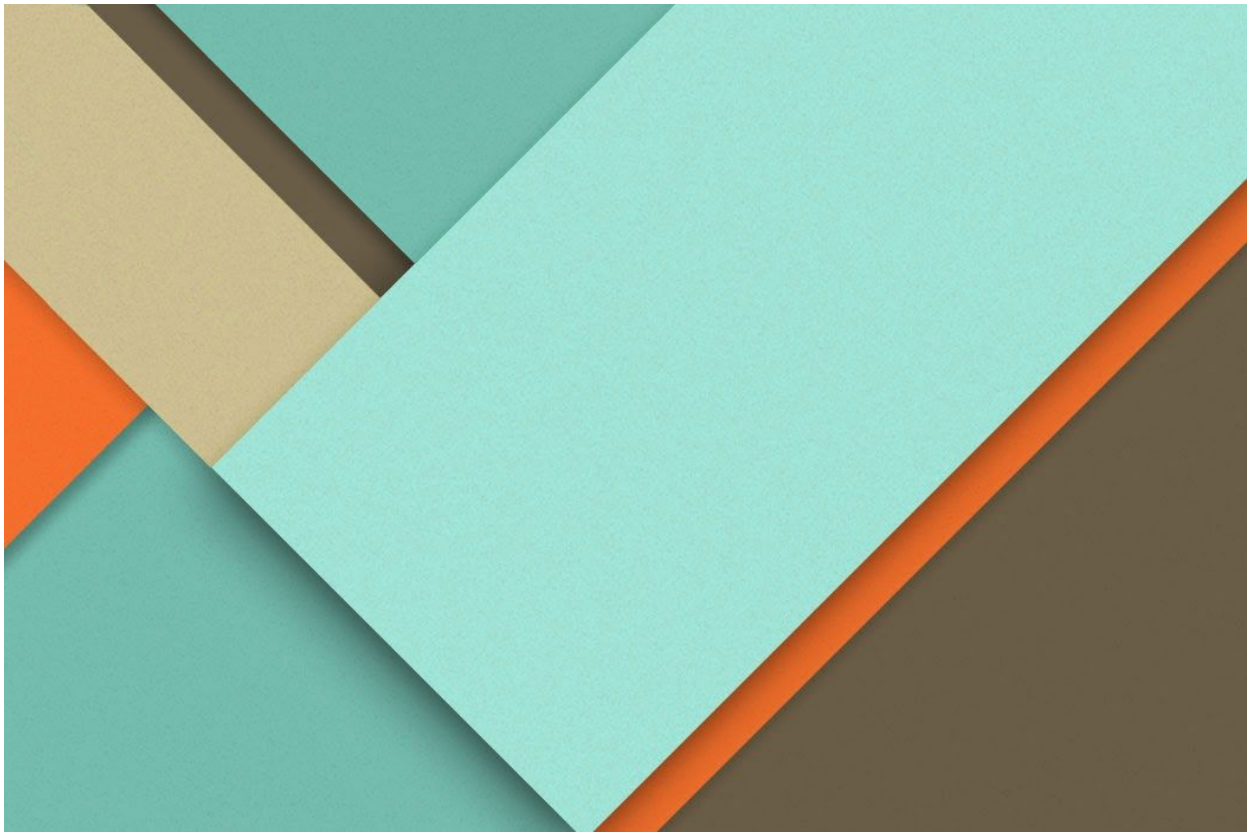


ID Anywhere

Proof of concept personal digital identification system to be integrated with merchant POS terminals.

Lewis Cummins

University of the West of England - UWE

16014766

UFCFR4-45-3

Supervisor: James Lear

I. Acknowledgements

I would like to acknowledge my supervisor James Lear, for all his help and guidance. I would also like to acknowledge my friends and family, for all of their support over the past few years and exhaustive efforts of proof reading my work.

II. Glossary

QR Code - Quick Response Code, holds information in a machine-scannable format. Like a barcode, but QR can store more data.

Minimum Viable Product - the initial version of the final application which is presented in a compact form. (Agarwal, 2018)

1.0 Project Introduction

1.1 Domain

1.1.1 Identification

There are many methods of identification present in modern society. Traditional methods of identification before the digital explosion consisted of a contract held between a person and a government body, in the form of legal documents such as a driver's license, passport or national identity card. All of these will have security measures to prevent counterfeiting and prove authenticity, to show that the bearer of the document is the person represented in the document. These documents would then be checked by another person, such as a cashier at a supermarket when selling age restricted products. This is also used in age restricted venues nationwide to prevent access for persons aged under 18 or 21. They usually check the identification documents of everyone regardless of appearance.

1.1.2 Age Restrictions

Age restrictions are put in place on products and establishments to prevent harm to underage people. These restrictions could be on alcohol or tobacco products, or video games and certain weapons e.g. knives. All of these things have been given an age restriction because experts and research believe that you shouldn't have access to use or consume these items due to the effects they have on the person.

1.2 Aim

The aim of this project is to create a proof of concept workflow that would allow for a safer, more secure and more convenient method of identification for selling age restricted products. This could be expanded to also allowing people into clubs where identification is usually required every time you enter, regardless of how old you may look. This project initially will focus on the United Kingdom, using passports and driving licenses issued by the UK government.

1.3 Structure

An in depth research section will be conducted on the domain and key details will be extracted. For example, security issues with the current ways of identifying people and how to circumvent them, any legal information surrounding identification are two examples. Following that, more technical details surrounding the actual production of the system will

be written up, this will be from looking at current technologies - programming languages, AI frameworks for image and facial recognition, suitable hosting platforms - and comparing the advantages and disadvantages and how they would fit in to the system, as well as any personal preference to technologies through previous use being considered.

Requirements will then be written up and ranked in terms of how important they are to the core functionality of the system, as well as how pertinent they are to ensure a good user experience for all parties involved. These requirements will be the foundation upon which the design is considered and built. Using software design methodologies to provide a high level architecture, with low level components discussed in the following sections.

Implementation will consist of an early prototype with very basic, high level functionality proving that the concept works. Following that, development will commence to produce a minimum viable product based upon necessary requirements, any other requirements that are not necessary to the minimum viable product, but nice to have to round off the software will be implemented.

Finally, testing of all the components will ensure to root out bugs and uncaught issues from development, this will consist of field testing and user acceptance testing.

2.0 Research

2.1 Overview

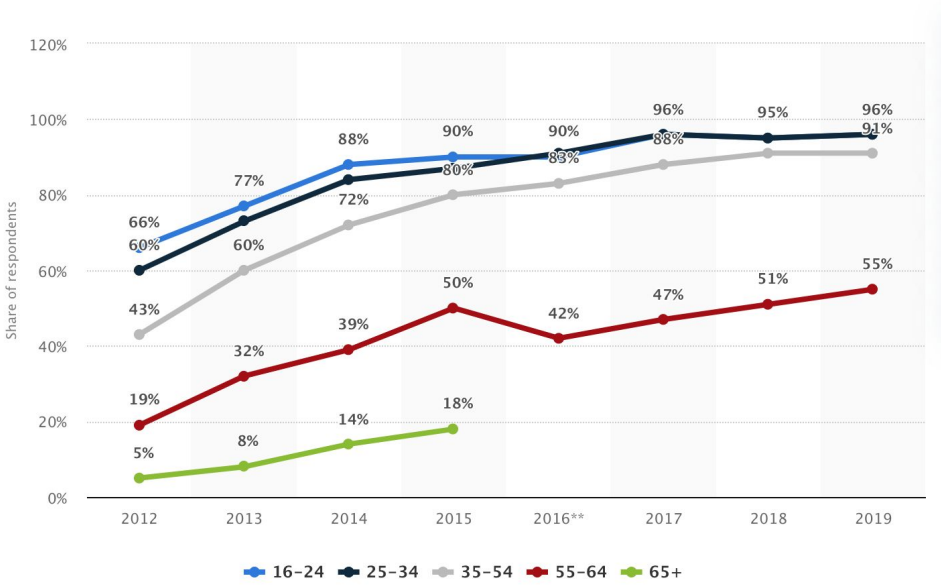
This section will be a culmination of research into various topics that relate to ID Anywhere. It will consist of in-depth discussion and analysis of the domain, current methods of identification, any existing products that relate closely to the application, different technologies and their potential use within the system, development methodologies and documentation processes.

2.2 Merchant Offline Identification

Current methods of identification at the merchant level exist through the consumer presenting their passport or driving license to the cashier. The cashier will examine the document, and, using their knowledge about what constitutes a valid ID, accept or deny the request to allow the consumer to purchase the item. Now, this is how it has always been, and is a pretty simple method to implement and use. It caters to society's need for everything to be quick and easy, especially in a busy supermarket. There are trade offs to using this method. There have been studies on underage people attempting and successfully acquiring alcohol in shops. One study conducted by Morleo *et al.* (2010) in the

North of England looked at the use of fake identification to purchase alcohol amongst 15-16 year olds. Morleo *et al.* (2010) asked these teenagers a list of questions pertaining to alcohol consumption, how they obtain their alcohol and if they have used a fake ID. The statistics showed that over a quarter (28.3%) of 15-16 year old participants who drank reported having bought their own alcohol and of that percentage, 19% had at least one form of fake identification (Morleo *et al.*, 2010). The sample size was 9833 and so, roughly speaking around 1500 of those students had fake identification to purchase alcohol. This is a relatively large amount of young students and is a byproduct of the current identification system not being completely secure and safe.

Fake IDs are extremely easy to obtain. If you were to search 'Fake ID UK' in your browser, chances are the first 5 results would give you very good replicas. If a shop assistant has had a long day and is not paying attention, they could take a quick glimpse at the ID and it may come across as genuine, leading to a young person getting access to age restricted products. There is always room for error with humans, machines too. However, machines are usually better at a task when optimised to do it. Considering this chart below from Statista (2019) using data from a report published by Ofcom, which is currently inaccessible, it shows that 96% of young people ages 16-24 have a smartphone in the United Kingdom.



Details: United Kingdom; Ofcom; 2012 to 2019; 2,675*; 16 years and older; Online survey

Figure 1: Smartphone ownership penetration in the United Kingdom (UK) in 2012-2019, by age (Statista, 2019)

From the studies shown, it is clear to see that the current method of identification for shops has known flaws, and that 96% of the most at risk section of the population have a device that potentially could resolve this problem, it shows that there is potential for digital identification to start becoming the norm in this area of society. This will be discussed more in **section 2.3**.

2.3 Digital Identification

Digital identification is still a fledgling industry. It is the process of establishing who or what something is and any information about them or it through the use of technology. There are many new methods of identifying a person being introduced into society, one that has been implemented in over 40 countries (GOV.UK, 2017) is biometric scanning at airports to get through the border as opposed to handing your passport to a border control officer. It saves a lot of time and ensures that no one can use someone else's passport, nor can they use a fake one. The passport includes a chip with the holder's facial biometric (GOV.UK, 2017). This is then fed into a machine, and your face examined and compared with the biometric data in the passport, also known as facial recognition technology. The use of this technology in passport control has proven to be very successful, however other uses such as in public places has caused controversy when looking at how it affects privacy for individuals and also the security of data stored. This is why it is imperative when it comes to ID Anywhere, that data is stored securely, as dealing with people's identity is a very delicate scenario; in the digital age, there are always people trying to deceive others or infiltrate software systems to obtain data.

2.4 Government Guidelines

The government has guidelines on their website regarding how to correctly check a person's identity (GOV.UK, 2019). They have defined the identity checking process in 5 parts:

1. get evidence of the claimed identity ('strength')
2. check the evidence is genuine or valid ('validity')
3. check the claimed identity has existed over time ('activity')
4. check if the claimed identity is at high risk of identity fraud ('identity fraud')
5. check that the identity belongs to the person who's claiming it ('verification')

Each of these identification steps has corresponding scores associated with them, between 1 and 4. So, for each of the 5 steps listed above, a corresponding score is given. (GOV.UK, 2019) states that how much confidence you have in the identity check depends on:

- how many pieces of evidence you collect
- which parts of the identity checking process you do (1 to 5 above)
- what scores you get for each part of the identity checking process

The combinations provided by the different parts of the identity checking process are known as 'Identity Profiles'. These profiles relate to a level of confidence in the claim of identity:

- low confidence (previously known as 'identity level 1')
- medium confidence (previously known as 'identity level 2')
- high confidence (previously known as 'identity level 3')
- very high confidence (previously known as 'identity level 4')

GOV.UK (2019) states that you should aim to get a higher level of confidence in someone's identity if you or your service are at high risk of identity-related crime. For the purposes of ID anywhere, it would be imperative to obtain at least medium confidence due to the scale of the project, however it would be beneficial to score at high or very high confidence. To do this, the application will have to achieve scores in the 5 identity checking steps that correspond with the profile attached to high or very high confidence.

2.4.1 The 5 steps of Identity Checking

For each of the 5 steps, a score is achieved. The scores work in a way that you can only receive score 2 if all requirements for score 1 have also been achieved.

Note: None of these steps have been written by myself, they are all a part of the document provided and constantly updated by (GOV.UK, 2019). I am simply including them in the report as a point of reference for you to read in a more concise format and to comment on difficulties in incorporating them in an application.

Strength

Some pieces of evidence are stronger than others, meaning they are harder to forge and counterfeit. The strength of the evidence as stated by (GOV.UK, 2019) depends on:

- what security features are on it (for example a hologram or an electronic chip)
- what information it has
- how the person proved their identity to get the evidence

The maximum score that can be issued for strength is a 4.

To achieve score 1 the document must contain 2 of these items:

- the claimed identity's name
- the claimed identity's date of birth
- the claimed identity's place of birth
- the claimed identity's address

- the claimed identity's biometric information (these are measurements of biological or behavioural characteristics, like an iris or fingerprint)
- a photo of the claimed identity
- a reference number

To achieve score 2 the document must achieve score 1 plus:

Include unique information pertaining to:

- the identity
- that piece of evidence

It must also:

- show the person's name instead of any pseudonyms, aliases or nicknames (if the evidence includes a name)
- be protected by physical security features that stop it from being reproduced without specialist knowledge or information (if the evidence is a physical document)
- be protected by cryptographic security features that can correctly identify the organisation that issued it (if the evidence includes digital information)

To achieve score 3 the document must achieve score 2 plus:

- it includes information that's unique to both the identity and that piece of evidence
- the organisation that issued the evidence made sure it was received by the same person who applied for it
- the organisation that issued the evidence checked the person's identity in a way that follows the Money Laundering Regulations 2017 (GOV.UK, 2017)

It must also:

- show the person's 'official' name instead of their initials or synonyms, for example 'Julian' instead of 'Jules' (if the evidence includes a name)
- be protected by physical security features that stop it from being reproduced without specialist equipment (if the evidence is a physical document)

The evidence must also include one of the following:

- a photo of the person
- biometric information that uses cryptographic security features to protect its integrity
- cryptographic security features that can be used to identify the person who owns the evidence (this includes evidence with cryptographic chips and digital accounts that are protected by cryptographic methods)

Finally to achieve score 4 it must achieve score 3 plus:

- it includes biometric information
- all digital information (including biometric information) is protected by cryptographic security features
- the cryptographic security features can prove which organisation issued the evidence
- the organisation that issued the evidence proved the person's identity by comparing and matching the person to an image of the claimed identity from an authoritative source

The difficulties of including these checks into the system include using optical character recognition to compare information presented by the user to validate information is correct. Furthermore, there will need to be a portal present for a user to physically check the provided evidence.

Validity

Some pieces of evidence need to be checked for validity. This is checking that the evidence has not expired or not been reported lost/stolen. The maximum score for this section is 4.

To achieve a score of 1 the person checking the evidence can validate that:

- they're checking an original, certified copy or scan of the evidence
- there are no errors on the evidence, like wrong paper type, spelling mistakes, irregular use of fonts or missing pages
- the details, layout or alignment of the evidence look the way they should
- any logos look the way they should
- any references to information are the same across the evidence (for example if the body text of a letter references an address, this should match the address shown at the top of the letter)

To achieve score 2, score one must be achieved plus the person checking the ID must do one of these 3:

- confirm the evidence is valid
- confirm the visible security features are genuine (these are security features that can be seen without using specialist light sources)
- confirm the ultraviolet (UV) or infrared (IR) security features are genuine

To Achieve score 3, score 2 must be achieved plus the person checking the ID must complete all of the following

- confirm the evidence is valid or check the evidence has not been cancelled, lost or stolen

- confirm the visible security features are genuine
- confirm the UV or IR security features are genuine
- check the evidence has not expired

Finally, to achieve score 4, everything in score 3 must be achieved along with:

- confirm the visible security features are genuine
- confirm the UV or IR security features are genuine
- confirm the cryptographic security features on the evidence are genuine
- check the evidence has not expired
- check the evidence has not been cancelled, lost or stolen

From the above, it is noticeable that the steps are quite similar, the more of the ID you check, the more you can confirm it is valid. There are additional parts to this that make it more difficult to achieve a score of 3 or higher, which may affect the level of confidence the application is able to achieve, however as this is a proof of concept project it does not matter too much. Anyone using the system would also need to have training on how to successfully validate ID documents. This will also require a portal to view the provided documentation.

Activity

This is the process of checking that the ID has interacted with other establishments, proving it has a history attached to it and is not an ID conjured out of thin air. This part of the process does not have to take place in all systems. This has been excluded from this report as it does not seem necessary to ID Anywhere, and an explanation to this is provided in section **2.4.2**.

Identity Fraud

This is a process of making sure that the claimed identity is not of someone that could be considered high risk. This could be anyone in the public eye, such as a celebrity or politician. The maximum score for this section is 3.

To achieve score 1 you must use an authoritative source to check if the claimed identity has either:

- had its details stolen (even if those details have not been used fraudulently yet)
- been reported as stolen

If this has happened, you must conduct additional checks against the ID. You must also check to see if the ID is a known fraudulent ID. If it is, additional evidence must be requested.

To achieve score 2 you must achieve score one plus:

- You must also use an authoritative source to check that the claimed identity:
- belongs to someone who's still alive
- is known by an organisation that should have a record of that person (for example an Electoral Registration Office in a local authority)
- is at a usual risk of being impersonated (for example, a 'politically exposed person' like a politician or judge is at a higher than usual risk of being impersonated)

If any of these are true, additional checks must be carried out.

To achieve score 3 you must achieve score two plus use additional authoritative sources to validate this information.

Verification

The final check. This is ensuring that the person going through the identity checking person is the same person as what the identity reflects. The highest score for this section is a 4.

To achieve score 1 you must:

Get the person to give you answers to questions that do not change over time. These are known as Knowledge Based Verification questions (KBV hereafter).

They come in strengths of low, medium and high. The number you ask depends on the strength of the questions. You must ask one of the following to achieve a score of one:

- 2 low quality KBV challenges
- 4 low quality multiple choice KBV challenges
- 1 medium quality KBV challenge
- 2 medium quality multiple choice KBV challenges
- 1 high quality KBV challenge

Some examples of low strength are:

- about the claimed identity
- clear and simple so the person knows exactly what you're asking them
- about something the claimed identity can reasonably be expected to know

An example of medium strength must contain low strength qualities plus:

- be based on information from a source that did its own identity checks on the claimed identity
- share codes, similar to a one-time password sent to the claimed identity's phone, in a way that means you can be sure they were given to the claimed identity (if you use them)

An example of high strength must contain medium strength qualities plus information must be based on:

- from a source that checked the claimed identity was who they said they were in a way that follows the Money Laundering Regulations 2017 (GOV.UK, 2017)
- from a source that makes sure the information cannot be accessed, modified or created by its employees
- from a source that's separate from your organisation
- from a source that's regulated by a statutory or independent body
- that cannot be known or accessed by anyone apart from the claimed identity and their immediate family without breaking the law (for example, you should not use the information that you know is available on the dark web)

To achieve score 2, you must achieve score 1 plus do one of the following:

- make sure the person physically matches the photo on or associated with the strongest piece of genuine evidence you have of the claimed identity (you can do this in person or remotely)
- make sure the person's biometric information (such as their face or fingerprints) matches biometric information from (or associated with) the strongest piece of genuine evidence you have of the claimed identity
- ask the person to complete multiple 'dynamic' KBV challenges that only the claimed identity should be able to do

To achieve a score of 3, you must achieve score 2 plus one of the following:

- make sure they physically match the photo on (or associated with) the strongest piece of genuine evidence you have of the claimed identity
- make sure their biometric information matches biometric information from (or associated with) the strongest piece of genuine evidence you have of the claimed identity

This shows that if you can successfully use facial recognition to match the person's face to the one on the passport or driving license, you can get a score of 3. I will implement this in ID Anywhere. The guidelines and legislation are extremely verbose. This is a good thing, as it protects people from having their identities hijacked. But, it does pose further complexity to the application, with red tape and barriers to entry of devising an architecture that correctly handles appropriate functions to achieve the desired confidence level by fulfilling an identity profile.

2.4.2 Identity Profiles

Of the 4 identity profiles, there is no point looking into the first 1 (Activity), as it does not achieve our desired confidence level in our users identity.

Depending on the amount of evidence collected during the check, that is, how many documents you are using to verify the person, there are different profiles within the 4

confidence brackets that you can match up to. For the purposes of this application, it makes sense to collect 2 pieces of evidence, a driving license and passport. For a high confidence rating and collecting 2 pieces of evidence, there are 3 profiles that you can match up too. A very high confidence rating has 2 profiles you can match. For the purposes of research, only the most valid profiles will be included, differences compared to explain why it is most valid. The unused profiles will be attached to the appendix. The scores needed using 2 pieces of evidence for high and very high confidence are:

High Confidence - 2 Pieces of Evidence

Check	Score (1st Evidence)	Score (2nd Evidence)
Strength	4	3
Validity	3	3

Check	Score
Activity	Not Needed
Identity Fraud	2
Verification	3

GOV.UK (2019) say that by meeting this identity profile, you will:

- have very strong evidence that shows the claimed identity exists
- have another piece of strong evidence that shows the claimed identity exists
- know both pieces of evidence are genuine and valid
- have made sure you or your service have reduced the risks of any known identity fraud associated with the claimed identity
- be confident the person matches either the photo or biometric information that's shown on the evidence

The other 2 potential profiles seemed to be less intuitive to match in terms of this application. The first one *(Appendix Item A)* had a score of 3 instead of 4 for the first evidence strength check, but included a 3 for activity. The reasoning behind this is if you cannot get the highest score for the strength of 1 of your pieces of evidence, you cannot say that the activity of the individual has fully been checked. This means to check if the ID has been used elsewhere and exists assuredly. If you can get a 4 you are saying that one of the pieces of evidence is completely verified, and that the governing body that issued the

ID would have done their due diligence on the individual to prove that the ID exists in the real world. This is what I need to achieve for ID Anywhere as it should be more focussed on verifying existing documents than doing background activity checks, it detracts from the nature of the application and also It wouldn't be able to do a better job than the government. The second one *(Appendix Item B)*, had a score of 2 for both strength and validity of the 2nd piece of evidence. But had a 2 for activity as opposed to Not Needed. Following a similar argument for the first outcasted profile, If a full strength check can be carried out, then there is no need to check activity and therefore removes redundant operation on the applications part.

Very High Confidence - 2 Pieces of Evidence

Check	Score (1st Evidence)	Score (2nd Evidence)
Strength	4	4
Validity	4	4

Check	Score
Activity	Not Needed
Identity Fraud	2
Verification	4

GOV.UK (2019) state that by meeting this identity profile, you will:

- have more than one piece of very strong evidence that shows the claimed identity exists
- know the evidence is genuine and valid
- have made sure you or your service have reduced the risks of any known identity fraud associated with the claimed identity
- be confident the person's biometric information matches what's shown on the evidence

There is only 1 other profile *(Appendix Item C)* that can be matched for very high confidence and 2 pieces of evidence. The difference is that it allows a 3 for strength and validity in the second piece of evidence but you must now complete an activity check to a score of 2 and Identity Fraud check to a score of 3. These are unnecessary steps that others that have

issued the id's being used as evidence would have undertaken already and completed to a high standard.

2.5 Existing Products

2.5.1 Yoti

Figure 2: Yoti QR code for merchants (Yoti, n.d.)

There are not many digital identification systems out there free and available to use. The only similar system to ID Anywhere is Yoti, which also uses the same basic concepts. Yoti is a software system that deals with loads of different aspects of Identity and QR code ticketing (Yoti). The general workflow for their identification is a supermarket or local retailer signs up to the programme and has their own unique QR code. When people that need to have their age verified try to make a purchase, they are asked to scan the merchant's QR code with their Yoti app. This then shows the merchant they are either above or below the required age for this particular request. However, they do mention on their website that this cannot be used for alcohol.

The concept of ID Anywhere provides the advantage of each user having their own QR code as a representation of their identity. A disadvantage of the approach Yoti has taken is the ability to forge what is displayed on the screen. Their way of combating this is something

called Liveness anti-spoofing. Yoti describe it as “users prove they’re a real person by taking a scan of their face in a robust, anti-spoofing liveness test, passing a liveness test gives us high confidence that we’ve captured an image of an actual face in real time, not an automated bot or spoof through a mask or photograph.” This is a brilliant feature, not just in terms of technical achievement, but if we look at the government regulations surrounding the 5 steps of identity checking (**2.4.1)**, It covers the verification check not just at the point of signing up, but continuously. This aids in preventing ID theft and also proving interaction with multiple establishments.

2.5.2 Summary

Yoti is a very similar app, already well established and developed. The advantage of finding a product similar to the concept of ID Anywhere is a good thing.the good and bad points can be extracted from their system and used during the design and implementation. For example, their anti-forgery measures are impressive in the application. The idea of having the QR code on the user’s phone is a better approach because it provides a more portable means of identification.

2.6 Methodologies

2.6.1 Agile and Scrum

Figure 3: Agile development lifecycle (Hoek, 2018)

Agile as a methodology of work, is used in many industries; most notably software engineering. The agile manifesto, first published in 2001 by Beck *et al.* (2001), defines 12 principles to be followed. These principals are extremely lightweight and produce a methodology of continuous and constant updates to stakeholders, with regular update cycles for review and a constantly underlying focus of self-management. It is a methodology developed for the modern age, in a time where technology is constantly evolving and requirements may be different one week from the next. The general ethos of agile is ‘less is more’, as given by one of the principals “Simplicity--the art of maximizing the amount of work not done--is essential.” (Beck *et al.*, 2001). The most common form of agile is Scrum, where each software release goes through a full cycle of typically 2-4 weeks (usually called Sprints) where everything from planning and design, to development and testing and finally documentation is written. This allows the team to progress in smaller steps when

compared with other, more granular and process oriented development systems (such as Waterfall, discussed in **2.6.3**). This may produce issues, though. If you are not documenting everything all the time, some functionality may get lost or not be completely updated for the end of a sprint.

2.6.2 Kanban

Kanban is an extension of the agile methodology that, like Scrum, focuses on just-in-time delivery of functionality and managing the amount of work in progress (WIP) (Inflectra.com). Rather than focussing on small sprints to get work done iteratively per release, it does it in an incremental process, where a board is used to manage the current state of features in relation to the software development lifecycle. I think this methodology is a good extension of agile, especially for this kind of project. The reason behind this is that I can have a complete board set up for each functional requirement (see section **3.4**) in my work space, and ensure no blocking ensues. It also allows for changes to be made to a product mid stream, whereas traditional methodologies of agile (Scrum) this would be strongly discouraged. With any type of agile methodology though, there are drawbacks. With Scrum, there is a set time limit for functionality. You know how long you have to complete the work. Kanban does not use specific time limits, so if you are not managing your time correctly, a backlog in the pipeline could occur that would then lead to unfinished work leading to delayed releases.

Example Kanban Board (HIVE, 2020)

2.6.3 Waterfall

The waterfall methodology is an old, outdated but nonetheless thorough methodology of software development that focuses on a top to bottom method of development. Sharma (2016) provides a good explanation of the process, "The whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. This means that any phase in the development process begins only if the previous phase is complete." It allows for a lot of control at each stage, so everything is defined and ready for the next stage to begin. Take the design stage, all designs for the entire system will be created here, and be fed to the development team once complete. This is a good thing as at least the development team has a complete idea of the system. However, what if the design is wrong or what if the requirement that was defined that led to this design was wrong? It causes a backpropagation of error potentially leading to the way the customer explained the feature. The common consensus across developers do not think that, in this day and age, the waterfall method is a valid methodology for software development.

2.7 Technology

2.7.1 C#

C# is a general object-oriented programming (OOP) language for networking and Web development. C# is specified as a common language infrastructure (CLI) language. It provides static typing, and a number of frameworks to develop online web applications and APIs. One issue with C# is that it could only ever be compiled down to a program executable on Windows. With the introduction of .NET Core, a new OpenSource framework developed by microsoft, the main issue with C# has now been remedied, which makes it a viable competitor in the realm of portable applications (Which has always been dominated by Java). While C# is not as fast as other programming languages such as C or C++, it makes up for it in a higher level approach to developing and scaling large web projects, through OOP and existing frameworks, as well as being integrated with microsoft cloud computing platform, Azure **(2.7.4)**. This allows for quick and easy deployment to a managed hosted environment.

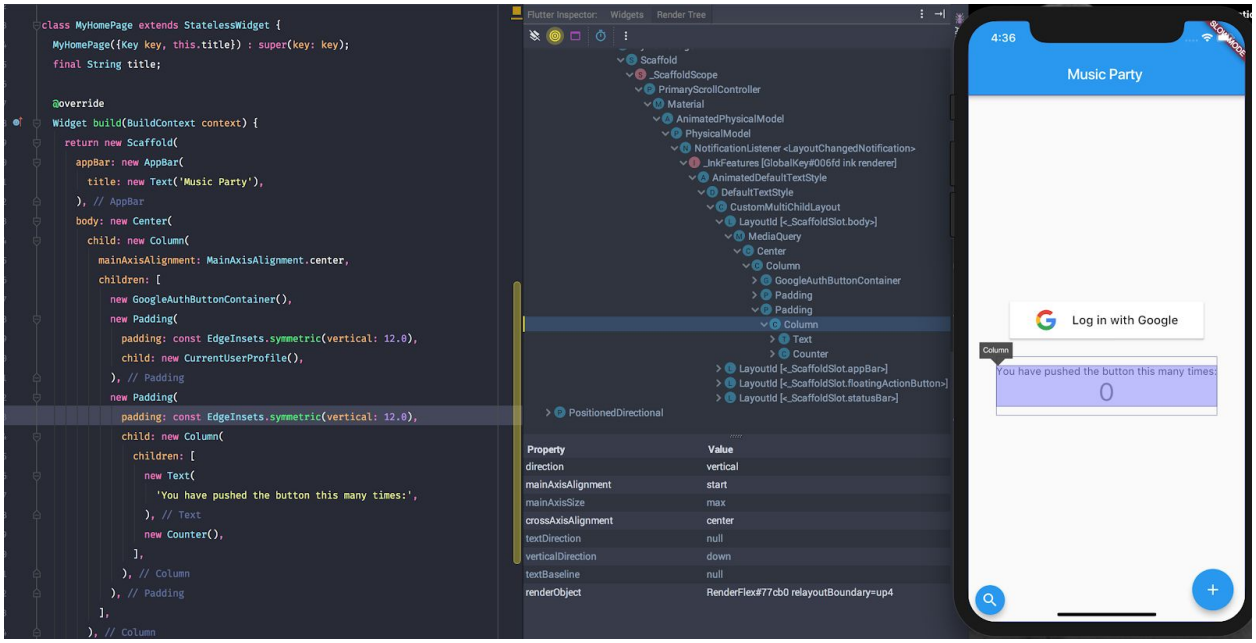
2.7.2 Python

From the official Python website (Python.org), Python is defined as an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level builtin data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. It is extremely versatile and small prototypes can be written up extremely quickly. It is quickly becoming the most used language in the world, with Google adopting it as their primary language. A conducted survey produced by Stack Overflow (2019) showed that Python is the second most loved

and fourth most used across the sample of developers. It is interpreted rather than compiled; rather than building the project into an executable, the scripts are run line by line, and errors only occur when they are hit. As well as interpreted, it has dynamic semantics. This means, the concept of a variable having a type does not exist. Types do, but any type can be stored into a variable or passed to a function. The main disadvantage to using an interpreted, dynamically typed programming language is that it is very easy to make a mistake that will hinder the program at some point. This will not be caught until this part of the program is run. A type safe language such as C# would catch any type discrepancies during compilation.

2.7.3 Flutter (Dart)

Flutter is a mobile development framework developed for the language Dart. Dart is a programming language like C#, however, depending on the target platform, has multiple methods of compilation in comparison to C#'s JIT compilation. From Dart's official website, it explains these methods for programs targeting devices (mobile, desktop, server, and more), Dart Native includes both a Dart VM with JIT (just-in-time) compilation and an AOT (ahead-of-time) compiler for producing machine code (Dart.dev, 2020). Flutter, is a framework that allows for the easy production and design of mobile applications as well as deployment. You can compile down to both Android and IOS from the same source code, which is incredibly versatile when it comes to mobile development. Usually, you would have 2 separate sources for the separate platforms if developing using the native toolkit for both operating systems. Another feature of Flutter that makes it preferential for mobile development is that Flutter considers everything a Widget. When designing wireframes, the inheritance model present in the structure of the code as well as the view displayed on a device is unparalleled when it comes to mobile design and development.



Flutter Inheritance Structure - (Flutterbyexample.com, 2018)

2.7.4 Microsoft Azure

Hosting may be necessary in ID Anywhere for any web service or API. Microsoft Azure is Microsoft's cloud computing platform. It provides a wide range of functionality from hosting web pages to machine learning. Hosting through a cloud computing provider is a common choice in the software sector, as they provide all the infrastructure at a relatively low cost. Also, you have dedicated 24 hour servers and backups. Horizontal and vertical scaling is all managed for you too. All of these features are amazing advantages to cloud hosting. On the flip side, you do not have complete control over your hardware than if you were to self-host or rent servers. For certain applications, especially hardware centric ones, then the cloud computing route may not be where you would want to go. However, for most general purpose websites and web services, it is a perfect technology to incorporate into your repertoire.

2.7.5 Firebase

Firebase is described as a "comprehensive app development platform" by Google. (2020). It includes many features including realtime database storage, image and file storage, crash analytics, a machine learning toolkit and more. It is most suitably described as a 'Backend as a Service'. By using Firebase, you abstract away a lot of common functionality such as authentication. It also eradicates the need for manually setting up a data store. This removes initial technical debt for the startup of a project which allows the developer to focus on the development of core business functionality right at the start. Firebase can be integrated into a project in two ways. Firstly, a standard library of common functionality

used to interact with firebase is provided for most commonly used languages such as Python, Java and C Sharp, as well as Google's own languages Go and Dart. Secondly, there is a REST API that communicates with each project individually.

2.7.6 QR Codes

A QR ("quick response") code is a two dimensional barcode invented by the Japanese corporation Denso Wave (Kieseberg *et al.*, 2010). Information is encoded in both the vertical and horizontal direction, thus holding up to several hundred times more data than a traditional bar code. A typical barcode can only hold a maximum of 20 digits, whereas a QR Code can hold up to 7,089 characters (Kieseberg *et al.*, 2010). This is useful, as it can store a vast amount of data. This could be a GUID (Globally Unique Identifier) or a pattern of strings where each part of the string represents a different part of information, like a MRZ (Machine Readable Zone) on a Passport. An issue that presents itself with this is that QR codes can be read by anyone and the text interpreted by anyone. In order to secure the text, it would need to be encrypted and decrypted elsewhere to avoid malicious parties being able to understand your ID system.

2.7.7 MongoDB

MongoDB is a NoSQL data store that can be used to store non-relational data. MongoDB. (2020) states that the technology is used by millions of developers and companies such as Facebook and Google. It encourages the developer to store data in documents and objects rather than relational database tables. It appeals to the widespread use of object oriented programming and making a data store an extension of the code, rather than a separate entity. It is a very good tool for storing small amounts of data. Another use case could be for the extraction of session when a system is deployed in a load balanced environment where a user's session must be accessible from all instances of the system.

2.7.8 Microsoft SQL Server

Microsoft SQL Server is a relational database server that allows for the creation of SQL based databases Microsoft. (2019). SQL server is a licensed application, which means that it has a cost to its usage unless you are a student. The main benefit of using it is if most of your development relies heavily on the Microsoft technology stack. It integrates with their Azure cloud hosting platform seamlessly with dedicated resources and quick startup. It can scale to monolithic applications. It also comes with Apache Spark and Hadoop file storage Microsoft. (2020). These two technologies are used for big data applications. Clusters of data are stored on multiple Hadoop servers, and Spark allows for queries to be made against this data which can then be aggregated down into tables stored in a Sql Server database.

2.8 Documentation

2.8.1 Volere

The volere model of documentation and requirement building is a standard for precisely documenting and extracting system requirements. The template, as defined by Robertson and Robertson (2012) and has 4 main sections, with subsections underneath.

1. **Project Constraints**
    a. The Purpose of the Project
    b. The Stakeholders
    c. Solution Constraints
    d. Naming conventions
    e. Relevant Facts and Assumptions
2. **Functional Requirements**
    a. List of Requirements
    b. Rationale of Requirements
3. **Non-functional Requirements**
    a. Look and Feel Requirements
    b. Usability Requirements
    c. Performance Requirements
    d. Operational and Environmental Requirements
    e. Maintainability and Support Requirements
    f. Security Requirements
    g. Cultural Requirements
    h. Legal Requirements
4. **Project Issues**
    a. Open Issues
    b. Tasks
    c. Risks

2.10 Summary

Everything researched in this section may prove vitally useful to the design and development of ID Anywhere. Drawing on previously developed systems to help show what a successful system in the market should aspire to be like, such as Yoti. I will outline the requirements in the next section; drawing on the Volere template of requirements engineering. It is necessary to use a method such as Volere, as it is a revised and professionally accepted method of extracting requirements from a problem domain. Using this will allow me to go into a higher level of detail than if I were to not use it at all. Looking at the methodologies researched (Agile with Scrum, Kanban and Waterfall), it is fairly obvious that, for this system and the current state of software engineering in general, the waterfall method would be an inadequate methodology. Unseen issues may occur in terms of fulfilling an identity profile, discussed in section 2.4.2, during the implementation. It

would be unwise, therefore, to execute a complete design and implementation without provisions in place to revise. I would opt to use Kanban over Agile and Scrum, due to the fact that Scrum would work better in a team environment, and executing the methodology as a singular implementor over inflates the workflow. Instead, by following the Kanban process and using a sufficient Kanban board, I could work effectively and constantly visualise the current state of the system.

In terms of technology researched, there is a strong enough case for C# to be considered a perfect choice for use as the principal language for ID Anywhere, with it's strong static typing and the fact that it is intertwined with Microsoft Azure for cloud hosting and easy deployment making it a good choice for any web applications or api's needed. However, there are benefits to Python as opposed to C# to warrant its inclusion, if necessary. The benefit of using the intermediate compilation as well as quick prototyping makes it a viable solution to quickly test out any new ideas, as well as using it for developing any daemon services (That is a service that runs constantly in the background). Finally, flutter provides an extremely strong case for Mobile development, with cross-platform compilation and it's extremely intuitive UI method of design and development. C# does also boast that ability (Through the use of a technology called Mono, that was not discussed as it does not currently sit at the same standard of flutter for developing modern mobile applications) but, Flutter holds a much stronger claim to being used for any mobile development.

3.0 Requirements

3.1 Overview

The requirements for ID Anywhere will be extracted from the principal Idea and research pertinent to it. These will be processed according to the volare model. This includes defining any functional and non-functional requirements for the system. Functional being requirements for the system to achieve the Minimum Viable Product in terms of being a usable system and Non-functional being anything relating to the nature of the application including but not limited to security requirements, performance requirements and legal requirements.

3.2 Constraints

3.2.1 Project Purpose

The purpose of ID Anywhere is to satisfy the requirements of the Computing Project module as part of my degree at UWE, as well as improving my core skills of designing and implementing software from conception to birth. By looking at the current developments in digital identification in the UK, a satisfactory system will be developed to aid in the

identification of individuals to merchants through use of their smart device. This project will be made open source when complete.

3.2.2 Stakeholders

The stakeholders of the intended system are as follows:

- Project Supervisor - James Lear
- Module Leader - Paul Raynor
- Developer - Lewis Cummins
- Testers - Students and Colleagues
- Course Leader - Elias Pimenidis

3.2.3 System End Users

General Population

The general population (Namely, people aged 16-30 who have a higher likelihood of being asked for identification) are the main users of the client application that presents the QR code. Considerations need to be made to make the application intuitive and user friendly, with a design that provides a seamless user experience. They will have all of the contact time with the mobile application that provides the ID code. They will have no responsibilities other than the initial sign up process, where they will need to provide information about themselves and documents that backup their claim to a specific identity. Furthermore, there may be action required if it is found that their ID is being abused i.e. lending out to friends that are not of age.

Independent Retailers

Independent retailers are a core target audience. As they may only have one shop that they own and operate, you cannot assume that they will have all of the technology available to a large national retailer. However, you can assume that they will have a smartphone. As it is imperative that all retailers have access to functionality that validates the QR code presented from the client, a separate piece of functionality could be added to the client that allows for handheld verification, as opposed to providing an API that can be integrated into a POS terminal (However, if they have the infrastructure readily available, then there is no reason why they could not use that).

Nationwide Merchants

Nationwide merchants would have more of an interaction with this system as the footfall through their establishments is higher than independent retailers. Due to this, you would

expect them to have the highest contact time with the Integrated API for validating users. The responsibility to handle the integration API however will most probably not fall to them. This depends on whether or not they own and maintain their own POS terminal.

3.2.4 Project Constraints

In terms of how the project will be constrained to architecture; it will need a mobile application (Flutter), an API (C# .NET Core) a web application (C# .NET Core) and a work daemon for handling preliminary identification information correlation. This will be discussed and conceived in **section 4.0** (Design).

Users will be able to download the mobile app from the Google Play Store or on the Apple App Store. The API and Web app will be hosted in Microsoft Azure, with the daemon and API sitting on the same virtual machine. Due to the interactive nature of the system, correct security requirements will need to be defined and adhered to in **section 3.5.**

There exists no cost constraints for this application, as it is not funded and is a university dissertation.

In terms of time constraints, the initial release is due to release on March 2nd, with primary testing to occur. The final release will be on April 3rd, where the system is due to be ready for deployment.

3.2.4 Project Assumptions

- Assume that all devices are connected to the internet, either wirelessly through WiFi or mobile data transmission through 3G and 4G.
- Assume users that apply for ID Anywhere have appropriate identification documents (But always check for fraud).

3.3 Requirement Types

As I am following the volere template for requirements extraction and engineering, I will use the appropriate requirement types as defined by Robertson and Robertson (2012). They are split into functional and non-functional requirement types. If a type falls into the format of ‘1.x’ then it is a functional requirement, these are requirements pertaining to the operation of the system. For example, any interface, communication, business logic and database operations fall under functional requirements. If a type falls into the format of ‘2.x’ then it is a non-functional requirement (relating to the nature of the application and it’s wider effect). All requirements will have priority factors attached to them, known as MoSCoW (Project Smart, 2020). These factors are must, should, could and wont. They state how valuable the requirements are to the fundamental operation of the base system. Not all of them may be implemented, and some kept for future revisions to ID Anywhere.

Type #	Requirement Type	Definition
1.1	Functional: API	The fundamental functionality required by the subject matter of the API
1.2	Functional: Mobile	The fundamental functionality required by the subject matter of the mobile application
1.3	Functional: Web App	The fundamental functionality required by the subject matter of the Web Application
1.4	Functional: Python/Mongo Work Queue	The fundamental functionality required by the subject matter of the Python/Mongo Work Queue
1.5	Functional: API Database	The fundamental functionality required by the subject matter of the API Database
1.6	Functional: Firebase Cloud Platform	The fundamental functionality required by the subject matter of the Firebase Cloud Platform
2.1	Non Functional: Security	The properties that the modules of ID Anywhere must have in relation to security
2.2	Non Functional: Performance	The properties that the modules of ID Anywhere must have in relation to culture and politics
2.3	Non Functional: Operational	The properties that the modules of ID Anywhere must have in relation to the

		operation
2.4	Non Functional: Legal	The properties that the modules of ID Anywhere must have in relation to legal
2.5	Non Functional: Maintainability	The properties that the modules of ID Anywhere must have in relation to maintainability
2.6	Non Functional: Cultural & Political	The properties that the modules of ID Anywhere must have in relation to performance
2.7	Non Functional: Usability	The properties that the modules of ID Anywhere must have in relation to usability

3.4 Functional Requirements

3.4.1 Definitions

ID	Type	Priority	Dependants	Description
FN01	**1.1**	M	**FN03**	The API **must be** able to create a new user during signup.
FN02	**1.1**	M	**FN03**	The API **must** be able to login a user.
FN03	**1.5**	M	**/**	The API database **must** be created with the designed schema.
FN04	**1.1**	M	**FN02**	The API **must** be configured to sign and issue JWT for authentication and authorization purposes.
FN05	**1.2**	M	**FN01**	The mobile application **must** provide a way for the user to create an account.
FN06	**1.2**	M	**FN05**	The mobile application **must** provide a way to login a user that has signed up.

FN07	**1.6**	M	/	The firebase cloud platform **must** be initialised to allow for cloud image storage
FN08	**1.2**	M	**FN07, FN06**	The mobile application **must** allow the user to upload a profile picture to firebase.
FN09	**1.2**	M	**FN07**	The mobile application **must** allow the user to upload a picture of their driving license to firebase.
FN10	**1.2**	M	**FN07**	The mobile application **must** allow the user to upload a picture of the back of their driving license to firebase.
FN11	**1.2**	M	**FN07**	The mobile application **must** allow the user to upload a picture of their passport to firebase.
FN12	**1.2**	M	**FN09**	The mobile application **must** read information off of the license to prove that it is a license image.
FN13	**1.2**	M	**FN10**	The mobile application **must** read information off of the passport to prove it is a passport image.
FN14	**1.1**	M	**FN12, FN13**	The API **must** allow upload of information parsed from passport and license.
FN15	**1.4**	M	**/**	A mongo db instance **must** be set up to interact with the python daemon.
FN16	**1.4**	M	**FN15**	The python work queue daemon **must** be able to pick up work jobs stored in the queue and validate them.
FN17	**1.2**	M	**FN16**	The API **must** be able to allow the daemon to update a user's status based on the processed information
FN18	**1.2**	M	**FN17**	The API **must** create a final job if the result passed from the daemon shows information is valid.
FN19	**1.2**	S	**/**	The API **should** be able to create actions on a user account that the user can appropriately respond to.
FN20	**1.4**	S	**FN19**	The mobile app **should** be able to check any

				actions on the users account through the api and appropriately display them.
FN21	**1.1**	M	**FN01, FN02**	The API **must** be extended to allow administrator accounts to be created by an existing admin
FN22	**1.1**	M	**FN21**	The API **must** be extended to allow administrators to login
FN23	**1.3**	M	**FN22**	The Web App **must** have a login page for users and admins.
FN24	**1.3**	M	**FN23**	The Web App **must** display a dashboard for admins when logged in.
FN25	**1.3**	M	**FN24**	The Web App **must** allow admins to select jobs from the dashboard
FN26	**1.3**	M	**FN25**	The Web App **must** allow admins to view a job, and correct any information that has incorrectly been read from the images of the passport and license.
FN27	**1.1**	M	**/**	The API **must** be able to provide an endpoint for the webapp to mark a job as complete based upon what they have found.
FN28	**1.3**	S	**FN22**	The Web App **should** display a dashboard for users
FN29	**1.3**	S	**FN28**	The Web App **should** allow users to lock and unlock their ID.
FN30	**1.3**	S	**FN28**	The Web App **should** allow users to unlock their ID tied to their device.
FN31	**1.3**	C	**FN28**	The Web App **could** show users the last few places their ID was used.
FN32	**1.1**	M	**/**	The API must be able to generate a code that represents the user's profile and age, which can be used for validation.
FN33	**1.1**	M	**FN32**	The API must be able to validate a code that represents the user's profile and age.
FN34	**1.4**	M	**FN13, FN12**	The mobile app must be able to display a QR code that represents the user's profile and

				age, which can be used for validation.
FN35	**1.2**	W	**/**	The mobile application won't support integrated code scanning.
FN36	**1.2**	W	**/**	The mobile application won't support an ID being on multiple devices.

3.4.2 Rationale

- **FN01:** *The API must be able to create a new user during signup.*

  All users need to have an account to ID Anywhere. This allows them to then continue with the process of uploading their documents and completing verification.

- **FN02:** *The API must be able to login a user:*

  A centralised location to process user login is required. This will allow access to protected resources and prevent unauthorised access.

- **FN03:** *The API database must be created with the designed schema.*

  All data that needs persisting will need to be stored in a database. A required schema will be devised during the design phase, and a database can be created. Some considerations will need to be either code first or database first creation

- **FN04:** *The API must be configured to sign and issue JWT for authentication and authorization purposes.*

  When a user is logged in, a cookie or token needs to be generated that validates the user and each request they make. A JWT (Json Web Token) is a standardised token format that Is used all across the web for authenticating users against a server, as well as storing information such as their name.

- **FN05:** *The mobile application must provide a way for the user to create an account.*

  As the mobile platform is the main interface the user has into ID Anywhere, they will need to be able to create an account. This will include providing information such as name, email address and password. This will be sent to the API where, if the information is valid, their account will be created and stored in a database.

- **FN06:** *The mobile application must provide a way to login a user that has signed up.*

  As the mobile platform is the main interface the user has into ID Anywhere, they will need to be able to login. They will be able to provide a user name or password or use the integrated biometric fingerprint login if their smartphone has the capability.

- **FN07:** *The firebase cloud platform must be initialised to allow for cloud image storage*

The firebase cloud platform is a BaaS (Backend as a service) which provides features such as cloud authentication, realtime nosql data storage and file storage. This needs to be initialised for the project, and a connection made between the mobile app and the platform to allow image and data storage.

- **FN08:** *The mobile application must allow the user to upload a profile picture to Firebase.*

  When the user has created an account and logged in for the first time, they need to have the ability to upload a profile picture. This will be the picture that merchants see as well as what is used for facial recognition matching with their provided identity documents. Requirements for the image will be shown to the user.

- **FN09:** *The mobile application **must** allow the user to upload a picture of their driving license to Firebase.*

  Whilst setting up their account, the user must be able to upload a picture of their driving license, this is so that it may be used to verify information about them and fulfill requirements of creating an identity profile as defined in the research section.

- **FN10:** *The mobile application must allow the user to upload a picture of the back of their driving license to Firebase.*

  Whilst setting up their account, the user must be able to upload a picture of the back of their driving license, this is so that it may be used to verify information about them and fulfill requirements of creating an identity profile as defined in the research section.

- **FN11:** *The mobile application must allow the user to upload a picture of their passport to firebase*

  Whilst setting up their account, the user must be able to upload a picture of their passport, this is so that it may be used to verify information about them and fulfill requirements of creating an identity profile as defined in the research section.

- **FN12:** *The mobile application must read information off of the license to prove that it is a license image.*

  After the picture of the license is uploaded to the mobile app, by extracting information using OCR (Optical Character Recognition) this can be used to validate information from other documents and prevent erroneous or false images make it further through the processing chain.

- **FN13:** *The mobile application must read information off of the passport to prove it is a passport image.*

  After the picture of the passport is uploaded to the mobile app, by extracting information using OCR (Optical Character Recognition) this can be used to validate information from other documents and prevent erroneous or false images make it further through the processing chain.

- **FN14:** *The API must allow upload of information parsed from passport and license*

  Information read off of the images uploaded by the user must be uploaded so that it can be processed by the mongo work queue.

- **FN15:** *A mongo db instance must be set up to interact with the python daemon.*

  In order for the information uploaded in the form of jobs to be stored and read, a mongo instance must be set up so that the API can push the information there.

- **FN16:** *The python work daemon must be able to pick up work jobs stored in the queue and validate them.*

  In order to perform further validation on the information extracted from the images, the information needs to be cross referenced, as well as information such as the MRZ on the passport extracted, and use of facial recognition models to confirm information pertains to the right person.

- **FN17:** *The API must be able to allow the daemon to update a user's status based on the processed information*

  When the daemon is finished working, it must indicate whether or not the data passed the predefined checks and is now ready for processing by the admin. This needs to be secured so only the daemon can notify the API through this channel.

- **FN18:** *The API must create a final job if the result passed from the daemon shows information is valid.*

  This job created will be used to display information to the administrator. They will then be able to cross reference this with the images and then verify any information and make sure the images are legitimate.

- **FN19:** *The API should be able to create actions on a user account that the user can appropriately respond to*

  Actions alert the user as to any issues that have occured during the verification process or otherwise, that allows them to react and provide more information. This is necessary in case the user provides unclear images or false images.

- **FN20:** *The mobile app should be able to check any actions on the users account through the api and appropriately display them.*

  When the user logs in, the application could asynchronously check for actions and display them to the user so they can then complete the appropriate resolutions.

- **FN21:** *The API must be extended to allow administrator accounts to be created by an existing admin*

Administrator accounts should only be used for processing information about user's profiles. They should not be able to be created anywhere, only by other administrators otherwise unauthorised access may be gained to user's private data.

- **FN22:** *The API must be extended to allow administrators to login*

  Administrators will be able to log in through the same web application, although will have different functionality presented to them.

- **FN23:** *The Web App must have a login page for users and admins*

  Both users and administrators will need to be able to log in to the web app, a dedicated login page will allow them an entry point into the application.

- **FN24:** *The Web App must display a dashboard for admins when logged in.*

  A dashboard allows the administrator a home page to all of the different functionality. It is important that the admin can easily access whatever they need to complete their job and validate the user.

- **FN25:** *The Web App must allow admins to select jobs from the dashboard*

  Admins must validate the information on the jobs and be able to select whatever job they decide to complete next from a centralised location.

- **FN26:** *The Web App must allow admins to view a job, and correct any information that has incorrectly been read from the images of the passport and license.*

  When the admin opens the job, they will have to examine all of the information and make a professional decision about whether or not to accept or deny the user based on the images and data extracted from them.

- **FN27:** *The API must be able to provide an endpoint for the webapp to mark a job as complete based upon what they have found.*

  *When the admin has decided on the validity of the users information, the resulting accept or deny must update the user's account to being fully valid or potentially marking an action (***FN19)** *on the account.*

- **FN28:** *The Web App should display a dashboard for users*

  Users should be able to log into the web application to access functionality for their account that is not available on the mobile application.

- **FN29:** *The Web App should allow users to lock and unlock their ID*

  If the user has lost their phone or someone has stolen their device, then it would be necessary to prevent the ID being used in case someone is able to access their ID Anywhere application and falsely represent the user. This is a potential feature that will potentially be pushed to further iteration should the project overrun.

- **FN30:** *The Web App should allow users to unlock their ID tied to their device.*

  When signing up, the user's account is tied to the device they sign up on. If the device is damaged or not currently in their possession, being able to unlock it every now and then would be a useful feature. This would need to be limited however, as otherwise people could just switch devices and allow friends to use their ID and they may not be of the required age. This is a potential feature that will potentially be pushed to further iteration should the project overrun.

- **FN31:** *The Web App could show users the last few places their ID was used*

  As a user experience feature, storing latitude and longitude and representing their locations on a map could provide a helpful feature to see where they use their most often, as well as a security measure in case the ID is hacked and used somewhere they have not been.

- **FN32:** *The API must be able to generate a code that represents the user's profile and age, which can be used for validation.*

  In order for the user to be identified at a location, they will need a standardised QR code that is generated whenever they click a button. This code will be valid for a set amount of time to prevent sharing the code.

- **FN33:** *The API must be able to validate a code that represents the user's profile and age.*

  When the code is scanned by a client trying to validate the user and their age, the code will need to be sent to the api for validation. This needs to happen through a server to prevent forgery and tampering with codes.

- **FN34:** *The mobile app must be able to display a QR code that represents the user's profile and age, which can be used for validation*

  *Following **FN33** and **FN34** the user will display a secure hashed string through a QR code that represents their ID Anywhere. This will only be displayed for a certain time period to prevent sharing.*

- **FN35:** *The mobile application won't support integrated code scanning*

  For this stage in the development, this won't be added in the current iteration. It would be a very good feature where clients would not have to support API integration into their point of sale workflow and could just scan codes from an app. However, It means adding extra accounts which validate merchants for a merchant account.

- **FN36:** *The mobile application won't support an ID being on multiple devices.*

  While some people have 2 phones, they can just have ID Anywhere on one of them. Usually people do not carry around 2 driving licenses. Having 2 devices allows people to

potentially share their ID with one other person, which could create a problem for teenagers sharing their ID with other underage teenagers.

3.5 Non-Functional Requirements

3.5.1 Definitions

ID	Type	Priority	Dependants	Description
NF01	2.1	M	/	Any and all HTTP communication **must** be done over TLS with valid PKI Certificates.
NF02	2.1	M	/	Any and all passwords stored **must** be Hashed using at least SHA256.
NF03	2.1	M	/	The use of CORS (Cross-Origin Resource Sharing) Policies **must** be used to protect specific API endpoints.
NF04	2.2	M	/	Any and all requests **should** not take more than 5 seconds to get a response.
NF05	2.2	S	/	Images **should** be cached when downloaded frequently and are not subject to change.
NF06	2.3	S	/	The Architecture **could** be able to support up to 50 concurrent users.
NF07	2.3	M	/	The API **must** never go down and handle any errors appropriately.
NF08	2.4	M	/	The system **must** comply with GDPR where appropriate.
NF09	2.4	M	/	The system **must** identify people based on the defined identity legislation.
NF10	2.5	M	/	All code and API endpoints **should** be well documented to allow for extension and usage with ease.
NF11	2.5	M	/	All changes made must be made into a version control system, and branches made for all sprints.
NF12	2.6	C	/	The system **could** notify the user that facial

				recognition is used in this application so that users can make informed decisions.
NF13	2.7	M	/	ID Anywhere **should** be intuitive enough for any User to pick up and use, and for administrators to only require a short training session to understand the admin functionality.
NF14	2.7	M	/	The styling of the system **must** be as uniform as possible to ensure familiarity with the system and attributing the brand.
NF15	2.7	S	/	ID Anywhere **should** ensure that as much data is collected from documents as possible to ease the burden of the administrators

3.5.2 Rationale

- **NF01:** *Any and all HTTP communication must be done over TLS with valid PKI Certificates.*

  TLS is HTTPS. It ensures all data sent is encrypted by the public and private keys of the server certificate and client certificate. This is necessary and is a web standard, that prevents intercepted traffic being used maliciously thus preventing man in the middle attacks.

- **NF02:** *Any and all passwords stored must be Hashed using at least SHA256.*

  Passwords should never be stored in plain text. By using a hash, we can hash any subsequently provided password and compare the two. This prevents data leaks providing passwords and when paired with https, makes it a lot harder for people with malicious intent to acquire sensitive information.

- **NF03:** *The use of CORS (Cross-Origin Resource Sharing) Policies must be used to protect specific API endpoints.*

  CORS prevents unwanted requests coming in from locations that the system does not know of. It essentially whitelists certain endpoints with only a certain number of hosts.

- **NF04:** *Any and all requests **should** not take more than 5 seconds to get a response*

  Users should never be waiting for requests to finish over long periods of time. This will cause them to be frustrated with the system and potentially lead to them not using it. All requests should be processed fast, especially due to the nature of ID Anywhere

attempting to speed up the workflow of identifying oneself.

- **NF05:** *Images should be cached when downloaded frequently and are not subject to change.*

  There is no need to keep using bandwidth to download images all the time when they are not subject to change, this will improve the performance by reducing the amount of requests made by the application.

- **NF06:** *The Architecture could be able to support up to 50 concurrent users.*

  In this current iteration of ID Anywhere, being a proof of concept, not much effort will be made to implement features that allow a heavy load, as there are a number of things to consider when horizontally scaling through a load balancer and code must be made to accommodate this change. However, 50 concurrent users should not be a bad stretch for the first iteration.

- **NF07:** *The API must never go down and handle any errors appropriately.*

  To prevent unforeseen errors causing the application to completely fail and require a restart, a suitable error handling strategy will need to be implemented to ensure adequate documentation of errors and to keep the system running smoothly.

- **NF08:** *The system must comply with GDPR where appropriate.*

  Due to new legislation being introduced to the EU, ID Anywhere has to clearly state how it will store a user's data, as well as comply with their wishes as to seeing what data is stored about them and asking for their data to be removed should they wish it to be.

- **NF09:** *The system must identify people based on the defined identity legislation.*

  *In **Section 2.4** the requirements for determining how sure one can be of the strength of a presented identity are extracted from GOV.UK (2019). Based on these requirements, the system should adequately identify.*

- **NF10:** *All code and API endpoints should be well documented to allow for extension and usage with ease.*

  All necessary endpoints that will be exposed to clients should be well documented so they know how to integrate them into their workflow. This also allows anyone who will work on the application in the future a quick way to get up to speed to start working.

- **NF11:** *All changes made must be made into a version control system, and branches made for all sprints.*

  Using version control is pivotal to tracking changes in the code and keeping revisions in case a bad change is made or code is lost. It also makes it a lot more maintainable as you can have multiple branches of the code at any time.

- **NF12:** The system could notify the user that facial recognition is used in this application so that users can make informed decisions.

  Some users may hold the view that facial recognition is unethical, even with a transparent privacy policy and data retention policy, their views might hold. Therefore making it clear in some way could help in not making people go against their political views.

- **NF13:** *ID Anywhere should be intuitive enough for anyone to pick up and use, and for administrators to only require a short training session to understand the admin functionality.*

  An Intuitive design is important for a mobile app, as well as for a web app. If a user cannot understand what to do next or is struggling finding a setting, it will frustrate them.

- **NF14:** *The styling of the system must be as uniform as possible to ensure familiarity with the system and attributing the brand.*

  The styling should be consistent as users will know what to expect from the application and where to expect certain functionality.

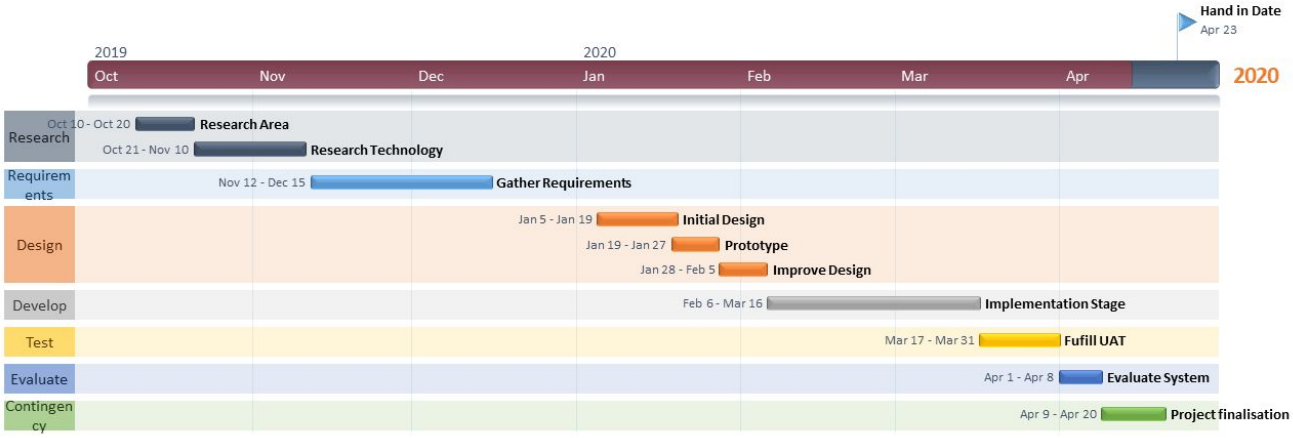
- **NF15:** *ID Anywhere should ensure that as much data is collected from documents as possible to ease the burden of the administrators.*

  By reducing the workload of the administrators, it increases the amount of jobs that they can process in a given time period..

3.6 Tasks and Issues

3.6.1 Project Outline

This is an outline of how I intend to proceed with the project, and certain milestones that need to be hit by certain dates. However, due to the nature of the project this may be subject to change as necessary.



3.6.2 Project Risks

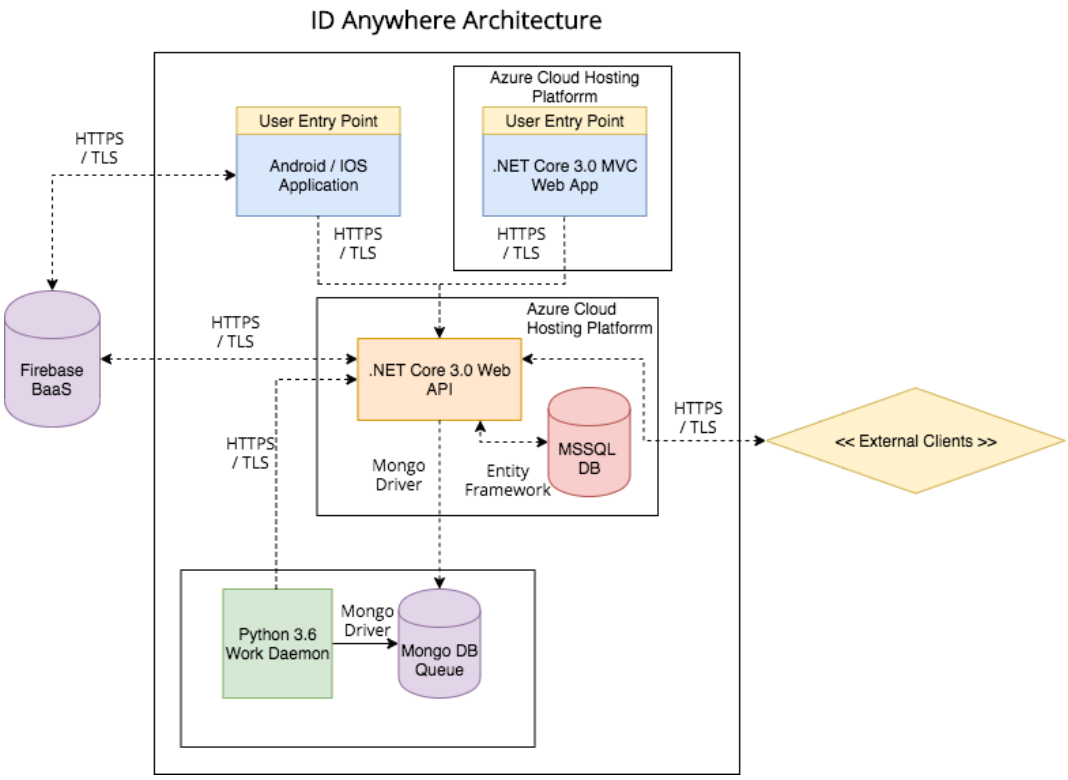
Risk	Severity 1-5	Likelihood of occurrence 1-5	Mitigation
The project overruns the time frame	3	2	The report will be as well fleshed out as possible before any programming is started, so a clear picture of the software is painted. That way, even if some of the code is unfinished, the majority of the project is not.
Loss of code	5	1	Continually commit changes to the cloud version control system.
Need to upskill	1	2/3	Ensure that all technical knowledge is acquired before development starts. Add contingency to allow for unforeseen technical knowledge gap.

4.0 Design

4.1 Overview

The design of ID Anywhere will be built from various UML & architecture diagrams. These will be broken down into the smallest components possible and the implementation devised from these diagrams. A suitable test plan will also be designed, based upon the requirements.

4.2 High Level System Architecture

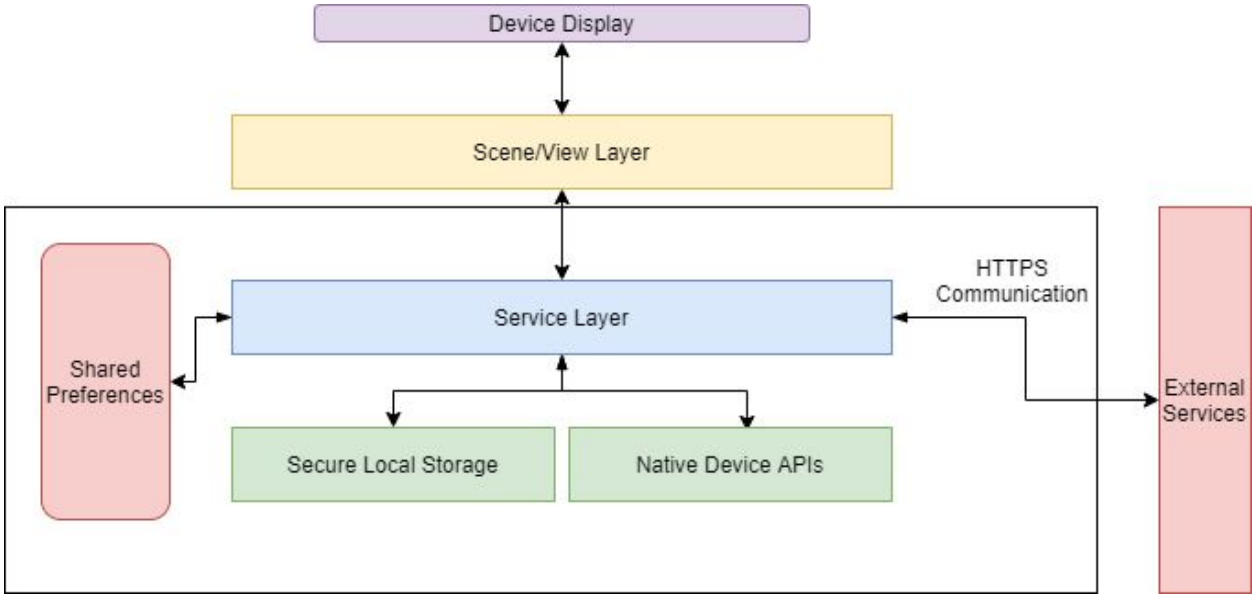


Above is the high level architecture of ID Anywhere, the various components will be broken down in detail and be architected their own way following web service and N tier architecture.

4.3 Logical Component Architecture

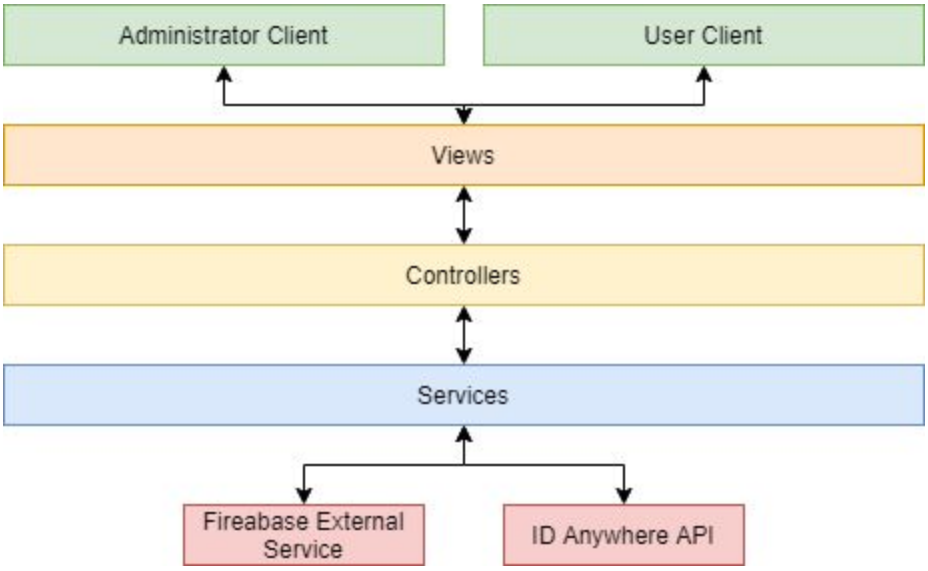
4.3.1 Mobile Application

The mobile application will need to communicate with external services such as the ID Anywhere API and Firebase through a service layer. This allows all communication logic to be contained in one place, and prevents any unwarranted communication occurring. The component view below shows how the application is architected to facilitate this, as well as the methods of displaying information to the user.



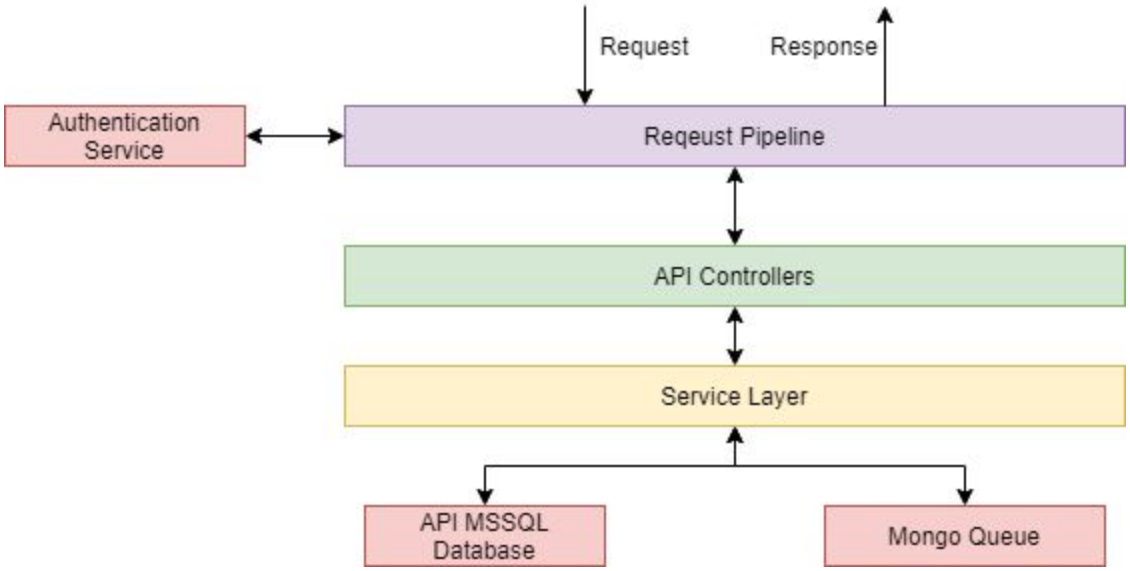
4.3.1 Web Application

The web application architecture follows a Model, View, Controller pattern, which is also part of the wider service architecture, where all communication and business logic is contained within the service layer itself.



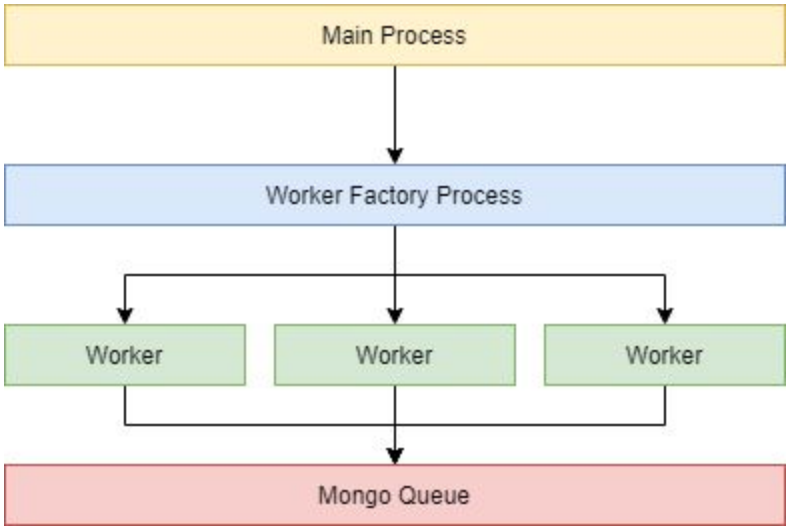
4.3.1 API

The API is where the majority of communication occurs. All components will make requests to it. Requests are passed through the initial pipeline which handles the management of authentication and relevant CORS policies, as well as routing to the correct API controller and request data mapping to relevant models. This is all handled with the .NET Core package.



3.1 Work Daemon

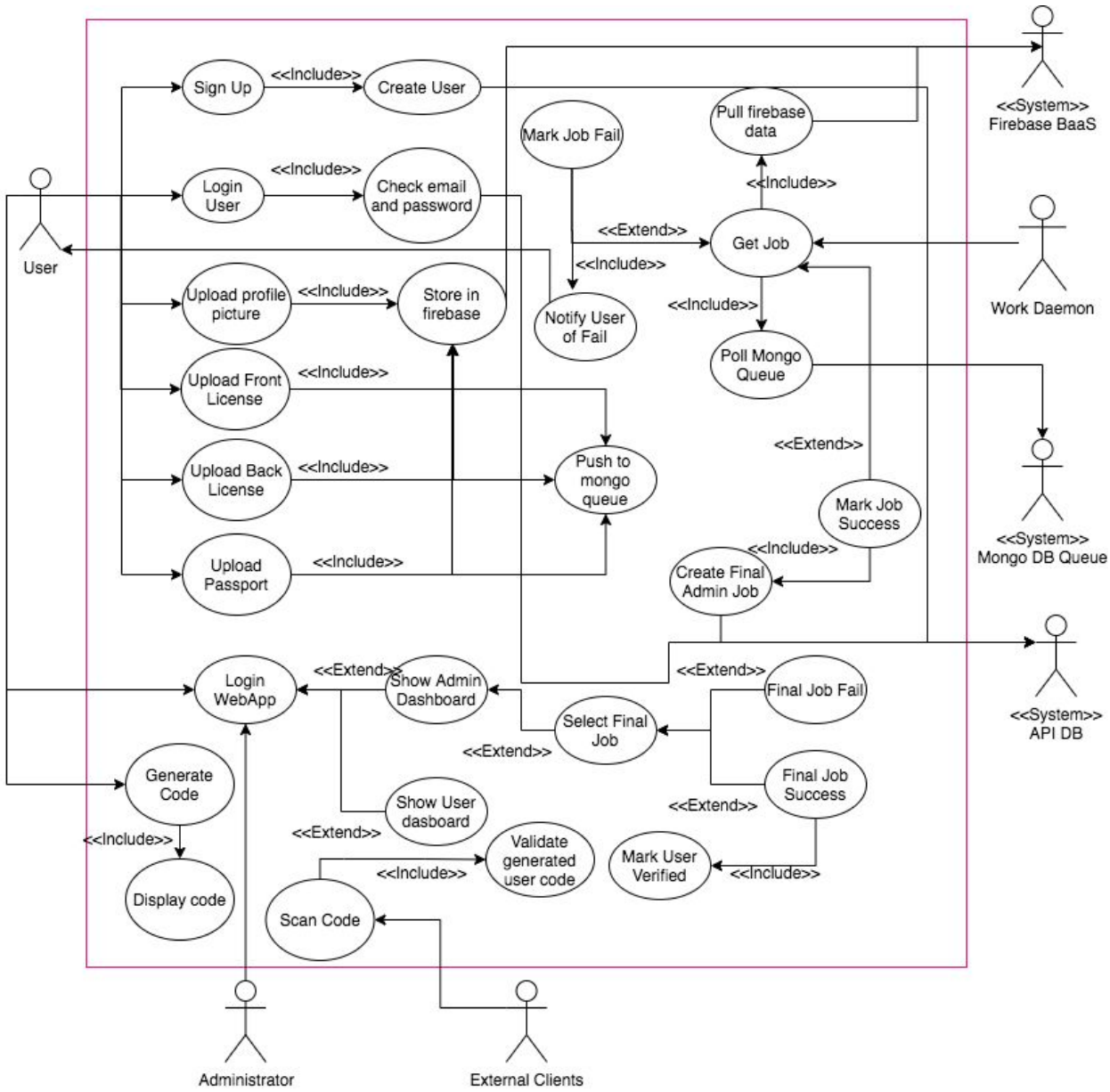
The initial python daemon structure must be created with the ability to spawn processes as well as safely close down the current workers without data loss or corruption.



By architecting the Python daemon in this way, it allows for a command to be passed down from the main process to the factory to tell all workers to complete their jobs, and then close themselves. This ensures a safe closure of the workers.

4.4 Use Case Design

4.4.1 Diagram



4.4.2 Narratives

In order to provide succinct design some of the use cases that are implicit in terms of their workflows will be ignored in this section of providing narratives. The purpose of these is to show the clear flow of events for intricate parts of the system and how these events affect the actors involved. The sign up and login workflows are extremely standard and very universal, the main workflow is the steps that a user needs to take from logging in for the first time to having a fully verified account; this is what these narratives will show.

4.4.2.1 “Upload Passport”’

Description: The user will open the mobile application and upload a picture of their passport. Information needs to be extracted using OCR from the picture.

Goals and Benefits: Uploading the picture allows the admin to view this later on when doing their own processing. By extracting information first and uploading it to the work queue to allow automatic processing and comparison, we can prevent wasting administrators time with unclear or false images.

Actors: User, Firebase Platform, MongoDB Queue

Triggers: A user needs to upload their passport.

Preconditions: User has created an account, logged in to the mobile app and is unverified.

Post conditions: The passport image is uploaded to Firebase and information extracted into the work queue. If all other documents have been uploaded, the job in the work queue is marked ready. If the information cant be read, another image is asked for by the user.

Main Flow of Events:

1. **User logs Into mobile applications**
2. **User moves to upload tab**
3. **User clicks upload passport button**
4. **User reads image requirements**
5. **User accepts image requirements**
6. **User chooses method of image upload**
    a. **Gallery**
    b. **Camera**
7. **System scans passport for First Name**
8. **System scans passport for Last Name**
9. **System scans passport for Date of Birth**
10. **System scans passport for Expiry Date**
11. **System scans passport for License Number**

12. **System scans passport for MRZ**
13. **System completes scan**
    a. **Pass**
    b. **Fail**

Alternate Flow of Events:

'6a Gallery'

1. **User selects image from gallery**

'6b Camera'

1. **User takes image from camera**

'13a Pass'

1. **Image is uploaded to Firebase**
2. **Image is passed to 'Push to mongo queue' use case**

'13b Fail'

1. **User is notified to provide another image**

4.4.2.2 ''Upload Front License'

Description: The user will open the mobile application and upload a picture of the front of their license. Information needs to be extracted using OCR from the picture of the.

Goals and Benefits: Uploading the picture allows the admin to view this later on when doing their own processing. By extracting information first and uploading it to the work queue to allow automatic processing and comparison, we can prevent wasting administrators time with unclear or false images.

Actors: User, Firebase Platform, MongoDB Queue

Triggers: A user needs to upload the front of their license.

Preconditions: User has created an account, logged in to the mobile app and is unverified.

Post conditions: The front of the license image is uploaded to firebase and information extracted into the work queue. If all other documents have been uploaded, the job in the work queue is marked ready. If the information cant be read, another image is asked for by the user.

Main Flow of Events:

1. **User logs Into mobile applications**
2. **User moves to upload tab**
3. **User clicks upload front of license button**

4. **User reads image requirements**
5. **User accepts image requirements**
6. **User chooses method of image upload**
    a. **Gallery**
    b. **Camera**
7. **System scans front of license image for First Name**
8. **System scans front of license image for Last Name**
9. **System scans front of license image for Date of Birth**
10. **System scans front of license image for Expiry Date**
11. **System scans front of license image for License Number**
12. **System completes scan**
    a. **Pass**
    b. **Fail**

Alternate Flow of Events:

‘6a Gallery’

2. **User selects image from gallery**

‘6b Camera’

2. **User takes image from camera**

‘13a Pass’

3. **Image is uploaded to Firebase**
4. **Image is passed to ‘Push to mongo queue’ use case**

‘13b Fail’

2. **User is notified to provide another image**

4.4.2.3 ‘’Upload Back License’

Description: As the back license image does not have any valuable information on it, the image will only be uploaded to Firebase and used during processing by an administrator.

Goals and Benefits: Having a picture of the back of their driving license allows the administrator to further identify whether or not the provided identity document is legitimate.

Actors: User, Firebase Platform, MongoDB Queue

Triggers: A user needs to upload the back of their license.

Preconditions: User has created an account, logged in to the mobile app and is unverified.

Post conditions: The front of the license image is uploaded to Firebase. If all other documents have been uploaded, the job will be marked as ready.

Main Flow of Events:

1. **User logs Into mobile application**
2. **User moves to upload tab**
3. **User clicks upload front of license button**
4. **User reads image requirements**
5. **User accepts image requirements**
6. **User chooses method of image upload**
    a. **Gallery**
    b. **Camera**
7. **Image uploaded to firebase by application**

Alternate Flow of Events:

'6a Gallery'

3. **User selects image from gallery**

'6b Camera'

3. **User takes image from camera**

4.4.2.4 Push to Mongo Queue"

Description: All information uploaded relating to the user's documents will be pushed to the same job in the mongo work queue. If no information has been created for a user yet, then the job is initialised with the data. When all information necessary has been uploaded for a specific user, their job will be marked as ready for processing.

Goals and Benefits: Stores the data uploaded from either the passport or the front of the license.

Actors: System, Mongo Work Queue

Triggers: A user uploads a document to the API

Preconditions: User has uploaded an image, and the information has been read off and sent to the API.

Post conditions: The information will be added either to a new job, if the user has not uploaded any documents yet, or an existing job, if the user has already uploaded one of the three documents.

Main Flow of Events:

1. **Information Uploaded to API from mobile application**
2. **Information passed to service layer**

3. **Queue checked for existing job**
    a. **Job Doesn't Exist**
4. **Information uploaded to job**
5. **API checks if the job has all information is uploaded**
    a. **Yes**
    b. **No**
6. **API informs mobile application of success**

Alternate Flow of Events:

‘3a Job Doesn't Exist’

1. **API creates new job in work queue for user**

‘5a Yes’

1. **Job is marked as ready for processing by the python daemon**

‘5a No’

1. **Job is left to wait for more information**

4.4.2.5 ‘Poll Mongo Queue’

Description: The Daemon process will poll the mongo queue for jobs that have been marked as ready then select the oldest one to process.

Goals and Benefits: By polling the work queue every (N) seconds (dependent on configuration) a job can be selected for processing.

Actors: Mongo work queue, Python daemon.

Triggers: The Python daemon is started.

Preconditions: Mongo queue must be set up and be able to receive jobs and other information, daemon must be started.

Post conditions: A job will now be taken from the queue into a python worker, which will be processed in the ‘Process Job’ use case.

Main Flow of Events:

1. **Python worker factory starts**
2. **Spawns worker processes**
3. **Worker connect to queue**
4. **Worker searches queue for oldest ready jobs**
    a. **Selects a job**
5. **Worker waits for X time before checking for another job.**

Alternate Flow of Events:

‘4a Selects a job’

1. **Job is marked as in progress by daemon**
2. **Passed to 'Process Job' use case**

4.4.2.6 'Process Job'

Description: The Python daemon processes the information extracted from the images in the mobile app and cross references the information. Once this passes or fails, the user will know if their verification status is now pending or more information is required.

Goals and Benefits: By automating the check of certain information on the documents provided, time wasted can be reduced by administrators, as unclear or incorrect images can be filtered out as well as anyone attempting to provide fraudulent information.

Actors: Python daemon

Triggers: The Python daemon picks a job up from the queue.

Preconditions: A Job has been selected by the daemon from the work queue and it is ready for processing.

Post conditions: The information uploaded by the user will either be deemed suitable or unsuitable and the verification process can continue on based on the decision of the daemon.

Main Flow of Events:

1. **Worker gets user ID from the job**
2. **Worker downloads user profile picture from Firebase**
3. **Worker downloads passport picture from Firebase**
4. **Worker passes images to facial recognition package**
5. **Package compares images**
    a. **No Match**
6. **Worker compares license first name and passport first name**
    a. **No Match**
7. **Worker compares license last name and license last name**
    a. **No Match**
8. **Worker checks that the license has not expired.**
    a. **Expired**
9. **Worker checks that the passport has not expired.**
    a. **Expired**
10. **Worker extracts date of birth from license number and compares with date of birth on license**
    a. **No Match**
11. **Worker parses the MRZ read off of the passport.**
12. **Worker checks date of birth on passport against the one from the MRZ.**
    a. **No Match**

13. **Worker checks date of birth on passport against date of birth on license.**
    a. **No Match**
14. **Worker checks number of incorrect details**
    a. **Over allowed threshold**
15. **Execute 'Mark Job Success' use case.**

Alternate Flow of Events:

'5a No Match'

1. **Go to 'Mark Job Fail' use case**

'6a No Match'

1. **+1 to error count**

'7a No Match'

1. **+1 to error count**

'8a Expired'

1. **Go to 'Mark Job Fail' use case**

'9a Expired

1. **Go to 'Mark Job Fail' use case**

'10a No Match'

1. **+1 to error count**

'12a No Match'

1. **+1 to error count**

'13a No Match'

1. **+1 to error count**

'14a No Match'

1. **+1 to error count**

4.4.2.7 'Mark Job Success'

Description: If all of the information is cross referenced and accepted to a suitable standard, then the job is marked as success and a final job created. The user is then moved into a pending status awaiting final processing by an administrator.

Goals and Benefits: Marking a job for success by the daemon allows the api to then create a new job for administrators to process.

Actors: Python Daemon, API

Triggers: Python daemon finishes processing a job

Preconditions: The daemon must have processed a job and deemed most of the information on the job suitable for final processing by the administrators.

Post conditions: A final job will be created for the administrator to process through the web application.

Main Flow of Events:

1. **Python worker gets user ID of job**
2. **Worker POST user ID, verified boolean and success message to API**
3. **API Finds job in queue by user id**
4. **API makes final job object**
5. **API copies job license first name to final job**
6. **API copies job license last name to final job**
7. **API copies job license date of birth to final job**
8. **API copies job license expiry to final job**
9. **API copies job license number to final job**
10. **API copies job passport first name to final job**
11. **API copies job passport last name to final job**
12. **API copies job passport date of birth to final job**
13. **API copies job passport expiry to final job**
14. **API copies job passport number to final job**
15. **API copies job passport number to final job**
16. **API inserts final job into API database**
17. **API deletes job from mongo work queue.**

4.4.2.8 'Mark Job Fail'

Description: If all of the information is cross referenced and it is found not to be to a suitable standard, then the job will fail and the user will be notified that there was an issue with their information they processed and more information is required.

Goals and Benefits: This prevents the information in the job getting through any further in the verification process and potentially allowing a disingenuous user acquiring an ID Anywhere account.

Actors: Python daemon, API

Triggers: Python daemon finishes processing a job

Preconditions: : The daemon must have processed a job and deemed most of the information on the job unsuitable for final processing by the administrators.

Post conditions: A job has been marked as failed and requires more information from the user.

Main Flow of Events:

1. **Python worker gets user ID of job**
2. **Worker POST user ID, failed verification boolean and fail message to API**
3. **API add action on user account**
4. **API find job by user ID**
5. **API set job not ready**

4.4.2.9 'Select Final Job'

Description: An administrator can select a final job from the list of jobs that are available to complete.

Goals and Benefits: Administrators need a way to select jobs for processing. Also, in order to fulfill the government guidelines for identity outlined in **section 2.4,** a real person with qualifications to spot fraudulent documents must view the images to ensure they are not fake.

Actors: Administrator, API

Triggers: Administrator logs into the web app and selects a job.

Preconditions: At least one job has been preprocessed by the python daemon and marked a success where a final job has been created.

Post conditions: An administrator now has a job that they can validate and change any information that has been incorrectly parsed from the images.

Main Flow of Events:

1. **Administrator logs in to web application**
2. **Dashboard is displayed**
3. **Web app gets list of jobs from API**
4. **Administrator clicks a job button on dashboard**
5. **Web app requests job from API**
    a. **Job doesn't exist**
6. **API returns job object**
7. **Web app displays the image of the user's passport**
8. **Web app displays editable text box with passport first name from final job**
9. **Web app displays editable text box with passport last name from final job**
10. **Web app displays editable text box with passport date of birth from final job**
11. **Web app displays editable text box with passport expiry from final job**
12. **Web app displays editable text box with passport number from final job**
13. **Web app displays editable text box with passport MRZ from final job**

14. **Web app displays the image of the user's front of license.**
15. **Web app displays editable text box with license first name from final job**
16. **Web app displays editable text box with license last name from final job**
17. **Web app displays editable text box with license date of birth from final job**
18. **Web app displays editable text box with license expiry from final job**
19. **Web app displays editable text box with license number from final job**
20. **Web app displays the image of the user's back of the license.**
21. **Web app displays accept job button**
22. **Web app displays deny job button**

Alternate Flow of Events:

'5a Job Doesn't exist'

1. **API returns error**
2. **Web app receives error**
3. **Web app displays error to user**

4.4.2.10 'Final Job Success'

Description: If the administrator processes the final job and has corrected any mistakes and accepts the provided images, the final job will be marked a success.

Goals and Benefits: Marking a final job as success will complete the verification process for a particular user, allowing them to generate codes and use them with clients that integrate their Point of Sale into the ID Anywhere workflow.

Actors: Administrator, API, User

Triggers: An administrator has finished processing a job

Preconditions: The administrator has selected a final job for processing.

Post conditions: A user is now fully verified.

Main Flow of Events:

1. **Administrator makes any necessary changes to values in textboxes when checking against images**
2. **Administrator clicks accept button for passport**
3. **Administrator clicks accept button for front of license**
4. **Administrator clicks accept button for back of license**
5. **Web app notifies API job successful**
6. **API stores Driving license number in database**
7. **API stores Passport Number in database**
8. **API stores confirmed date of birth in database**
9. **API deletes final job**
10. **API sets user as verified**

4.4.2.11 'Final Job Fail'

Description: If the administrator processes the final job and is not happy with the quality of images or any fraudulent images/information have gotten through, they can deny the job and the user's status will move back to unverified.

Goals and Benefits: This final check will prevent people from printing off images of a passport and license or simply printing off text that the system might pick up.

Actors: Administrator, API

Triggers: An administrator has finished processing a job

Preconditions: An administrator has selected a final job for processing.

Post conditions: The user is marked as unverified and a request for more information is made from the system to the user.

Main Flow of Events:

1. **Administrator makes any necessary changes to values in textboxes when checking against images**
2. **Administrator Inspects passport image**
    a. **Not Valid**
3. **Administrator Inspects front of license image**
    a. **Not Valid**
4. **Administrator Inspects back of license image**
    a. **Not Valid**

Alternate Flow of Events:

'2a Not Valid'

1. **Mark passport not valid**
2. **Send to API**
3. **API create action on account that passport is not valid**

'3a Not Valid'

4. **Mark front of license not valid**
5. **Send to API**
6. **API create action on account that front of license is not valid**

'4a Not Valid'

7. **Mark back of license not valid**
8. **Send to API**
9. **API create action on account that back of license is not valid**

4.4.2.12 'Generate Code'

Description: User generates a code valid for (N) seconds which can be displayed in the users device.

Goals and Benefits: By displaying the string in the form of a qr code, it ensures a universal way that the ID can be interpreted.

Actors: User, API

Triggers: User clicks button on mobile application.

Preconditions: A user has created an account, logged in and their account is fully verified.

Post conditions: A code is returned from the API that can be displayed on the mobile application.

Main Flow of Events:

1. **User goes to code page of mobile application**
2. **User clicks generate code button**
3. **API receives request**
4. **API gets driving license number**
5. **API gets passport number**
6. **API gets user ID**
7. **API generates random GUID**
8. **API creates code from four inputs**
9. **API stores code in cache with timestamp for expiry**
10. **API returns code to mobile application**
11. **Mobile application displays QR code with code as value**
12. **Mobile application displays timer for validity**

Alternate Flow of Events:

4.4.2.13 'Validate User Generated Code'

Description: A Code is sent to the API from an external client wishing to validate the age and validity of the user represented through the code.

Goals and Benefits: By allowing the external clients to send through a scanned user code and the required age, they can receive back a boolean value as well as an image of the scanned user. This makes it easy to validate the user is who they say they are.

Actors: User, External Clients, API

Triggers: An external client scans a user's code.

Preconditions: A user must generate a code through their mobile application and have it display on their device.

Post conditions: The code provided is validated against the required age provided by the external client and a result returned back to them to accept or deny the users ID to engage in whatever process required them to be identified.

Main Flow of Events:

1. **External Client scans code**
2. **External client sends code and required age to API**
3. **API gets time of request**
4. **API checks code against cache**
    a. **Does not exist**
5. **API checks timestamp against cached timestamp**
    a. **Expired**
6. **API gets user date of birth**
7. **API checks date of birth against required age**
    a. **Too young**
8. **API returns OK**
9. **External client has verified the user**

Alternate Flow of Events:

‘4a Does not exist’

1. **Invalid code provided**
2. **Return error to external client**
3. **External client denies verification to user**

‘5a Expired’

1. **Invalid code provided**
2. **Return error to external client**
3. **External client denies verification to user**
4. **User can generate another code to try again**

‘7a Too Young’

1. **API returns too young error**
2. **External client denies verification to user**

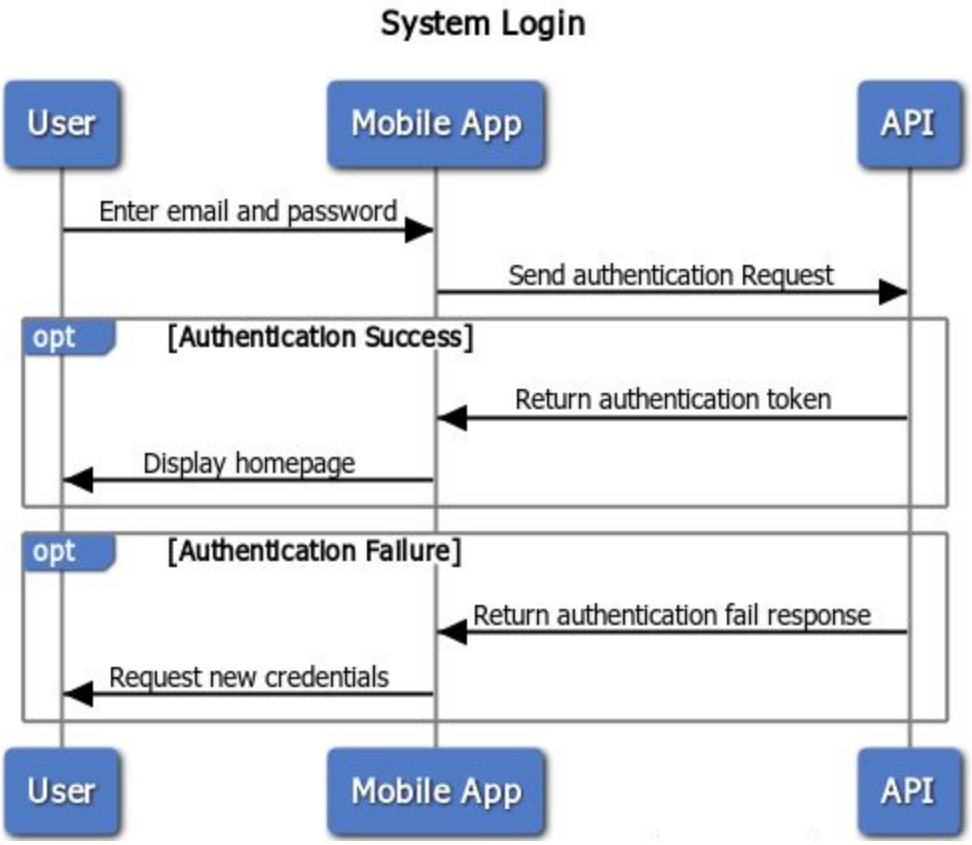
4.5 Sequence Design

Sequence diagrams show the flow of execution for functionality within ID Anywhere. System level sequence diagrams show how each component of the architecture interacts with each other for a specific workflow whereas the service level diagrams show the internal communication of different interfaces within the components themselves. Core

functionality will be broken down into sequence diagrams to allow easy development of methods and interfaces.

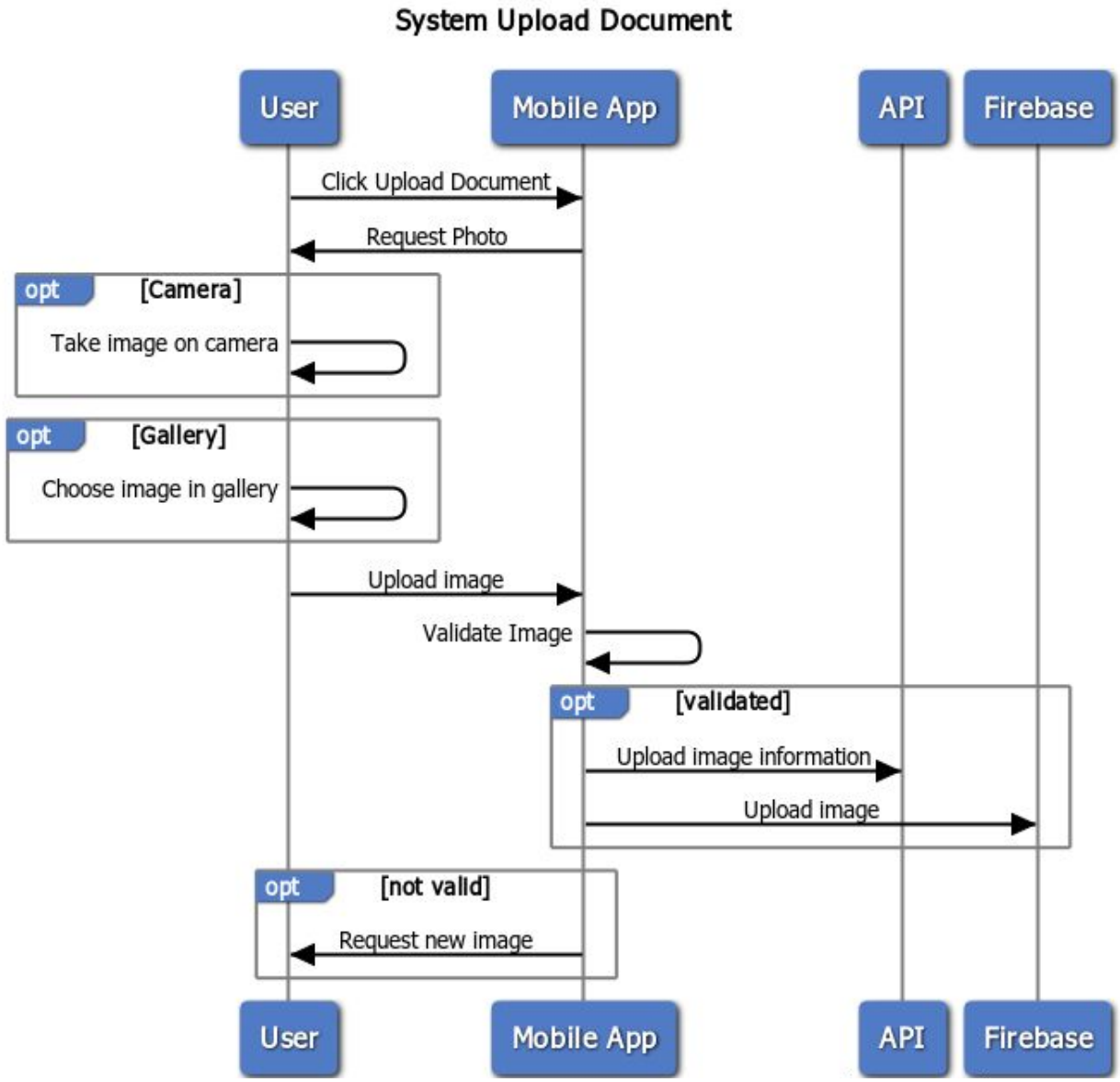
4.5.1 System Level

Login



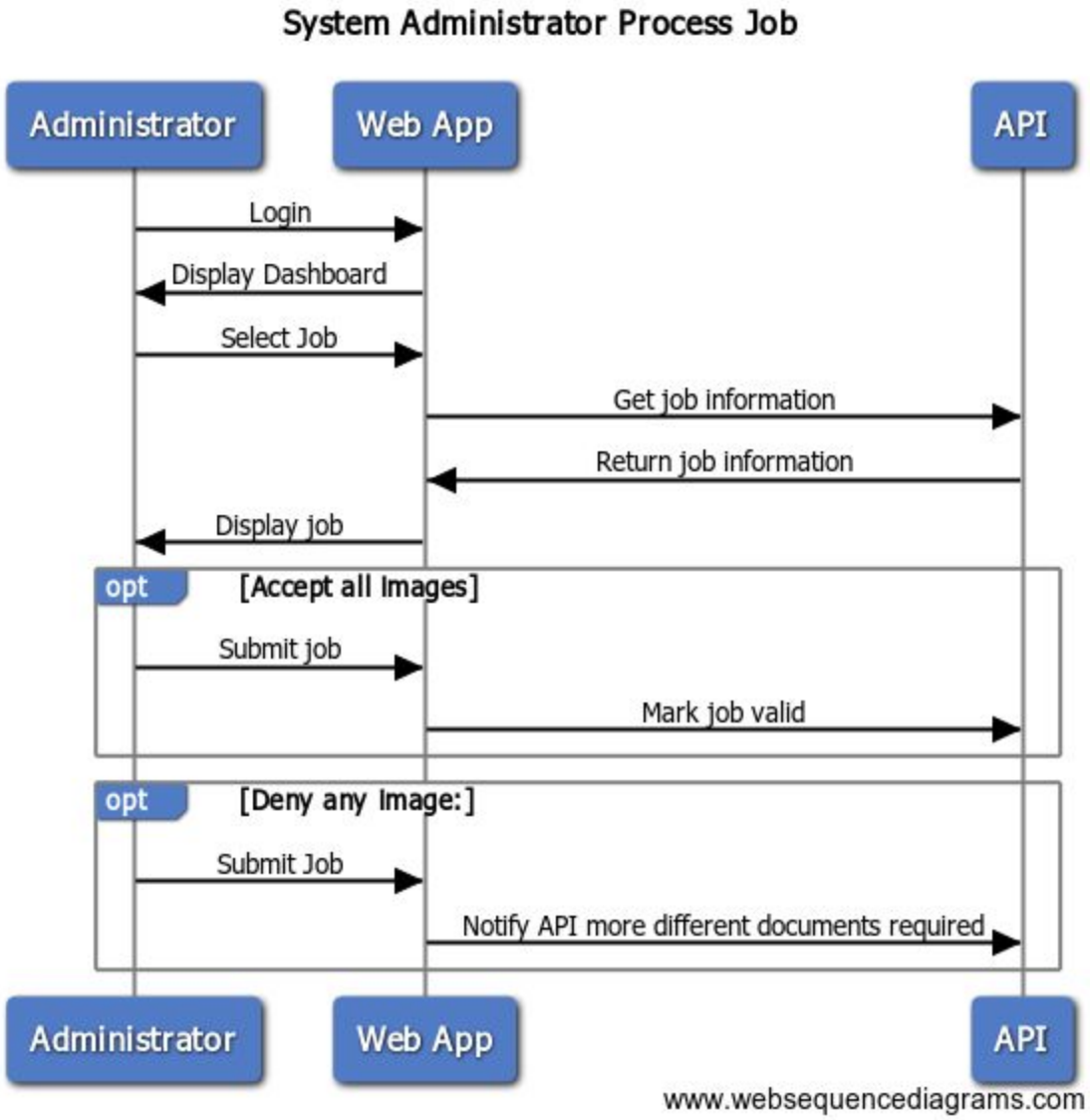
This diagram shows the high level sequence of events that a user and the system will go through when undertaking the login process.

Upload document



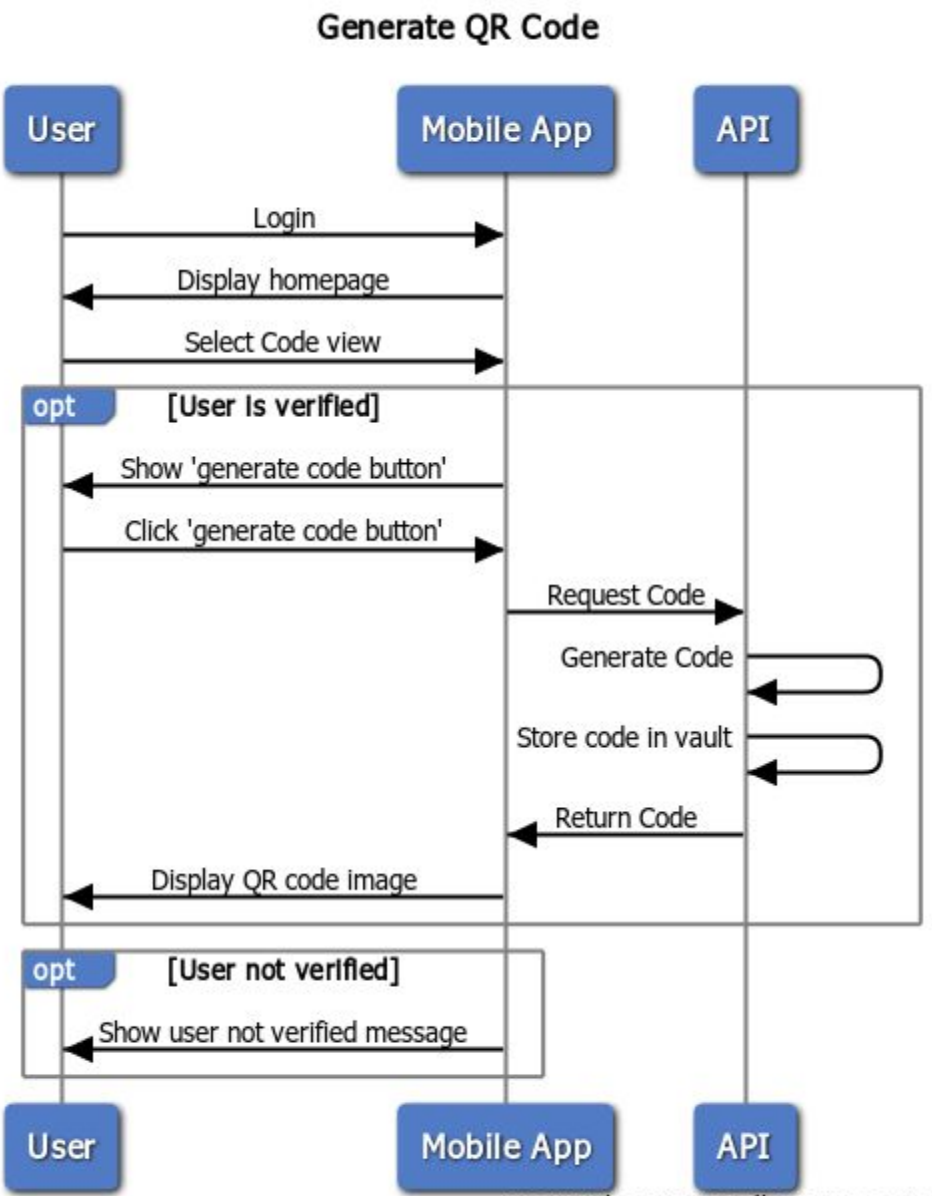
This diagram shows the high level sequence of events that a user and the system will go through when undertaking the process of uploading documents. In the paired service level diagram, it is shown that different types of image require different methods and levels of verification.

Administrator Process Job



This diagram shows the high level sequence of events that an administrator and the system will go through when processing a job for a particular user.

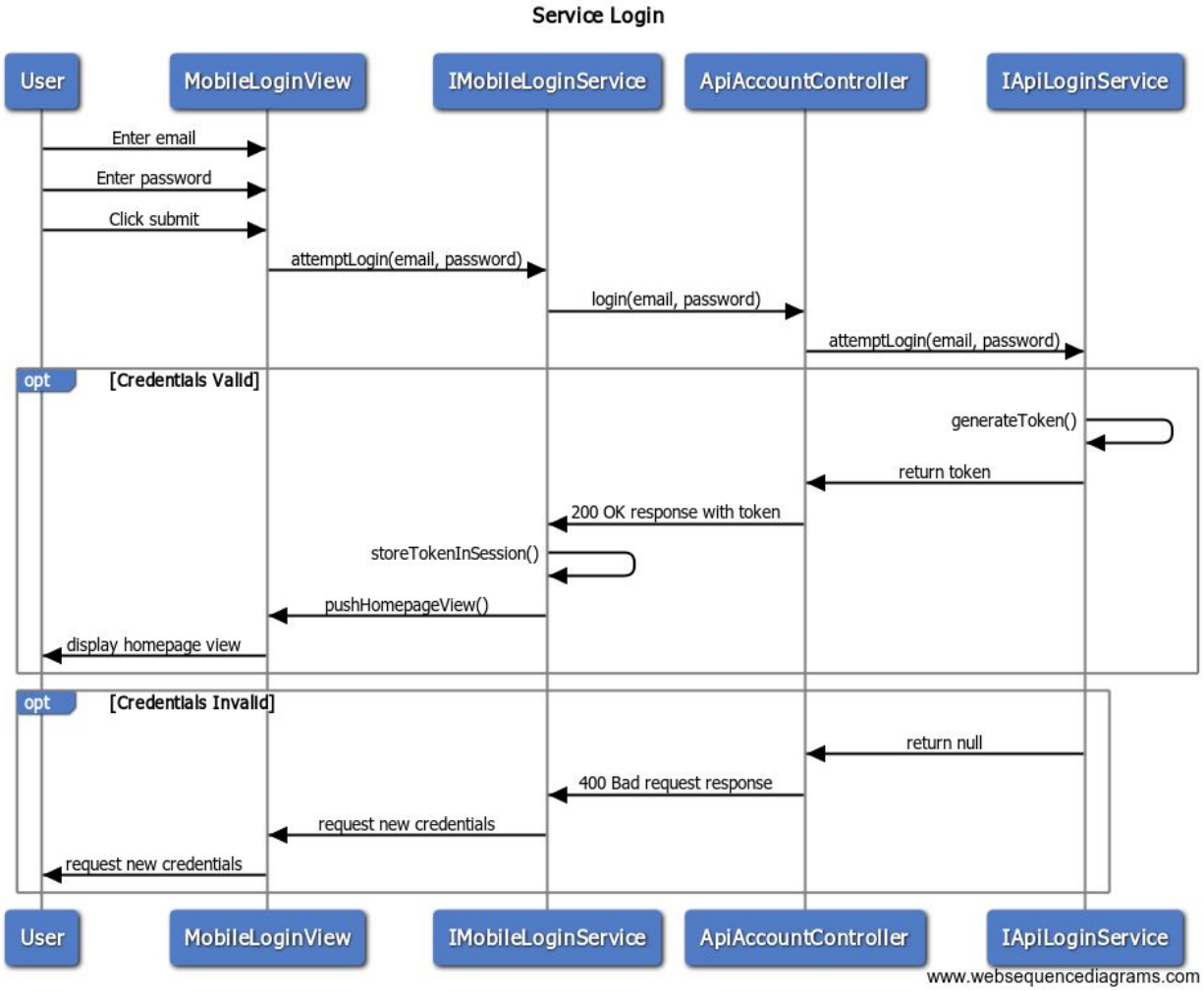
Generate QR code



This diagram shows the high level sequence of events that a user and the system will go through when processing a job for a particular user.

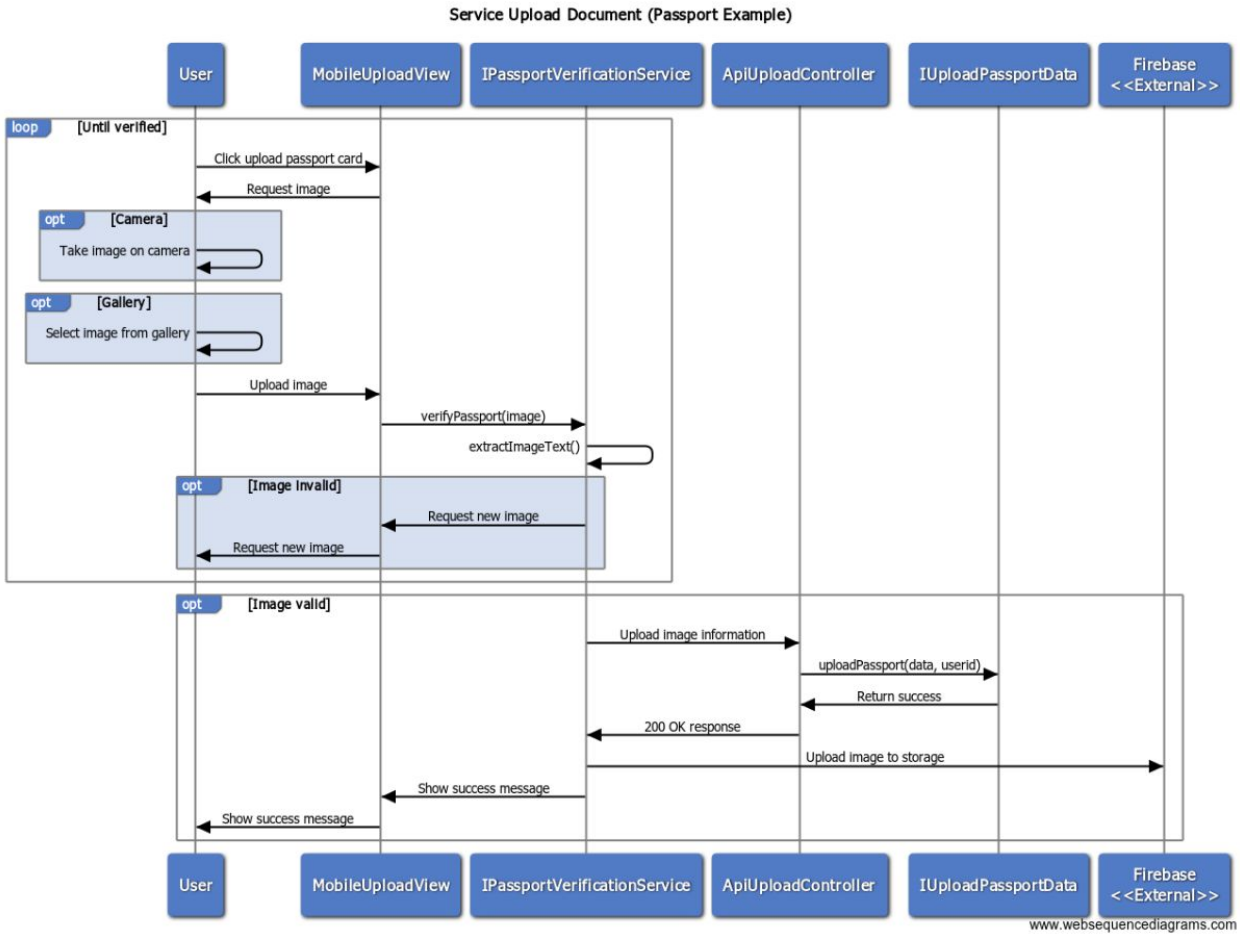
4.5.1 Service Level

Login



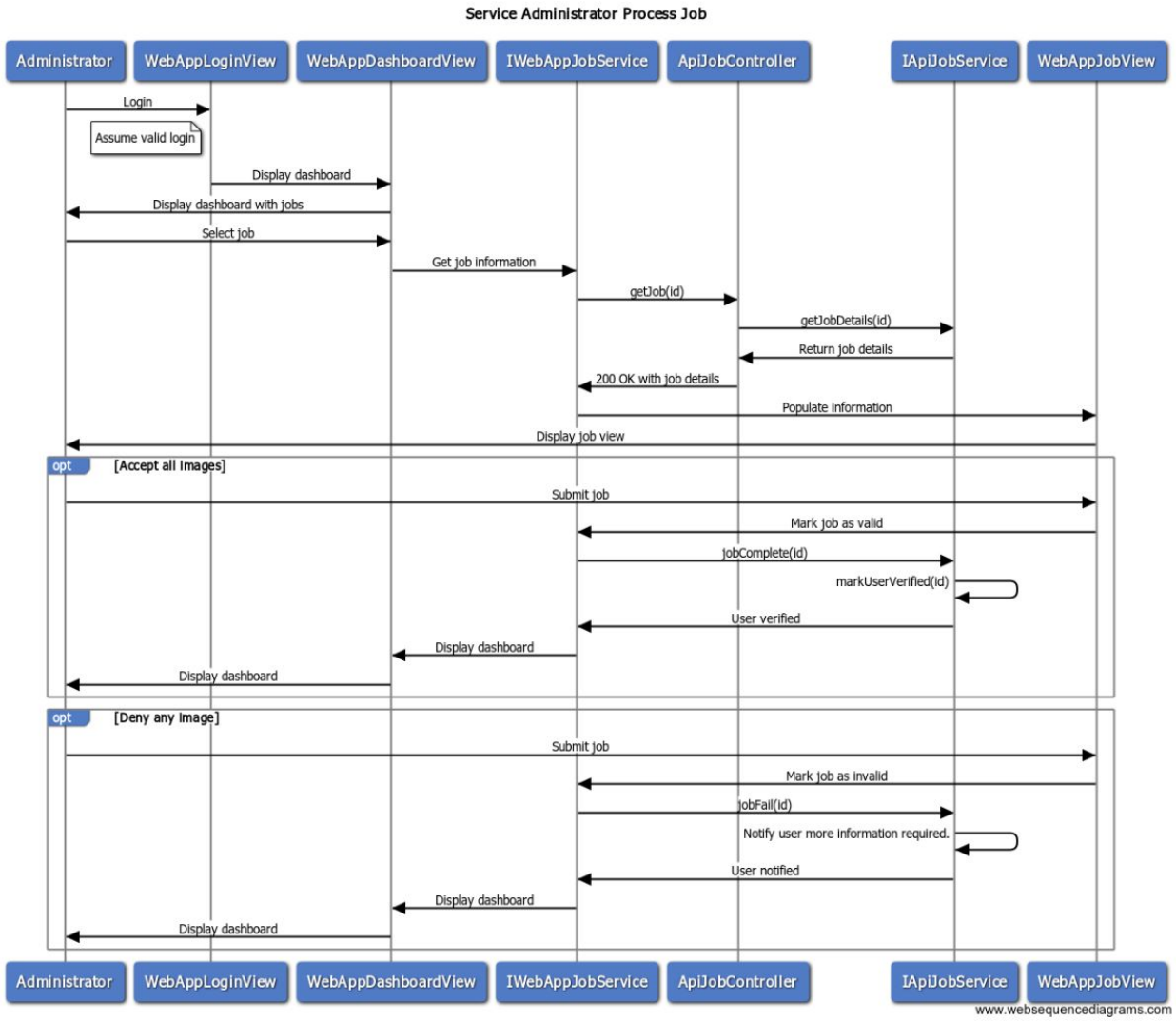
This diagram shows the servicelevel sequence of events that a user and the system will go through when logging in a user. Credentials are checked against existing users, if it exists, an authorisation token is generated and passed to the mobile app. This is stored in the session and used as and when necessary to authenticate against the API.

Upload document



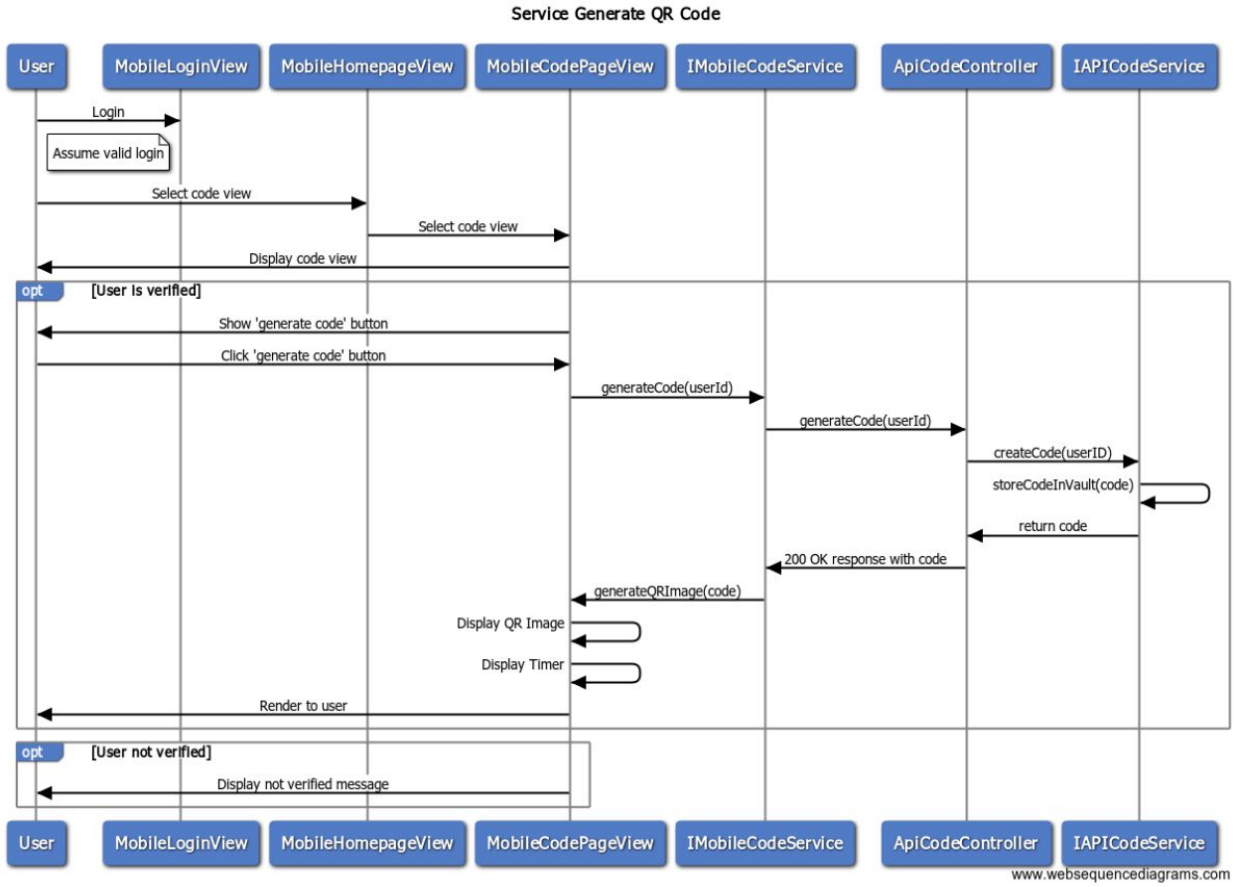
This diagram shows the service level sequence of events that a user and the system will go through when a user is uploading images of their identity documents. There will be slight variation for the 3 different required documents.

Administrator Process Job



This diagram shows the service level sequence of events that a user and the system will go through when an administrator is processing a job.

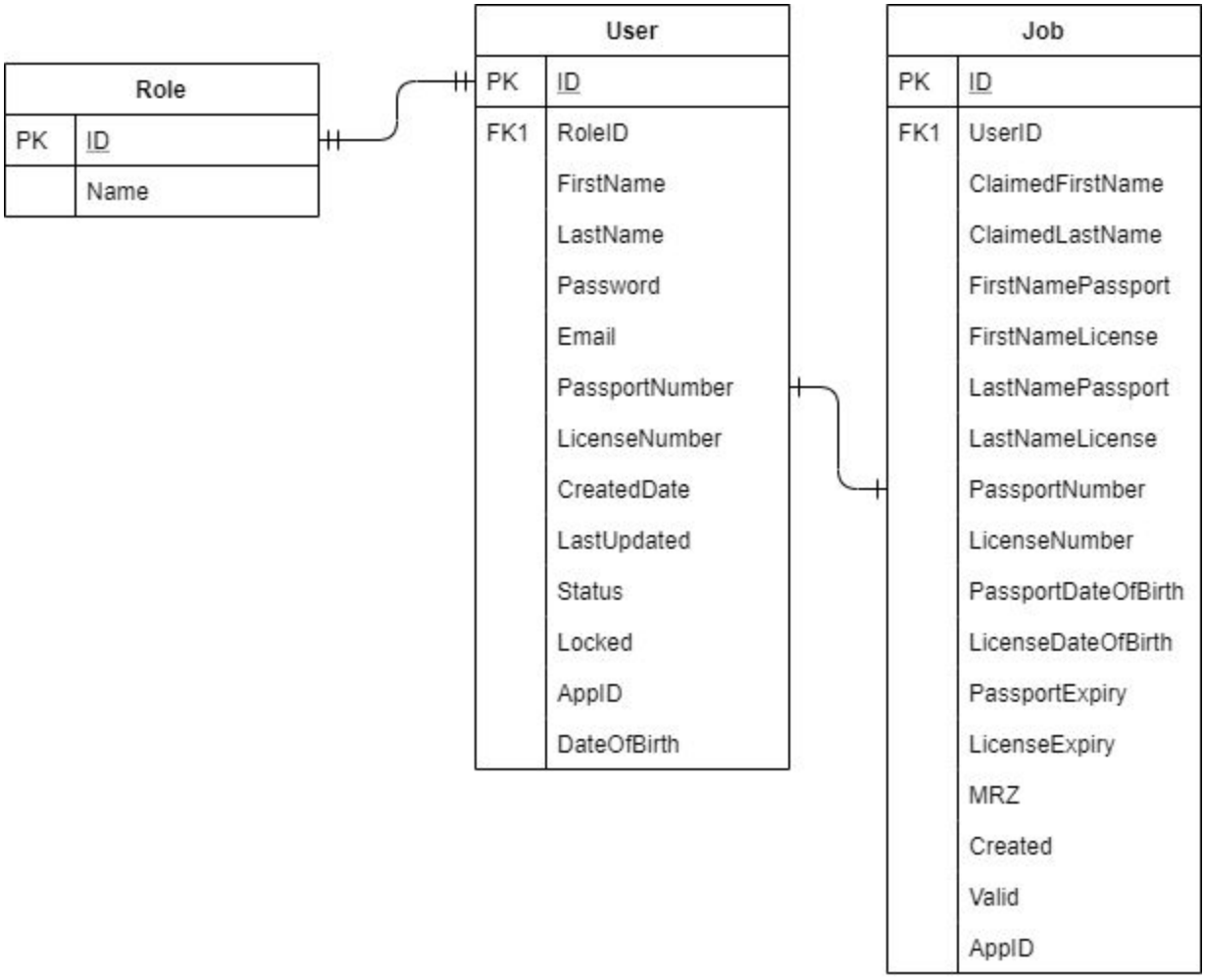
Generate QR code



This diagram shows the service level sequence of events that a user and the system will go through when a user requests an identity code.

4.6 Database Design

The below diagram is the entity relationship diagram for the required tables and columns of the API database. It represents how the data will be organised and persisted as well as the relationships between the data.



Role Data Model

Field	Data Type	Rationale
ID	int	The primary key identifier.
Name	VarChar(10)	The name of the role.

User Data Model

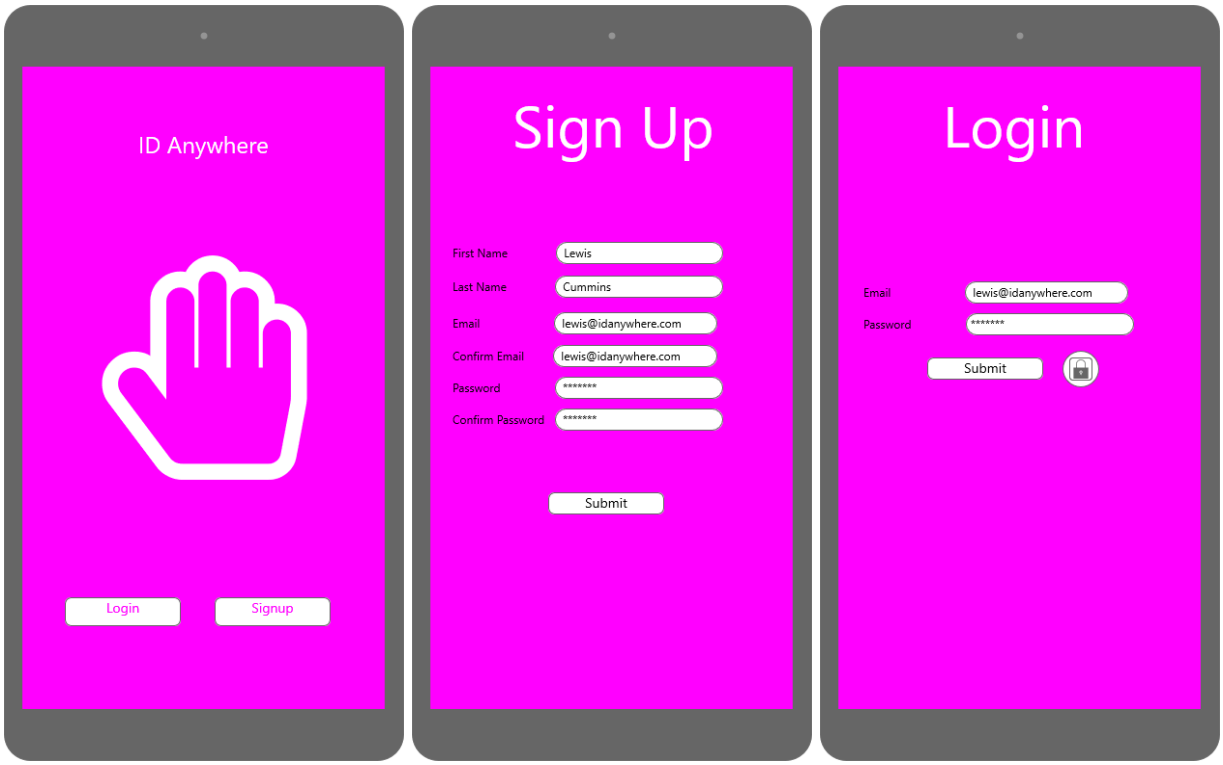
Field	Data Type	Rationale
RoleID	int	The ID of the user's role.
FirstName	VarChar(32)	The user's first name
LastName	VarChar(32)	The user's last name
Password	VarChar(512)	The user's hashed password
Email	VarChar(32)	The user's email address
PassportNumber	VarChar(9)	The user's passport number obtained after final job validation
LicenseNumber	VarChar(15)	The user's license number obtained after final job validation
CreatedDate	DateTime	The date the user was created
LastUpdated	DateTime	Last updated date, used to prevent concurrency issues.
Status	bool	Current verification status of the user.
Locked	bool	Whether or not the user is locked.
AppID	varchar(64)	The app id of the user's device
DateOfBirth	DateTime	The user's date of birth obtained after final job validation

Job Data Model

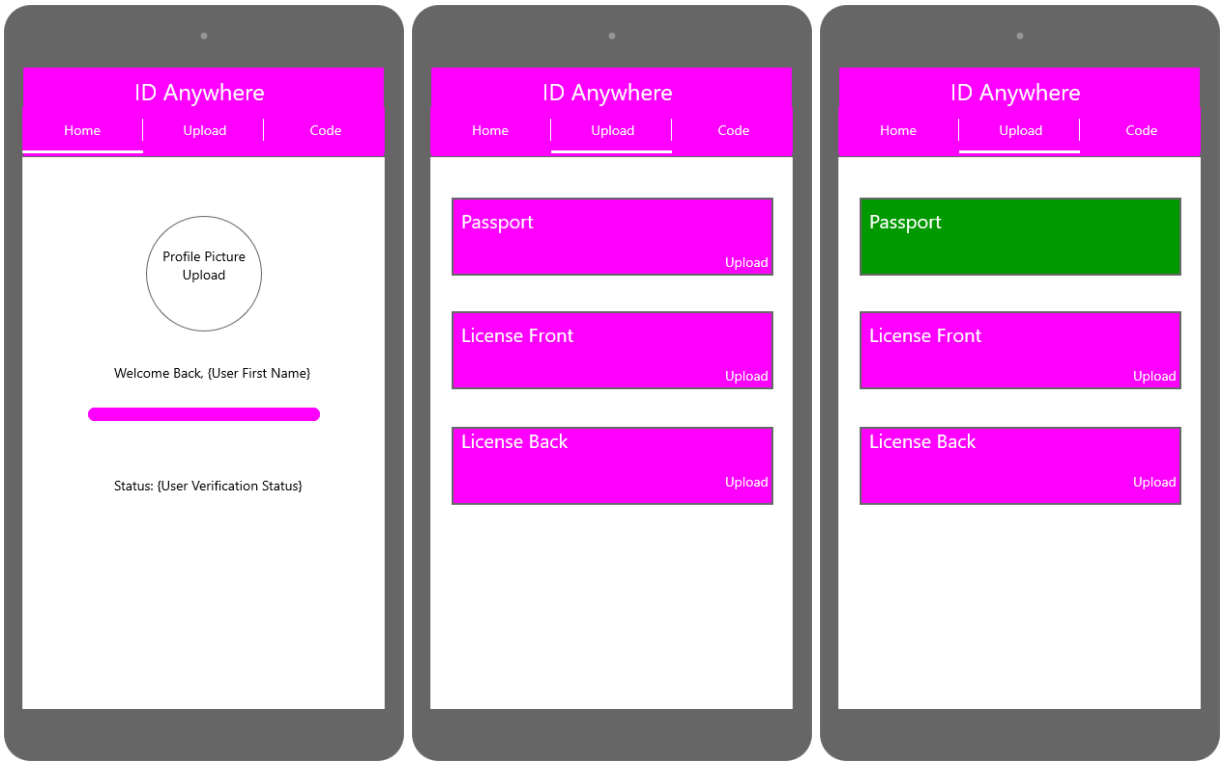
Field	Data Type	Rationale
UserID	int	Foreign key, The ID of the user's role.
FirstNamePassport	VarChar(32)	The first name extracted from the passport
FirstNameLicense	VarChar(32)	The first name extracted from the license
LastNamePassport	VarChar(32)	The last name extracted from the passport
LastNameLicense	VarChar(32)	The last name extracted from the license
PassportNumber	VarChar(9)	The id number extracted from the passport
LicenseNumber	VarChar(16)	The id number extracted from the license
PassportDateOfBirth	DateTime	The date of birth extracted from the passport
LicenseDateOfBirth	DateTime	The date of birth extracted from the license
PassportExpiry	DateTime	The expiry extracted from the passport
LicenseExpiry	DateTime	The expiry extracted from the license
MRZ	VarChar(88)	The machine readable zone text extracted from the passport
Created	DateTime	The time the job was created
Valid	bool	Whether or not the job is valid, all jobs that are valid are complete and subject to deletion.

4.7 Wireframes

The below diagrams are first draft user interface designs for the mobile application and web app. They are subject to change during implementation to allow tweaks in styling that cannot be conveyed through these images. This would be variables like colour of components as well as font style and size. This will be left up to the developer, in this case me, to decide on what looks best in the context of the provided wireframe.



The above three wireframes are the homepage, login and signup views for the mobile application. They are first draft ideas on how the app should look. The functionality that needs to occur will be derived from the requirements, use case diagrams and the sequence diagrams (if appropriate to this workflow).



These three wireframes show the homepage a user will see, with the option to upload their profile picture. The next two are the upload areas the user will need to upload their documents to. If an upload is successful, information is extracted and stored in the API, then the card will turn green, indicating a complete document. If the development reaches the point of applying actions on a user's account, then the green cards may turn back to pink if more information or a different image is required.

These final two wireframes show the code page of the mobile application, where a user can request an identity code in the form of a QR image. The QR code will only display if the user is verified. This will all be handled in the API, all the mobile application will need to do is wait for the code in the response and display it. When an external implementer of ID Anywhere scans this code, they will need to send it to the API's validation endpoint.

4.8 Test Plan

The test plan for ID Anywhere follows the fulfillment of UAT's (User acceptance tests) based upon the derived requirements. Each individual part of the architecture will be assigned a table like the one below.

ID	Test	Steps	Expected Outcome	Actual Outcome	Pass/Fail
TEST_ID	Ensure all work is complete	1. Do Work	Work is completed	Work is completed	Pass

A number of tests will be created to fulfill requirements, each step will be carried out for each of the tests. If the expected and actual outcome match, then the test will pass, else they will fail. Any failed tests will be reviewed and discussed in the evaluation. Having an automated unit test suite connected to a continuous integration / deployment pipeline would be the more robust solution and for a deployed application, is essential. However, due to the scale and nature of this application conflicting with time constraints, I will not be implementing this.

5.0 Implementation

5.1 Sprint Overview

The implementation of ID Anywhere will be split into sprints using a kanban board to monitor task progress. Requirements to be completed within a sprint will be chosen from the list of functional requirements. These sprints will aim to group likewise functionality that is necessary for the next sprint to commence, as well as ensuring any dependent requirements are completed. Due to the size of the project, not every line of code and piece of functionality will be documented in this section, but the most important parts of the requirements will be explained as well as any issues that are run into in every sprint. All non functional requirements will be implicitly adhered to throughout development and any choices made that are based upon them documented.

5.2 Sprint 1

During this sprint, the initial API and mobile application will be scaffolded. Login and signup functionality will be created, and any constraints appropriately put in place.

5.2.1 Requirements

- **FN01:** *The API must be able to create a new user during signup.*
- **FN02:** *The API must be able to login a user:*
- **FN03:** *The API database must be created with the designed schema.*
- **FN04:** *The API must be configured to sign and issue JWT for authentication and authorization purposes.*
- **FN05:** *The mobile application must provide a way for the user to create an account.*
- **FN06:** *The mobile application must provide a way to login a user that has signed up.*

5.2.2 Problems & Solutions

The API is created first, with two endpoints. https://DOMAIN:PORT/api/account/login and https://DOMAIN:PORT/api/account/signup. Endpoint routes in the API are mapped like

- api/ - This is always at the front of the endpoint for all API functionality
- [controller]/ - This is the name of the controller that holds the endpoint. In the case of login and signup, the controller is called AccountController.
- [action] - This is the name of the function in the controller

These two endpoints will be called from the mobile application and login application and the required information will be sent to them. Due to **NF01**, all communication must occur with TLS. Therefore, I will use the domain 'lewiscummins.dev' for all hosting, as using a defined domain allows for the use of a correct TLS certificate.

Sign Up

For the signup workflow, we require some personal information about the potential user. Following the wireframe for signup in **Section 4.7,** the mobile application will provide text boxes for:

- First name
- Last name
- Email (+ Confirm)
- Password (+ Confirm)

Once this information is input by the user, the information is sent via HTTP POST to the signup endpoint mentioned in the above paragraph. By sending this information in the JSON body of the request like so:

The .NET Core framework has mapping functionality in place to map information from the body and parameters of a http request to the parameters of the endpoint in the controller. In this case, we are expecting a SignUpVM parameter to be passed in. The body of the HTTP request will be read and any parameters with the same keys in the provided JSON and the SignUpVM class will be populated in the class instance and used as a parameter for the method.

```
public class SignUpVM
{
    [Required]

    public string FirstName { get; set; }

    [Required]

    public string LastName { get; set; }

    [Required]
    [DataType(DataType.EmailAddress)]

    public string Email { get; set; }

    [Required]

    public string Password { get; set; }

    [Required]

    public string AppID { get; set; }
}
```

All of this information is stored in the database in the User table, and the password is hashed via SHA512. provided all checks pass such as email is not already in use or the device they are signing up from is not already registered. Before the information can be stored, the database must be created using the database design from **section 4.6.** The way this is created, rather than writing SQL, is using our Object Relational Mapper tool Entity Framework Core to scaffold the tables from data models created in code. This is also known as Code First database design. Below are the images of our data models in code, which are then referenced in the principal class known as the API Context, this is registered on startup of the application and is the bridge between the code and the database.

```
public class RoleDM : BaseDM
{

    public string Name { get; set; }
}
```

```
public class JobDM : BaseDM
{
    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string UserId { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string ClaimedFirstName { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string ClaimedLastName { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string FirstNamePassport { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string FirstNameLicense { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string LastNameLicense { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string LastNamePassport { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string PassportNumber { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string LicenseNumber { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public DateTime PassportDateOfBirth { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public DateTime LicenseDateOfBirth { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public DateTime LicenseExpiry { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public DateTime PassportExpiry { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string MRZ { get; set; }

    2 references | lewjc, 30 days ago | 1 author, 1 change
    public DateTime Created { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public bool Valid { get; set; }

    1 reference | lewjc, 30 days ago | 1 author, 1 change
    public string AppId { get; set; }
}
```

```
public class UserDM : BaseDM
{
    2 references | lewjc, 62 days ago | 1 author, 1 change
    public string FirstName { get; set; }

    1 reference | lewjc, 62 days ago | 1 author, 1 change
    public string LastName { get; set; }

    1 reference | lewjc, 62 days ago | 1 author, 1 change
    public string Password { get; set; }

    2 references | lewjc, 62 days ago | 1 author, 1 change
    public string Email { get; set; }

    2 references | lewjc, 62 days ago | 1 author, 1 change
    public long? PassportNumber { get; set; }

    2 references | lewjc, 62 days ago | 1 author, 1 change
    public string LicenseNumber { get; set; }

    0 references | lewjc, 62 days ago | 1 author, 1 change
    public DateTime Created { get; set; } = DateTime.Now;

    0 references | lewjc, 62 days ago | 1 author, 1 change
    public DateTime LastUpdate { get; set; } = DateTime.Now;

    4 references | lewjc, 62 days ago | 1 author, 1 change
    public UserStatus Status { get; set; }

    1 reference | lewjc, 62 days ago | 1 author, 1 change
    public bool Locked { get; set; } = false;

    5 references | lewjc, 59 days ago | 1 author, 1 change
    public string AppID { get; set; }

    2 references | lewjc, 30 days ago | 1 author, 1 change
    public DateTime? DateOfBirth { get; set; }

    [ForeignKey("Role")]
    0 references | 0 changes | 0 authors, 0 changes
    public int RoleID{ get; set; }

    2 references | 0 changes | 0 authors, 0 changes
    public RoleDM Role { get; set; }

}
```

```
public class ApiContext : DbContext
{


    public DbSet<UserDM> Users { get; set; }


    public DbSet<JobDM> Jobs { get; set; }


    public DbSet<RoleDM> Roles { get; set; }


    public ApiContext(DbContextOptions<ApiContext> options) : base(options)
    {

    }


    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<UserDM>().Property(u => u.Status).HasConversion<string>();
        modelBuilder.Entity<UserDM>().ToTable("User");
        modelBuilder.Entity<JobDM>().ToTable("Jobs");
        modelBuilder.Entity<RoleDM>().ToTable("Roles");

        base.OnModelCreating(modelBuilder);
    }
}
```

The connection string to the database also needs to be provided, via a configuration file, this will contain all of configuration for the API.

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=IdAnywhereApiDev;Trusted_Connection=True;MultipleActiveResultSets=true"
},
```

When the application starts, all data models registered in the API Context will be checked against the tables that exist in the database, if they already exist they are ignored but if they dont they are created. This gives us a great advantage that we can just write code and add or remove tables as and when necessary.

Biometric Registration

Once the information has been accepted by the API and stored in the database, a 200 OK response is sent to the mobile application. The application will then ask the user if it would like to register their username and password with their biometric fingerprint used on their mobile (Only if the functionality is supported). If yes, then whenever login occurs, fingerprint auth will always appear first. If no or it is not supported then the option still exists on the login page but will not appear automatically.

```
class LocalAuthenticationService {

  LocalAuthenticationService();

  final _auth = LocalAuthentication();
  Future<bool> isProtectionEnabled () async =>
    (await SharedPreferences.getInstance()).getBool(Flags.useLocalAuth) ?? true;

  bool isAuthenticated = false;

  Future<bool> authenticate(String reason) async {
    if (await isProtectionEnabled()) {
      try {
        return await _auth.authenticateWithBiometrics(
          localizedReason: reason,
          useErrorDialogs: true,
          stickyAuth: true,
        );
      } on PlatformException catch (e) {
        print(e);
      }
    }

    return false;
  }
}
```

Login

Depending on whether or not automatic biometric login was selected by the user, when the user opens the login page, they will either have to present their fingerprint on screen or enter their username and password (**See 5.3.3)**. Presenting the fingerprint retrieves the username and password stored locally in secure encrypted storage. This is then sent to the login endpoint, which maps the email and password into the LoginVM. Alternatively, if the user types their email and password, this information is sent to the endpoint. The main difference between these two methods is that, during biometric login the password is sent hashed, as it is stored hashed. During the login process, various information is checked about the account.

Ensuring the username and password match an account, ensuring the account is not locked etc.

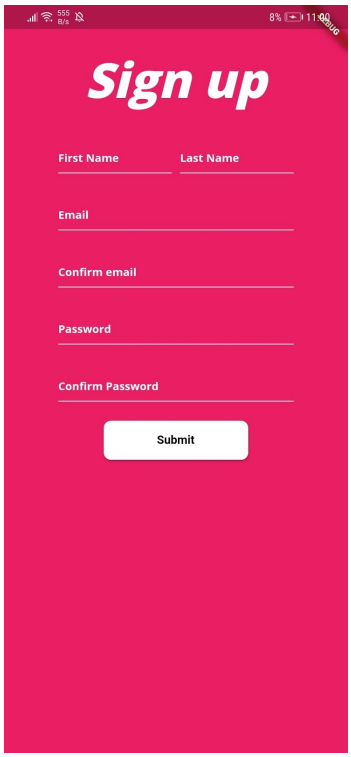
```
if (user == null)
{
  ServiceResult.Errors.Add("Email or password is incorrect.");
}
else if (user.AppID != sm.AppID && !isBypass)
{
  ServiceResult.Errors.Add("The device you are logging in from is not the one bound to your account.");
}
else if (user.Locked)
{
  ServiceResult.Errors.Add("This account is locked. Please unlock through the web portal.");
}
else
{
  string token = GenerateJwt(user);
  ServiceResult.Values.Add("token", token);
  ServiceResult.Values.Add("firstname", user.FirstName);
  ServiceResult.Values.Add("status", user.Status.ToString());
}
```

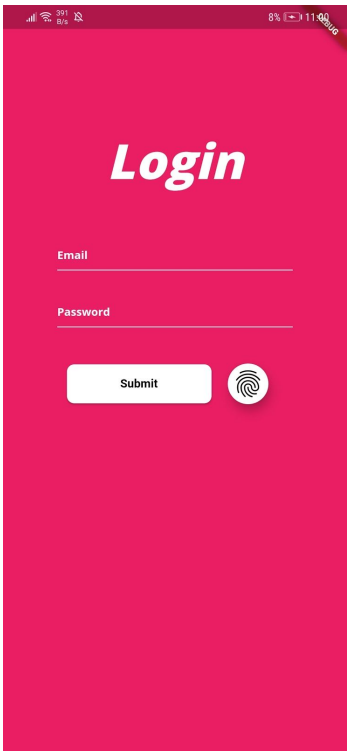
After all of the checks have passed, the JWT is created with claims relating to the user. This is then passed to the mobile application in the response, and it will be the mobile applications job to pass this in the 'Authorization' HTTP header whenever a request for a secured resource is made (marked by the authorization attribute in .NET Core).

```
1 reference | lewjc, 41 days ago | 1 author, 4 changes
private string GenerateJwt(UserDM user)
{
  // User exists and is not locked; let us generate a token and login.
  var tokenHandler = new JwtSecurityTokenHandler();
  var key = Encoding.ASCII.GetBytes(configuration.GetValue<string>("AppSettings:Secret"));

  var tokenDescriptor = new SecurityTokenDescriptor
  {
    Subject = new ClaimsIdentity(new Claim[]
    {
      new Claim(ClaimTypes.Name, user.ID.ToString()),
      new Claim("AppID", user.AppID),
      new Claim(ClaimTypes.Role, user.Role.Name)
    }),
    Expires = DateTime.UtcNow.AddHours(1),
    SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
  };
  var token = tokenHandler.CreateToken(tokenDescriptor);
  return tokenHandler.WriteToken(token);
}
```

This is now all of the development complete for Sprint 1, any images of the user interface development exists below in section **5.2.3**, These are still subject to change.

5.2.3 Development Images





5.3 Sprint 2

This sprint will focus on adding functionality to the mobile application so that the user can upload images of their documents as well as allowing the app to read information off of the images. The biggest challenge that will arise during this sprint will be the parsing of text from the images.

5.3.1 Requirements

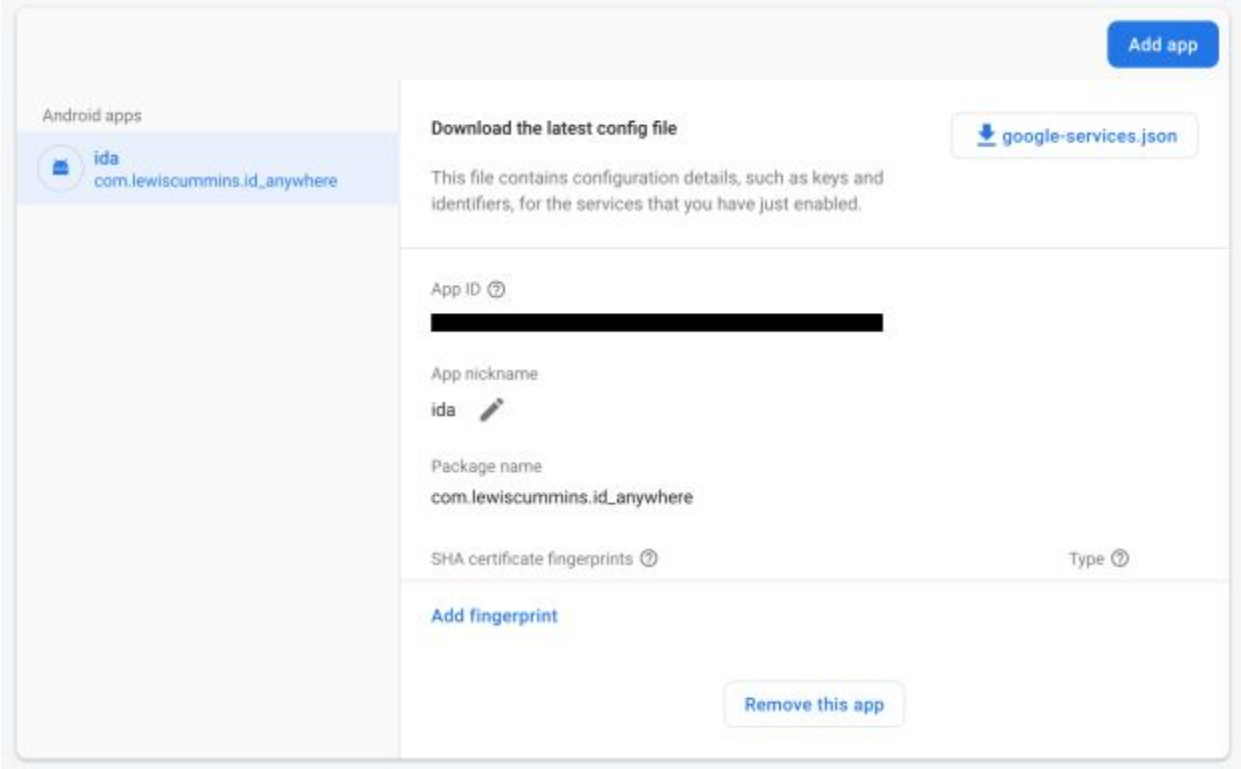
- **FN07:** *The Firebase cloud platform must be initialised to allow for cloud image storage*
- **FN08:** *The mobile application must allow the user to upload a profile picture to Firebase.*
- **FN09:** *The mobile application must allow the user to upload a picture of their driving license to Firebase.*
- **FN10:** *The mobile application must allow the user to upload a picture of the back of their driving license to Firebase.*
- **FN11:** *The mobile application must allow the user to upload a picture of their passport to Firebase*

- **FN12:** *The mobile application must read information off of the license to prove that it is a license image.*
- **FN13:** *The mobile application must read information off of the passport to prove it is a passport image.*

5.3.2 Problems & Solutions

Firebase

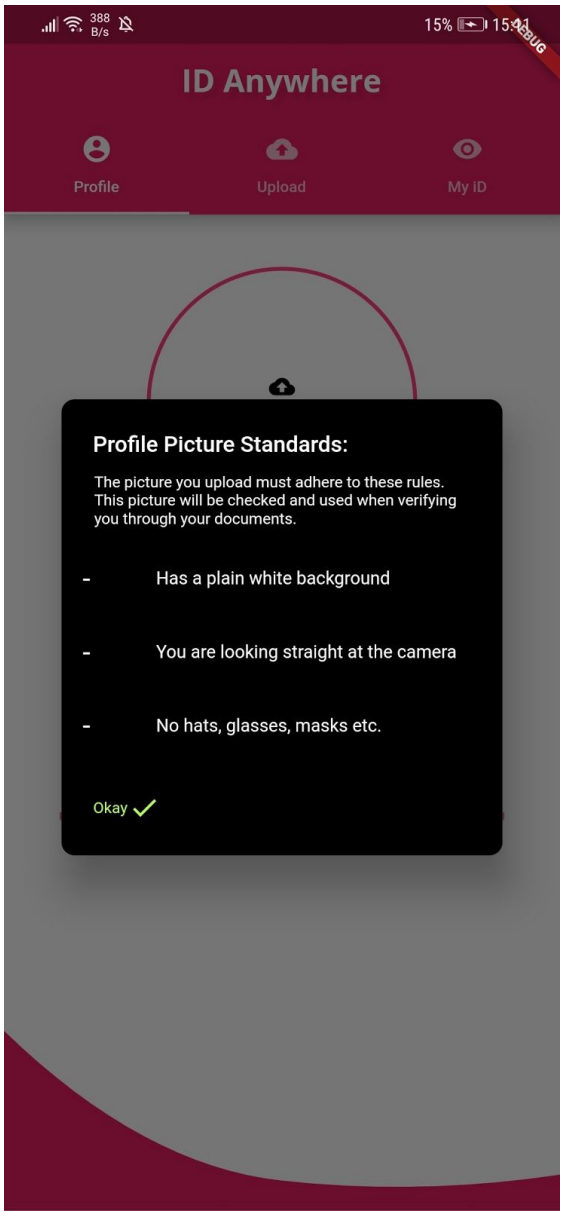
The first requirement that needs to be completed during this sprint is **FN07**. This is because most of the requirements are tied to uploading images to firebase. The setup of firebase is extremely straightforward. It has a free tier of use if you are below the usage requirements for the day. E.g. anything less than 50,000 requests in a day is free. You must add applications that are going to be using your Firebase Backend as a Service and a configuration file is provided with all of your allowed functionality.



For the sake of security, the app id has been redacted. From here, the google-services.json file is downloaded and inserted into your application. Now you can connect to any firebase services as long as you have the required packages downloaded into your project. Flutter has the complete range of Firebase packages all ready to install through the flutter package manager.

Profile Picture

When the user first logs in, they are presented with the homepage (replicated from the homepage wireframe). This is where they will be asked to upload an image of themselves, following the provided guidelines. The user clicks on the upload circle, and the guidelines are displayed.

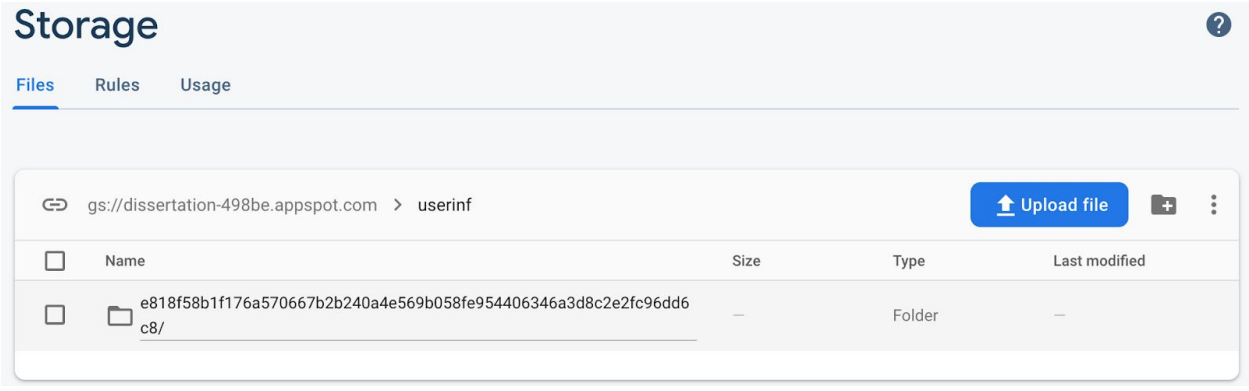


Flutter makes it really easy to display dialogs like the guidelines above. All that is needed is to call the 'showDialog' method on the current build context for the page. From there, a widget can be passed into the builder parameter. This is the widget that will be displayed.When guidelines are accepted, the user is provided with an option to select a picture from their gallery or take from their camera. This will be standard behaviour for all image upload.

Once the picture is taken or selected, it is sent to the firebase application and stored under a hash of the user's Device ID (this is the unique identifier set by the device manufacturer).

```
await showDialog(
    context: context,
    barrierDismissible: false,
    builder: (_) {
      return SimpleDialog(
        backgroundColor: Colors.black,
        shape: RoundedRectangleBorder(
          borderRadius: new BorderRadius.circular(10.0),
        ), // RoundedRectangleBorder
        title: Text(
          "Profile Picture Standards:",
          style: TextStyle(fontSize: 20, color: Colors.white),
        ), // Text
        children: <Widget>[
          Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                  Container(
                      padding: EdgeInsets.only(left: 5),
                      width: MediaQuery.of(context).size.width * 0.7,
                      child: Text(
                        ("The picture you upload must adhere to these rules. " +
                            "This picture will be checked and used when verifying you through your documents. "),
                        style: TextStyle(color: Colors.white),
                      )) // Text // Container
                ], // <Widget>[]
              ), // Row
              SizedBox(
                height: 10,
              ), // SizedBox
```

```
StorageReference cloudStorage =
    await FirebaseConnection().getUserStorageReference();
cloudStorage = cloudStorage
    .child(DeviceInfoHelper.hashId(await DeviceInfoHelper.getDeviceId()));
final StorageUploadTask uploadTask =
    cloudStorage.child("profile_picture").putFile(image);
```



Unified Upload Card

To fulfill requirements **FN09, FN10** and **FN11** the user needs a location to upload their documents. The best course of action to ensure minimal code replication is to create a reusable upload card that can be used for the passport, driving license front and back. The code snippet above shows the information required for each upload card. The key parameter is the imageSelectedCallback. This is the function that is executed once an image has been selected. By using this parameter, the card is now completely reusable as separate code can be executed for the passport, front of license and back of the license. In terms of creating the 3 upload cards the snippet below shows all the code required for the entire upload page.

```
@override
Widget build(BuildContext context) {
  // Here we need to do a lot of work.
  return Column(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.center,
    children: <Widget>[
      Spacer(),
      IDAnywhereUploadCard(
        title: 'Driving License Front',
        validateImageCallback: licenseService.verifyFront,
        informationDialog: license(true),
        flag: Flags.frontLicenseUploaded,
        complete: widget.frontOfLicenseUploaded,
      ), // IDAnywhereUploadCard
      IDAnywhereUploadCard(
        title: 'Driving License Back',
        validateImageCallback: licenseService.verifyBack,
        informationDialog: license(false),
        flag: Flags.backLicenseUploaded,
        complete: widget.backOfLicenseUploaded,
      ), // IDAnywhereUploadCard
      IDAnywhereUploadCard(
        title: 'Passport',
        validateImageCallback: passportService.verify,
        flag: Flags.passportUploaded,
        informationDialog: passport(),
        complete: widget.passportUploaded
      ), // IDAnywhereUploadCard
      Spacer(),
      Container(
        color: Colors.white,
        child: CustomPaint(
          size: Size(MediaQuery.of(context).size.width, 296.2),
          painter: CurvePainter(),
        ), // CustomPaint
      ), // Container
    ], // <Widget>[]
  ); // Column
}
```

Each card is passed a function from either the passport or license service, when an image is selected from the gallery or taken from the camera it is passed to the callback. Each service will attempt to extract information from the images and then upload them to Firebase. The information for the passport and front of the license will also be uploaded to the API, however the development of those endpoints will occur in Sprint 3.

Passport Image Information Extraction

To satisfy requirement **FN13**, key text must be extracted off of the passport image. By including a package called Firebase Vision and ML Kit, there is a prebuilt library for reading text off of images. By passing the image into the OCR engine, the text is returned in the form of blocks and lines, where a block is a block of text identified and a line is all the lines in the block. One major issue faced with identification of text from a passport is that it is rather small text and there is a lot of detail on the paper. So, while one heading of a passport reads “Given names/Prėnomes (2)”, it might extracted as “Givnnames/Prėnoms(2)”. Where characters are missing or in the wrong order or spaces are not interpreted. In order to get around this issue, I decided to compute the levenshtein distance for each extracted line where there was a number or bracket at the end of it, which was found via regular expression.

```
class LevenshteinAlgorithm {
  static Future<bool> run(String a, String b, int threshold) async{
    a = a.toLowerCase();
    b = b.toLowerCase();

    if(b.length < (a.length / 2)){
      return false;
    }

    if (a?.isEmpty ?? true) {
      if (b?.isNotEmpty ?? true) {
        return b.length < threshold;
      }
      return 0 < threshold;
    }

    if (b?.isEmpty ?? true) {
      return threshold < a.length;
    }

    // If our source is greater than the
    if (a.length > b.length) {
      String temp = b;
      b = a;
      a = temp;
    }

    int m = b.length;
    int n = a.length;
    // create and initialise the distance matrix
    var distance = new List.generate(2, (_) => new List<int>.filled(m + 1, 0));

    for (int j = 1; j <= m; j++) {
      distance[0][j] = j;
    }

    int currentRow = 0;
    try {
      for (int i = 1; i <= n; ++i) {
        currentRow = i & 1;
        distance[currentRow][0] = i;
        int previousRow = currentRow ^ 1;
        for (int j = 1; j <= m; j++) {
          int cost = (b[j - 1] == a[i - 1] ? 0 : 1);
          distance[currentRow][j] = (min(
              min((distance[previousRow][j] + 1),
                  (distance[currentRow][j - 1])),
              (distance[previousRow][j - 1] + cost)));
        }
      }
    } on Exception catch (e) {
      print(e);
      return false;
    }

    return distance[currentRow][m] < threshold;
```

The levenshtein algorithm computes the distance between two strings. It works out how many insertions and/or deletions must be made to make the target string the destination string. By identifying the lines that are the titles of the information on the passport in code, we can find the values belonging to the titles. The information required is:

- First name
- Last name
- Expiry
- Date of Birth
- Passport Number
- MRZ

By creating a class called UploadCertainty and giving each a text string to find, we can find the levenshtein distance for each line that matches our regex, and if the levenshtein distance is less than 3, we are confident that we have found the title we are looking for.

```
final List<UploadCertainty> validators = [
  UploadCertainty(      Lewis Cummins, a month ago • Add verify licesnse and pas
      desiredString: "Passport No./Passeport No.", levenshteinThreshold: 3),
  UploadCertainty(desiredString: "Surname/Nom (1)", levenshteinThreshold: 3),
  UploadCertainty(
      desiredString: "Given names/Prénoms (2)", levenshteinThreshold: 3),
  UploadCertainty(
      desiredString: "Date of birth/Date de naissance (4)",
      levenshteinThreshold: 3),
  UploadCertainty(
      desiredString: "Date of expiry/Date d'expiration (9)",
      levenshteinThreshold: 3),
];
```

For example, say a line matches the regex and has a number at the end of the string. We know now that it is a title, we need to figure out which one it is. So, we compute the levenshtein distance of the found string against **each** one of the strings stored in the list of upload certainty objects. If the distance is less than the threshold (in this case 3) then we know that the title that is matched in the upload certainty is the one we need. Therefore, we can go to the next text line in the list to get the value. So, if the distance is less than 3 for the 2nd upload certainty in the list, then we know that the value in the next line will be the surname.

The MRZ is extracted differently, the last block is taken and all text found in the block added together in order, if the length is equal to 88, we know we have the MRZ as the length of any MRZ.

Once all information is extracted, the service will then validate all of the values and get them ready for sending to the API when the endpoints are complete.

License Image Information Extraction

As opposed to extracting information from the passport, the license image is a lot easier. The information is a lot clearer, and there is less noise on the image, making it easier for the OCR engine to extract. Going to great lengths and implementing levenshtein for the

license would be overkill and largely unnecessary. Instead, the start of each line of text found will be split by the '.' character. This is because lines of information will all start like 1. or 2. The information we need off of the license is similar to the passport:

- First name
- Last name
- Expiry
- Date of Birth
- License Number

```
final Map<String, Function(int, int, VisionText, LicenseModel model)>
    desiredLineIndicators = {
  "1": extractLastName,
  "2": extractFirstName,
  "3": extractDateOfBirth,
  "4b": extractExpirationDate,
  "5": extractLicenseNumber,
};
```

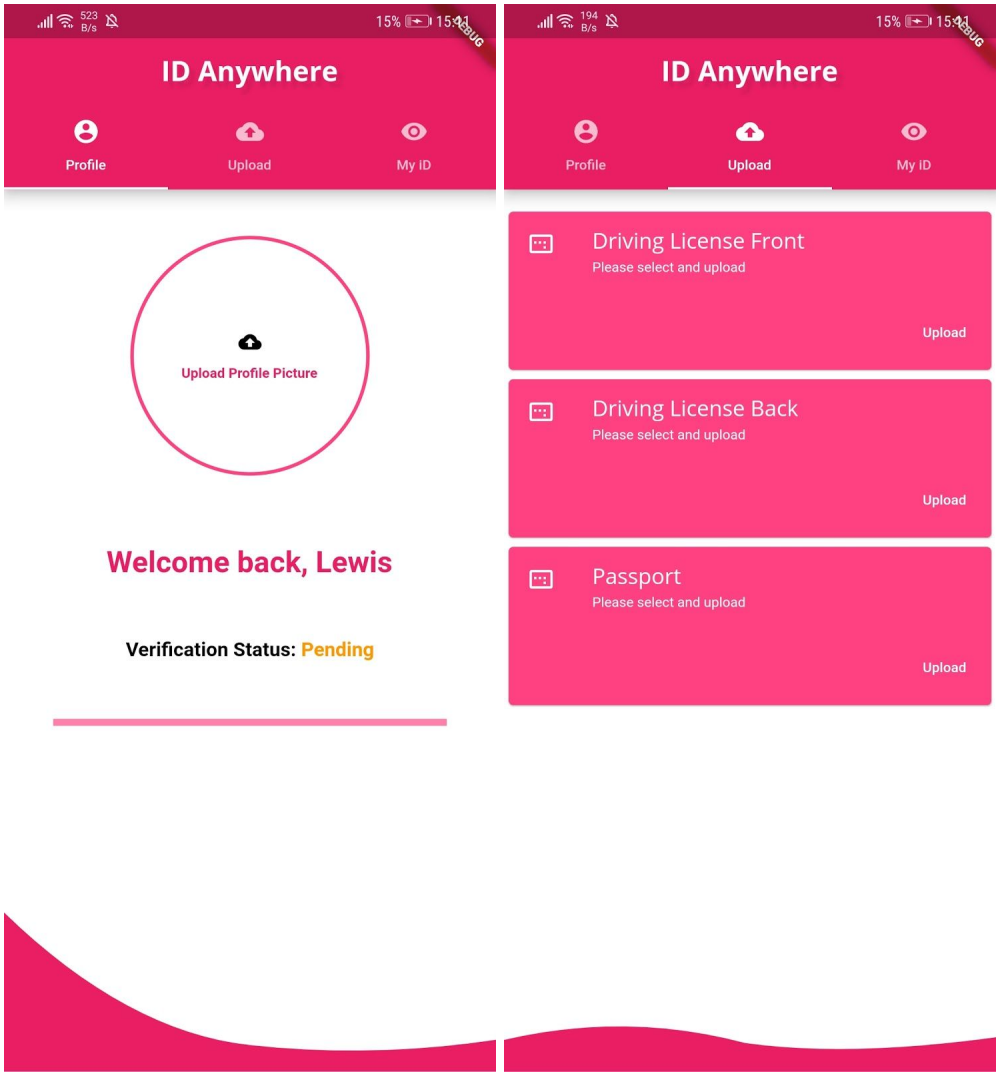
By passing the start of the line before a '.' we can map that to functions which will then look at the rest of the text in the line identified or the next line of text and attempt to find the value. An example of this is extracting the last name. If the start of the line begins with a 1 then the app will run the extractLastName method.

```
static void extractLastName(
    int currentBlock, int currentLine, VisionText text, LicenseModel model) {
  try {
    model.lastname =
        text.blocks[currentBlock].lines[currentLine].text.split(" ")[1];
  } on IndexError catch (e) {
    print(e);
    return null;
  }
}
```

Once all of the values have been found, the information will be uploaded to the API and the license image will be stored in firebase. For the back of the license, no information is required to be read off of it, the image is only uploaded for admins to validate during the final job processing.

This is now all of the development complete for Sprint 2, any images of the user interface development exists below in section **5.3.3**, These are still subject to change. Furthermore, the code for uploading to the API may need to be altered slightly as the endpoints will be created in Sprint 3, and some changes may occur.

5.3.3 Development Images



5.4 Sprint 3

This sprint will focus on adding the endpoints to the API that the mobile app may send the data to. From there, the API will store the information uploaded by the mobile application into the mongo db instance.

5.4.1 Requirements

- **FN14:** *The API must allow upload of information parsed from passport and license*
- **FN15:** *A mongo db instance must be set up to interact with the python daemon.*

5.4.2 Problems & Solutions

MongoDB Instance

Initialising a MongoDB instance is a simple process. It can be deployed locally or on a server elsewhere and communication can be made over port 27017 (MongoDB default port). For the purpose of development, I will deploy it locally. Once it is installed using the installer, the mongo action is added to your PATH environment variable, so it can be called from anywhere like so:

The mongo API used to interact with this instance is extremely intuitive and easy to use, as well as being extremely uniform across languages. We have to connect to this instance from our API so we can upload data to it. By installing the .NET Mongo API package into our project, we can initialize a client that is connected to our instance. We first provide configuration data on how to connect to the instance. In our case, it is on our local machine on the default port. The db and collection can be anything as mongo will create them if we try to access them and they do not exist.

This information is then subsequently loaded into the provided client class to form a connection.

```
3 references | lewjc, 31 days ago | 1 author, 1 change
public class MongoWorkQueue : IMongoWorkQueue
{
    private readonly MongoClient _client;
    private readonly IMongoDatabase _idAnywhere;
    private readonly IMongoCollection<WorkDocument> _workQueue;
    private readonly ILogger _logger;

    0 references | lewjc, 31 days ago | 1 author, 1 change
    public MongoWorkQueue(IConfiguration configuration, ILogger logger)
    {
        _logger = logger.ForContext<MongoWorkQueue>();
        try
        {
            var host = configuration["MongoWorkQueueSettings:Host"];
            var port = configuration["MongoWorkQueueSettings:Port"];
            var db = configuration["MongoWorkQueueSettings:Db"];
            var collection = configuration["MongoWorkQueueSettings:Collection"];
            var url = new MongoUrl($"mongodb://{host}:{port}");
            _client = new MongoClient(url);
            _idAnywhere = _client.GetDatabase(db);
            _workQueue = _idAnywhere.GetCollection<WorkDocument>(collection);
        }
        catch (Exception e)
        {

        }
    }
```

```
IMongoCollection<WorkDocument>
Gets a collection.
```

We now have a connection to the mongo db instance. If we register our 'IMongoWorkQueue' as a singleton service, we can then inject it into our services that are used by our endpoints defined late in the sprint to add and remove and update our information passed from the mobile application to the API.

API Endpoints

- **https://lewiscummins.dev:8000/api/upload/passport**
- **https://lewiscummins.dev:8000/api/upload/licensefront**
- **https://lewiscummins.dev:8000/api/upload/licenseback**

This is what one of the endpoints look like in code, and all 3 will be similar.

```
[HttpPost]
0 references | lewjc, 31 days ago | 1 author, 1 change
public async Task<ActionResult> Passport(PassportVM passportVM)
{

    if (ModelState.IsValid)
    {
        string userId = User.Claims.Where(x => x.Type == ClaimTypes.Name).FirstOrDefault().Value;
        string appId = User.Claims.Where(x => x.Type == "AppID").FirstOrDefault().Value;
        bool uploaded = await uploadService.AddPassportDataToJob(mapper.Map<PassportSM>(passportVM), userId, appId);
        if (uploaded)
        {
            return Ok();
        }
    }
    return new StatusCodeResult(400);
}
```

Our controller is also secured via the Authorize attribute.

```
[Route("api/[controller]/[action]")]
[ApiController]
[Authorize]
1 reference | lewjc, 31 days ago | 1 author, 1 change
public class UploadController : ControllerBase
```

This means we can use the JWT generated when a user is logged in (Sprint 1) and passed in the request from the mobile application in the authorization http header to find out information about our user, this will be their ID and their mobile application ID. This also means that if someone does not have a valid token created from the--+++++ API, they will not be able to upload anything, securing our resources. This behaviour will occur for most other endpoints to prevent unnecessary access.

So, when the user has uploaded a picture on their mobile application, and all information was successfully read and has been passed to the endpoint, then the business logic must take care of finding any information about the user already in a queue

Work Document Processing

The data stored in the queue must be stored in a document. This is a non-relational partition of data stored in key value pairs in the mongo collection. Below is an image of the various parts of the data stored. This will be retrieved from a python worker when it is running.

```
public class WorkDocument
{
    [BsonId]
    2 references | lewjc, 31 days ago | 1 author, 1 change
    public ObjectId ID { get; set; }

    9 references | lewjc, 31 days ago | 1 author, 1 change
    public PassportData PassportData { get; set; }

    8 references | lewjc, 31 days ago | 1 author, 1 change
    public LicenseData LicenseData { get; set; }

    3 references | lewjc, 31 days ago | 1 author, 1 change
    public bool BackLicenseUploaded { get; set; }

    7 references | lewjc, 31 days ago | 1 author, 1 change
    public string UserId { get; set; }

    3 references | lewjc, 31 days ago | 1 author, 1 change
    public string AppID { get; set; }

    1 reference | lewjc, 31 days ago | 1 author, 1 change
    public bool Ready { get; set; }
}
```

```
public class PassportData
{
    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string FirstName { get; set; }
    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string LastName { get; set; }

    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string Number { get; set; }
    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string MRZ { get; set; }

    2 references | lewjc, 31 days ago | 1 author, 1 change
    public DateTime DateOfBirth { get; set; }

    2 references | lewjc, 31 days ago | 1 author, 1 change
    public DateTime Expiry { get; set; }
}
```

```
public class LicenseData
{
    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string FirstName { get; set; }
    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string LastName { get; set; }

    2 references | lewjc, 31 days ago | 1 author, 1 change
    public string Number { get; set; }

    2 references | lewjc, 31 days ago | 1 author, 1 change
    public DateTime DateOfBirth { get; set; }

    2 references | lewjc, 31 days ago | 1 author, 1 change
    public DateTime Expiry { get; set; }
}
```

All of this information must be populated for the document to be marked as ready. When the API is attempting to add information to the work document for a particular user, the API will look to see if a work document exists already in the queue for the user, if it does, it adds the new information to it but if it does not, it creates the document, adds the information and then pushes it to the queue. This is shown in the below code snippet.

```
2 references | lewjc, 31 days ago | 1 author, 1 change
public async Task<bool> AddPassportDataToJob(PassportSM passportSM, string userId, string appId)
{
  var currentDocument = await _workQueue.FindByUserId(userId);
  if (currentDocument == null)
  {

    // This is the first thing that a user has uploaded. Create a new work task for them.
    var document = new WorkDocument()
    {
      UserId = userId,
      PassportData = CreatePassportData(passportSM),
      AppID = appId
    };

    bool didCreate = await _workQueue.CreateWorkDocumentAsync(document);

    if (!didCreate)
    {
      _logger.Error("Failed to create new work document using passport data.");
      return false;
    }

    CheckUploadJobComplete(document);
    // All good, work document created.
    return true;
  }
  else
  {
    currentDocument.PassportData = CreatePassportData(passportSM);
    bool didCreate = await _workQueue.UpdateWorkDocumentAsync(currentDocument);

    if (!didCreate)
    {
      _logger.Error("Failed to create new work document using passport data.");
      return false;
    }

    // All good, work document created.
    CheckUploadJobComplete(currentDocument);
    return true;
  }
}
```

5.5 Sprint 4

This sprint will focus on the development of the python daemon, and also the creation of the endpoint that the daemon uses to communicate back to the API information regarding the job it has finished processing.

5.5.1 Requirements

- **FN16:** *The python work daemon must be able to pick up work jobs stored in the queue and validate them.*
- **FN17:** *The API must be able to allow the daemon to update a user's status based on the processed information*
- **FN18:** *The API must create a final job if the result passed from the daemon shows information is valid.*

5.5.2 Problems & Solutions

Process Hierarchy

Following the component architecture for the daemon in **section 4.3.4**, The main process creates an instance of a work factory, then spawns this in a process. The work factory then creates as many processes as the user wants, using the workers variable in the work factory.

```
def main():
    # if(len(sys.argv) == 1):
    #     print("No arguments provided",)
    #     return
    print(header)
    print(
        "\n[STARTING ID ANYWHERE VERIFICATION SERVICE] VERSION: {0}\n ".format(
            version))

    valid_arguments = arg_parser((sys.argv.copy()).pop(0))
    if(not valid_arguments):
        print("Invalid arguments")
        return

    work_factory = work.workfactory.WorkFactory(1, 15)
    p = Process(target=work_factory.initialise)
    p.start()

    while(True):
        exit = input('')
        if(exit.lower() == 'exit'):
            print("Waiting for jobs to finish...")
            work_factory.finish_work()
            print("bye")
            p.terminate()
            sys.exit()
```

```
class WorkFactory():

    __worker_pool = []
    __worker_sleep = 0
    __finalise_work = False
    __workers = 0

    def __init__(self, workers=1, worker_sleep=15):
        self.__workers = workers
        self.__worker_sleep = worker_sleep
        print("[INITIALISED WORK FACTORY]")

    def initialise(self):

        print("[GENERATING WORKERS]")

        for x in range(self.__workers):
            uid = str(uuid.uuid4())
            worker = VerifyProcessWorker(uid, self.__worker_sleep)

            worker_proc = Process(target=worker.run,
                                  args=(self.__finalise_work,))
            worker_proc.start()
            print("[CREATED WORKER {}]".format(uid))
            self.__worker_pool.append(worker_proc)

    def finish_work(self):
        self.__finalise_work = True
        for worker in self.__worker_pool:
            worker.join()
```

The workers are appended to a pool, the factory can then keep track of them. When the command has been issued to finish work ready for exit, the boolean passed to the workers will be changed and the workers can act on that appropriately.

```python
class VerifyProcessWorker():

    __running_job = False
    __job_pipeline = None
    __job_refresh = 0
    __id = None

    def __init__(self, _id, job_refresh=15):
        self.__id = _id
        self.__job_refresh = job_refresh

    def run(self, exit_condition):
        self.__job_pipeline = MongoWorkQueue(parent=self.__id)

        try:
            while True:
                if(exit_condition):
                    return
                job = self.__job_pipeline.get_next_job()
                if(job is None):
                    time.sleep(self.__job_refresh)
                    continue

                print("[SELECTED JOB {}]".format(str(job["_id"])))
                self.__process_job(job)

        except Exception as e:
            print(e)

    def __process_job(self, job):
        update_user_status(*verify(job), id=job["UserId"])
```

Similar to the API, the worker processes must be able to connect to the work queue. Fortunately, mongo also provides a python API for connecting and communicating to a mongo instance. It is exactly the same as the .NET one, and has been wrapped in a class called MongoWorkQueue, on line 13 above.

Verification Process

Each worker will select a job then pass the job into the verify function, this performs all of the data comparison, facial recognition and the MRZ information extraction. One issue is that in order to perform recognition, images needed to be downloaded from firebase. This is possible through an early stage library to connect to firebase.

```
path = os.path.join(os.getcwd(), "dissertation-498be-firebase-adminsdk-od080-e7eb596b9e.json")
cred = credentials.Certificate(path)

app = firebase_admin.initialize_app(cred, {
    'storageBucket': 'dissertation-498be.appspot.com'
}, name='storage')
```

The image above shows the loading of the firebase settings for our application.

```
bucket = storage.bucket(app=app)

default_bucket = "userinf/{}/".format(job['AppID'])
passport_image_blob = default_bucket + "passport"
user_image_blob = default_bucket + "profile_picture"
passport_image_blob = bucket.blob(passport_image_blob)
user_image_blob = bucket.blob(user_image_blob)

passport_bytes = passport_image_blob.download_as_string()
profile_picture_bytes = user_image_blob.download_as_string()

passport_image = Image.open(io.BytesIO(passport_bytes))
user_image = Image.open(io.BytesIO(profile_picture_bytes))

check_face_result = __check_face(passport_image, user_image)
```

To download, the location must be specified and the bytes downloaded. Back in sprint 2, we uploaded images for a particular user in relation to their application ID, as it is unique. When images are needed to be downloaded, then the relative firebase storage location is simply userinf/{AppID}/{NameOfImage}.

```python
def compare_mobile_data(license_data: dict, passport_data: dict):

    today = datetime.datetime.now().date()
    error_count = 0

    if (not (license_data["FirstName"].lower() ==
        passport_data["FirstName"].lower())):
        error_count +=1
    if (not (license_data["LastName"].lower() ==
        passport_data["LastName"].lower())):
       error_count +=1
    if (not (license_data["DateOfBirth"].date() ==
        passport_data["DateOfBirth"].date())):
       error_count +=1
    else:
        if(not __verify_driving_license_number(license_data["Number"])):
            error_count +=1
        else:
            dob_from_number = license_data["Number"][5:11]      You, 22 days ago • Complete python
            year = dob_from_number[0]+dob_from_number[-1]
            month = int(dob_from_number[1] + dob_from_number[2])
            day = int(dob_from_number[3] + dob_from_number[4])

            current_year = datetime.datetime.now().year
            prefix = 20
            if(current_year - 2000 < int(year)):
                prefix = 19
            year = int("{}{}".format(prefix, year))
            dob_from_number = datetime.datetime(year, month, day).date()

            if(not dob_from_number == license_data["DateOfBirth"].date()):
                error_count +=1

    if(not license_data["Expiry"].date() > today and passport_data["Expiry"].date() > today):
        return -1


    return error_count
```

To allow for minor discrepancies in automatic data harvesting, an error count is kept for every piece of information that does not match from the passport and license. Rather than failing at one mismatch, the error count is returned from the function and can be used to determine whether or not the information is viable enough to go for human processing.

The worker will then need to notify the API as to whether or not the job passed or failed.

API Endpoint

```
[EnableCors("InternalVerificationService")]
0 references | 0 changes | 0 authors, 0 changes
public async Task<ActionResult> JobComplete(JobCompleteVM vm)
{
  if (vm.Verified)
  {
    var result = await finalJobPreperationService.CreateFinalJob(vm.UserId);

    if (result.Valid)
    {
      return Ok();
    }

    return BadRequest(new { result.Errors });
  } else
  {
    // Perform Actions on account
  }

  return BadRequest();
}
```

This is the endpoint that the python worker processes will notify the api as to a pass or fail. Rather than securing it with 'Authorize', CORS has been used. The InternalVerificationService Policy only allows requests that come from the domain of the python verification service.

```
var verificationServiceHosts =  Configuration.GetSection("AppSettings:VerificationServiceHosts").Get<List<string>>().ToArray();
services.AddCors(x =>
{
  x.AddDefaultPolicy(y => y.AllowAnyOrigin().AllowAnyMethod().AllowAnyMethod());
  x.AddPolicy("InternalVerificationService", y => y.WithMethods("post").AllowAnyHeader().WithOrigins(verificationServiceHosts));
});
```

This information needs to be provided in the API configuration

```
"AppSettings": {
  "Secret": "E1cILLLz5LGnkTGarHFa6hhB2y8rV6MZE1cILLLz5LGnkTGarHFa6hhB2y8rV6MZ",
  "VerificationServiceHosts": [ "localhost" ],
```

All host names in the list will be allowed to access the endpoint. The endpoint then creates 'Final job', this information is displayed to the administrator in the web application, to be completed in sprint 5.

5.6 Sprint 5

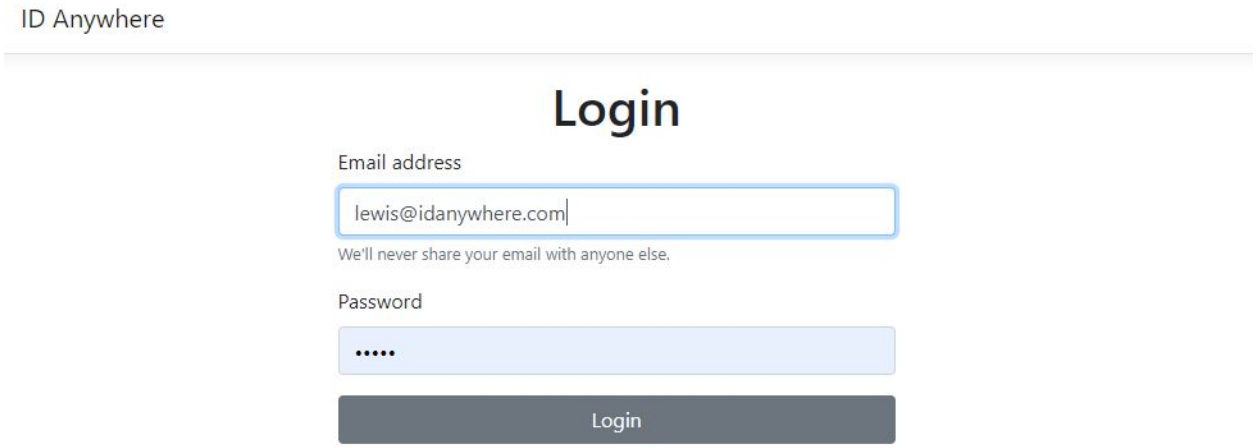
5.6.1 Requirements

- **FN21:** *The API must be extended to allow administrator accounts to be created by an existing admin*
- **FN22:** *The API must be extended to allow administrators to login*
- **FN23:** *The Web App must have a login page for users and admins*
- **FN24:** *The Web App must display a dashboard for admins when logged in.*

5.6.2 Problems & Solutions

This sprint is where the initial web application is created and a base of functionality added so that administrators can validate a user's documents.

When the application is run for the first time, an initial super admin account is created in the database. From there, the admin can log into the web application through the provided log in page below.



All login functionality still occurs in the API, so the username and password is presented to the web app and the data is sent to the API. Most communication will be done in this way and the code follows this format:

```
public async Task<LoginResult> AttemptLogin(LoginViewModel model)
{
  var request = new HttpRequestMessage(HttpMethod.Post,
      $"{configuration["Api:Url"]}account/login");
  request.Headers.Add("Accept", "application/json");
  StringContent content = new StringContent(JsonSerializer.Serialize(model), Encoding.UTF8, "application/json");
  request.Content = content;

  using var client = clientFactory.CreateClient();
  HttpResponseMessage response = null;
  try
  {
    response = await client.SendAsync(request);
  }
  catch (Exception e)
  {
    Console.Write(e);
  }

  if (response.IsSuccessStatusCode)
  {
    using var responseStream = await response.Content.ReadAsStreamAsync();
    var result = await JsonSerializer.DeserializeAsync
        <LoginResult>(responseStream);

    return result;
  }
  else
  {
    if ((int)response.StatusCode == 400)
    {
      var result = new LoginResult();
      using var responseStream = await response.Content.ReadAsStreamAsync();
      result.Errors = await JsonSerializer.DeserializeAsync
        <List<string>>(responseStream);
      return result;
    }
    return null;
  }
}
```

This code will be abstracted out to create a generic communication function to send and receive information from the API.

Once an admin is logged in, they can create another administrator account via the /Admin/CreateAdmin view.

Note: All styling has been omitted at this stage due to time constraints on finishing the project. This will be revisited in the final sprint if there is time to fulfill **NF14**.

Initial Dashboard

The administrator dashboard view will display jobs that the admin needs to complete. As of this sprint, the dashboard is currently empty. The functionality for making the dashboard interactive will be implemented in the following sprint.

5.7 Sprint 6

5.7.1 Requirements

- **FN25:** *The Web App must allow admins to select jobs from the dashboard*
- **FN26:** *The Web App must allow admins to view a job, and correct any information that has incorrectly been read from the images of the passport and license.*
- **FN27:** *The API must be able to provide an endpoint for the webapp to mark a job as complete based upon what they have found.*

5.7.2 Problems & Solutions

Requesting Jobs

When the administrator goes to their dashboard, a request for active jobs that need to be completed will be made to the API.

In order to only select a certain amount of jobs per page, manual paging has been implemented.

```
public List<JobSM> GetJobs(int pageSize, int pageNumber)
{
  var jobs = Db.Jobs.Where(x => !x.Valid).OrderByDescending(k => k.Created).Skip(pageNumber == 1 ? 0 : pageNumber * pageSize).Take(pageSize).ToList()
  return mapper.Map<List<JobDM>, List<JobSM>>(jobs);
}
```

These jobs need to be selectable by the administrator, and when selected display the information relating to the job. The Image below shows the dashboard populated with jobs.

To display the job, the .NET Core cshtml pages are utilised, allowing the integration of csharp code and HTML/CSS. The requested jobs can be passed into the page and rendered into HTML using tag helpers. This greatly reduces repeated code.

```
<div class="container">
  <div class="row">
    <div class="col-sm-12 text-center" style="overflow:scroll; overflow-x: hidden;
max-height: 75vh; min-height: 75vh">
      <h1>Jobs</h1>
      <div class="list-group" id="jobs-list">

        @foreach (var job in Model.JobsList)
        {
          <a asp-action="ProcessJob" asp-route-id="@job.ID" list-group-item list-group-item-action active">
            <div class="d-flex w-100 justify-content-between">
              <h5 class="mb-1">@job.ClaimedFirstName @job.ClaimedLastName</h5>
              <small>@job.Created.Date.ToLongDateString()</small>
            </div>
            <p class="mb-1">Please Complete this job by @job.Created.AddDays(7).Date.ToLongDateString()</p>
            <small>@job.GetHashCode()</small>
          </a>
        }
      </div>
    </div>
  </div>
  <br/>
  <div class="row">
    <div class="col-sm-12 ">
      <nav aria-label="...">
        <ul class="pagination justify-content-center">
          <li class="page-item disabled">
            <span class="page-link">Previous</span>
          </li>
          <li class="page-item"><a class="page-link" href="#">1</a></li>
          <li class="page-item active" aria-current="page">
            <span class="page-link">
              2
              <span class="sr-only">(current)</span>
            </span>
          </li>
          <li class="page-item"><a class="page-link" href="#">3</a></li>
          <li class="page-item">
            <a class="page-link" href="#">Next</a>
          </li>
        </ul>
      </nav>
    </div>
  </div>
```

Displaying Jobs

When a job is selected, all information relating to the images a user has submitted is populated into text boxes beside the image. In order to display the images, a download link is generated for 5 minutes for the images for the particular user the job is concerned with. This prevents the images being exposed for a large amount of time available for download.

Accepting and Denying

An administrator can accept or deny each image as it comes. They are tasked with:

- Making sure all information in the textboxes matches the images
- If any image does not look valid or is fake, the administrator must deny the useage.

First Name

LEWIS

Last Name

CUMMINS

Number

54[redacted]

Expiry

11-01-2027

Date of Birth

[redacted]

MRZ

[redacted]

Are you happy that:

- The image is of a valid passport
- All information is correct as seen on the image.

NO YES

First Name

LEWIS

Last Name

CUMMINS

Number

CUMM[redacted]

Expiry

31-01-2030

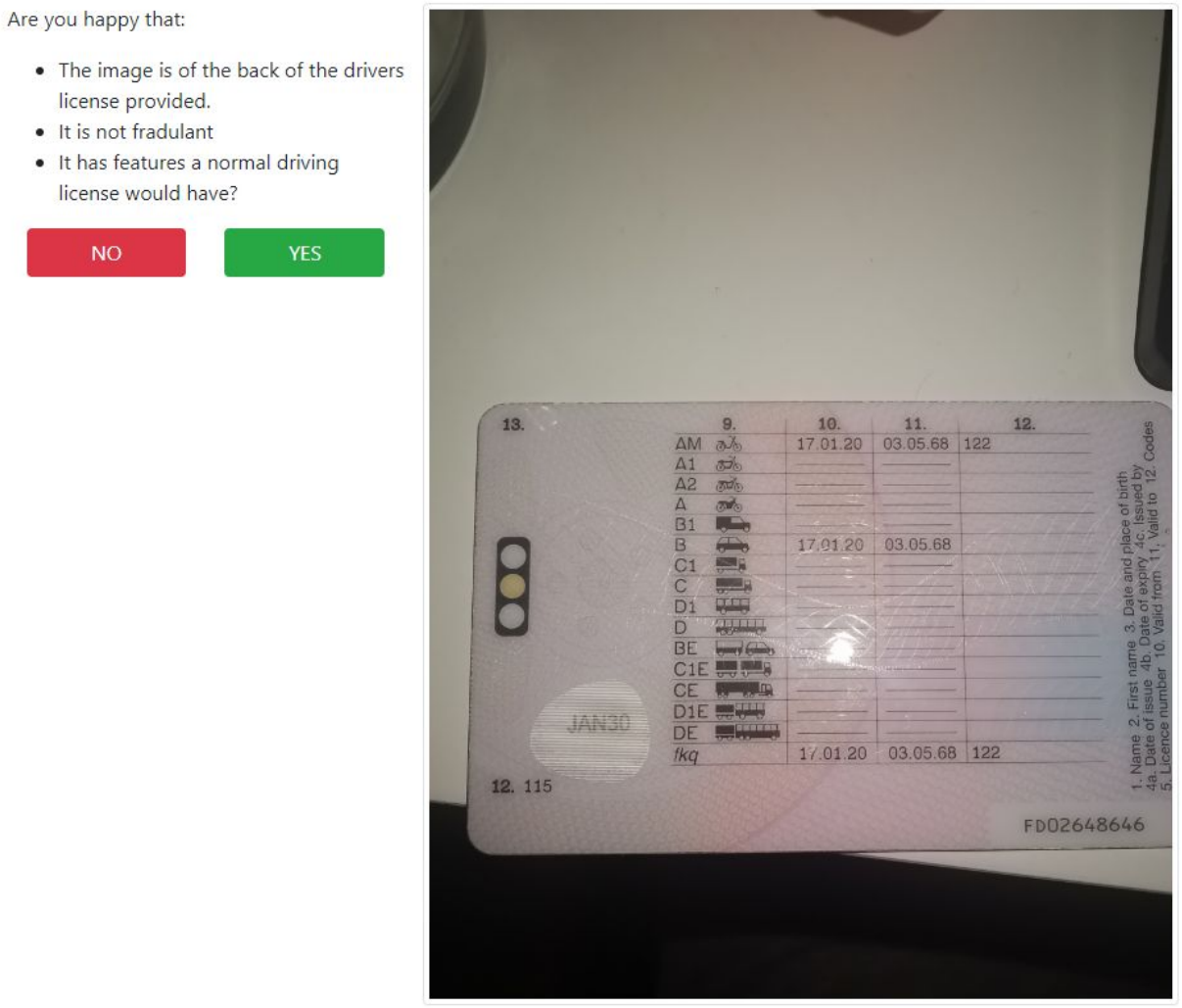
Date of Birth

[redacted]

Are you happy that:

- The image is of a valid license
- All information is correct as seen on the image.

NO YES



If no is clicked on any of the images, the user will not be validated, and an action added on the user account to update their documents. This will restart the process, and the information will have to once again be processed through the python worker.

If all images are accepted, the user is marked as verified. This allows them to generate codes for validation, this development will occur in the next sprint.

5.7.3 Development Images

5.8 Sprint 7

5.8.1 Requirements

- **FN32:** *The API must be able to generate a code that represents the user's profile and age, which can be used for validation.*

  In order for the user to be identified at a location, they will need a standardised QR code that is generated whenever they click a button. This code will be valid for a set amount of time to prevent sharing the code.

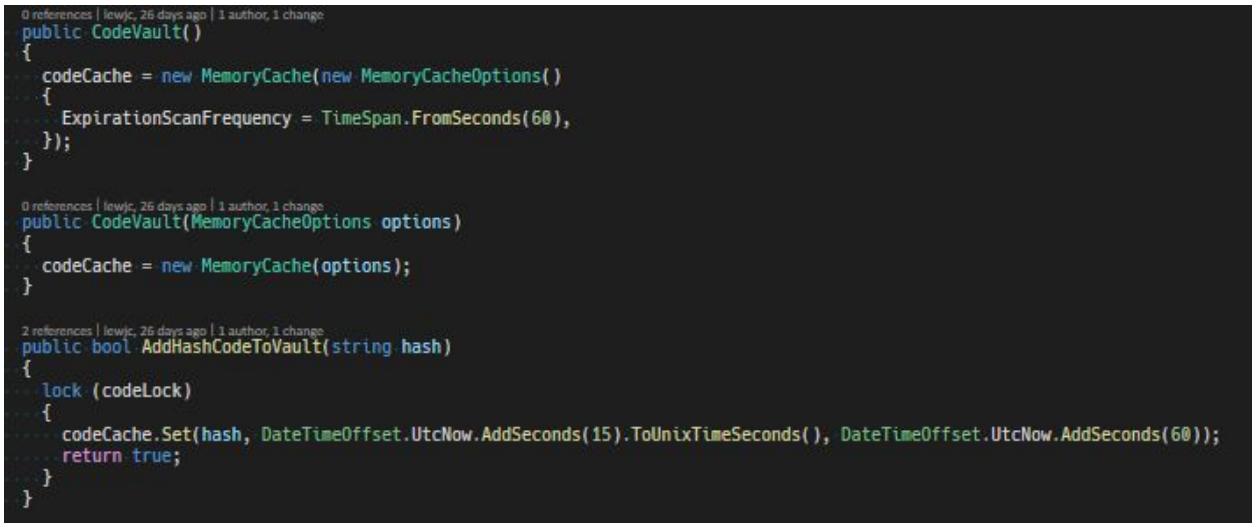
- **FN33:** *The API must be able to validate a code that represents the user's profile and age.*

  When the code is scanned by a client trying to validate the user and their age, the code will need to be sent to the api for validation. This needs to happen through a server to prevent forgery and tampering with codes.

- **FN34:** *The mobile app must be able to display a QR code that represents the user's profile and age, which can be used for validation*

5.8.2 Problems & Solutions

The biggest constraint around generating and storing a code is maintaining integrity and keeping it secure. The codes are generated through the API and stored in an in memory timed cache, which invalidates and removes the codes after 15 seconds.



```
0 references | lewjc, 26 days ago | 1 author, 1 change
public CodeVault()
{
    codeCache = new MemoryCache(new MemoryCacheOptions()
    {
        ExpirationScanFrequency = TimeSpan.FromSeconds(60),
    });
}

0 references | lewjc, 26 days ago | 1 author, 1 change
public CodeVault(MemoryCacheOptions options)
{
    codeCache = new MemoryCache(options);
}

2 references | lewjc, 26 days ago | 1 author, 1 change
public bool AddHashCodeToVault(string hash)
{
    lock (codeLock)
    {
        codeCache.Set(hash, DateTimeOffset.UtcNow.AddSeconds(15).ToUnixTimeSeconds(), DateTimeOffset.UtcNow.AddSeconds(60));
        return true;
    }
}
```

Code Generation

To generate a code, an authorized request must be made to the endpoint api/code/generateidentitycode.

```
public async Task<ActionResult> GenerateIdentityCode()
{
  string userId = User.Claims.FirstOrDefault(x => x.Type == ClaimTypes.Name).Value;
  var code = await codeService.CreateCode(userId);

  if (code == null)
  {
    return new StatusCodeResult(500);
  }

  return Ok(new
  {
    code
  });
}
```

To generate the code, four pieces of information are needed:

- The user's ID
- The user's passport number
- The user's license number
- A random GUID

This information is passed into a class called CodeBuilder. Following the builder pattern allows for the creation of the code in a contained way, allowing it to be reused whenever necessary. The code for the CodeBuilder is on the next page.

```
public async Task<string> CreateCode(string userId)
{
  var guid = Guid.NewGuid().ToString();
  var user = await Db.Users.FindAsync(int.Parse(userId));
  var passportNumber = user.PassportNumber;
  var licenseNumber = user.LicenseNumber;
  var appId = user.AppID;
  CodeBuilder builder = new CodeBuilder();
  var hashedCode = builder
    .AddAppId(appId)
    .AddGUID(guid)
    .AddLicenseNumber(licenseNumber)
    .AddPassportNumber(passportNumber.ToString())
    .Build();
  codeVault.AddHashCodeToVault(hashedCode);
  return hashedCode;
}
```

```
public class CodeBuilder
{
  private readonly Dictionary<string, string> builder;

  1 reference | lewjc, 26 days ago | 1 author, 1 change
  public CodeBuilder()
  {
    builder = new Dictionary<string, string>();
  }

  1 reference | lewjc, 26 days ago | 1 author, 1 change
  public CodeBuilder AddAppId(string appId)
  {
    builder["1"] = appId;
    return this;
  }

  1 reference | lewjc, 26 days ago | 1 author, 1 change
  public CodeBuilder AddPassportNumber(string passportNumber)
  {
    builder["2"] = passportNumber;
    return this;
  }

  1 reference | lewjc, 26 days ago | 1 author, 1 change
  public CodeBuilder AddLicenseNumber(string licenseNumber)
  {
    builder["3"] = licenseNumber;
    return this;
  }

  1 reference | lewjc, 26 days ago | 1 author, 1 change
  public CodeBuilder AddGUID(string guid)
  {
    builder["4"] = guid;
    return this;
  }

  1 reference | lewjc, 26 days ago | 1 author, 1 change
  public string Build()
  {
    if(builder.Keys.Count != 4)
    {
      throw new Exception("Not all data fileds required have been added to the CodeBuilder.");
    }
    string code = builder["1"] + "." + builder["2"] + "." + builder["3"] + "." + builder["4"];
    using var hasher = new SHA256Managed();
    var codeBytes = Encoding.UTF8.GetBytes(code);
    var password = hasher.ComputeHash(codeBytes);
    return BitConverter.ToString(password).Replace("-", string.Empty).ToLower();
  }
}
```

Code Validation

```
public async Task<ActionResult> VerifyIdentityCode(CodeVerificationVM viewModel)
{
  DateTimeOffset datetime = DateTimeOffset.UtcNow;
  if (ModelState.IsValid)
  {
    string userId = User.Claims.FirstOrDefault(x => x.Type == ClaimTypes.Name).Value;
    var sm = mapper.Map<CodeVerificationSM>(viewModel);
    var result = await codeService.ValidateCode(datetime, sm, userId);

    if (result.Valid)
    {
      return Ok();
    }
    else
    {
      return Ok(new { result.Errors });
    }
  }

  return BadRequest();
}
```

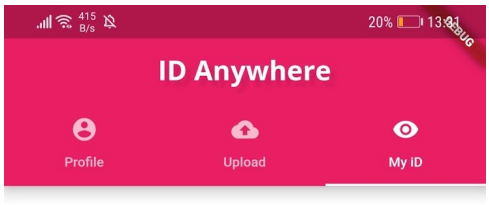
Validating a code is extremely simple, all that is needed is the code and the required age needed to be considered valid for the transaction that is occuring. As the code is only valid for 15 seconds from when it is generated, codes cannot be shared across multiple people. They are also hashed, making it impossible for the information to be seen.

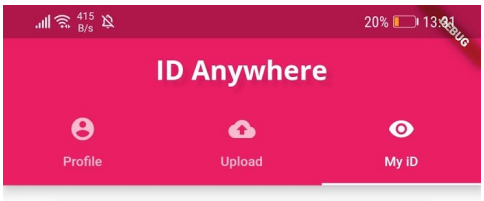
```
public bool ValidateHash(string hash, long currentTime)
{
  long? timestamp = 0;
  lock (codeLock)
  {
    timestamp = codeCache.Get(hash) as long?;
  }

  if (timestamp == null || timestamp == 0)
  {
    // Could not find code
    return false;
  }

  if (currentTime - timestamp < 0)
  {
    // Code is expired.
    return false;
  }

  return true;
}
```

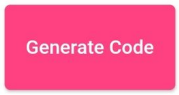
Displaying a code

In order to display a code on the mobile application for a user, the user must first be verified. If they are verified, then when they swipe to the 'My iD' tab, a button is displayed. On click, a request is made to the code generation endpoint. When the code is returned, it is stored and displayed as a QR code. This is valid for whatever the configured time is, defaulting to 15 seconds **(See 5.8.3 Development Images)**. All the external provider needs to do is scan the code and send it to the validate code endpoint along with the required age for the product that needs validating.

5.8.3 Development Images







Code valid for 15 seconds

5.9 Sprint 8

5.9.1 Requirements

- **FN19:** *The API should be able to create actions on a user account that the user can appropriately respond to*
- **FN20:** *The mobile app should be able to check any actions on the users account through the api and appropriately display them.*
- **FN28:** *The Web App should display a dashboard for users*
- **FN29:** *The Web App should allow users to lock and unlock their ID*
- **FN30:** *The Web App should allow users to unlock their ID tied to their device.*

5.9.2 Problems & Solutions

Due to the time constraints, this sprint was not completed. As these requirements were not considered a 'Must' for this iteration of the application, this is acceptable. However, the missing functionality will be discussed in the evaluation.

6.0 Testing

6.1 Overview

The user acceptance tests will cover the testing of requirements from a perspective of user functionality. If a test passes, it is implied that all components of the system used during the test performed their role correctly, and no errors occurred during the specific test flow of events.

6.1 Mobile app UAT

ID	Test	Steps	Expected Outcome	Actual Outcome	Pass/Fail
M_TC01	Sign up	1. Open Mobile App 2. Click Sign Up 3. Insert Details 4. Click Submit	User is created in API Database & redirected to login	User is created in API Database & redirected to login	Pass
M_TC02	Login	1. Open Mobile App 2. Click Login 3. Enter Username or Password 4. Click enter	User is logged in & redirected to the homepage	User is logged in & redirected to the homepage	Pass
M_TC03	Upload profile picture	1. Open Mobile App 2. Login 3. Go to homepage 4. Click upload profile picture 5. Select image 6. Profile picture uploaded 7. Profile picture displayed	Profile picture is uploaded to Firebase & displayed on the homepage.	Profile picture is uploaded to Firebase & displayed on the homepage	Pass

M_TC04	Upload passport	1. Open Mobile App 2. Login 3. Go to upload tab 4. Click upload passport 5. Select valid passport image 6. Text extracted from passport 7. Image uploaded to firebase 8. Data uploaded to work document	Passport data is extracted and uploaded to a work document and image uploaded to Firebase	Passport data is extracted and uploaded to a work document and image uploaded to Firebase	Pass
M_TC05	Upload front of license	1. Open Mobile App 2. Login 3. Go to upload tab 4. Click upload front of license 5. Select valid front license image 6. Text extracted from license 7. Image uploaded to firebase 8. Data uploaded to work document	License data is extracted and uploaded to a work document and image uploaded to Firebase	License data is extracted and uploaded to a work document and image uploaded to Firebase	Pass
M_TC06	Upload back of license	1. Open Mobile App 2. Login 3. Go to upload tab 4. Click upload back of license 5. Select valid back license image 6. Image uploaded to firebase	Back of license image uploaded to Firebase	Back of license image uploaded to Firebase	Pass
M_TC07	Generate Code	1. Open Mobile App 2. Login 3. Go to code tab	QR Code is scanned and valid for 15 then	QR Code is scanned and valid for 15 then it	Pass

		4. Click Generate code button 5. Code is displayed through QR code	it disappears.	disappears.	

6.2 Web App UAT

ID	Test	Steps	Expected Outcome	Actual Outcome	Pass/Fail
W_TC01	Administrator Login	1. Open Web Application 2. Go to login page 3. Enter email and password 4. Click Login 5. Redirect to Admin dashboard	Admin is logged in and dashboard is displayed with jobs	Admin is logged in and dashboard is displayed with jobs	Pass
W_TC02	Administrator Select Job	1. Go to Web Application 2. Login 3. Select Job from dashboard	Information about the job is displayed	Information about the job is displayed	Pass
W_TC03	Administrator Accept Job	1. Go to Web Application 2. Login 3. Select Job from dashboard	Job is marked as valid and user the job relates to is marked as verified	Job is marked as valid and user the job relates to is marked as verified	Pass

		4. Go through and edit all information that is incorrect for passport 5. Accept passport image 6. Go through and edit all information that is incorrect for front of license 7. Accept front license image 8. Accept back license image 9. Click submit			
W_TC04	Administor Deny Job	1. Go to Web Application 2. Login 3. Select Job from dashboard 4. Deny Passport image 5. Deny front of license Image 6. Deny back of license image 7. Click submit	Job is marked as invalid and more information is required from the user	Job is marked as invalid and more information is required from the user however the functionality to notify the user is currently missing	Fail

The final user acceptance test failed due to missing functionality that was supposed to be completed in sprint 8. The requirements were not a “must have requirement’ however the functionality prevents a more fluid workflow for the user to prove their identity.

7.0 Project Evaluation

7.1 Overview

I have learnt a great deal developing ID Anywhere from an idea into a fully fledged proof of concept. This evaluation will cover the main learning outcomes I have covered, what I would do next for ID Anywhere and what I would've done if I had more time.

7.2 Learning Outcomes

7.2.1 Problem

From researching the problem, I have learnt that there is currently no way for identity to be verified 100% autonomously quite yet. Due to a lack of standards that are currently in place, It is challenging to implement a system in the area of identification as there will always be people attempting to get around the system. This means that there must be persistent monitoring of usage and constant security updates. Furthermore, It will take more projects like ID Anywhere and Yoti being created to help push the sector forward in fleshing out issues with digital identification.

7.2.2 Area

The whole sector of identity verification is extremely detail oriented, the business logic must be flawless and all of the legislation and red tape makes it a tougher area to approach when developing a system. What would be beneficial is having a product owner that was someone who knew a lot about the field and all of the bases that need to be covered. Therefore, if a system like this was to be developed further, a greater team would be needed with sufficient knowledge of the identity process to help the requirements engineering and design of the system. Having this team in place would help eradicate the issue I had of not completely adhering to the identity profiles and ensuring the system covered all of the legislative basis.

7.2.3 Technology

Technology improves so much year after year, and the libraries that I was able to use were extremely well written and the functionality was very good. However, there are still issues with optical character recognition of images. There is still a lot that can be improved and because of this, workarounds have to be made, such as my implementation of the levenshtein distance algorithm to recognise passport values. The architecture of the system was intended to be deployed through the azure cloud hosting platform. I had an issue with the cost of renting servers, so I was not able to complete the deployment of the application.

However, I did deploy it locally for testing and demonstration purposes and everything worked flawlessly. Therefore I am confident in saying that the cloud deployment would have been a success.

7.2.4 Project

During the implementation of ID Anywhere I learnt that every detail must be meticulously defined prior to the development of the application but that will only get you so far. Issues always seem to appear during development that were completely unforeseen during the requirements gathering and system design. I think that this makes a strong case for newer methodologies such as Agile + Scrum and reinforces the point that the waterfall methodology as a standalone is not viable for any system. One incorrect requirement or unseen issue can snowball into major technical debt as well as preventing deadlines from being met. However, even with another methodology such as Agile, this issue is not eradicated. I did not manage to complete all of the development ID Anywhere. It was around 95% complete but was missing some quality of life features that would make the application more fluid. I chalk this up to over ambition for the project. However, the application still works as intended but would not be viable for release just yet. Future development would need to occur to flesh out the workflows to improve the user experience with ID Anywhere.

7.3 Challenges & Improvements

If I had more time, the number one thing I would have modified and improved would have been the way JWT are signed and issued. Next time, I would abstract the functionality away from the API and create an identity server which could be used by the web application and API. This is because of the service based architecture ID Anywhere was built upon, the Secret used across the API and Web Application was tightly coupled. It had to exist in both locations to validate signatures. If all of this was done in a localised place, as opposed to on the API and Web App, the authorization and authentication would be seamless, intuitive and architecturally sound. Furthermore, I was not able to complete Sprint 8 due to the time constraints I had on my project. This problem has occured because the project was a bit ambitious and could have been made a bit smaller, especially for a first draft. I've definitely learnt to reduce project size to help manage expectations in the future. This ensures that the project is not rushed and all requirements are completed to a high standard, with minimal if any defects.

One other definite improvement that could be made would be incorporating taking a video during the verification phase. One example of this is Monzo. During the identity verification stage, the user is asked to take a video of them holding their identity document and saying

a phrase. This is then used as one part of evidence during verification. It does add additional complexity into validating the speech, and comparing the images + faces. However, it is a very good solution as it further prevents fraudulent documents and identity theft.

7.4 Closing Thoughts

Overall, I am extremely happy with how the first iteration of ID Anywhere has turned out. I'm going to be developing this further in my spare time as an open source project on github with the hope that one day it would be deployed in society. As outlined in the evaluation, I have learnt a lot and I am really glad I chose to attempt a project of this size. It has taught me the value of explicit requirements and the need for processes when developing applications. It will without a doubt help me with my career in the future after university.

Bibliography

Agarwal, R. (2018). Why Is Minimum Viable Product The "MVP" of Mobile App Development? - Algoworks. [online] Algoworks. Available at: https://www.algoworks.com/blog/minimum-viable-product-for-mobile-apps/

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001). Manifesto for Agile Software Development. [online] Agilemanifesto.org. Available at: http://www.agilemanifesto.org/

Dart.dev. (2020). Platforms. [online] Available at: https://dart.dev/platforms

Google. (2020). Firebase. [online] Available at: https://firebase.google.com/

Flutterbyexample.com. (2018). Setup and Tools. [online] Available at: https://flutterbyexample.com/setup-and-tools/ [Accessed 15 Jan. 2020].

GOV.UK. (2017). Biometric passports. [online] Available at: https://www.gov.uk/government/publications/biometric-passports-and-passport-readers/biometric-passports-and-passport-readers

GOV.UK. (2017). Money Laundering Regulations 2017. [online] Available at: https://www.gov.uk/government/consultations/money-laundering-regulations-2017

GOV.UK. (2019). Identity proofing and verification of an individual. [online] Available at: https://www.gov.uk/government/publications/identity-proofing-and-verification-of-an-individual/identity-proofing-and-verification-of-an-individual

Hoek, J. (2018). What is the Agile Development Cycle? A Quick Intro to Agile Development. [online] Mendix. Available at: https://www.mendix.com/blog/pursuing-a-full-agile-software-lifecycle/

Kieseberg, P., Leithner, M., Mulazzani, M., Munroe, L., Schrittwieser, S., Sinha, M. and Weippl, E. (2010). QR code security. Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia - MoMM '10.

Microsoft. 2020. SQL Server 2019 Microsoft. [online] Available at: <https://www.microsoft.com/en-in/sql-server/sql-server-2019

MongoDB. (2020). The Most Popular Database For Modern Apps. [online] Available at: https://www.mongodb.com.

Morleo, M., Cook, P., Bellis, M. and Smallthwaite, L. (2010). Use of fake identification to purchase alcohol amongst 15-16 year olds: a cross-sectional survey examining alcohol access, consumption and harm. Substance Abuse Treatment, Prevention, and Policy, 5(1).

Project Smart, 2020. Moscow Method. [online] Project Smart. Available at: https://www.projectsmart.co.uk/moscow-method.php

Python.org. (n.d.). What is Python? Executive Summary. [online] Available at: https://www.python.org/doc/essays/blurb/

Robertson, J. and Robertson, S. (2012). Volere Requirements Specification Template Edition 16. 16th ed. Atlantic Systems Guild Limited.

Sharma, L. (2016). WaterFall Model in Software Development Life Cycle | SDLC. [online] TOOLSQA. Available at: https://www.toolsqa.com/software-testing/waterfall-model/ [Accessed 15 Jan. 2020].

Stack Overflow. (2019). Stack Overflow Developer Survey 2019. [online] Available at: https://insights.stackoverflow.com/survey/2019

Statista. (2019). UK: smartphone ownership by age 2018 | Statista. [online] Available at: https://www.statista.com/statistics/271851/smartphone-owners-in-the-united-kingdom-uk-by-age/

Hive. (2020). What is a Kanban Board? | Hive, Project Management Tool. [online] Available at: https://hive.com/blog/what-is-a-kanban-board/

Inflectra.com. (no date). What is Agile Kanban Methodology? Learn the Methods & Tools. [online] Available at: https://www.inflectra.com/methodologies/kanban.aspx

Yoti. (no date). Your digital identity | Yoti. [online] Available at: https://www.yoti.com/

Appendix

Item A - Unused Identity Profile 1 (High Confidence) - (GOV.UK, 2019)

Check	Score (1st Evidence)	Score (2nd Evidence)
Strength	3	3
Validity	3	3

Check	Score
Activity	3
Identity Fraud	2
Verification	3

Item B - Unused Identity Profile 2 (High Confidence) - (GOV.UK, 2019)

Check	Score (1st Evidence)	Score (2nd Evidence)
Strength	4	2
Validity	3	2

Check	Score
Activity	2
Identity Fraud	2
Verification	3

Item C - Unused Identity Profile 1 (Very High Confidence) - (GOV.UK, 2019)

Check	Score (1st Evidence)	Score (2nd Evidence)
Strength	4	3
Validity	4	3

Check	Score
Activity	2
Identity Fraud	3
Verification	4