2024

Doccy

CI601 – The Computing Project

INDIVIDUAL REPORT
FIN WATLING

Project Supervisor: Andrew Montgomery

Secondary Reader: Marcus Winter

Word count: 11587

STUDENT NUMBER | 20838503

Abstract [1.0]

The overarching aim of my project was to create a software product that allows for the easy management and secure storage of a user's personal documents, allowing them to easily prove their identity without the hassle of searching through their emails and personal files to find the required documents at the time of use. The software allows the user to store document securely and share selected documents with a third party via email. The sharing process is intended to be as simple and intuitive as possible both for the sender and the recipient. The product will be referred to as "Doccy" but is also referred to as "software product" when expanding on more technical aspects of the project throughout this document. The final product that I have created works well and allows for the secure storage and intuitive third-party sharing of user documents from within the software product.

Table of Contents

Introduction [2.0]

In the same weeks that I was brainstorming ideas for my final year project, I was also applying for new accommodation and parking permits in Brighton. While I was compiling the required identity documents, I found myself wishing that there was an easier way of storing and sharing these personal files. Due to screening companies needing up-to-date documents, each time I was needing to prove my identity I would have to scour my emails and find the documents from where I had previously shared them with other companies. I wished that there was a secure, central location that I could upload my personal documents to, to ensure that whenever needed, I would have these documents all in one place.

One might argue that personal document storage solutions such as Google Drive or Dropbox may be suitable for this use. For some people, this may be fine. However, I didn't want to have the hassle of creating my own directories/folders, managing the document storage structure myself. I did notice that there were more specialised storage solutions available, such as Egnyte and Smart Vault, however, these solutions were not targeted at the solo casual user and instead were designed for large scale business integration.

Instead of using these services, creating a single software solution that is entirely focused on storing and sharing personal proof of address documents would be ideal, and presented an interesting challenge.

For these reasons, I decided to base my project around this idea and create a personal document storage solution that kept ease of use, security, and accessibility at its forefront.

Aims and Objectives [3.0]

As stated in the introduction, the main aim of the project is to create a document management system that is easy to use for the end user, with a focus on security. The user should be able to sign up with their email address. Once this email address is verified, the user should be able to access the documents area within the system. From here, the user can upload, share documents with a third party, and delete documents.

The software will take the form of a web application. The final version of the website is a Blazor server application with a Microsoft SQL Server database for storage. The initial development plan was to use the React JavaScript framework in conjunction with Firebase as the storage solution. These changes were unexpected; however, they were for good reason, and I'll be explaining them further later in the report. The main aim to highlight here is that the software product was intended to be a web application.

I'm intending for the software's design to be as clutter-free as possible, with an intuitive flow to perform all common user actions. I will undertake user testing to ensure that the application delivers the intended user experience, and I am sure that some changes will be made to functionality based on this testing.

Although the previous aims are important, the most important aspect of my project to consider is the security and integrity of the data being stored. As the users will be uploading their personal identification documents, I will have to be incredibly careful with how I'm managing access to said data. As previously stated, the users will be verified via email before they can access their document area. This is not only important from a personal security standpoint, but also important for the safety of those receiving the documents through the share functionality.

Although the creation of the software artifact is seen as the most important objective of the project, the documentation of said creation is just as important as the final product. Without proper documentation, testing and progress tracking, I will most likely fail to meet my development targets and will struggle when it comes to writing the rest of this final report. For this reason, an important aim of the project is to keep on top of project management processes and ensure that I keep track of all issues encountered during the project's development.

The project's overall success will be measured based on its adherence to the described aims above, and the degree of this success will be discussed in the evaluation section of this document.

Another note to add here is that when I wrote my initial project proposal, I had intended to add more depth to the sharing functionality within the system. I had intended for the user to be able to choose between multiple share types, such as Email, SMS, or by the scanning of a generated QR code. I also had the aim to allow the user to set an expiration date for the documents shared, so that after a certain period, the recipient would have to request access to the file(s) again. The fact that this functionality was not possible to implement due to technical restrictions beyond my control highlights the importance of thorough research and planning, including into edge cases of the system. I will be writing more about this in the technical challenges section of the document.

Deliverables [3.1]

1. Final Report
2. Working code and Database Schema
3. SUS Usability testing undertaken
4. Evidence of question asked in the Microsoft Forum

Deliverable 1

This is this report itself. This includes the detailed breakdown of the project, and contains an all-encompassing description of the project's development, along with critical discussion on the project’s outcome, all major areas of personal development and learning, and any supporting documents created during the development process. I will also include a link to the live web application in the appendix.

Deliverable 2

This the actual code (and database schema) developed to meet the requirements. This includes all classes, razor files and configuration files within the Doccy Visual Studio solution and will consist of:

DataAccessLibrary – A Class Library, which contains all models associated with the creation of User and Document objects within the system, along with the classes written to allow the main Blazor Server project “Doccy” to make calls to the database.

Doccy – A Blazor Server Project that contains all the files of the website, this includes the component files, layout files, razor pages and configuration files. This also contains sub folders for the main services created: **Authentication** which contains the classes concerned with the user authentication service, **Communication** which contains the classes concerned with the email send service.

Database Schema – The database schema and any supporting documentation

Deliverable 3

This is the feedback that I receive and final calculation from performing a SUS usability survey, I will include the summary and score of this testing within this document, but I will also include the individual’s responses in a separate folder within the submission. All data will be anonymous.

Deliverable 4

This is a PDF that I will be including in my submission that demonstrates my use of the Microsoft forums. I was struggling with a specific question regarding the deployment of my Blazor Server application to the Azure Web Service and so posted a question and received some feedback from the wider Microsoft community. This feedback was helpful, and I managed to find the solution to the issue myself. I'm adding this to support the narrative that the project was a huge learning experience for me.

Project Planning [4.0]

Methodology [4.1]

I had originally intended for my project to adopt the Scrum development methodology during the development of my project.

Scrum is an Agile approach that utilizes a product backlog of work items, these work items are then added to a planned sprint (usually lasting 1-2 weeks), the work items are selected for the upcoming sprint based on a meeting with the team in which they discuss the achievable development progress during the sprint's timeline. These work items are then assigned to individuals on the team and the work is pushed to a development branch. After the sprint is finished and the work items added to the sprint are considered done and tested, the team will usually review the sprint, save the state of the project, and turn their attention to the next sprint.

This methodology is incredibly common and during my placement year I worked within a team that utilized this approach.

However, the Scrum methodology works best in medium to large development teams. As I'll be working individually, I believe a Scrum/Kanban hybrid will suit me best.

In essence, Kanban operates by representing work items as cards on a board. This board is usually split up into sections that describe the various states of the work items, such as "To Do", "In Progress" and "Done". Usually, the Kanban approach would limit the number of cards in each state to prevent overloading of tasks.

As I'll be using Azure DevOps to manage my project, I believe that I can create a good middle ground between the two approaches by using both the "Work items" board and creating a single 2-month long sprint. By doing this, I can add all work items that I **know** will need to be completed at the start of development, such as work items concerned with planning UI design, setting up the required databases and hosting solutions etc. Later, if and when requirements change or new work items that weren't originally identified in the planning stages arise, I can add these new items to the backlog and add them to the sprint as I see fit.

I will be pushing my work to a master branch which is also hosted on Azure DevOps. Whilst developing a new feature I can push to another branch, "dev" for instance. Periodically, as work items are pushed and tested, my commits for each work item will be linked to the work items using the Task ID's and I can merge this development branch with master. Once this merge completes, the work items will be marked as "Done". This will ensure that the master branch stays a stable, usable version of the product.

This hybrid model means that I have both the flexibility that Kanban boards allow, whilst still retaining some of the SCRUM methodology.

Another benefit of this method is that whilst developing new features, I can check-out the development branch locally, and run my Blazor Server application on localhost. Allowing for easy testing of new features without affecting the master production version which will be hosted on an Azure app service.

Development Tools [4.2]

Throughout the development process, I utilized many different tools. These are as follows:

Visual Studio

I used Visual Studio 2022 as my main development IDE. It is one of the most used and powerful IDEs on the market, and I was completely familiar with its operation due to having worked with it for a year on my placement to develop .NET solutions. Visual Studio has some excellent plugins for writing in many different languages, and as Visual Studio and Blazor Server are both Microsoft products, the Blazor integration is seamless. I can run my Blazor Application on localhost without installing any external plugins, and easily set breakpoints and step through my code in real time to find and solve bugs.

Azure DevOps

Azure DevOps is the platform I am using for my project management. It provides tools for continuous integration, including deployment to the Azure App Service. Again, I used DevOps extensively during my placement year and so was quite familiar with its operation. I was not familiar however, with setting up a team within the system as I only ever used it as a low privilege user. Its integration with Visual Studio and the other Azure services made it an obvious choice for managing the development process.

SQL Server Management Studio 19

SQL Server Management Studio is a database administration tool that allows you to log in to a database server of your choice and perform operations on the tables within it. I used it to create all my database tables and would use it to run T-SQL queries to test that the tables are behaving as expected. Again, I used this product extensively during my placement year which made me very comfortable with its use.

ChatGPT

ChatGPT is an advanced language model. I used the tool to streamline my development process. It can give some incredibly useful advice on a wide range of topics. If I found myself encountering an error in my program that I didn't understand, showing the debugger output to ChatGPT would give me tips on how to solve the issue and explain the error in detail if it has encountered it before. It's a very useful tool, however, in its current state it should only be used for guidance as occasionally the language model would give false information and would need to be corrected.

Chrome Developer Tools

The Chrome F12 developer tools window is a very useful utility for any kind of web development. Using it, I can easily check the console, file sources, elements and network activity when browsing/testing a website. This is invaluable when debugging the live site as it gives a great overview of any issues being encountered, allowing for immediate troubleshooting and optimization of web performance directly in the browser environment.

Azure Portal – App Service

The Azure Portal App Service is where my live web application is hosted, it includes some incredibly useful metrics that allow me to fully understand the load that is being put on my application at any given time, including server errors, data in, data out, requests, and response time. Each of these metrics are customizable and you can choose the period that is being displayed. The free tier is all I need for the project as it allows up to 60 minutes compute time per day. It also includes a Deployment Centre tool which allows for the constant monitoring of a git repository. Once set up, whenever a commit is pushed to the branch selected in the Deployment Centre settings, the changes are pushed to the live site.

Azure Portal – Communication Service

The Azure Portal also includes an area called the Communication Service, this service allows users to set up an Email/SMS service to allow for the sending of emails and text messages via the Azure Communication Service API. This tool is extremely intuitive, and I found the setup easy, there doesn't seem to be a limit for the number of emails sent per day when using the free service, so it has been very useful during testing. However, the SMS area is a paid tier only and so I couldn't use it for my project. The only other limitation is that the free email tier doesn't allow for sending emails via a custom domain. However, this isn't a big issue as users have no reason to reply directly to my email service.

Azure Portal – SQL Server & Database

I am hosting my SQL server and Database using the Azure SQL Servers area. I have really enjoyed using this service as it's extremely easy to get a database and server up and running. The free tier allows for 2GB database storage, and its wide range of customization features makes it an obvious choice for my project. Like the Azure App service, there are automatically generated graphs to view, which show many different metrics one might need to determine load on the database. The connection strings area is very easy to understand, and it provides default strings pre-formatted for use with ADO.NET, JDBC, ODBC, PHP and GO which is useful. It also allows for great customization on different security options that I may want to enable to protect the data stored on my server.

Task Breakdown & Timeline [4.3]

Appendix [1.1] - In this screenshot you can see my Azure DevOps Task board for my Sprint 1. This shows some of the tasks in the "To Do", "Doing", and "Done" state. This sprint, along with the work items board is what I used to track my tasks and get an idea of my burn down rate.

Appendix [1.2] – This screenshot from my interim report shows the initial plan for developing the different areas of the system. Although this doesn't mirror exactly how the development process progressed, I tried to stick to this schedule as much as possible. However, when working on a solo project there are many unknown variables and issues that arise that cannot be pre-planned and so there was a considerable deviation from the initial plan.

Appendix [1.3 – 1.4] – These screenshots show the DevOps Doccy Team backlog. This is a list of all 46 tasks that were added to the board and includes the various states that they were in at the time of capture. These tasks are added to the sprint **Appendix [1.2]** as I work through them.

Appendix [1.5] – This screenshot shows the Azure DevOps Doccy Summary page project stats. This shows the number of work items created and completed, and the number of commits made in the last 30 days. This screenshot also includes information about the percentage of pipeline builds that succeeded in the same amount of time.

Appendix [1.6] – This screenshot shows the DevOps Doccy Team Overview page. This page can be customized to show different graphs to the team to get a better understanding of the project's progress. The screenshot provided shows that the burndown for this sprint is -0.2. This is an unfortunate drawback to using the single sprint method that I outlined in my methodology section. Because whilst writing this report I'm having to add more tasks for myself to complete, it shows a negative burndown as the project's task scope has been increased. Although this isn't too much of an issue because I understand the reason behind the negative burn down, it leaves me questioning my true average burndown over the course of the project. This impacts the data that I can present in my report and so is something I'll be discussing in my final evaluation section.

Initial Plan [4.4]

The below steps outline the initial plan that I had set when writing my interim report (based on Gant chart in **Appendix [1.2]**. These steps did change as the database used in the final product is different, as is the web framework and language, however, I still believe it shows a good overview of my planned steps as the key features (DB setup, UI design, Initial Blazor commits) were completed in the same order.

Initial UI Design

Using Figma.com to design the initial UI specifications. Free tier has everything I need, I can consult my peers whilst designing my user interface to ensure that the design is intuitive and concise.

DevOps Repo Setup

I'll be setting up the DevOps repository, this time includes any research needed to get the repository up and running. I will mainly be using YouTube tutorials for this but if I come across any issues, I can ask a question on an online forum specialising in Azure DevOps or speak to some colleagues that have been through the process before. I'll also ensure that I familiarize myself with the Azure DevOps documentation.

DevOps Pipeline Setup

As above, this process will be new to me, and I'll need to include the research time in the plan.

Initial create-react-app push to Repository

This step shouldn't take long as all as I have been through these steps previously for separate projects. I don't envision there being any issues surrounding this process.

Implement UI Design and layout within React (buttons nav etc)

Implementing the UI design and layout will take some time as I'll need to write the CSS styling using Tailwind and create all the necessary html components. I have been through these steps previously for my Advanced Web Development project and this information is still fresh in my mind. I don't envision any problems with the step but as most of it is trial and error, I'll need to set aside more time for this step than the previous.

Set-Up Firebase DB

Setting up the Firebase database is the next step. I practiced setting up a test database when I was researching which database I should use, and so I am now familiar with this process. The element that will take the most time is ensuring that I have a suitable database design plan laid out.

I will need to ensure that the tables are set up as I need them for the final product including any encryption properties. Again, this may take longer than expected as although I have used Firebase previously, I haven't used it for a large project such as this and so may need to familiarize myself with features that I haven't touched before.

Set up hosting service (AWS) and connect to Azure DevOps Pipeline

I'll need to ensure that I have a place to host my web application. In the steps before this, I can run the React application locally, however, in this step I'll be moving towards a fully hosted solution. I haven't done this before and so I will need to give myself some extra time to allow for delays in learning the basics of AWS hosting and creating a pipeline that can automatically deploy the live version of my web application to the server.

AWS has comprehensive documentation that can help me with this process, and I'll need to familiarize myself with it. Although I don't want to rely on YouTube tutorials, aside from with the documentation there are a few good videos I have found of this process which I can use as a reference when undertaking the task myself.

Begin development of core features

After all these steps are in place, I can begin the development of the core features required by my application. I can begin to implement the functional requirements and write test cases for each so that my project can be fully tested when pushed to the repository. Given that all the previous steps have been done correctly, I shouldn't have to back track and reconfigure much with my project setup. I'll ensure that I monitor Azure DevOps and link each of my tasks to the git pushes. This will create parent-child links and enable me to handle any bugs that may occur in the future using the ID of the task that caused/fixed the issue.

Review plan and adjust according to progress

At this point, I should be able to further plan my time, I will have learned a significant amount about my project build and based on my Azure DevOps data will be able to give more accurate "effort required" figures to each task that I raise for myself. It is from here that I can plan future development efforts. It may be that at this point I can release an early version of the product to the live server for some more thorough testing with live data.

Risk Analysis [4.5]

Risk Description	Likelihood	Impact	Mitigating Action
Azure (host) experiencing downtime	Very Low	High	Close monitoring of Azure services to find a temporary alternative host if Azure is going to experience downtime.
Sickness	Medium	High	Ensure I adhere to strict task prioritization and communicate well with supervisors. Apply for extensions should I require them.
Skill Gaps	High	Medium	Ensure that I allocate sufficient time during my project for learning new technologies that I haven't used before.
Unforeseen Technical issues	High	High	Unfortunately, there isn't much that can be done to plan for unforeseen issues. However, a robust and comprehensive plan of tasks along with alternative technologies can help mitigate issues that can't be solved.
Time Management issues	Medium	High	Ensuring that I follow the tasks set out in the project plan and keep to a strict number of work hours on my project per week.
Computer hardware failure	Very Low	Medium	As my work will be pushed to the remote development branch after each work session, this shouldn't have a large impact on the project. Using hardware monitoring tools such as CCleaner may be a good prevention method.
Change of user requirements	High	High	It is important to manage the scope of my project, ensuring that the requirements don't stray too far from the initial plan. Frequent project requirement reviews early in the project may help to mitigate these requirement changes from happening later in the project when the impact would be higher.

Research [5.0]

Literature Review [5.1]

Pre-Project Research Conducted [5.2]

Firebase for sensitive data (Google, n.d.) (Colvin, 2021)

I came across this website when I was trying to decide which database solution I should use for my project. The article explains a vulnerability in the Firebase SDK which may cause some sensitive cached data to be saved to a .Json file that can be read as plaintext by anyone able to access the directory. This includes the authentication token and refresh token of the database connection. Although initially worrying, the article mentions that the issue is with the Firebase SDK itself and not Firebase as a whole. Firebase is accessible through a parallel API and that's what I'll be using for my project. This allows me to handle all cached data manually and although it will require some extra effort, it'll be worth it to use Firebase's user authentication features.

Verifying file integrity (Jones, 2022)

In this article from TechTarget, they discuss how to check and verify file integrity. I was researching this topic after my initial meeting with my supervisor in which I struggled to differentiate between authentication and verifying integrity. The article suggests the use of comparing hash codes and metadata to ensure that the file hasn't been tampered with in transit. It suggests that the use of automated checksum comparisons should not be used in isolation as this method alone can sometimes be worked circumvented.

Ensuring data integrity with hash codes (Microsoft, 2023)

This .NET learn article gives a more granular example of using hash codes to ensure the integrity of data. The article shows a walkthrough of generating a hash and then verifying that hash within a .NET project. Although this isn't entirely relevant to my project as I'm using JavaScript with

React – it’s still a useful example of how hash codes can be utilized to ensure the integrity of data.

Sending SMS with AWS Amplify (Liendo, 2021)

This article explores the key concepts behind Amazon’s SNS service and demonstrates its use using AWS Amplify. The article contains a watch-along video as well as code snippets to show the process of sending an SMS message from within a JavaScript project. This article also explains the benefits of Amazon’s Simple Notification Service and the possible reasons for its use. The article also explains and demonstrates the setup and authentication process that developers need to complete before they can connect to the service’s API.

Azure DevOps Scaling (Microsoft, 2023)

This Microsoft Learn article discusses many useful aspects of Azure DevOps development and explains the DevOps layout. It discusses the importance of limiting user visibility within the project scope and further explains how to limit access to areas of the project that may need some level of access control.

Research Conducted During Development [5.3]

BCrypt Password Hashing Library (Arias, 2021) (BCrypt, 2023)

This is the documentation for the node.bcrypt.js JavaScript library. This library is used to hash passwords for safe storage and authentication purposes. The library I ended up using was a NuGet package that works with .NET. Although this documentation is for the JavaScript package, a lot of the documentation was still relevant and gave me an overview of how to use the BCrypt library. BCrypt works by using a random string of data (the salt) during the password hashing process and storing it in the database alongside the hash. When we take user input for the password, BCrypt can take the user’s entered password, hash it again using the salt and compare the stored hash to the generated hash from the user's input. If these values are the same, the user can be authenticated as this means they typed in the correct password.

Data protection under GDPR (Your Europe, n.d.)

The information that this article on GDPR (General Data Protection Regulation) provided was invaluable during the development process and impacted my choice of database provider. I had originally planned to use Firebase to store the user data and personal documents. However, I could not be 100% sure that Firebase conformed correctly to these GDPR requirements as the data could be stored outside of the EU, "Storage of data outside the EU is forbidden by the GDPR". In contrast, my research into Azure hosting showed that they are fully GDPR compliant, and they offer the option of choosing where each database solution is hosted.

Microsoft Azure General Data Protection Regulation Summary (Microsoft, 2023)

This Microsoft Learn article contains a summary of the GDPR both in the context of the law and in the context of the Microsoft ecosystem. It gives a rounded explanation of what GDPR is, why it matters, and goes on to discuss the processes that Microsoft follows to ensure that they are fully GDPR compliant across their ecosystem. The article also has links to a tool called the Microsoft Compliance Manager which is used to "automatically assess and manage compliance across your multicloud environment". Although I haven't used this tool in my solo project, it is worth noting that it exists in the future if I wished to scale a project and needed to measure my compliance.

Host and deploy server-side Blazor apps (Microsoft Learn, 2024)

This Microsoft Learn article includes some useful information on hosting and deploying server-side Blazor applications. This article is entirely relevant to my development process as I had to follow these steps when I was deploying my application. The article also includes valuable information on Blazor's SignalR Service. The SignalR service allows for real-time updating of

Blazor components. When a Blazor server application is run, a SignalR connection between the client and server is created, this allows for the application's UI to update in real time based on data changes within the server. This was especially useful within the development of my application because it meant that I didn't need to call any update functions when a document was uploaded, the page's data grid would automatically refresh once it detected a change in the stored data.

CI/CD For Blazor Applications in Azure DevOps (Terehan, 2022)

This article from C# Corner discusses the basic implementation and use of CI/CD within a Blazor project. It assumes that you have very limited prior knowledge of CI/CD which was very useful to help me understand how to create and run a pipeline within DevOps for the first time. Although I had worked with pipelines in the past on my placement year, I was never tasked with setting up a brand-new pipeline, so this was a new challenge for me which I really enjoyed. I had some issues with my pipeline during the development process and this will be documented in the Technical Challenges section of the document.

To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? (Russell Sears, 2006)

I came across this research paper when considering the data storage options available to me when developing my application. The technical report outlines the pros and cons of BLOB (Binary Large Object) storage and includes a considerable amount of high-level discussion of the differences between storing BLOBS in a database versus a filesystem. The report is a deep dive into the technicalities of the two storage options, even going as far as to discuss the performance impacts of free space on the drives, and how the two compare when low on storage space. The research paper provides a comprehensive analysis of the two types and is exactly what I was searching for. Reading and understanding it aided me in making my decision of using a database to store my files, since the size of the documents I'd be storing would be less than a few MB on average.

Legal and Ethical Issues Considered [5.4]

Throughout the development process, it has been important to consider a wide range of potential legal and ethical issues. The most obvious of which is the legal storage of personally identifiable data. As my application is concerned with the secure storage of personal identification documents, it has been important to research and understand the various laws and regulations surrounding this topic.

GDPR [5.4.1]

The first main regulation I've had to consider is the GDPR or General Data Protection Regulation.

The main principles of the GDPR when considering the storage of data are:

Data minimization – This is the process of ensuring that the company collects only the minimum amount of data for the purpose it is being collected. For example, if our application only requires the user to give us their full name to be functional, we do not, and cannot collect their address, or other personal details.

Integrity and confidentiality – This principle, in essence, means that as a company we need to ensure that personal data is kept safe from unlawful processing, accidental deletion, alteration or loss of any kind.

Storage limitations – This principle is concerned with the amount of time data is stored. Ideally, the customer's data should be stored for the shortest amount of time possible. It is encouraged for companies to review or erase data after a certain amount of time. An example of this could be a yearly review, if a customer hasn't accessed their data within the last year, they will be sent a notification and if no action is made, deletion of their data will occur. (Iubenda, n.d.)

Another aspect to consider is the safe storage of customer passwords. As I'm using BCrypt to salt and hash my passwords before they're stored in the database, I conform to the GDPR's rules. BCrypt operates using a one-way hashing function, this means that even if I wanted to, I don't have the ability to access any unencrypted passwords.

As my website will be taking input of sensitive information such as personal documents, full name, email address and phone number, I will need to ensure my website uses HTTPS instead of the standard HTTP. This means that the network traffic is encrypted using SSL or TLS between the client and server.

Competitor Analysis [5.5]

Throughout my investigation into potential competitors, I have struggled to find an application or software that directly competes with my solution. The closest competitors that I found while researching have either been mainstream cloud storage solutions, which do not focus solely on storing and sharing personal identification documents, or, business file storage solutions, which are not designed to be used for personal document storage and are instead used to store and collate documents within a business setting. It is still helpful however, to analyse these solutions as there is significant crossover that may help me to gather requirements in the initial stages of development.

Cloud Storage solutions – Google Drive and Dropbox

Whilst my application is not a traditional cloud storage solution, I think it is important to investigate the current leading cloud storage solutions to determine what features make them stand out in their field, along with identifying any drawbacks. This is an important process to help determine if there is sufficient gap in the market for a product like Doccy to succeed.

Google Drive and Dropbox share a few similarities to the product I'm developing. I will be discussing these similarities along with any notable differences that I identify within my own solution:

Unique Log-in / User Accounts

Both Google Drive and Dropbox are only available to users who create an account, this is how I intend for my product to operate. The user will be required to create an account before they can access the documents area to upload and share personal documents.

File Upload

Google Drive and Dropbox allow for file upload. Like my solution, you can upload more than one file at once, and the file upload process is straightforward and user friendly. One aspect of the file upload process that I hadn’t considered before this research was the ability to drag files directly into the browser to upload them. Both Google Drive and Dropbox support this functionality and it’s something that I am now looking to implement thanks to the product research.

File Sharing

The sharing of files operates differently within Google Drive and Dropbox than within my application. When using these services, it is necessary to have an account to **access** the files being shared with you. When sharing a file to an email address from within Google Drive, the share process does not always send an email to the user you share with. Instead, the shared file will appear in the shared files area within the application. This functionality forces the recipient to create an account to access the files and is a major drawback to using these file sharing services in my opinion. I’d like my application to be able to share to email directly so that the recipient doesn’t need to create an account to view the documents.

Security Features

The notable security mechanisms that Google Drive and Dropbox utilize are:

Encryption: Data within Google Drive and Dropbox is fully encrypted using 256-bit AES encryption. I intend to follow suit in this regard to ensure that my BLOB data is fully encrypted within the database. I also intend to enable “Ledger mode” on my database tables, I will discuss this in further detail in the **Key Development Choices [7.1.4]** section of this document.

Two Factor Authentication: Accounts within Google Drive and Dropbox are authenticated using two factor authentication. My application will require users to verify their email address and will not allow users to upload or share documents until this process has been completed.

Business File Storage Solutions – Egnyte and SmartVault

When searching for other potential competitors, I came across Egnyte and Smart Vault. These companies offer online document storage services; however, these services are targeted at businesses rather than the individual user. I was initially keen to explore the products to help gather requirements for my own solution, but I found it hard to find any examples of the software’s UI or file upload/management functions due to the inability to sign up to the services as an individual, which greatly limited my ability to analyse the competitor’s features.

Another drawback of using these solutions for personal document storage is the price. Because the solutions are aimed at businesses, the fees involved would be far too high to be viable - Egnyte starts at 15 pounds per month per user and SmartVault starts at 30 pounds per month.

The final drawback of these solutions to note is that, as they are not primarily concerned with the single user, they include many business integration features with their suites which would not be utilized by someone trying to use the storage solutions for themselves. The target audience of the competitors is entirely different to my solution.

Competitor Analysis Conclusion [5.6]

In conclusion, I believe that there is a substantial gap in the market for an online personal document storage solution, I haven’t been able to find a product that fits the description of Doccy entirely and the solutions available are either targeted at businesses, like Egnyte and

SmartVault, or are not directly concerned with the storage and easy sharing of personal documents, like Google Drive and Dropbox. The main disadvantage of the cloud solutions to note are that a login is required in all cases to access the shared documents, which could affect the user's experience, especially when sharing identity documents with a company that may not have the facilities to create an account simply to access the files.

Requirements [6.0]

The requirements in the tables below have been identified by analysing potential competitors, along with brainstorming and planning sessions that I undertook at the start of the project. considering the functional, visual, and non-functional requirements within the context of the intended use of the system. Although these requirements will most likely evolve and change throughout the development process as requirement gaps are identified, the tables below show the initial requirements from which the development plan and backlog tasks will be created. They are tiered based on priority; the lower priority items are stretch goals.

Functional Requirements [6.1]

Priority: High, Medium, Low

Requirement	Type	Priority
User should be able to create an account and log in using email and password	Functional	H
User should be able to log in using their Google account	Functional	L
User should be able to upload documents	Functional	H
User should be able to add tags/categories to documents	Functional	L
User should be able to share a single document	Functional	H
User should be able to share multiple documents at the same time	Functional	H
Documents should be shared via encrypted means	Functional	H
Documents should be encrypted when stored in the database	Functional	H
User should be able to verify their account using their email	Functional	M
User should be able to delete a document from the system	Functional	H
User should be able to choose the method of sharing a document	Functional	M

Visual Requirements [6.2]

Priority: High, Medium, Low

Requirement	Type	Priority
UI should be intuitive and easy to use	Cosmetic	H
UI should have a light and dark mode that adapts to the user's default device setting	Cosmetic	L
UI should have minimal load time	Cosmetic	H
UI should be a single page application – reducing load time	Cosmetic	H
Any animations or effects should not be invasive or distracting	Cosmetic	H
UI should adapt based on device, fully responsive	Cosmetic	M
UI should be simple and sleek, no bright or disorienting colours	Cosmetic	H
UI should be styled in a way to allow for maximum accessibility	Cosmetic	H

Non-Functional Requirements [6.3]

Priority: High, Medium, Low

Requirement	**Type**	**Priority**
There should be no wait times longer than 2 seconds unless it's related to uploading large files or an outside service (eg. Google login)	Non-functional	H
All sensitive data should be encrypted during storage	Non-functional	H
All sensitive data should be encrypted during transmission	Non-functional	H
The system should be compatible with all major web browsers	Non-functional	H
The system should comply with General Data Protection Regulation (GDPR)	Non-functional	H
The system should be fully documented with comprehensive documentation for both users and developers	Non-functional	H

Development [7.0]

Key Development Choices [7.1]

Switching to Blazor from React [7.1.1]

Early on during the development process, I made the choice to switch from React to Blazor for my application's development.

The main reason for this decision is that React uses JavaScript, whereas Blazor uses C#. Throughout my placement year I was developing .NET applications using C# and so I was far more familiar with C# compared with JavaScript. The reason I chose to develop my application using React in the initial stages was because I wasn't aware that the Blazor framework existed at all, and I was developing a web application using React in a university module at the time of my project planning phase, so I chose it as I thought it would feel more familiar to me.

A slight disadvantage of using Blazor is that it is a far newer framework compared to React. This meant that there was less learning material available online when researching the development of Blazor applications, especially for the newest features that Microsoft introduced in November 2023 with ASP .NET 8.

I thoroughly enjoyed using Blazor and I'll be using it in the future for any web application-based projects I embark on as I found it very easy to pick up compared with React. The versatility of Blazor was impressive; being given the option to use Object Oriented Programming principles alongside more traditional event driven programming was extremely useful.

Choosing Blazor Server over Blazor App/WebAssembly [7.1.2]

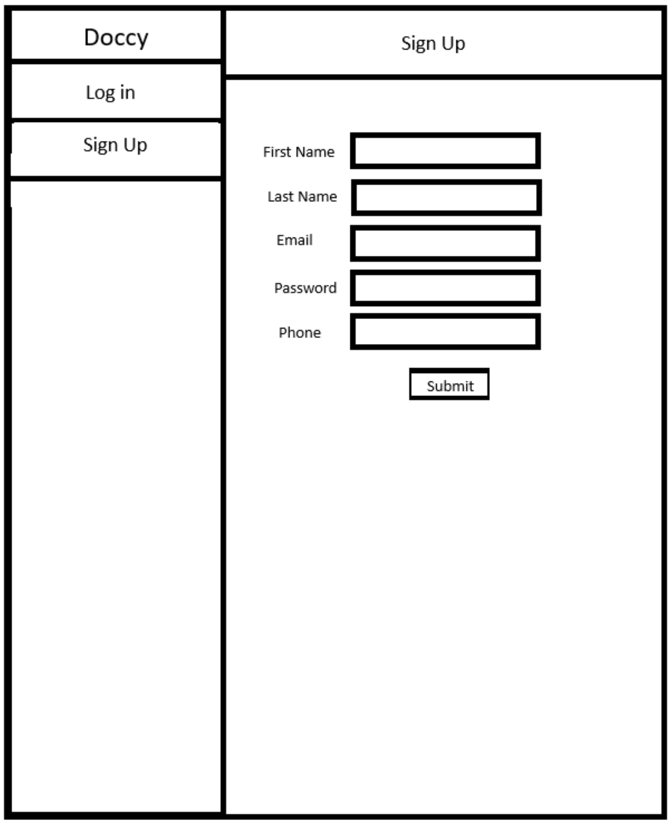
My main reasoning for using Blazor Server over Blazor App is that Blazor server uses a SignalR connection between the client and server, this allows for real-time updates of grid components. Another benefit of Blazor Server over Blazor App in my application is that the users of the web application cannot access any of the back-end code. The Blazor server is hosted within Azure, and the only files/folders that the client can access from the developer tools window is their own local/session storage. In contrast, within a Blazor WebAssembly project, the .NET runtime and the application code are downloaded to the browser, potentially exposing the business logic of the application.

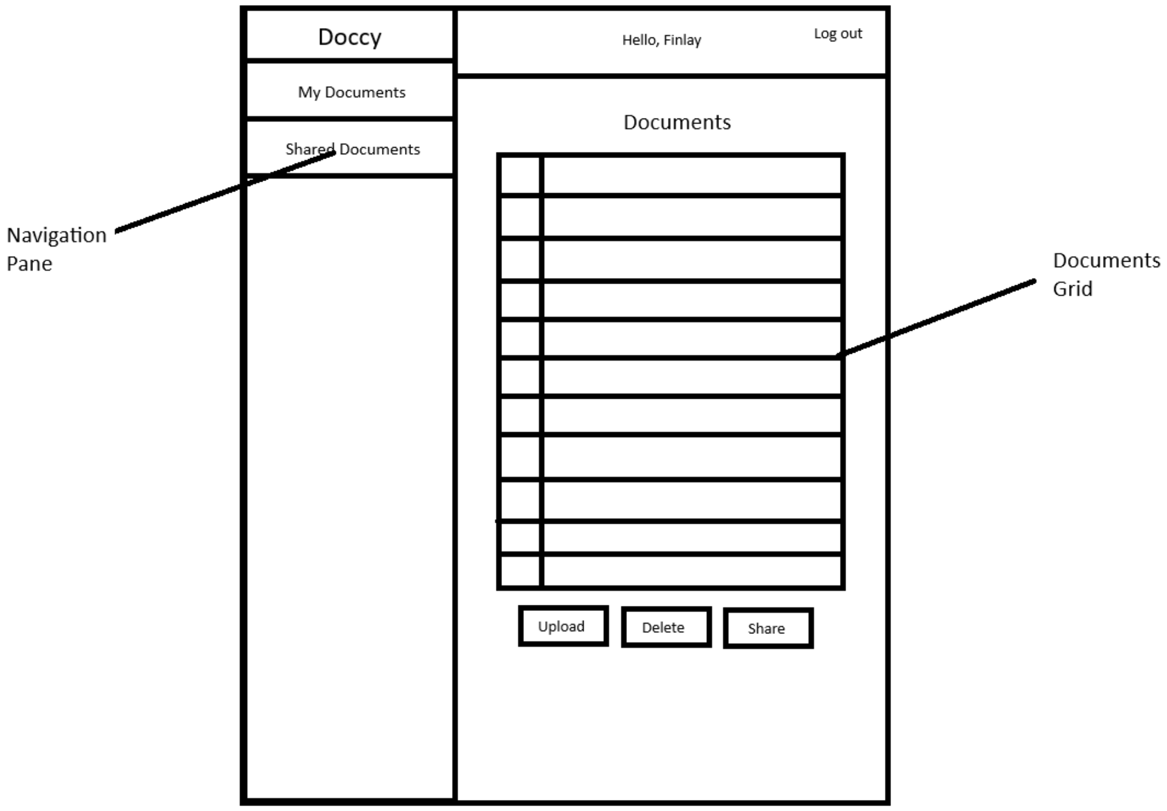
Switching from Firebase to SQL Server Database with Ledger Mode [7.1.4]

I had originally planned to use Firebase as the database provider for my application, however, as previously stated in my Literature Review section, during the initial stages of development I came across some documentation that suggested that Firebase was not compliant with GDPR laws. I did some further research into Azure Database hosting and found that they are compliant with GDPR laws and so I opted to use an SQL Server database hosted within Azure instead. Not only is the new solution compliant with GDPR, but using the Azure Database Portal I can make my database tables Ledger tables.

Ledger mode works by keeping a log of all transactions within the database meaning that once a transaction has been completed, it cannot be altered or deleted. This process also creates an audit trail that ensures the integrity of the data.

Initial UI Designs [7.2]





UI Design Discussion [7.2.1]

Early in the planning phase of my project I created the above UI Designs. They show a very basic overview of the application's appearance, along with the key layout of the application's functional features.

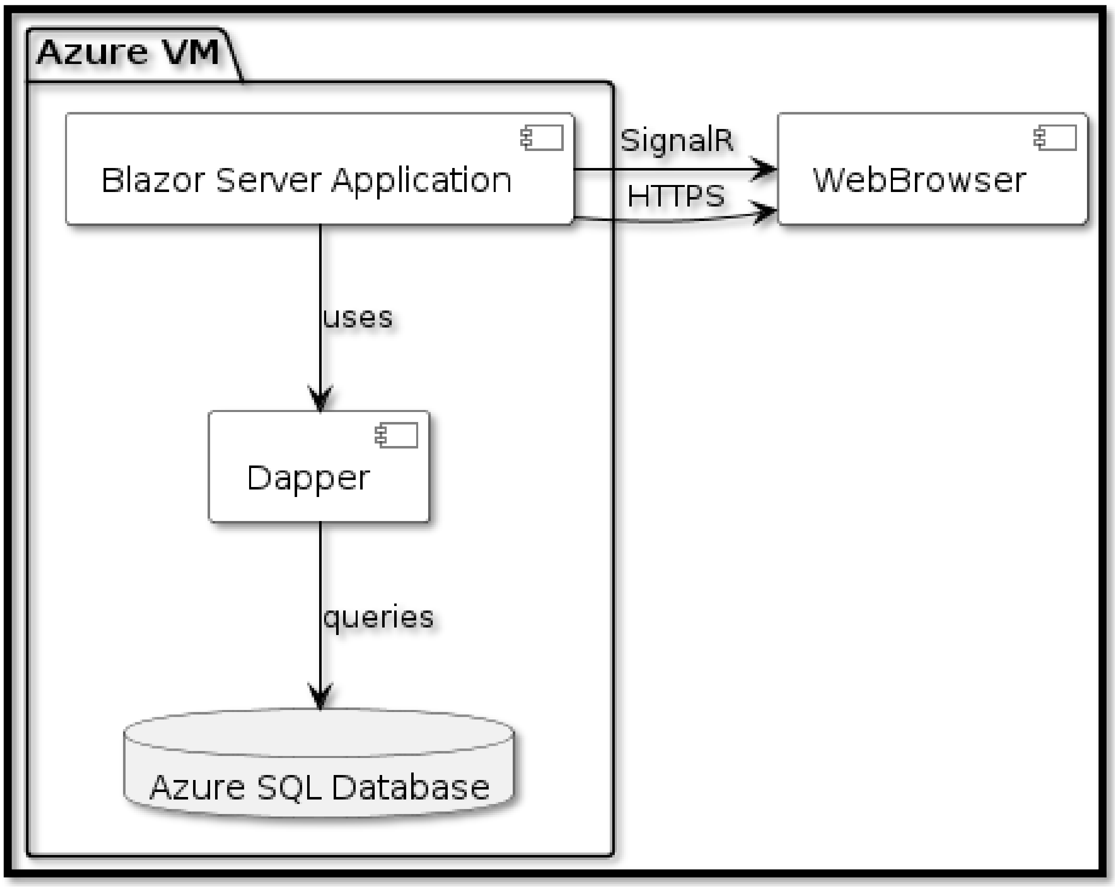
Side Nav Bar – In both screenshots, the side navigation panel is visible. This navigation pane is intended to stay visible on all pages of the application. The navigation pane's contents is changeable and displays different links to pages depending on the user's login and verification state. If the user is not logged in, as in the first design, the two available options are "Log in" and "Sign up". Once the user has signed up, a "Verify Account" link will become visible. Once the verification process has been completed, the Nav Bar will change to allow users to navigate to the "My Documents" and "Shared Documents" pages. The second screenshot shows the application in the logged in state.

Header – In both screenshots, the header bar is visible. This header bar is to display information on the logged in user, such as their username/email. The header bar will also include a log out button so that the user can easily log out regardless of what page they're viewing.

Documents Grid – This is a GridView element that will display the documents that the user has uploaded to the system. It will include information about the documents such as Title, Document Type, Upload Date, File Size and Last Edited Date. This grid will also include a checkbox to allow for the selection of documents. Below the documents grid are the buttons used to manage documents. Upload will open a file browser to allow the user to upload one or multiple files. Delete will allow the user to delete the selected document from the Doccy system, and Share will open the application's share functionality. The Documents grid will have a maximum of 500 rows returned to lighten the load on the application.

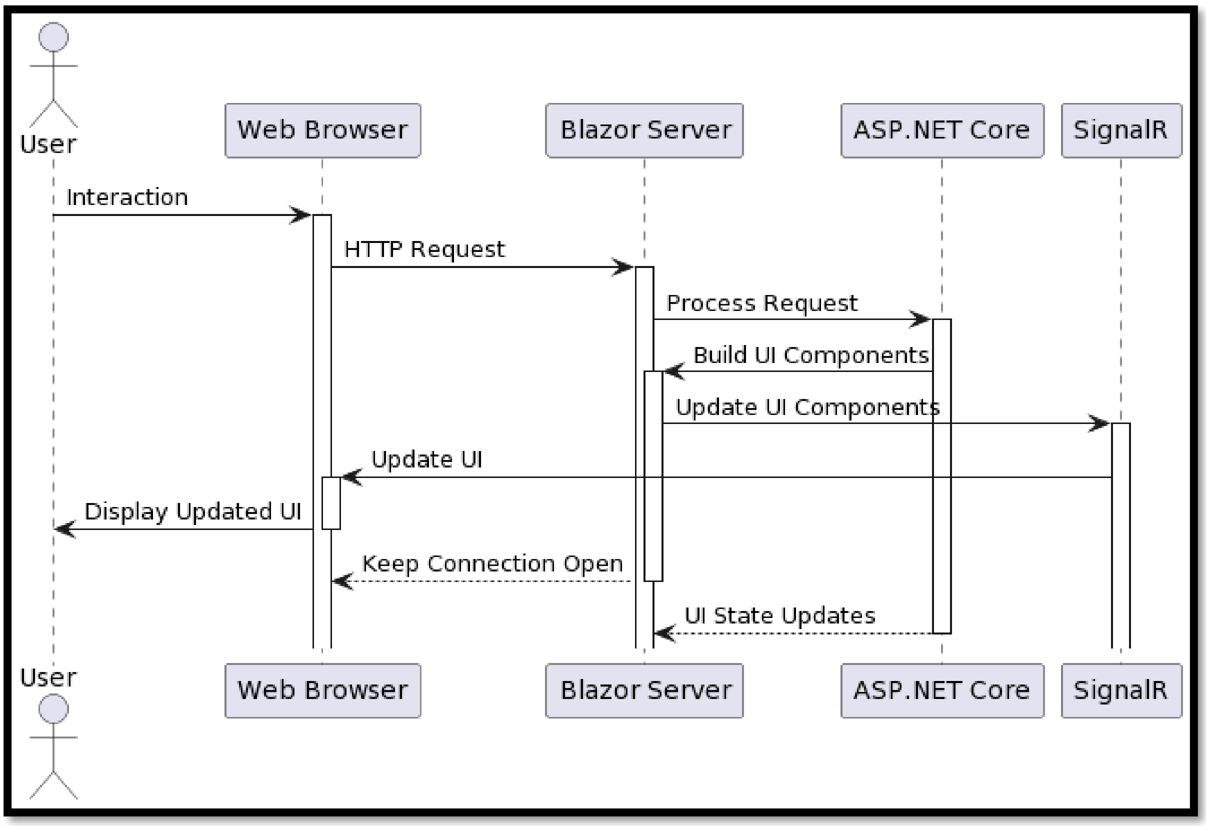
Architecture [7.3]

Azure Server Architectural Diagram [7.3.1]



The above diagram illustrates the overall architecture of my web application. The Blazor Server Application is hosted on an Azure virtual machine, the application's code uses a .NET NuGet library called Dapper to communicate with the Microsoft SQL Server Database which is hosted on the same virtual machine. The Blazor Server Application serves the client's web browser with the webpage via HTTPS and sets up a SignalR connection to allow for secure real-time transfer of data.

Application Sequence Diagram [7.3.2]



This second diagram is a sequence diagram which demonstrates the sequence of processes within the system. When the user interacts with the web browser, an HTTPS request is sent to the Server. Once this request has been sent, the application processes the request and builds the required UI components. The UI components are then updated on the server side, the SignalR connection updates the client's components and the user receives the updates within their browser. The SignalR connection stays open for the duration of the user's session within the web application.

Functionality [7.4]

In this section I'm going to reference the screenshots of the final version of the application to demonstrate the functionality of the finished software product.

When the user first navigates to the site, they are sent to the home page **Appendix [2.4]**. From this screen the user can either sign up **Appendix [2.0]** or log in **Appendix [2.1]** to the system.

If signing up for the first time, once logged in, the user will be directed to the account verification page **Appendix [2.2]**. The user will be automatically sent a verification email **Appendix [2.3]** containing a unique, random, 6-digit code, this code will need to be submitted on the verification screen before they can navigate to the documents page **Appendix [2.5 – 2.6]**. If the user has lost the verification email, they can press a button on the verification screen to send another verification code.

Once the user has verified their email address, they will be redirected to the documents page **Appendix [2.5 – 2.6]**. From here, they have the option to **upload** one or more files, **download** one or more files, **share** one or more files, or **delete** one or more files. They can select the files to operate on using the select checkbox next to each file. Once the "Share Selected" button is pressed, a dialog box will open prompting the user to enter the email address of the recipient **Appendix [2.9]**. Once the files have been shared, the documents page will automatically refresh and show the newly shared file(s) in the "Shared Documents" section **Appendix [2.8]**. The recipient will then receive an email with the documents a short time later (5-10 seconds) **Appendix [2.7]**.

Technical Challenges [7.5]

Throughout the development of my web application, I encountered various technical challenges that slowed my development process, and in some cases caused me to change requirements. In this next section I'll be detailing the main technical challenges that I faced, along with the solution that I implemented to circumvent the issue at hand.

Document Sharing Issue [7.5.1]

The documents stored within my database are stored as encrypted BLOB binary objects. This is a decision that I made based on the database provider research I conducted at the start of the development process. When I was creating the share functionality for my application, my initial plan was to include the attachment in the body of the email as a deserialized BLOB object. Sharing the documents in this way would allow me to set an expiration date for the document, which was an initial requirement identified. However, this was not possible since many email providers (including Gmail) do not allow documents to be displayed as serialized BLOB data due to security restrictions. The only way of displaying inline images in the email body is to reference where they are hosted.

e.g <img src="https://www.google.com/img/HOSTEDIMAGEPATH">

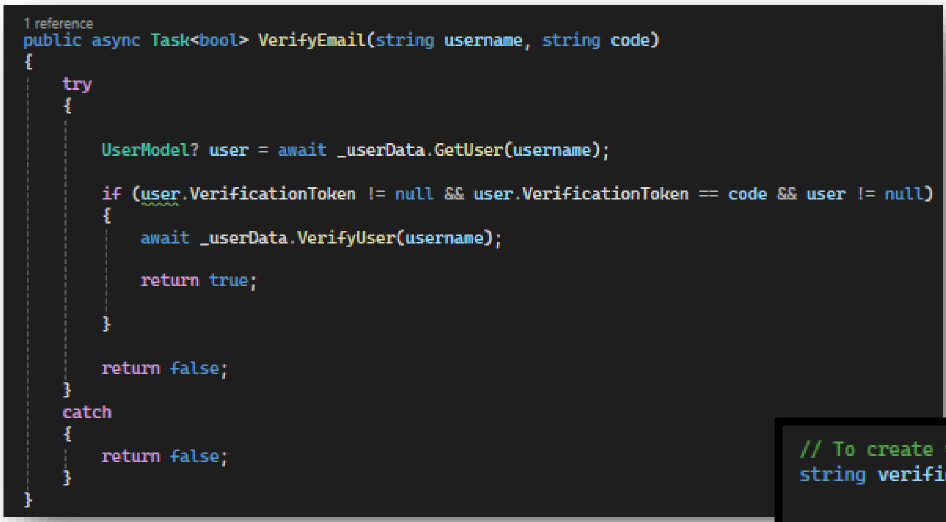
I only realised this limitation after a significant amount of development had taken place, including the finalization of my database design and deployment.

For this reason, I had to change the functionality of the project and instead include emails as attachments that don't have the ability to expire. I would have been able to add the document expiry functionality had I opted to use a Filesystem to store the documents as I'd just need to include a public link to the stored file.

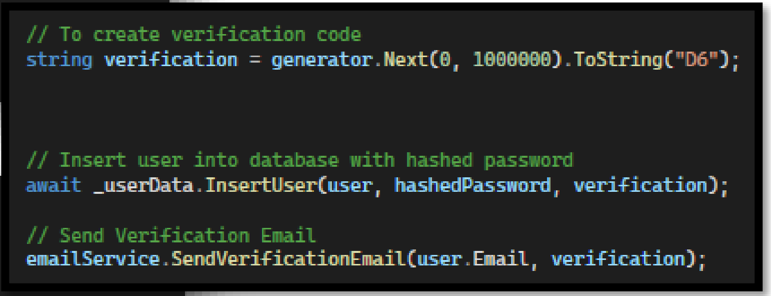
Although disappointing, this is a great learning experience and helped highlight to me the importance of thorough research in all areas of the intended functionality of the software.

Custom Email Verification System [7.5.2]

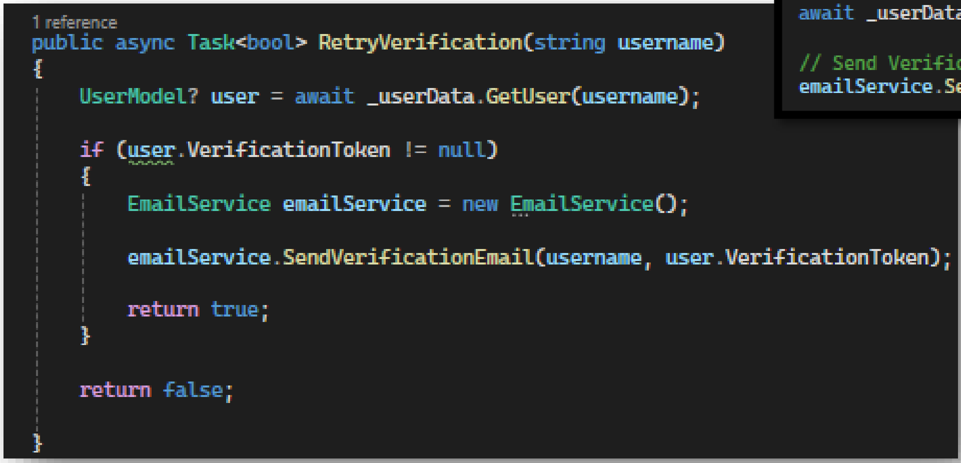
To ensure that users only created accounts with their own email addresses, I implemented an email verification system. This verification system uses the Azure Email Service to send a verification code to the user. When the user submits the sign-up process, the verification code is generated and stored in a column in the user table. At the same time, the same verification code is sent via email to the user's email address using the Azure Email Service. On the verification screen, the user is prompted to enter the verification code. Once submitted, the VerifyEmail method looks up the verification code from the database, if the stored verification code matches the entered verification code, the user is considered verified and the flag "isVerified" is set to true in the user table.



```
1 reference
public async Task<bool> VerifyEmail(string username, string code)
{
    try
    {
        UserModel? user = await _userData.GetUser(username);

        if (user.VerificationToken != null && user.VerificationToken == code && user != null)
        {
            await _userData.VerifyUser(username);

            return true;

        }

        return false;
    }
    catch
    {
        return false;
    }
}
```



```
// To create verification code
string verification = generator.Next(0, 1000000).ToString("D6");


// Insert user into database with hashed password
await _userData.InsertUser(user, hashedPassword, verification);

// Send Verification Email
emailService.SendVerificationEmail(user.Email, verification);
```



```
1 reference
public async Task<bool> RetryVerification(string username)
{
    UserModel? user = await _userData.GetUser(username);

    if (user.VerificationToken != null)
    {
        EmailService emailService = new EmailService();

        emailService.SendVerificationEmail(username, user.VerificationToken);

        return true;
    }

    return false;

}
```

Issues With Singleton UserContext Class [7.5.3]

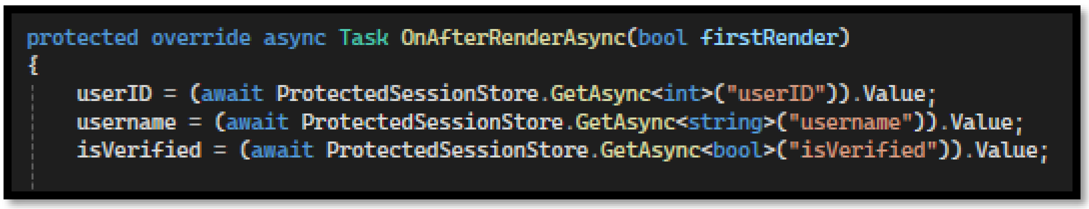
In the initial stages of development, I was using a class that I created named UserContext to manage the login state of the users. This class contained fields such as Username, isLoggedIn and expiryDT. When the user logged in, the UserContext would be set globally for that user. This was useful because it meant that I could just import the single instance of UserContext within each razor file and easily access and modify the user's information, login state and login expiry time - **builder.Services.AddSingleton<UserContext>();**

However, when it came to testing, I realised that I had made a big mistake. I logged into the application on my account on my Windows PC, then tried to log in on my phone to test the application's compatibility. When I accessed the web application on my phone, I was surprised to see that I was already logged in on my phone somehow.

I spent some time troubleshooting and finally realised that due to the way Blazor Server works, all clients are accessing a **single instance of the web application**. This meant that there was only one UserContext class shared between every user of the application. In its current state, there could be only one user logged in at any given time.

To mitigate this issue, I removed the UserContext class in its entirety and replaced it with ProtectedSessionStorage. This stores the username and login state within the scope of the browser window and can be accessed by all razor pages, so a global singleton is not needed. (Microsoft Learn, 2024)

The code below shows the OnAfterRenderAsync method within my Documents .razor page. After the page has rendered for the first time, the userID, username and verification state variables are set from the browser storage.



```
protected override async Task OnAfterRenderAsync(bool firstRender)
{
    userID = (await ProtectedSessionStore.GetAsync<int>("userID")).Value;
    username = (await ProtectedSessionStore.GetAsync<string>("username")).Value;
    isVerified = (await ProtectedSessionStore.GetAsync<bool>("isVerified")).Value;
```

Pipeline Issues [7.5.4]

My initial plan was to implement a continuous integration/deployment pipeline from within Azure DevOps. This can be done by creating a pipeline that builds, tests, and then finally deploys the pre-built application to the Azure App Service.

However, due to restrictions presumably put in place by the University of Brighton Azure admin team, it was not possible for me to create a pipeline in this way.

I researched alternatives and found that within the Azure App Service, there was an area called the Deployment Centre. This area allows for the constant monitoring of a Git branch hosted on Github or Azure Repos. As my application code's repository was hosted on Azure Repos, I could link the master branch with the Deployment Centre and whenever it detected a change to the master branch, the application's code would be built and deployed to the live site.

This worked great for some time, and then suddenly I was hit with build and deploy errors in the Deployment Centre logs whenever I pushed to master.

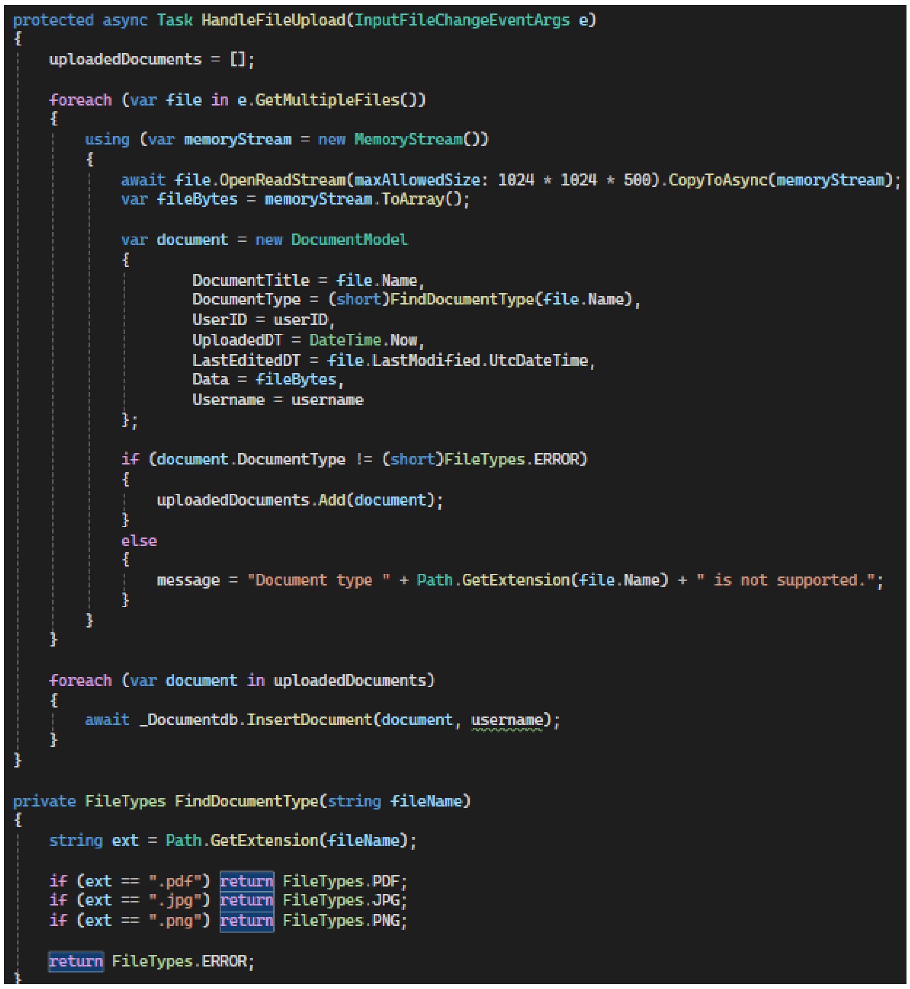
These errors were from the build service, MSBuild-16.4. I spent a considerable amount of time googling the issue, looking for anyone who had the same issue as me. The application worked fine on localhost but was not deploying correctly. After some time, I decided to post a question on the Microsoft Forums to see if anybody could help. I will include this full conversation in my submission folder as it's too large to insert in the appendix.

It turns out that the issue I had found was a bug in the MSBuild-16.4 Microsoft Build engine, making it unable to determine the version of .NET to use to build my project. To fix the issue, I switched my application's host VM from a Windows machine to a Linux machine. The Linux machine uses a different build tool named Oryx Build. This could correctly determine the version and it built and deployed successfully.

Although the post replies didn't directly help me solve the issue, I inserted the solution to the post in the hopes that it may help others in the future.

Handling File Upload [7.5.5]

Although I didn’t experience any major issues during the creation of the file upload process, I wanted to showcase some of the methods I wrote to handle this functionality. The main method concerned with the upload of files is my HandleFileUpload method. This method is asynchronous and returns a Task, this is because async methods that return void cannot be awaited. The area of the below code I wanted to highlight is the use of Enums. FileTypes is an Enum that I created, it was useful when determining if a user uploaded a filetype that was not allowed – the Error Enum would be returned, which makes the code a lot more readable and easier to work with. This was my first time using Enums in my own code and so it was an interesting learning experience.



```
protected async Task HandleFileUpload(InputFileChangeEventArgs e)
{
    uploadedDocuments = [];

    foreach (var file in e.GetMultipleFiles())
    {
        using (var memoryStream = new MemoryStream())
        {
            await file.OpenReadStream(maxAllowedSize: 1024 * 1024 * 500).CopyToAsync(memoryStream);
            var fileBytes = memoryStream.ToArray();

            var document = new DocumentModel
            {
                    DocumentTitle = file.Name,
                    DocumentType = (short)FindDocumentType(file.Name),
                    UserID = userID,
                    UploadedDT = DateTime.Now,
                    LastEditedDT = file.LastModified.UtcDateTime,
                    Data = fileBytes,
                    Username = username
            };

            if (document.DocumentType != (short)FileTypes.ERROR)
            {
                uploadedDocuments.Add(document);
            }
            else
            {
                message = "Document type " + Path.GetExtension(file.Name) + " is not supported.";
            }
        }
    }

    foreach (var document in uploadedDocuments)
    {
        await _Documentdb.InsertDocument(document, username);
    }
}

private FileTypes FindDocumentType(string fileName)
{
    string ext = Path.GetExtension(fileName);

    if (ext == ".pdf") return FileTypes.PDF;
    if (ext == ".jpg") return FileTypes.JPG;
    if (ext == ".png") return FileTypes.PNG;

    return FileTypes.ERROR;
}
```

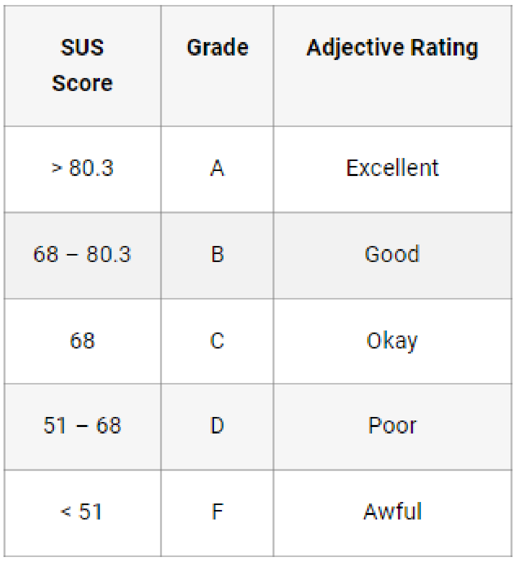
Testing [7.6]

SUS System Usability Scale Score [7.6.1]

To measure my system's UX usability, I opted to use the System Usability Scale.

To measure this, a questionnaire is sent to X amount of people to test the application and answer questions accordingly.

I have included the evidence as a separate file in this submission, but the average usability score from 5 surveyed users was **85.5**. Any score above 80.3 is considered excellent.



SUS Score	Grade	Adjective Rating
> 80.3	A	Excellent
68 – 80.3	B	Good
68	C	Okay
51 – 68	D	Poor
< 51	F	Awful

Self-Testing [7.6.2]

Test No	Scenario	Expected Result	Actual Result	Pass/Fail
1	Log in	User is logged in, and able to access Documents area, log out button appears	User is logged in, and able to access Documents area, log out button appears	PASS
2	Log out	User is logged out, unable to navigate to restricted pages (Documents)	User is logged out, unable to navigate to restricted pages (Documents)	PASS

3	Sign up	When entry valid, sign-up form redirects to log in page	When entry valid, sign-up form redirects to log in page	PASS
4	Upload single document	Choose files button opens file browser, single document is selected and appears in My Documents grid	Choose files button opens file browser, single document is selected and appears in My Documents grid	PASS
5	Upload multiple documents	Choose files button opens file browser, multiple documents are selected and appear in My Documents grid	Choose files button opens file browser, multiple documents are selected and appear in My Documents grid	PASS
6	Delete single document	Select a single document, press delete button, document is removed from My Documents grid	Select a single document, press delete button, document is removed from My Documents grid	PASS
7	Delete multiple documents	Select multiple documents, press delete button, documents are removed from My Documents grid	Select multiple documents, press delete button, documents are removed from My Documents grid	PASS
8	Share single document	Select single document, click share selected and enter email. Document should be sent within 10 seconds and document is added to shared documents grid	Select single document, click share selected and enter email. Document should be sent within 10 seconds and document is added to shared documents grid	PASS
9	Share multiple documents	Select multiple documents, click share selected and enter email. Documents should be sent within 10 seconds and documents are added to shared documents grid	Select multiple documents, click share selected and enter email. Documents should be sent within 10 seconds and documents are added to shared documents grid	PASS
10	Download single document	Select single document, press download button. Download begins in browser	Select single document, press download button. Download begins in browser	PASS
11	Download multiple documents	Select multiple documents, press download button. Download begins in browser	Select multiple documents, press download button. Download begins in browser	PASS
12	Verify account	Verification email is sent on first time login, when verification code is entered, user is redirected to Documents page	Verification email is sent on first time login, when verification code is entered, user is redirected to Documents page	PASS
13	Resend Verification email	Verification email is resent upon button press	Verification email is resent upon button press	PASS
14	Document Type correctly identified on document upload	When document is uploaded, correct file extension is identified and shown in Document Type column	When document is uploaded, correct file extension is identified and shown in Document Type column	PASS

Web Accessibility Evaluation [7.6.3]

To analyse the accessibility of my web application I used the WAVE web access evaluation tool. This tool showed that I had no major accessibility errors, however, there were two contrast errors within the Nav bar. As I've completed the web application, I can't fix these issues now. However, it is something to keep in mind in the future. I believe the issue is to do with the on-hover colour/transparency change within the navigation bar. (Wave, n.d.)

Evaluation [8.0]

End-Product Evaluation [8.1]

Overall, I think that the final product meets the expectations I set out during the initial planning phase. I have created a system that allows for the secure storage of personal documents and allows for the sharing of these documents via email. Although the UI design is simple, it is fully functional, and the usability of the application has been proven through the SUS evaluation that took place during testing.

As previously stated, I would have loved to include multiple sharing methods, and implemented document access expiry dates. However, due to unforeseen email provider restrictions discussed in the Technical Challenges section, it was not possible to include these features in this version of the product.

I am proud of the security features that I implemented, the email verification system, use of HTTPS, encrypted password storage, encrypted document BLOB storage, and sending of documents via encrypted email all contribute to the initial goal of creating a service that is as secure as possible and adheres to GDPR restrictions. The only change I would make in this area is to host a review of the customer data stored, ensuring that data is deleted after a certain period.

Methodology Evaluation [8.2]

The kanban-scrum mixed methodology that I used worked well during development to a certain extent. However, due to utilizing a single sprint format, I was unable to make use of many of the Azure DevOps built-in dashboard graphs. This made it harder to track the progress I was making week-by-week and left me rushing around in the final stages of development. I think that if I had implemented a more traditional scrum methodology, I would have been able to include more features within the final version of my product. This is definitely something I'll need to consider in my future projects.

General Reflection [8.3]

The part of the project I enjoyed the most was the constant problem solving. Although at times the process has been highly challenging, it reinforced to me that I'd enjoy a career in software development. It's a great feeling when you find the solution to a problem that has been plaguing development for multiple days and is a feeling I'll be chasing in future projects.

Throughout the project I surprised myself with how quickly I can learn a new technology when I stick at it. I had no previous experience with Blazor and at the end of the development process I am very confident with its use and feel that I understand the different concepts and techniques required well.

Something I'd like to work on is my project planning and timekeeping. As previously stated, I found it hard to track the timings of the development process. I often found it hard to create a study plan and stick to it diligently. I think the main reason for this is that it was a self-managed project and therefore I had no "manager" personality to apply pressure to me, I only had myself to answer to. I think that this boils down to experience and discipline working by myself and is something I aim to improve by continuing to work on self-managed projects in my own time.

Learning in Modules [8.4]

Throughout the creation of my final year project, I utilized many different skills and techniques that I learned during my time at university. Not only did I learn these skills from the modules I completed, but also from time spent working full time on my placement year.

Some of the key areas to note are:

Web Application Development skills – During my time at university I completed several different Web development modules. Introduction to Web Development (CI435) in year 1, Web Application Development (CI527) in year 2, and Advanced Web Application Development (CI609) in year 3. These modules gave me a wide understanding of the web development process and the knowledge I learned across the modules was vital to the completion of my final year project.

Database skills – Although I did learn some basic SQL skills in year 1 of university, most of my database knowledge was built during my placement year at Infor. I worked across multiple development teams, writing , maintaining, and troubleshooting T-SQL queries. This learning experience helped me greatly during the development of this project as I already knew how to set-up, manage, and write complex Microsoft SQL queries.

Conclusion [9.0]

In conclusion, the development of Doccy was an incredible learning experience for me. I have successfully managed the creation of a software artifact from start to finish, and I am very proud of the final product's functionality and usability. I thoroughly enjoyed the development process and although it was stressful at times due to technical issues and some oversights during the initial planning process, the skills that I have developed because of overcoming these challenges will be carried forward with me throughout my career in the industry.

I intend to continue working on and updating Doccy in the months following the submission of this report, and hope that with the addition of a fileserver to store documents, I can fully implement the document expiry function that I had set out to create at the start of the development process. I have learned many new technologies and improved my development skills so much in such a short space of time thanks to this project. I'm very proud of the progress that I have made across my four years at university, and I'm excited to develop my skills further upon graduating.

Meetings With Supervisor [10.0]

Viva Monday 27th November 2023 - Viva meeting to present interim report to readers.

Viva Feedback 5th December 2023 - Email with word document feedback on interim report, many key areas to improve.

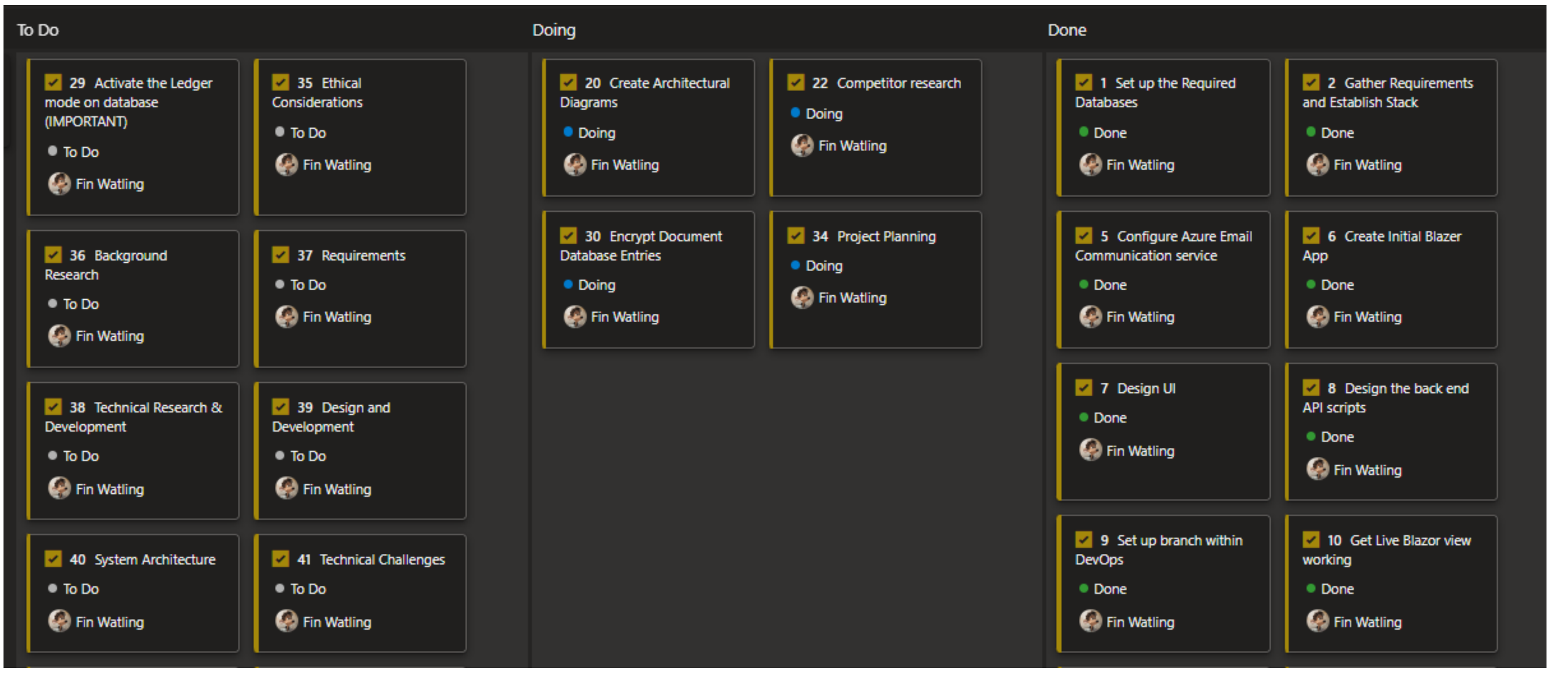
23rd April 2024 - Primary - General questions surrounding changes of requirements.

2nd May 2024 - Primary - Report Contents review and feedback.

Git Repository [11.0]

To allow for easy viewing of my code and other evidence, I have uploaded my submission to Github. It is hosted at: https://github.com/FinWatling/Doccy.

Appendix [1.1]



To Do

- 29 Activate the Ledger mode on database (IMPORTANT) — ● To Do — Fin Watling
- 35 Ethical Considerations — ● To Do — Fin Watling
- 36 Background Research — ● To Do — Fin Watling
- 37 Requirements — ● To Do — Fin Watling
- 38 Technical Research & Development — ● To Do — Fin Watling
- 39 Design and Development — ● To Do — Fin Watling
- 40 System Architecture — ● To Do — Fin Watling
- 41 Technical Challenges — ● To Do — Fin Watling

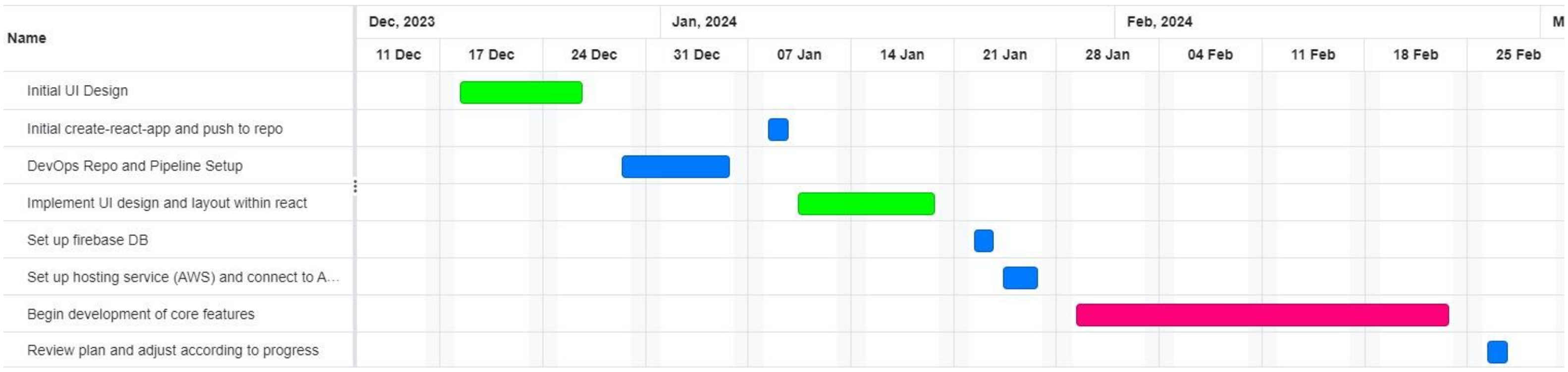
Doing

- 20 Create Architectural Diagrams — ● Doing — Fin Watling
- 22 Competitor research — ● Doing — Fin Watling
- 30 Encrypt Document Database Entries — ● Doing — Fin Watling
- 34 Project Planning — ● Doing — Fin Watling

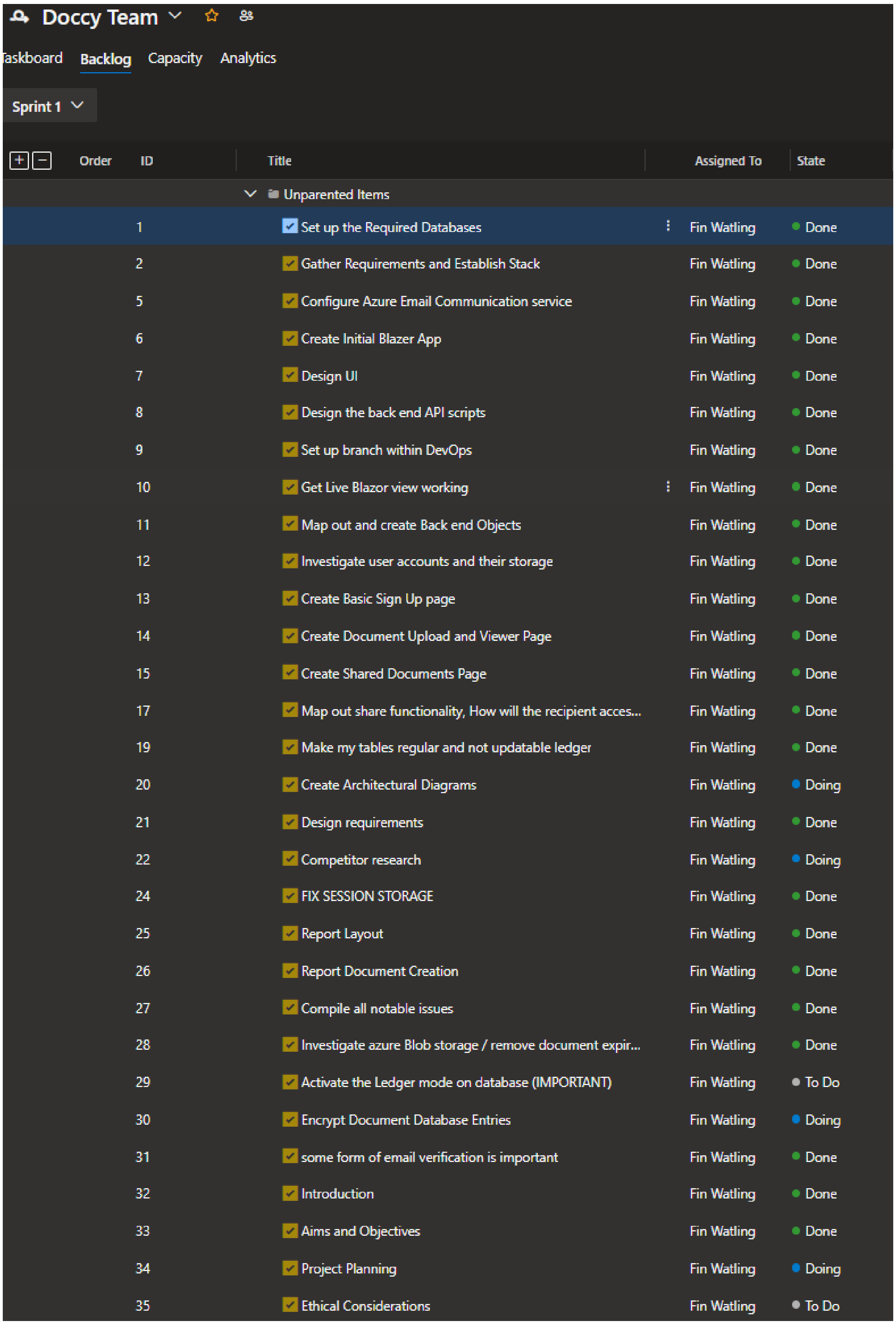
Done

- 1 Set up the Required Databases — ● Done — Fin Watling
- 2 Gather Requirements and Establish Stack — ● Done — Fin Watling
- 5 Configure Azure Email Communication service — ● Done — Fin Watling
- 6 Create Initial Blazer App — ● Done — Fin Watling
- 7 Design UI — ● Done — Fin Watling
- 8 Design the back end API scripts — ● Done — Fin Watling
- 9 Set up branch within DevOps — ● Done — Fin Watling
- 10 Get Live Blazor view working — ● Done — Fin Watling

Appendix [1.2]

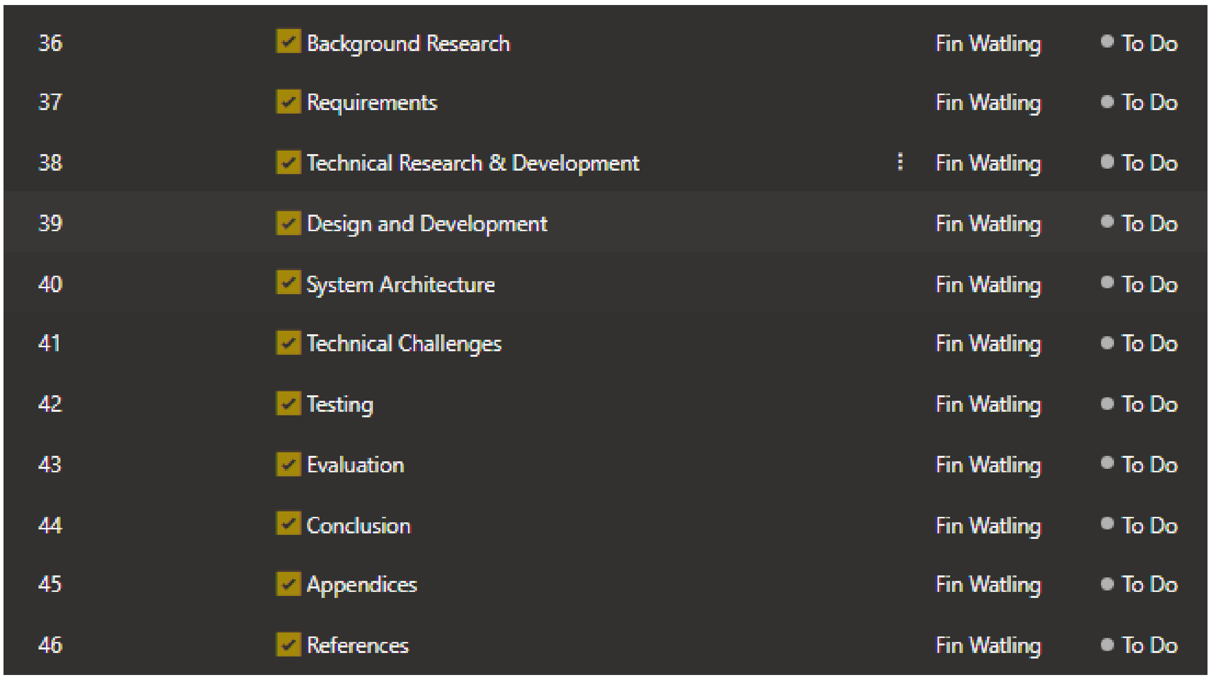


Appendix [1.3]



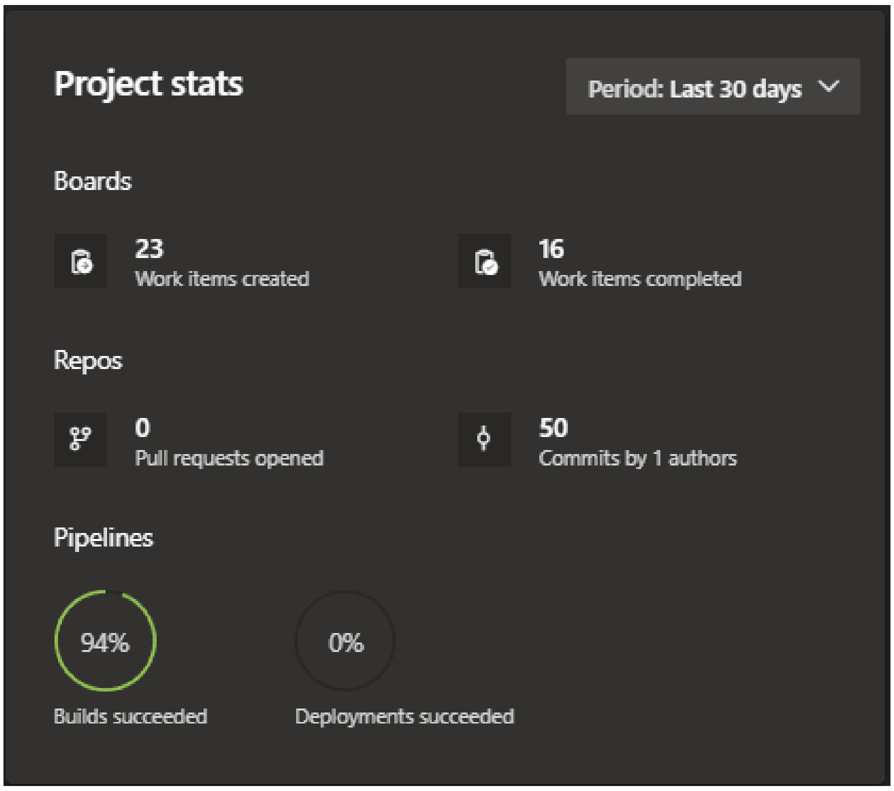
Order	ID	Title	Assigned To	State
		Unparented Items		
	1	Set up the Required Databases	Fin Watling	Done
	2	Gather Requirements and Establish Stack	Fin Watling	Done
	5	Configure Azure Email Communication service	Fin Watling	Done
	6	Create Initial Blazer App	Fin Watling	Done
	7	Design UI	Fin Watling	Done
	8	Design the back end API scripts	Fin Watling	Done
	9	Set up branch within DevOps	Fin Watling	Done
	10	Get Live Blazor view working	Fin Watling	Done
	11	Map out and create Back end Objects	Fin Watling	Done
	12	Investigate user accounts and their storage	Fin Watling	Done
	13	Create Basic Sign Up page	Fin Watling	Done
	14	Create Document Upload and Viewer Page	Fin Watling	Done
	15	Create Shared Documents Page	Fin Watling	Done
	17	Map out share functionality, How will the recipient acces...	Fin Watling	Done
	19	Make my tables regular and not updatable ledger	Fin Watling	Done
	20	Create Architectural Diagrams	Fin Watling	Doing
	21	Design requirements	Fin Watling	Done
	22	Competitor research	Fin Watling	Doing
	24	FIX SESSION STORAGE	Fin Watling	Done
	25	Report Layout	Fin Watling	Done
	26	Report Document Creation	Fin Watling	Done
	27	Compile all notable issues	Fin Watling	Done
	28	Investigate azure Blob storage / remove document expir...	Fin Watling	Done
	29	Activate the Ledger mode on database (IMPORTANT)	Fin Watling	To Do
	30	Encrypt Document Database Entries	Fin Watling	Doing
	31	some form of email verification is important	Fin Watling	Done
	32	Introduction	Fin Watling	Done
	33	Aims and Objectives	Fin Watling	Done
	34	Project Planning	Fin Watling	Doing
	35	Ethical Considerations	Fin Watling	To Do

Appendix [1.4]



36	✓ Background Research	Fin Watling	● To Do
37	✓ Requirements	Fin Watling	● To Do
38	✓ Technical Research & Development	Fin Watling	● To Do
39	✓ Design and Development	Fin Watling	● To Do
40	✓ System Architecture	Fin Watling	● To Do
41	✓ Technical Challenges	Fin Watling	● To Do
42	✓ Testing	Fin Watling	● To Do
43	✓ Evaluation	Fin Watling	● To Do
44	✓ Conclusion	Fin Watling	● To Do
45	✓ Appendices	Fin Watling	● To Do
46	✓ References	Fin Watling	● To Do

Appendix [1.5]



Project stats

Period: Last 30 days

Boards

23 Work items created

16 Work items completed

Repos

0 Pull requests opened

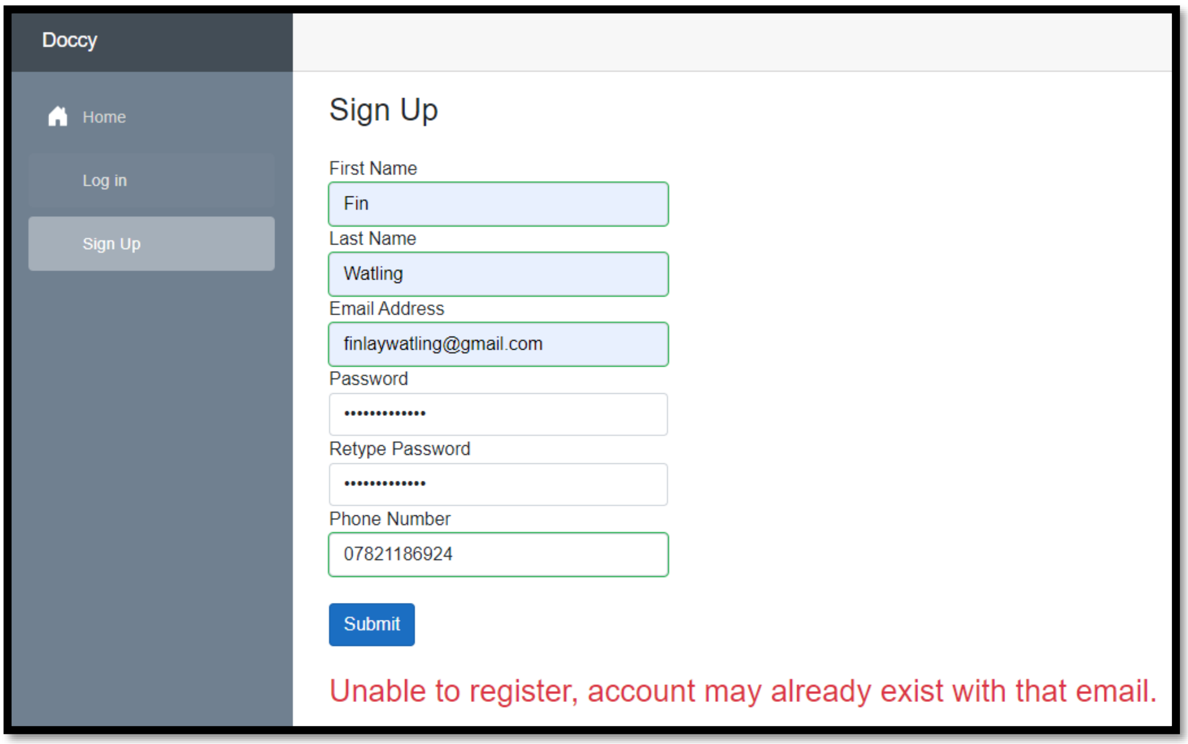
50 Commits by 1 authors

Pipelines

94% Builds succeeded

0% Deployments succeeded

Appendix [2.0]



Sign Up

First Name: Fin

Last Name: Watling

Email Address: finlaywatling@gmail.com

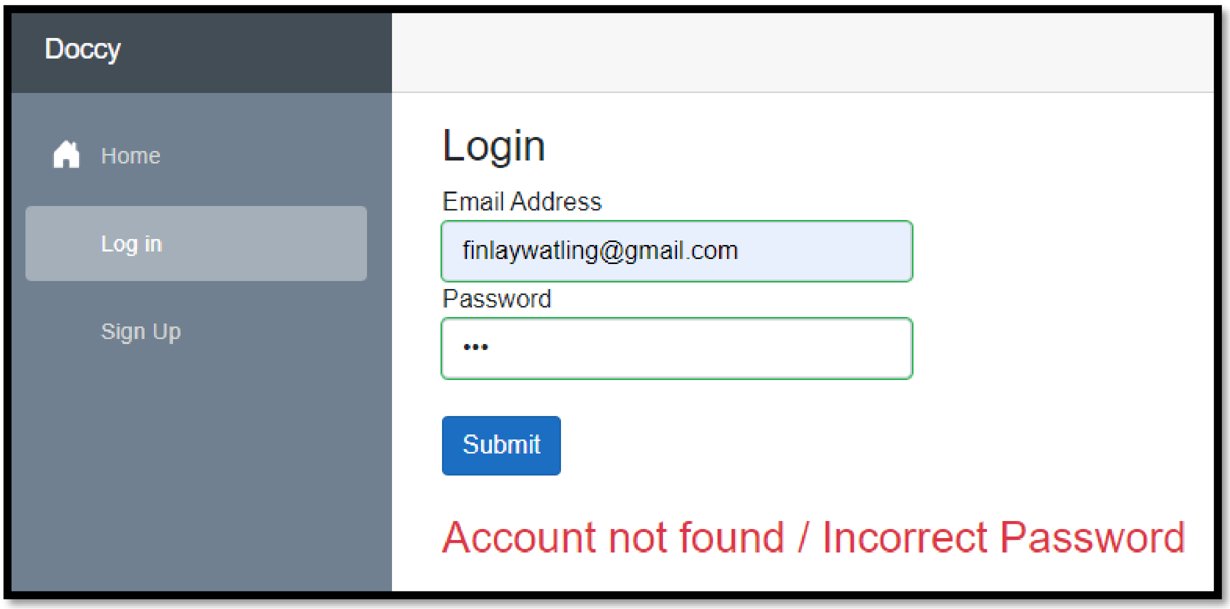
Password: •••••••••••••

Retype Password: •••••••••••••

Phone Number: 07821186924

Unable to register, account may already exist with that email.

Appendix [2.1]



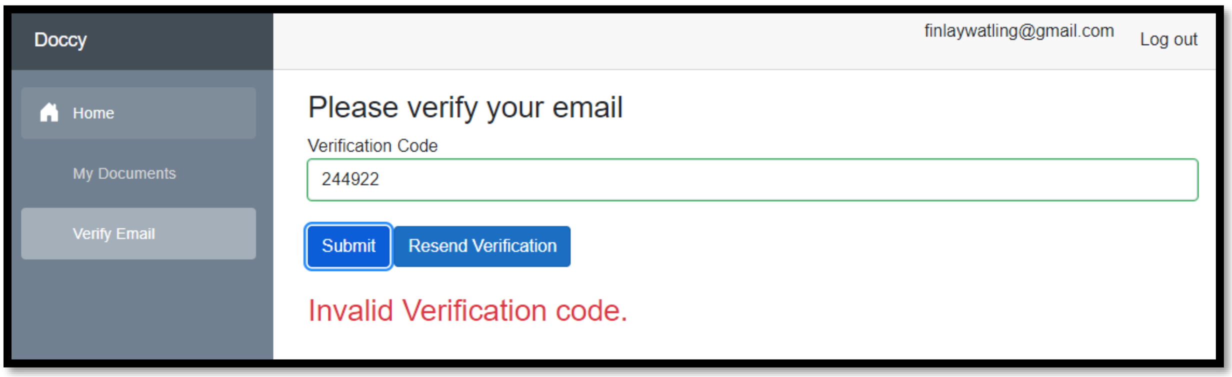
Login

Email Address: finlaywatling@gmail.com

Password: •••

Account not found / Incorrect Password

Appendix [2.2]



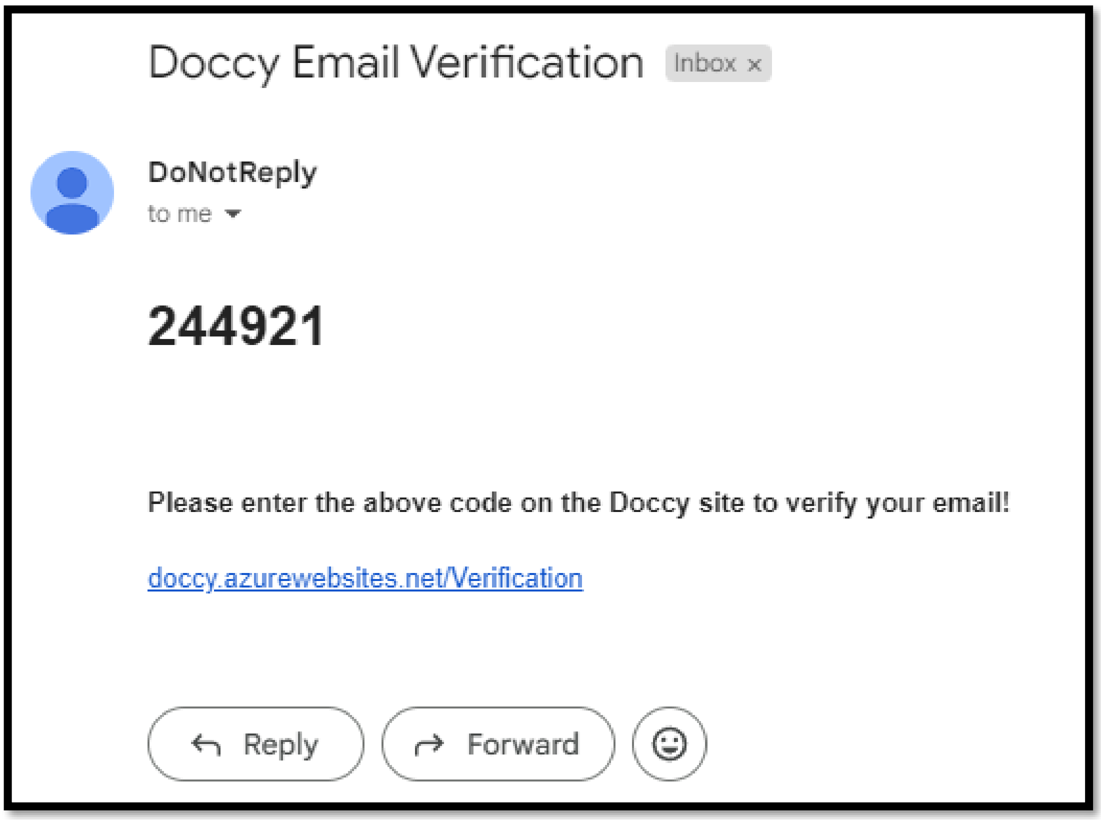
Please verify your email

Verification Code

244922

Invalid Verification code.

Appendix [2.3]



Doccy Email Verification

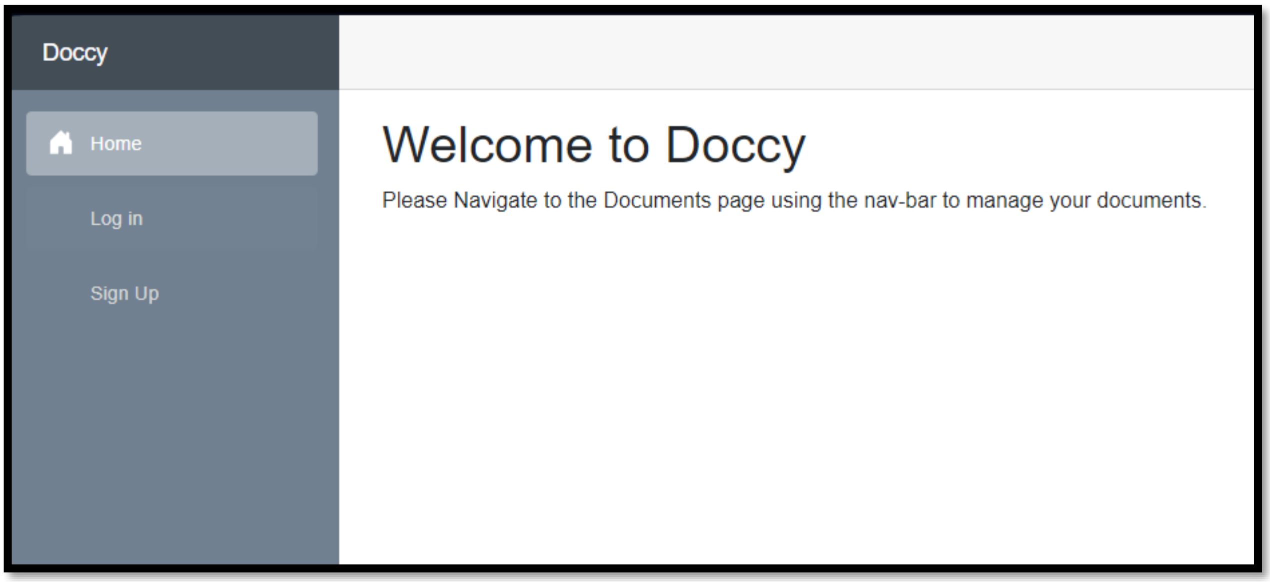
DoNotReply

244921

Please enter the above code on the Doccy site to verify your email!

doccy.azurewebsites.net/Verification

Appendix [2.4]


Doccy

Home

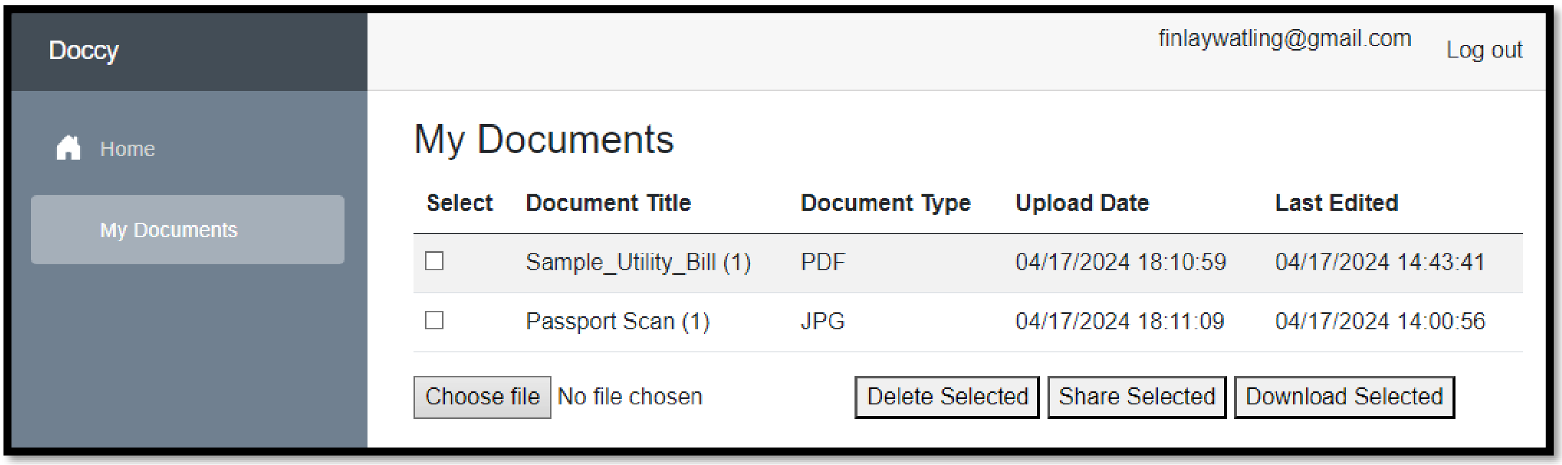
Log in

Sign Up

Welcome to Doccy

Please Navigate to the Documents page using the nav-bar to manage your documents.

Appendix [2.5]


Doccy

Home

My Documents

finlaywatling@gmail.com

Log out

My Documents

Select	Document Title	Document Type	Upload Date	Last Edited
☐	Sample_Utility_Bill (1)	PDF	04/17/2024 18:10:59	04/17/2024 14:43:41
☐	Passport Scan (1)	JPG	04/17/2024 18:11:09	04/17/2024 14:00:56

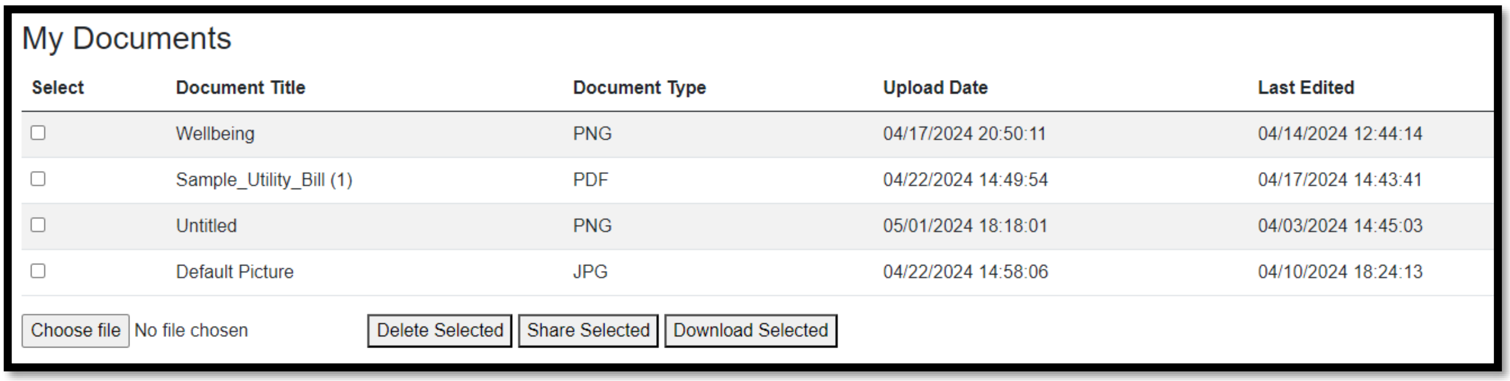
Choose file No file chosen

Delete Selected

Share Selected

Download Selected

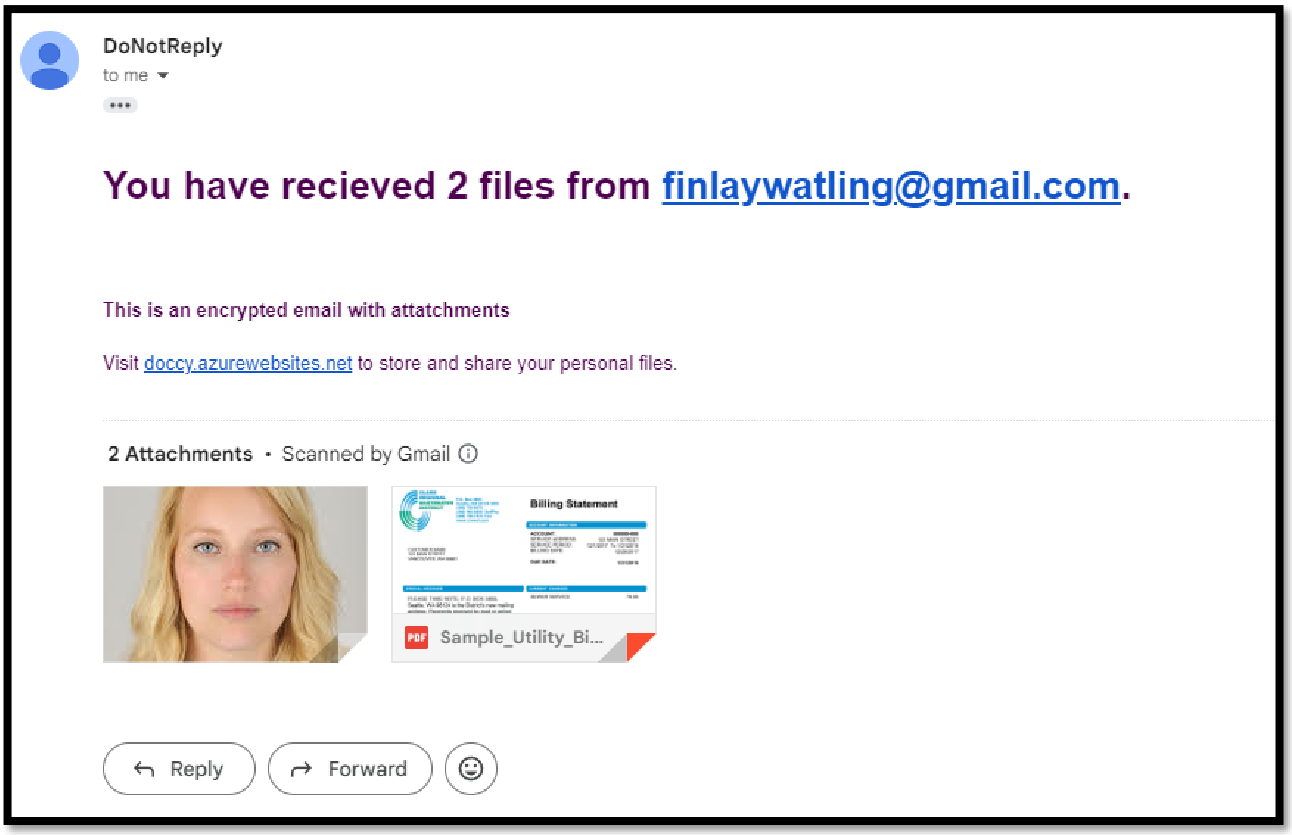
Appendix [2.6]



My Documents

Select	Document Title	Document Type	Upload Date	Last Edited
☐	Wellbeing	PNG	04/17/2024 20:50:11	04/14/2024 12:44:14
☐	Sample_Utility_Bill (1)	PDF	04/22/2024 14:49:54	04/17/2024 14:43:41
☐	Untitled	PNG	05/01/2024 18:18:01	04/03/2024 14:45:03
☐	Default Picture	JPG	04/22/2024 14:58:06	04/10/2024 18:24:13

Appendix [2.7]



DoNotReply
to me

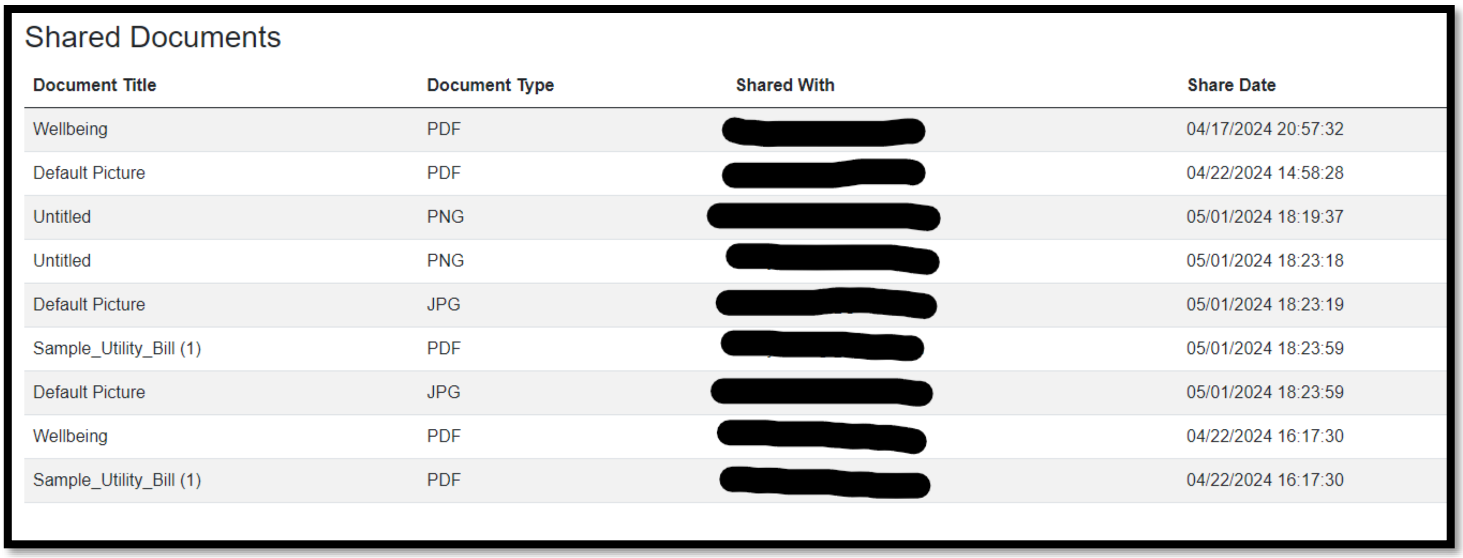
You have recieved 2 files from finlaywatling@gmail.com.

This is an encrypted email with attatchments

Visit doccy.azurewebsites.net to store and share your personal files.

2 Attachments • Scanned by Gmail

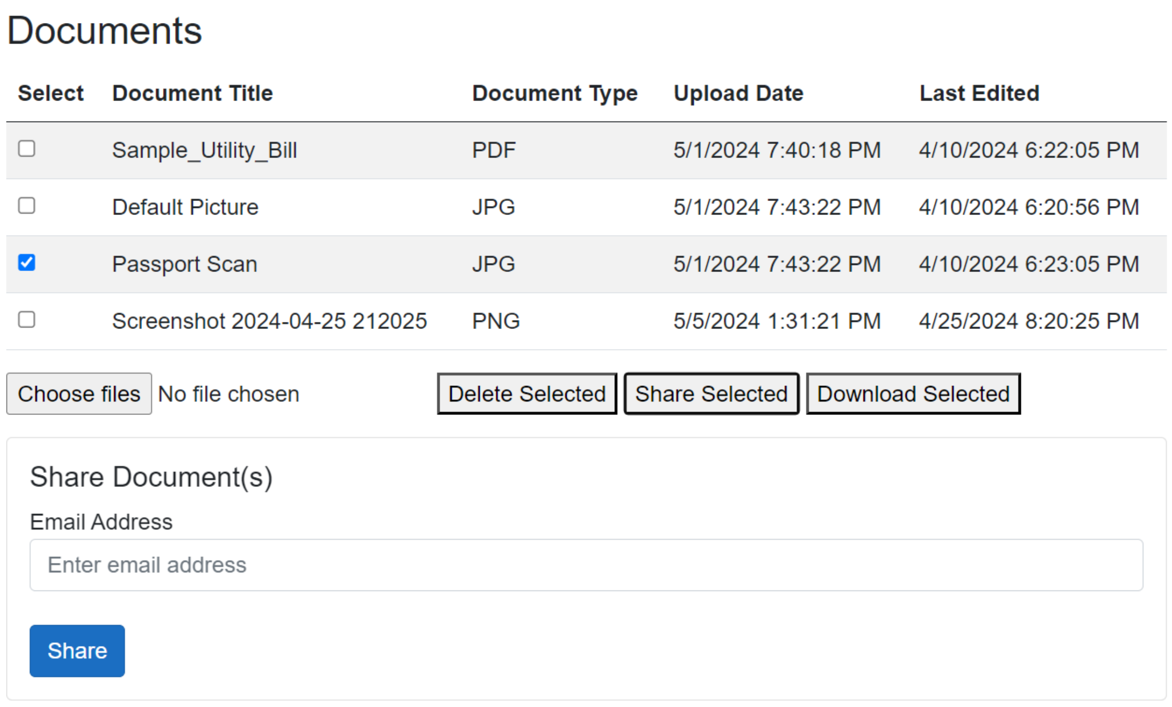
Appendix [2.8]



Shared Documents

Document Title	Document Type	Shared With	Share Date
Wellbeing	PDF	[redacted]	04/17/2024 20:57:32
Default Picture	PDF	[redacted]	04/22/2024 14:58:28
Untitled	PNG	[redacted]	05/01/2024 18:19:37
Untitled	PNG	[redacted]	05/01/2024 18:23:18
Default Picture	JPG	[redacted]	05/01/2024 18:23:19
Sample_Utility_Bill (1)	PDF	[redacted]	05/01/2024 18:23:59
Default Picture	JPG	[redacted]	05/01/2024 18:23:59
Wellbeing	PDF	[redacted]	04/22/2024 16:17:30
Sample_Utility_Bill (1)	PDF	[redacted]	04/22/2024 16:17:30

Appendix [2.9]



Documents

Select	Document Title	Document Type	Upload Date	Last Edited
☐	Sample_Utility_Bill	PDF	5/1/2024 7:40:18 PM	4/10/2024 6:22:05 PM
☐	Default Picture	JPG	5/1/2024 7:43:22 PM	4/10/2024 6:20:56 PM
☑	Passport Scan	JPG	5/1/2024 7:43:22 PM	4/10/2024 6:23:05 PM
☐	Screenshot 2024-04-25 212025	PNG	5/5/2024 1:31:21 PM	4/25/2024 8:20:25 PM

Share Document(s)

Email Address

Enter email address

Link to live site: https://doccy.azurewebsites.net/

Bibliography

Arias, D. (2021, February 25). *Hashing in Action: Understanding bcrypt*. Retrieved from Auth0: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/

BCrypt. (2023, September 20). *bcrypt*. Retrieved from Npm: https://www.npmjs.com/package/bcrypt

Colvin, T. (2021, November 23). *Read this before you use Firebase for sensitive data*. Retrieved from Medium: https://tdcolvin.medium.com/read-this-before-you-use-firebase-for-sensitive-data-797f00db8753

Google. (n.d.). *Firebase*. Retrieved from Firebase: https://firebase.google.com/

Iubenda. (n.d.). *GDPR Data Storage: What Businesses Need to Know*. Retrieved from Iubenda: https://www.iubenda.com/en/help/122437-gdpr-data-storage-what-businesses-need-to-know

Jones, j. (2022, March 9). *How to check and verify file integrity*. Retrieved from TechTarget: https://www.techtarget.com/searchcontentmanagement/tip/How-to-check-and-verify-file-integrity#:~:text=Content%20security%20teams%20can%20validate,movements%20%2D%2D%20or%20unauthorized%20access.

Liendo, M. (2021, October 14). *Send an SMS to Customers Using React and AWS Amplify*. Retrieved from Focus Otter: https://blog.focusotter.com/send-an-sms-to-customers-using-react-and-aws-amplify

Microsoft. (2023, October 20). *About projects and scaling your organization*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/azure/devops/organizations/projects/about-projects?view=azure-devops

Microsoft. (2023, March 1). *Ensuring Data Integrity with Hash Codes*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/dotnet/standard/security/ensuring-data-integrity-with-hash-codes

Microsoft. (2023, April 19). *General Data Protection Regulation Summary*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-gb/compliance/regulatory/gdpr

Microsoft Learn. (2024, January 3). *ASP.NET Core Blazor state management*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/aspnet/core/blazor/state-management?view=aspnetcore-8.0&pivots=server

Microsoft Learn. (2024, April 11). *Host and deploy server-side Blazor apps*. Retrieved from Microsoft Learn: https://learn.microsoft.com/en-us/aspnet/core/blazor/host-and-deploy/server?view=aspnetcore-8.0

Russell Sears, C. v. (2006). *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?* California: Microsoft Research, University of California.

Terehan, M. (2022, July 6). *CI/CD For Blazor Applications In Azure Devops*. Retrieved from C-sharpcorner: https://www.c-sharpcorner.com/article/cicd-for-blazor-applications-in-azure-devops/

Wave. (n.d.). *Wave accesibility evaluation tool*. Retrieved from Wave: https://wave.webaim.org/

Your Europe. (n.d.). *Data Protection under GDPR*. Retrieved from Your Europe: https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_en.htm