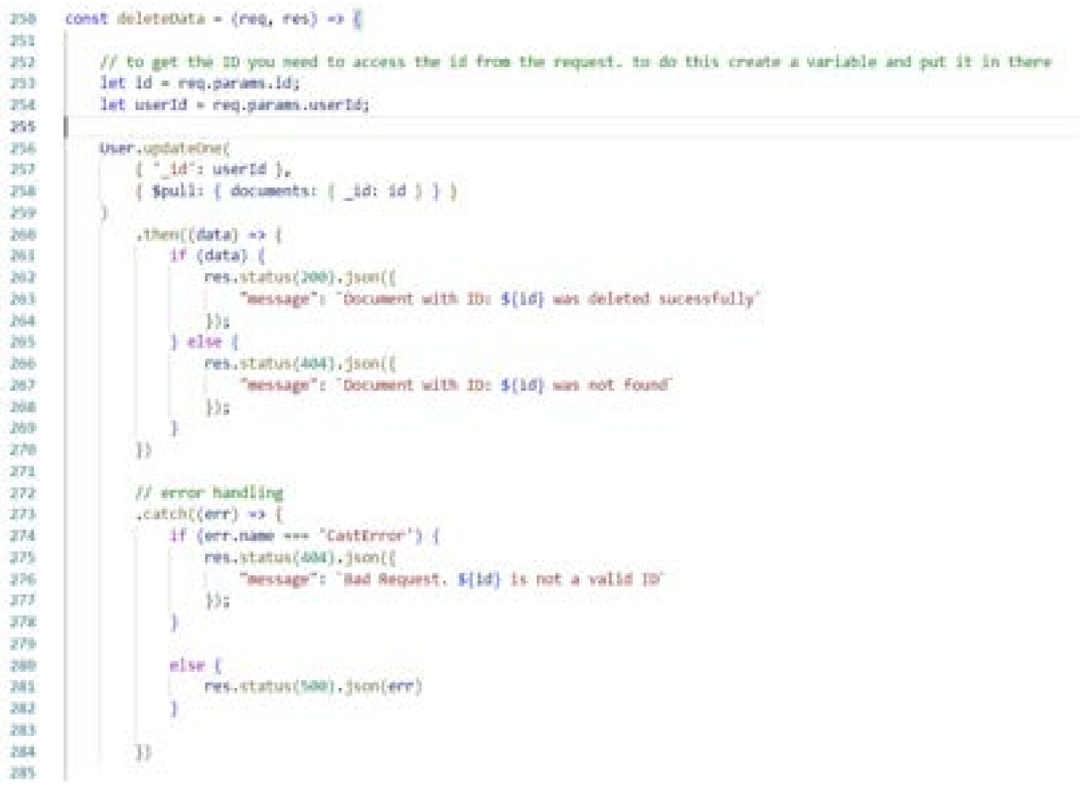


```
const updateData = (req, res) => {
    let id = req.params.id;
    User.findOneAndUpdate({ 'documents._id': id }, {
        $set: {
            'documents.$.title': req.body.title,
            'documents.$.imgPath': req.body.imgPath
        }
    })
        .then((data) => {
            if (data) {
                res.status(200).json(data)
            } else {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
        })
        .catch((err) => {
            if (err.name === 'ValidationError') {
                console.error('Validation Error!', err);
                res.status(422).json({
                    "msg": "Validation Error",
                    "error": err.message
                });
            }
            else if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
            else {
                console.error(err);
                res.status(500);
            }
        });
}
```

Figure 131 Update Data

Figure 131 shows the update data function. To start, the code will first receive a req and res as parameters. The id is then retrieved from the request parameters and set to the variable “id”. It will then use the “findOneAndUpdate” method to find the specific document, and update the corresponding fields to the req.body that the user choses to input. It will then check to see if it is successful, with the use of error handling, and respond with a status 200 if it is successful.

Finally, the delete data function was also updated.



```
const deleteData = (req, res) => {

    // to get the ID you need to access the id from the request. to do this create a variable and put it in there
    let id = req.params.id;
    let userId = req.params.userId;

    User.updateOne(
        { "_id": userId },
        { $pull: { documents: { _id: id } } }
    )
        .then((data) => {
            if (data) {
                res.status(200).json({
                    "message": `Document with ID: ${id} was deleted sucessfully`
                });
            } else {
                res.status(404).json({
                    "message": `Document with ID: ${id} was not found`
                });
            }
        })

        // error handling
        .catch((err) => {
            if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }

            else {
                res.status(500).json(err)
            }

        })
```

Figure 132 Delete Data

Figure 132 shows the delete data. It first takes the user and document Id, then removes it by pulling that document from the user. If it is successfully deleted, it returns with a success status of 200.

With these modifications made, the users can now perform CRUD functionalities on their own specific set of documents.

It is important to note that a similar procedure will occur when developing the folder structure of the application and will not be discussed in much detail in that phase of implementation, except noting the changes.

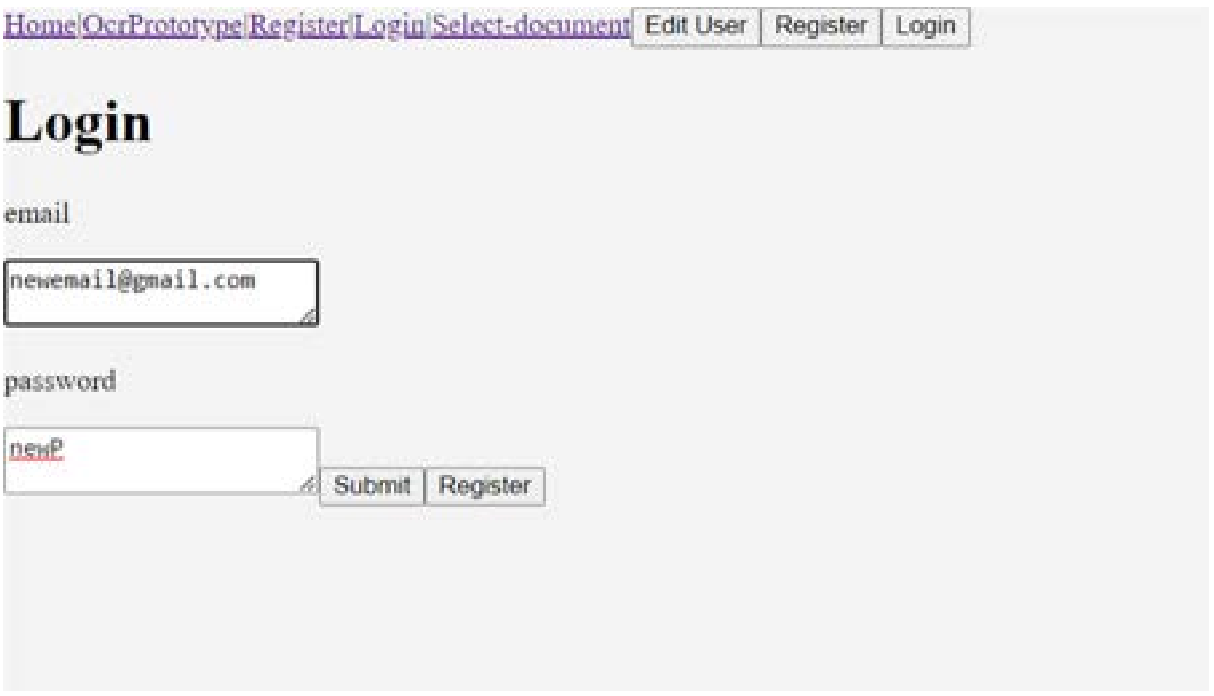
5.10.1.3 Item 3 – Adding basic styling to pages.

The goal of this sprint was to add very basic navigation and UI elements to the project. This was implemented very quickly to allow the developer to navigate through the pages and display the required information when developing. Further styling of the application will be completed in a separate sprint.

The figures below will showcase the different pages that were built.

Figure 133 OCR Reader V1

Figure 133 shows the first iteration of the OCR reader.



Login

email

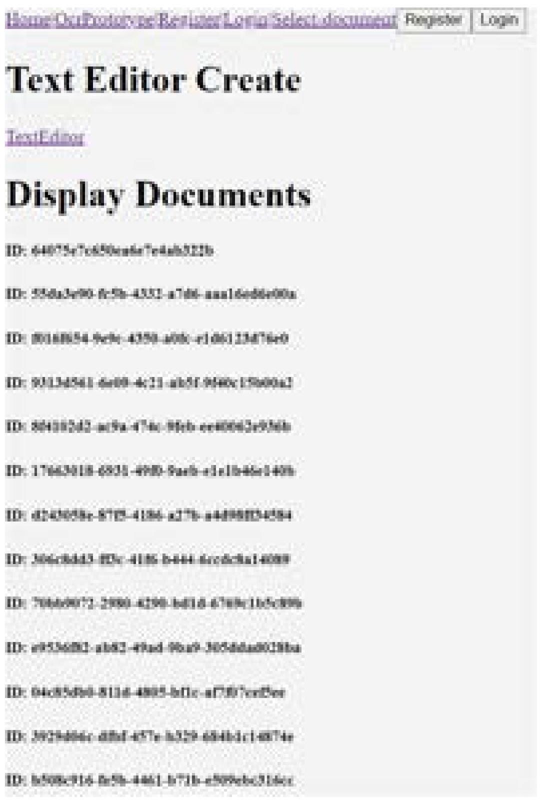
newemail@gmail.com

password

newP

Figure 134 Login V1

Figure 134 shows the first iteration of the login page.



Text Editor Create

TextEditor

Display Documents

ID: 64075e7c650ea6e7e4ab322b

ID: 55da3e90-fc5b-4332-a7d6-aaa16ed6e00a

ID: f016f654-9e9c-4350-a0fc-e1d6123d76e0

ID: 9313d561-6e09-4c21-ab5f-9f40c15b00a2

ID: 804102d2-ac9a-474c-9feb-ee40062e936b

ID: 17663018-6931-49f0-9aeb-e1e1b46e140b

ID: d243058e-87f5-4186-a27b-a4d98ff34584

ID: 306c8dd3-ff3c-41f6-b444-6ccdc8a14089

ID: 70bb9072-2980-4290-bd1d-6769c1b5c89b

ID: e9536f82-ab82-49ad-9ba9-305dda0028ba

ID: 04c85db0-811d-4805-bf1c-af7f07cef5ee

ID: 3929d06c-dfbf-457e-b329-6b4b1c14874e

ID: b508c916-fe5b-4461-b71b-e509ebc316cc

Figure 135 Displaying All Documents

Figure 135 shows the documents all being displayed.

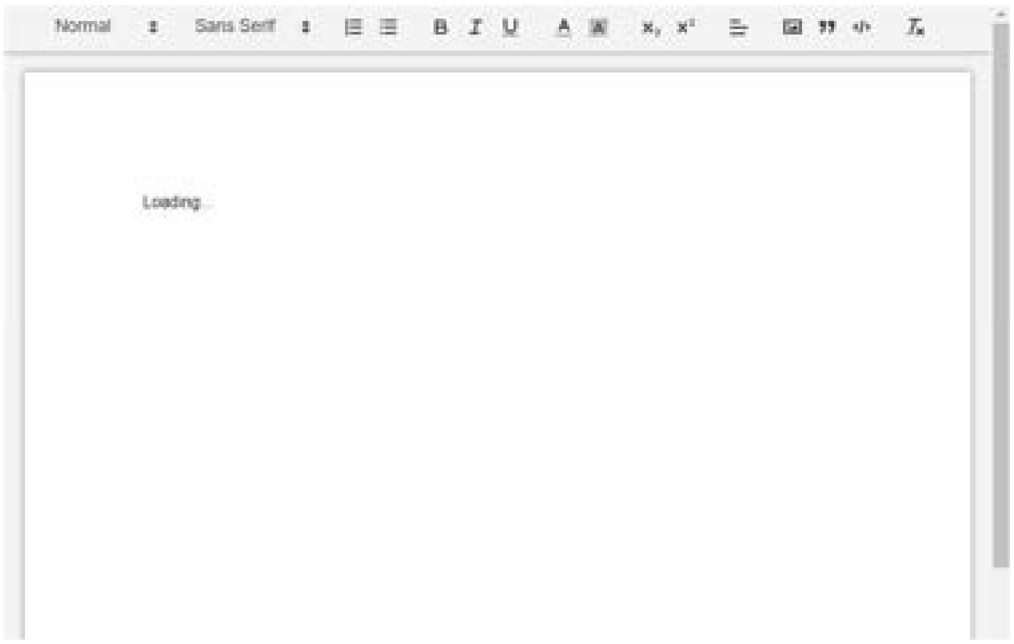


Figure 136 Text Editor V1

Figure 136 shows the original text editor design.

These pages contained the styling for the prototypes developed and will be further styled in a later sprint.

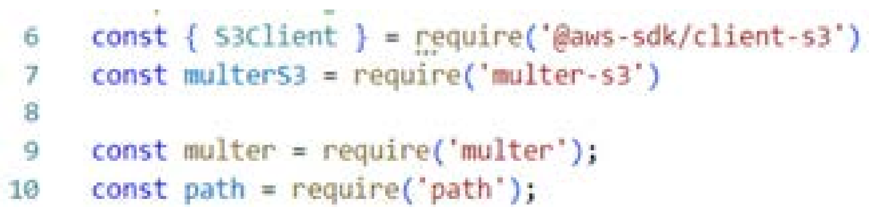
5.10.1.4 Item 4 – Include an image upload feature for documents.

The goal of this item was to allow users to upload images. The image upload feature was used when uploading images for documents, thus it is a prominent feature that must be developed.

A file is first created in the utils folder, called image_upload.js, seen in figure 137.

Figure 137 Utils Folder

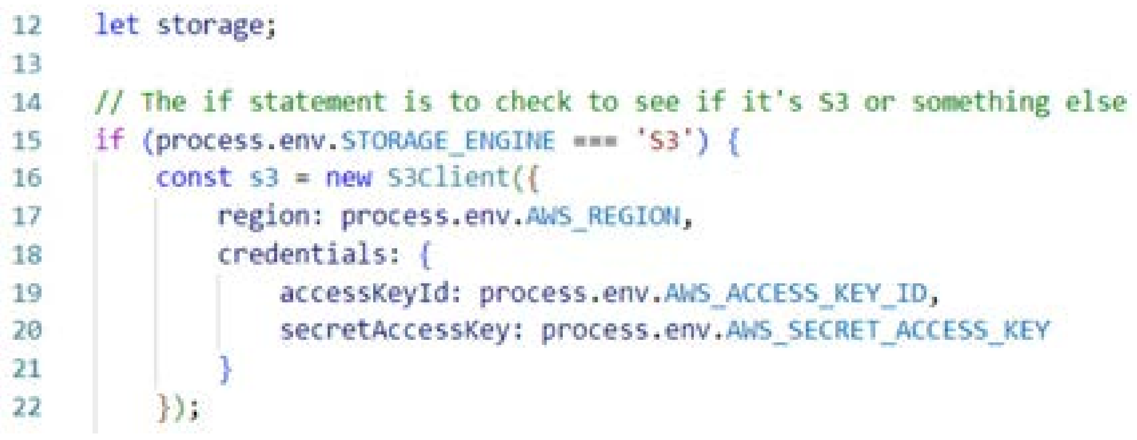
In this file, multer-s3 and s3-client are initiated. These are packages that are required to be installed and will allow users to connect to AWS S3 Buckets and use that as a place to store their images, rather than the local desktop folder.



```
const { S3Client } = require('@aws-sdk/client-s3')
const multerS3 = require('multer-s3')

const multer = require('multer');
const path = require('path');
```

Figure 138 Multer-s3

The figure above shows the code requiring multer-s3 and the S3-Client.

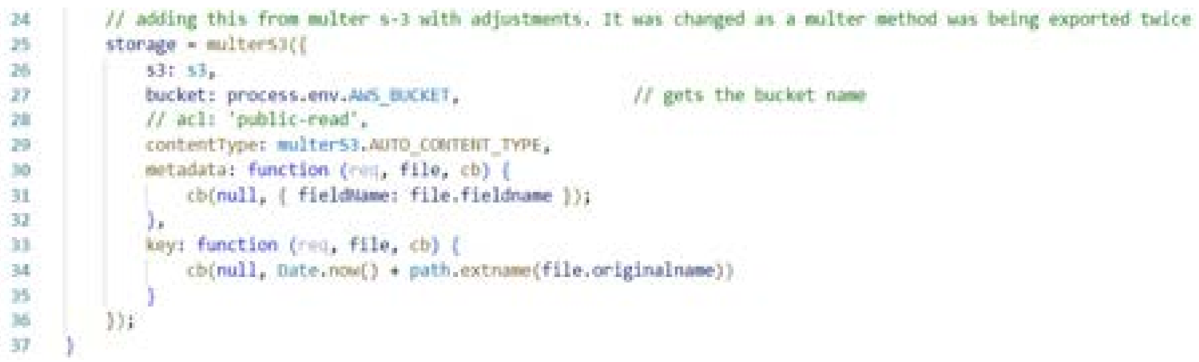


```
12  let storage;
13
14  // The if statement is to check to see if it's S3 or something else
15  if (process.env.STORAGE_ENGINE === 'S3') {
16      const s3 = new S3Client({
17          region: process.env.AWS_REGION,
18          credentials: {
19              accessKeyId: process.env.AWS_ACCESS_KEY_ID,
20              secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY
21          }
22      });
```

Figure 139 "If" Statement

Figure 139 then shows an if statement, connecting to the AWS Storage system. It uses the constants from "process.env", which includes the storage engine, the access key, and the secret access key that is given when creating a bucket in AWS.

Storage is also declared in line 12, and is used in the following code:



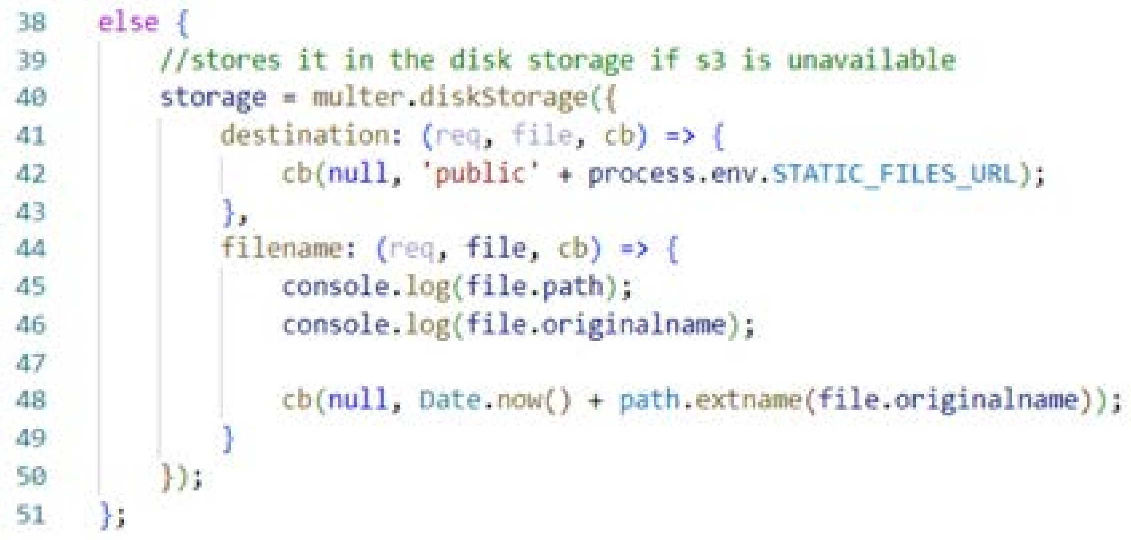
```
    // adding this from multer s-3 with adjustments. It was changed as a multer method was being exported twice
    storage = multerS3({
        s3: s3,
        bucket: process.env.AWS_BUCKET,             // gets the bucket name
        // acl: 'public-read',
        contentType: multerS3.AUTO_CONTENT_TYPE,
        metadata: function (req, file, cb) {
            cb(null, { fieldName: file.fieldname });
        },
        key: function (req, file, cb) {
            cb(null, Date.now() + path.extname(file.originalname))
        }
    });
}
```

Figure 140 Second Half of "if" Statement

Figure 140 shows the second half of the if statement. It uses multerS3 as a storage, and uses the settings given in the Multer documentation seen here: (multer-s3, 2022)

This was added with s3 adjustments in mind. The code defines the settings for uploading files to AWS S3 Bucket using Multer Middleware, which includes the name, the content type, the data, and the key, which generates a unique file with the date timestamp.

If it cannot connect to AWS, an else statement is created to store it locally.

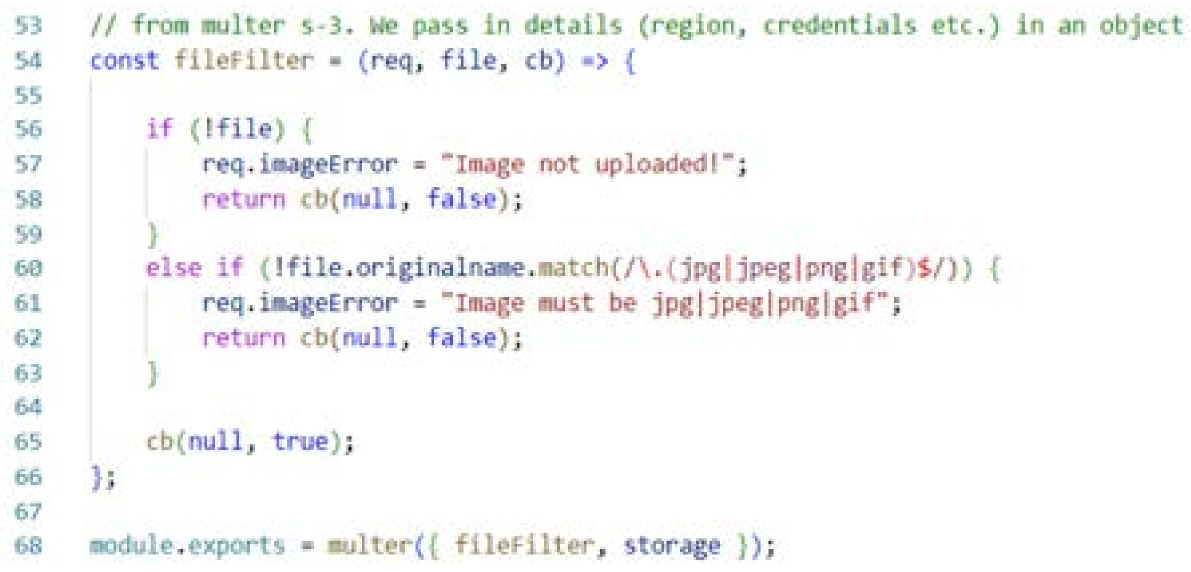


```
else {
    //stores it in the disk storage if s3 is unavailable
    storage = multer.diskStorage({
        destination: (req, file, cb) => {
            cb(null, 'public' + process.env.STATIC_FILES_URL);
        },
        filename: (req, file, cb) => {
            console.log(file.path);
            console.log(file.originalname);

            cb(null, Date.now() + path.extname(file.originalname));
        }
    });
};
```

Figure 141 "else" Statement

Figure 141 shows this else statement, storing it to the local disk storage, instead of AWS Server.

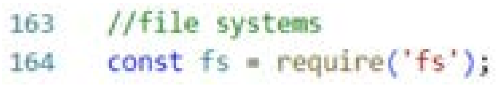
A file filter function is also required to ensure that the file is specifically an image and can be seen in figure 142 below.



```
// from multer s-3. We pass in details (region, credentials etc.) in an object
const fileFilter = (req, file, cb) => {

    if (!file) {
        req.imageError = "Image not uploaded!";
        return cb(null, false);
    }
    else if (!file.originalname.match(/\.(jpg|jpeg|png|gif)$/)) {
        req.imageError = "Image must be jpg|jpeg|png|gif";
        return cb(null, false);
    }

    cb(null, true);
};

module.exports = multer({ fileFilter, storage });
```

Figure 142 File Filter

To allow for an image upload to be used, changes were made to the code of the document_controller.js file to cater for this.

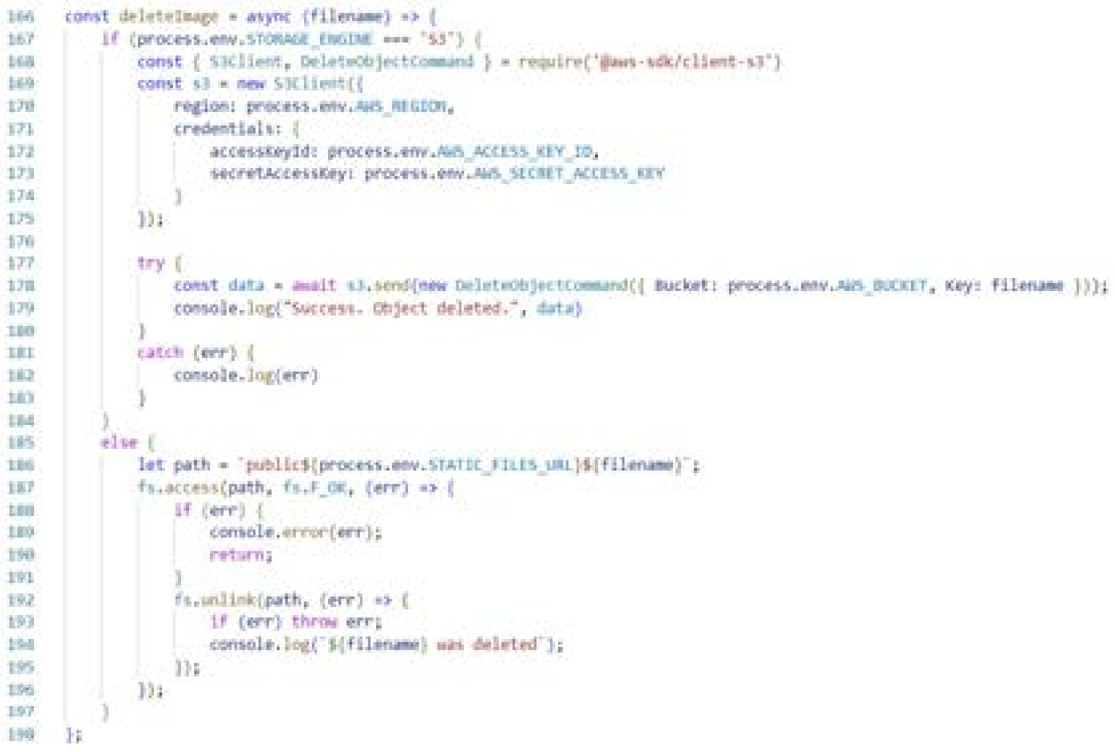


```
//file systems
const fs = require('fs');
```

Figure 143 File System

File systems were first added. This is seen in the figure above.

A deleteImage function was then created, which will be used with the updated functions.

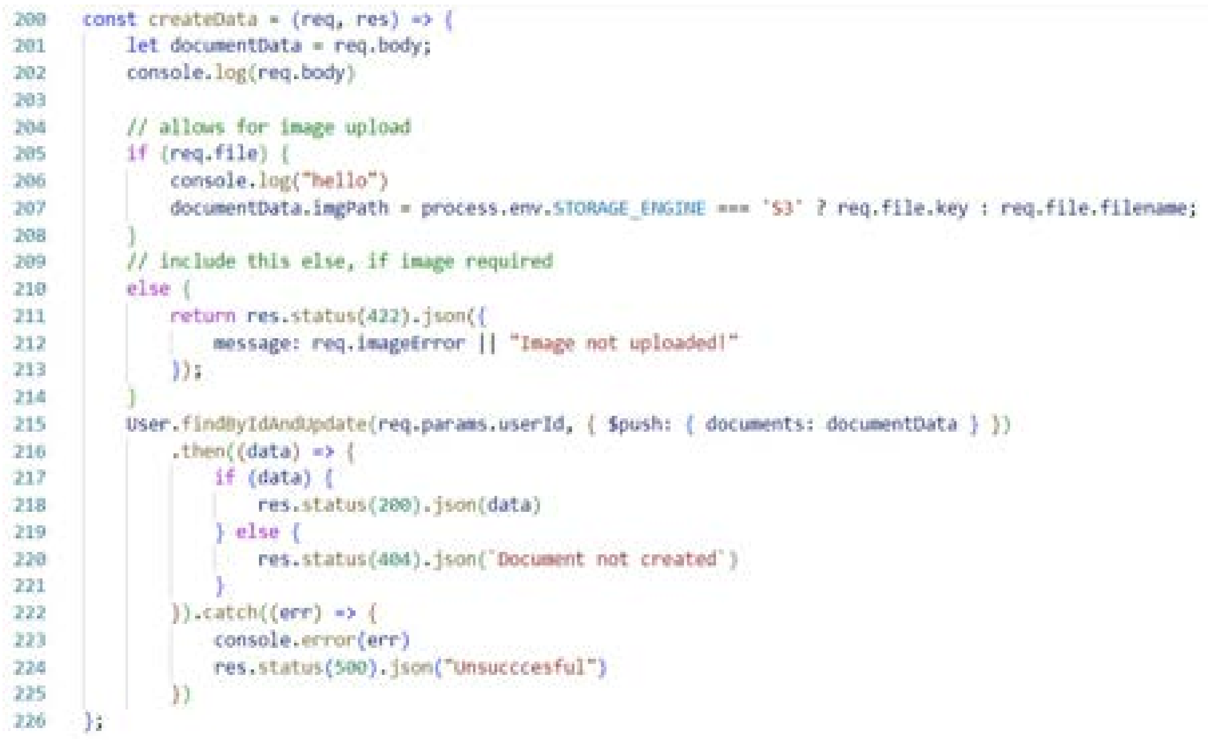


```
const deleteImage = async (filename) => {
    if (process.env.STORAGE_ENGINE === "S3") {
        const { S3Client, DeleteObjectCommand } = require('@aws-sdk/client-s3')
        const s3 = new S3Client({
            region: process.env.AWS_REGION,
            credentials: {
                accessKeyId: process.env.AWS_ACCESS_KEY_ID,
                secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY
            }
        });

        try {
            const data = await s3.send(new DeleteObjectCommand({ Bucket: process.env.AWS_BUCKET, Key: filename }));
            console.log("Success. Object deleted.", data)
        }
        catch (err) {
            console.log(err)
        }
    }
    else {
        let path = `public${process.env.STATIC_FILES_URL}${filename}`;
        fs.access(path, fs.F_OK, (err) => {
            if (err) {
                console.error(err);
                return;
            }
            fs.unlink(path, (err) => {
                if (err) throw err;
                console.log(`${filename} was deleted`);
            });
        });
    }
};
```

Figure 144 Delete Image Function

Figure 144 shows the deleteImage function. It first receives a filename as a parameter. The if statement then ensures that the STORAGE_ENGINE is set to "S3". It then deletes the image that is found in the AWS SDK, or the local storage system. It then logs a success or failure message depending on the result.

CreateData was updated accordingly, to cater for the image upload when creating a new document. This can be seen in the code below:



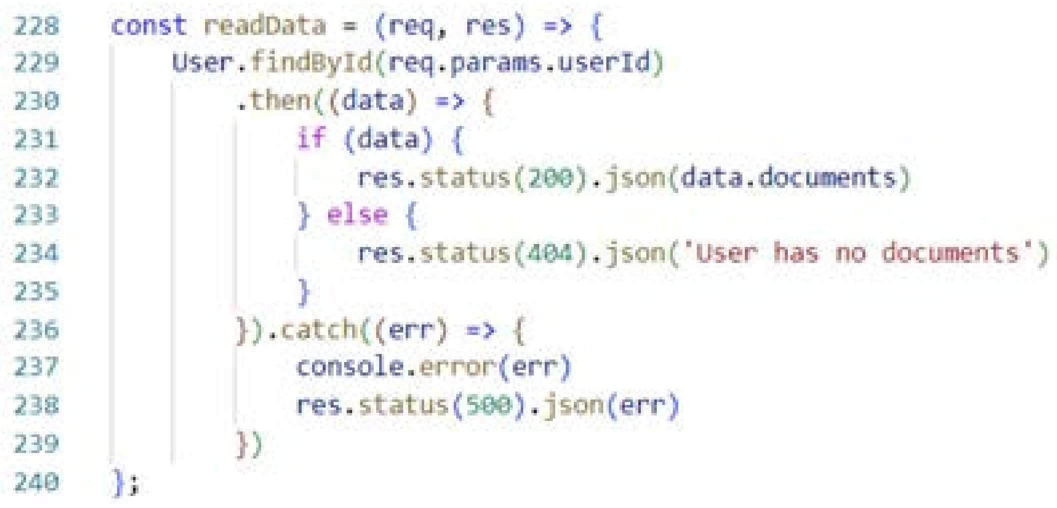
```
200  const createData = (req, res) => {
201      let documentData = req.body;
202      console.log(req.body)
203
204      // allows for image upload
205      if (req.file) {
206          console.log("hello")
207          documentData.imgPath = process.env.STORAGE_ENGINE === 'S3' ? req.file.key : req.file.filename;
208      }
209      // include this else, if image required
210      else {
211          return res.status(422).json({
212              message: req.imageError || "Image not uploaded!"
213          });
214      }
215      User.findByIdAndUpdate(req.params.userId, { $push: { documents: documentData } })
216          .then((data) => {
217              if (data) {
218                  res.status(200).json(data)
219              } else {
220                  res.status(404).json(`Document not created`)
221              }
222          }).catch((err) => {
223              console.error(err)
224              res.status(500).json("Unsucccesful")
225          })
226  };
```

Figure 145 Create Data Updated

Figure 145 now has an "if" statement that requests for the file, which will check whether the STORAGE_ENGINE variable has "S3", if it does it uploads it to AWS as a file key, and if not uploads it to the local storage. Otherwise, it returns an error.

Line 215 onwards then uploads the data.

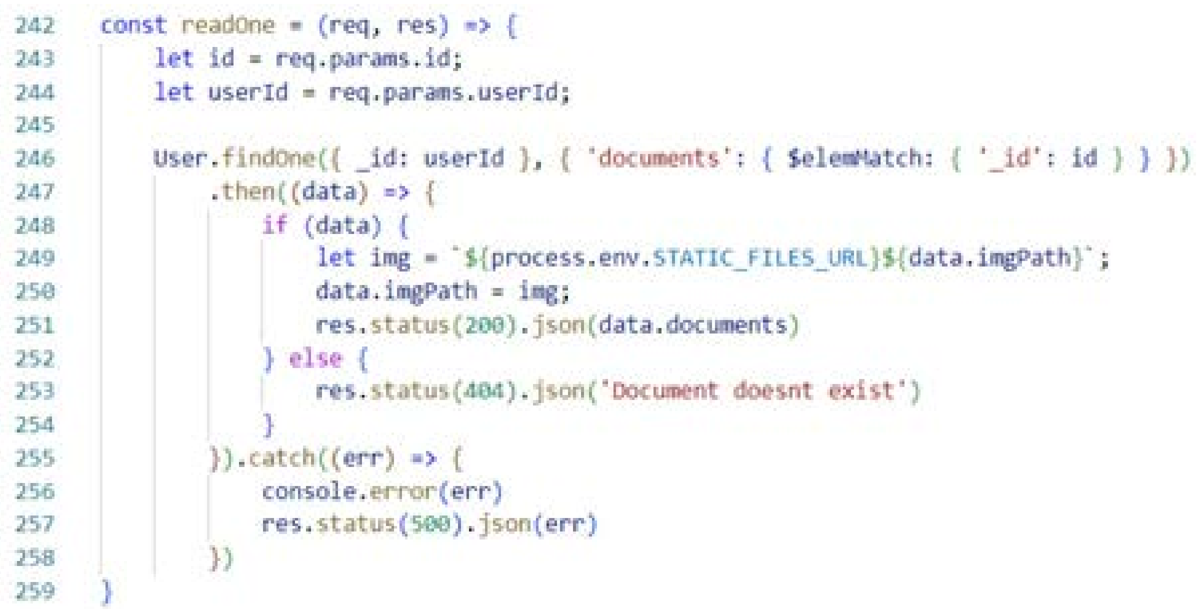
The ReadData function remains the same, and can be seen below:



```
const readData = (req, res) => {
    User.findById(req.params.userId)
        .then((data) => {
            if (data) {
                res.status(200).json(data.documents)
            } else {
                res.status(404).json('User has no documents')
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json(err)
        })
};
```

Figure 146 Read Data Updated

Minor changes were made to the “readOne” function and can be seen in figure 147:

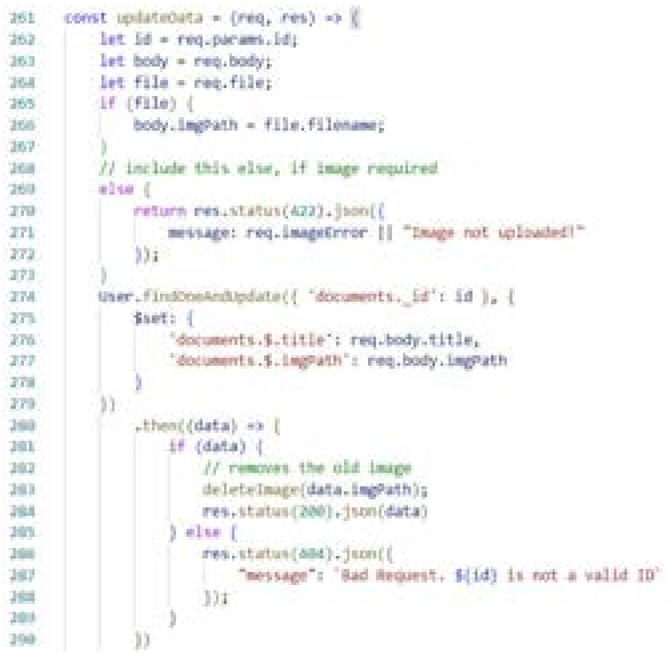


```
const readOne = (req, res) => {
    let id = req.params.id;
    let userId = req.params.userId;

    User.findOne({ _id: userId }, { 'documents': { $elemMatch: { '_id': id } } })
        .then((data) => {
            if (data) {
                let img = `${process.env.STATIC_FILES_URL}${data.imgPath}`;
                data.imgPath = img;
                res.status(200).json(data.documents)
            } else {
                res.status(404).json('Document doesnt exist')
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json(err)
        })
}
```

Figure 147 Read One Updated

Figure 147 shows the updated readOne function, and within the “if” statement, it takes the STATIC_FILES_URL constant in the .env file and combines it with the image path. The data.imgPath variable is then set to this, allowing the image to be returned, alongside the rest of the document details.

The update data function in this version of the code was also accepting images to be changed, however this was reverted to just accept the title of the document to be changed.

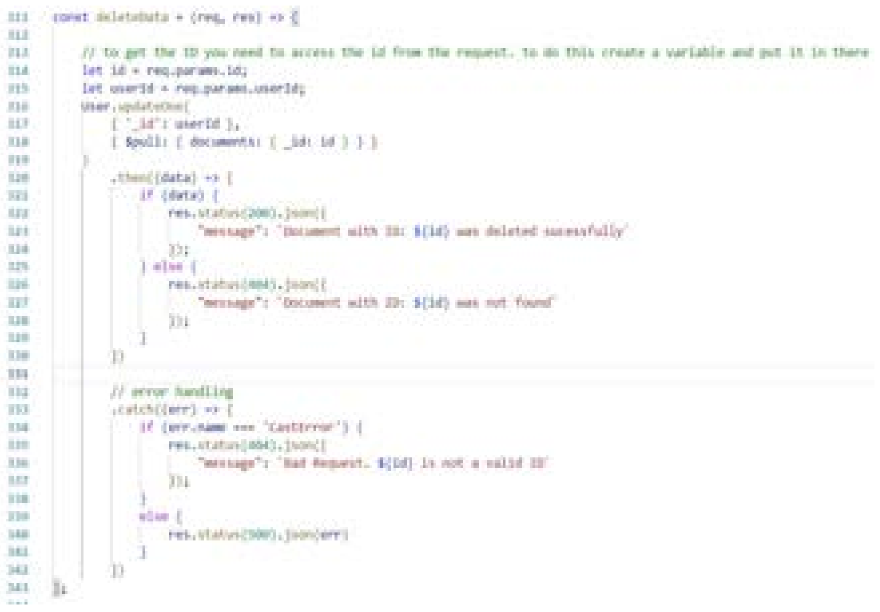


```
const updateData = (req, res) => {
    let id = req.params.id;
    let body = req.body;
    let file = req.file;
    if (file) {
        body.imgPath = file.filename;
    }
    // include this else, if image required
    else {
        return res.status(422).json({
            message: req.imageError || "Image not uploaded!"
        });
    }
    User.findOneAndUpdate({ 'documents._id': id }, {
        $set: {
            'documents.$.title': req.body.title,
            'documents.$.imgPath': req.body.imgPath
        }
    })
        .then((data) => {
            if (data) {
                // removes the old image
                deleteImage(data.imgPath);
                res.status(200).json(data)
            } else {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
        })
```

Figure 148 File Checker

The process however can still be seen in figure 148, where it checks to see if the file is available and updates it.

Finally, the delete data function was not altered and deletes the document with the id being found. This is seen in figure 149.

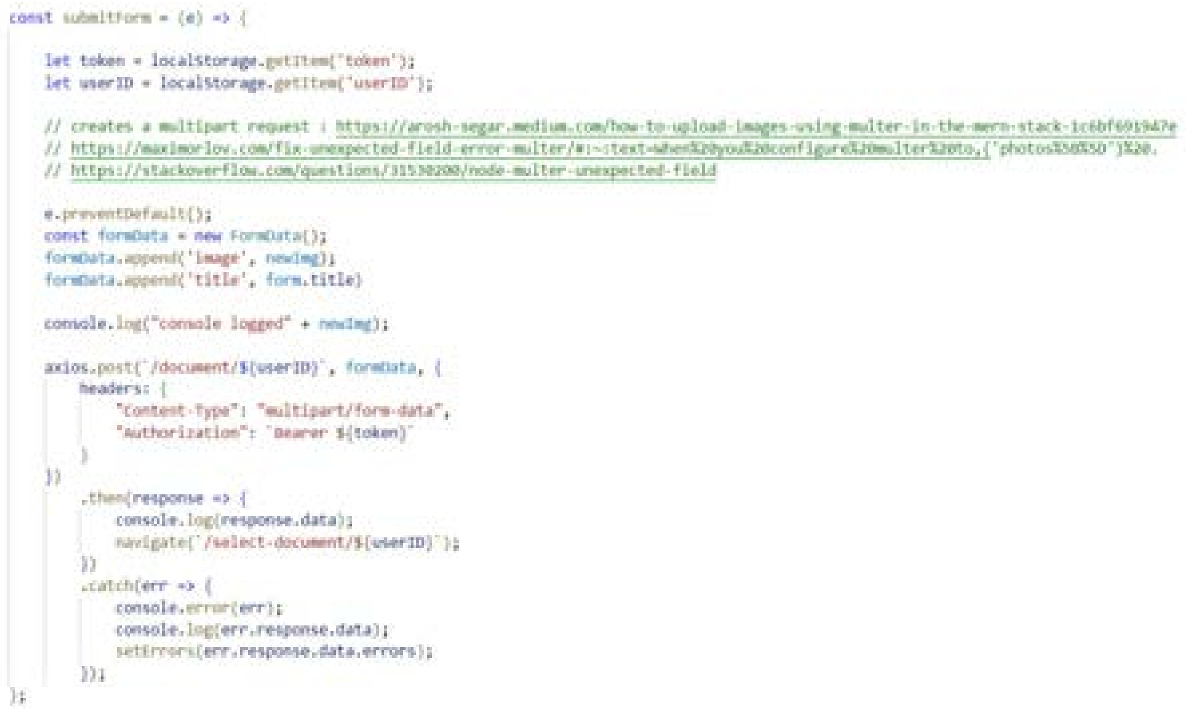


```
const deleteData = (req, res) => {

    // to get the ID you need to access the id from the request. to do this create a variable and put it in there
    let id = req.params.id;
    let userId = req.params.userId;
    User.updateOne(
        { '_id': userId },
        { $pull: { documents: { _id: id } } }
    )
        .then((data) => {
            if (data) {
                res.status(200).json({
                    "message": `Document with ID: ${id} was deleted sucessfully`
                });
            } else {
                res.status(404).json({
                    "message": `Document with ID: ${id} was not found`
                });
            }
        })

        // error handling
        .catch((err) => {
            if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
            else {
                res.status(500).json(err)
            }
        })
};
```

Figure 149 Delete Data

Now that the backend of the image upload has been completed, the frontend of the system will now be displayed.

The main changes made will be seen in the createDocuments.js file, where the submitForm method will be altered to allow the submission of images. This was done by creating a multipart form, that accepts images, and can be seen in figure 150:

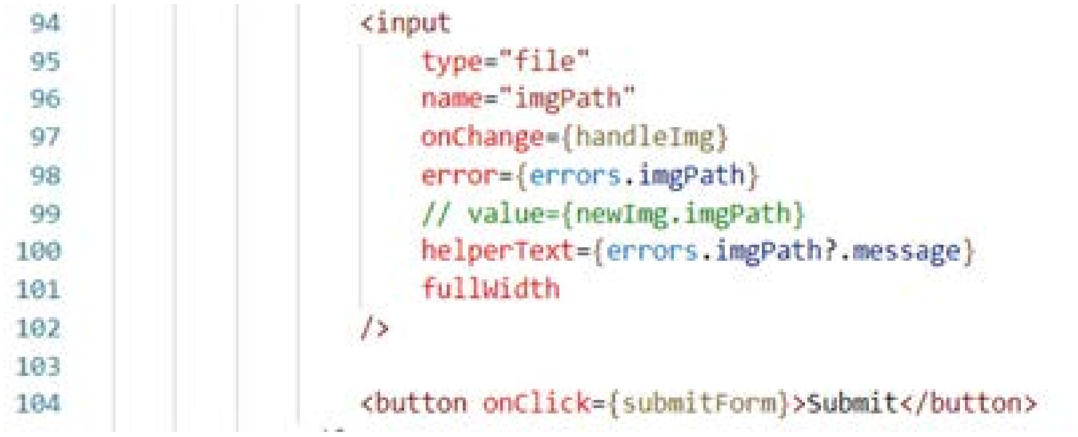


```
const submitForm = (e) => {

    let token = localStorage.getItem('token');
    let userID = localStorage.getItem('userID');

    // creates a multipart request : https://arosh-segar.medium.com/how-to-upload-images-using-multer-in-the-mern-stack-1c6bf691947e
    // https://maximorlov.com/fix-unexpected-field-error-multer/#:~:text=when%20you%20configure%20multer%20to,('photos%5B%5D')%20.
    // https://stackoverflow.com/questions/31530200/node-multer-unexpected-field

    e.preventDefault();
    const formData = new FormData();
    formData.append('image', newImg);
    formData.append('title', form.title)

    console.log("console logged" + newImg);

    axios.post(`/document/${userID}`, formData, {
        headers: {
            "Content-Type": "multipart/form-data",
            "Authorization": `Bearer ${token}`
        }
    })
        .then(response => {
            console.log(response.data);
            navigate(`/select-document/${userID}`);
        })
        .catch(err => {
            console.error(err);
            console.log(err.response.data);
            setErrors(err.response.data.errors);
        });
};
```

Figure 150 Form Updated

The form takes in the image and uploads it as a part of the form data. Since it is a multipart form, it can allow images to be accepted.

The input was also added in the form to allow for the acceptance of images seen in figure 151.



```
<input
    type="file"
    name="imgPath"
    onChange={handleImg}
    error={errors.imgPath}
    // value={newImg.imgPath}
    helperText={errors.imgPath?.message}
    fullWidth
/>

<button onClick={submitForm}>Submit</button>
```

Figure 151 Input Updated

The result of this allows for a successful image upload feature, that will also be used when uploading an image to the folders, as well as storing images for the OCR Reader.

5.10.1.5 Item 5 – Bug Fixes and Problems

The goal of this item is to discuss the different problems that occurred during the development of this sprint.

Problem 1:

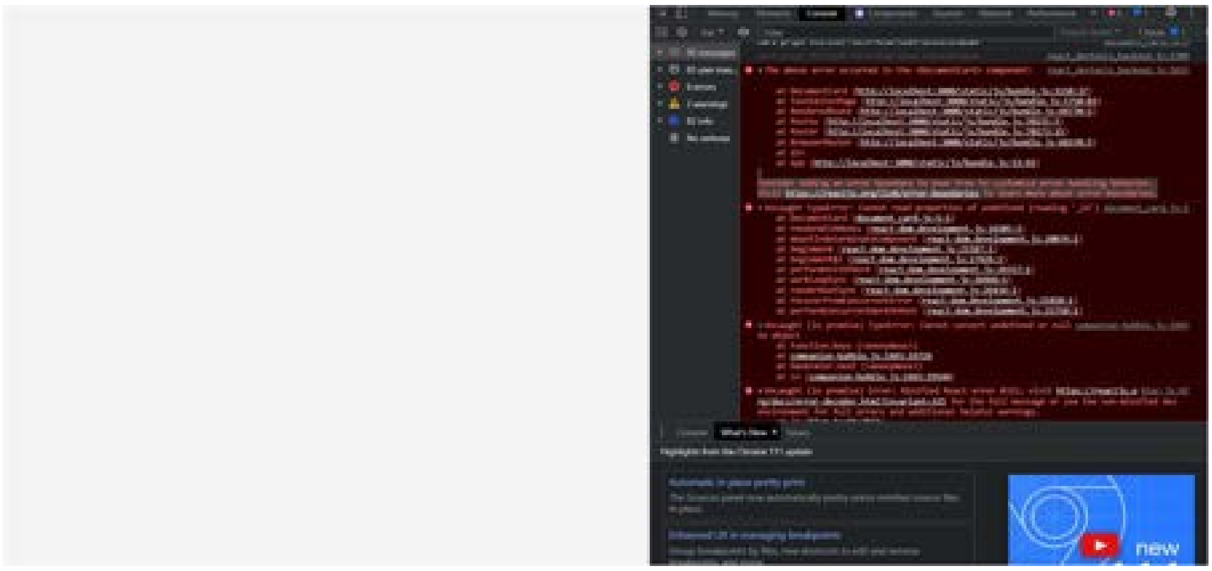
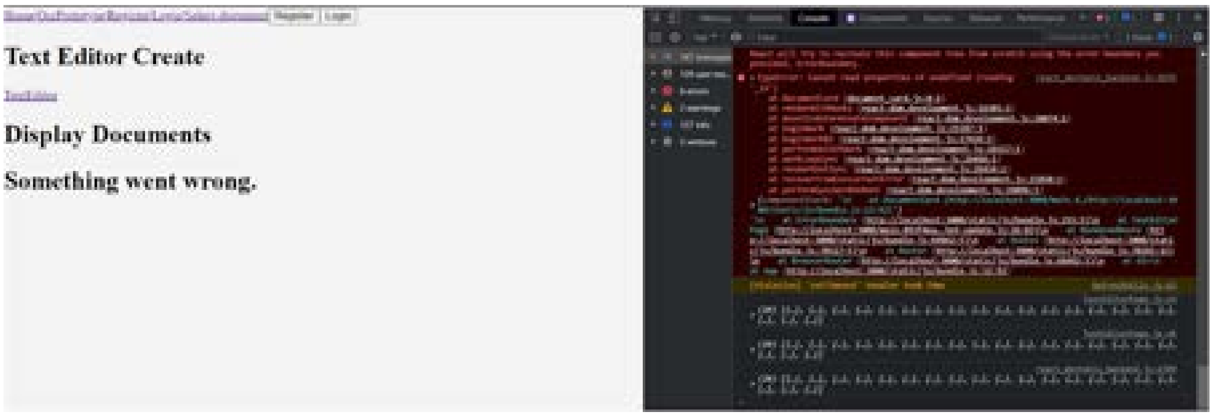


Figure 152 Problem (1)

Figure 152 displays a problem where the documents list was not appearing.

The first fix that was attempted was by adding an error boundary to the code.

This can be seen in figure 153:



Text Editor Create

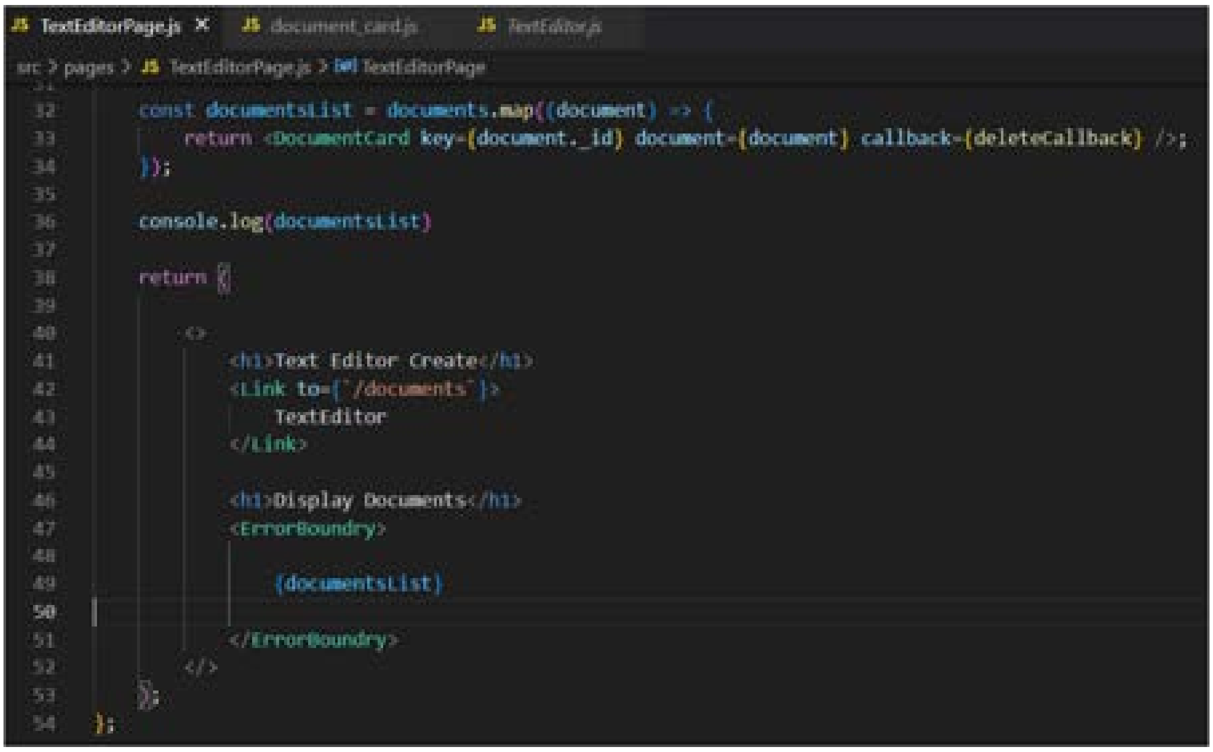
TextEditor

Display Documents

Something went wrong.

Figure 153 Result

The result of this brought back the pages, however despite this, the problem was still occurring. This was fixed by removing document card from the return, and instead putting "documentList" in between the error boundary. This can be seen in the figure below:



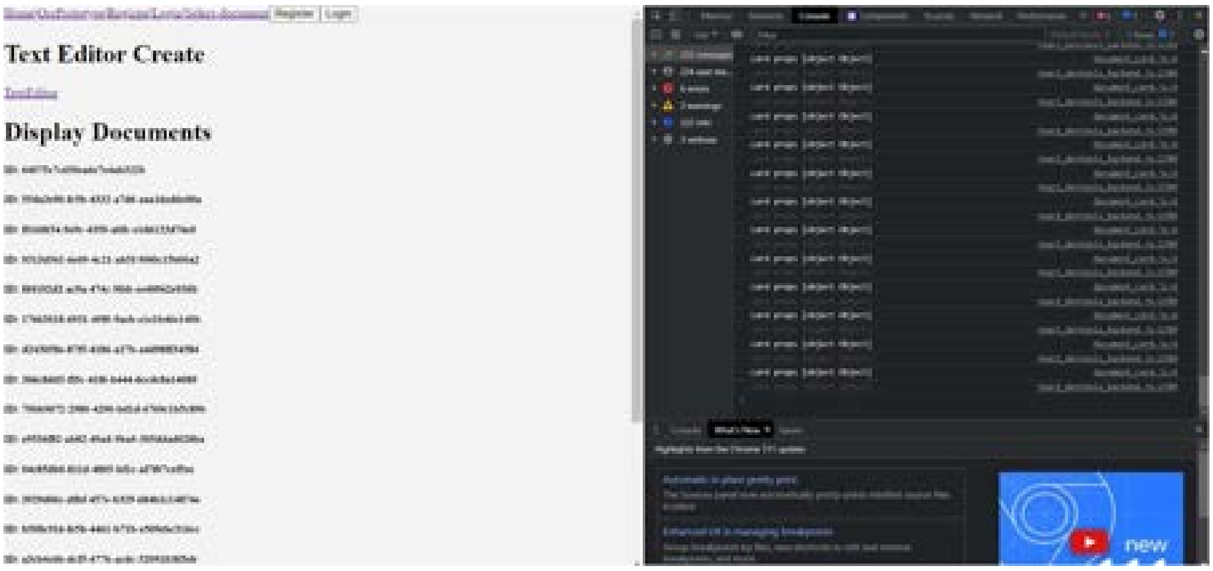
TextEditorPage.js × | document_card.js | TextEditor.js

src > pages > TextEditorPage.js > TextEditorPage

```
    const documentsList = documents.map((document) => {
        return <DocumentCard key={document._id} document={document} callback={deleteCallback} />;
    });

    console.log(documentsList)

    return (

        <>
            <h1>Text Editor Create</h1>
            <Link to={`/documents`}>
                TextEditor
            </Link>

            <h1>Display Documents</h1>
            <ErrorBoundry>

                {documentsList}


            </ErrorBoundry>
        </>
    );
};
```

Figure 154 Result

And the result can be seen in figure 155:



Text Editor Create

TextEditor

Display Documents

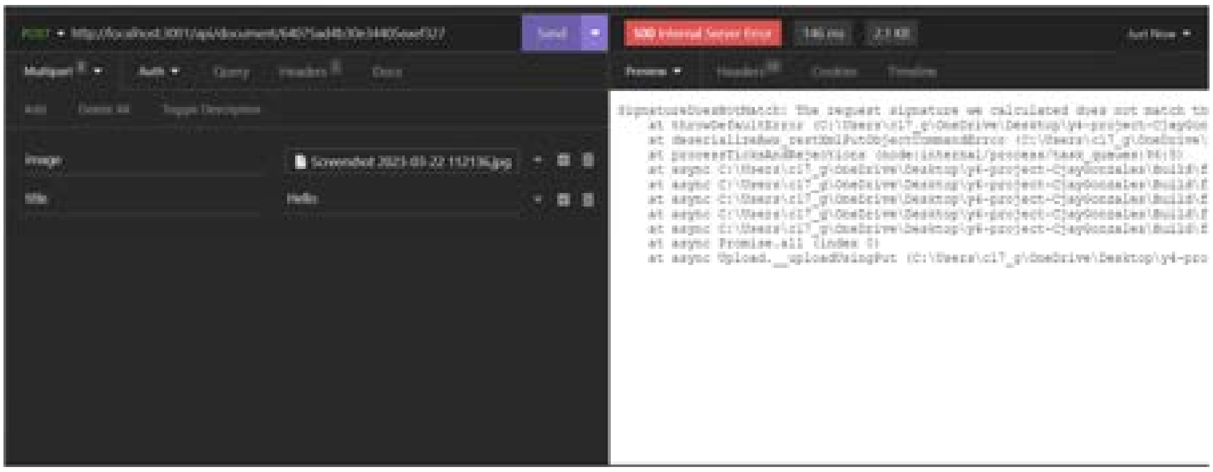
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]
ID: [illegible]

Figure 155 Result

The problem was solved, and the document id was being displayed for all documents.

Problem 2:

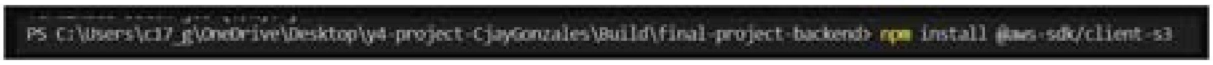
The next problem that occurred was during the development of the image upload feature. The error can be seen in figure 156:



```
SignatureDoesNotMatch: The request signature we calculated does not match th
    at throwDefaultError (C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGon
    at deserializeAws_restXmlPutObjectCommandError (C:\Users\c17_g\OneDrive\
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGonzales\Build\f
    at async C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGonzales\Build\f
    at async C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGonzales\Build\f
    at async C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGonzales\Build\f
    at async C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGonzales\Build\f
    at async Promise.all (index 0)
    at async Upload.__uploadUsingPut (C:\Users\c17_g\OneDrive\Desktop\y4-pro
```

Figure 156 Problem 2

This was occurring because aws-sdk/client-s3 was not installed on the system, and was fixed by an "npm install" command, as seen in figure 156:



```
PS C:\Users\c17_g\OneDrive\Desktop\y4-project-CjayGonzales\Build\final-project-backend> npm install @aws-sdk/client-s3
```

Figure 157 Result

Problem 3:

Problem 3 took slightly longer to fix. A Multer error was occurring, and did not let me upload my images, the error can be seen in figure 158:

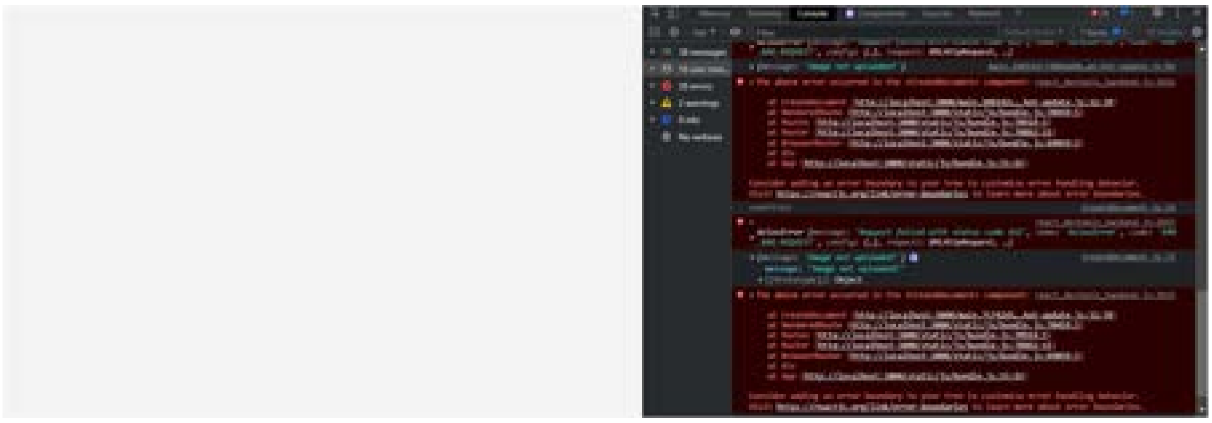
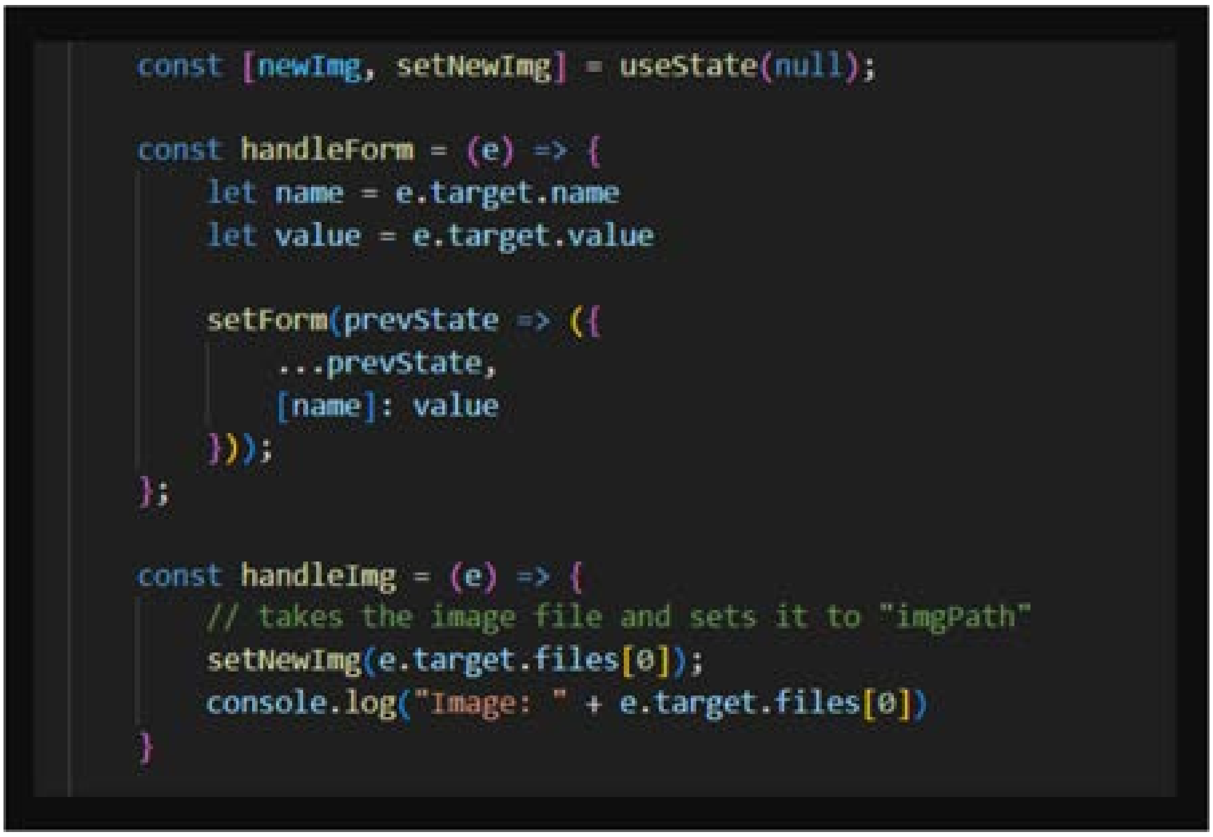


Figure 158 Problem 3

The problem was that the form didn't allow for images to be uploaded, and the frontend needed to be altered to accept a multiform data.

The handle image function was added to allow images to be uploaded seen in figure 159:

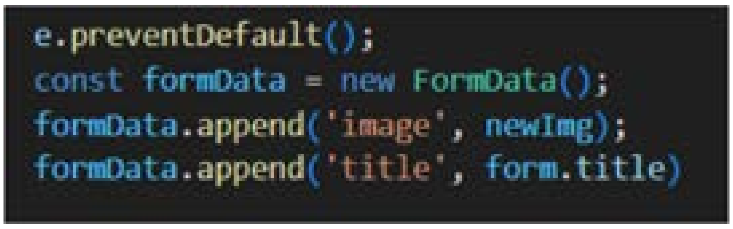


```
const [newImg, setNewImg] = useState(null);

const handleForm = (e) => {
    let name = e.target.name
    let value = e.target.value

    setForm(prevState => ({
        ...prevState,
        [name]: value
    }));
};

const handleImg = (e) => {
    // takes the image file and sets it to "imgPath"
    setNewImg(e.target.files[0]);
    console.log("Image: " + e.target.files[0])
}
```

Figure 159 Result

Figure 160 shows sets the image to newImage.

Once this was completed another error occurred, preventing the image upload. This was fixed by changing “imgPath” to “image” in the form.append line in the figure below:



```
e.preventDefault();
const formData = new FormData();
formData.append('image', newImg);
formData.append('title', form.title)
```

Figure 160 Result

Once this was completed, the image upload feature was fully functional, and the user can successfully upload images alongside the document.

5.11 Sprint 6 – Testing

5.11.1 Goal

The goal of this sprint was to continue with the implementation of the application, and to perform parts of user testing, fixing bugs when necessary, and preparing the application for user testing by implementing the UI elements. The items for this sprint are:

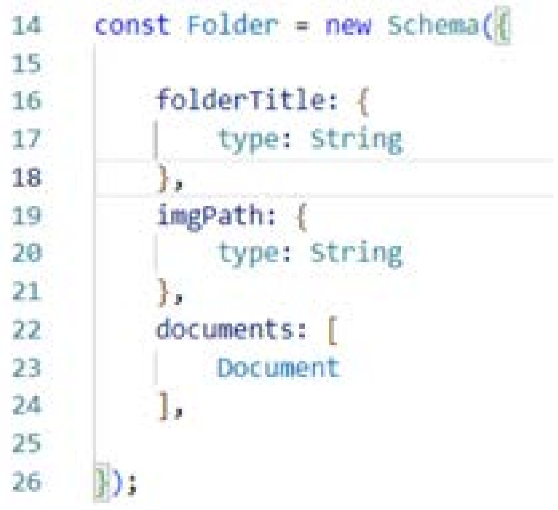
- **Item 1:** Developing the folders for documents, including CRUD and image upload for folders.
- **Item 2:** Adding Backend for OCR Reader, including CRUD, saving images and results, and displaying on a separate page.
- **Item 3:** Implementing UI Elements for the application.
- **Item 4:** User Testing
- **Item 5:** OCR Accuracy Testing
- **Item 6:** Bug Fixes and Problems

5.11.1.1 Item 1 – Developing the folders for documents, including CRUD and image upload for folders.

The goal of this item was to develop a folder structure, including crud functionalities and image upload for the folder system that the documents will be stored in. This was completed in a similar way to how the user can access their own data. For this reason, the code will not be discussed in as much detail, as it was already covered in detail in previous sprints. It will only be briefly covered.

The same can be said about the image upload feature, and will also be briefly covered, as it was explained in greater detail in a previous sprint.

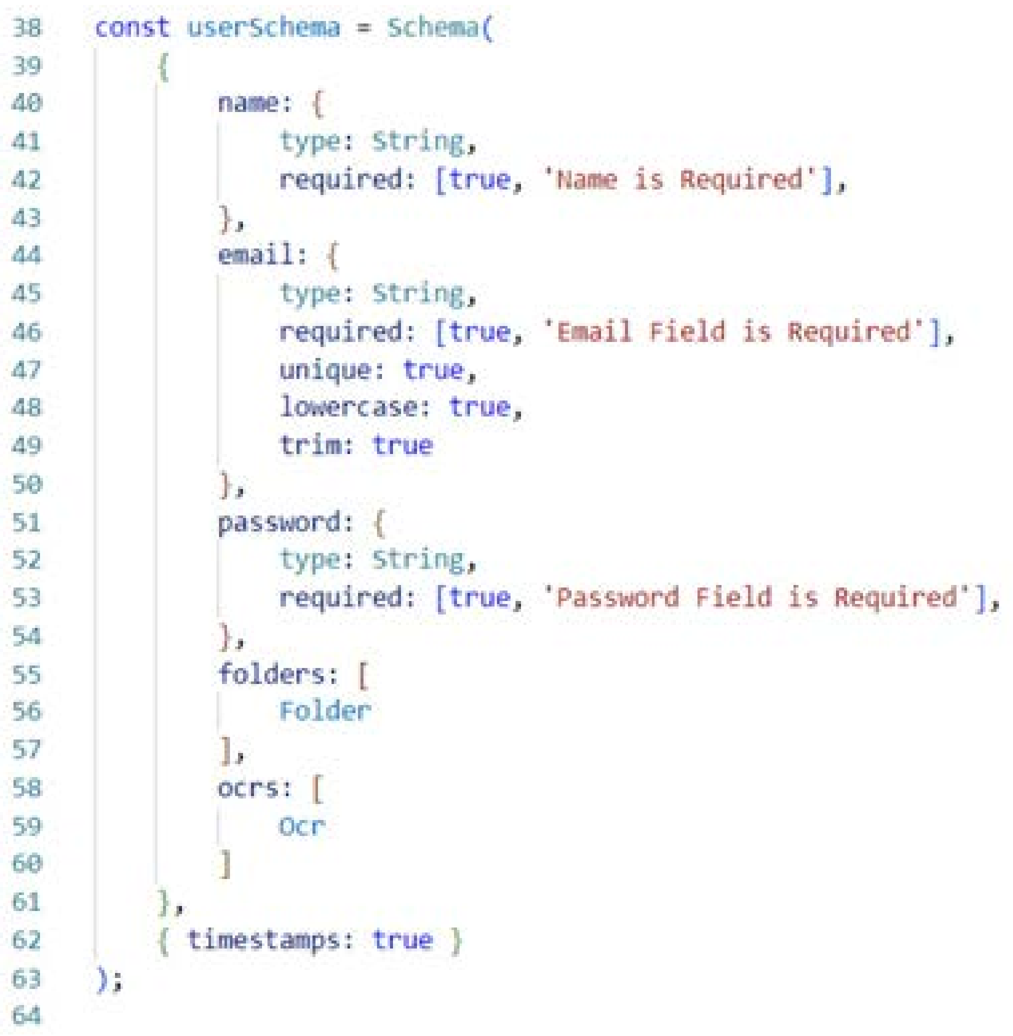
A folder schema was first created, then a reference was put in the user schema. This can be seen in the figure below.



```
const Folder = new Schema({

    folderTitle: {
        type: String
    },
    imgPath: {
        type: String
    },
    documents: [
        Document
    ],

});
```

Figure 161 Folder Schema

Figure 161 shows the folder schema, which contains the documents schema, containing the imgPath for the image upload. This is then put into the user schema, seen in the figure below.



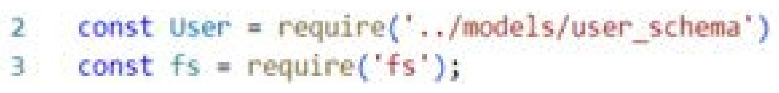
```
const userSchema = Schema(
    {
        name: {
            type: String,
            required: [true, 'Name is Required'],
        },
        email: {
            type: String,
            required: [true, 'Email Field is Required'],
            unique: true,
            lowercase: true,
            trim: true
        },
        password: {
            type: String,
            required: [true, 'Password Field is Required'],
        },
        folders: [
            Folder
        ],
        ocrs: [
            Ocr
        ]
    },
    { timestamps: true }
);
```

Figure 162 Updated User Schema

Figure 162 shows the updated user schema, containing the folders array. This is used to store the documents and folders to each specific user.

CRUD functionalities for the folder were then created. This was done in a similar way to how the CRUD for documents were made.

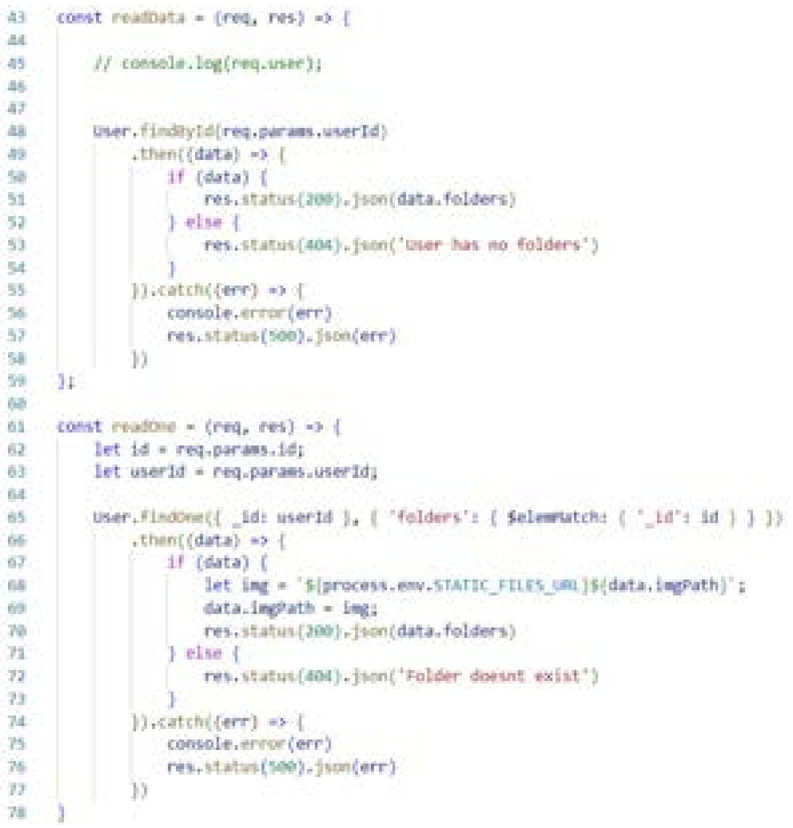
It first required the user_schema.js and the fileSystem, seen in the figure below.



```
const User = require('../models/user_schema')
const fs = require('fs');
```

Figure 163 File System

It then had similar functionality to how the document was read, however instead of the code accessing the document, it accesses the folder instead. This can be seen in figure 164.

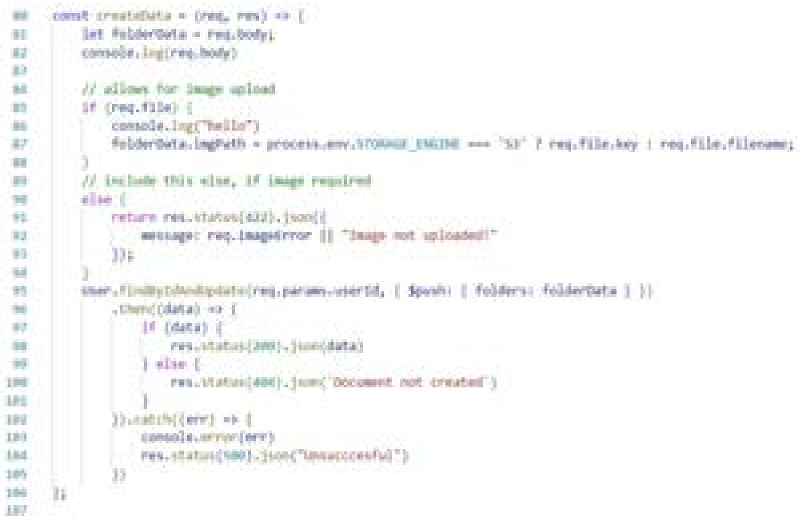


```
const readData = (req, res) => {

    // console.log(req.user);


    User.findById(req.params.userId)
        .then((data) => {
            if (data) {
                res.status(200).json(data.folders)
            } else {
                res.status(404).json('User has no folders')
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json(err)
        })
};

const readOne = (req, res) => {
    let id = req.params.id;
    let userId = req.params.userId;

    User.findOne({ _id: userId }, { 'folders': { $elemMatch: { "_id": id } } })
        .then((data) => {
            if (data) {
                let img = `${process.env.STATIC_FILES_URL}${data.imgPath}`;
                data.imgPath = img;
                res.status(200).json(data.folders)
            } else {
                res.status(404).json('Folder doesnt exist')
            }
        }).catch((err) => {
            console.error(err)
            res.status(500).json(err)
        })
}
```

Figure 164 Read One and Read Data

Figure 164 shows the readData, and readOne methods, accessing the folder instead of the documents.

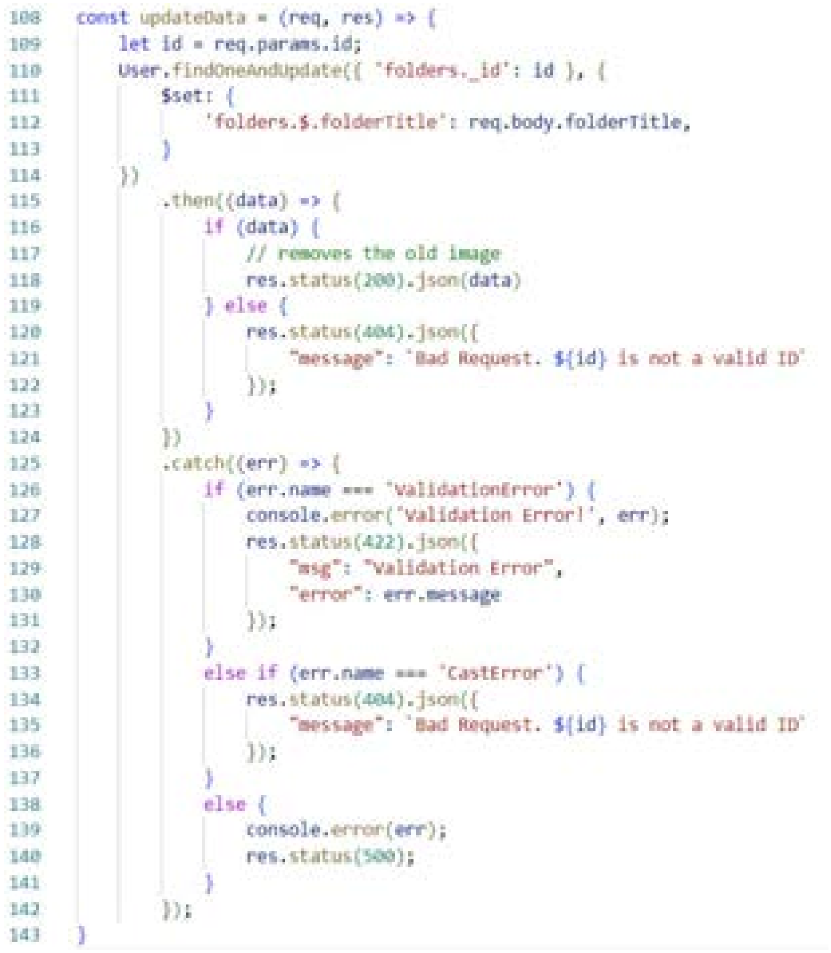
Create data was also similarly implemented, as it pushes it to the folder data to the folders array using the findByIdAndUpdate function (seen in line 95), with the image upload functionality from lines 85-94. The figure below shows the code for the createData function.



```
80  const createData = (req, res) => {
81      let folderData = req.body;
82      console.log(req.body)
83
84      // allows for image upload
85      if (req.file) {
86          console.log("hello")
87          folderData.imgPath = process.env.STORAGE_ENGINE === 'S3' ? req.file.key : req.file.filename;
88      }
89      // include this else, if image required
90      else {
91          return res.status(422).json({
92              message: req.imageError || "Image not uploaded!"
93          });
94      }
95      User.findByIdAndUpdate(req.params.userId, { $push: { folders: folderData } })
96          .then((data) => {
97              if (data) {
98                  res.status(200).json(data)
99              } else {
100                 res.status(404).json('Document not created')
101             }
102         }).catch((err) => {
103             console.error(err)
104             res.status(500).json("Unsuccessful")
105         })
106 };
107
```

Figure 165 Create Data

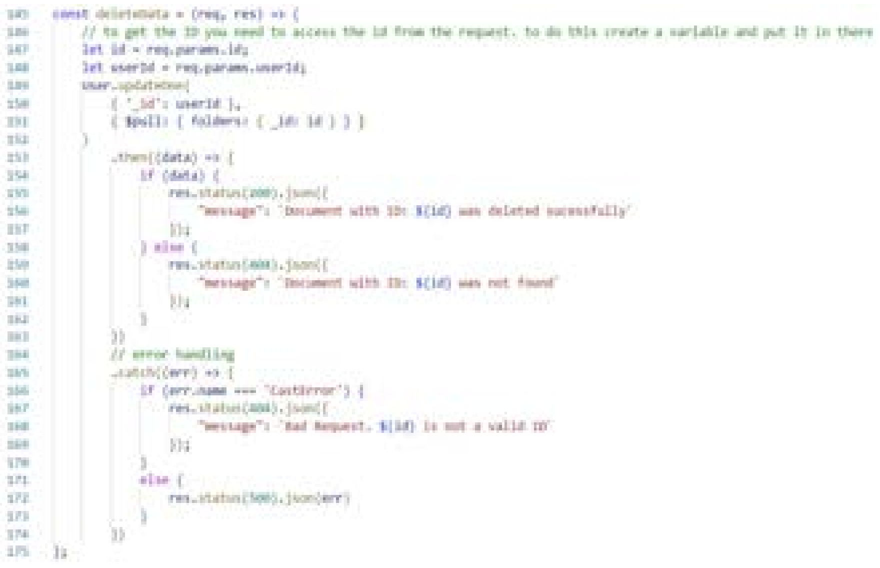
Update data was then implemented in a similar way, seen in figure 166, requesting the folderTitle, and containing the image functionalities.



```
const updateData = (req, res) => {
    let id = req.params.id;
    User.findOneAndUpdate({ 'folders._id': id }, {
        $set: {
            'folders.$.folderTitle': req.body.folderTitle,
        }
    })
        .then((data) => {
            if (data) {
                // removes the old image
                res.status(200).json(data)
            } else {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
        })
        .catch((err) => {
            if (err.name === 'ValidationError') {
                console.error('Validation Error!', err);
                res.status(422).json({
                    "msg": "Validation Error",
                    "error": err.message
                });
            }
            else if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
            else {
                console.error(err);
                res.status(500);
            }
        });
}
```

Figure 166 Update Data

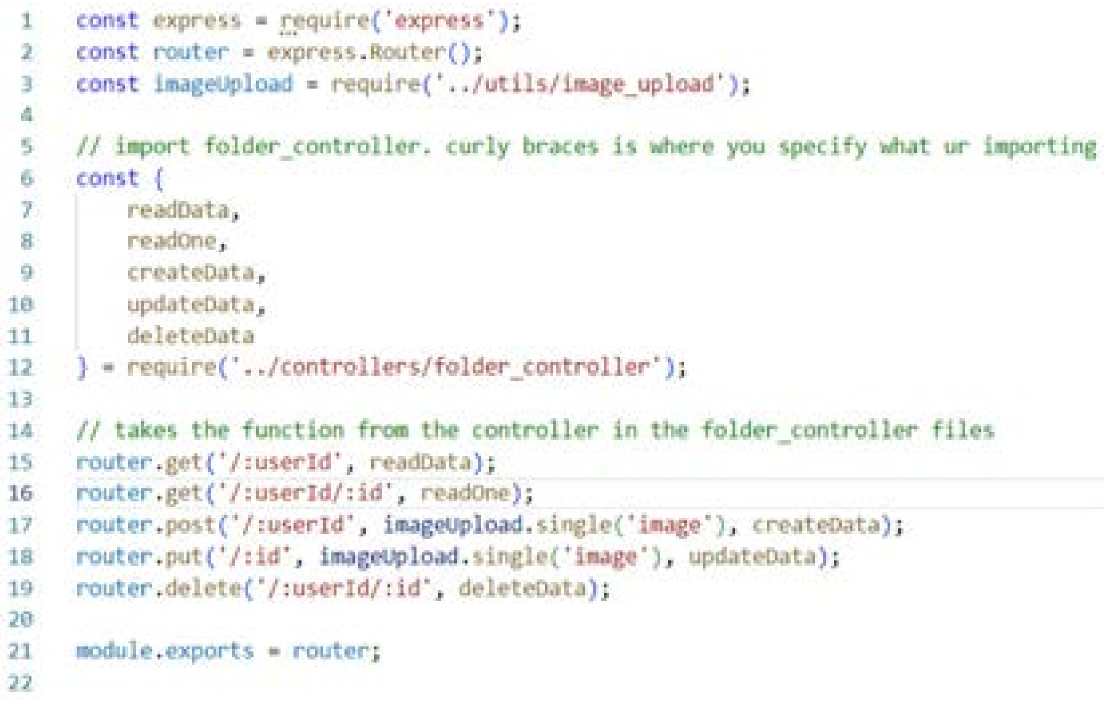
Finally, the delete data function, seen in figure 167.



```
const deleteData = (req, res) => {
    // to get the ID you need to access the id from the request. to do this create a variable and put it in there
    let id = req.params.id;
    let userId = req.params.userId;
    User.updateOne(
        { '_id': userId },
        { $pull: { folders: { _id: id } } }
    )
        .then((data) => {
            if (data) {
                res.status(200).json({
                    "message": `Document with ID: ${id} was deleted successfully`
                });
            } else {
                res.status(404).json({
                    "message": `Document with ID: ${id} was not found`
                });
            }
        })
        // error handling
        .catch((err) => {
            if (err.name === 'CastError') {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
            else {
                res.status(500).json(err)
            }
        })
};
```

Figure 167 Delete Data

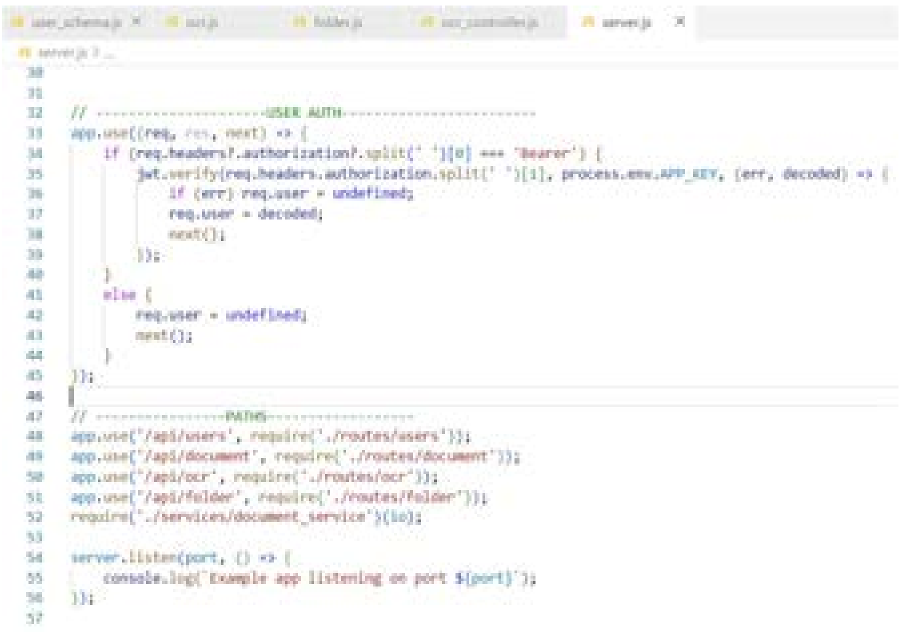
The routes for the folders were then created and can be seen in figure 168.



```javascript
const express = require('express');
const router = express.Router();
const imageUpload = require('../utils/image_upload');

// import folder_controller. curly braces is where you specify what ur importing
const {
    readData,
    readOne,
    createData,
    updateData,
    deleteData
} = require('../controllers/folder_controller');

// takes the function from the controller in the folder_controller files
router.get('/:userId', readData);
router.get('/:userId/:id', readOne);
router.post('/:userId', imageUpload.single('image'), createData);
router.put('/:id', imageUpload.single('image'), updateData);
router.delete('/:userId/:id', deleteData);

module.exports = router;
```

Figure 168 Routes

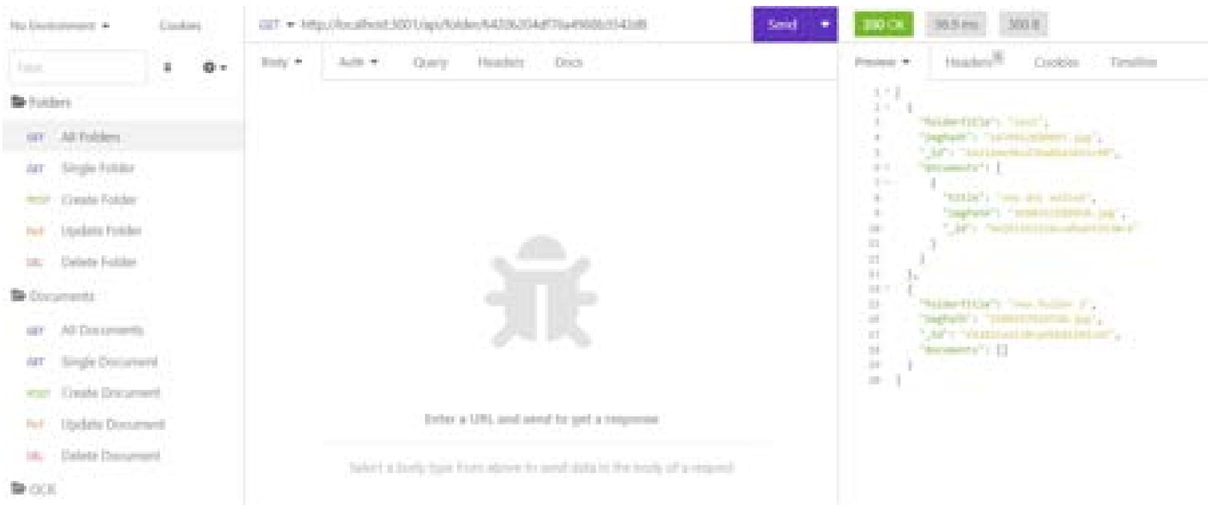
The paths are then set in the server.js file, and can be seen in figure 169:



```javascript


// ----------------------USER AUTH------------------------
app.use((req, res, next) => {
    if (req.headers?.authorization?.split(' ')[0] === 'Bearer') {
        jwt.verify(req.headers.authorization.split(' ')[1], process.env.APP_KEY, (err, decoded) => {
            if (err) req.user = undefined;
            req.user = decoded;
            next();
        });
    }
    else {
        req.user = undefined;
        next();
    }
});

// ----------------PATHS------------------
app.use('/api/users', require('./routes/users'));
app.use('/api/document', require('./routes/document'));
app.use('/api/ocr', require('./routes/ocr'));
app.use('/api/folder', require('./routes/folder'));
require('./services/document_service')(io);

server.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
});
```

Figure 169 Paths

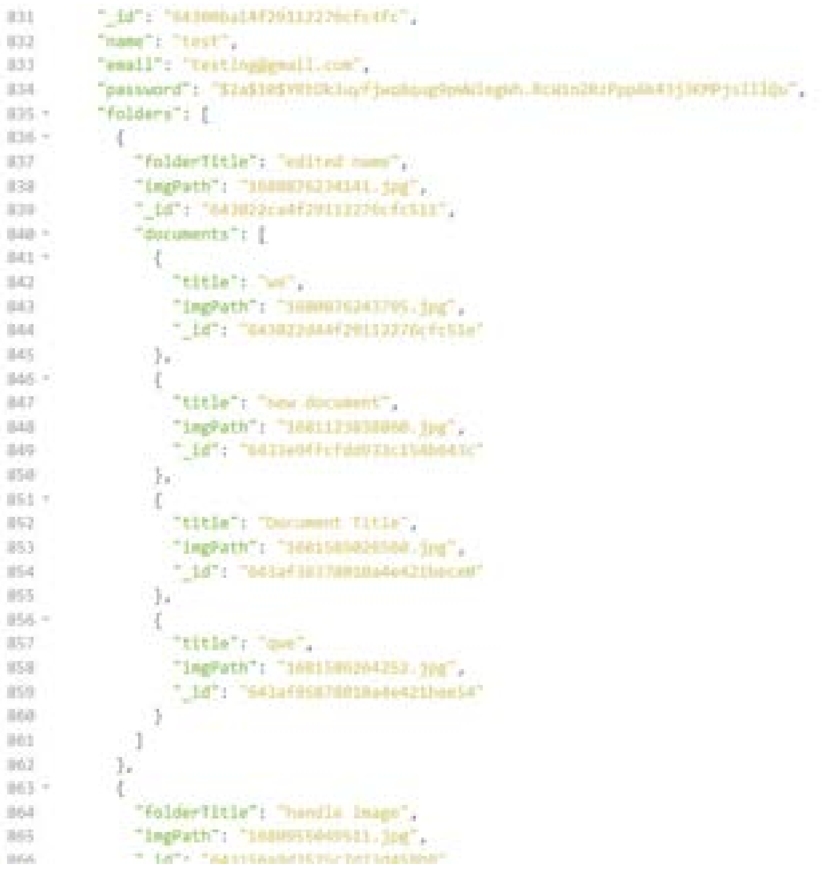
After some bug fixes, the folder structure now works in the insomnia backend, and can be seen in figure 170:



```
[
  {
    "folderTitle": "test",
    "imgPath": "[illegible].jpg",
    "_id": "[illegible]",
    "documents": [
      {
        "title": "new doc edited",
        "imgPath": "[illegible].jpg",
        "_id": "[illegible]"
      }
    ]
  },
  {
    "folderTitle": "new folder 2",
    "imgPath": "[illegible].jpg",
    "_id": "[illegible]",
    "documents": []
  }
]
```

Figure 170 Insomnia

The documents are now being stored within the folders, and when the user's information is viewed, the folders and documents are now being stored with the user. This is seen in figure 171.



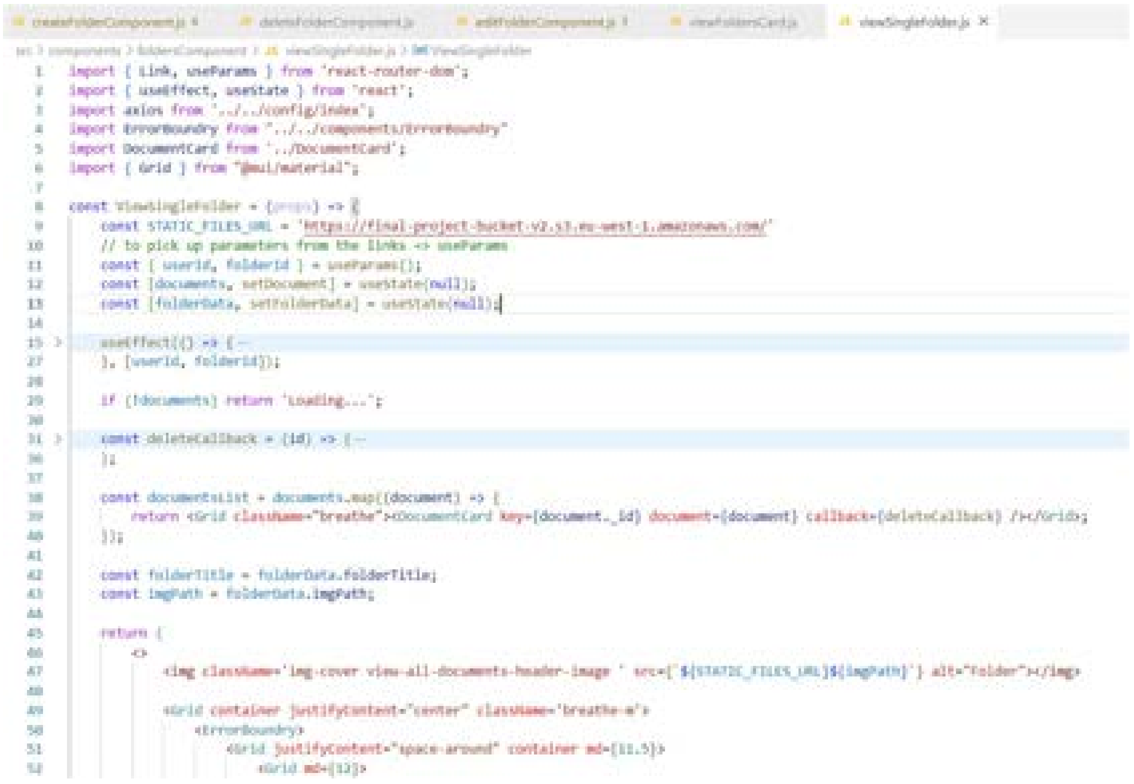
```
    "_id": "64300ba14f29112276cfc4fc",
    "name": "test",
    "email": "testing@gmail.com",
    "password": "$2a$10$[illegible]",
    "folders": [
      {
        "folderTitle": "edited name",
        "imgPath": "1680876234141.jpg",
        "_id": "643022ca4f29112276cfc511",
        "documents": [
          {
            "title": "we",
            "imgPath": "1680876243795.jpg",
            "_id": "643022d44f29112276cfc51e"
          },
          {
            "title": "new document",
            "imgPath": "1681123838060.jpg",
            "_id": "6431e04fcfdd033c1540641c"
          },
          {
            "title": "Document Title",
            "imgPath": "1681580026560.jpg",
            "_id": "643af3817801 8a4e421bece0"
          },
          {
            "title": "qwe",
            "imgPath": "1681580264252.jpg",
            "_id": "643af9587801 8a4e421bee54"
          }
        ]
      },
      {
        "folderTitle": "handle image",
        "imgPath": "1680955049511.jpg",
        "_id": "[illegible]"
```

Figure 171 Insomnia Result

Figure 171 shows a user account, with a folders array, containing many folders, and each folder containing many documents.

The image path is also being stored in each folder.

CRUD functionalities were then built in the frontend of the application and are similar in how the structure of the frontend for the documents were created. As that was explained in greater detail in a previous sprint, only a screenshot of one of the pages, and the filenames will be shown. This can be seen in the figure below.



createFolderComponent.js · deleteFolderComponent.js · editFolderComponent.js · viewFoldersCard.js · viewSingleFolder.js

src > components > foldersComponent > viewSingleFolder.js > ViewSingleFolder

```
import { Link, useParams } from 'react-router-dom';
import { useEffect, useState } from 'react';
import axios from '../../config/index';
import ErrorBoundry from "../../components/ErrorBoundry"
import DocumentCard from '../DocumentCard';
import { Grid } from "@mui/material";

const ViewSingleFolder = (props) => {
    const STATIC_FILES_URL = 'https://final-project-bucket-v2.s3.eu-west-1.amazonaws.com/'
    // to pick up parameters from the links -> useParams
    const { userId, folderId } = useParams();
    const [documents, setDocument] = useState(null);
    const [folderData, setFolderData] = useState(null);

    useEffect(() => {…
    }, [userId, folderId]);

    if (!documents) return 'Loading...';

    const deleteCallback = (id) => {…
    };

    const documentsList = documents.map((document) => {
        return <Grid className="breathe"><DocumentCard key={document._id} document={document} callback={deleteCallback} /></Grid>;
    });

    const folderTitle = folderData.folderTitle;
    const imgPath = folderData.imgPath;

    return (
        <>
            <img className='img-cover view-all-documents-header-image ' src={`${STATIC_FILES_URL}${imgPath}`} alt="Folder"></img>

            <Grid container justifyContent="center" className='breathe-m'>
                <ErrorBoundry>
                    <Grid justifyContent="space-around" container md={11.5}>
                        <Grid md={12}>
```

Figure 172 View Single Folder

Figure 172 shows the viewSingleFolder.js file, and the other headings show the CRUD components that the application uses to create a new folder.

The implementation of the folder structure allowed users to store documents in more organized manner, and an example of the finished folder structure can be seen in figure 173.

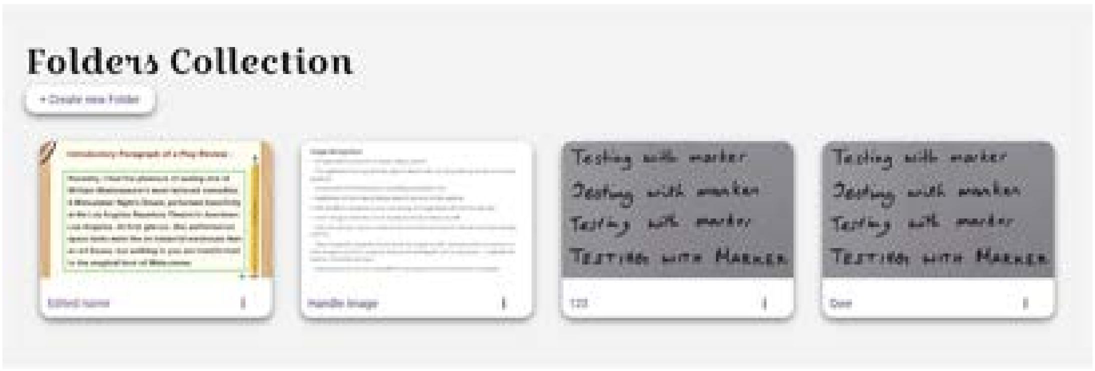


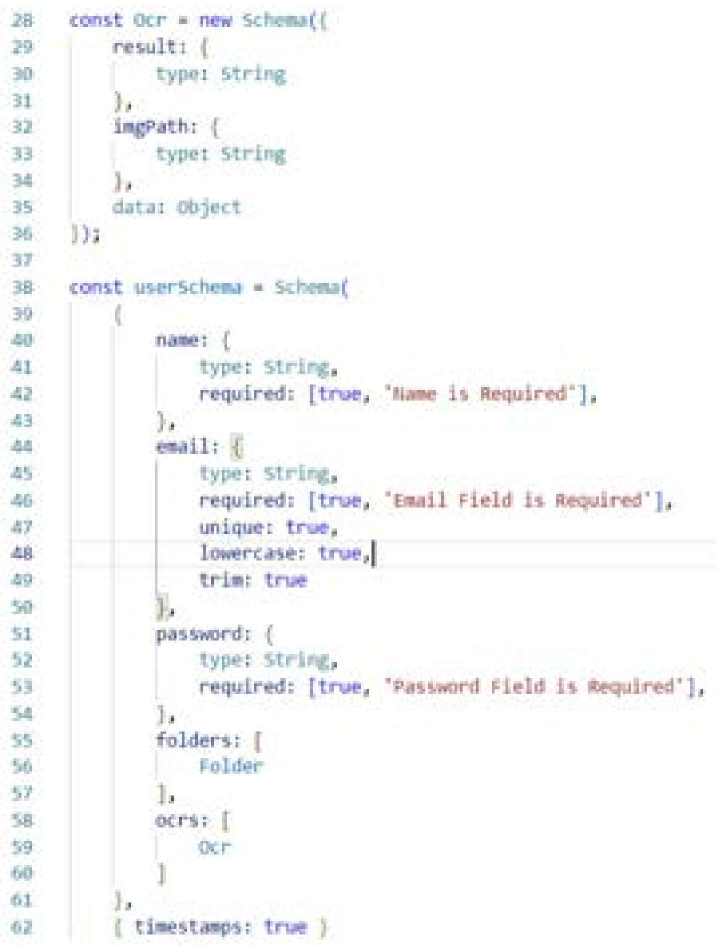
Figure 173 Folder Collection

The documents are now accessible in a folder system.

5.11.1.2 Item 2 – Adding Backend for OCR Reader, including CRUD, saving images and results, and displaying on a separate page.

The goal of this sprint was to develop a backend for the OCR Reader, as it cannot store the data that it returns. The steps to build this was again similar in how it was implemented in previous sprints and will only be discussed briefly.

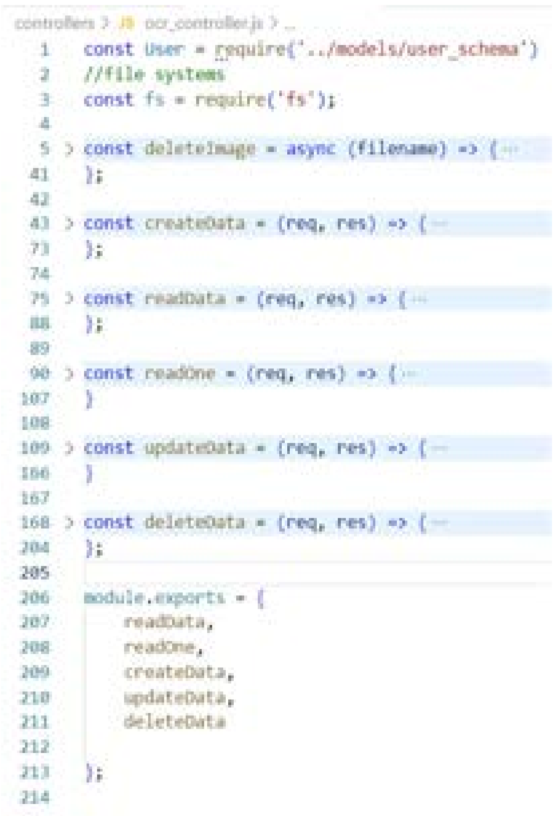
The OCR schema was first created, and added to the user schema, seen in figure 174:



```
const Ocr = new Schema({
    result: {
        type: String
    },
    imgPath: {
        type: String
    },
    data: Object
});

const userSchema = Schema(
    {
        name: {
            type: String,
            required: [true, 'Name is Required'],
        },
        email: {
            type: String,
            required: [true, 'Email Field is Required'],
            unique: true,
            lowercase: true,
            trim: true
        },
        password: {
            type: String,
            required: [true, 'Password Field is Required'],
        },
        folders: [
            Folder
        ],
        ocrs: [
            Ocr
        ]
    },
    { timestamps: true }
```

Figure 174 OCR Schema

Similar CRUD functionalities were the created in the ocr_controller.js file:



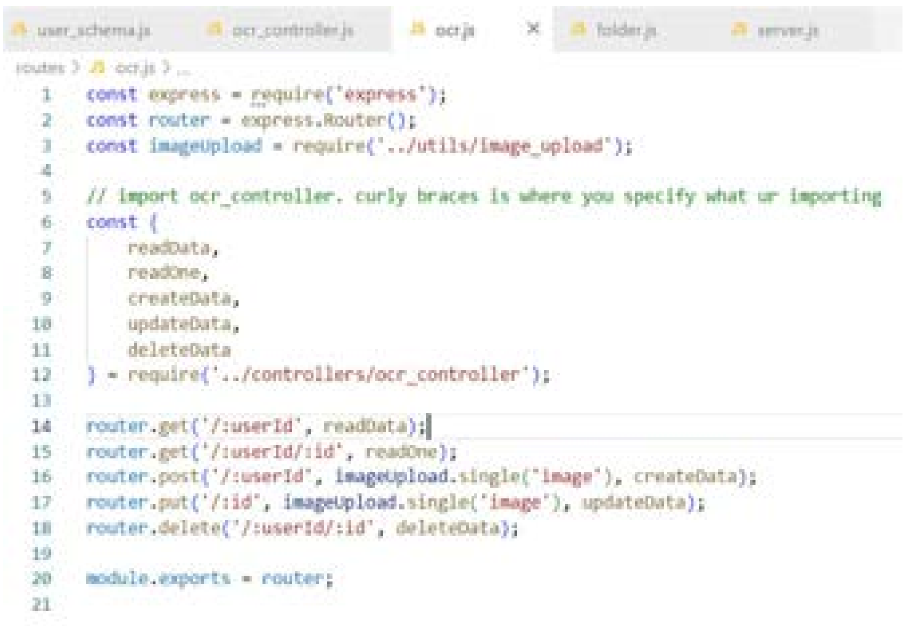
controllers > ocr_controller.js > ...

```
const User = require('../models/user_schema')
//file systems
const fs = require('fs');

const deleteImage = async (filename) => {⋯
};

const createData = (req, res) => {⋯
};

const readData = (req, res) => {⋯
};

const readOne = (req, res) => {⋯
}

const updateData = (req, res) => {⋯
}

const deleteData = (req, res) => {⋯
};

module.exports = {
    readData,
    readOne,
    createData,
    updateData,
    deleteData

};
```

Figure 175 Functions

Figure 175 only shows the function names, as they have already been discussed in greater detail. There were no major changes made, except editing it to fit the OCR files, rather than the previous files.

Once this was completed, the routes were then set, seen in figure 176:



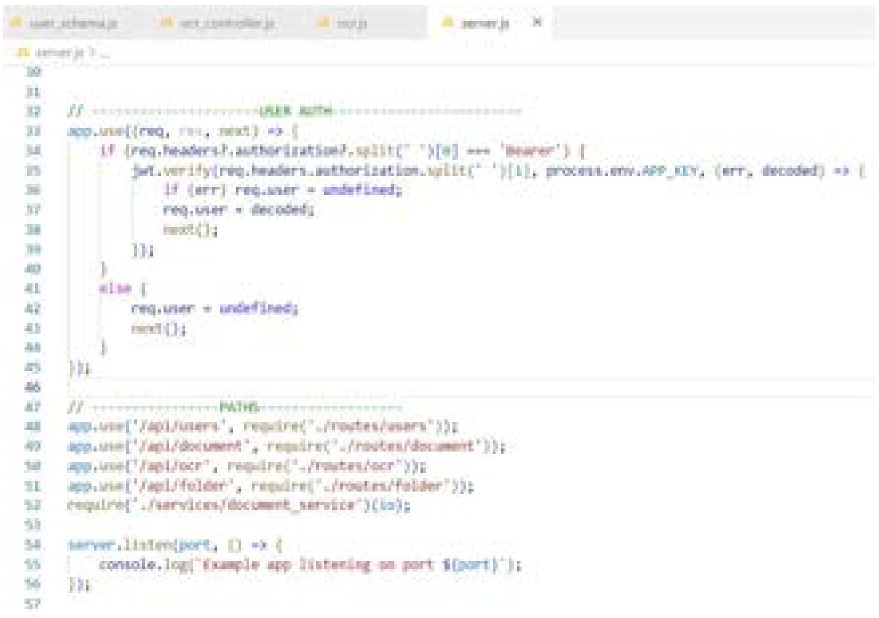
user_schema.js | ocr_controller.js | ocr.js × | folder.js | server.js

routes > ocr.js > ...

```
const express = require('express');
const router = express.Router();
const imageUpload = require('../utils/image_upload');

// import ocr_controller. curly braces is where you specify what ur importing
const {
    readData,
    readOne,
    createData,
    updateData,
    deleteData
} = require('../controllers/ocr_controller');

router.get('/:userId', readData);
router.get('/:userId/:id', readOne);
router.post('/:userId', imageUpload.single('image'), createData);
router.put('/:id', imageUpload.single('image'), updateData);
router.delete('/:userId/:id', deleteData);

module.exports = router;
```

Figure 176 Routes

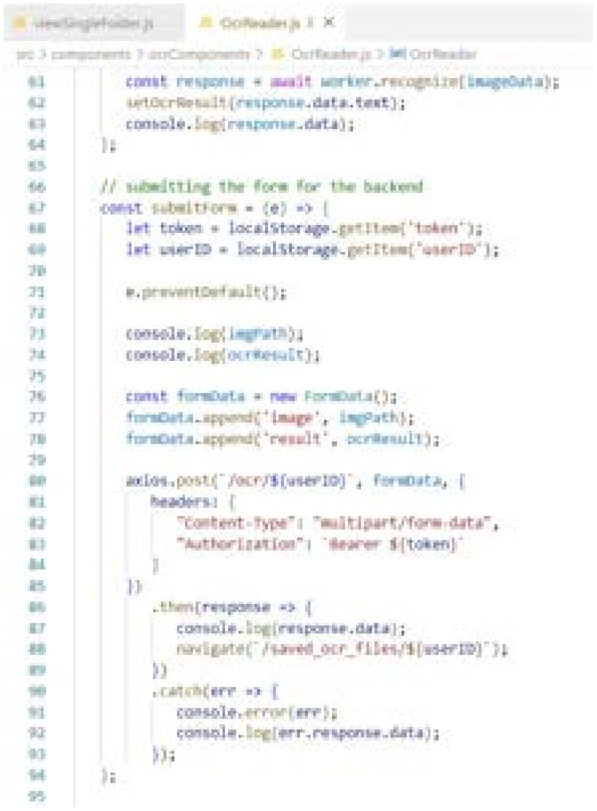
Finally, it was set in the server.js file, seen in the figure below:



```
// ----------------------USER AUTH-------------------------
app.use((req, res, next) => {
    if (req.headers?.authorization?.split(' ')[0] === 'Bearer') {
        jwt.verify(req.headers.authorization.split(' ')[1], process.env.APP_KEY, (err, decoded) => {
            if (err) req.user = undefined;
            req.user = decoded;
            next();
        });
    }
    else {
        req.user = undefined;
        next();
    }
});

// ----------------PATHS--------------------
app.use('/api/users', require('./routes/users'));
app.use('/api/document', require('./routes/document'));
app.use('/api/ocr', require('./routes/ocr'));
app.use('/api/folder', require('./routes/folder'));
require('./services/document_service')(io);

server.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
});
```

Figure 177 Server.js

As the OCR Reader is set in the frontend, the results from the OCR Reader are what is stored in the results section. It can be seen in figure 178:



```
        const response = await worker.recognize(imageData);
        setOcrResult(response.data.text);
        console.log(response.data);
    };

    // submitting the form for the backend
    const submitForm = (e) => {
        let token = localStorage.getItem('token');
        let userID = localStorage.getItem('userID');

        e.preventDefault();

        console.log(imgPath);
        console.log(ocrResult);

        const formData = new FormData();
        formData.append('image', imgPath);
        formData.append('result', ocrResult);

        axios.post(`/ocr/${userID}`, formData, {
            headers: {
                "Content-Type": "multipart/form-data",
                "Authorization": `Bearer ${token}`
            }
        })
            .then(response => {
                console.log(response.data);
                navigate(`/saved_ocr_files/${userID}`);
            })
            .catch(err => {
                console.error(err);
                console.log(err.response.data);
            });
    };
```

Figure 178 OCR Reader Frontend

These are received after the OCR Reader has finished extracting the results, and the user will need to manually save this result by using the submitForm function.

Once the submit button is hit, the form is then stored in the user's profile, which is seen in figure 179.

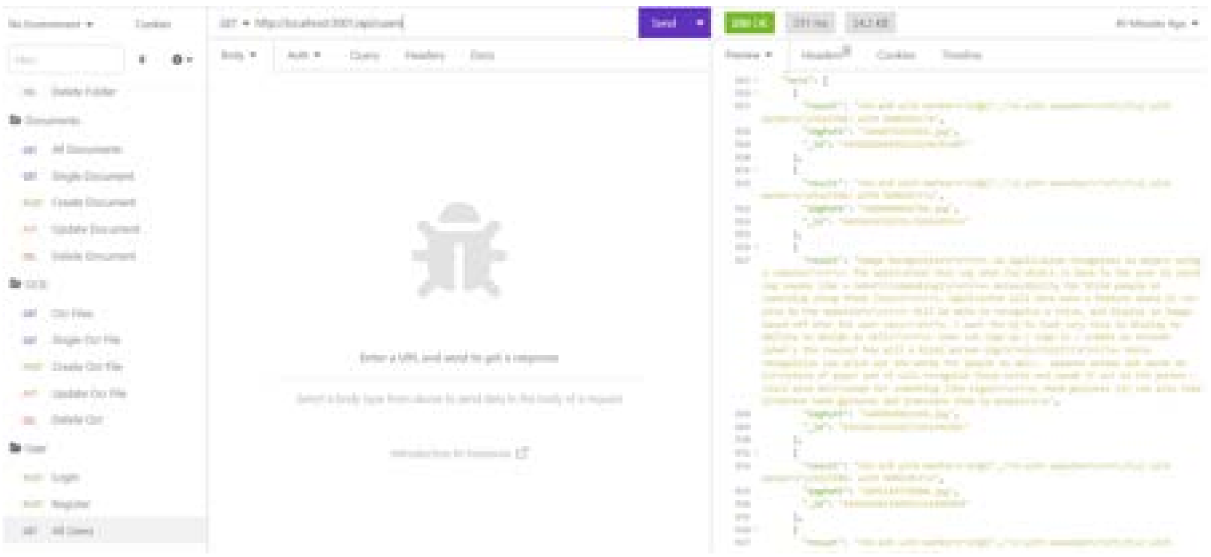
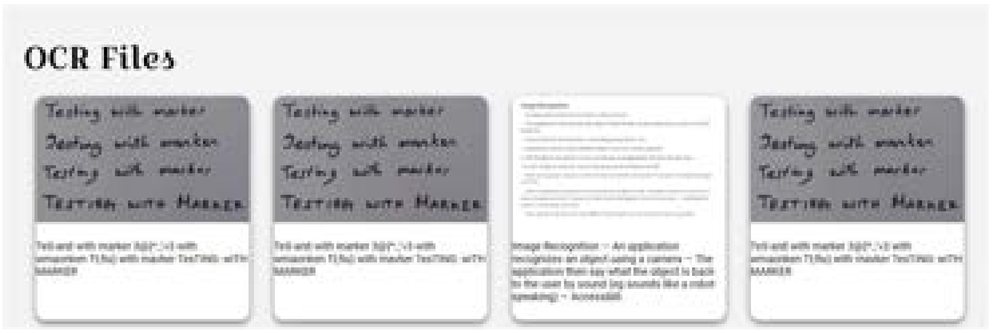


Figure 179 Insomnia Storage

This data can also be accessed and selected in the frontend, seen in figure 180.



Test and with marker 3@!*,\3 with amaorken T(fu) with marker TesTING wiTH MARKER

Test and with marker 3@!*,\3 with amaorken T(fu) with marker TesTING wiTH MARKER

Image Recognition ~ An application recognizes an object using a camera ~ The application then say what the object is back to the user by sound (eg sounds like a robot speaking) ~ Accessibili

Test and with marker 3@!*,\3 with amaorken T(fu) with marker TesTING wiTH MARKER

Figure 180 Frontend Storage

5.11.1.3 Item 3 – Implementing UI Elements for the application.

The goal of this item was to implement the UI elements that the application will be using. The project will be using some components from MUI, such as the navbar, and grid system, and will also be using fonts from Google Fonts, and icons from Font Awesome.

As it is mostly UI elements, the code will only be briefly discussed as it is mostly going to focus on the styling and look of the application.

The homepage was styled first, which can be seen in figure 181.

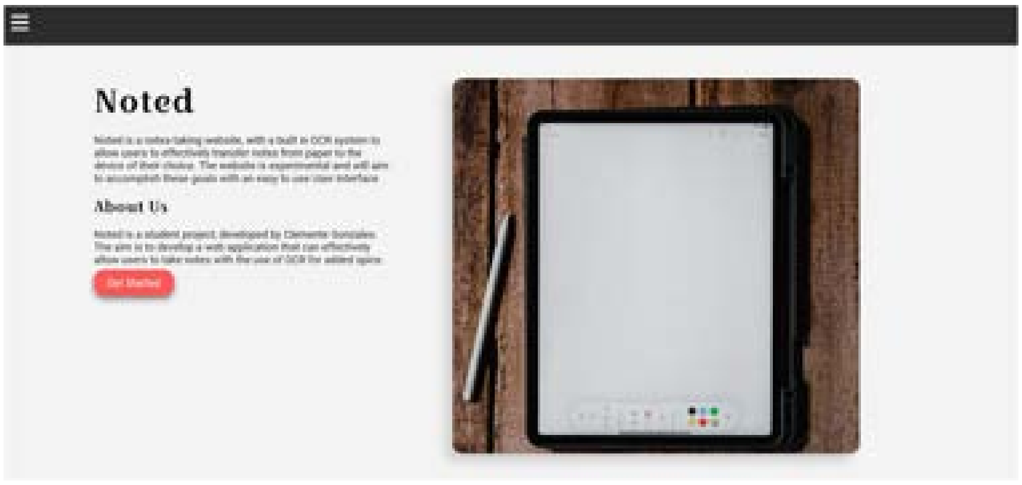
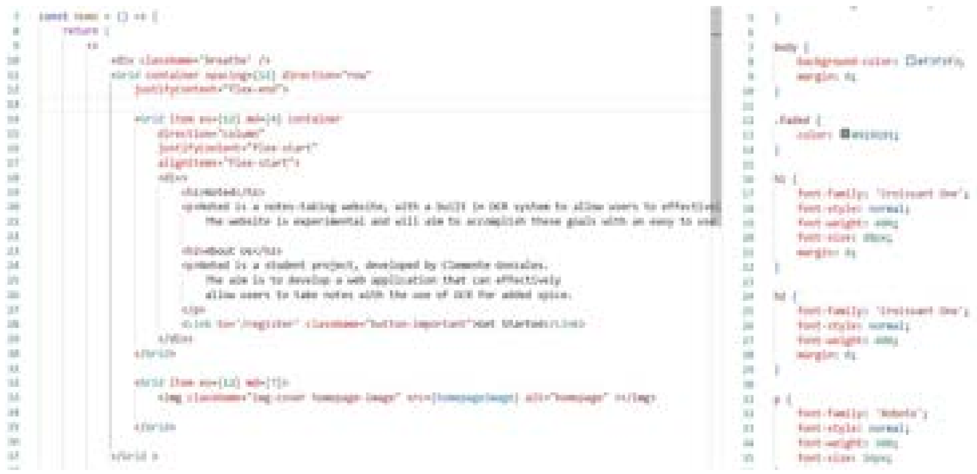


Figure 181 Homepage Styling

The use of the MUI Grid system was used throughout the project and provided great help with the positioning of the application.

The code and some parts of the CSS can be seen in figure 182, although it will not be discussed in full detail, only highlighting the important parts:



```
const Home = () => {
    return (
        <>
            <div className='breathe' />
            <Grid container spacing={[4]} direction="row"
                justifyContent="flex-end">

                <Grid item xs={12} md={4} container
                    direction="column"
                    justifyContent="flex-start"
                    alignItems="flex-start">
                    <div>
                        <h1>Noted</h1>
                        <p>Noted is a notes-taking website, with a built in OCR system to allow users to effectivel
                            The website is experimental and will aim to accomplish these goals with an easy to use

                        <h2>About Us</h2>
                        <p>Noted is a student project, developed by Clemente Gonzales.
                            The aim is to develop a web application that can effectively
                            allow users to take notes with the use of OCR for added spice.
                        </p>
                        <Link to='/register' className="button-important">Get Started</Link>
                    </div>
                </Grid>

                <Grid item xs={12} md={7}>
                    <img className="img-cover homepage-image" src={homepageImage} alt="homepage" ></img>

                </Grid>

            </Grid >
```

```
}

body {
    background-color: #f4f4f4;
    margin: 0;
}

.faded {
    color: #a9a9a9;
}

h1 {
    font-family: '[illegible]';
    font-style: normal;
    font-weight: 400;
    font-size: 48px;
    margin: 0;
}

h2 {
    font-family: '[illegible]';
    font-style: normal;
    font-weight: 400;
    margin: 0;
}

p {
    font-family: 'Roboto';
    font-style: normal;
    font-weight: 400;
    font-size: 16px;
```

Figure 182 Code for Homepage Styling

Figure 182 shows an example of the grids being used to align the items in the correct place. The CSS code to the right also shows the styling of the application.

The navbar was then altered. The navbar uses the MUI Navbar for the sliding function and can be seen in figure 183:

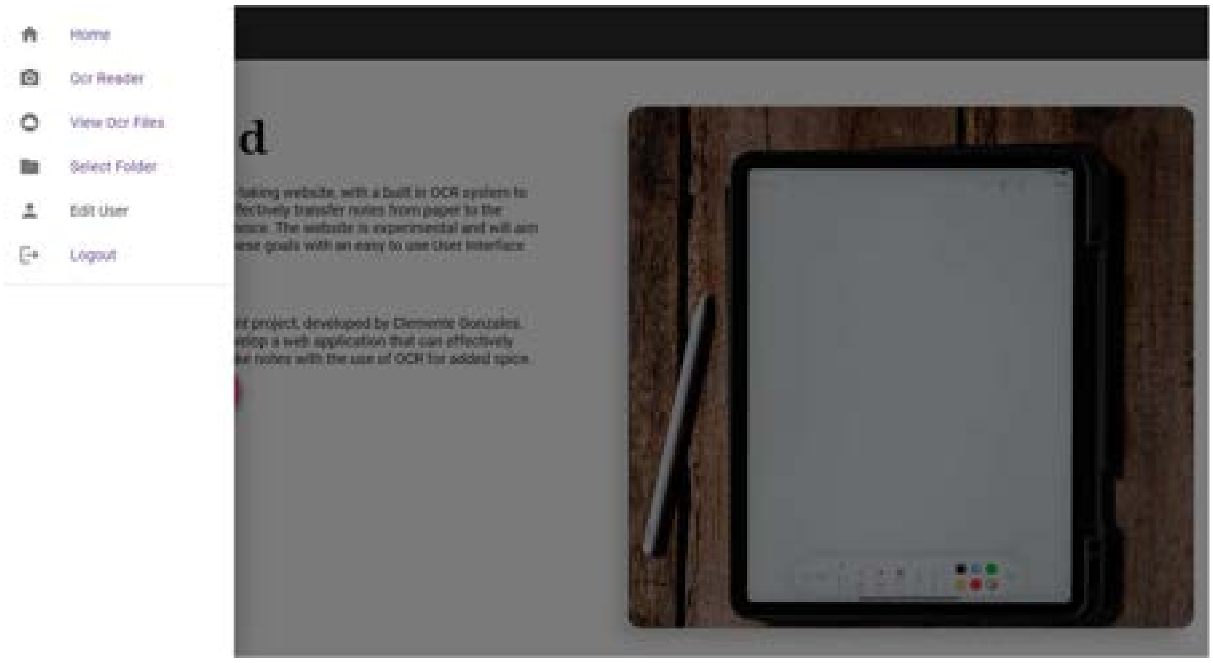
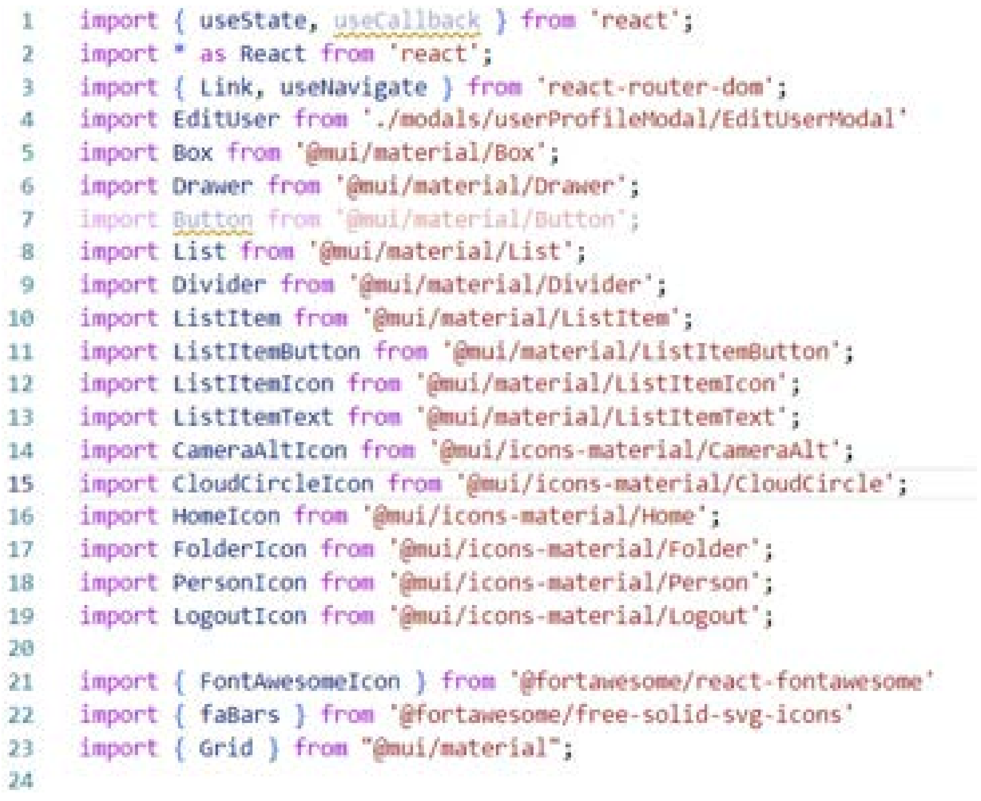


Figure 183 Navbar Styling

The figure above shows the navbar and is slightly different to how the initial design was made, however it contains the links that redirect the user to the desired pages. The navbar also contains icons from Font Awesome and was imported to suit the side-nav.

The code can be seen in the figure below:

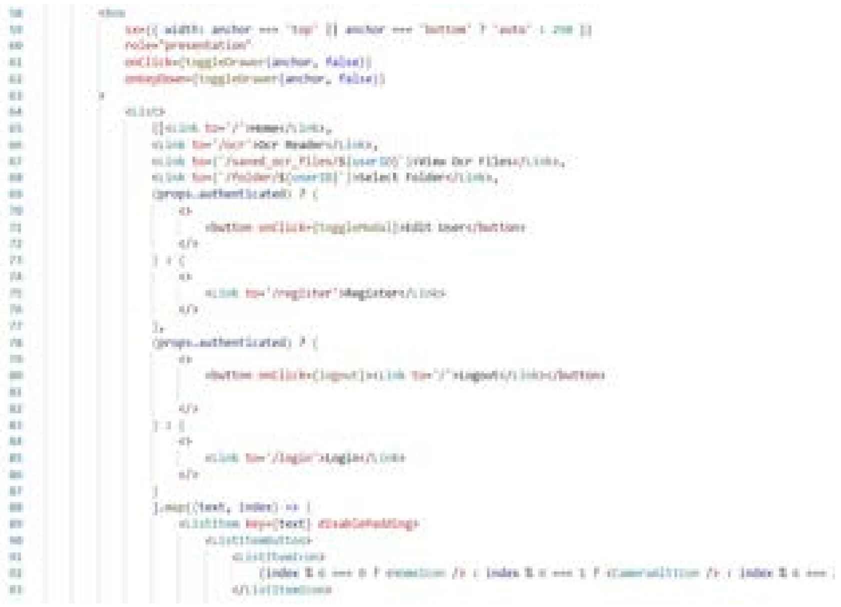


```
import { useState, useCallback } from 'react';
import * as React from 'react';
import { Link, useNavigate } from 'react-router-dom';
import EditUser from './modals/userProfileModal/EditUserModal'
import Box from '@mui/material/Box';
import Drawer from '@mui/material/Drawer';
import Button from '@mui/material/Button';
import List from '@mui/material/List';
import Divider from '@mui/material/Divider';
import ListItem from '@mui/material/ListItem';
import ListItemButton from '@mui/material/ListItemButton';
import ListItemIcon from '@mui/material/ListItemIcon';
import ListItemText from '@mui/material/ListItemText';
import CameraAltIcon from '@mui/icons-material/CameraAlt';
import CloudCircleIcon from '@mui/icons-material/CloudCircle';
import HomeIcon from '@mui/icons-material/Home';
import FolderIcon from '@mui/icons-material/Folder';
import PersonIcon from '@mui/icons-material/Person';
import LogoutIcon from '@mui/icons-material/Logout';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faBars } from '@fortawesome/free-solid-svg-icons'
import { Grid } from "@mui/material";
```

Figure 184 Navbar Code

First the imports can be seen, including the icons from fontawesome

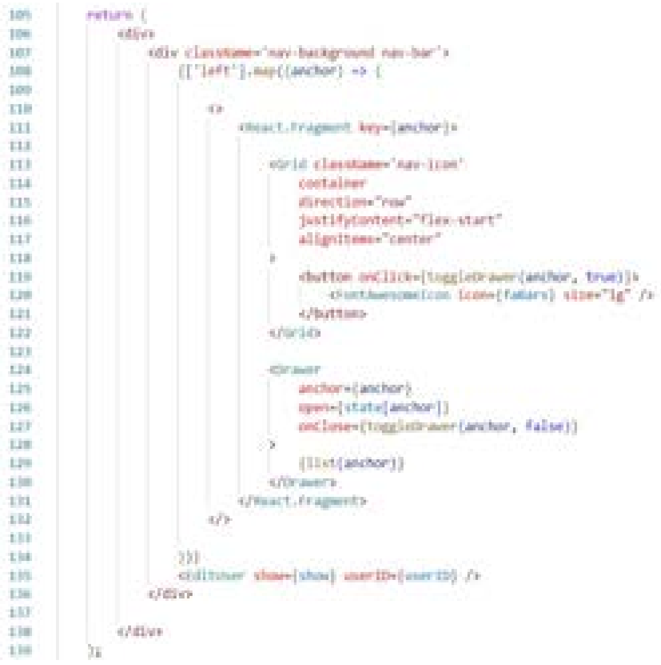
It uses the MUI Navbar to redirect users as seen in figure 185:



```
<Box
    sx={{ width: anchor === 'top' || anchor === 'bottom' ? 'auto' : 250 }}
    role="presentation"
    onClick={toggleDrawer(anchor, false)}
    onKeyDown={toggleDrawer(anchor, false)}
>
    <List>
        {[<Link to='/'>Home</Link>,
        <Link to='/ocr'>Ocr Reader</Link>,
        <Link to={`/saved_ocr_files/${userID}`}>View Ocr Files</Link>,
        <Link to={`/folder/${userID}`}>Select Folder</Link>,
        (props.authenticated) ? (
            <>
                <button onClick={toggleModal}>Edit Users</button>
            </>
        ) : (
            <>
                <Link to='/register'>Register</Link>
            </>
        ),
        (props.authenticated) ? (
            <>
                <button onClick={logout}><Link to='/'>Logout</Link></button>

            </>
        ) : (
            <>
                <Link to='/login'>Login</Link>
            </>
        )
        ].map((text, index) => (
            <ListItem key={text} disablePadding>
                <ListItemButton>
                    <ListItemIcon>
                        {index % 4 === 0 ? <HomeIcon /> : index % 4 === 1 ? <CameraAltIcon /> : index % 4 ===
                    </ListItemIcon>
```

Figure 185 Navbar

Finally, it returns the mapped components seen in figure 186:



```
return (
    <div>
        <div className='nav-background nav-bar'>
            {['left'].map((anchor) => (

                <>
                    <React.Fragment key={anchor}>

                        <Grid className='nav-icon'
                            container
                            direction="row"
                            justifyContent="flex-start"
                            alignItems="center"
                        >
                            <button onClick={toggleDrawer(anchor, true)}>
                                <FontAwesomeIcon icon={faBars} size="lg" />
                            </button>
                        </Grid>

                        <Drawer
                            anchor={anchor}
                            open={state[anchor]}
                            onClose={toggleDrawer(anchor, false)}
                        >
                            {list(anchor)}
                        </Drawer>
                    </React.Fragment>
                </>

            ))}
            <EditUser show={show} userID={userID} />
        </div>

    </div>
);
```

Figure 186 Navbar

The OCR Reader is then styled, and can be seen in figure 187:

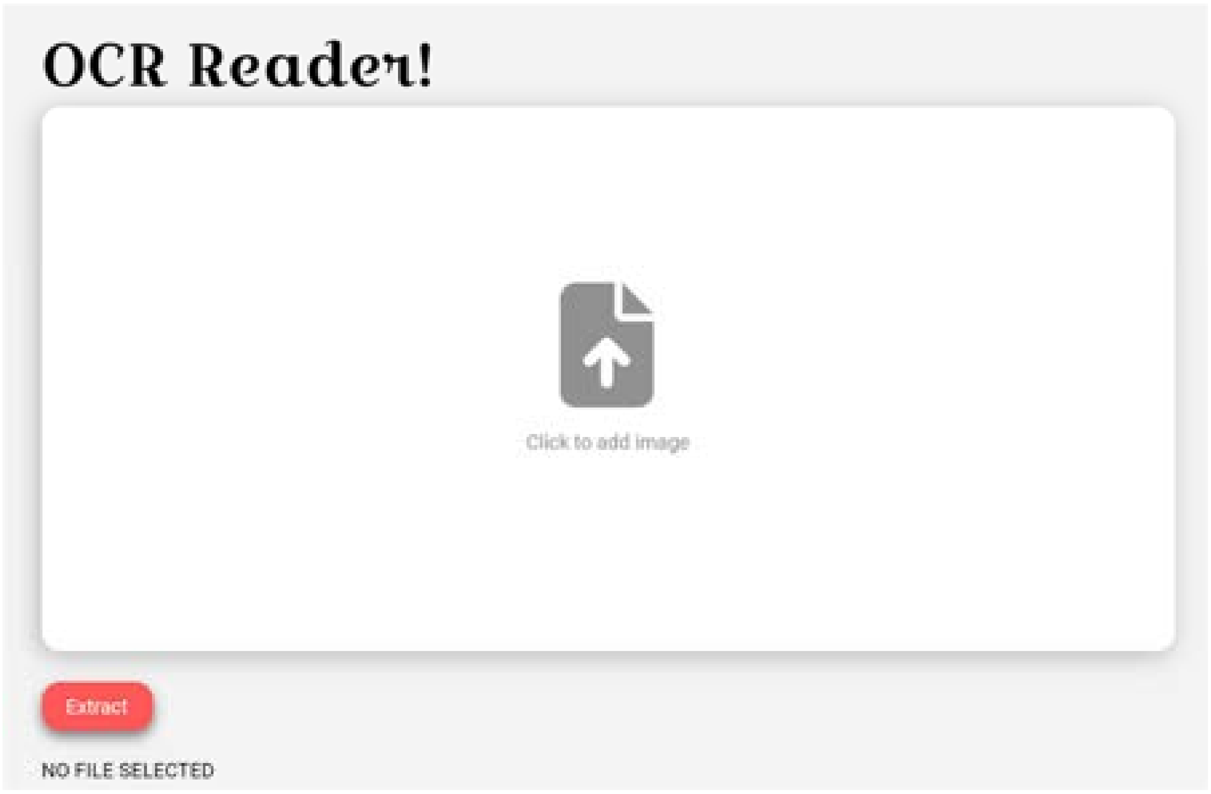
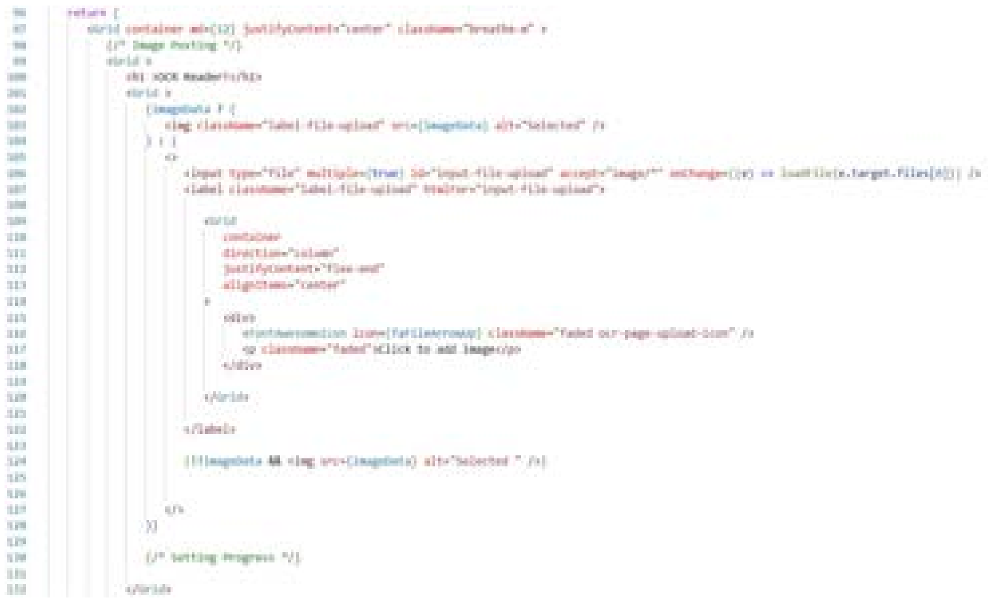


Figure 187 OCR Reader Styling

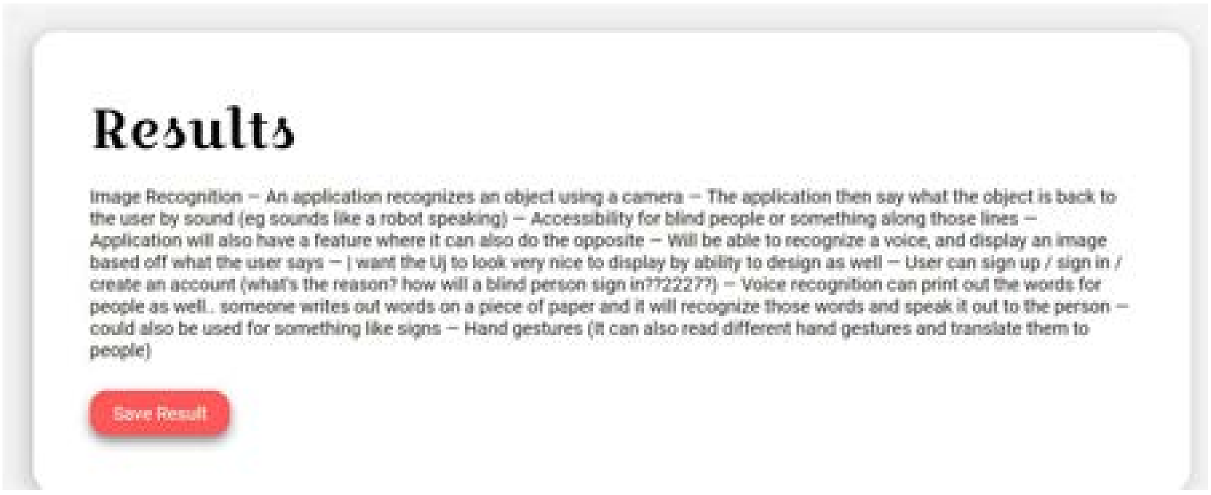
The OCR Reader again uses the grid system and the icons from Font Awesome. No major changes in code besides the CSS files seen in figure 188.



```
return (
    <Grid container mb={12} justifyContent="center" className="breathe-a" >
        {/* Image Posting */}
        <Grid >
            <h1 >OCR Reader!</h1>
            <Grid >
                {imageData ? (
                    <img className="label-file-upload" src={imageData} alt="Selected" />
                ) : (
                    <>
                        <input type="file" multiple={true} id="input-file-upload" accept="image/*" onChange={(e) => loadFile(e.target.files[0])} />
                        <label className="label-file-upload" htmlFor="input-file-upload">

                            <Grid
                                container
                                direction="column"
                                justifyContent="flex-end"
                                alignItems="center"
                            >
                                <div>
                                    <FontAwesomeIcon icon={faFileArrowUp} className="faded ocr-page-upload-icon" />
                                    <p className="faded">Click to add image</p>
                                </div>

                            </Grid>

                        </label>

                        {!!imageData && <img src={imageData} alt="Selected " />}


                    </>
                )}

                {/* Setting Progress */}

            </Grid>
```

Figure 188 OCR Reader Code

The results also show when it completes the extraction of the text seen in the figure below:

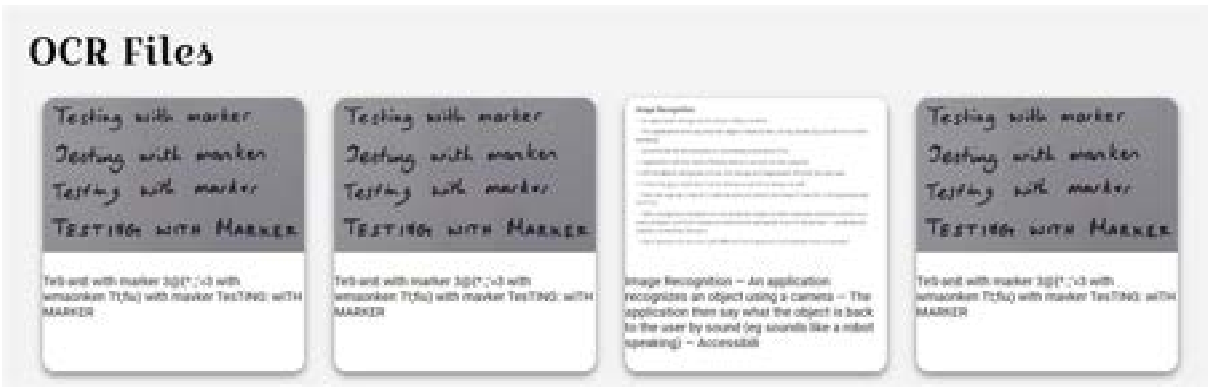


Results

Image Recognition – An application recognizes an object using a camera – The application then say what the object is back to the user by sound (eg sounds like a robot speaking) – Accessibility for blind people or something along those lines – Application will also have a feature where it can also do the opposite – Will be able to recognize a voice, and display an image based off what the user says – | want the Uj to look very nice to display by ability to design as well – User can sign up / sign in / create an account (what's the reason? how will a blind person sign in??2227?) – Voice recognition can print out the words for people as well.. someone writes out words on a piece of paper and it will recognize those words and speak it out to the person – could also be used for something like signs – Hand gestures (It can also read different hand gestures and translate them to people)

Figure 189 OCR Reader Result

The View All OCR Files was then styled. This can be seen in figure 190:



OCR Files

Teb-and with marker 3@(*.;<3 with wmaonken Tt(fu) with maiker TesTING: wiTH MARKER

Teb-and with marker 3@(*.;<3 with wmaonken Tt(fu) with maiker TesTING: wiTH MARKER

Image Recognition – An application recognizes an object using a camera – The application then say what the object is back to the user by sound (eg sounds like a robot speaking) – Accessibili

Teb-and with marker 3@(*.;<3 with wmaonken Tt(fu) with maiker TesTING: wiTH MARKER

Figure 190 OCR Reader Files

The results were stored in a card and uses the grids to position them. When it is clicked, it displays a modal and can be seen in figure 191:

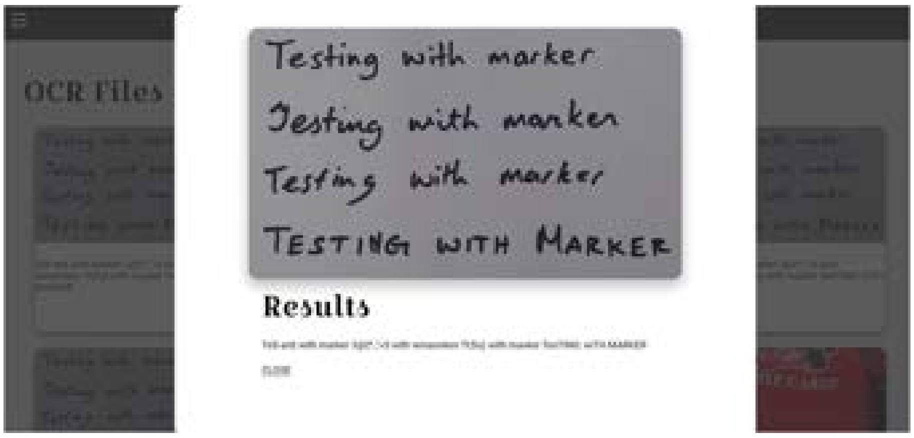


Figure 191 Viewing OCR File

The folders collection was then designed and can be seen in figure 192:

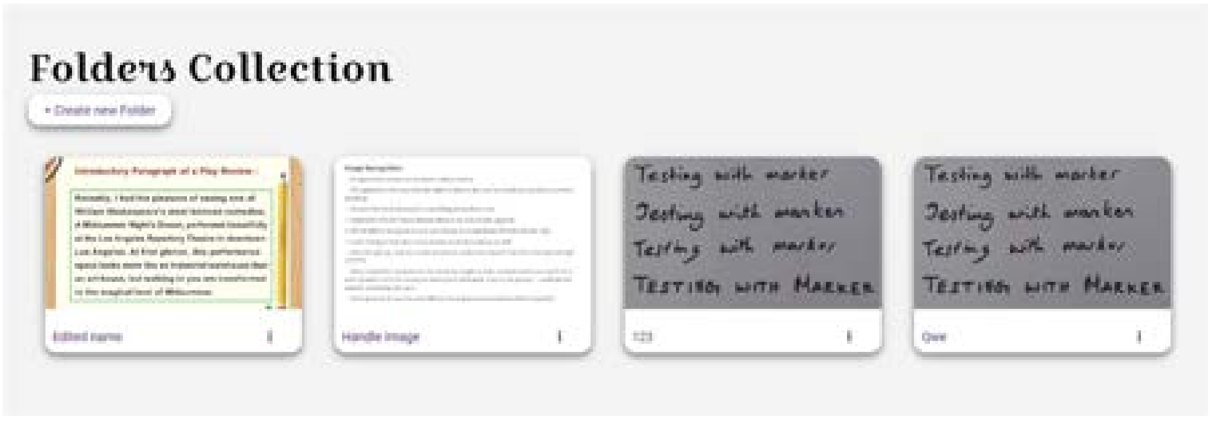


Figure 192 Folder Collection

It was like when the user views all the OCR Files, and the card contains a modal that allows users to edit or delete the folder, seen in figure 193:

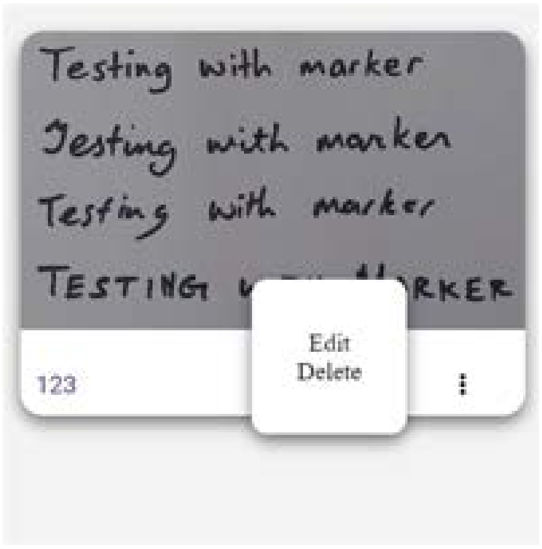


Figure 193 Edit and Delete

The styling for the create folder page was then completed, and the Create Document page is very similar to the design below:

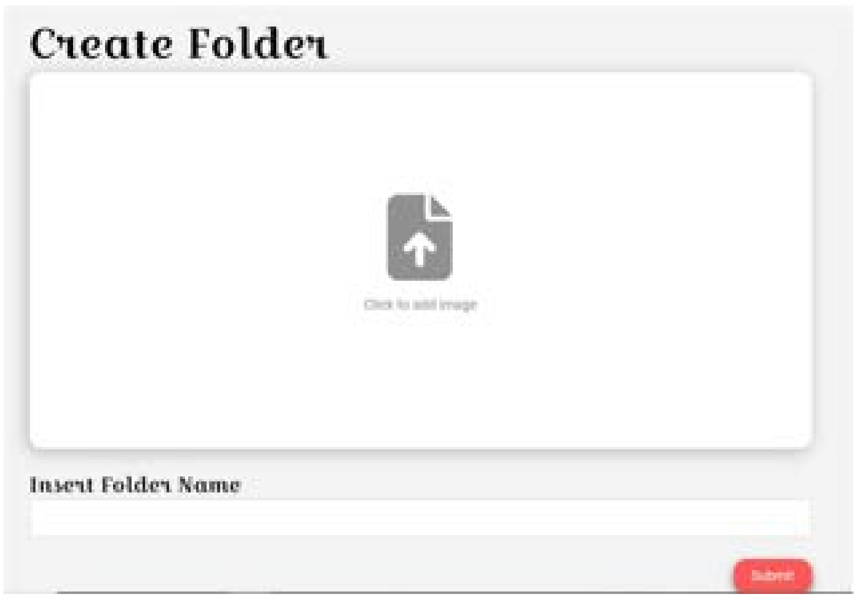


Figure 194 Create Folder

After that was completed, the view all documents page, or the single folder was then styled. It is similar to the view all folders; however, the single folder contains the image that the user placed in. It can be seen in figure 195:

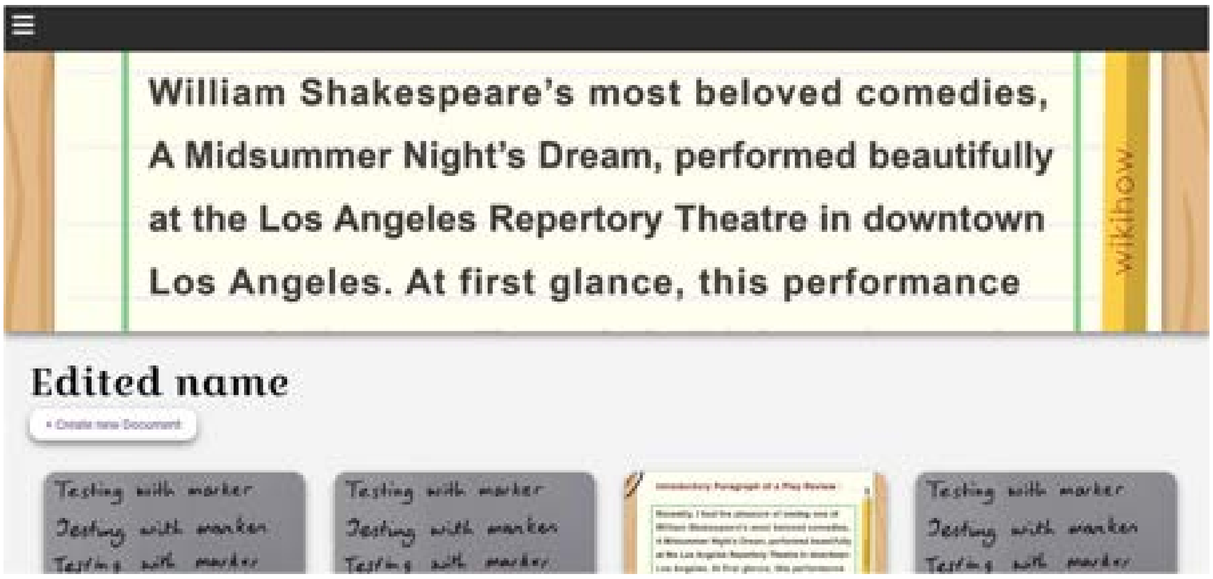

William Shakespeare's most beloved comedies,
A Midsummer Night's Dream, performed beautifully
at the Los Angeles Repertory Theatre in downtown
Los Angeles. At first glance, this performance
wikihow
Edited name
+ Create new Document
Testing with marker

Figure 195 View All Documents

Now when viewing a single document, the text editor now has an image, and the title of the document at the top of the page, seen in figure 196:

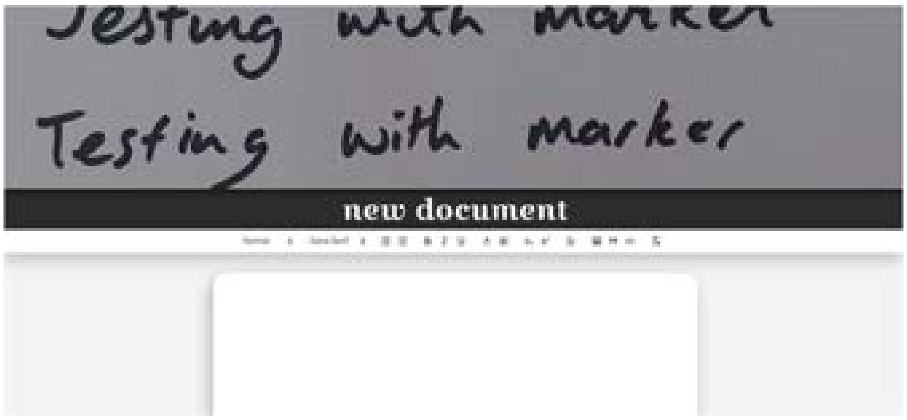


Figure 196 View Single Document

Regarding the user profile, an edit user modal was created to allow users to edit their details seen in figure 197:

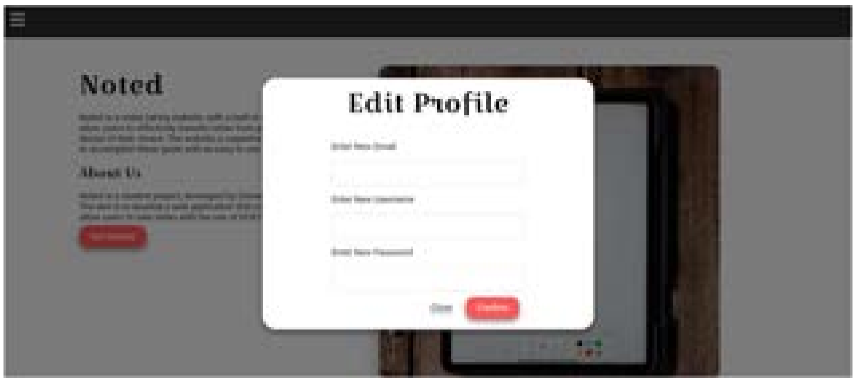


Figure 197 Edit Profile

Finally, the login and register pages were also completed and styled, and can be seen in figure 198 and figure 199:

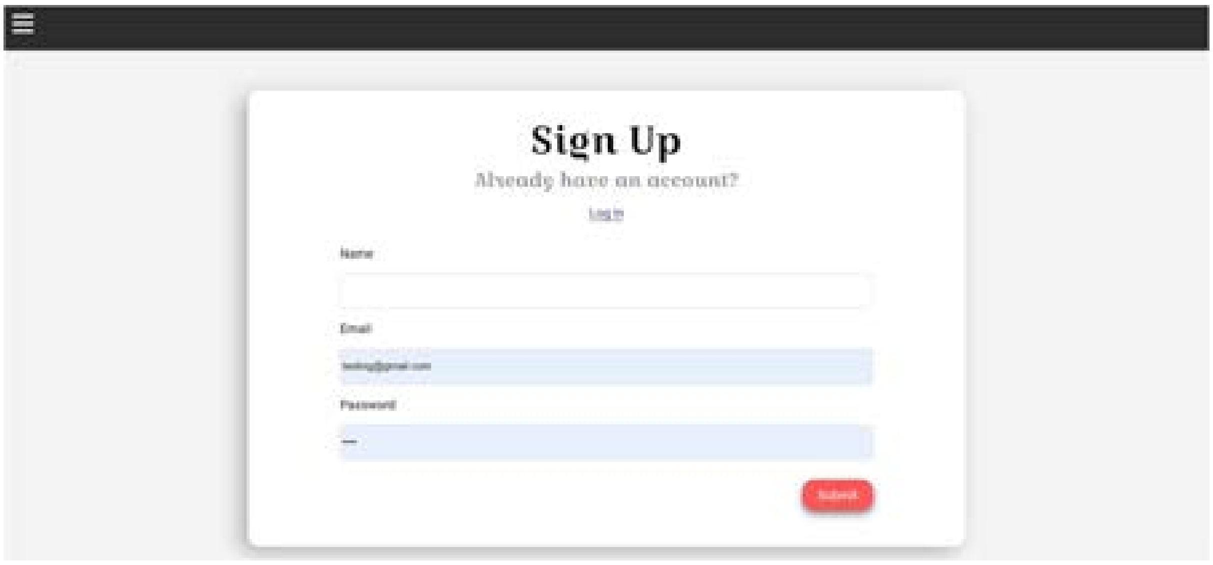


Figure 198 Register

Sign up page seen above.

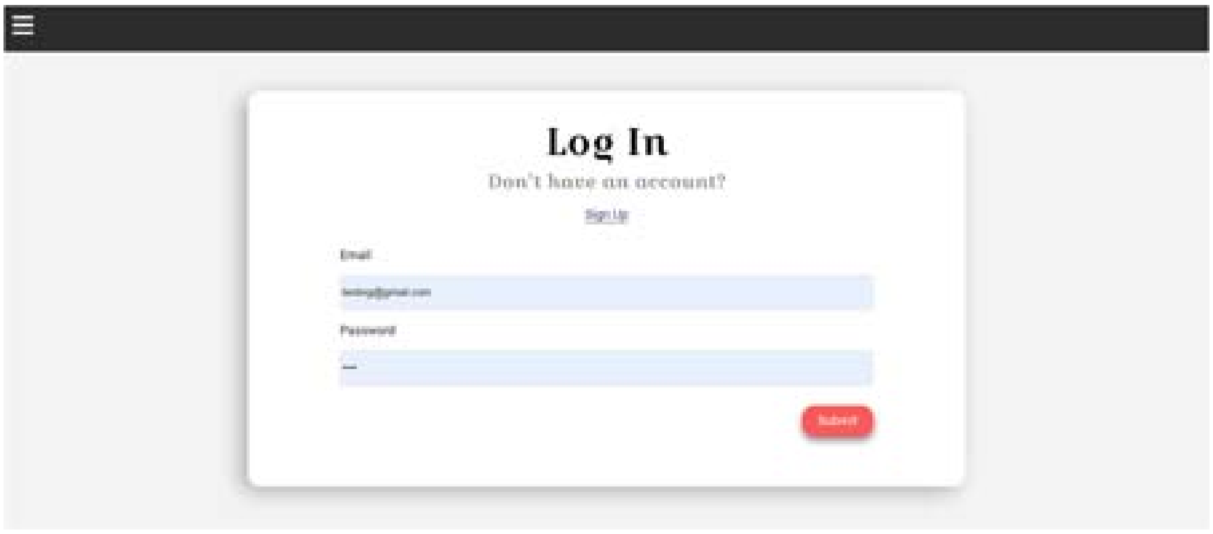


Figure 199 Login

Login page seen above.

5.11.1.4 Item 4 – User Testing

User testing was also performed during this stage of the application. Users were given a scenario and task to accomplish, as well as explore the application to see if they encounter any problems when using the application. This was to ensure that the bugs that can potentially be encountered are known and that potential fixes can be performed to allow for a higher-quality product.

User testing will be discussed in greater detail in the next section of this document.

5.11.1.5 Item 5 – OCR Accuracy Testing

OCR Accuracy was also tested. A resource was found to apply changes before using the OCR Reader, including gaussian blurs, setting the colours to monochromatic, etc. and was used to test how accurate the OCR Reader can be. This will also be discussed in a later section of the document and the code used will be credited there too.

5.11.1.6 Item 5 – Bug Fixes and Problems

Only a few minor problems were encountered during this sprint.

Problem 1:

One of which included the authentication failing, when the user is updating their details. This can be seen in figure 200:

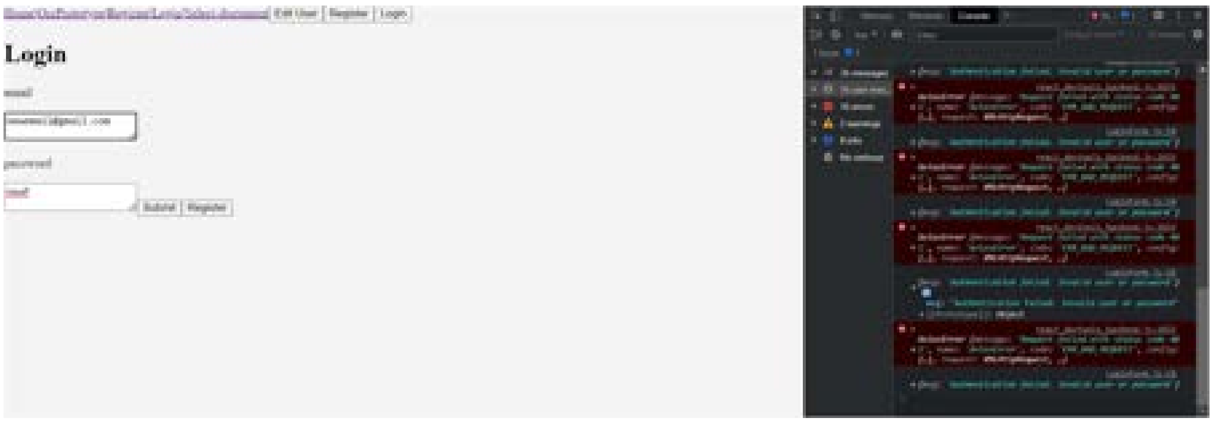
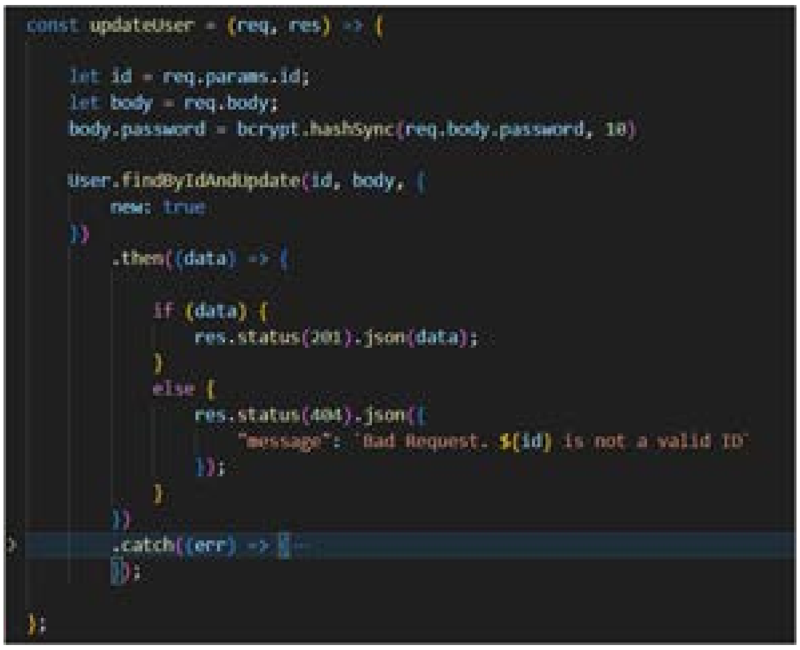


Figure 200 Problem 1

The solution of this problem was to add encryption to the “updateUser” method in the backend of the application. This is shown in the figure below.

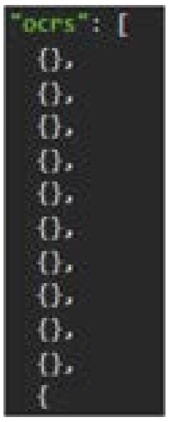


```
const updateUser = (req, res) => {

    let id = req.params.id;
    let body = req.body;
    body.password = bcrypt.hashSync(req.body.password, 10)

    User.findByIdAndUpdate(id, body, {
        new: true
    })
        .then((data) => {

            if (data) {
                res.status(201).json(data);
            }
            else {
                res.status(404).json({
                    "message": `Bad Request. ${id} is not a valid ID`
                });
            }
        })
        .catch((err) => {…
        });

};
```

Figure 201 Result

Problem 2:

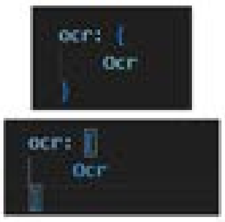
The second problem that was encountered was OCR in the backend was returning empty objects. This can be seen in figure 202.



```
"ocrs": [
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {},
  {
```

Figure 202 Problem 2

The solution to this problem was to add square brackets in the user_schema.js when including it in the user's schema. This is seen in figure 203.



```
ocr: {
    Ocr
}
```

```
ocr: [
    Ocr
]
```

Figure 203 Result

This changes it to an array, allowing the objects to be stored.

5.12 Sprint 7 – Thesis

5.12.1 Goal

The goal of this sprint is to finish the project, including the writeup for the thesis, and the final implementations, and bug fixes of the project. The items for this sprint can be seen:

- **Item 1:** Thesis Writeup
- **Item 2:** Deployment
- **Item 3:** Figures of the Final product

5.12.1.1 Item 1 – Thesis Writeup

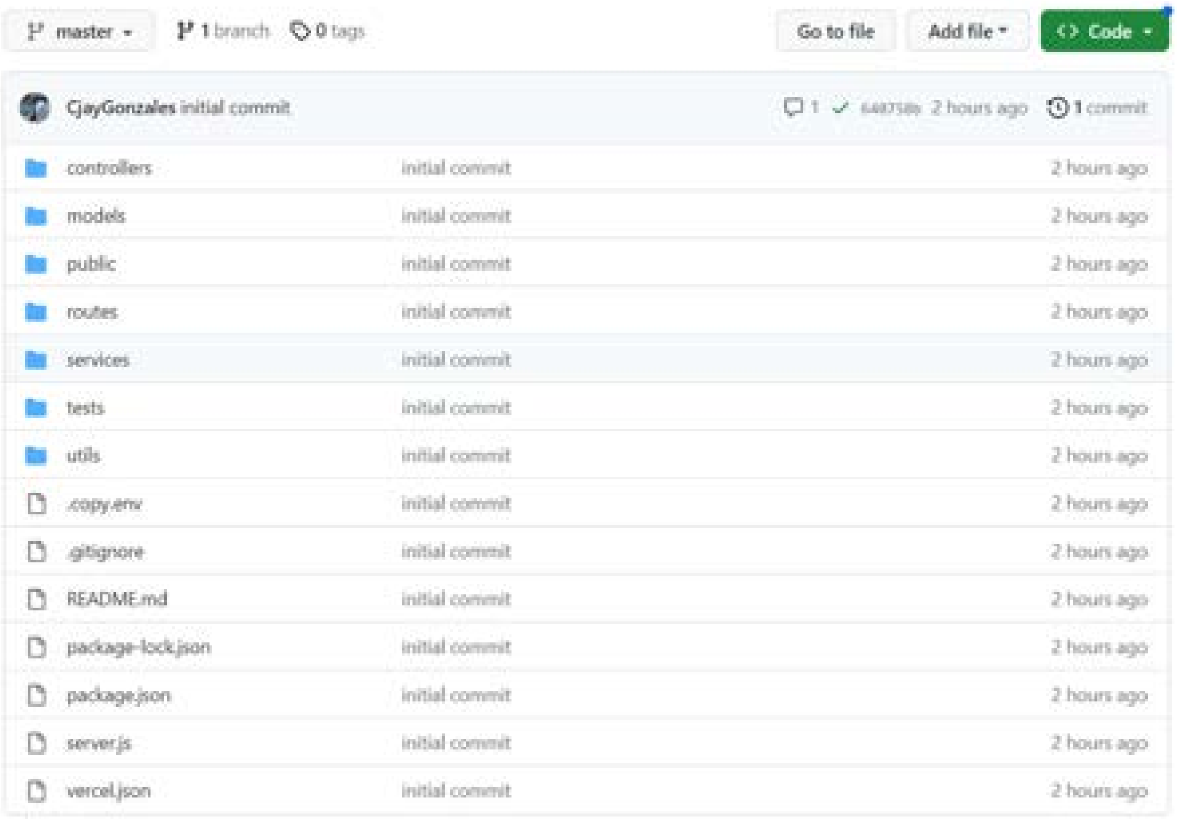
The goal of this item is to complete the writeup for the thesis. This was to complete the final parts of the document and to update parts of the document that needed to be updated and changed. No major parts of the code were changed and was mostly worked on the documentation of the project.

5.12.1.2 Item 2 – Deployment

The goal of this item is to deploy the project to be used by other people. The deployment of the project will be done at the very end, when the application is ready to be deployed. It will be deployed using Vercel and Firebase to allow users to access the application.

The backend is first deployed, with the use of Vercel.

To deploy the backend, a separate repository was created to allow Vercel to specifically use the code for the backend. This can be seen the figure below:

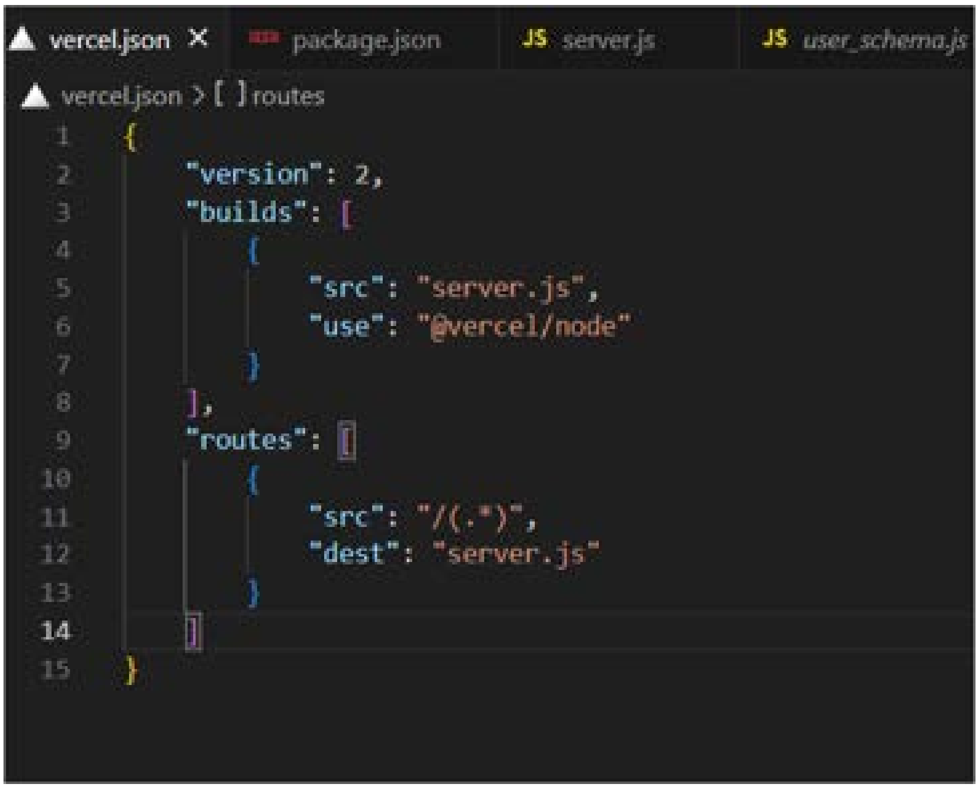


CjayGonzales initial commit 1 [illegible] 2 hours ago 1 commit

controllers	initial commit	2 hours ago
models	initial commit	2 hours ago
public	initial commit	2 hours ago
routes	initial commit	2 hours ago
services	initial commit	2 hours ago
tests	initial commit	2 hours ago
utils	initial commit	2 hours ago
.copy.env	initial commit	2 hours ago
.gitignore	initial commit	2 hours ago
README.md	initial commit	2 hours ago
package-lock.json	initial commit	2 hours ago
package.json	initial commit	2 hours ago
server.js	initial commit	2 hours ago
vercel.json	initial commit	2 hours ago

Figure 204 Second Backend GitHub

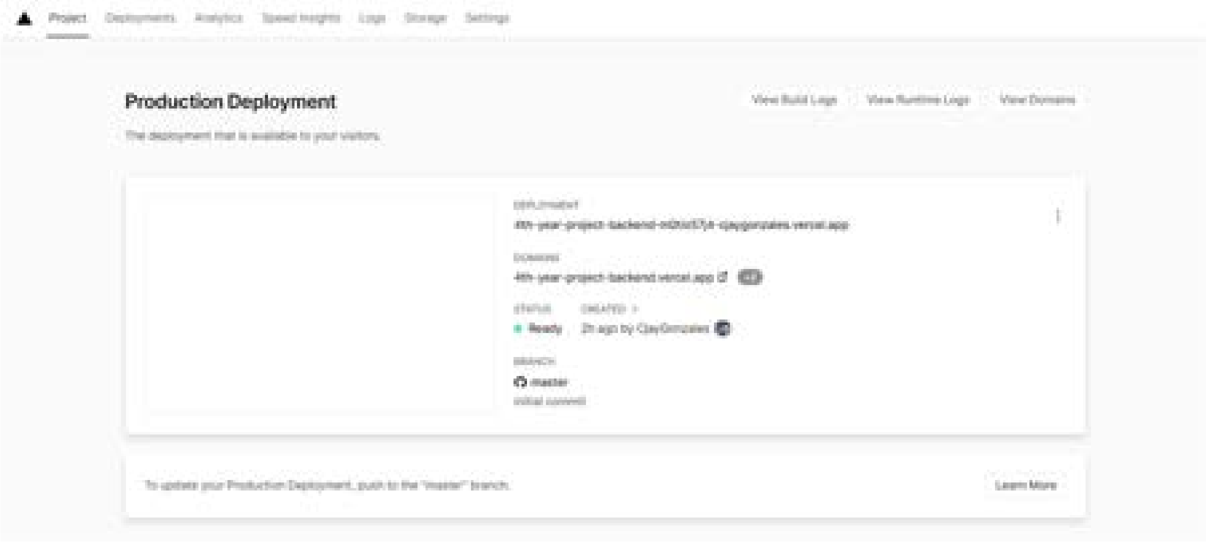
A Vercel.json file was then created seen in the figure below:



```
{
    "version": 2,
    "builds": [
        {
            "src": "server.js",
            "use": "@vercel/node"
        }
    ],
    "routes": [
        {
            "src": "/(.*)",
            "dest": "server.js"
        }
    ]
}
```

Figure 205 Vercel.json file

Once these were setup, the project was then configured, and deployed using the Vercel website. The .env files were set up during configuration. This can be seen in the figure below:



Production Deployment

The deployment that is available to your visitors.

DEPLOYMENT
4th-year-project-backend-[illegible]-claygonzales.vercel.app

DOMAINS
4th-year-project-backend.vercel.app

STATUS CREATED
Ready 2h ago by ClayGonzales

BRANCH
master
initial commit

To update your Production Deployment, push to the "master" branch.

Figure 206 Deployment of Backend

After deployment of the backend, the frontend was then deployed. This was done using Firebase to deploy the frontend of the application.

A new Firebase project was created seen in the figure below:

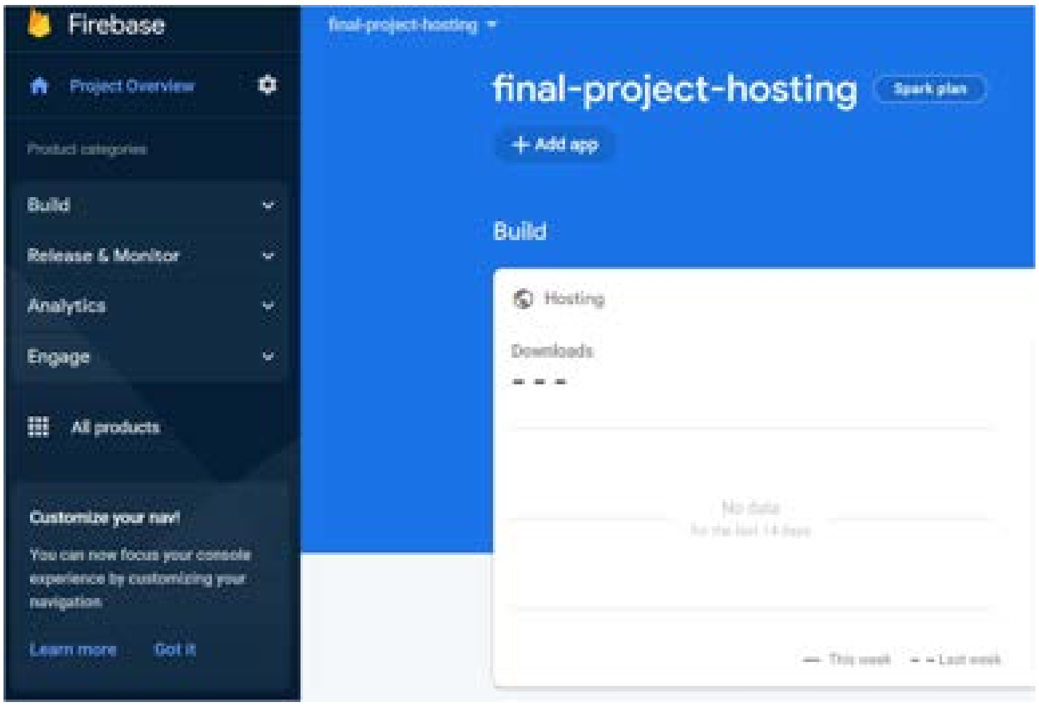


Figure 207 Firebase

A file called ".firebaserc" was created, with the ID of the hosted website. The file can be seen below:

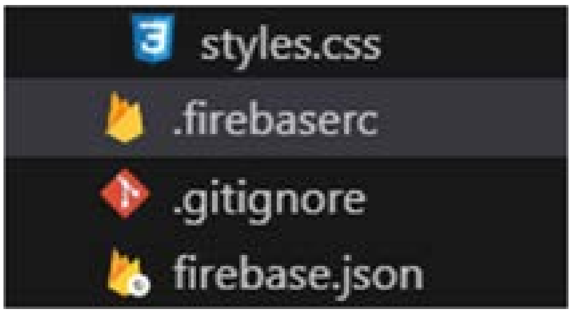
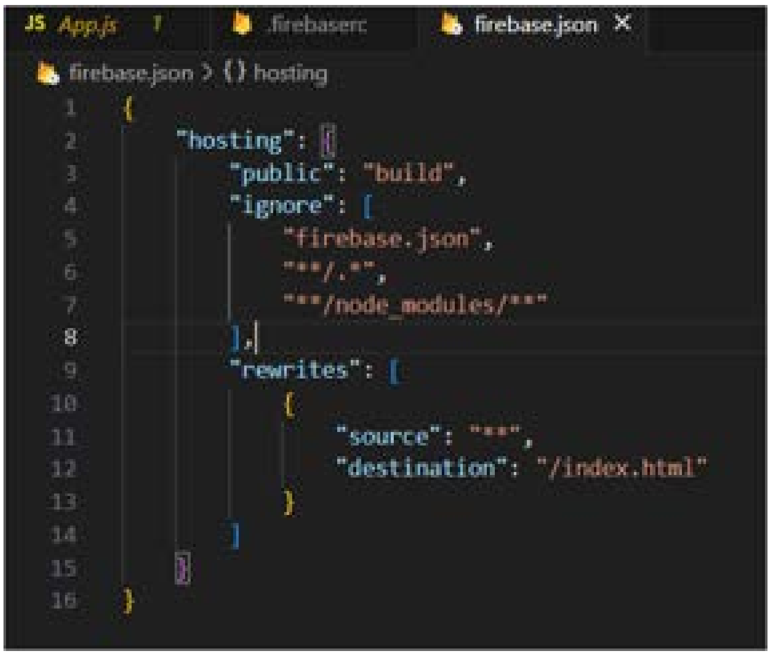


Figure 208 New Firebase Files

The firebase.json file was then created. This was for firebase find the files needed, that in which is the builds, which can be seen in the figure below:



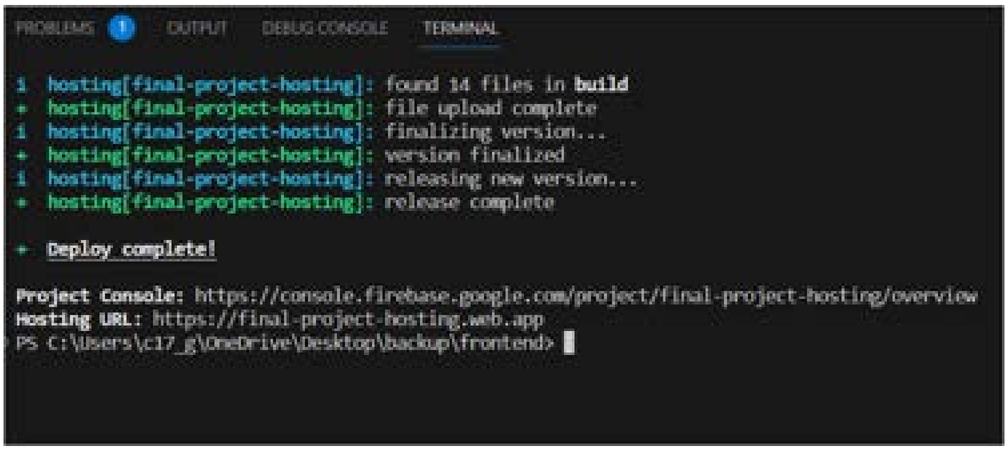
```json
{
    "hosting": {
        "public": "build",
        "ignore": [
            "firebase.json",
            "**/.*",
            "**/node_modules/**"
        ],
        "rewrites": [
            {
                "source": "**",
                "destination": "/index.html"
            }
        ]
    }
}
```

Figure 209 Firebase.json

Finally, a set of commands were used to deploy the application. These commands consisted of:

- npm install -g firebase-tools
- npm run build
- firebase login
- firebase deploy

Once these commands were completed, the result can be seen at the figure below:



```
i  hosting[final-project-hosting]: found 14 files in build
+  hosting[final-project-hosting]: file upload complete
i  hosting[final-project-hosting]: finalizing version...
+  hosting[final-project-hosting]: version finalized
i  hosting[final-project-hosting]: releasing new version...
+  hosting[final-project-hosting]: release complete

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/final-project-hosting/overview
Hosting URL: https://final-project-hosting.web.app
PS C:\Users\c17_g\OneDrive\Desktop\backup\frontend>
```

Figure 210 Deployment

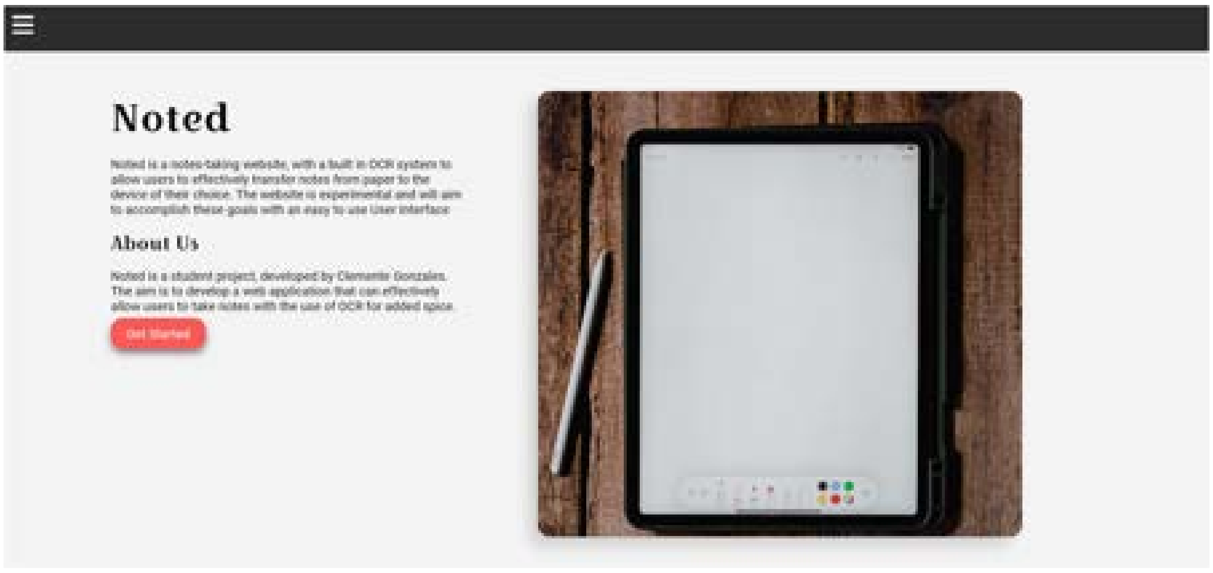
The link of the deployed project can be seen in Appendix D.

Within the deployed project, there are some problems that have occurred. Most notably in the documents side of the application, where some parts of the application are not working in the deployed project, although still works in the final local version. Due to time constraints, these problems in the deployed application were not fixed, however works properly in the local version.

5.12.1.3 Item 3 – Figures of the Final Product

This item is to allow reader to see images of the final product.

Homepage:



Noted

Noted is a notes-taking website, with a built in OCR system to allow users to effectively transfer notes from paper to the device of their choice. The website is experimental and will aim to accomplish these goals with an easy to use User Interface

About Us

Noted is a student project, developed by Clemente Gonzales. The aim is to develop a web application that can effectively allow users to take notes with the use of OCR for added space.

Figure 211 Homepage

Register:

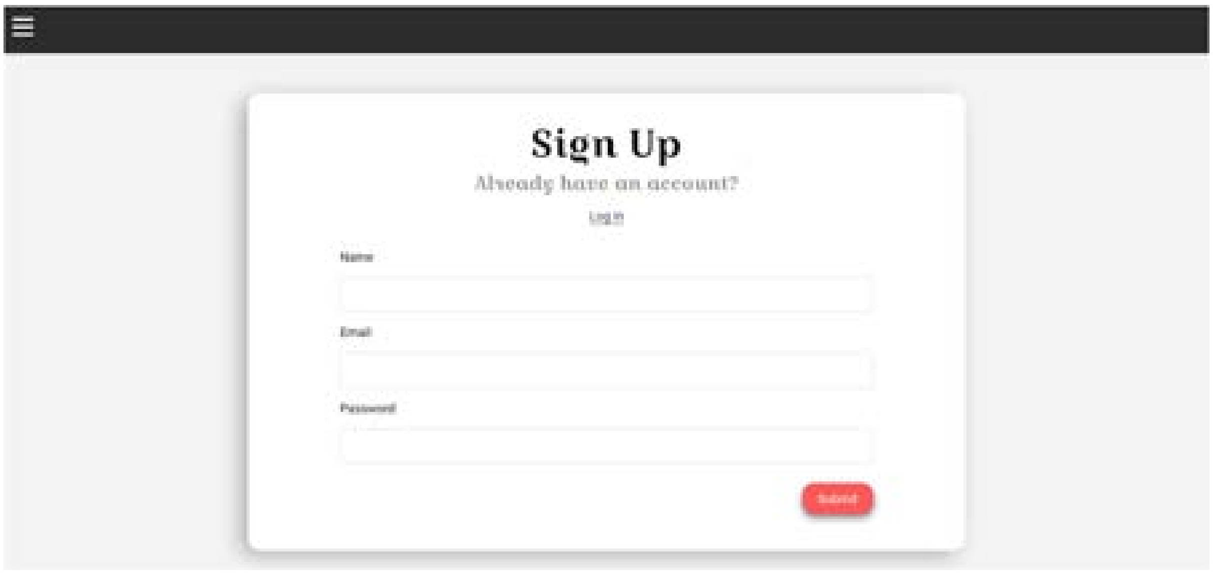


Figure 212 Register

Login:

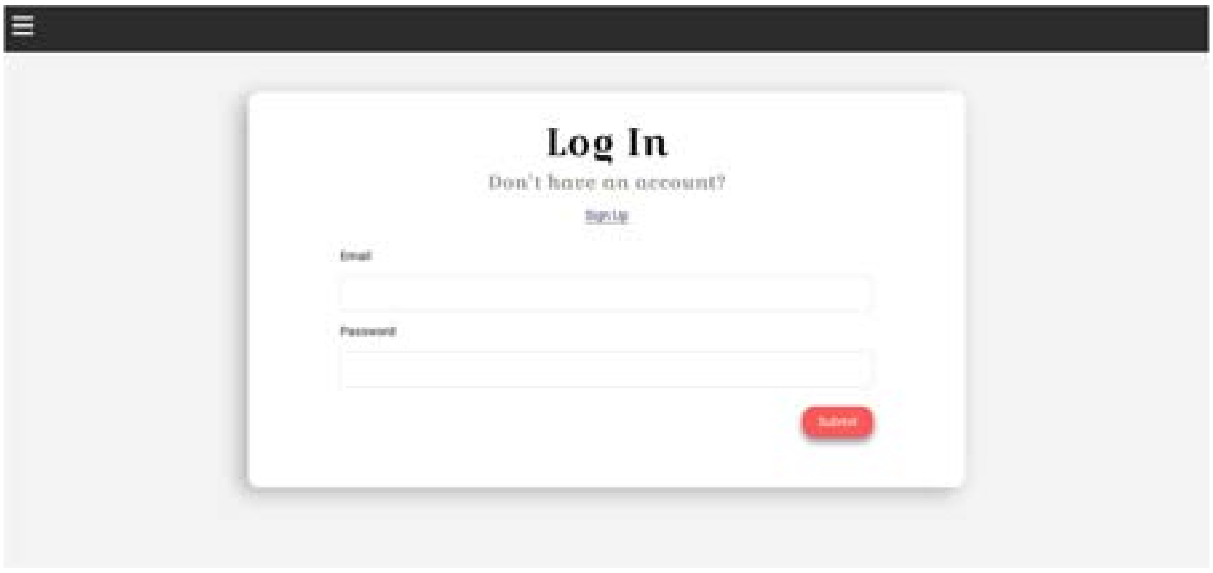


Figure 213 Login

OCR Reader:

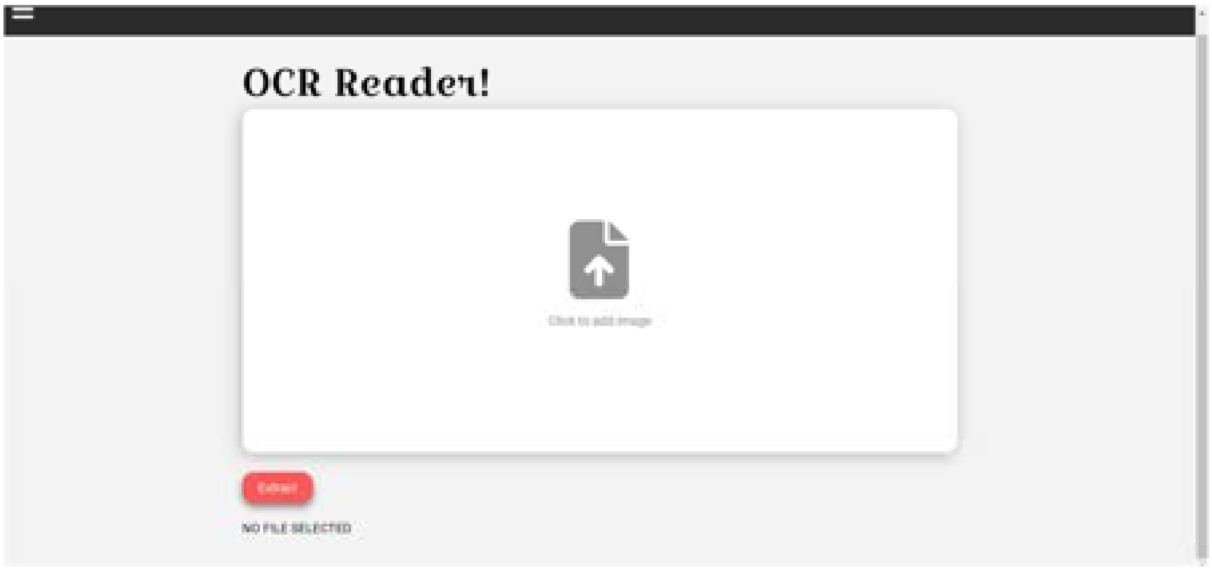
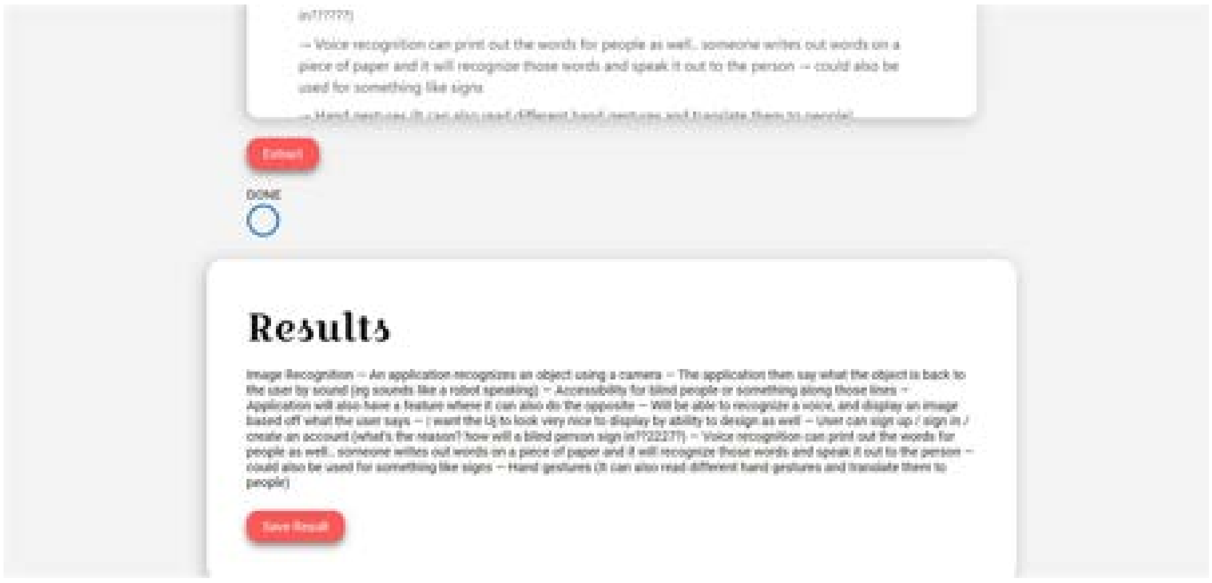


Figure 214 OCR Reader

OCR Reader Results:


in??????)

→ Voice recognition can print out the words for people as well.. someone writes out words on a piece of paper and it will recognize those words and speak it out to the person → could also be used for something like signs

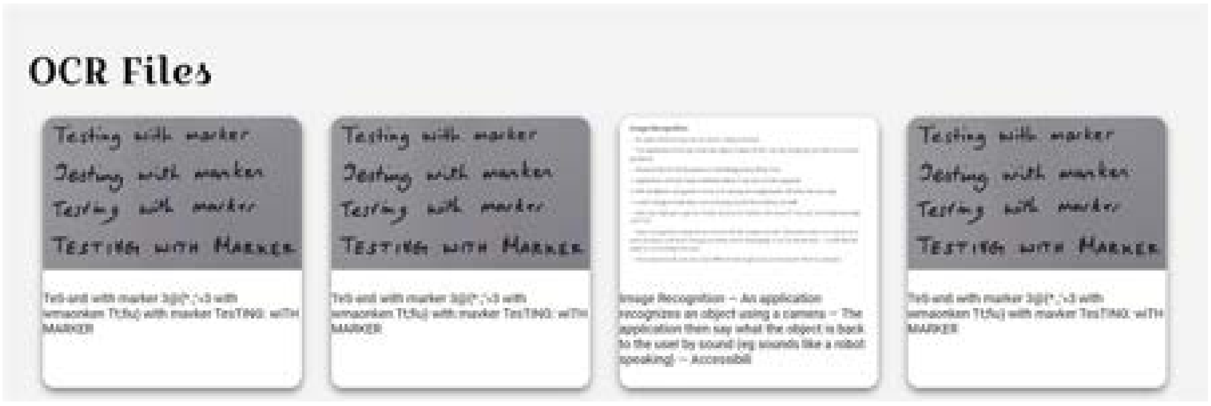
→ Hand gestures (It can also read different hand gestures and translate them to people)

Results

Image Recognition → An application recognizes an object using a camera → The application then say what the object is back to the user by sound (eg sounds like a robot speaking) → Accessibility for blind people or something along those lines → Application will also have a feature where it can also do the opposite → Will be able to recognize a voice, and display an image based off what the user says → I want the Ui to look very nice to display by ability to design as well → User can sign up / sign in / create an account (what's the reason? how will a blind person sign in??2227?) → Voice recognition can print out the words for people as well.. someone writes out words on a piece of paper and it will recognize those words and speak it out to the person → could also be used for something like signs → Hand gestures (It can also read different hand gestures and translate them to people)

Figure 215 OCR Reader Results

Viewing All OCR Files:



OCR Files

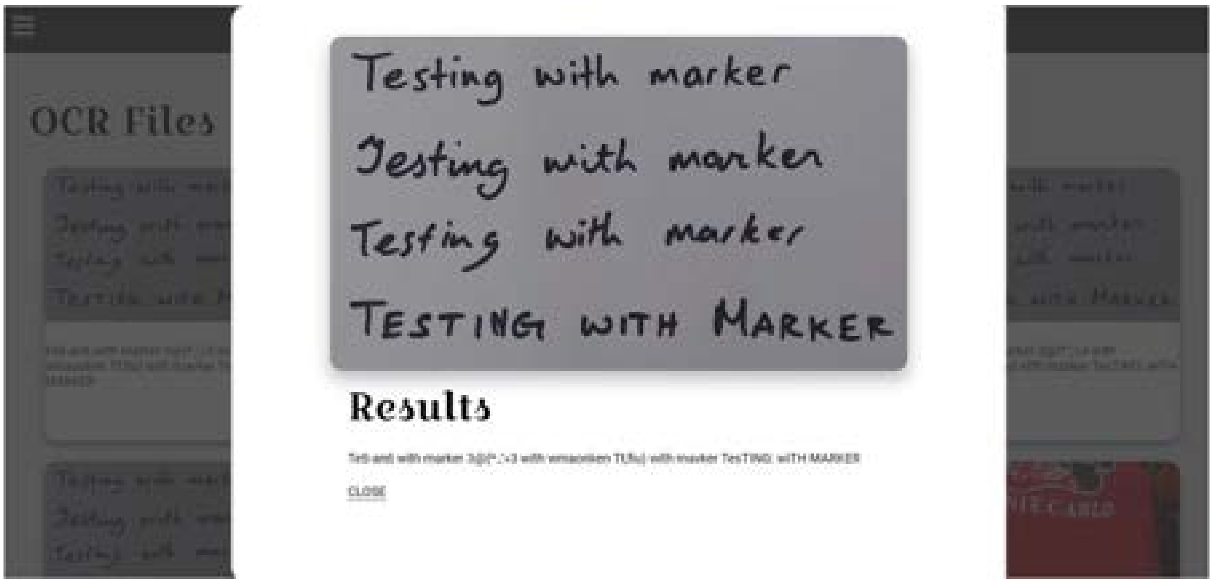
Testing with marker
Jesting with marken
Testing with marker
TESTING WITH MARKER

TeS-and with marker 3@(*,;<3 with wmaonken Tl;fu) with maxker TesTING: wlTH MARKER

Image Recognition ~ An application recognizes an object using a camera ~ The application then say what the object is back to the user by sound (eg sounds like a robot speaking) ~ Accessibili

Figure 216 View All OCR Files

Modal When Clicking to View OCR File:



Testing with marker
Jesting with marken
Testing with marker
TESTING WITH MARKER

Results

TeS-and with marker 3@(*,;<3 with wmaonken Tl;fu) with maxker TesTING: wlTH MARKER

Figure 217 View Single OCR File

View All Folders:

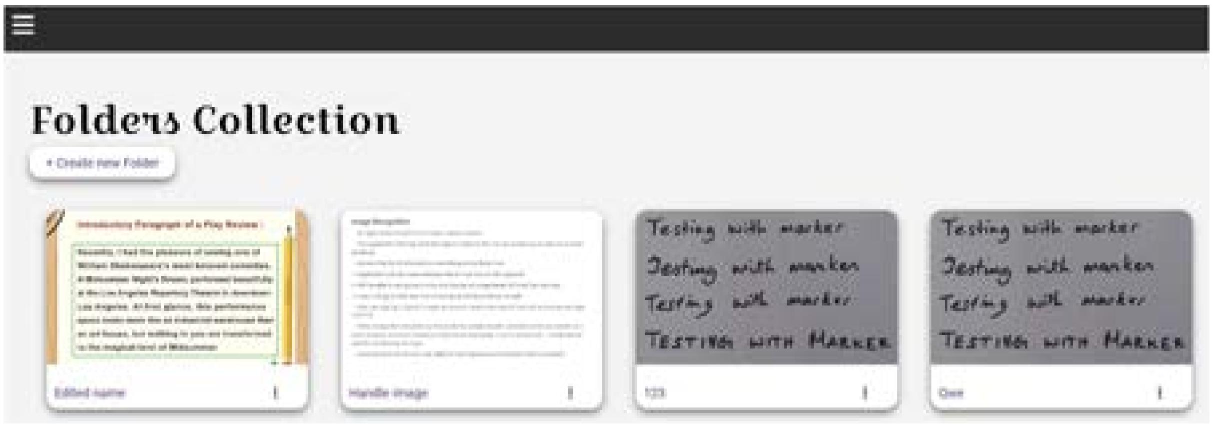


Figure 218 View All Folders

Create Folder:

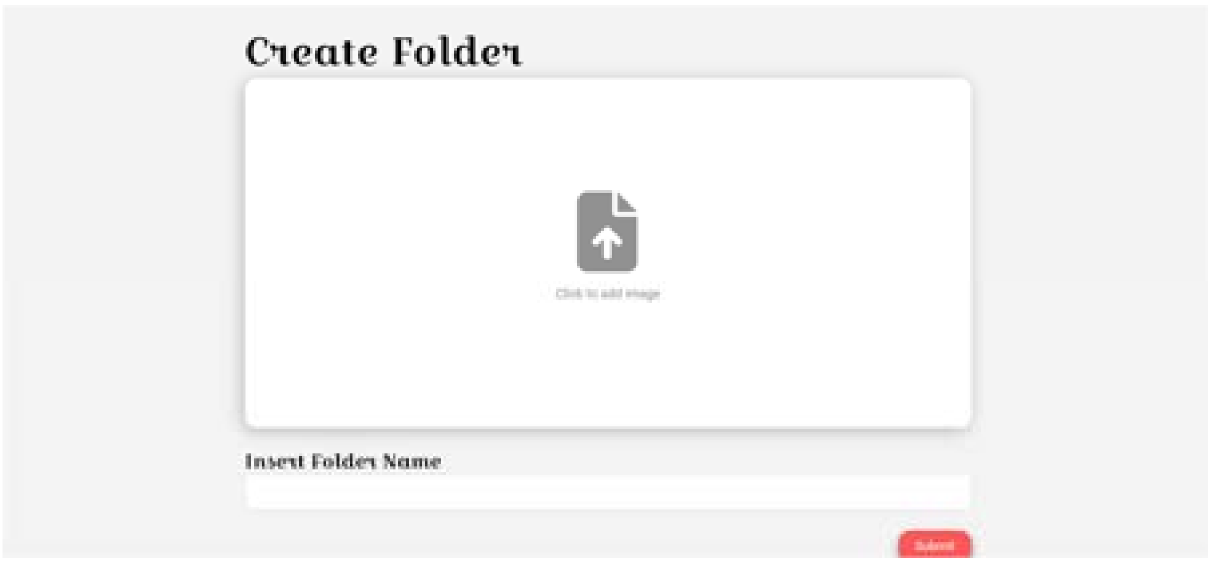


Figure 219 Create Folder

View All Documents / View A Single Folder:

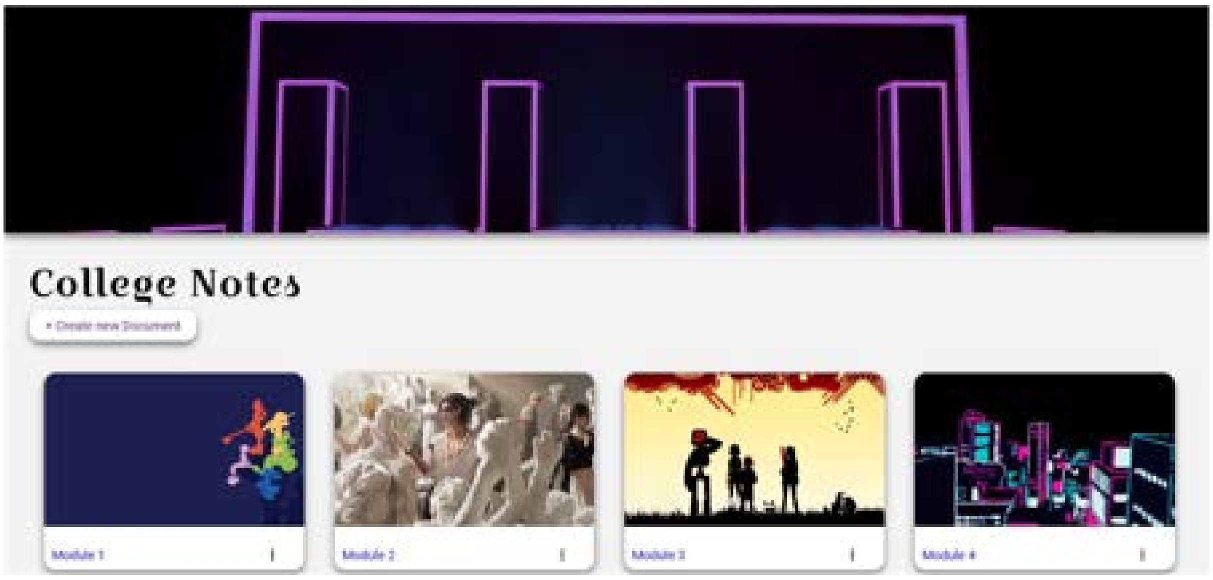


Figure 220 View All Documents

Viewing The Text Editor:

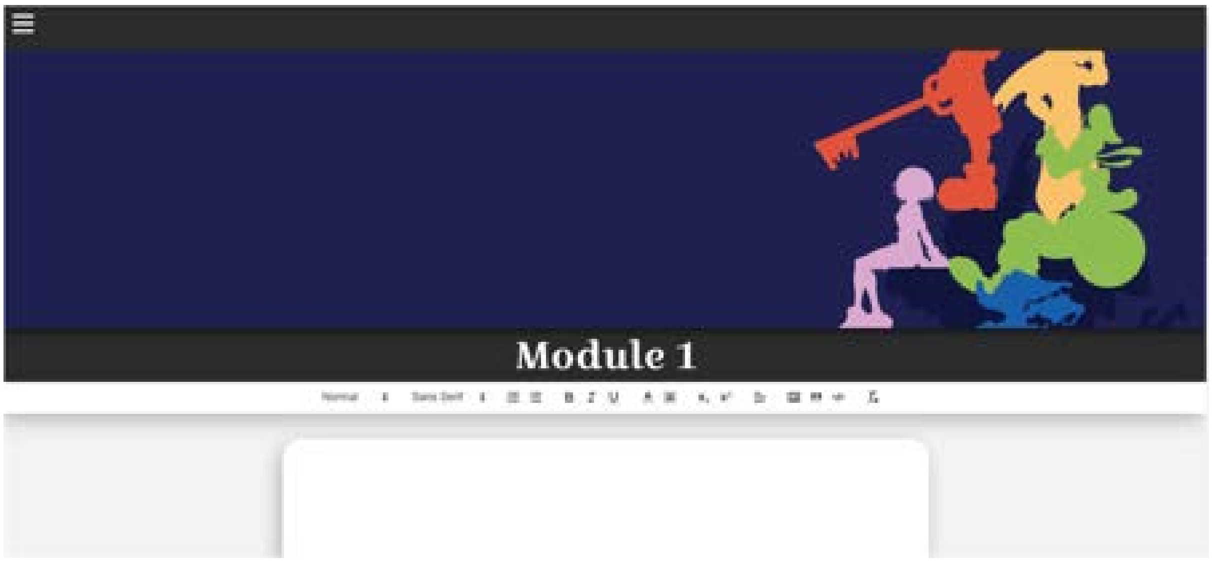


Figure 221 View Text Editor

Minor changes may be made before deployment for bug fixes / usability however the application will remain mostly the same for the most part.

5.13 Conclusion

In conclusion, the implementation of the application was displayed throughout this chapter. The project was developed using the SCRUM methodology consisting of 7 sprints in development of the application.

The SCRUM Methodology was discussed in detail in this chapter, explaining the different roles, sprints, and items within the sprints.

The technologies to be used were also discussed in this chapter, discussing React, GitHub / Git, etc.

These sprints each had their own items that was a backlog of features that were to be completed for the application.

The sprints consisted of:

- **Sprint 1:** Research
- **Sprint 2:** Design
- **Sprint 3:** Implementation 1
- **Sprint 4:** Design 2
- **Sprint 5:** Implementation 2 and Testing 1
- **Sprint 6:** Testing 2
- **Sprint 7:** Thesis

This chapter displayed the work and the development process of the application and explained some of the decisions made during development.

Through this, the application that was developed was a note taking application, that can use OCR in React.js, with a login and register, and folder system that can be used to store the notes found within the application.

6 Testing

6.1 Introduction

This chapter discuss the forms of testing that has been done during the development of the application. This chapter contains two sections:

1. Functional Testing
2. User Testing

Functional testing is a kind of software testing, where the application is tested for its functional requirements. The app is tested to see if the output expected is accurate to the output returned by the application. These tests are based on the requirements of the application, and its results will display whether that piece of functionality in the application is working, and easy to use.

User testing will look at the application, to see if a user is able to interact with the program in a way that is easy to use. This can include how the user navigates through the application, or whether they can find functionality easily.

6.2 Functional Testing

This section discusses the functional testing that were carried out throughout the application. The functional test categorised in:

- Navigation
- CRUD

Black Box Testing is generally used during functional testing, which means that the tester is only interested in the actual output matching the expected output of the application.

6.2.1 Navigation

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Navigate to Sign up Page from homepage	Click on Register in Nav or "Get Started button on Homepage	Redirect the user to Register page	Redirects user to register page	User can successfully navigate to register page

2	Create an account with invalid details.	Name: NewName Email: anotherInvalidEmail Password: 1	Don't allow user to sign up with invalid details	Allows user to sign up with invalid details	The register form should have a way to check it's contents to prevent invalid user email.
3	Sign up with valid details	Name: newTestingName Email: newTestingName@gmail.com Password: password	Allow user to sign up with valid details	User signs up with valid details	Password verification could be added
4	Log out of applicatio n	User navigates to "logout" button on navbar	Logs user out and redirects to homepage	The user is logged out and redirected to homepage	
5	Navigate to Log In page	User navigates to "login" page through navbar	User is redirected to "Login" page	User was redirected to "Login" page	
6	Login with incorrect details	Email: Emailnotavailable Password: incorrectPassword	User is not permitted to log in and validation error message is shown	User was not logged in but no validation error was shown	Validation should be implement ed if time is available
7	Login with correct details	Email: newTestingName@gmail.com Password: password	User gets logged in	User was logged in	User should be redirected to folders page
8	Navigate to OCR Reader Page	User clicks "OCR Reader" in nav	User is redirected to OCR Reader page	User gets redirected to OCR Reader page	

9	Use OCR Reader	Add image to extract text and hit the "extract" button	Text gets extracted	Text was extracted and results were shown	The page should move down when page is shown
10	Save OCR Result	User should push the "Save Result" button	Saves result and redirects user	Saves result and redirects user	
11	Navigate to View All OCR Files	User either navigates to view all files from the "Save Result" button or through navbar	Redirects user to see all OCR Results	User is redirected to see all OCR Results	
12	View Single OCR File	User should click on the card to view the details of a single file	User opens the modal for OCR File	Modal is opened for single OCR File	
13	Navigate to Folders page	Click on the "Select Folder" button in nav	The user is redirected to view all their folders	The user is redirected to their folders	
14	Navigate to create folder page	Click on the "Create Folder" button	User is redirected to create folder page	User was redirected to create folder page	
15	Use Create Folder	Insert an image and title. Title: Folder Test	User creates a folder and is redirected to folders page	User creates folder and was redirected to folders page	
16	Select one folder	Click on the single folder	User views content of single folder	User views content of single folder	
17	Navigate to create documents page	User clicks on "Create Document"	User is brought to the create document page	User was brought to the create document page	

18	Use Create Documents	User inputs an image and document title Title: New Document	User makes a new document and is redirected	User made a new document and was redirected	
19	Navigate to text editor	Click to view the document text editor	User is redirected to the text editor	User was redirected to text editor	
20	Use Text Editor	Input some text to use the text editor	Text inputted to text	Text was inputted and saved to text	There should be a way to show that the text editor autosaves
21	Return to homepage from text editor	Click on homepage button in Nav	Redirects user to homepage from anywhere	User is redirected to homepage	
22	Navigate to use edit user modal	Click on “User Edit” modal in navbar	User Edit Modal appears	User edit modal appeared	

6.2.2 CRUD

Test No	Description of test case	Input	Expected Output	Actual Output	Comment
1	Create New User	Input user details	User is created	User was created	
	Update User	Input new user details	User is updated	User was updated	
	View User Details	Not available in front end	Not available in front end	Not available in front end	Should be implemented if there is time
	Delete User	Not available in front end	Not available in front end	Not available in front end	Should be implemented if there is time
	Create New OCR File	Input image to extract and save results	Text from image to be extracted and results saved	Text from image was extracted and results were saved	
	View OCR File	Go to View all OCR page	All saved OCR Results are viewable	All saved OCR Results were viewable	
	Update OCR File	Function not available in frontend	Function not available in frontend	Function not available in frontend	Should be implemented if there is time
	Delete OCR File	Function not available in frontend	Function not available in frontend	Function not available in frontend	Should be implemented if there is time
	Create New Folder	Input folder details	Folder should be created	Folder was created	
	View All Folders	Go to view all folders page	All folders should be displayed	All folders were displayed	
	View Single Folder	Go to a single folder	The documents stored in the	The documents in the single	

			folder should be displayed	folder were displayed	
	Update Folder	The edit folder modal should appear	The folder title should be edited	The folder title was edited	
	Delete Folder	The delete folder button should be clicked	The folder should be deleted	The folder was deleted	There should be something to notify the user that the folder was deleted
	Create New Document	The user should go to the create new document page and input the details for a new document	The new document should be created	The new document was created	
	View All Documents	The user will navigate to the view all documents page	All the documents created within that folder should be displayed	All the documents created within that folder are displayed	
	View Single Document	The user will click into the document and see the contents of that document	The document should display the content from the text editor and the saved text	The document displayed the content from the text editor and the saved text	
	Update Document	The user should open the edit user modal and input the new information	The edit user modal should appear and save the updated document	The edit user modal appears and saves the updated document	

	Delete Document	The delete button was pushed to delete the document	The document should be deleted	The document was deleted	

6.2.3 Discussion of Functional Testing Results

Although there are some features missing, namely the verification of the register and login forms, the functional testing went mostly as intended, with most features being available to use. The more important CRUD Functionalities were all working as intended and performed the tasks that was needed to be performed.

Improvements can still be performed, however due to time constraints, it might not be possible to implement all improvements to the application.

6.3 OCR Reader Accuracy Testing

The accuracy of the OCR Reader was tested using various methods that consist of:

- Colour Inversion
- Threshold Filtering
- Dilation
- Gaussian Blur
- Noise Removal

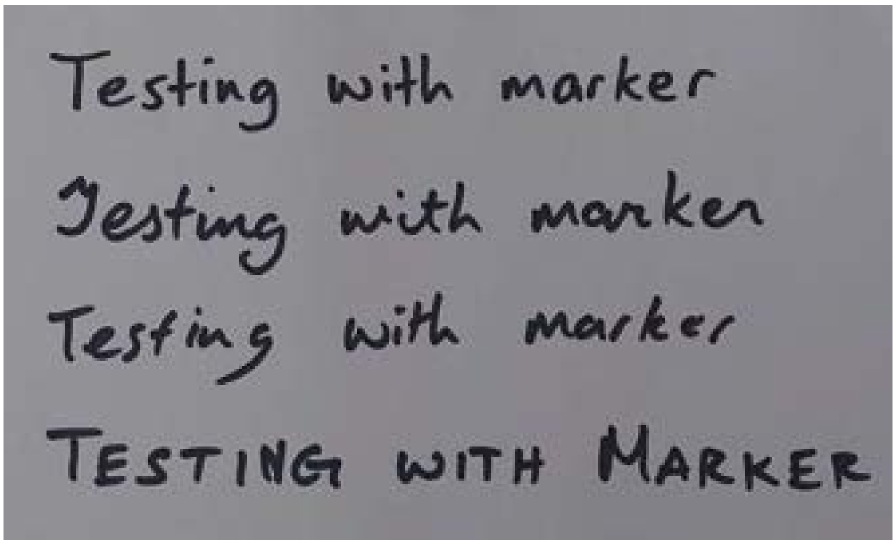
Some of these methods were mentioned in the research section, such as noise removal, threshold filtering and colour inversion, although the topics that were not covered in the research section will be briefly explained in their respective section.

The purpose behind testing the OCR Reader was to see if the accuracy could be improved upon by applying these pre-processing methods. As discussed in the research section of the report, these methods can greatly alter the results of the accuracy, by potentially making it easier for the Tesseract OCR library to read the inputted image and extract the text.

The code used for pre-processing can be found here: (Chan, 2020)

Note that this code was only used for testing and was only applied to see how it would affect the accuracy of the output. The code was not used in the final project.

The image that will be tested will be figure 215 seen below.



Testing with marker
Testing with marker
Testing with marker
TESTING WITH MARKER

Figure 222 Tested Image

The results for this image will be discussed in detail on how it was tested and a collection of image results from testing the OCR Reader, listing what functions were used together when attempting to read it.

It is also worth noting that the package "p5" was installed during the tests to enable the filter algorithms. P5 can be found here: (p5, 2023)

The reason behind using the selected image, is to test four different things:

- Regular handwriting
- Joint handwriting
- Writing Quickly
- Writing in block capitals

This was done to test different scenarios that the user could potentially find themselves in when taking notes.

6.3.1 No Pre-Processing

The result shown below will be the accuracy of the OCR Reader without any pre-processing applied. This can be seen in the figures below:

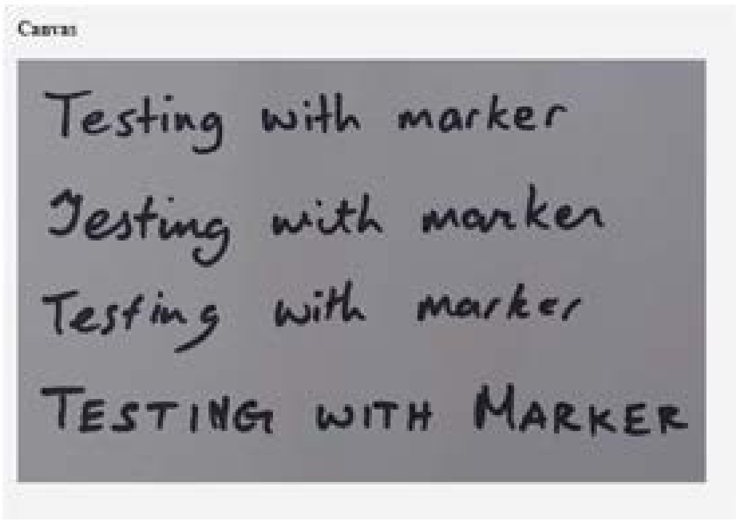
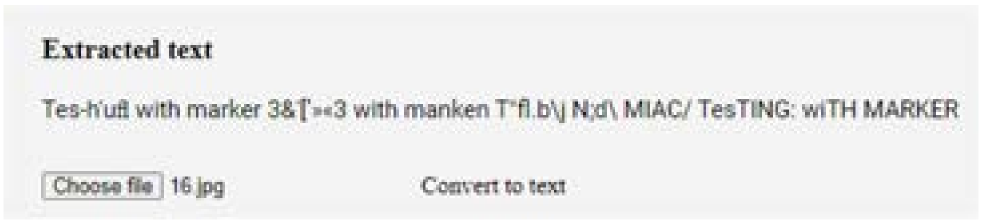


Figure 223 No Pre-processing applied.



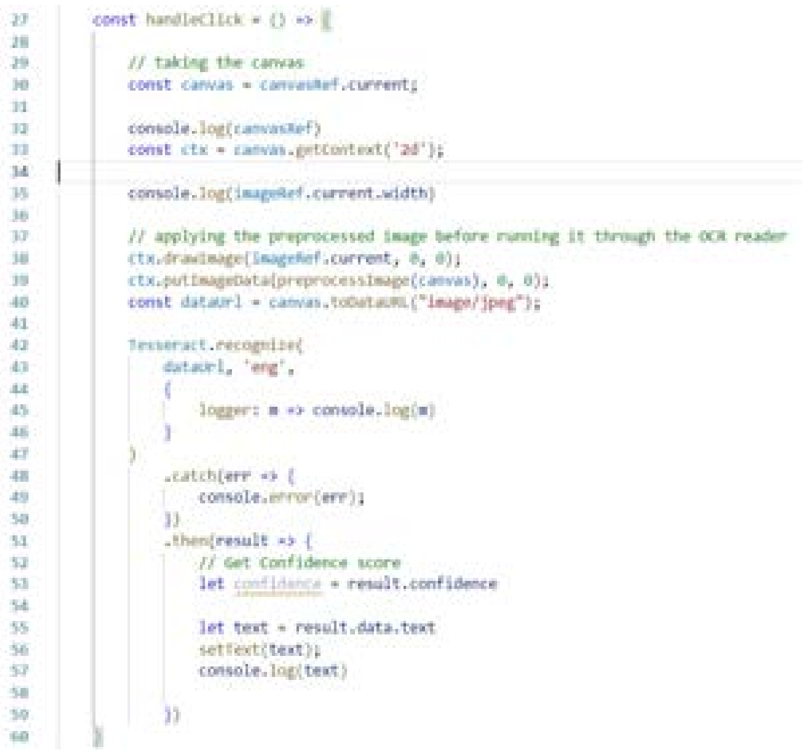
Extracted text

```
Tes-h'uft with marker 3&['»«3 with manken T"fl.b\j N;d\ MIAC/ TesTING: wiTH MARKER
```

Figure 224 Result

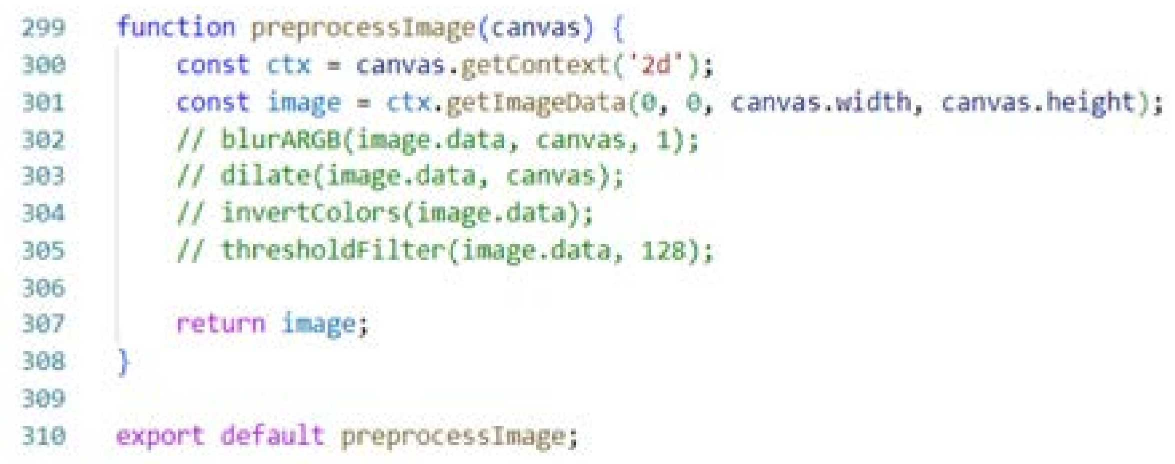
The results shown in the figure above has some inaccuracies, although reads most of the text. Some of the words are incoherent, and some parts of it were extracted properly, seen at the end when it is reading the capitalized font.

The changes in the code can be seen in figure 218 and 219, where it is applied to a canvas, and read with most of the pre-processing functionality commented out.



```javascript
const handleClick = () => {

    // taking the canvas
    const canvas = canvasRef.current;

    console.log(canvasRef)
    const ctx = canvas.getContext('2d');

    console.log(imageRef.current.width)

    // applying the preprocessed image before running it through the OCR reader
    ctx.drawImage(imageRef.current, 0, 0);
    ctx.putImageData(preprocessImage(canvas), 0, 0);
    const dataUrl = canvas.toDataURL("image/jpeg");

    Tesseract.recognize(
        dataUrl, 'eng',
        {
            logger: m => console.log(m)
        }
    )
        .catch(err => {
            console.error(err);
        })
        .then(result => {
            // Get Confidence score
            let confidence = result.confidence

            let text = result.data.text
            setText(text);
            console.log(text)

        })
}
```

Figure 225 Testing Code



```javascript
function preprocessImage(canvas) {
    const ctx = canvas.getContext('2d');
    const image = ctx.getImageData(0, 0, canvas.width, canvas.height);
    // blurARGB(image.data, canvas, 1);
    // dilate(image.data, canvas);
    // invertColors(image.data);
    // thresholdFilter(image.data, 128);

    return image;
}

export default preprocessImage;
```

Figure 226 Pre-process Image Function

Figure 218 and 219 shows the slight alterations made in the OCR Reader code, notably using the pre-process function before returning the results, although in this case it is commented out, and returns the default result.

We will first start by adding a Gaussian Blur to the pre-processing.

6.3.2 Gaussian Blur

The gaussian blur can help by reducing the noise found within the image that is submitted. The image that was tested looked much smoother than the original and can be seen in figure 220.

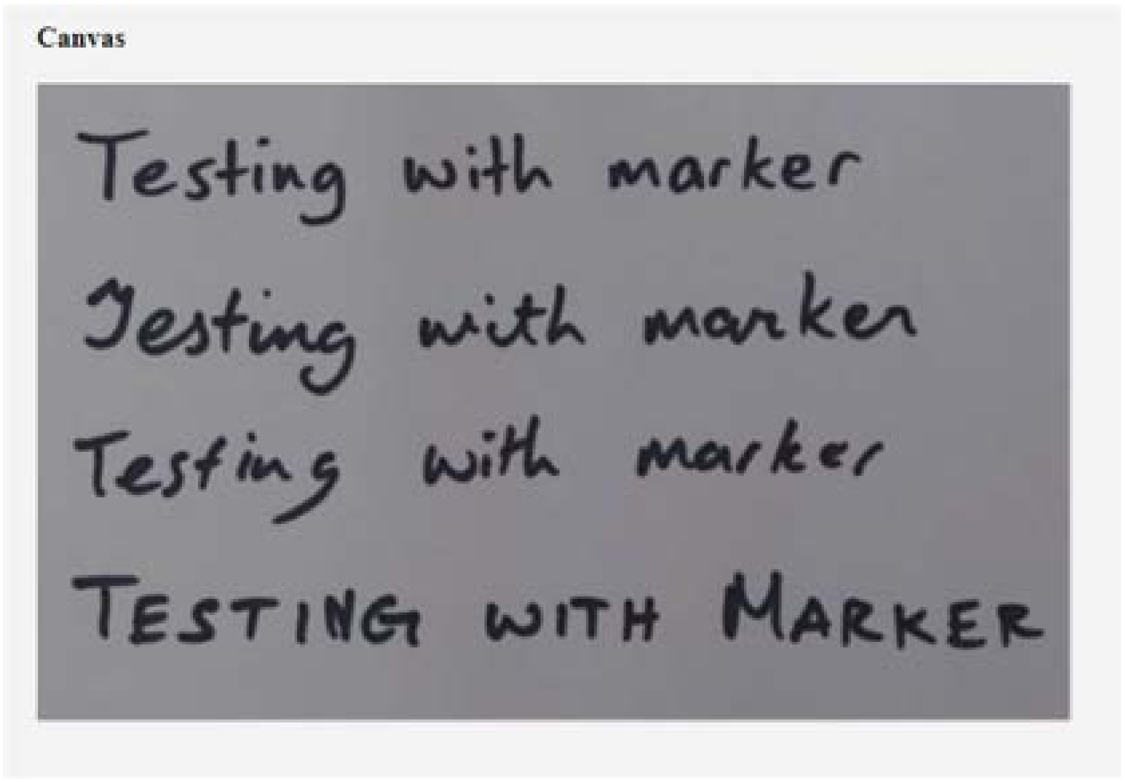
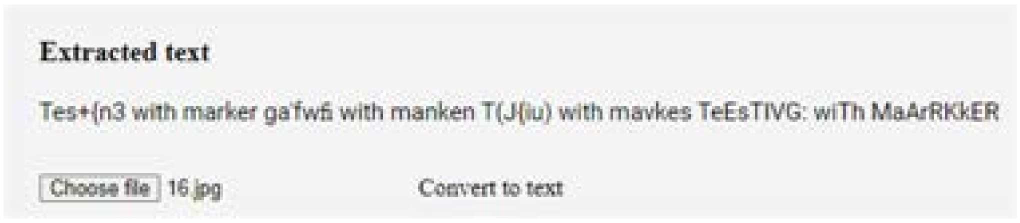


Figure 227 With Gaussian Blur

The result shown contains less noise and is slightly smoother. This could potentially provide a more accurate reading. The result can be seen in figure 221.



Extracted text

```
Tes+(n3 with marker ga'fw₆ with manken T(J(iu) with mavkes TeEsTIVG: wiTh MaArRKkER
```

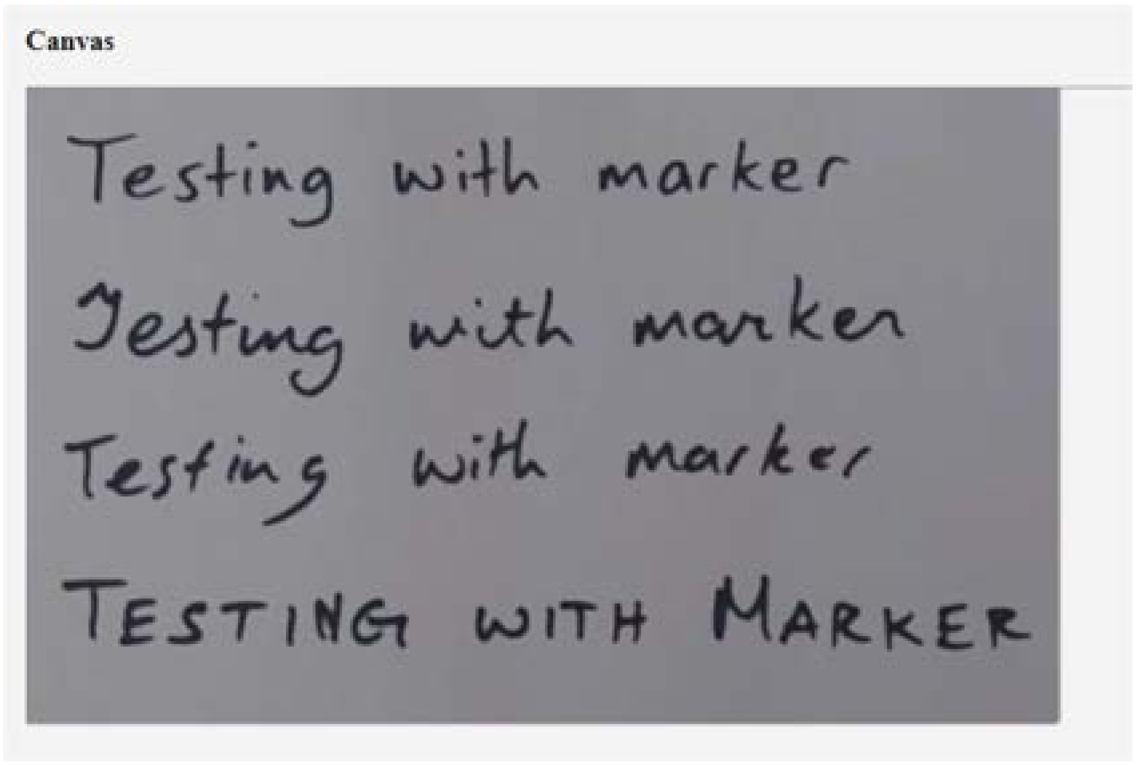
Figure 228 Result

The result of the extracted text is similar, with some parts of inaccuracies.

The next method of pre-processing to be applied will be image dilation, on top of the Gaussian Blur.

6.3.3 Image Dilation

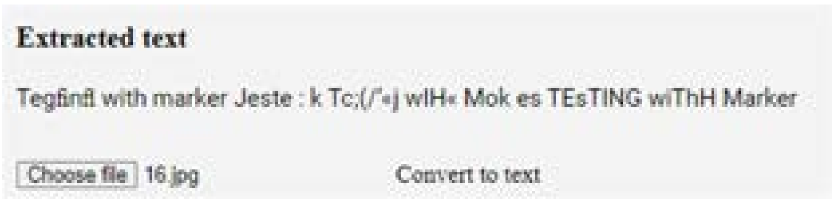
Image dilation simply brightens the lighter pixels. The result of the inputted text can be seen in figure 222:



Testing with marker
Testing with marker
Testing with marker
TESTING WITH MARKER

Figure 229 Image Dilation

The font looks slightly thinner than the previous result, and lead to a slightly different result after text extraction. This is seen in figure 223:



Tegfinfl with marker Jeste : k Tc;(/"•j wIH• Mok es TEsTING wiThH Marker

Figure 230 Result

The result shows a slightly more accurate reading, showing the block capital results being the most accurate.

The next pre-processing step will be the colour inversion, as well as the previous pre-processing methods.

6.3.4 Colour Inversion

Colour inversion will invert the colours for text extraction. The inputted image before extraction is seen in figure 224:

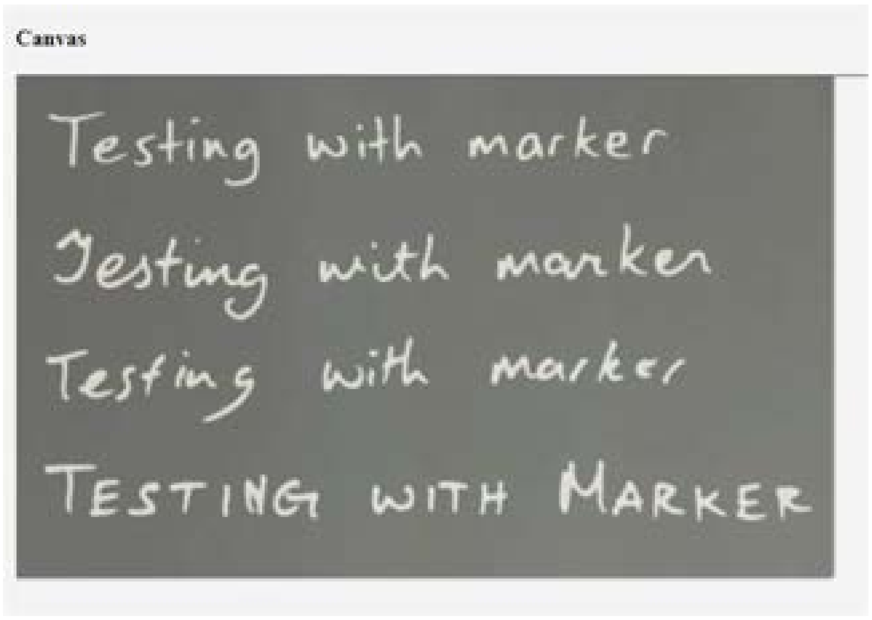
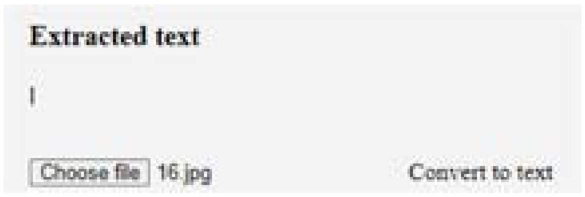


Figure 231 Colour Inversion

This will most likely not be very effective, as it is searching for the typography but finds it difficult to find the text due to the text blending in with the background.

The result of the text extraction is in figure 225:



```
|
```

Figure 232 Result

Very minimal text was extracted as it couldn't find any text to extract.

Finally, the threshold filter will be applied to show all the pre-processing methods combined.

6.3.5 Threshold Filter

The final filter makes the dark pixels darker, and light pixels lighter. Now that all the pre-processing filters have been applied, the result is seen in figure 226:

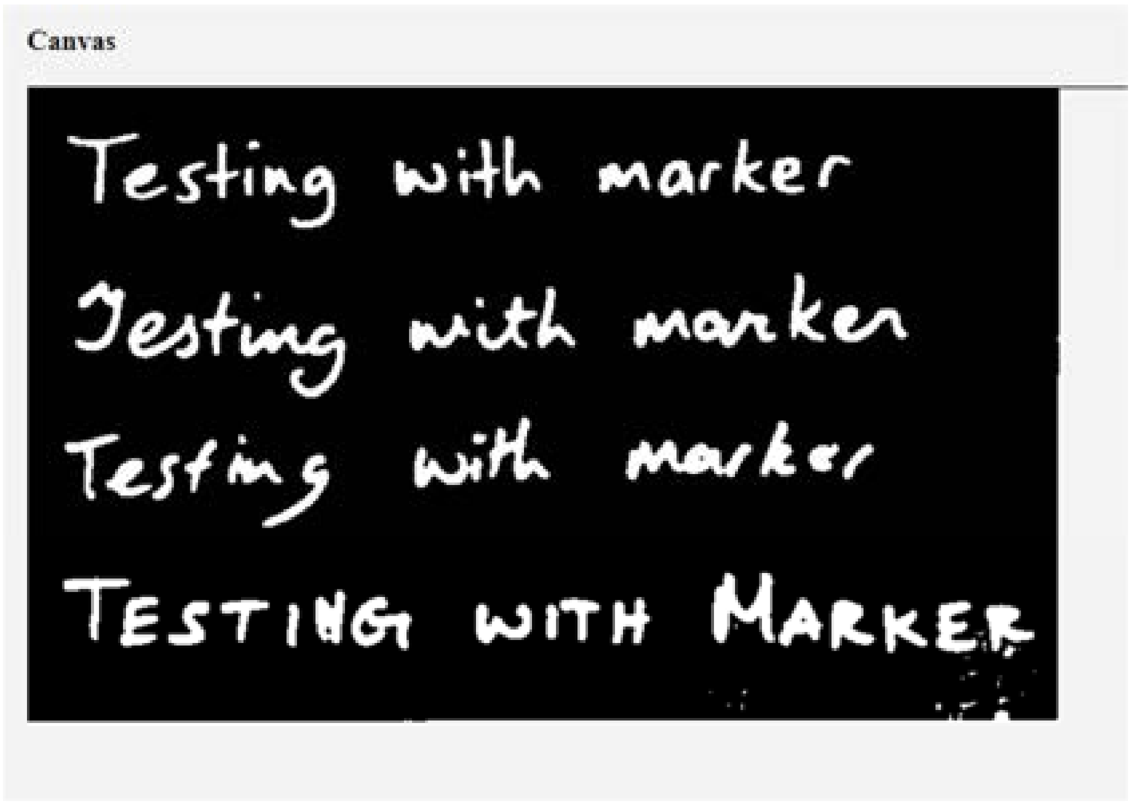
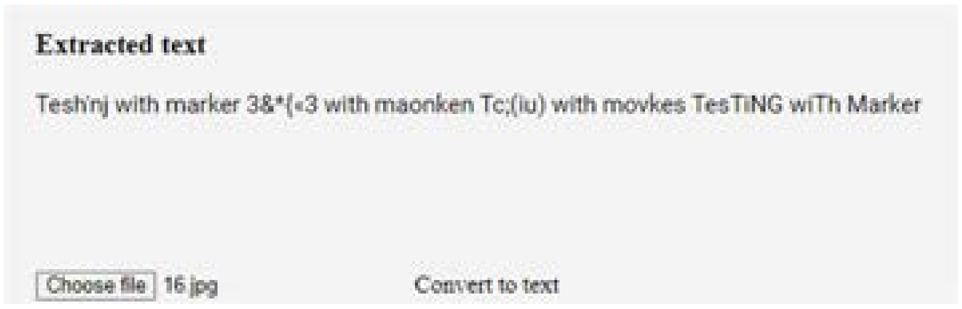


Figure 233 Threshold Filter

With the threshold filter, it blackens the dark pixels, and turns the light pixels to white, allowing for easier reading. The result is seen in figure 227:


Extracted text

Tesh'nj with marker 3&*(«3 with maonken Tc;(iu) with movkes TesTiNG wiTh Marker

Figure 234 Result

It is slightly more accurate, and as seen in the result the block text is the most accurate, as it finds it easiest to read with all the pre-processing filters applied.

6.3.6 Mixing Filters

To test the accuracy fully, mixing the different filters and testing the results will be done. As the filters were discussed in more detail in their respective sections, the name of the filter and the result will simply be shown. The filters selected will be in a random order to find the best source of accuracy.

Gaussian Blur + Threshold Filter:

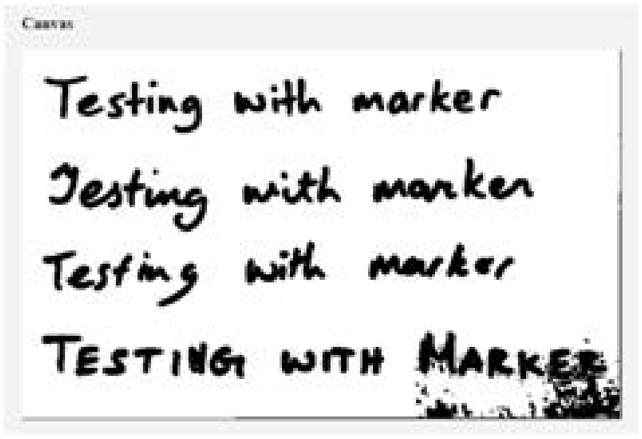
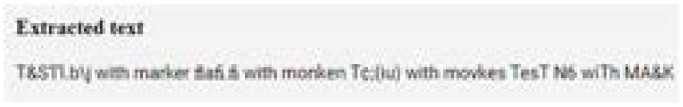


Figure 235 Gaussian Blur + Threshold Filter



Extracted text

T&STI.b'j with marker ﾀaß.ß with monken Tc;(iu) with movkes TesT N6 wiTh MA&K

Figure 236 Result

Gaussian Blur + Dilation + Threshold Filter:

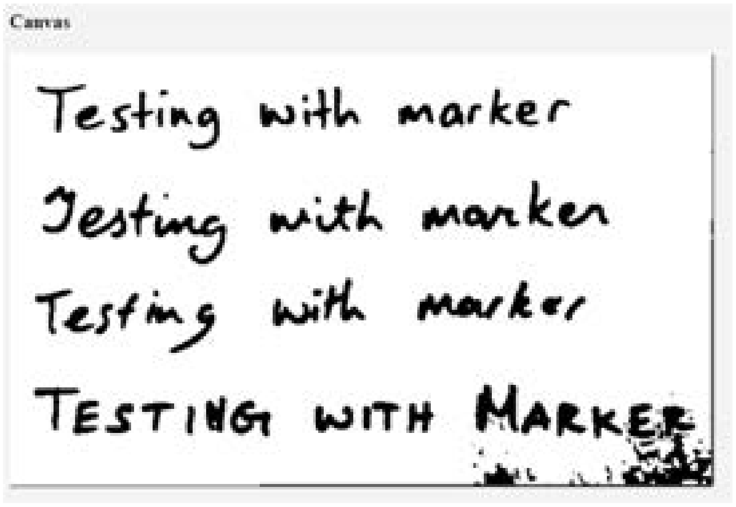
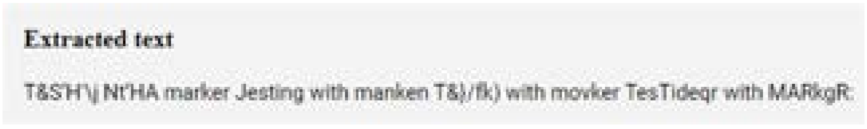


Figure 237 Gaussian Blur, Dilation and Threshold Filter



Extracted text

T&S'H'\j Nt'HA marker Jesting with manken T&)/fk) with movker TesTideqr with MARkgR:

Figure 238 Result

Dilation + Threshold Filter

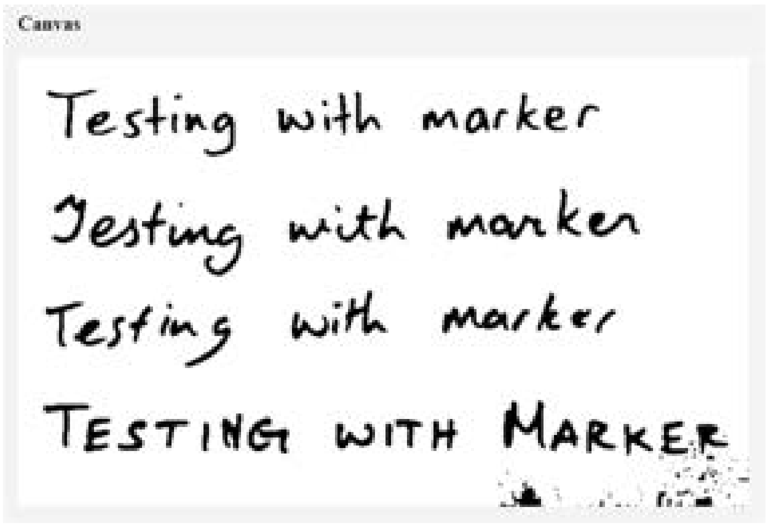
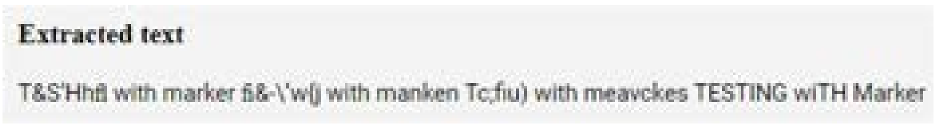


Figure 239 Dilation and Threshold Filter


Extracted text

T&S'Hhß with marker ß&-\'w(j with manken Tc;fiu) with meavckes TESTING wiTH Marker

Figure 240 Result

Colour Inversion + Threshold Filter:

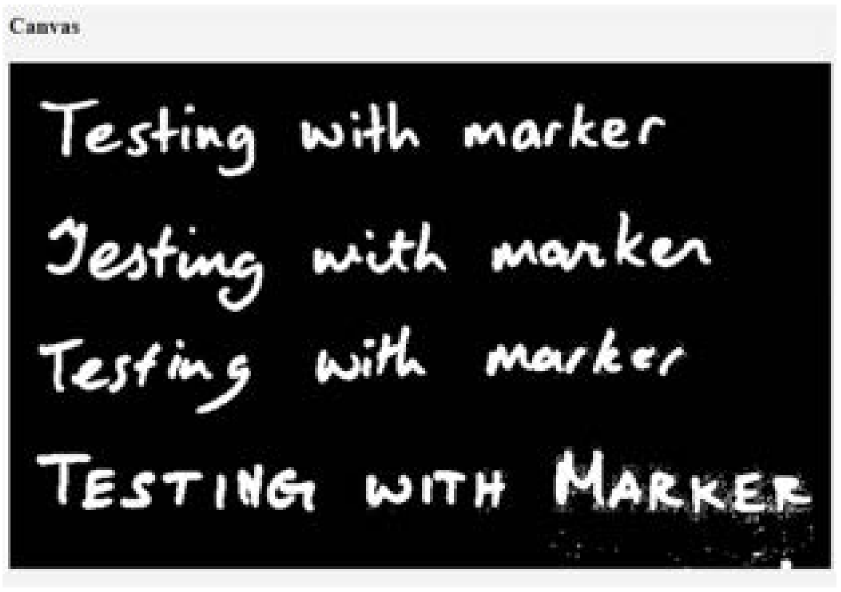
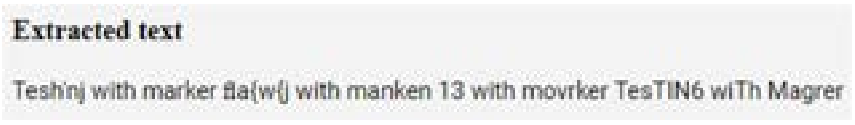


Figure 241 Colour Inversion and Threshold Filter


Extracted text

Tesh'nj with marker ßa{w(j with manken 13 with movrker TesTIN6 wiTh Magrer

Figure 242 Result

Threshold Filter Only:

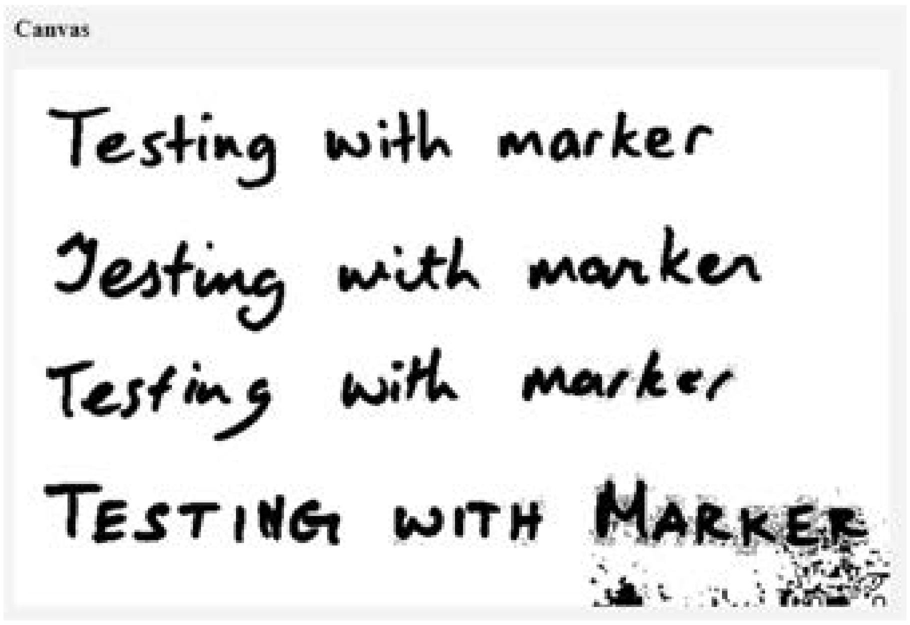
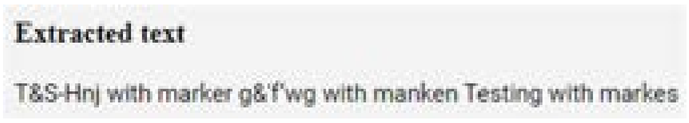


Figure 243 Threshold Filter Only



Extracted text

T&S-Hnj with marker g&'f'wg with manken Testing with markes

Figure 244 Result

6.3.7 Conclusion

In conclusion pre-processing allowed the developer to test the accuracy of the OCR Reader using the methods shown.

From the results, the most notable combinations are dilation and threshold filter and all the pre-processing methods combined.

These pre-processing methods are slightly more accurate compared using no filters at all, however more tests that uses a variety of different images can yield different results.

6.4 User Testing

The purpose of user testing is to unravel underlying issues that are still present in the application. User testing will reveal issues found in the navigation of the application, and hidden bugs that the developers have not found.

This allows developers to improve their application, by performing changes in the design and bug fixes, as well as improving usability for common users.

6.4.1 Application Overview

The goal of the application is to allow users to create notes and use OCR functionalities to help with their notes taking process, and potentially transfer notes from images.

Users should be able to view their own work and organize their notes into the required folders.

The users should also be able to view all their own files.

Research was done before development of the application to ensure that the functionality and design of the app was of good quality and met the user’s needs.

6.4.2 User Tasks

During the user testing phase, the users were given tasks that they needed to accomplish. The goal of this was to see if the users can navigate their way through the application without encountering any issues, and use the required features as intended.

This also allowed users and developers to encounter bugs and issues that are relevant to the application during user testing. This allows developers to see flaws and improve the application.

User 1:

User 1 was given the tasks of:

- Sign up.
- User OCR Reader.
- Create a folder.
- Create a Document.

Some Issues that this user encountered were:

- This user did not know what OCR is.
- When the OCR Reader has been completed, the user was unsure on where to see the result, as the page didn't move down to display the result.
- The Navbar Link does not work, the user was trying to click on the general area of the link, including the icon and was not navigating to the page. It only worked when the user clicks on the words.

- The user was not a huge fan of the homepage.
- The user tried to choose several images. Breaks when the user picks several documents. This might be because one of the files that the user has chosen was not supported.
- The user was confused on whether the document was saving. There should be an icon to indicate that it autosaves every 2 seconds.

This provided great insight a few flaws of the application and shows parts of the application that need to be fixed for usability purposes. The simple issues found within the application, namely the issues with the nav bar should be addressed quickly to ensure that the user-interface of the application is of high standard.

User 2:

User 2 was given the task of:

- Sign up.
- User OCR Reader
- Create a folder.
- Create a Document
- Copy and paste OCR results to document.
- Edit the document name and folder name.
- Edit User Content

Things that were worth noting from this user were:

- User was having problems with the navbar again, clicking around and it not navigating.
- The user managed to register properly without minor issues. They expected the enter key to work and although both users expected to be redirected to a different page.
- When using the OCR Reader, after the results are shown an auto scroll should be done as this user didn’t know either.
- The user didn't know that it autosaves either.
- This user however didn’t know how to find folders and the documents page. The user was struggling to find how to create a document as it is nested in the folder. The naming should be clearer in the navigation.
- There were no problems when it came to editing the folder and document.
- When editing their profile however, the password was still being shown. The user was wondering if there was a way to hide or show this password or if there was a method to implement it.
- The user wasn't very fond of the white background and would prefer a slightly darker design. A dark mode could be implemented for this cause; however, it might take too long to develop a mode like that.

- The user also thinks that the folders page looks slightly empty when there are no folders available to it.
- The user also would like to see if they can align the folders and documents to the left.
- A slight separation line from the create folder and the folders themselves.
- Adding a logo to the navbar would be nice too.
- The user liked the on-hover effect.

This user provided great insight into the problems still prevalent in the application, and these problems will hopefully be addressed and fixed.

6.4.3 Usability Testing Participants

There were two students that participated in the usability testing, who are familiar with technology and notes-taking applications. Even so, the students still needed explanation on what OCR was, and how those features worked, and what it does.

The target demographic for this application was reached within the participants, being college students who regularly take notes, and managed to yield results that were insightful to the application, to allow for a more user-friendly application.

The participants found that the app itself wasn't too difficult to navigate and found the design and user-interface to be simple and easy to follow.

6.4.4 Usability Factors

6.4.4.1 Ease of use

The ease of use of the application should be of high quality. The usability of the application should be of the highest standard, so that when users are using the application, problems don't occur that could potentially frustrate the user and stop them from returning to the website.

The user testing will ensure that problems that some users are facing now can be fixed during development to create a more user-friendly product.

6.4.4.2 Test Environment

The user testing was conducted in a small classroom in college, with a laptop that the application will run on, as well as the developer observing their actions and taking notes on how the user is performing.

The test is conducted on the developer's laptop, to ensure the intended screen sizes that should have been used, as compatibility with different screen sizes were unavailable.

The user is free to ask questions from the developer in case the users being tested are unable to find specific parts of the application, which occurred during the user testing, as one of the users were unable to find the "folders" page.

The reason for taking notes is to ensure that the developer can take in as much information as possible in short bullet-points. This information includes what the developer expected the user to do, as well as noting down things that the developer didn’t expect the user to do.

Feedback was also given, the developer asks the users what they liked and didn’t like about the application, and the developer noted down the feedback that was returned and can be seen in the results of the user tasks.

6.5 Conclusion

In conclusion, the testing chapter carries out the different forms of testing, ranging from the navigation of the application to the CRUD functionalities seen throughout the app, as well as how the users interacted with it.

Different forms of user-testing were performed, and feedback was taken to allow for development and for improvements to be made throughout the application itself.

This phase of development is essential in the improvement of the application and improve the overall quality of the final product.

Although there may not be enough time to apply these changes, the information gathered from the user testing is crucial in letting the developer know the flaws within the product.

The OCR Reader was also tested, to see if the accuracy of it can also be improved upon by adding pre-processing methods before text extraction begins.

This chapter proved to be essential in the development of the application, for seeking flaws and how to improve the product.

7 Project Management

7.1 Introduction

The project management phase describes how well the project was managed, and how the student and the supervisors worked with each other to help with the creation of the application. The various phases of the project are shown, as well as the steps that were taken to before the development of the application.

The discussion of Trello, GitHub and a project management journal were also used to assist during the project management phase.

7.2 Project Phases

This section describes the different phases throughout the project and will explain any issue that arose from each phase.

7.2.1 Proposal

The proposal phase of the application was to allow students to come up with ideas for projects that they can work on. During this phase ideas were brainstormed, and the idea that was agreed upon was to develop a note taking application, with the use of OCR functionalities.

This idea was expanded upon throughout the various phases of the project to develop into the final product.

As notes taking applications were always commonly used in college, ideas for functionality were already being thought upon, that could potentially be developed for the application.

The idea for this project was then discussed with the supervisor of this project and allowed research to begin, starting with the gathering of requirements.

7.2.2 Requirements

The requirements gather phase consisted of the developer researching separate, already established applications to perform find functionalities that a developer can implement.

Websites such as Notion, Google Keep, etc. were researched to find the functional, and non-functional requirements for the application, as well as evaluating the advantages and disadvantages that could be seen within each application.

Interviews and surveys were also conducted to see which features should be prioritised during the development phase.

Personas were then created to simulate a user with different needs using the application.

The design of each application was also researched, to find suitable designs for the project to be developed.

Feasibility of the application was then discussed, looking at the most important pieces of functionality that should be prioritised during development.

Once this stage was completed, the design of the application was next to be worked on.

7.2.3 Design

This phase of the project consisted of two parts, which included the program design and the user interface design.

The program design discussed the different technologies that the application would be using. The application uses the MERN Stack, which consisted of Mongo, Express, React and Node, as well as various libraries such as TesseractOCR.js and Quill.js for example that was used throughout the application.

The different design patterns were also discussed, with the MVC design pattern being one of them.

Once this was done, the architecture of the application was covered, and the design for the backend of the application was shown, including the database design and the API Design.

The second part of this phase was the interface design, which was the development of the UI for the application. The project was designed in Figma and was completed in various steps, such as creating paper prototypes, wireframes, user-flow diagrams, a style guide, and a finalized design.

Once this phase was completed, implementation of the full project began.

7.2.4 Implementation

The implementation phase of the application showed the process of development, and how the entire project was put together. The implementation phase consisted of 7 Sprints and uses the SCRUM Methodology to develop the project.

Each sprint consisted of various items that were to be completed during that sprint to allow for manageable time-management and give the developer a clear goal to work towards during that sprint.

7.2.5 Testing

The testing phase of the project was to ensure that the quality of application was of good quality. Testing consisted of both functional, and user testing and covered the various parts of the application, including the navigation, the CRUD functionalities and UI, to ensure that everything was working as expected.

User testing was then performed to find flaws that the developers might not have known about and to ensure that it is user-friendly and provides great usability.

7.3 Teamwork

7.3.1 Roles

The roles of this project were between the supervisor and the student.

Clemente Gonzales provided the role of a full-stack developer, creating the project and accomplishing the different goals that were seen throughout the various sprints.

Cyril Connolly oversaw the project, and as a supervisor, acted as the role of the SCRUM Master, ensuring that the weekly sprints were completed and that enough work was being done, and provided by the developer, as well as guidance throughout the entirety of the development of the project.

7.3.2 Communication

The communication between the student and the supervisor were very consistent. Weekly meetings began in around mid-October and were kept consistent throughout the following months to ensure that the project was progressing well.

Emails and Teams messages were sent if any issues occurred, or if an urgent question was needed to be asked.

This allowed progress to be shown throughout the weeks and ensure that the student was on the right track throughout the duration of the project.

7.3.3 Difficulties

During the development of the application, difficulties were found throughout various phases of development. Some of which included bug fixes, where the developer would spend several days attempting to fix one bug and losing time and progress because of it.

7.3.4 Resolving Difficulties

To ensure that work was still being done throughout the many weeks of development, focusing on different parts of the application was done whenever a prevalent bug occurred. For example, if an issue occurred and was not solved within the span of a few hours, different parts of the application would be worked on to ensure that work was still accomplished that day and return to the problem on a different day.

7.4 SCRUM Methodology

The SCRUM Methodology proved to be an effective method of developing a large-scale project such as this one. It ensured that the weekly sprints were completed and proved to be a highly effective method of encouraging productivity when needed.

The goals being divided into the 7 sprints worked very well and ensured that enough time was given to the development of different parts of the application.

As the requirements were gathered at the beginning of the project, the developers knew what needed to be accomplished to produce the final product and gave them clear goals to work towards throughout each sprint.

The project backlog showed the different functionalities that needed to be accomplished throughout the duration of the project and was updated to show which pieces of functionality were completed and which still need to be implemented.

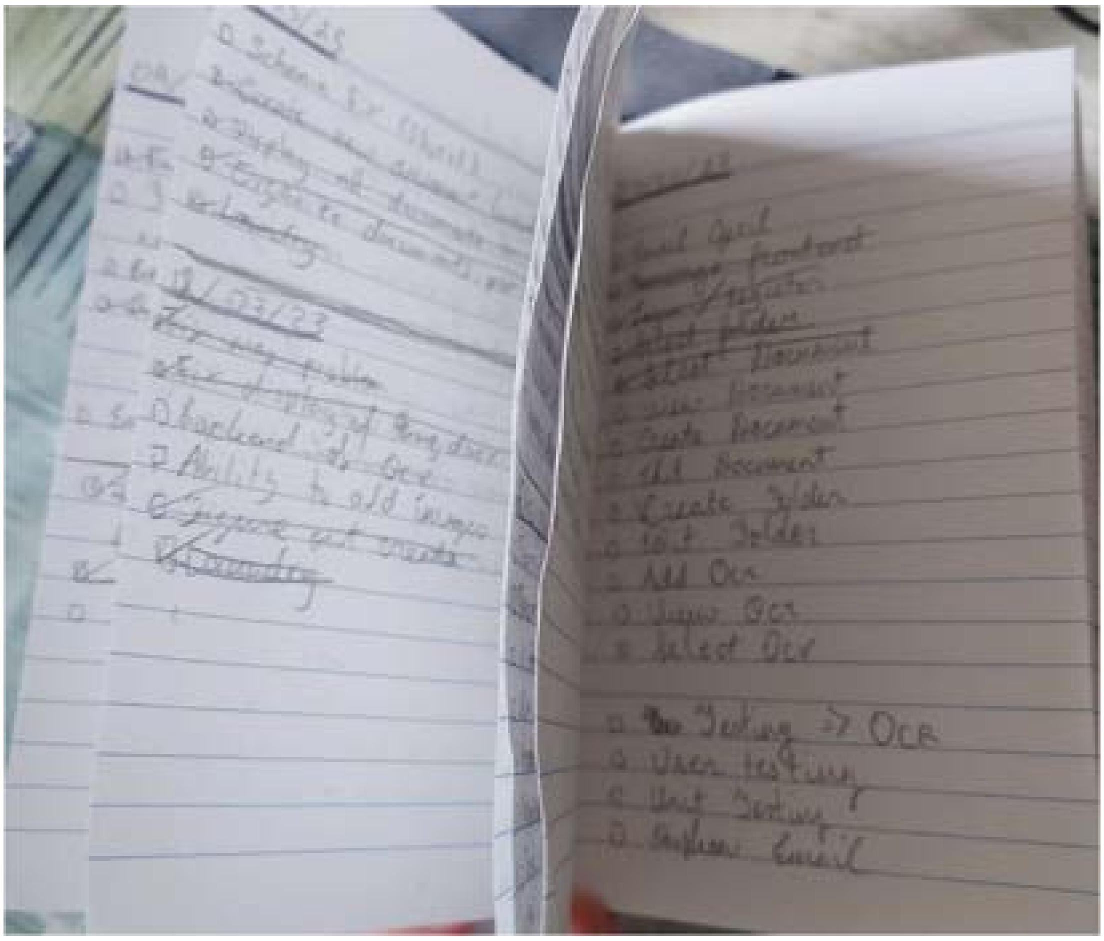
7.5 Project Management Tools

7.5.1 Pen and Paper Notes

Although there are various applications that can be used to manage parts of a project, pen and paper checklists was used for this application.

The reason behind this was because pen and paper allowed me, the developer to consistently think of different tasks that needed to be completed on that day. When a project management app such as Trello is being used, the updates towards that app and the tasks that needed to be completed tend to decline the further into the project, and because of this pen and paper was instead used as there was no need to share the tasks, as it was a solo project, and the tasks were discussed weekly with the supervisor.

The to-do list was dated to show when each task was being completed and can be seen in figure 238.



☐ Schema [illegible]
☐ Display of documents
☐ [illegible] documents
~~[illegible]~~
[illegible]/03/23
~~[illegible]~~
~~[illegible]~~
☐ Backend to OCR
☐ Ability to add images
~~[illegible]~~

~~Build Grid~~
~~[illegible] frontend~~
~~[illegible]~~
~~Select folder~~
~~Edit Document~~
☐ View Document
☐ Create Document
☐ Add Document
☐ Create Folder
☐ Edit Folder
☐ Add OCR
☐ View OCR
☐ Select OCR

☐ Testing -> OCR
☐ User testing
☐ Unit Testing
☐ [illegible] Email

Figure 245 Pen and Paper To-Do lists

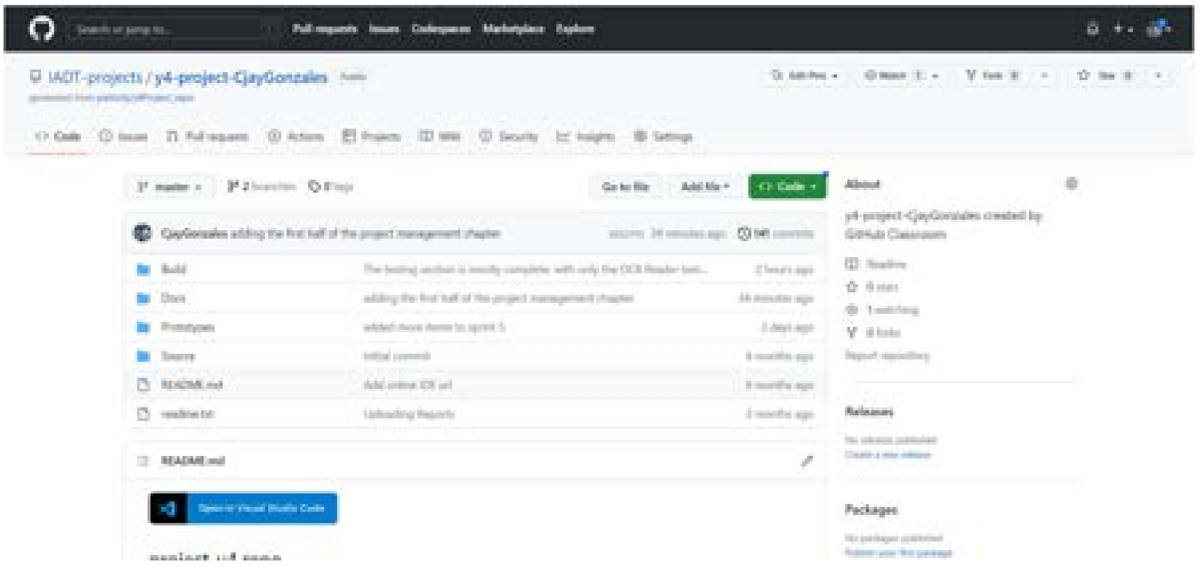
7.5.2 GitHub

GitHub is an online tool that allows developers to store their code, as well as allow for version control. GitHub and Git are essential in the development of the application, as its allowance for version control proved to be useful when looking back over previous versions of the code. GitHub also shows the progress that was undertaken throughout development and shows the work done throughout the various sprints.

GitHub was an easy way to store the code in case the developer needed to return to a previous version or act as a method to back up their code, in case it becomes corrupt, or progress gets lost.

GitHub also kept the changes that the developer made throughout the development of the application, and the different version of commits can be accessed and viewed if needed.

A screenshot of GitHub can be seen below in figure 239:



CjayGonzales adding the first half of the project management chapter

Build	The testing section is mostly complete, with only the OCR Reader test...	2 hours ago
Docs	adding the first half of the project management chapter	34 minutes ago
Prototypes	added more items to sprint 5	3 days ago
Source	initial commit	[illegible] months ago
README.md	Add online IDE url	[illegible] months ago
readme.txt	Uploading Reports	2 months ago

y4-project-CjayGonzales created by GitHub Classroom

Figure 246 GitHub Repository

7.5.3 Journal

A journal was not kept during the development of the application, but the pen and paper notes / to-do list allowed the developer to keep in track of the different goals that were completed in each sprint, and tasks that needed to be accomplished throughout the day. The tasks that were incomplete during that day were moved on to the next.

Problems and difficulties were also recorded in Notion, to ensure that the problems were being listed down if any of them occurred.

A journal itself was not kept, although there were many parts of the application that listed the progress and problems whenever they occurred.

7.6 Reflection

7.6.1 Your views on the project

From my personal perspective, the project turned out to be quite successful, achieving the main functionality and goals that were set out from the beginning of the project. The implementation of the OCR Reader and tweaking parts of it to test out the functionality and accuracy was incredibly rewarding to work with as it was something different from the regular projects that are generally worked on. Even though it was a website, there were many aspects to it that changed it so that it provided a different experience from the usual projects.

The application that was produced however could have done with more polishing as it contained a variety of bugs that were not solved due to the time constraints. Some features that could have greatly benefited the application were not implemented and could have been great features to have included.

Despite these flaws, the project was successful and accomplished its goal. The overall quality of the project created was of an overall high standard with completed functionality.

7.6.2 Completing a large software development project

The completion of a large software development project worked on by a single individual gave a great insight into the lifestyle of a full-stack developer, working on the own or on a team. This was beneficial in showing the many steps of developing a completed product.

The various sprints that occurred throughout the project also provided a great general guideline in what should be accomplished within those sprints, and the functionalities that were to be focused on for that sprint.

Overall, it was a crucial learning experience and showed me how to progress through problems whenever they are encountered, as well as communicating with my supervisor to show the progress that was completed though out the various weeks of development and implementation.

7.6.3 Working with a supervisor

Working with a supervisor was very helpful. Cyril consistently provided great feedback throughout the weekly meetings and would provide guidance on what to focus on during that phase of the project, and any extras that can be added. He provided constructive criticism during the writeup of the application and provided some great ideas to help the project stand out and be more in line with what needed to be developed.

Communication overall was very good. Emails and Microsoft Teams messages were regularly sent, as well as updates regarding the progress of the project.

Overall, Cyril provided great help and guidance as a supervisor and guided me towards the right direction throughout the duration of the project.

7.6.4 Technical skills

This project managed to improve some parts of my technical skills. The OCR part of the application was something very different and managed to improve my competency in building projects such as this one. The project managed to expand my skillsets. Even though it was still in a field that I was comfortable with, there were parts of the project that were very different from what I would normally work on and was a very valuable learning experience.

The project also managed to develop my web development skills, as well as my design skills through the duration of the project.

Overall, the most valuable part of this project was learning the new technologies and how to incorporate and implement them with the React framework.

7.6.5 Further competencies and skills

Learning more complex languages / coding practices is a goal that should be worked towards. This can include developing a project in a completely new language to learn how something works or creating projects and seeing how to implement specific features.

This can help improve my flexibility as a developer and help expand my skillsets into doing more things that’s not just web-based applications.

Another skill that should be learned is learning more effective ways to problem solve, as well as writing more efficient code, rather than using the quickest methods to solve a problem, although this will mostly be a skill that will be acquired over time.

7.7 Conclusion

In conclusion, the project that was developed was a web-based notes taking application that contained OCR functionalities.

The chapter discussed how the project was managed, explaining the many phases of the project that occurred during implementation, which discussed what was done in each sprint of development.

The chapter also discussed how teamwork was done, with the constant communication with the supervisor, and explaining how difficulties were encountered and resolved during development.

The project management tools that were used to assist development were also shown during this chapter, and explains how the different tasks were completed, and the methods for listing down those various tasks.

A reflection of the project was then discussed, allowing me to give my insights and parts of the project that could have been done better and improved upon.

8 Conclusion

The final product that was built consisted of a web-application that allowed users to take down notes and use OCR software to allow text to be extracted from outside sources. The overall goal of the project was to build a web application that contained functionalities that were different from what the developer was comfortable with. The inclusion of the OCR functionality was different and allowed me to expand my knowledge as a developer.

The technologies seen in the application consisted of various libraries. This included TesseractOCR.js, Quill.js, the MERN Stack, which were MongoDB, Express.js, React.js, and Node.js, and various other packages for different functionalities. AWS Buckets were also used for storing the images that were uploaded to the application, and Vercel and Firebase were used to host the application.

Research was conducted to find the various features that the application has. This ranged from conducting surveys and interviews, as well as researching various other applications to find the main functionalities that those applications had.

Additionally, research into how OCR worked was also done, which provided a better insight into how the system extracts text, through the various methods of text extraction.

The design was then formed, which included the software design and the User Interface Design. The software design section discussed the design patterns seen using the MERN Stack, the database design, the API Design, the application architecture, and the process design. While the User Interface Design provided the wireframes, the User Flow Diagram, the Style Guide, and showed the final design of the application.

Implementation then showed the steps taken to develop the application. The SCRUM Methodology was used to develop the application, which divided the whole application into achievable goals that ran through two-week sprints over the course of several sprints. Various functionalities were developed during the sprints to create the final product.

Testing was then conducted to uncover underlying flaws that were found throughout the application and allow these flaws and issues to be fixed. This was completed through the means of functional testing, and user testing, both of which covered different parts of the application to find flaws within them.

The project management chapter discussed the various methods that was used to assist the development of the application, which discussed the roles provided, difficulties and how these difficulties were resolved, as well as showing the tools that assisted development, such as GitHub, and the notes.

The result provided a professional looking web application that fulfilled the goals and functionalities that were set during the start of development. The overall application was polished with minimal bugs, and accomplished the functionalities that were set at the start of development.

The project could be further developed by adding extra functionalities, such as the to-do list, sharing notes, and improving the accuracy of the OCR Reader, potentially adding an options modal to allow users to adjust the OCR Reader to increase its accuracy.

React Native and media queries should also be included if the application were to be developed further, as well as implementing the criticisms found during user testing.

The application could potentially run ads or contain a premium version to allow users to use some functionalities as a business opportunity, although the application should stay mostly free to use.

Many new skills were learned and refined during the development of this application. The ability to problem solving was focused on. During development, a variety of bugs and errors would often occur, and a common problem that would happen is spending too much time finding a solution and solving that issue. This often led to only small amounts of work being accomplished that day, and the fix for this was to simply return to the issue after implementing other parts of the application, leading to more being accomplished throughout development.

The project also allowed me to learn new technologies, that of which I was not very familiar with, for example TesseractOCR.js was something very different from what is normally done and gaining experience in developing something new was a very rewarding experience.

Coding skills were also refined during development, as there were also parts of the project that I was already familiar with, for example the use of MongoDB and Express.js, as well as the use of the SCRUM Methodology and regularly communication with my supervisor.

8.1 Final Words

Throughout the development of this project, I managed to learn and develop many new skills which can help prepare me for the workplace. Time management and problem solving were crucial and were given a chance to develop during the duration of this project.

My ability to code was also expanded upon, as there were many instances during development where I needed to develop something that I was not too familiar with, which helped expand my capabilities as a developer.

Overall, this project provided an excellent challenge and produced a worthwhile product. Although there were still flaws within it, it helped expand my abilities as a developer, and I hope to continue to keep polishing my abilities and keep learning new technologies and concepts as a developer.

9 References

basarat. (2022, 7 13). *Image to Text Conversion with JavaScript // OCR (Optical Character Recognition)*. Retrieved from YouTube: https://www.youtube.com/watch?v=zBET2cEKths&ab_channel=basarat

Bhaskar, N. U. (2011). General principles of user interface design and websites. *International Journal of Software Engineering (IJSE),*, 2(3), 45-60.

Chan, M. (2020, 11 12). *Using JavaScript to Preprocess Images for OCR*. Retrieved from DEV Community: https://dev.to/mathewthe2/using-javascript-to-preprocess-images-for-ocr-1jc

David, H. (2022, 6 27). *Hubspot*. Retrieved from What is React.js? (Uses, Examples, & More): https://blog.hubspot.com/website/react-js#:~:text=The%20React.,you%20would%20with%20vanilla%20JavaScript.

Eikvil, L. (1993). Optical character recognition. *citeseer. ist.* , psu. edu/142042. html, 26.

Express - Node.js web application framework. (2017). Retrieved from Expressjs.com: https://expressjs.com/

Ferguson, K. (2021). *Application architecture*. Retrieved from TechTarget: application architecture

Fleming, J. &. (1998). Web navigation: designing the user experience. *Sebastopol, CA: O'reilly.* , p. 166.

Gardner, B. S. (2011). Responsive web design: Enriching the user experience. *Sigma Journal: Inside the Digital Ecosystem*, 11(1), 13-19.

Getting Started – Amazon Simple Storage Service (S3) – AWS. (2023). Retrieved from Amazon Web Services, Inc.: https://aws.amazon.com/s3/getting-started/#:~:text=Amazon%20Simple%20Storage%20Service%20(Amazon,at%20any%20time%2C%20from%20anywhere.

Goel, V. K. (2019). Text extraction from natural scene images using OpenCV and CNN. *Int. J. Inf. Technol. Comput. Sci*, 11(9), 48-54.

Introduction | Socket.IO. (2023, 3 7). Retrieved from Socket.io: https://socket.io/docs/v4/

Introduction to Mongoose for MongoDB. (2018, 2 11). Retrieved from freeCodeCamp.org: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

Johnson, J. (2020). Designing with the mind in mind: simple guide to understanding user interface design guidelines. *Morgan Kaufmann*.

Karnik, N. (2018, 2 11). *Introduction to Mongoose for MongoDB*. Retrieved from freeCodeCamp.org: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

Kumar, S. (2022, June 29). *10 most popular design systems to learn from in 2022 for UX Designers*. Retrieved from Medium; UX Planet: https://uxplanet.org/10-most-popular-design-systems-to-learn-from-in-2022-for-ux-designers-18a24843a860

Lehal, G. S. (1999). Feature extraction and classification for OCR of Gurmukhi script. *VIVEK-BOMBAY-*, 12(2), 2-12.

Lopez, M. M. (2017). Deep Learning applied to NLP. *arXiv preprint*, arXiv:1703.03091.

Ma, D. L. (2000). Mobile camera based text detection and translation. *Department of Electrical Engg Department Stanford University*.

Matei, O. P. (2013). Optical character recognition in real environments using neural networks and k-nearest neighbor. *Applied intelligence*, 39(4), 739-748.

Mithe, R. I. (2013). Optical character recognition. *International journal of recent technology and engineering (IJRTE)*, 2(1), 72-75.

multer-s3. (2022, 5 31). Retrieved from npm: https://www.npmjs.com/package/multer-s3

Node.js. (2023). *About | Node.js*. Retrieved from Node.js: https://nodejs.org/en/about/

Ong, V. &. (2016). Using k-nearest neighbor in optical character recognition. *ComTech: Computer, Mathematics and Engineering Applications*, 7(1), 53-65.

p5. (2023, 2 22). Retrieved from npm: https://www.npmjs.com/package/p5

Patel, C. P. (2012). Optical character recognition by open source OCR tool tesseract: A case study. *International Journal of Computer Applications*, 55(10), 50-56.

Quill - Your powerful rich text editor. (2023). Retrieved from Quill: https://quilljs.com/#:~:text=Rich%20Text%20Editor-,Quill%20is%20a%20free%2C%20open%20source%20WYSIWYG%20editor%20built%20for,Settings

Ranjan, A. B. ((2021)). OCR Using Computer Vision and Machine Learning. In Machine Learning Algorithms for Industrial Applications. *Springer, Cham*, (pp. 83-105).

Rao, N. V. (2016). OPTICAL CHARACTER RECOGNITION TECHNIQUE ALGORITHMS. . *Journal of Theoretical & Applied Information Technology*, 83(2).

Simplified, W. D. (2021, 4 20). *How To Build A Google Docs Clone With React, Socket.io, and MongoDB*. Retrieved from YouTube: https://www.youtube.com/watch?v=iRaelG7v0OU&ab_channel=WebDevSimplified

Singh, A. B. (2012). A survey of OCR applications. *International Journal of Machine Learning and Computing*, 2(3), 314.

Smith, R. (2007, September). An overview of the Tesseract OCR engine. *In Ninth international conference on document analysis and recognition (ICDAR 2007)*, (Vol. 2, pp. 629-633). IEEE.

Stone, D. J. (2005). User interface design and evaluation. *Elsevier*.

Tesseract.js | Pure Javascript OCR for 100 Languages! (2023). Retrieved from Projectnaptha.com: https://tesseract.projectnaptha.com/

Verma, R. &. (2012). A-survey of feature extraction and classification techniques in OCR systems. *International Journal of Computer Applications & Information Technology*, 1(3), 1-3.

Visual Studio Code. (2021, 11 3). Retrieved from Visualstudio.com: https://code.visualstudio.com/docs/editor/whyvscode

Wang, T. W. ((2012, November)). End-to-end text recognition with convolutional neural networks. In Proceedings of the 21st international conference on pattern recognition (ICPR2012). (pp. 3304-3308). IEEE.

What Is GitHub? A Beginner's Introduction to GitHub. (2022, 12 13). Retrieved from Kinsta®: https://kinsta.com/knowledgebase/what-is-github/

What Is MongoDB? (2022). Retrieved from MongoDB: https://www.mongodb.com/what-is-mongodb

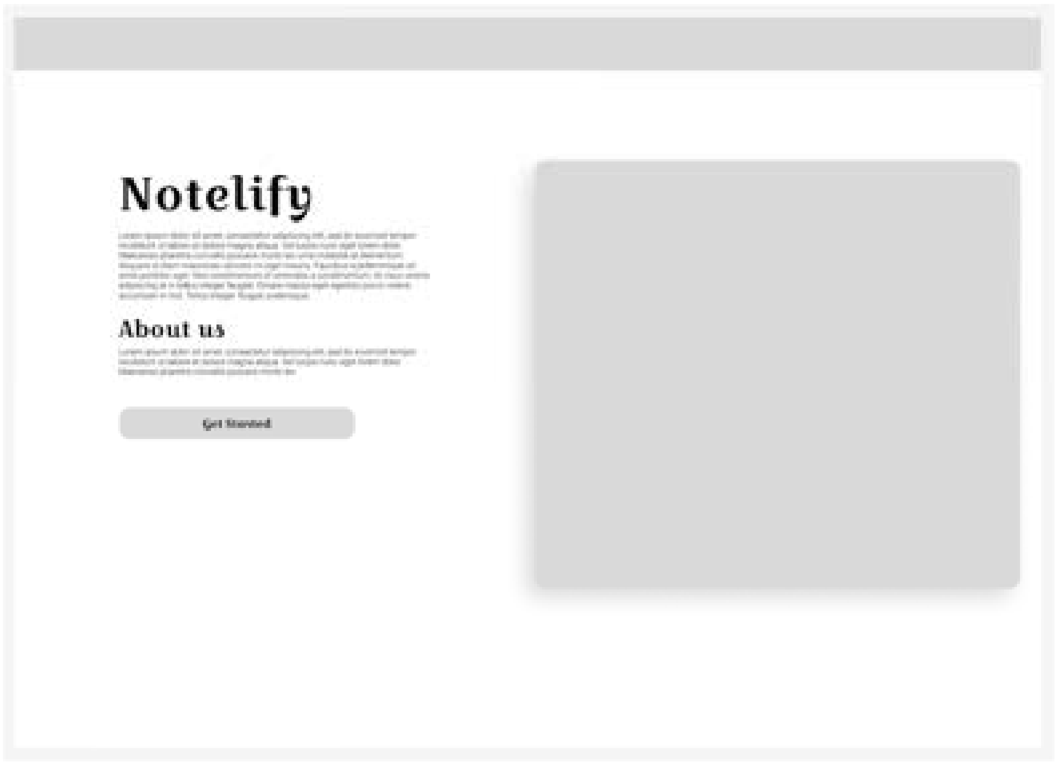
What Is Scrum Methodology? & Scrum Project Management. (2022, 12 23). Retrieved from Nimblework: What Is Scrum Methodology? & Scrum Project Management

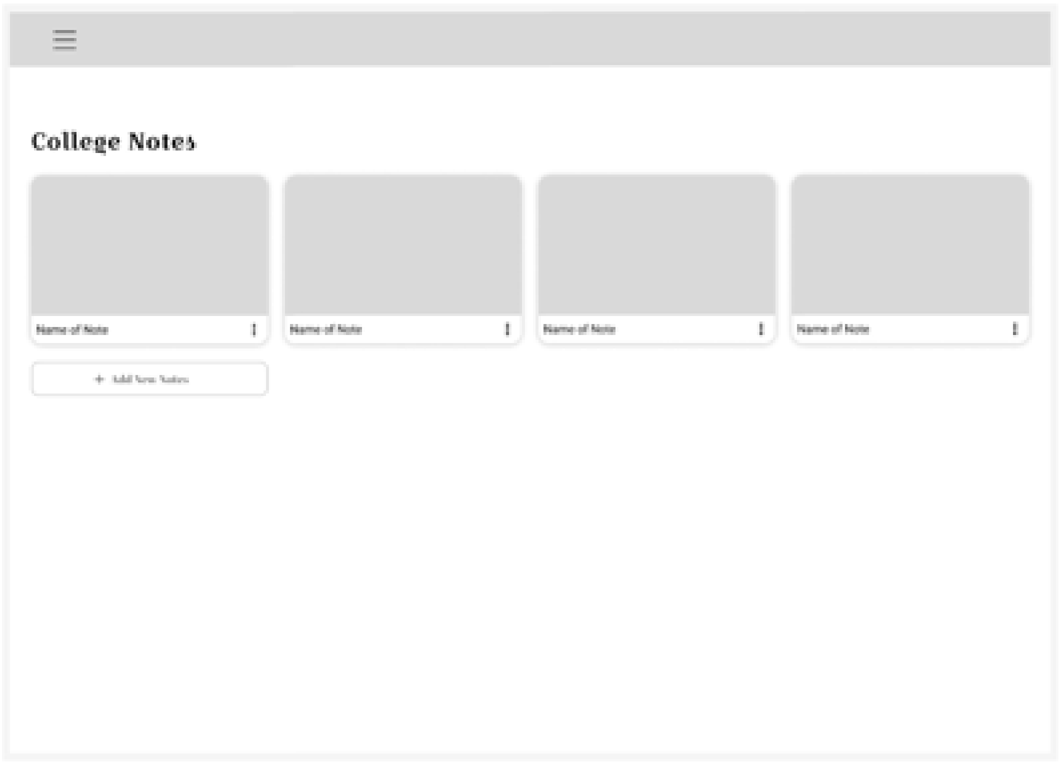
10 Appendices

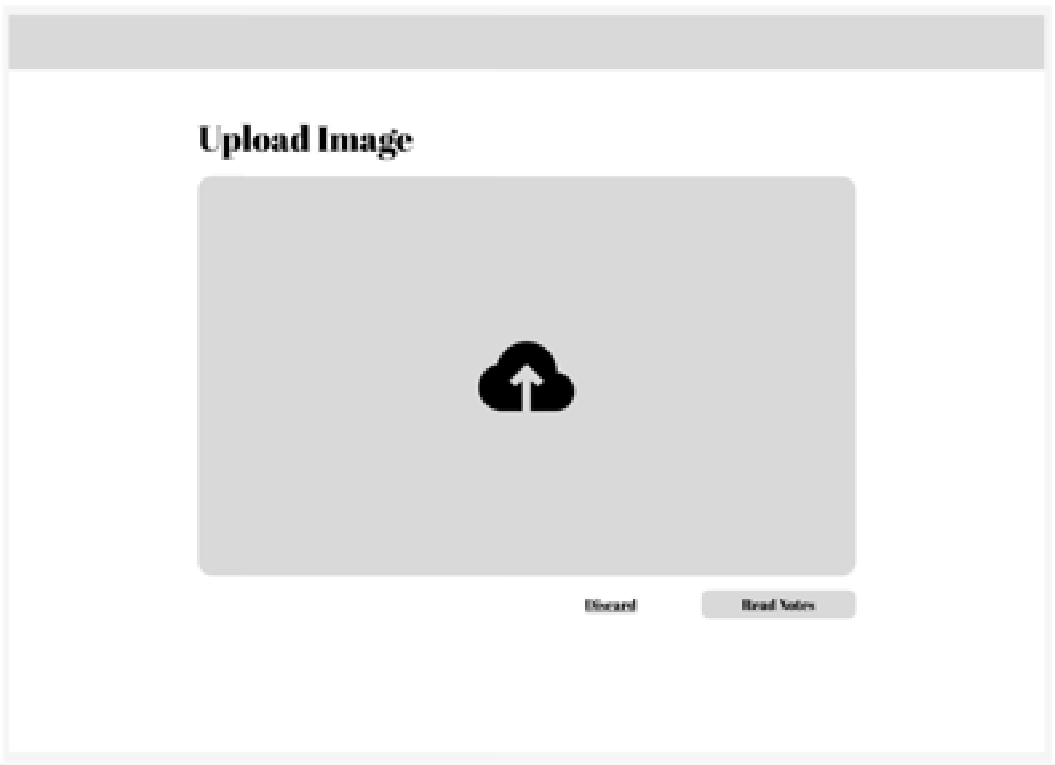
10.1 Appendix A - Wireframes

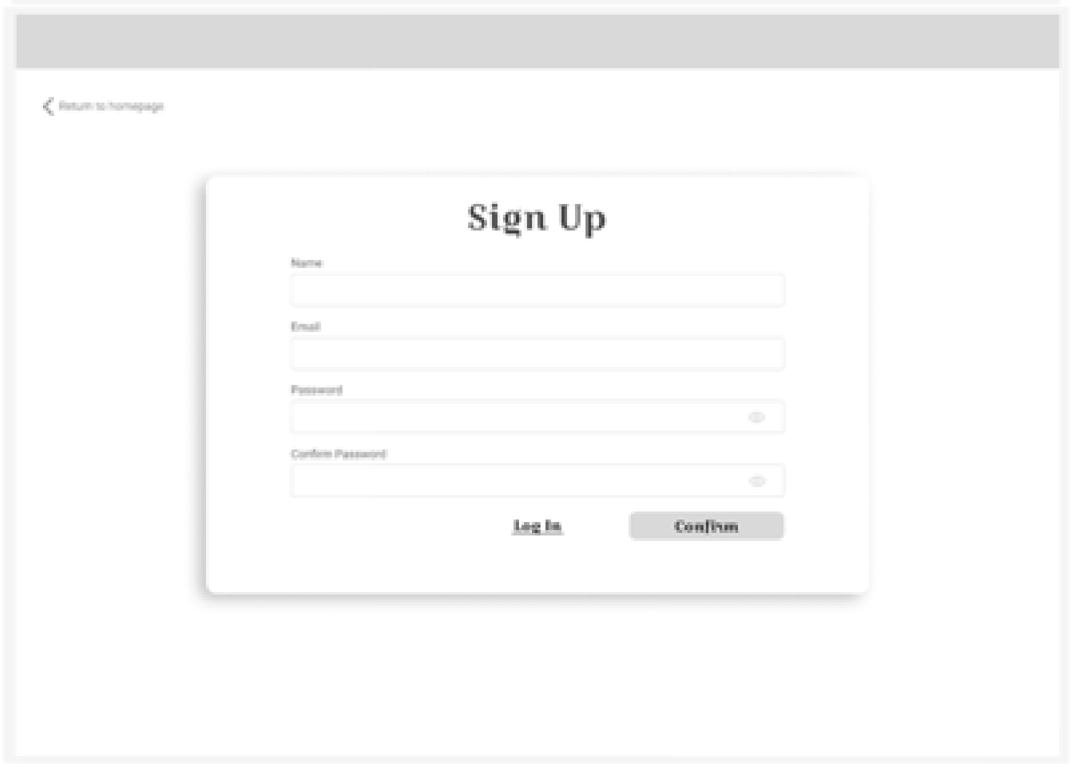
The link for the Figma File can be seen here:

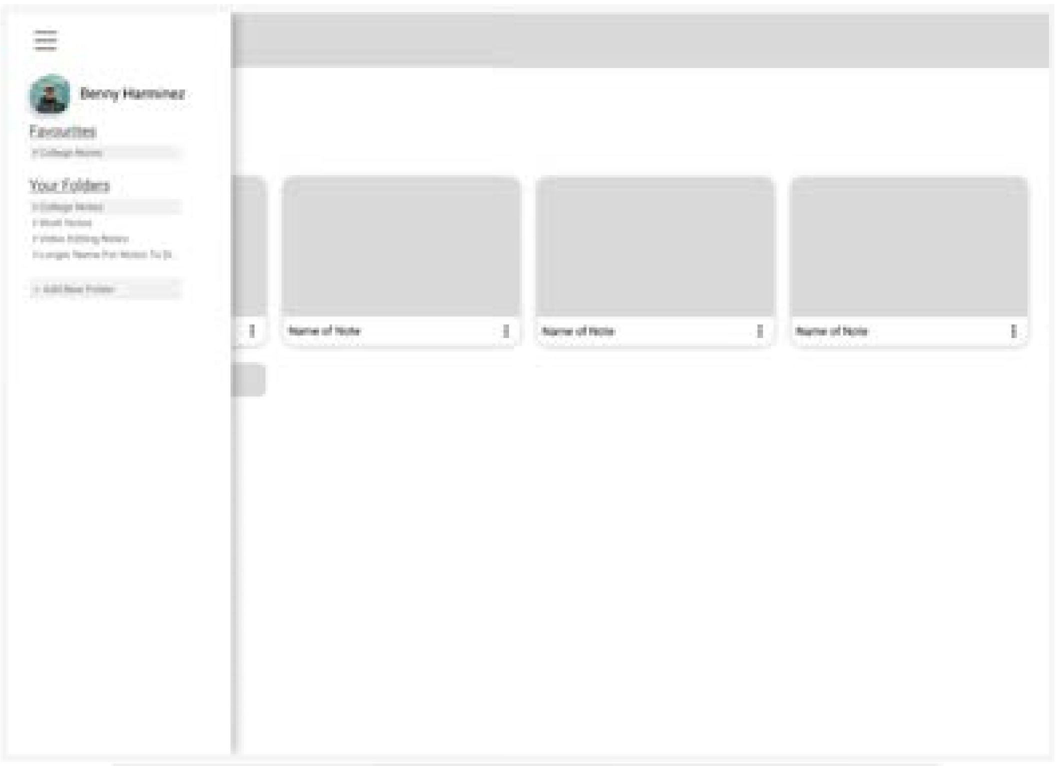
https://www.figma.com/file/8wushDYxUM3YNyoGlC2RUW/Final-Project?node-id=0%3A1&t=IChSggepWepbklIC-1

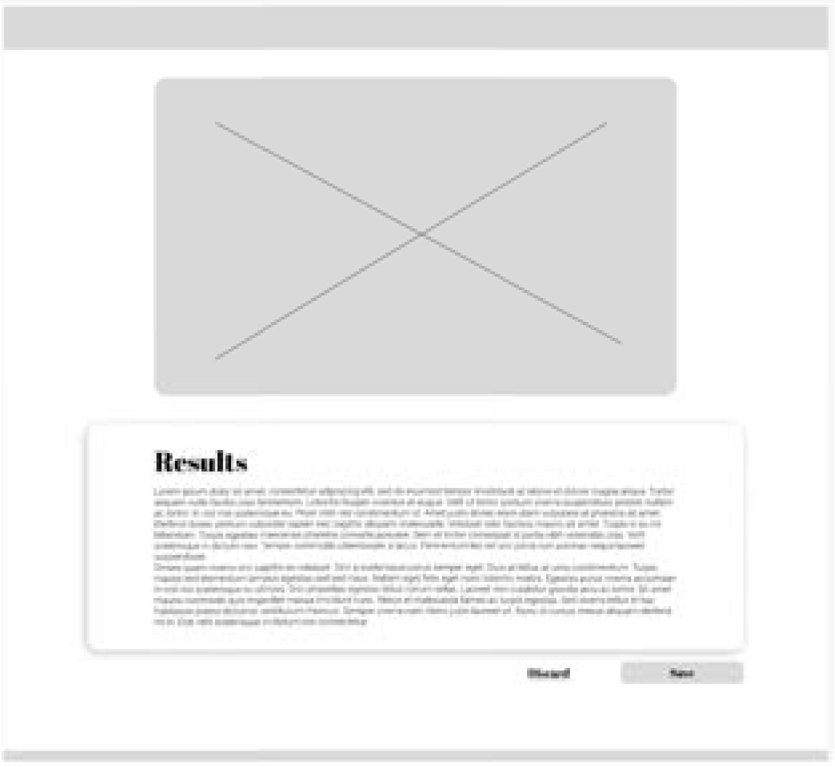




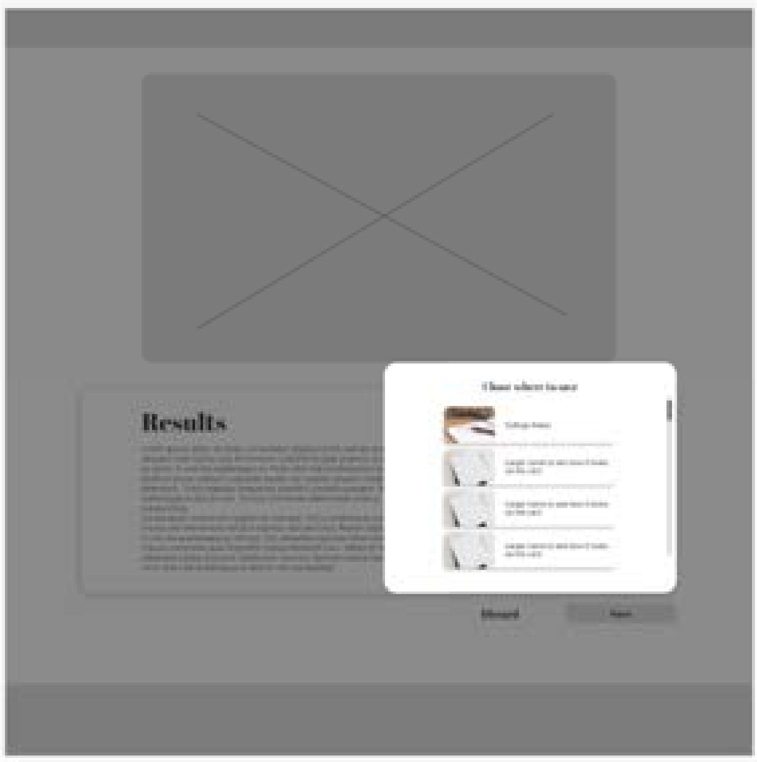

Upload Image
Discard
Read Notes

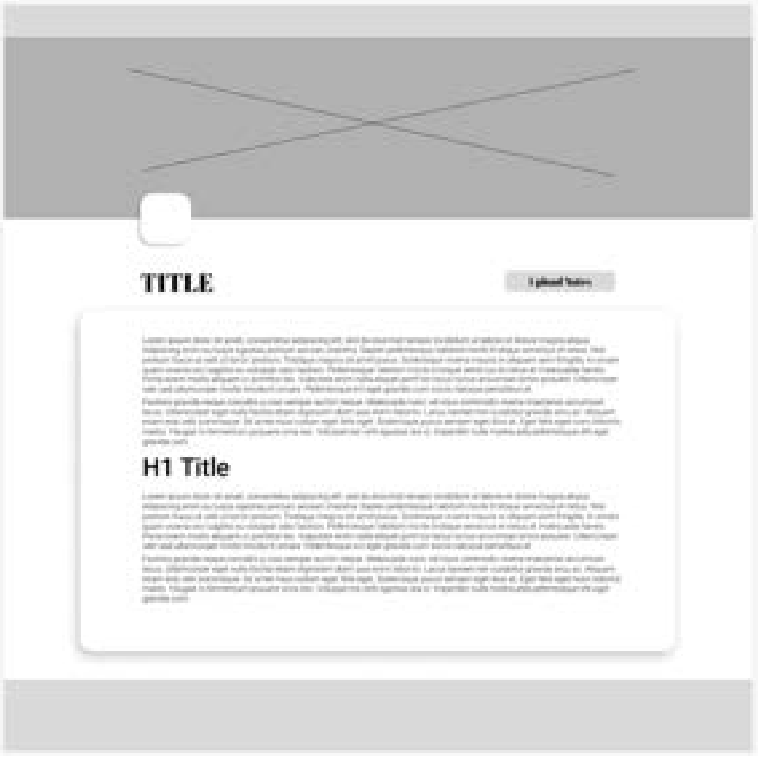

Return to homepage
Sign Up
Name
Email
Password
Confirm Password
Log In
Confirm


Benny Harminez
Favourites
College Notes
Your Folders
College Notes
Work Notes
Video Editing Notes
Longer Name For Notes To B
Add New Folder
Name of Note
Name of Note
Name of Note

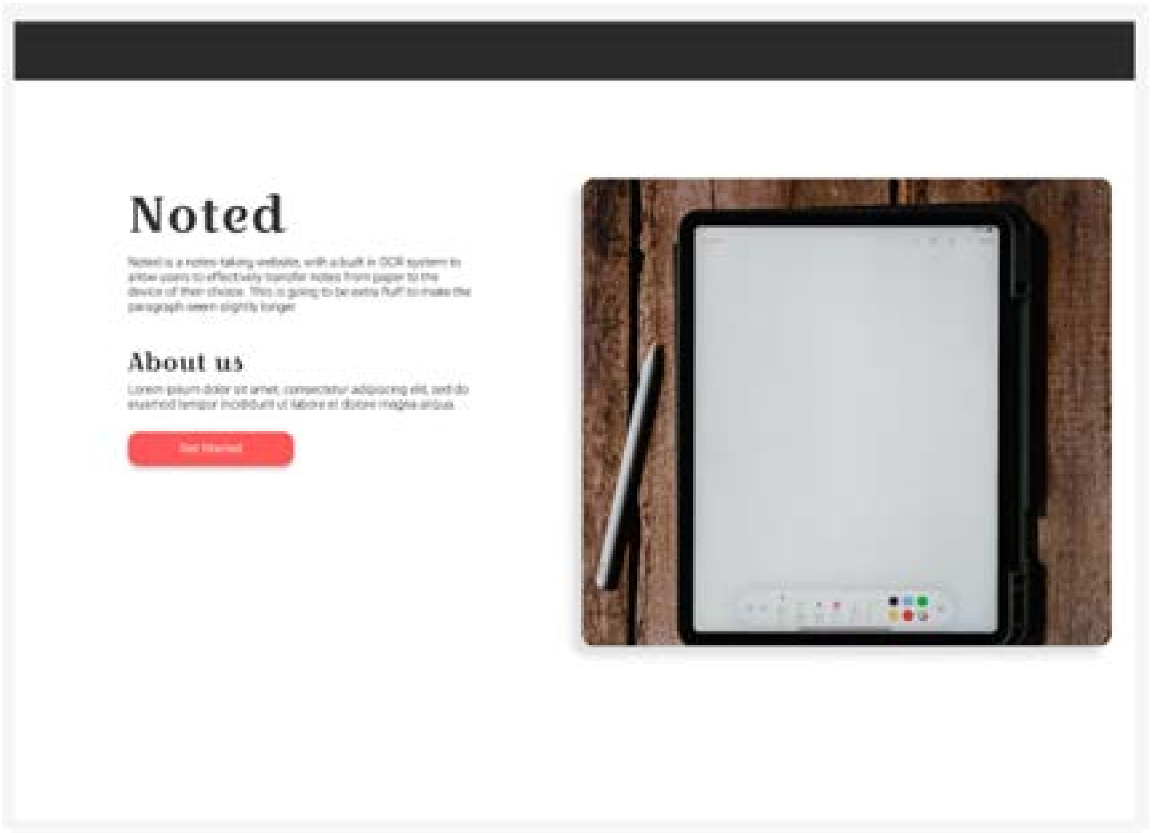

Results
Discard
Save

[illegible]


Chose where to save
Results
Discard
Save


TITLE
Upload Notes
H1 Title

10.2 Appendix B – Final Design

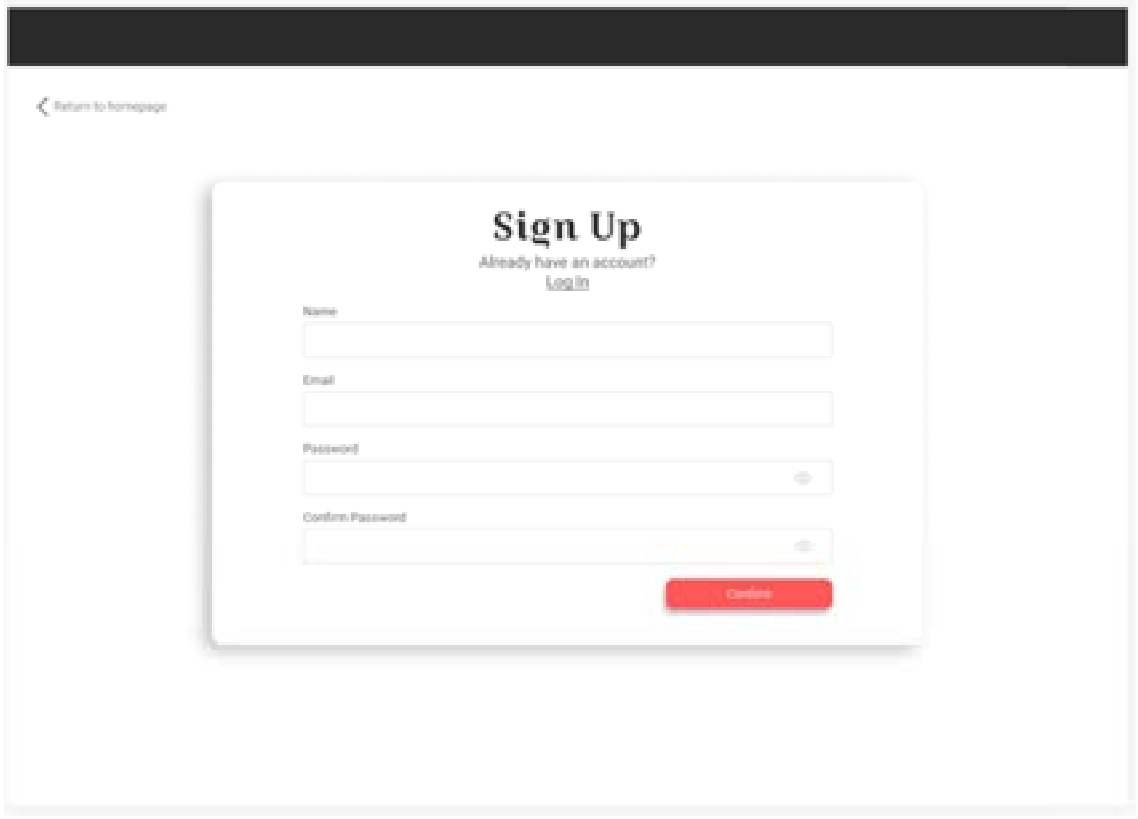


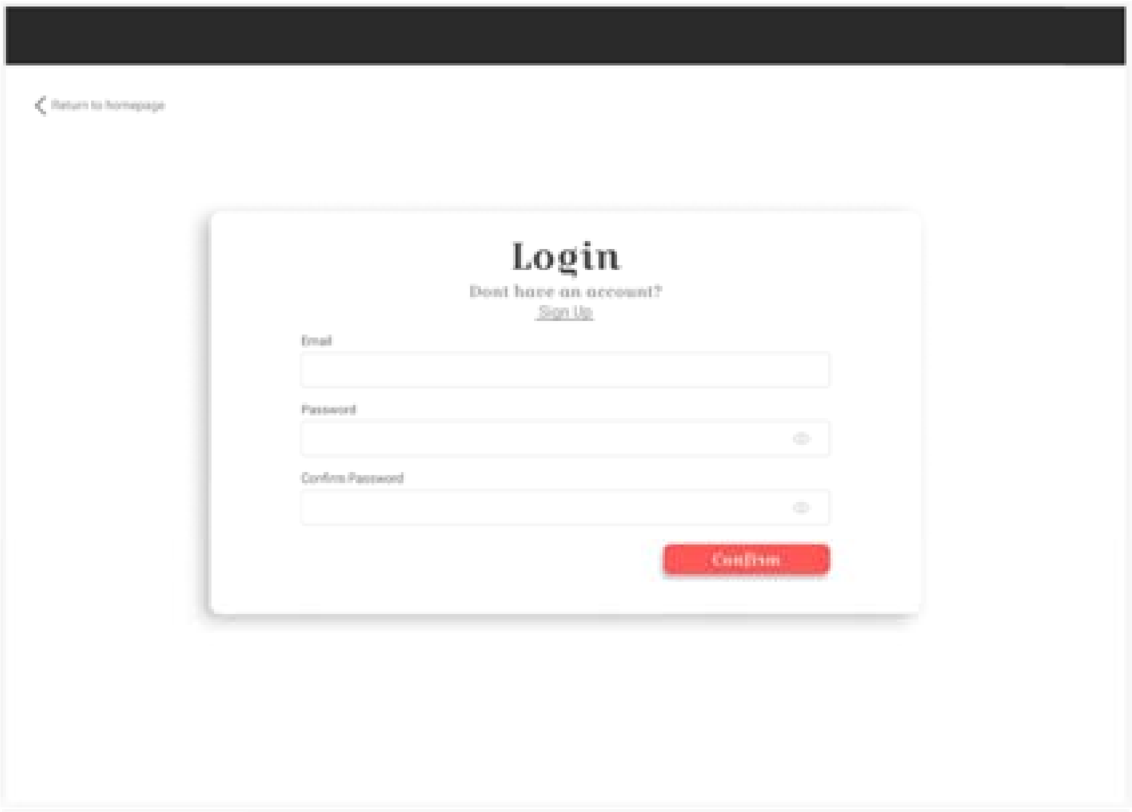
Noted

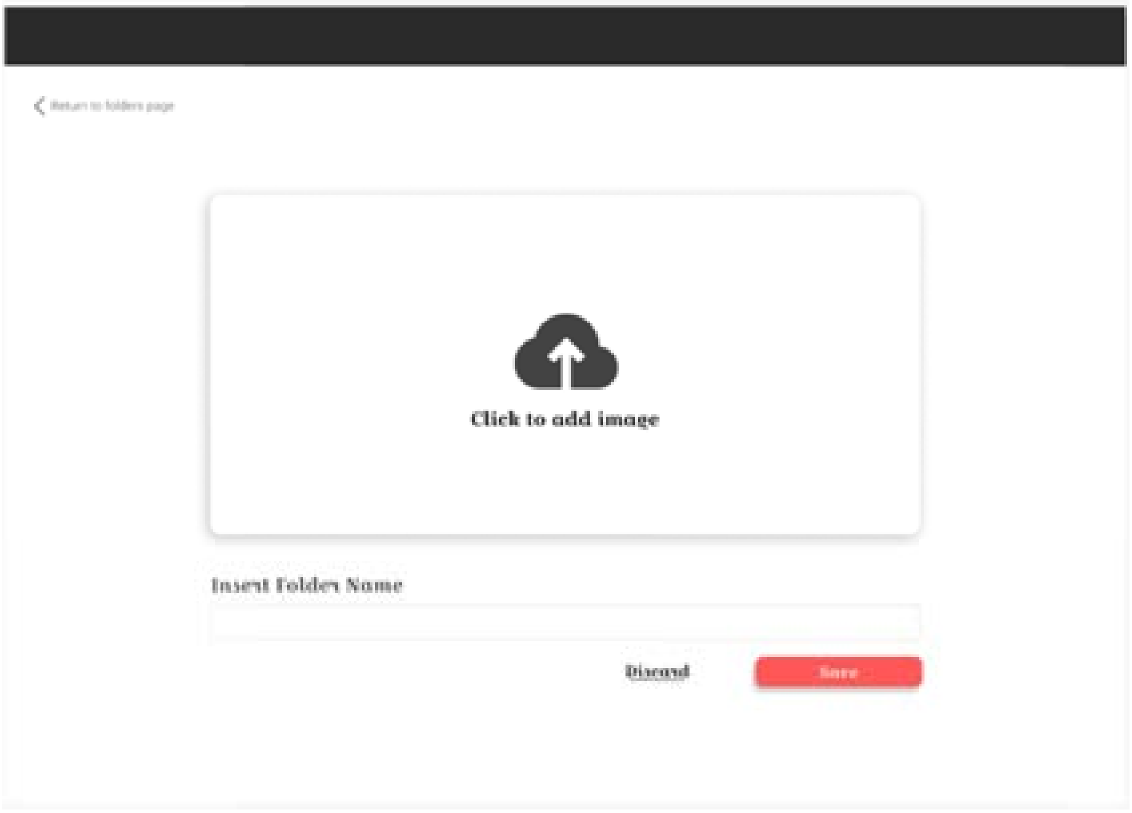
Noted is a notes-taking website, with a built in OCR system to allow users to effectively transfer notes from paper to the device of their choice. This is going to be extra fluff to make the paragraph seem slightly longer.

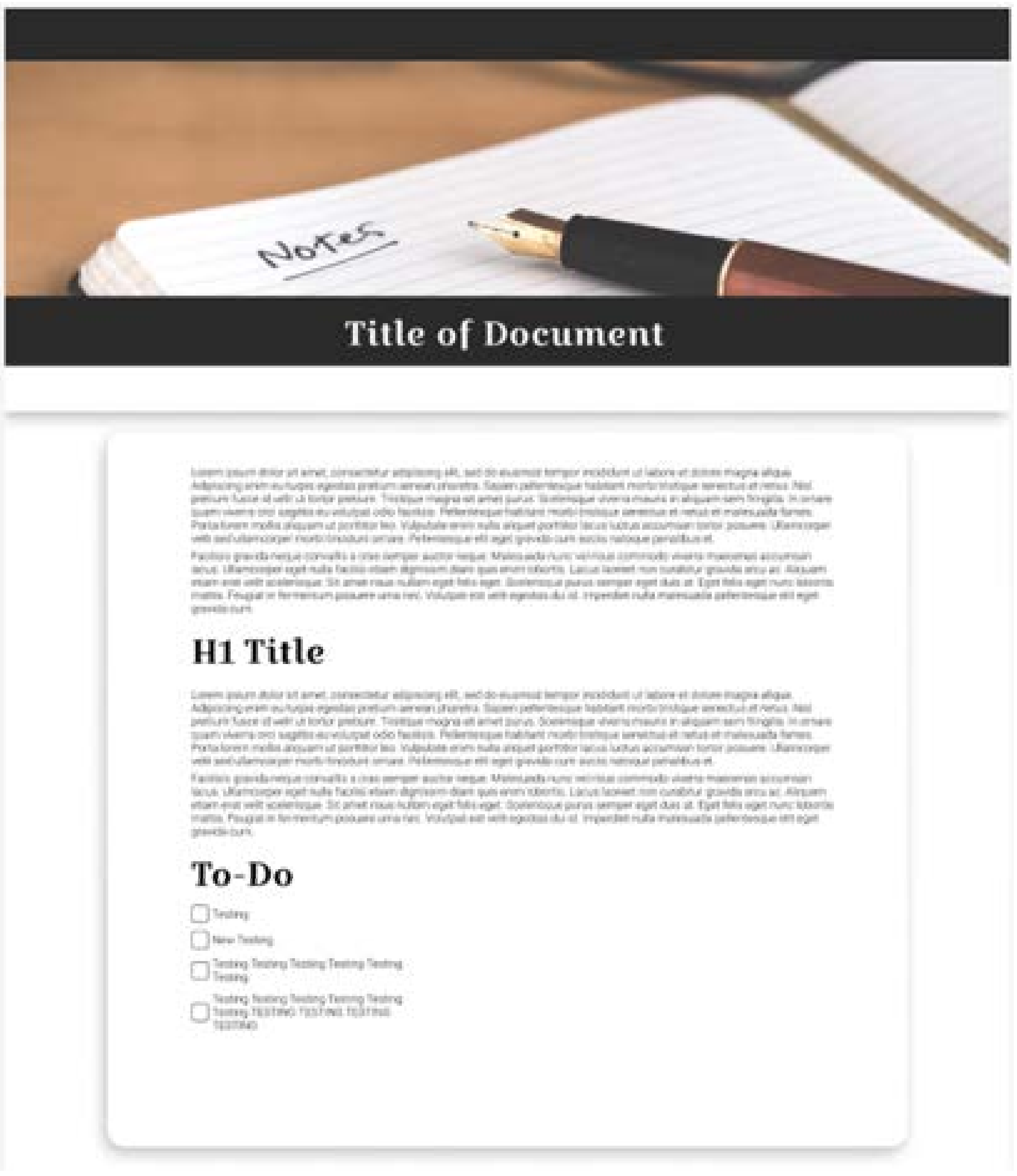
About us

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.




Return to homepage
Login
Dont have an account?
Sign Up
Email
Password
Confirm Password
Confirm


Return to folders page
Click to add image
Insert Folder Name
Discard
Save



Title of Document

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Adipiscing enim eu turpis egestas pretium aenean pharetra. Sapien pellentesque habitant morbi tristique senectus et netus. Nisi pretium fusce id velit ut tortor pretium. Tristique magna sit amet purus. Scelerisque viverra mauris in aliquam sem fringilla. In ornare quam viverra orci sagittis eu volutpat odio facilisis. Pellentesque habitant morbi tristique senectus et netus et malesuada fames. Porta lorem mollis aliquam ut porttitor leo. Vulputate enim nulla aliquet porttitor lacus luctus accumsan tortor posuere. Ullamcorper velit sed ullamcorper morbi tincidunt ornare. Pellentesque elit eget gravida cum sociis natoque penatibus et.

Facilisis gravida neque convallis a cras semper auctor neque. Malesuada nunc vel risus commodo viverra maecenas accumsan lacus. Ullamcorper eget nulla facilisi etiam dignissim diam quis enim lobortis. Lacus laoreet non curabitur gravida arcu ac. Aliquam etiam erat velit scelerisque. Sit amet risus nullam eget felis eget. Scelerisque purus semper eget duis at. Eget felis eget nunc lobortis mattis. Feugiat in fermentum posuere urna nec. Volutpat est velit egestas dui id. Imperdiet nulla malesuada pellentesque elit eget gravida cum.

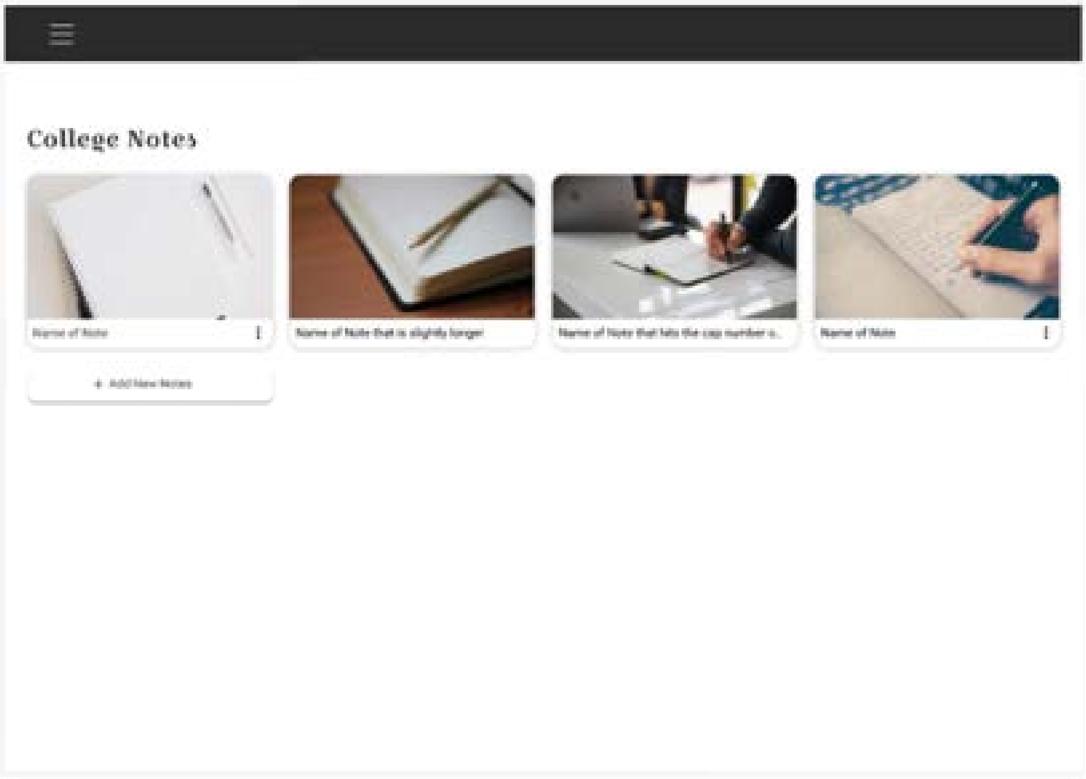
H1 Title

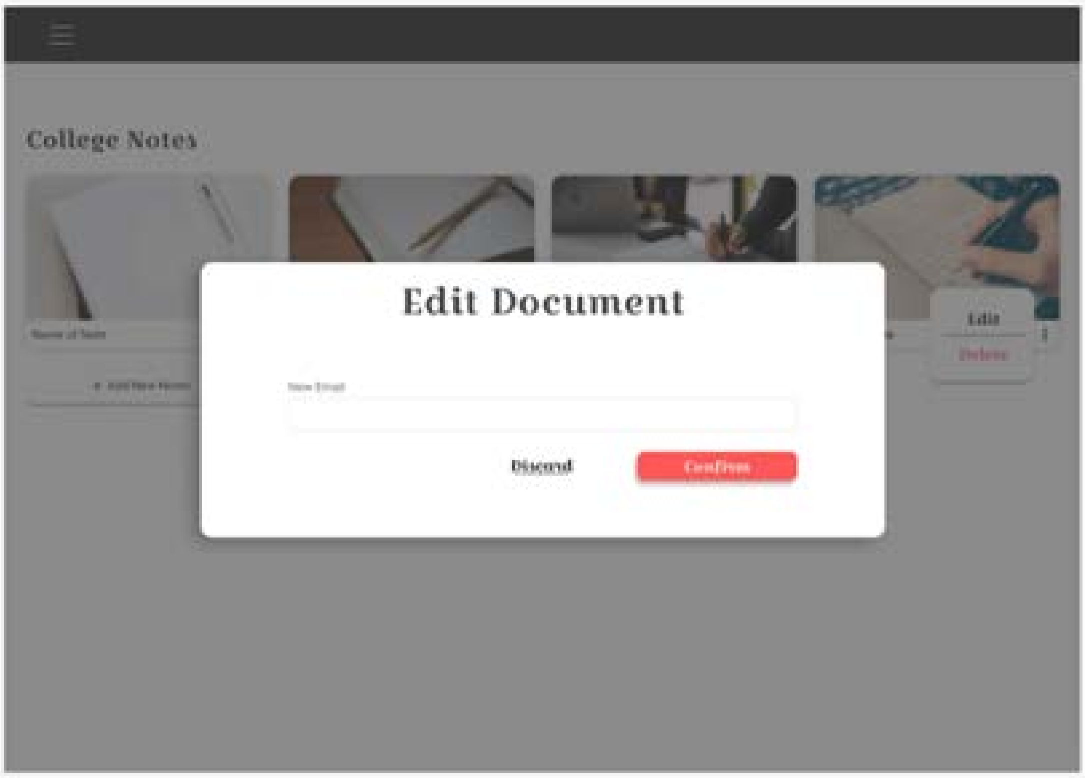
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Adipiscing enim eu turpis egestas pretium aenean pharetra. Sapien pellentesque habitant morbi tristique senectus et netus. Nisi pretium fusce id velit ut tortor pretium. Tristique magna sit amet purus. Scelerisque viverra mauris in aliquam sem fringilla. In ornare quam viverra orci sagittis eu volutpat odio facilisis. Pellentesque habitant morbi tristique senectus et netus et malesuada fames. Porta lorem mollis aliquam ut porttitor leo. Vulputate enim nulla aliquet porttitor lacus luctus accumsan tortor posuere. Ullamcorper velit sed ullamcorper morbi tincidunt ornare. Pellentesque elit eget gravida cum sociis natoque penatibus et.

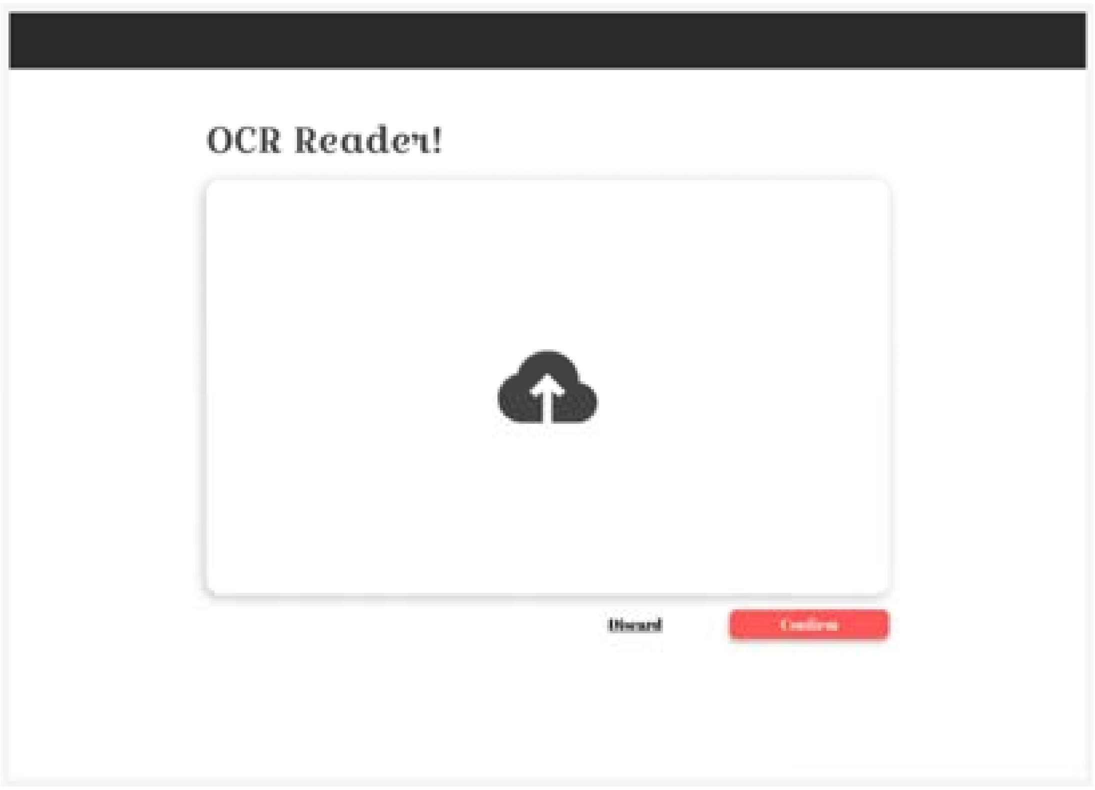
Facilisis gravida neque convallis a cras semper auctor neque. Malesuada nunc vel risus commodo viverra maecenas accumsan lacus. Ullamcorper eget nulla facilisi etiam dignissim diam quis enim lobortis. Lacus laoreet non curabitur gravida arcu ac. Aliquam etiam erat velit scelerisque. Sit amet risus nullam eget felis eget. Scelerisque purus semper eget duis at. Eget felis eget nunc lobortis mattis. Feugiat in fermentum posuere urna nec. Volutpat est velit egestas dui id. Imperdiet nulla malesuada pellentesque elit eget gravida cum.

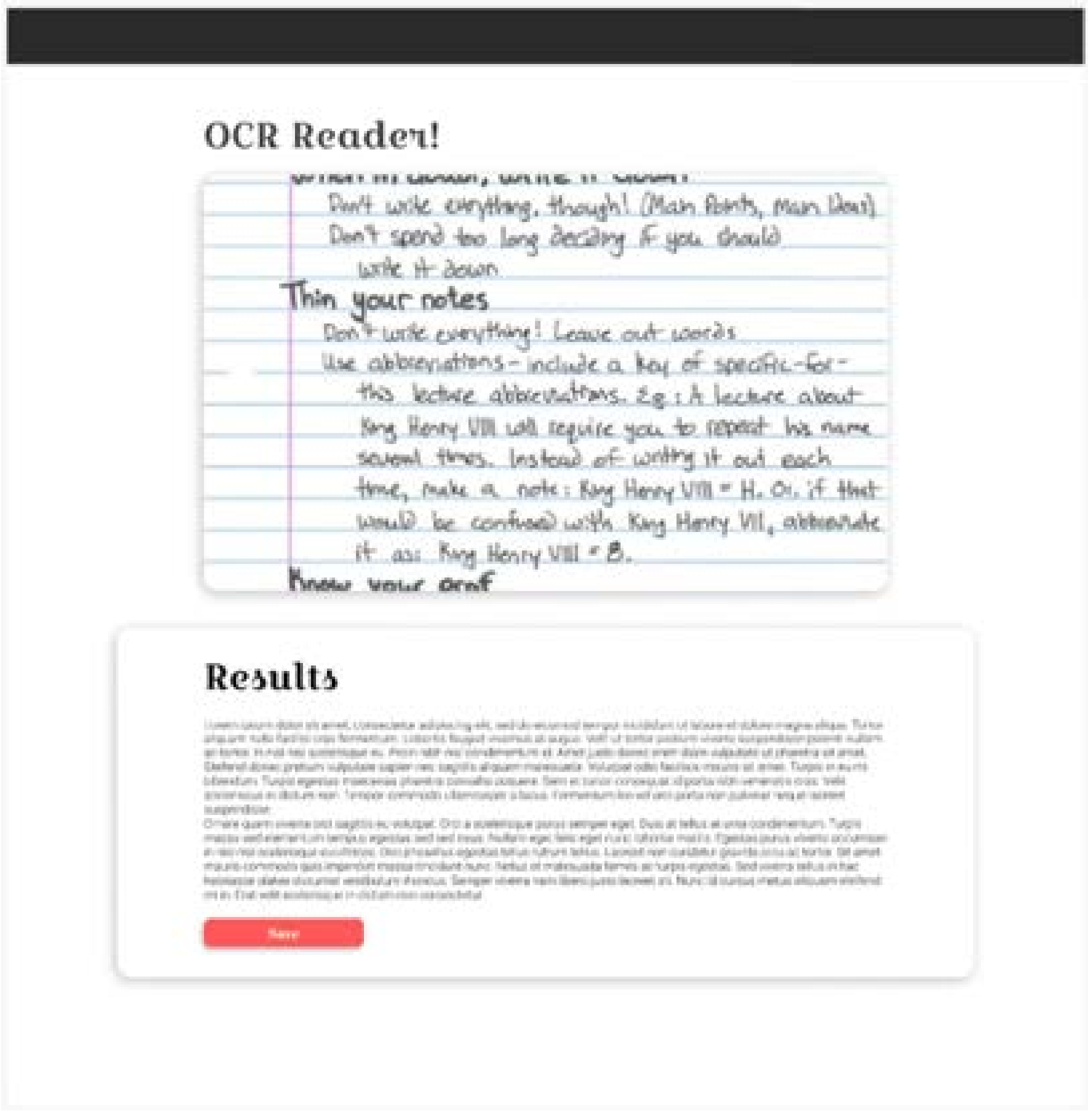
To-Do

- ☐ Testing
- ☐ New Testing
- ☐ Testing Testing Testing Testing Testing Testing
- ☐ Testing Testing Testing Testing Testing Testing TESTING TESTING TESTING TESTING


College Notes
Name of Note
Name of Note that is slightly longer
Name of Note that hits the cap number o...
Name of Note
+ Add New Notes


College Notes
Edit Document
New Email
Discard
Confirm
Edit
Delete


OCR Reader!
Discard
Confirm

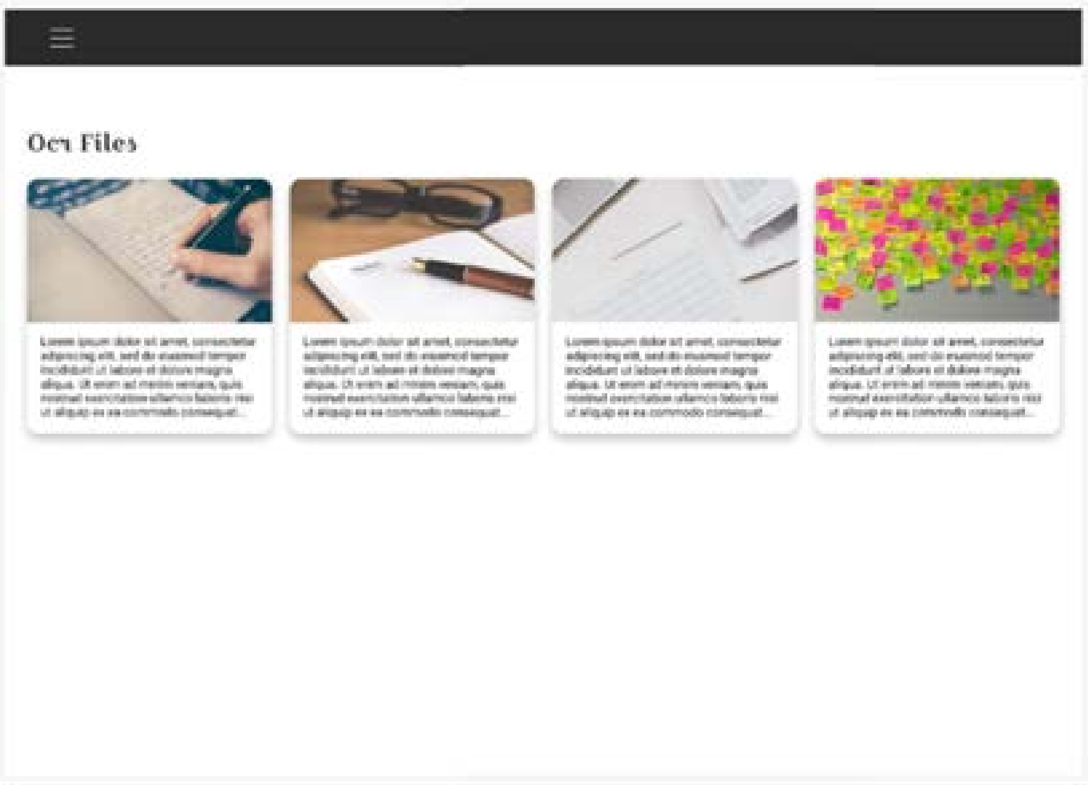

OCR Reader!

Don't write everything, though! (Main Points, main Ideas)
Don't spend too long deciding if you should
write it down
Thin your notes
Don't write everything! Leave out words
Use abbreviations - include a key of specific-for-
this lecture abbreviations. Eg: A lecture about
King Henry VIII will require you to repeat his name
several times. Instead of writing it out each
time, make a note: King Henry VIII = H. Or, if that
would be confused with King Henry VII, abbreviate
it as: King Henry VIII = 8.
Know your prof

Results

[illegible]

[illegible]

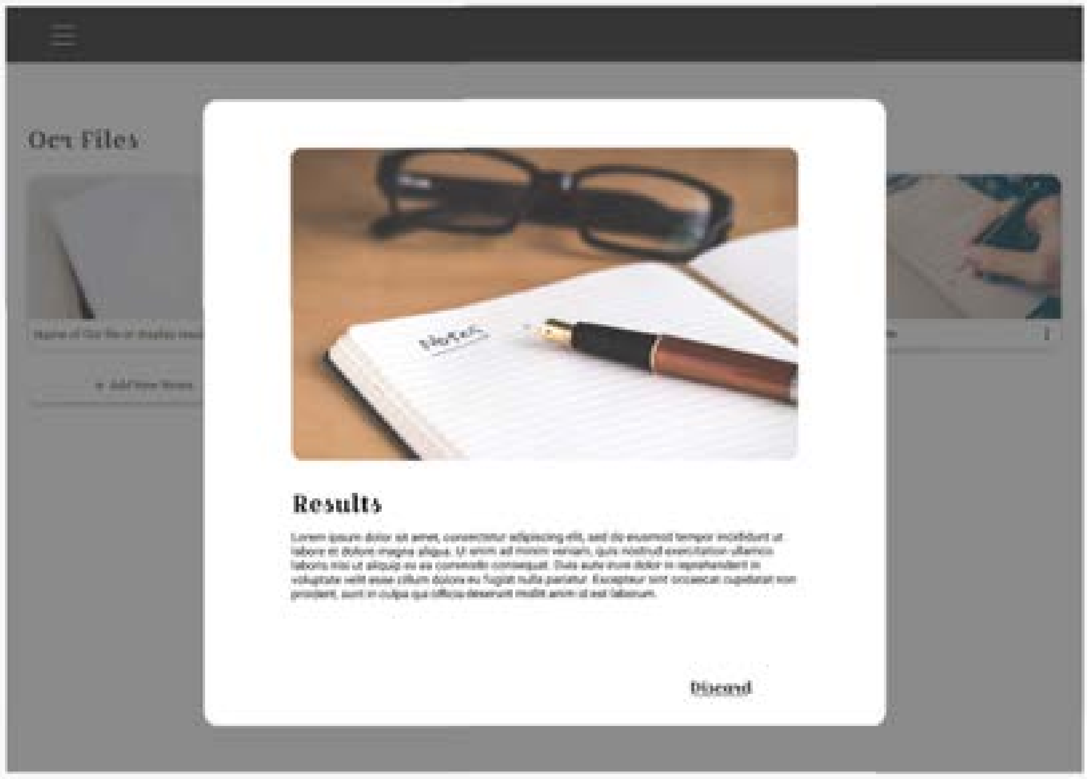

Our Files

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat...

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat...


Our Files

Name of the file or display name

+ Add New Notes

Results

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Discard

10.3 Appendix D – Important Links

GitHub of Final Application: https://github.com/IADT-projects/y4-project-CjayGonzales

Hosted Application: https://final-project-hosting.web.app/

Figma: https://www.figma.com/file/8wushDYxUM3YNyoGlC2RUW/Final-Project?node-id=0%3A1&t=IChSggepWepbklIC-1