

**Faculty of Physical and Applied Sciences
University of Southampton**

Robert Anderson

01/04/2017

Can a Biometric Authentication Framework be used as a
secure and flexible way of Authenticating users for web based
services?

Project supervisor: **Oliver Bills**
Second examiner: **Mike Poppleton**

A project progress report submitted for the award of
MEng Software Engineering

Abstract

This report details the current work undertaken for the research, design and implementation of a biometric authentication framework for web-based services. This framework will allow web services to utilise biometric authentication methods in order to provide a seamless login experience for their users.

Research into the current methods of authentication has shown that web based security is a point of contention for users and despite efforts from several areas, mainly password managers, these contentions are yet to be fixed. This research also highlights some of the key aspects where biometrics trumps legacy methods of authentication, and discusses its current uptake within general user authentication.

The report also describes requirements, design documentation and a specification which will allow the system to be implemented.

Key Terms

SLD - Secure login device

TEE - Trusted Execution Environment

WVAPI - Web Validation API

WAAPI - Web Account API

CF - Central Framework, this includes the validation and controller modules

DB - Database

Contents

Acknowledgements	5
1. Introduction	6
1.1. Goals	6
1.2. Scope	6
2. Background & Literature Search	7
2.1. Background Research	7
2.2. Literature Review	8
2.2.1. Biometrics	8
Transforming the feature data	10
Biometric Cryptosystems	11
2.2.2. Cloud Security	13
Cloud Application Security	13
Data Security	14
2.2.3. Cloud Computing	17
2.2.4. Usability of Biometrics	19
3. Design & Planning	20
3.1. Initial Design Analysis	20
3.1.1. Initial Design Decisions	21
Overview	21
Device Versions	21
Communication methods	21
Notifications	21
API's	22
Extension Systems	22
3.1.2. Designing for Security	23
3.2. Planning & Project Management	25
Work Breakdown	25
Comparison to Initial plan & Risk Mitigation	26
Tools	28
Progress	28
4. Design Specifications and Implementation	29
4.1 General Communication	29
4.2 Device Design	29
4.3 Framework Design	33
4.4 Extension Systems Design	40

4.5 Service Design	41
5. Testing	45
5.1 Regression Testing	45
5.2 User Testing	46
5.3. Biometric Simulation	46
6. Evaluation	47
6.1 Security	47
Misuse Case 1: Inject New Request:	48
Misuse Case 2: Fake Services	48
Misuse Case 3: Editing Token Redirects	48
Misuse Case 4: Cross Client Logins	48
Misuse Case 5: Intercepting Communications	49
Misuse Case 6: Imitating Secure Login Devices	49
Misuse Case 7: Imitating the Framework	49
6.2 Approach	50
6.3 Technical Evaluation	50
6.3.1 Tools and Software	50
6.3.2 Biometrics	51
Anonymity Method	51
6.4 User Testing Results	53
Conclusion	55
Further Work	55
References	56
Appendix	60
Appendix A - Hardware Costs & Design Archive	60
Appendix B - Initial Design Specification:	60
Appendix C - Asset Analysis:	63
Appendix D -Threat and Control Analysis:	64
Appendix E - Misuse Cases	65
Appendix F - Biometrics Security Analysis	69
Appendix G - Risk Analysis	71
Appendix I - Raw Questionnaire Results	72
Appendix J - Project Brief	73

Acknowledgements

I would like to thank Oli and Mike for their continued guidance throughout the project, along with Benjamin Tams for the use of his open source biometrics library. Also, I would like to extend my thanks to any participants of my user testing, who gave me invaluable data on the usability of my system.

1. Introduction

Web based security has always been a point of contention for users, as they want something which requires little/no effort on their part, but is secure enough to prevent fraudulent use. This is especially prevalent nowadays, where users have multiple accounts with different companies, which leads to people using weak (or similar) passwords in order to reduce the effort they have to expend to log into the site. This has caused people to seek out different ways of authenticating themselves, which has led to the use of biometrics.

Current biometrics allow a user to seamlessly authenticate themselves locally, but this technology has not yet been fully extended into the web. This project aims to design, implement and evaluate such a solution.

1.1. Goals

See project brief (Appendix J)

1. Evaluate the security challenges of storing and collecting biometric data within the framework.
2. Create a working prototype of the framework, which is both flexible and secure.
3. Develop a set of applications to enable the user to seamlessly login to biometrically enabled web services.

1.2. Scope

- The project will use all tools needed in order to design and implement the above goals.
- The project will not touch on:
 - Evaluation of different types of biometric data
 - Accuracy & performance of the methods to collect the data

2. Background & Literature Search

2.1. Background Research

Despite the advances in web technologies, there has been little advancement in generic authentication for web services. Passwords, which are disliked by most users (Adams et al, 1999) and security experts alike, are still the mainstay validation methods for almost all web services - despite the problems they are plagued with. These problems mostly stem from users not being able to accurately remember passwords. This leads to users using the same passwords and although statistics vary, it has been reported up to 73% of people use the same passwords for accounts (Morrison, K.).

Efforts have been made by security experts to increase the security of passwords, often by increasing the complexity - which makes the password harder to guess for hackers (Morris et al, 1979) and coupling passwords with something such as two-factor authentication - but these have mostly fallen on deaf ears, with 75% of people saying they have never used two factor authentication before (Imperium, 2013).

In order to increase security, password managers have become more and more popular with web users. They allow the user to store passwords, either on a server or locally, and will automatically log users into their accounts, allowing a more seamless interaction. However, many users worry about the security of their passwords. As described by (Karole & Saxena, 2011), users considered there to be a significantly greater perceived risk when using web-based managers as opposed to USB password managers.

Cloud Computing has also been utilised by third party authentication services, such as online password managers. These allow the user to save their passwords to their account, and retrieve them on any device without having to remember them. However, when surveyed, users distrusted these types of password managers the most (compared to USB and mobile managers), but they did find them significantly more convenient. (Karole & Saxena, 2011).

Recent innovations into the biometrics market have seen it skyrocket, with biometric authentication on all new devices. This is exceptionally prevalent with fingerprint authentication, which has proven to be exceptionally popular with consumers. A study by (Karthikeyan, 2014) showed that participants preferred using Touch ID for all authentication tasks over regular PIN verification. It is also significantly more secure than regular PIN's, which, given 6 attempts, have been shown breakable 12.39% of the time (Bonneau et al). This suggests that users like the technology, meaning if it is applied to web-based services, it may also be utilised in a similar way to current password managers.

2.2. Literature Review

This project contains several focus areas: the use of biometric authentication and the cloud platform the framework will be built on. This section details some of the previous research into these areas, with a particular focus on challenges implementations have faced and methods of combating these obstacles.

2.2.1. Biometrics

Biometric authentication is defined as the “automated methods of verifying or recognizing the identity of a living person based on a physiological or behavioral characteristic” (Miller 1988, Wayman 2000). These characteristics can vary from the shape of the face, to the way a person walks and can be used by machines to distinguish users based on the data given to it. These systems are also exceptionally accurate; a study carried out by NIST (National Institute of Science and Technology) in 2003 tested 34 commercially available systems and found that these systems had an average accuracy of around 97%, and a false positive rate of 0.01% (Wilson, 2004).

However, the invariance to time, which makes biometrics so useful as an authentication mechanism, is also its greatest weakness. It means that the data (once extracted) is permanently linked to the user and cannot be altered. If stolen, this data cannot be re-generated due to its uniqueness within the specific user - making it useless. This formed the basis of research into “cancellable biometrics” - a phrase coined by (Ratha et al, 2002) who introduced the use of “intentional and repeatable distortions” to the data in order to obscure it. In order to validate the user, the input image would be distorted in the same manner to the stored image and then compared. This meant that even if the data was stolen, the transform could be changed to generate a new template - thus making the original biometric usable again.

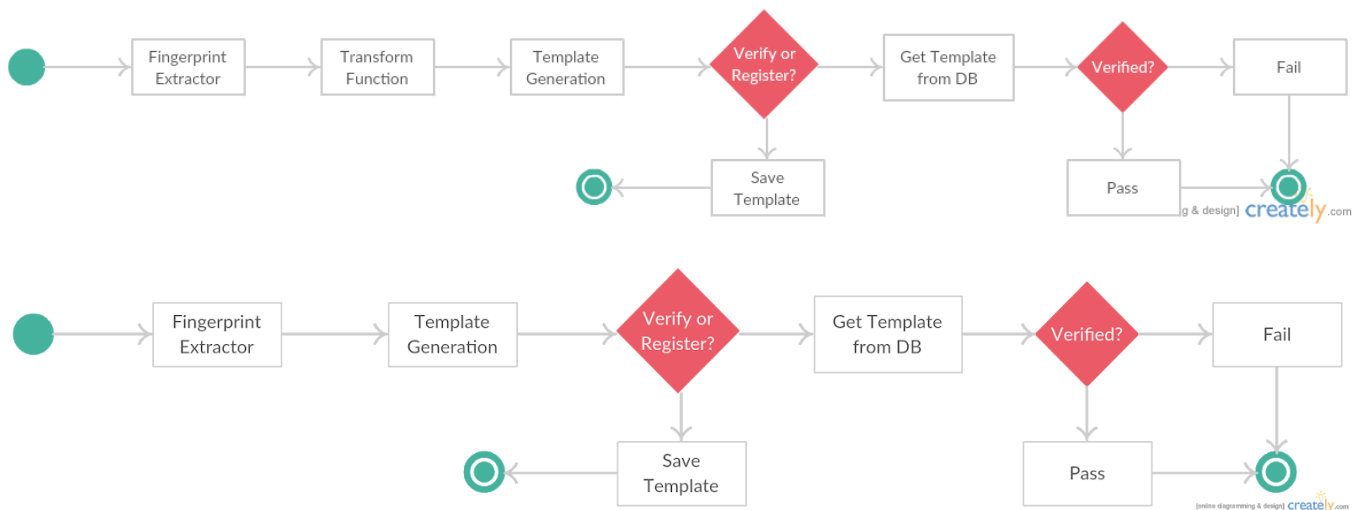


Figure 2.2.1: Comparison between a standard biometric system (bottom) and an anonymous biometric system (top).

A set of four requirements was outlined by (Bolle, R.M, 2002) for systems implementing cancellable biometrics, these are:

1. Diversity - The same template cannot be used in two different systems.
2. Reusability - The system must be able to easily generate & revoke templates
3. Non-invertibility - New templates cannot be used to reverse engineer the original template.
4. Performance - The generation of the template should not impact performance.

Since the initial research into anonymity with biometrics, new methods of developing anonymous templates have been developed. These now sit into two broad categories: Transforming the feature data itself and the use of cryptography within biometric systems.

Transforming the feature data

This category followed similar lines to known methods of securing data, using hashes - such as the ones used for passwords - to create a secure and untraceable template.

One approach applied is biohashing, where a secret key and a fixed length biometric feature vector were combined to form a biohash (Jain, 2007). When a user wishes to verify themselves, they present the original biometric (i.e. their fingerprint) and the secret key, which are then combined to create a hash in the transform space - which can then be compared using distance operators to verify its integrity.

Biohashing itself is also very accurate - with an Equals Error Rate (EER) of 0% (Topcu et al, 2013). Topcu also highlighted the speed that the verification takes place; this is mostly due to the binary nature of the hash, which allows the Hamming distance operator to be used making far faster to compute than if Euclidian distance was used.

This method does have some obvious disadvantages. If a secret key is compromised then the biohash is no longer secure as, due to the invertibility of the transforms used, it would be possible to use the secret key to invert the biohash back to the original template. This has serious implications for privacy as it defeats the purpose of the system. Furthermore, as the verification of a hash takes place in the transformed space, the salting function must be carefully constructed so as to not weaken the verifiability of the data point.

Another transform based approach is using non-invertible functions to permanently alter the template. Assuming the function chosen is non-invertible it would then be computationally infeasible to get back to the original transform. An example of this is work done by Connell (Connell, 2006), who used the initial research from Ratha et al to select several irreversible functions to apply to a set of feature sets- giving an anonymous template for each user in the system. The algorithm showed that there was a noticeable drop in performance - with the false acceptance rate at 0.1 the acceptance rate was only 92.5% - decreasing in the same linear fashion as a general matching algorithm. But this research fails to address many real world challenges of fingerprint verification, such as dealing with imperfect data, as all of the data was specially selected and there is no mention of what the degradation of the initial template could do the matching performance.

Biometric Cryptosystems

The theory underpinning biometric cryptosystems has been around for awhile now, however only fairly recently have they come to the fore in terms of biometric anonymity. In general, these systems work by generating binary data from a template which can then be used to extract a cryptographic key from the inputted data. How this key is generated from the data is split into two categories: key binding and key generation.

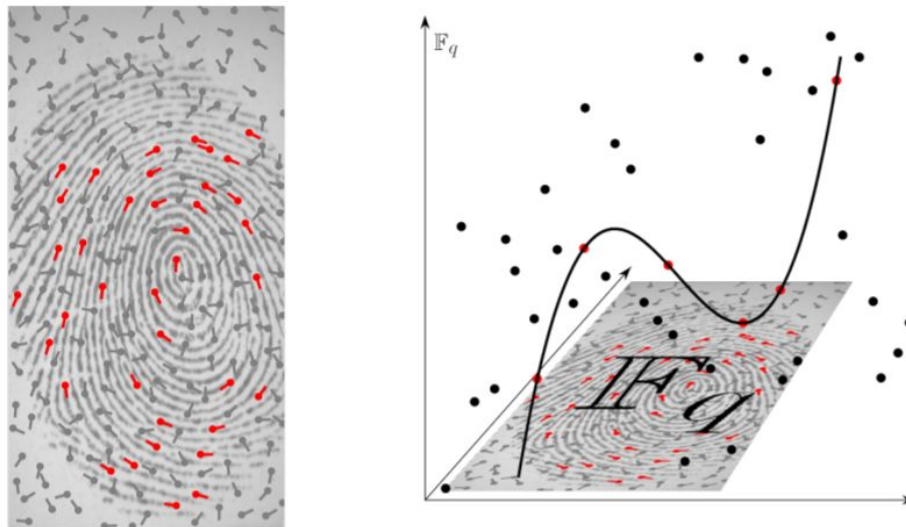


Figure 2.2.2: A visual representation of the polynomial constructed by the fuzzy vault

Reference: Tams, B., 2016

Key Binding focuses on binding a pseudo-random key to the biometric template itself. The most common example of this is fuzzy commitment. Originally proposed by Juels and Sudan (Juels et al, 2006), the fuzzy commitment scheme uses polynomials to encode a secret key within the template, using a large number of chaff points to 'hide' the encoded key. In order to match two templates they must be compared in terms of matching points, and if they substantially overlap with each other the polynomial can be re-created and the secret key revealed. However the real world implementation was questioned by Clancy (Clancy, 2003) who proposed a modified version called the fingerprint vault, which used multiple features as unordered elements of a locking set. This proved far more effective for real world utilisation, however it required that all of the templates were pre-aligned, an issue that is both computationally expensive and complex to implement.

Key Generation focuses on creating keys from the biometric data. Early attempts by Tuyls (Tuyls et al, 2005) used four Gabor filters (Yager et al, 2004) to extract a consistent and reliable set of feature vectors, which was then used to generate a key. Despite this, the False Acceptance Rate ranged between 2.5% (89 bit key) and 3.2% (45 bit key) which is far too high for any viable system.

This section has highlighted the importance of using some method of anonymity when handling biometric data. Furthermore, this section has also reviewed these methods in terms of their fulfillment of the criteria set by (ISO/IEC 24745:2011.). Due to the research presented I have decided to use a fuzzy vault based implementation in order to anonymise the biometric data stored within the platform, as this gives both the robust anonymization and security that I require, which some of the transforms lack.

2.2.2. Cloud Security

The popularity of cloud computing in the last decade has led to significant reviews of the possible security risks involved with developing cloud platforms. In this section I will discuss the different methods of approaching application security within a 'Software as a Service' platform, but also how to implement a secure and flexible infrastructure which can cope with the user demand.

Cloud Application Security

Due to the importance of security for all cloud based applications, there has been a significant push to developing secure application frameworks which can be used to minimise the attack vectors for potential criminals. Work by Chen (Chen et al, 2012) has highlighted some of the vulnerabilities at each layer of a software system.

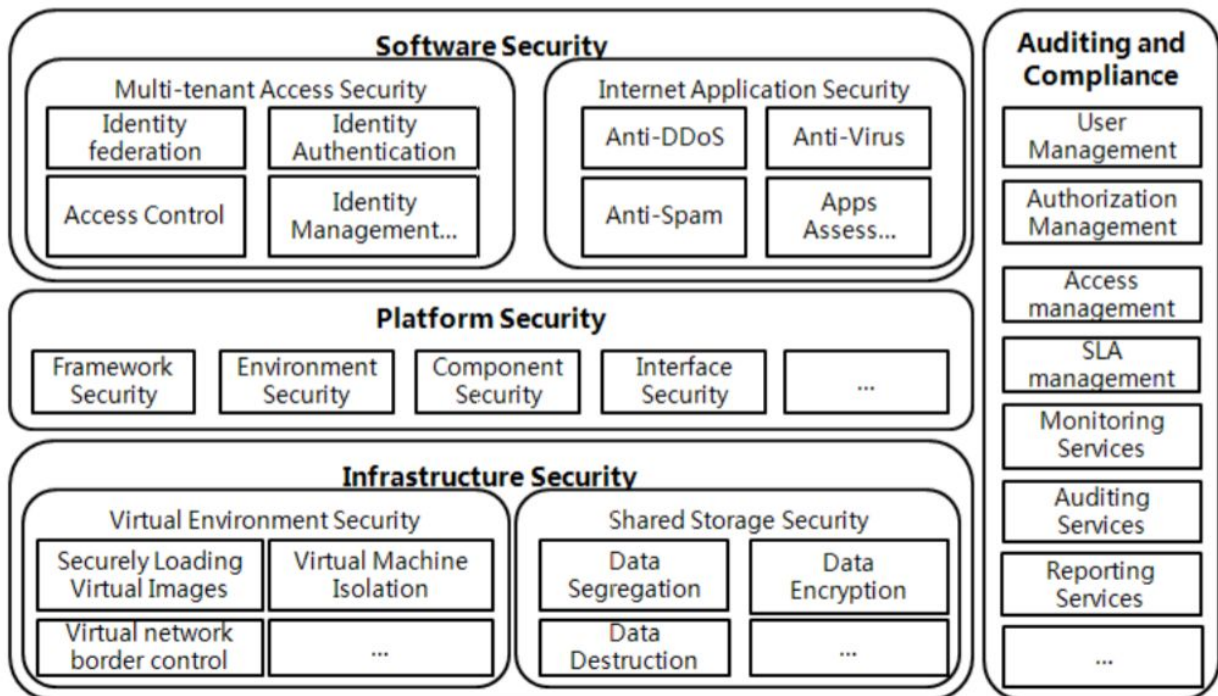


Figure 2.2.3: A selection of possible security measures

Reference: Chen, D. & Zhao, H., 2012

In order to better handle these security measures, four main categories have been defined (Sun et al, 2014). These form the backbone for all actions/services that would be required by a cloud based system. They are:

- Confidentiality
- Availability
- Integrity
- Privacy

Within the context of the 4 factors defined above, I will evaluate the security of the data passing through the system and methods of securing this data.

Data Security

Oracle (Oracle, 2007) developed a scheme to help build requirements for managing data by defining the activeness of the data. This was built on by the Cloud Security Alliance (CSA, 2011), who proposed breaking down the data security into 6 phases. This allows researchers to analyse attacks within specific phases, allowing all of the application to be secure, not just one aspect.

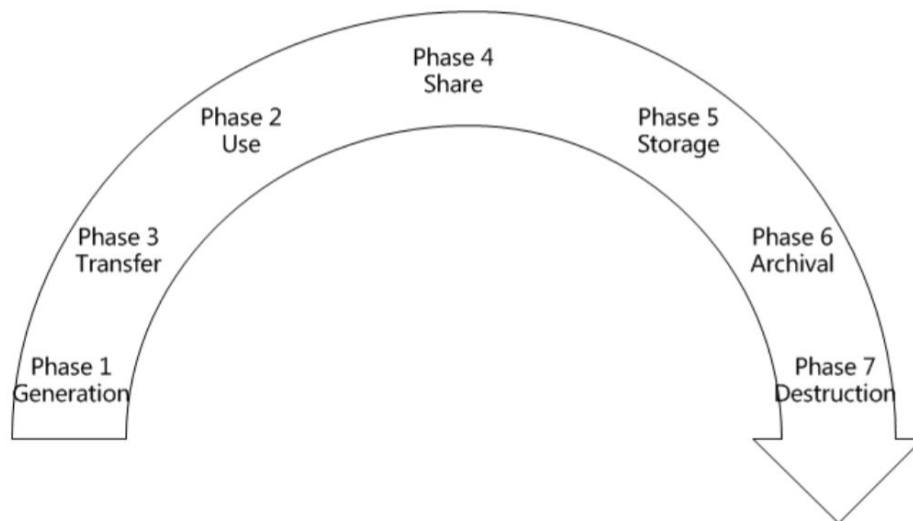


Figure 2.2.4: The data life cycle model

Reference: Chen, D. & Zhao, H., 2012

Phase 1: Generation

Within the cloud scenario there is little a cloud provider can do to secure this stage. It generally relies on the user having a safe and secure environment, free of malware or other potentially dangerous software. Software such as keyloggers would be able to pick up that a password has been entered without the cloud service provider even having touched the data (Subramanyam et al, 2003).

Phase 2: Transfer

This is one of the crucial aspects of security for cloud providers. When the data is being transferred it is in the public domain - anyone tapping networks would be able to listen in on the traffic. To ensure confidentiality, cryptography is used to alter the data. This could range from using standard protocols such as HTTPS/TLS, or implementing their own using pre-existing encryption algorithms. A combination of these may also be preferable, due to some of the weaknesses in TLS identified such as Heartbleed (Durumeric, 2014) and various Man in the Middle Attacks (Haque, 2016).

Other possible implementations look at combining RSA and 3DES. This gives the best of both worlds, as it gives the security of RSA for digital signatures (verifying the client) while 3DES is especially useful for encrypting blocks of data (Kaur, 2012).

To ensure integrity the data can also be hashed and sent along with the content (Meka, 2003) and verified server side.

Phase 3: Use

In many cases much of the data processed is not personal, and can be used in cleartext. However some data (such as passwords) may still want to be encrypted and/or hashed to ensure their confidentiality.

However, recently new methods such as homomorphic encryption have allowed applications to deal with the encrypted data directly (ie not decrypting it) (Gentry, 2009). Although it has often been criticised for being very computationally expensive, homomorphic encryption has been shown to be possible within realistic time (Lauter et al, 2011).

Phase 4: Sharing

As the 'Internet of Things' grows, more personal data is being collected and shared by cloud providers. This increase has made it impossible for users to manually choose the data they want to share, making it exceptionally difficult for the cloud providers to manage the data effectively.

Recently, methods such as OAuth2 have emerged. This allows users to be able use access resources on other machines without revealing their credentials. To do this, OAuth moved away from the vulnerable user-password paradigm and has adopted bearer tokens in its place, which can be used to access specific resources which are verified by the user (Hardt,

2012). However, OAuth2 is susceptible to attacks if not properly implemented; a formal analysis of several OAuth2 implementations showed many sites were vulnerable to Cross Site Request Forgery attacks and can even be used to gain access to a higher ranking site (eg Facebook) (Bansal et al, 2013). Correct implementations with refresh tokens have been shown to be resistant to most threats, only not mitigating a DoS on the resource itself (Sun et al, 2012).

Phase 5: Storing

Data storage is crucial to all cloud services, the data must be available on demand, completely confidential and able to rely on the data being correct without tampering.

In order to implement this the database often uses symmetric key encryption due to its light computational load. This in itself presents challenges, such as how the keys should be managed, as users are unlikely to be experienced enough to handle them securely, so have to be handled autonomously by the system (Bansal et al, 2013).

Phase 6: Archival

Archived data is often stored on older mediums for cost effectiveness (e.g. on tape), making physical security more of a priority.

This section has highlighted the importance of security at all layers of the implementation, and has discussed methods of alleviating the challenges mentioned in order to develop a secure system from the ground up. Within my implementation I will include a 'Data Life Cycle' which analyses possible attack vectors for malicious users and ways to mitigate them, using the methods described above.

2.2.3. Cloud Computing

The NIST definition of cloud computing (Mell et al) defines 5 key characteristics that all cloud computing platforms should inherit, which focus on the need for scalability and reliability within a service. These are On-demand Self Service, Broad network access, Resource Pooling, Rapid Elasticity and Measured Service however some of these are not applicable to this project. In this section I will cover research into Resource Pooling and Rapid Elasticity, the two key areas for this project.

Resource Pooling

All computing resources are grouped to serve customers using customised software. This can be handled in one of two ways, through multi tenant models (Chong et al, 2006), where sets of users are effectively processed together or through multi instances, where users are running on separate instances/machines all of which are managed by the server. This is a breakdown of the different factors:

Factors	Multi-tenant model	Multi-instance model
Cost	Economies of Scale - far more efficient in terms of cost	Higher cost due to the increase in machines/instances
Complexity	Can be very costly to maintain due complex databases and backends	Complex Image management
Isolation	Data is not isolated	Data is stored and processed in an isolated environment
Upgrades	Very difficult to push customised updates to clients	Very easy to push customised images/software to clients
Availability	Only available when the core system is difficult to move.	Can be moved/processed on different machines depending on maintenance
Performance	Greater performance maximisation	Difficult to estimate performance characteristics for machines

References: Signiant. (2016) Gordon, D. (2009), Leinwand, A. (2016), ServiceNow. (2015)

Rapid Elasticity

Rapid Elasticity (Elasticity in cloud computing) refers to the extent to which a system is able to automatically adapt to the changes in workload on a system.

The most common method of implementing elasticity is to create a set of rules using metrics such as CPU time and read/writes to control the number of machines in the working set, adding or removing them depending on the load. While this is effective, and used by large providers such as Amazon and Rackspace, it is also inefficient in terms of resources, as it creates sawtooth like graphs where machines are powered up and down quickly due to a sharp increase and drop in demand (Galante, 2012).

Other methods follow a more predictive approach. AzureWatch uses previous workload history, time and request sizes in order to predict the number of instances needed at any given time. This has also proven to be effective, and is now in use by Microsoft Azure Platform.

This section has discussed characteristics which should be required by any cloud provider when implementing a cloud based system. From this, It has highlighted that my framework must be scalable and resilient. Furthermore, it has shown the importance of using standard protocols to communicate, which points at the use of TLS and HTTPS for secure communications.

2.2.4. Usability of Biometrics

Research into usability is a key aspect of any authentication system, users want a system which is easy to use, consistent and secure. Figures have shown that despite the range of screen locking options, 34% of android users (Triggs, 2016) opt to have no locking mechanism at all - making their device easy to extract data from.

Despite this, things seem to have improved with the advent of better biometrics on mobile. Apple reported that users with biometric enabled iPhones use TouchID to unlock their phones 89% of the time. This is in favour of the less secure PIN, but also other biometric methods such as facial or voice recognition - and overall these devices are now more secure because of it.

So why the sudden increase? Research has shown that users find the TouchID system to be far easier to use than a regular PIN for most smartphone tasks as it takes less time and the users (on average) require less attempts to authenticate using TouchID. Despite this, users reported that they much preferred registering their PIN's as opposed to fingerprints, as it took almost double the time to register a new fingerprint compared to PIN.

This shows that users really care about the speed that they can authenticate at, and any significant slowdown will cause them to try new methods. However, it has also shown that (on average) users are prepared to put up with the register time if it means that they can authenticate quicker because of it.

Therefore, during testing it may be profitable to use speed as a metric for measuring the possible usability of the system, where if you are able to log in quickly, the system is more likely to be usable.

3. Design & Planning

As discussed in the literature review, there are a number of security factors which will affect the implementation of the platform. In the coming sections I will outline the initial design of the system, taking into account all of the security measures discussed in the literature review, then using this to build a set of requirements which can be used to enhance the project planning.

3.1. Initial Design Analysis

Initially, I generated a set of components that the system would need. These were defined at a high level in the **Initial Design Specification** (Appendix B), which included the functions they would need to perform, and how they would interact with the other components in given scenarios. From this, I created a set of requirements from the descriptions of the components - split into functional and nonfunctional requirements that form the basis of the design.

I then performed a **Asset Assessment** (Appendix C) in order to identify the valuable assets within the system and possible risks if this data were leaked. Although the exposure is not needed as this is not a commercial product, it helped in identifying the key points of the system which need to be controlled.

From the Asset Assessment I performed a **Threat and Control Analysis** (Appendix D), which defines a set of possible threats on the assets - methods that a malicious client could use in order to access/change the asset. To decide on the important aspects of this analysis, a **cost benefit system** was used. This looked at the probability of the threat occurring (as defined by the Threat Analysis) and the feasibility of implementing countermeasures; leaving out lower priority threats it was narrowed down to:

- Service spoofing to gain client information
- Snooping on communication to intercept biometrics
- Framework spoofing to get biometric information
- Device spoofing to imitate the verification procedure
- Biometrics security & anonymity
- Vulnerabilities in the infrastructure

In order to secure these areas, this design section will focus on several key areas:

- A specification for how services will be authenticated, and how requests generated by services are verified for their integrity
- A specification for how device verification will be implemented and secured
- An implementation to effectively anonymise & protect the biometric data stored
- The framework infrastructure for a secure platform

3.1.1. Initial Design Decisions

In order to develop effective security measures, I also completed a Data Life Cycle (see literature review, completed in Appendix F) for the most important assets defined in the asset analysis. This allowed me to explore securing the data at all aspects of its life, not just at one particular point. This has then been used to inform the rest of the design, and highlighted some key areas which should be decided on.

Overview

The framework will act as a third party system for authentication, much the same as the login via Google buttons seen on most sites. This means that the biometric verification requests must originate from a valid service. A user must register for the framework, which enables them to be able to utilise the login via BioGate functionality.

Device Versions

Due to how Android is designed, it is impossible to pull any biometric data from the device. Therefore, it was decided that two versions of the system should be implemented:

- Android (Mobile) Implementation, which uses local authentication within the OS.
- Desktop Implementation, which utilises the entire BioGate Framework.

It was realised early on that, although they verify in different locations, the system implementation would be very similar for both methods of authentication - meaning that the design paths could be combined, allowing for one single (main) design.

Communication methods

Initially, it was envisaged that all data transfer would be done using HTTPS POST. However, due to the complex nature of some of the transactions, a simple web server was not able to cope, as the HTTPS requests were blocking the main thread. Enabling multithreading would allow some leeway, but it was not a good fix for the problem as it meant that the system was limited to a specific number of users it could serve. This goes against all of the cloud principles of flexibility and scalability discussed, so a new design had to be created.

This design used the sockets in order to implement scalable and flexible requests - allowing the framework to continue processing other clients whilst worker threads process the biometric verification.

Notifications

In order to be able to notify the client that a login request has been made, the framework will utilise pre-existing push frameworks, such as Firebase for Android. However, after some research it was found that the Java client would not be able to utilise any of the operating system specific notifications, meaning that I must implement my own.

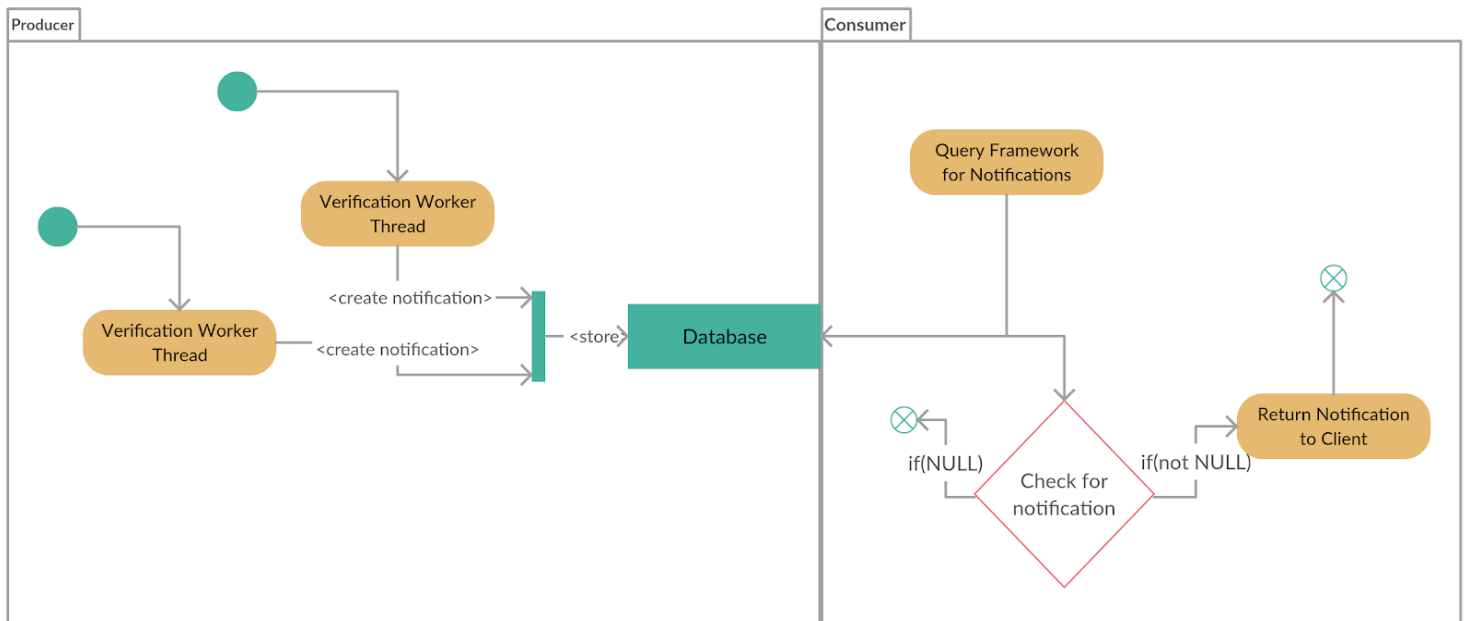


Figure 3.1.1.1: A State diagram to highlight the producer/consumer notification model

To do this, I used the Producer/Consumer model, where the framework produces notifications and places them into the database, and the client intermittently queries the database to see if any new products are ready to be consumed.

API's

Due to the use of sockets, the external API will be split into the web verification API (WVAPI) and the web accounts API (WAAPI). The WVAPI will utilise sockets to communicate with its clients, and will handle all biometric verification requests.

The WAAPI will utilise REST processing to handle all HTTPS requests made to it, which will handle all account registration and logins for both users and services.

This separation will allow the WVAPI to operate faster (a key consideration highlighted in the literature review), as it can be threaded to be as fast as possible.

Extension Systems

In order to show the framework in use, I will also have to develop a mock service, which implements the service specification discussed below.

Research has highlighted the possible use of indirect biometric authentication. This would work similarly to the password managers discussed in the background research, where the framework itself will use biometric authentication to verify a user, and then release a one time password which can then be used by a user to login. This would allow the system to have a instant impact, as it does not require services to implement the BioGate functionality. Designs of this will be discussed below.

3.1.2. Designing for Security

Security Requirements (SR)

The security requirements are used within the task planner (see project management), and are listed below:

ID	Security Requirement
SR1	Any communication to the system established by an external device must be verified and matched to a client using the Service/Device specifications.
SR2	All communication between system and clients must be sent over HTTPS.
SR3	If Biometric data is stored on the device it must be stored in trusted memory, and only be accessed by the TEE.
SR4	All Biometrics stored in the framework must be anonymised
SR5	Information stored in the database will be appropriately encrypted/hashed.
SR6	The Framework must be secured appropriately to prevent remote attacks.
SR7	Any One time passwords generated must conform to the RFC 6238 (TOTP)

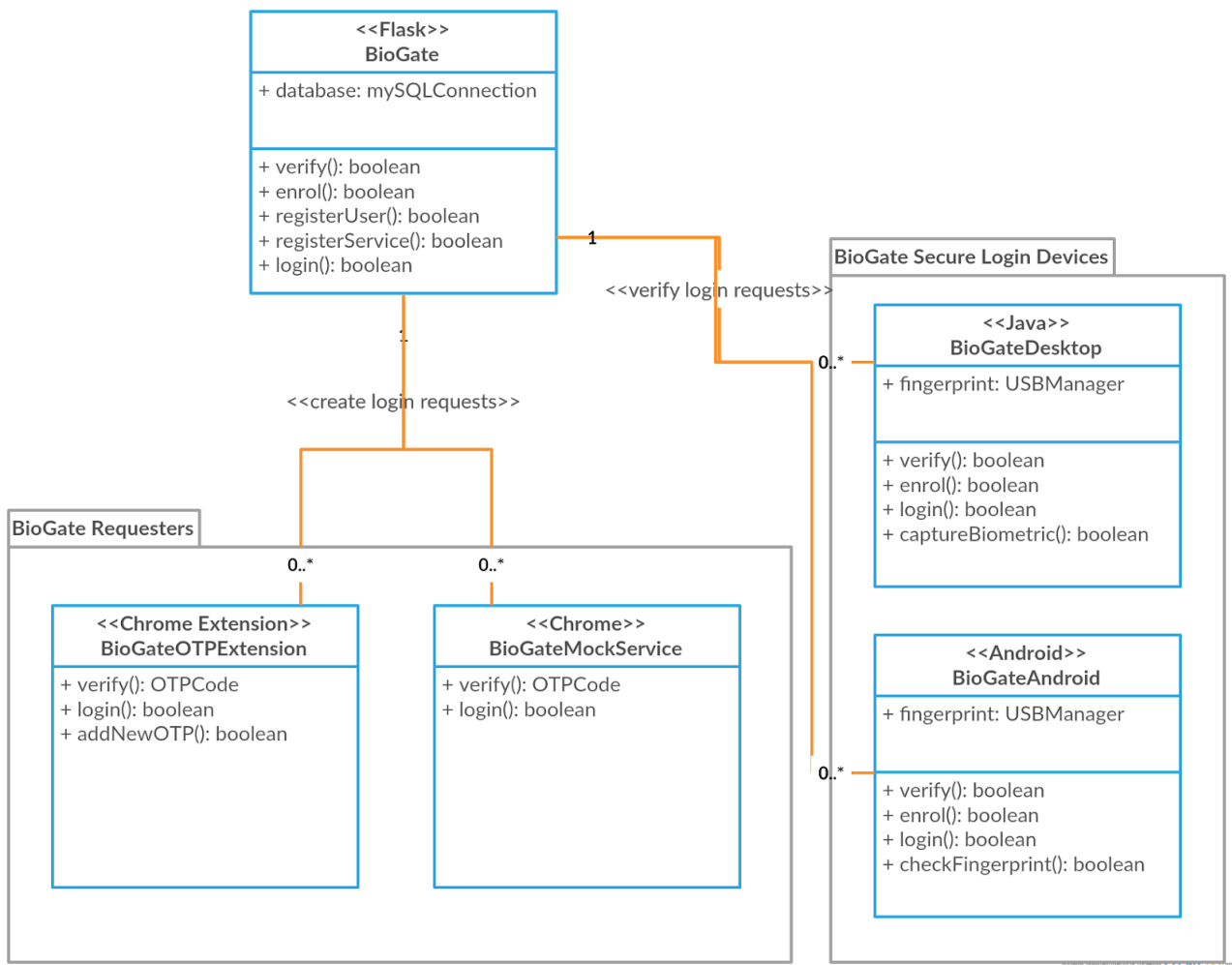


Figure 3.1.2.1: Abstracted class diagram showing each project within the overall codebase

3.2. Planning & Project Management

Work Breakdown

As the system has a modular design, with different possible components I decided to take elements from the Agile method of software development to increase my productivity (Ahmed et al, 2010).

Due to the nature of the design, I decided to use **sprints** to organise development. Taking the initial requirements generated from the design section, I developed a set of tasks to be implemented by either breaking down the **requirement** or keeping it a single task. This then allowed me to take specific tasks off the **product backlog** for each sprint, allowing the project to be developed in a logical order. When assigned to a sprint, each task was then given as **estimate** time value - which was then used for task tracking, allowing me to see if I was falling behind in any specific week. This allowed me to push tasks back to later sprints if I had issues with completing some tasks.

I also left myself a buffer period of around 4 weeks before easter if I needed to complete further work, meaning that when I went over my original estimates I was able to use this time to finish off specific areas which needed work.

This diagram shows my product backlog and task planner, which was updated each week as to the hours spent on each task. The colours indicate if the task went over on the time allotted for it, with red being over, orange being near and green being under.

Key		
	Under estimate	
	Within an hour of estimate	
	Over estimate by an hour or more	
	Incomplete	
S1	Refers to Task ID in Task Backlog	
5	Time to complete task in hrs	
Totals		
Sprint 1	Estimated	80
	Total	87
Sprint 2	Estimated	72
	Total	77
Sprint 3	Estimated	85
	Total	67
Sprint 4	Estimated	84
	Total	62

Figure 3.2.2: Table showing the time spend on each sprint and key for the task planner

Comparison to Initial plan & Risk Mitigation

As the initial gantt chart shows, I planned to finish my implementation at the end of February. Due to some issues with getting the fingerprint scanner the development end was pushed back to the end of march (See Risk Analysis - Appendix G). However, due to the ease of planning with the gantt chart I decided to begin report writing early (as you can see from the report task in the planner), meaning I was able to stay on track with the project - interleaving my report writing with my development. To account for this, another sprint was also added.

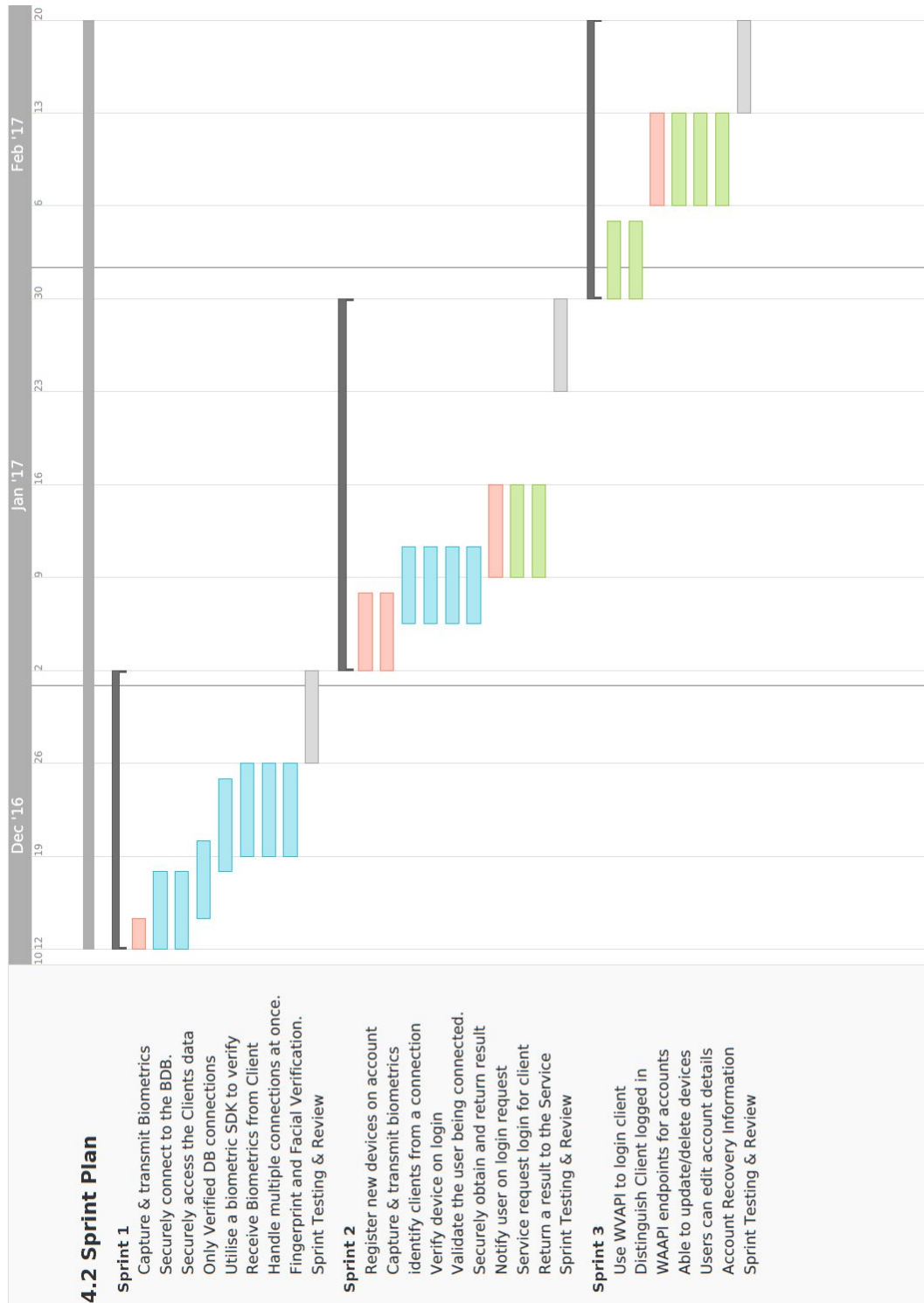


Figure 3.2.3: The initial sprint plan from the progress report

To plan for any potential threats to the completion of the project, I completed a **Risk Assessment** (Appendix G), outlining possible risks that could interrupt development, their severity (in terms of disruption caused * probability of occurring) and possible mitigation strategies. Consequently I was prepared for problems during development and had a clear action plan to resolve them.

Despite the planning, some problems did still occur - the most crucial of these being the delay in getting the fingerprint scanner. However, due to the task planning, I was able to begin report writing early (as demonstrated in the report task in the planner); as a result I was able to stay on track with the project.

The only other issue faced was some missed meetings with Oli, due to one/or both of us being busy/ill. In order to mitigate this, I wrote down any questions/points I wanted to raise in our meetings - allowing me to make sure that I didn't forget between meetings. I also recorded notes of our meetings to make sure I had something to refer back to. This enabled me to stay on top of my work and questions even when meetings were missed.

Tools

To speed up development, I used several tools to aid me. These are:

- **GitHub** for handling my codebase, allowing me to effectively handle all of the projects.
- **Pycharm & IntelliJ** for developing code, making it easier to find syntax errors in my code.
- **Creatley** for developing all UML diagrams.
- **Excel** for building and maintaining the task planner, which proved invaluable with managing the project.
- **Team Gantt** for building the initial gantt charts, however I switched over to excel after the progress report as it allowed me to have greater fine grained control of the my tasks.

Progress

In terms of work completed, I have completed almost all requirements completed during development of the project. The only main aspect omitted was the lack of support for other biometric methods (such as facial, iris). However, this would be fairly easy to implement due to the way the system was designed.

4. Design Specifications and Implementation

This section describes the low level detail for each component in the system, and then discusses how this component was implemented into the system, using UML to describe specific parts of a component.

4.1 General Communication

Requirements Covered: SR2

Design Specification:

All communications must be sent over HTTPS, at no point will HTTP/plaintext be used.

All communication will be made in the JSON form.

Design decisions and implementation:

A SSL certificate was generated from LetsEncrypt, which will be used to establish a HTTPS connection with all devices.

For Java, the simplejson library will be used to parse json, and the framework will use the inbuilt json parser in python.

4.2 Device Design

Requirements Covered: SR1, SR3

Design Specification:

Due to the security needed on the devices, I decided to require users to sign in on the devices before they can use them for biometric verification. When a user registers, they are creating an account - but cannot utilise the service until they have logged in on a biometric enabled device (android app or java client). On login, the device type is assessed and sent to the server as a SLD registration, which registers the device to the user's account and returns the client's id and secret. This secret can then be used to verify the devices authenticity when requested.

An overview of this specification is:

- All secrets must be stored securely.
- Secrets must be used when verifying the device for sensitive methods.
- The device must be able to easily support multiple fingerprint readers (if applicable).

Design decisions and implementation:

Registering SLD

As specified above, a user must sign into their account on a device before being able to utilise the framework. This is done by:

- The User entering their username and password.
- The Key type and value is for the specific device is tagged and sent along with the username and password to the server. To decide the key type:

Client Type	Key Type	Key Value
Android (Mobile)	firebase	Firebase Key assigned to android device.
Java	custom	

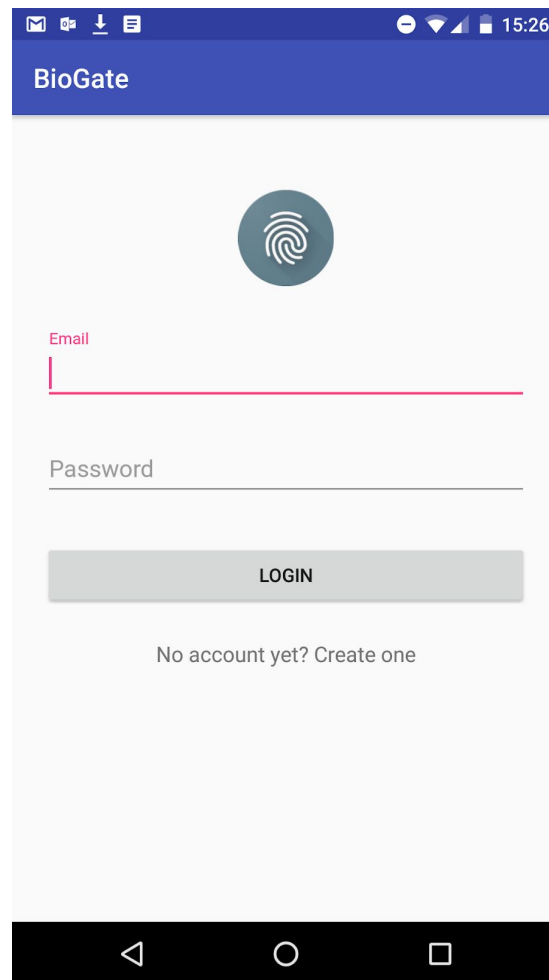


Figure 4.2.1: The Android login page

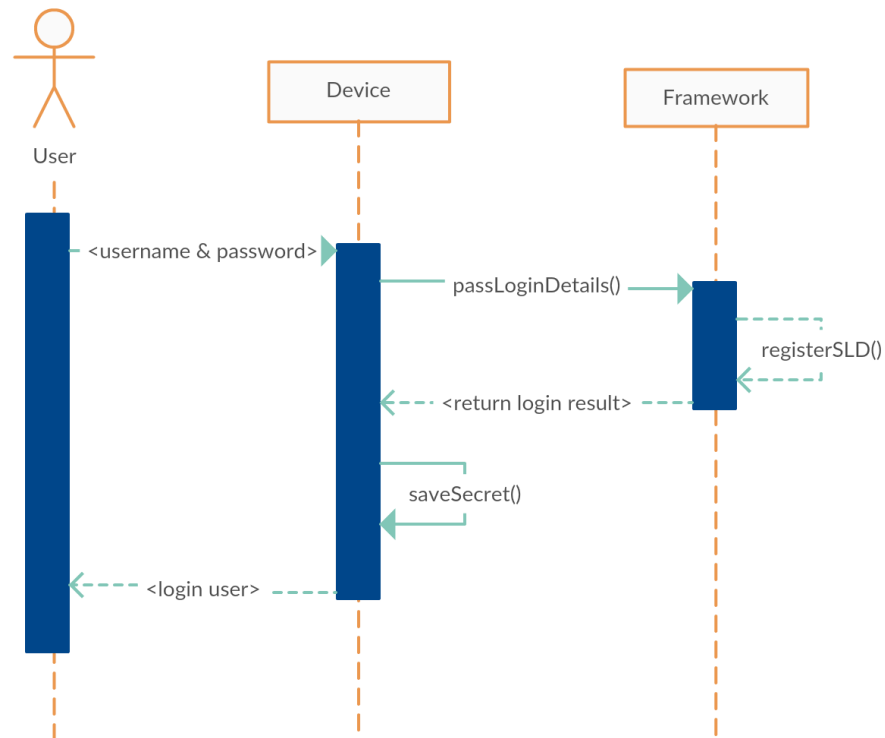


Figure 4.2.2: A Sequence diagram showing the login interactions.

This request will return a generated client_id and client_secret to be used to authenticate the device. Also, session tokens are generated, which are tied to the client device, meaning that a session token can only be used in conjunction with a client secret - preventing attackers stealing session tokens.

Android Verification

Local verification is possible if the mobile device is running android API v23 or higher, and has a biometric fingerprint scanner. Using this I developed an Android application which uses the FingerprintManager within Android. When the fingerprint sensor is triggered, the FingerprintManager will compare the fingerprint to the fingerprints enrolled on the device, and will return a result. This result is then sent to the Framework by the device.

Java Client Verification

The Java Client must be able to capture a fingerprint and transmit it to the Framework, which will then allow the Framework to verify the biometric and return a result.

In order to capture the fingerprint, a free fingerprint SDK was utilised in order to extract the data from the fingerprint reader. In order for the application to be extensible, the extraction is a simple interface, hiding the working of the SDK itself. This allows me to also write interfaces for other SDK's, making the application cover a wide range of devices.

Once the fingerprint is captured, it is sent to the Framework, which will return a response once verified.

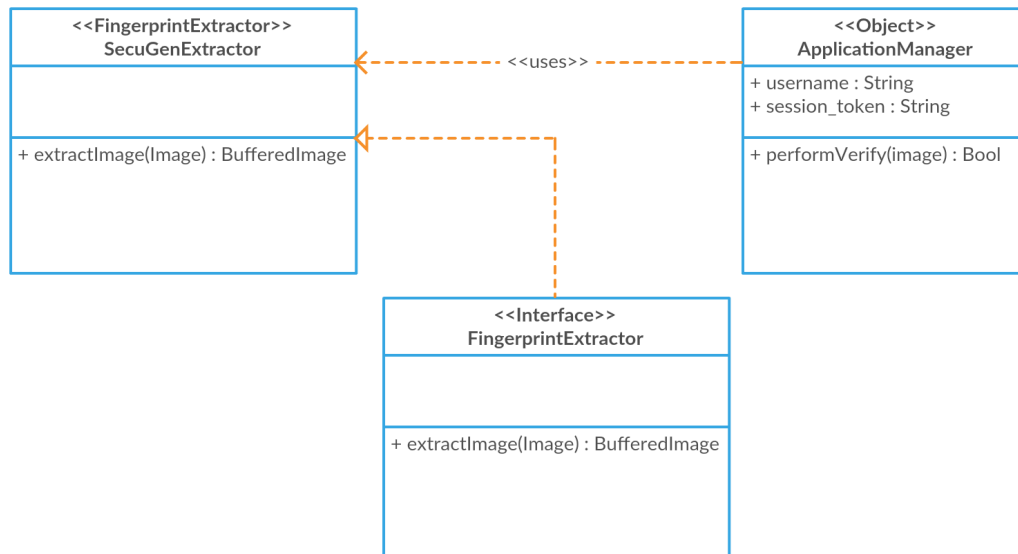


Figure 4.2.3: A Class diagram showing the extractor interfaces

4.3 Framework Design

Requirements covered: SR4, SR5, SR6

Design Specification:

The framework must be hosted on a secure server, using tried and tested software (preferably open source) to reduce the chance of vulnerabilities being exploited.

All data stored which is of value (defined in the asset analysis) must be hashed using SHA_256 for added security. This prevents attackers using the data if it is stolen.

All Biometric data must be securely anonymised and stored in the database in the anonymised form, no generic biometric data should be stored under any circumstances.

The SLD must be securely notified when a biometric request has been made, passing a session_id which can be used to validate the request.

Design decisions and implementation:

The server will be hosted by VPS.net, a reputable server hosting company, so the server should be protected from physical downtime. Furthermore, I will implement the framework in Python - as Python has very good image processing libraries (PIL, Pillow), which will allow me to easily handle the biometrics that pass through the system.

I will also be using Flask for the web service - which are more than capable of handling a high throughput. It is also well secured, with active development communities.

Database

I will be using MySQL as the database using mysql-python to connect to the framework code, which has been shown to be scalable and secure. Within this database there will be several different tables, shown in figure 4.3.1.

All passwords and secrets are hashed using the python library passlib, which uses the SHA_256 algorithm to irreversibly hash the data.

The database has its own handler, which is the only component which is able to access the database. All database calls must utilise methods within this object.

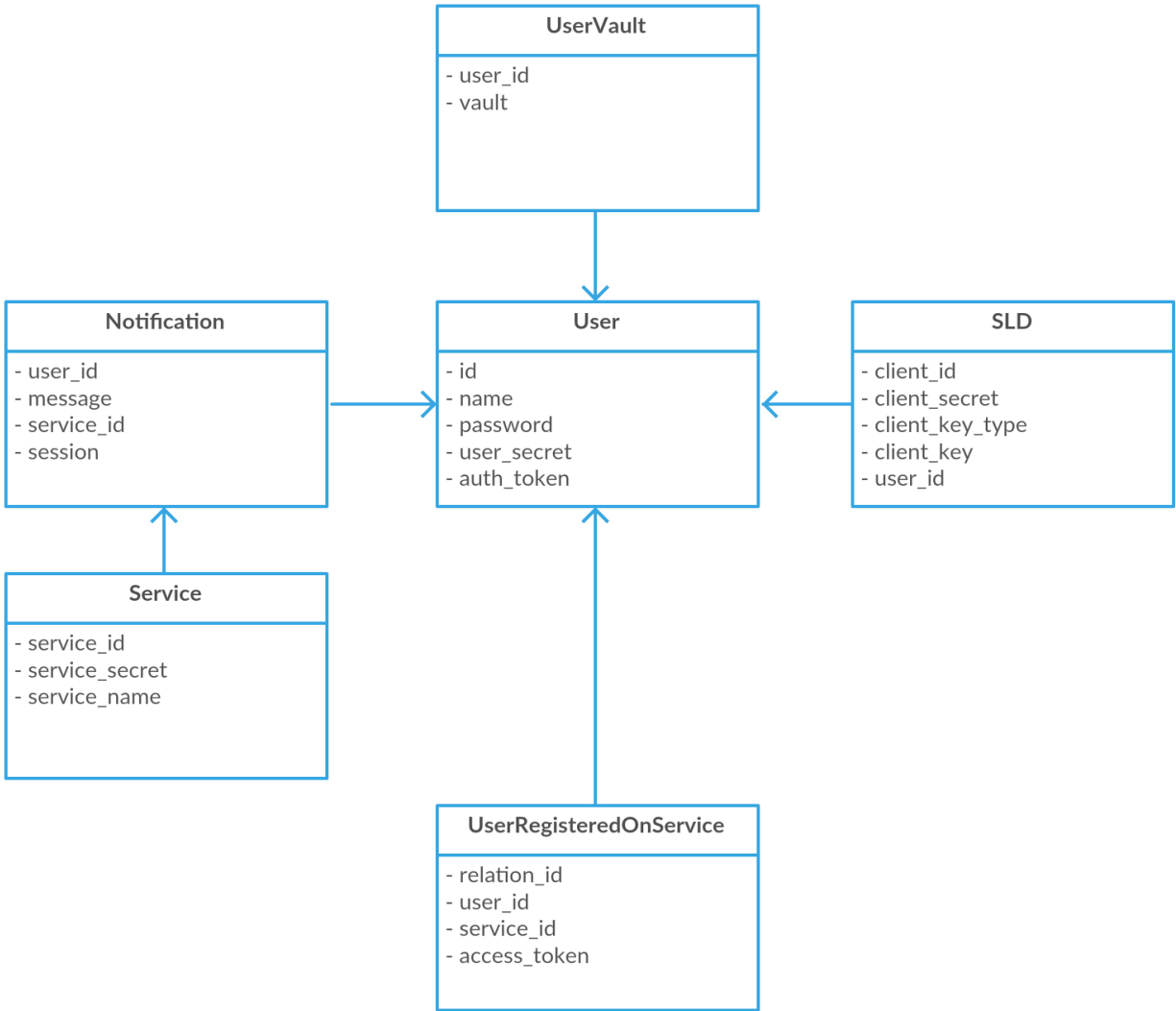


Figure 4.3.1: All Database tables for the standard biometric system

Framework Structure

A multi-instance model was chosen to handle requests. This allows multiple users to be served in tandem, giving the platform the flexibility it needs to authenticate multiple users as fast as possible. This happens in two different ways within the framework, these are:

The WAAPI:

- Flask will spin up a new thread for each client who has made a request to the framework, as these are all stateless REST queries, they will terminate quickly allowing clients in the waiting pool to be serviced quickly.

The WVAPI:

- The API will take any HTTPS requests incoming and spawn a worker thread to handle the client's request. This worker thread can independently communicate with the clients (not via the main API), allowing the API to be non-blocking.

Only the main API's (ie not the worker threads) can access the main BioGate, allowing these two input methods to be secured to prevent malicious requests. From this controller the Verification Module and the Database can be accessed (discussed below). This was edited slightly for the final design (combining the two API's), however the processing remained the same as initially envisaged.

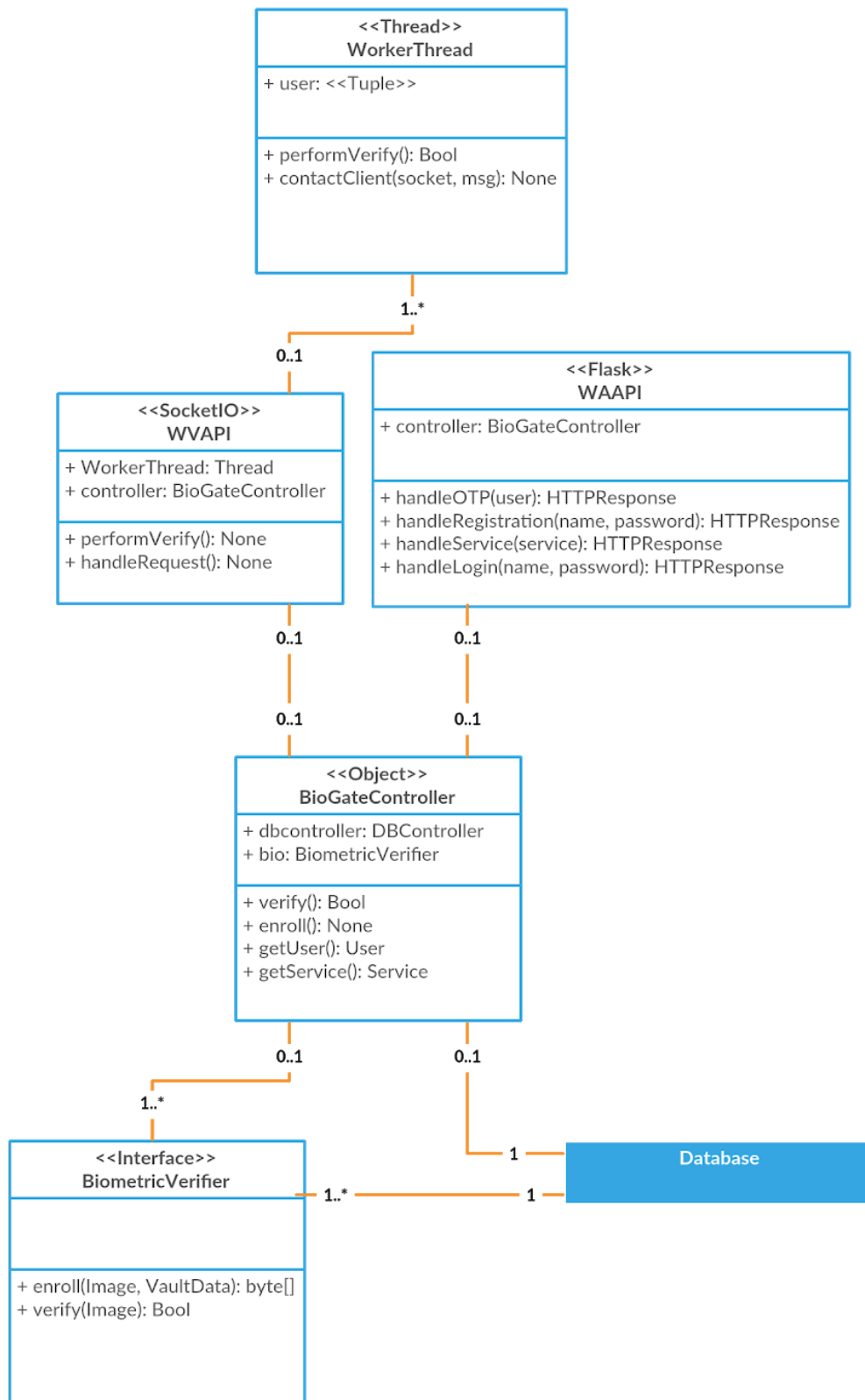


Figure 4.3.2: A high level class diagram of the frameworks structure

Biometric Verification

To anonymise the data, I will be using a biometrics library implemented by Benjamin Tams (Tams, 2011). This can be used to verify and enrol fingerprints into the system, returning either the anonymised data or a result. As with the java client, this is designed to be easily extensible by using Interfaces, which would allow other verifiers to be implementing allowing the system to be used for other biometrics, such as iris.

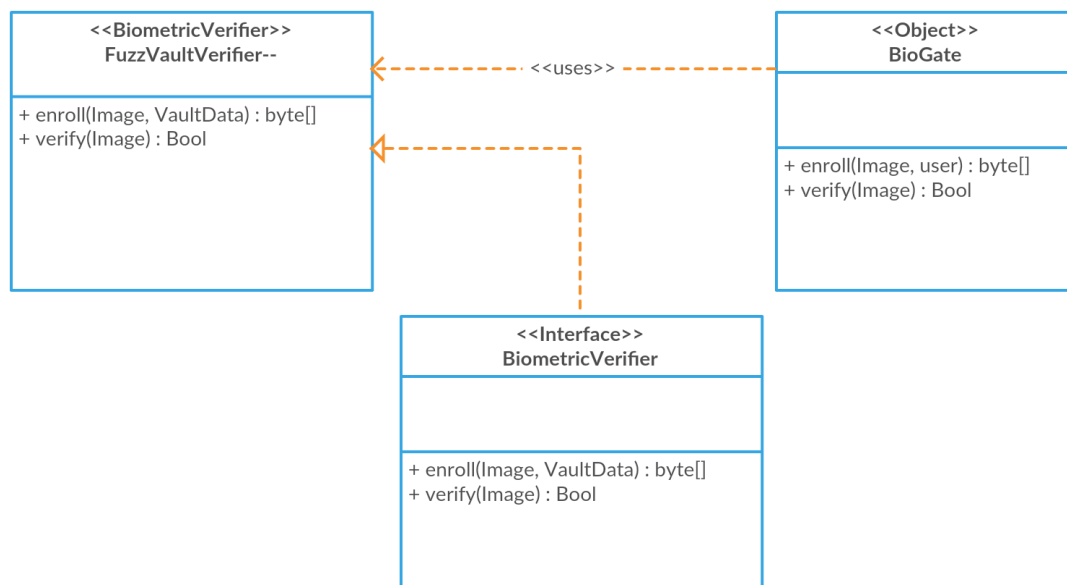


Figure 4.3.3: A Class diagram showing verifier interface

Verification Procedure

Once a Verification request enters the WVAPI, there is a series of steps that must be performed to check the integrity of the request, but also contact the necessary devices.

Initially, the request is checked to make sure that both the `service_id` and the `user_id` are valid, and that the user is registered to that specific service. The access token for the specific relation is also checked, making sure that the service has correctly gained access to the system using the authentication token. It also checks the type of the request, either being a request to log into a service or a request for an OTP release.

If the request passes this validation, the client is notified a login request has been made. A `session_id` is also tagged with this notification, so the response can be validated.

Once the client opens the notification, then a socket connection between the server and client is established, exchanging the validation data (`client_secret`, `client_id` and `session_id`). This verifies that the client connected is indeed the users registered SLD. After the server responds, the client sends the biometric data/result to the server.

Finally, if the client is communicating via the Java App then the biometric data is taken from the request and passed to the verification module via a python wrapper which calls the corresponding C code in the Thimble Library. This then either enrolls/verifies the client, returning a result depending on the success of the process.

This result is then returned to the web service which sent the request.

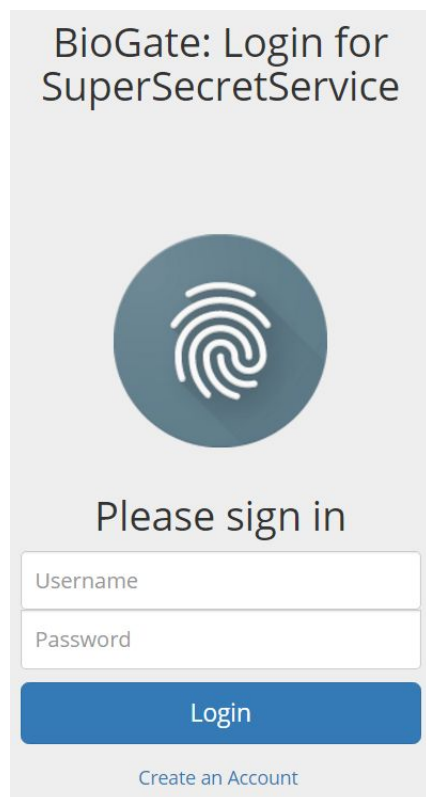


Figure 4.3.4: The OAuth login page displayed to clients, with the name of the service (SuperSecretService) displayed.

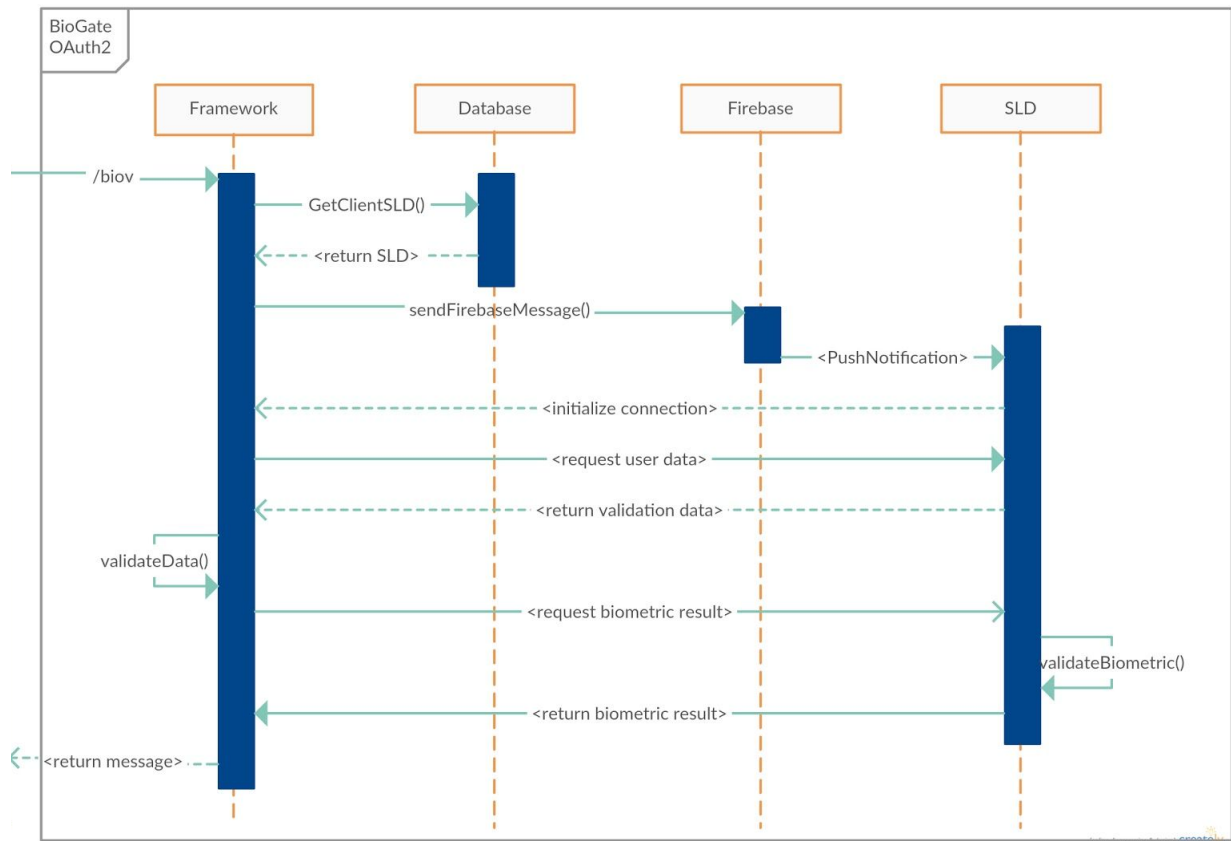


Figure 4.3.5: Sequence diagram showing the communication between client and framework for a biometric verification request

4.4 Extension Systems Design

Requirements covered: SR7

Design Specification:

In order to utilise the OTP (One time password) extension, a user must first enter an auth token into the system. These secrets are compatible with both Authy and Google Authenticator, and can be retrieved by those applications. This secret must also be given a unique name to identify it for that user.

When a user then requests the OTPs, the framework must first verify the user using the biometric verification module. As no service exists (OTP is separate from services), the default service OTPService will be used, and the request type must be otp.

If the biometric verification is successful the system will retrieve the OTP by pulling the correct key from the database and generating the password. This will then be returned to the user, allowing them to use the code.

Design decisions and implementation:

In order to make this as usable as possible, I decided to implement it as a chrome extension. This allows users to stay on the same web page, just opening a popup.

As the OTP extension uses both the WVAPI and the WAAPAPI, it must implement HTTPS to communicate with the framework, this is done through a set of javascript objects handling the request management, with JQuery and bootstrap handling the front end displaying to the user.

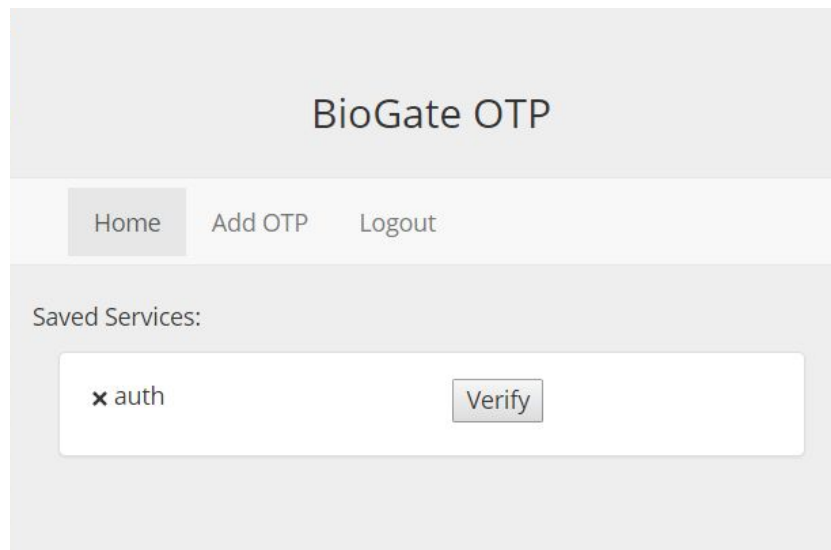


Figure 4.4: The OTP Extension with a single OTP saved

4.5 Service Design

Requirements Covered: SR1

Design Specification:

In order to prevent service spoofing, each service will be issued with an id and secret. These will be used to identify the service to the framework to ensure any request is valid.

If a user wishes to verify for a particular service through the framework, the user must be directed to the oauth login page provided. The service must not come into contact with the user's credentials at any point.

The service must then be able to use the authentication token returned by the successful login to verify a user for that particular service. This must be done securely.

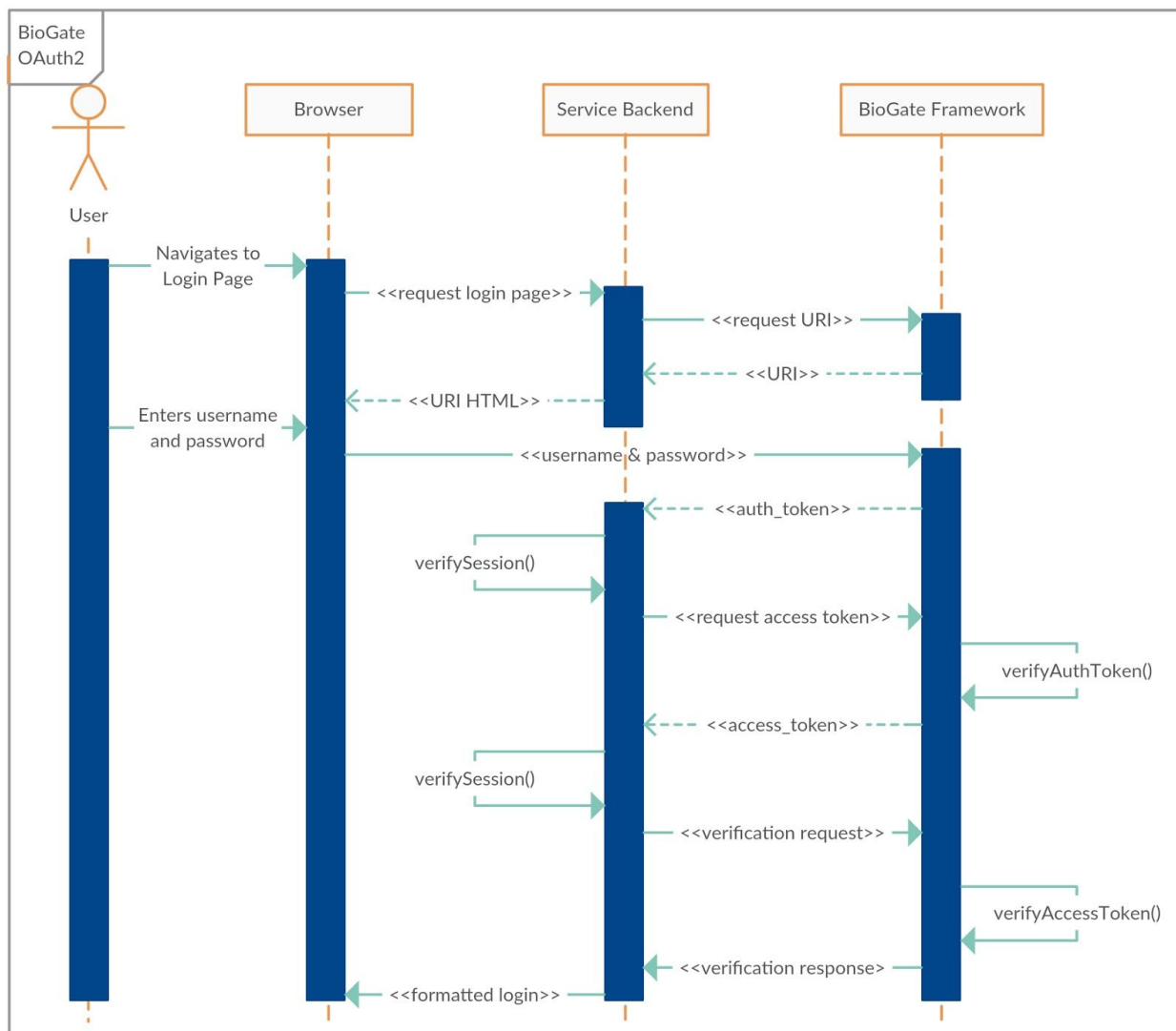
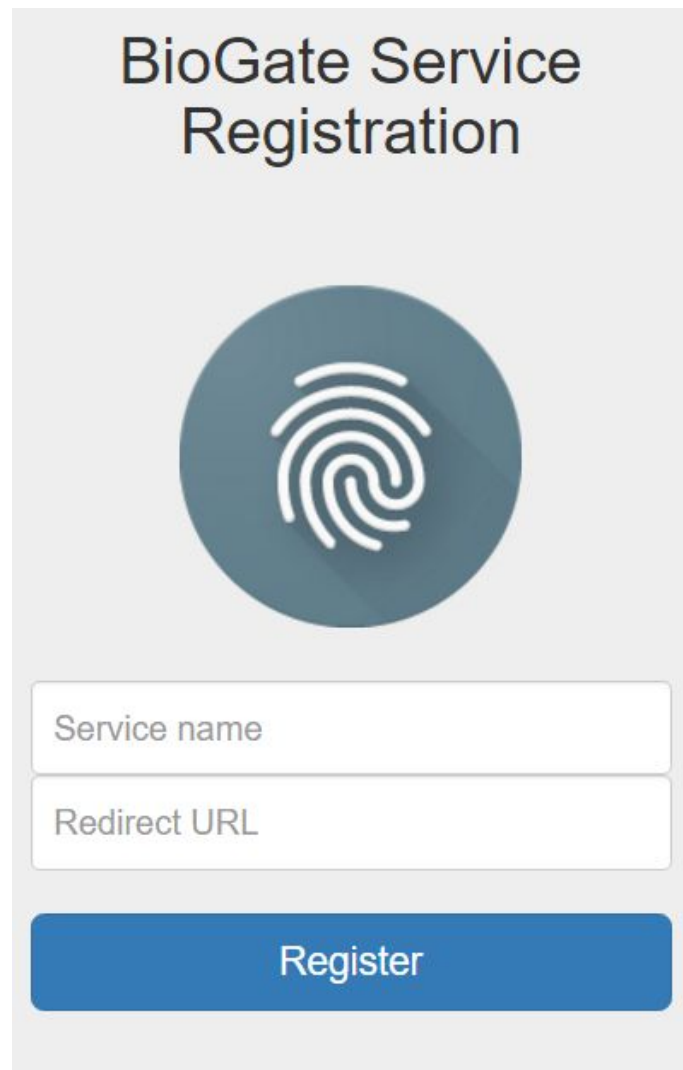


Figure 4.5.1: Sequence Diagram to show the interaction between browser & framework

Design decisions and implementation:

Registering a Service

In order to function, a service must first register to the framework through the web application form. This simply takes the service name, the redirect URL for authentication tokens and a contact email for the service admin. Once this is completed, the service will be returned a `service_id` and `service_secret`, which is used to verify its identity.



The image shows a web form titled "BioGate Service Registration". At the top, the title is in a large, dark font. Below the title is a circular icon with a white fingerprint pattern on a dark blue background. Underneath the icon are two input fields: "Service name" and "Redirect URL". At the bottom of the form is a blue button with the text "Register".

Figure 4.5.2: Service registration form

Securely Requesting Verification

In order to request a verification, the service must implement the OAuth2 specification, forcing all request to gain and release a set of tokens to verify the integrity of the request.

Here is an overview of the designed and implemented system:

- **Resource Server:** The Framework WVAPI which allows access to the verification module.
- **Client:** This is the Service which wants to utilise the resources on the Resource Server.
- **Authorization Server:** The Framework WAAPI, which handles all token generation and validation.
- **Resource Owner:** The User whose account is being accessed.

To authorize a request, a user must enter their username and password - this was deemed to be most secure compared to the other possible methods (such as an authorization code), as it would not require any data to be stored in the front end javascript - which is inherently insecure.

The authorization token is then returned to the redirect URL, which is specified on service registration.

The service then must exchange this authentication token for an access token, in order to do this, the WAAPI is queried, passing the service_id and the service_secret. This is verified, returning the access token. As timeout access tokens are being used, the token can only be used within 2 minutes of its generation else it will expire and a new one must be requested. This access token can then be used to access the WVAPI, completing the biometric verification request.

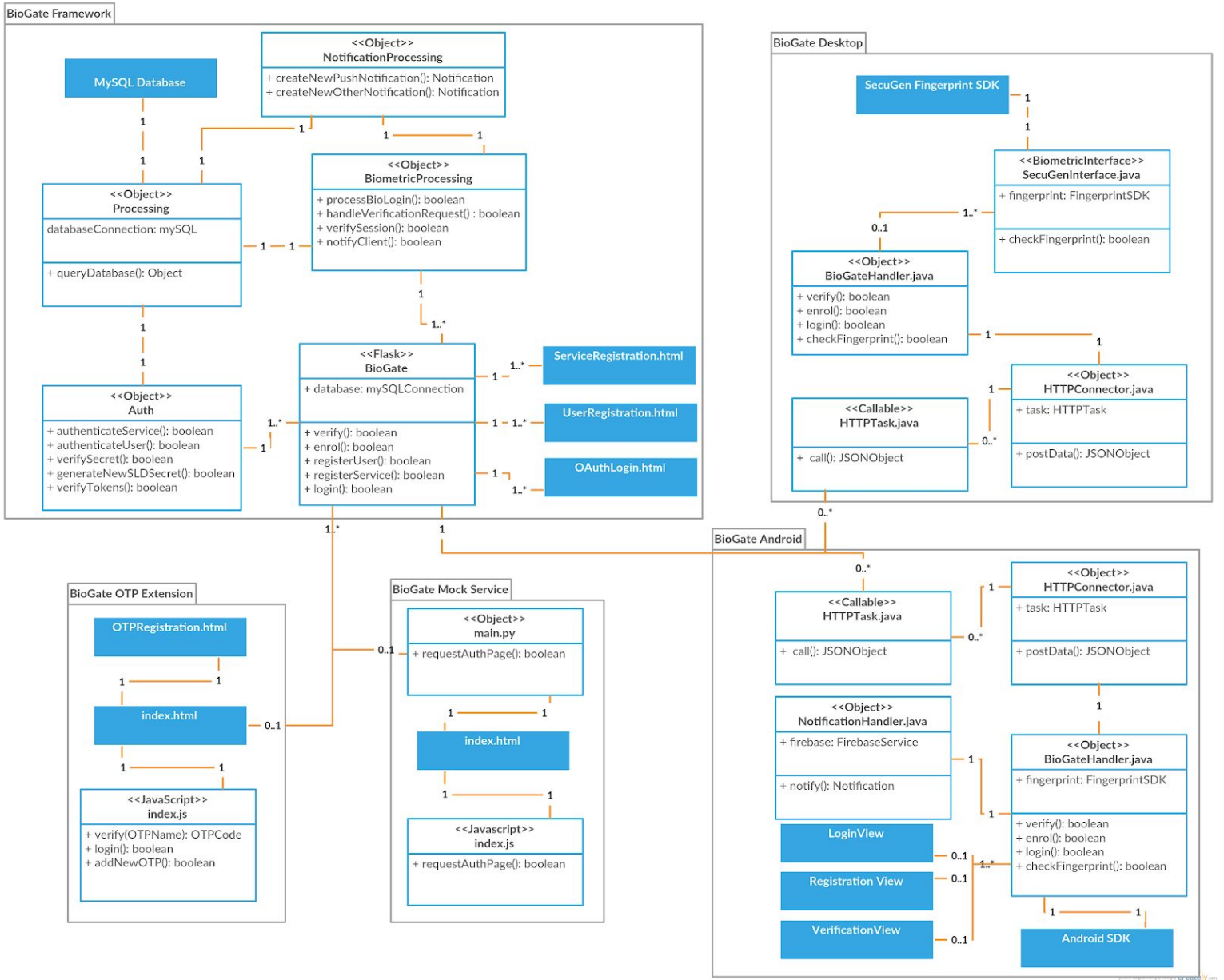


Figure 4.5.3: Abstracted version of the final system implementation

5. Testing

5.1 Regression Testing

In order to validate the results produced, I decided to take a two pronged attack to developing tests, the first one was using integrated unit tests in python to test each method individually, making sure it performed as it should for each possible case. This was done in parallel to development (as can be seen in the project management section), and these tests were run every time the project was built and deployed. This method was used to verify all aspects of the project, ensuring it worked correctly at all stages.

The second aspect was using a python library called pyresttest, which is used to define a set of unit tests to perform on REST services. This can then be used to test the endpoints that the client devices would connect to - passing in different sets of data and checking the response of the server. This is an extract from one the tests, which tests basic registration:

```
- test:
  - name: "Basic Registration"
  - group: "Login Tests"
  - generator_binds: {name: gen_name}
  - url: "/register"
  - method: "POST"
  - body: {template: {"name": "$name", "pw": "$name"}}
  - headers: {'Content-Type': 'application/json'}
  - validators:
    - compare: {jsonpath_mini: 'res', expected: true}
    - extract_test: {jsonpath_mini: 'uname', test: 'exists'}
  - extract_binds:
    - 'user_name': {jsonpath_mini: 'uname'}
```

Figure 5.1.1: An example test for testing the registration endpoint in the WAAPI

This test uses the generators within the testing framework to generate a username and POSTs it to the endpoint to test, and then the validators check the result returned.

These tests were developed from the specifications defined in each design section for the corresponding component, as discussed in the design section.

Through utilising these testing suites, I was able to test all of the functionality of my application throughout development ensuring the code that was created was up to standards.

5.2 User Testing

As discussed in the literature review, there is a significant correlation between the usability of a system and its use in the general public. Therefore, in order to ascertain the usability of my system I decided to implement user testing, to get feedback on the speed and ease of use.

After analysing previous research, I decided to focus on the speed of the system as a whole, from registering to verification. This seemed a crucial aspect, with faster systems being used more frequently than slower ones. To test this I asked the participants to complete a set of tasks, measuring how long it took to complete each one. These tasks were:

- Creating an account for the BioGate Framework.
- Registering their account with a mock service set up for the test.
- Logging into the service after registering.
- Log into their university account (www.sussed.com) via password authentication.

In order to maintain the same conditions, all tests were conducted:

- On the same PC/Mobile to prevent loading times affecting the results.
- Once the web page had loaded, to reduce skewed results from internet speed drops.
- Using the same timer.
- The timer was only stopped once the request response was returned to the device.

The other aspect to that was shown by the initial research was the worry about the security of the biometric data - with most of the concerns for biometrics being around this issue. The research further highlighted that users feel far more comfortable when their data is stored locally rather than on an external server. To explore this, I also asked participants to fill out a pre-testing and post-testing questionnaire, asking about their current authentication methods, and general feelings on the use of biometric authentication. This allows me to assess the participants' feelings towards using biometrics both locally and externally.

This was conducted by developing questionnaires on iSurvey, which the users then answered before or after the system testing.

The results for this testing can be found in the Evaluation section.

5.3. Biometric Simulation

In order to simulate the biometric capabilities, example fingerprints were used from the FVC 2000 DB2-B (FVC 2000 DB2-B, 2017) fingerprint database. This was used in the development of the verification module, and testing the client's ability to send the biometric data.

6. Evaluation

In order to effectively evaluate the project as a whole against the goals, this section will be broken down into three main areas of evaluation:

- The security of the application, in line with Goal #1 (See Introduction).
- The technical aspects of the application itself, including the use of specific biometric processes to validate users, in line with Goal #2.
- The usability of the system in comparison to existing systems and current research (see Literature Review), in line with Goal #3.

6.1 Security

The security focused design has also lead to many possible attack vectors being blocked during the planning stage - leading to a more secure system. To evaluate the possible attack vectors, I have combined the initial threat analysis and attempted to perform such an attack, evaluating the security mechanisms in place to defend against such an attack.

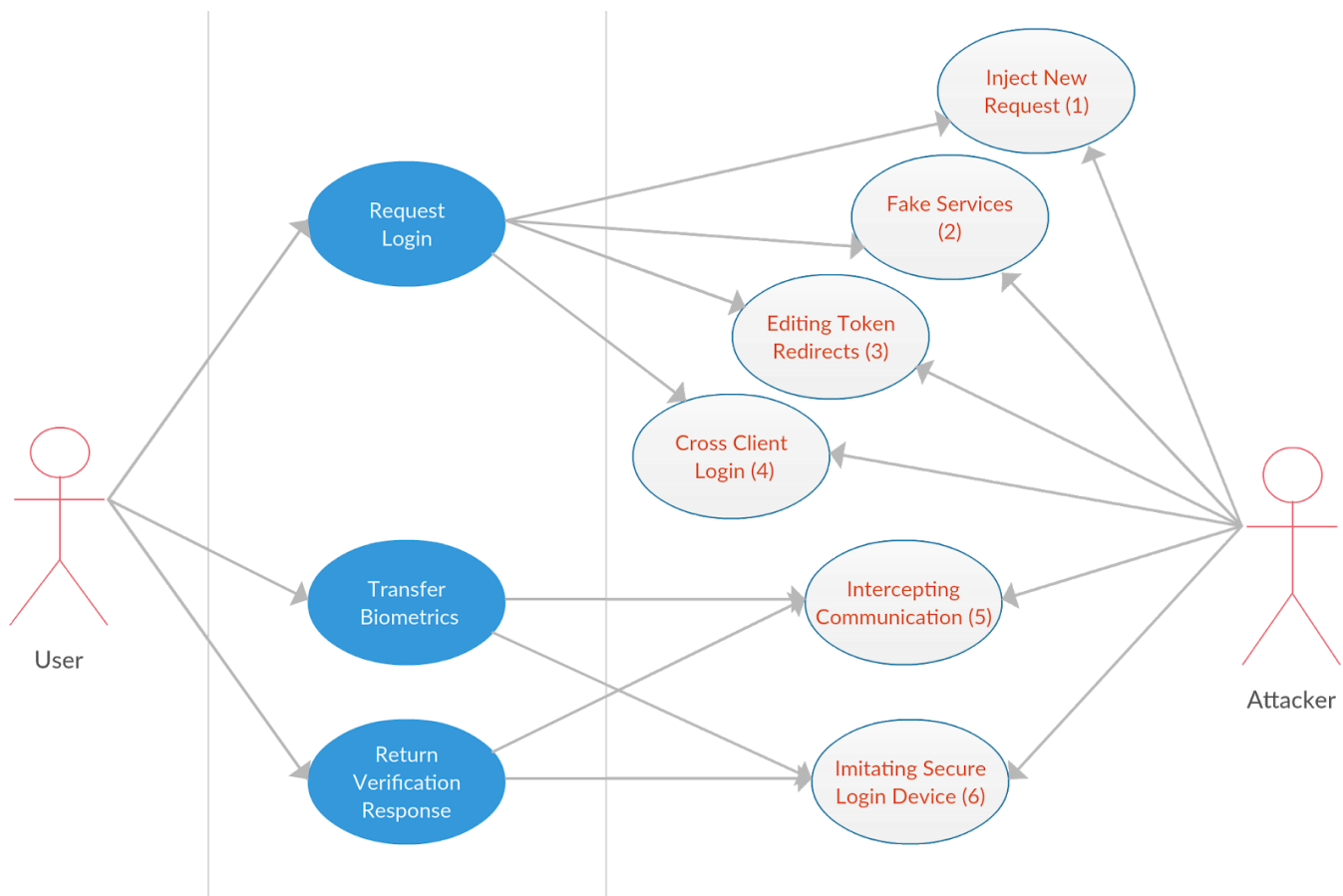


Figure 6.1.1: Misuse case diagram showing possible attack vectors for the use cases.

Each of these threats has been described as a misuse case, giving an evaluation of the security methods used. (Details for each misuse case can be seen in Appendix E)

Misuse Case 1: Inject New Request:

Implemented Security: The OAuth scheme for request authentication uses the **OAuth state variable**, which uniquely identifies each request. Preventing another login request using a verification message from a previous request.

Evaluation: This protects users against this type of attack, however it does not prevent users actually verifying the fake requests themselves, without them realising it is fraudulent.

Misuse Case 2: Fake Services

Implemented Security: The OAuth scheme for request authentication uses **IDs** and **Secrets** to verify the identity of any service making requests to the CF. The attacker cannot generate requests without knowing this secret information.

Evaluation: Securely storing the secrets can often be difficult, and if they are stolen then the service account is compromised. To fix this, the service should be able to generate new id and secret.

Misuse Case 3: Editing Token Redirects

Implemented Security: All Redirect URLs are set on service creation, therefore they are not sent over the network - but extracted directly from the database.

Evaluation: This is secure as the redirects can be.

Misuse Case 4: Cross Client Logins

Implemented Security: Access and Authentication tokens are used to verify a user's authenticity, if both do not match what is registered to the user's account then the request will fail.

Evaluation: Authentication tokens persist and cannot be re-generated; this makes the system less secure as, once stolen, they would be unable to be changed. Giving users a way to be able to revoke tokens would fix this insecurity.

Misuse Case 5: Intercepting Communications

Implemented Security: All requests must use HTTPS, it is not possible to connect using HTTP.

Evaluation: Any vulnerabilities in HTTPS could be exploited, leaving the request unsecured. For high value assets (biometric data) it may be profitable to encrypt the data using RSA to ensure its integrity.

To evaluate the SSL ciphers in use, I used SSLabs (SSLabs, 2017). The site tests ciphers in use and key lengths to evaluate the security. The framework scored a B (grades from A-F) as it accepts the RC4 Cipher, which is generally used on older browsers and has known insecurities, which is a possible vulnerability. Editing the Python SSLContext to prevent connections using the Cipher would prevent this.

Misuse Case 6: Imitating Secure Login Devices

Implemented Security: As SLD's are registered, they are issued a client_id and client_secret. These are known only by the device, so the device cannot be spoofed.

Evaluation: Storing the secret on a client's device is inherently insecure, if the app is analysed then it can be extracted. It may be more secure to fashion the secret so it can be stored in the Java Keystore, which is designed to store secure keys (Keystore, 2017).

Misuse Case 7: Imitating the Framework

Implemented Security: A static IP to connect to within the client.

Evaluation: Currently there is no framework verification - so if an attacker posed as the framework then they would be able to extract biometric data from the SLD. The attacker would have to edit the IP of the framework in the client.

This shows that the framework is resistant to many forms of cyber attack, allowing it to keep Clients data safe within the system - fulfilling Goal #1.

6.2 Approach

I believe the approach to planning and designing the project gave me a significant advantage, as the majority of the design could be simply translated into simple requirements. This allowed me to carefully plan my work, making sure to take into account both bug fixing, testing and report writing.

6.3 Technical Evaluation

6.3.1 Tools and Software

Due to the changes in design, I believe that the system that has been implemented is both scalable and flexible. Using HTTP POST has allowed an ease of implementation for web services wanting to implement the framework, making it easy to be utilised.

Using a multi-instance based model for processing was also a success, as although it gives a higher overhead, it allows for fast processing of connecting clients and as they are only connecting for a few seconds per request, the overhead was not too great.

However, using Sockets between the SLD's and the Framework slowed development significantly; there were many issues surrounding the Python to Java sockets and communicating in-between them.

Both Python and Flask were also a good fit for the project, as both are designed to be simple and (reasonably) fast. This has lead to fast and secure development, whilst maintaining the usability of the system. Other systems, such as node.js may have been faster, but lack the image processing capabilities necessary for the biometric processing.

Using Java for client side has both complicated and simplified the design of the client applications. This table sums up the advantages and disadvantages of using Java:

Using Java for Client Application	
Advantages	Disadvantages
Allows the code to be run on multiple OS's	Cannot use the inbuilt Push Services implemented by most modern operating systems
Allows the codebase from the Android version to be ported over	Requires Java to be installed on the client machine
Sped up development due to my previous knowledge in Java	

The technical approach to this project has allowed me to implement a fast and secure platform for the biometric authenticator, fulfilling Goal #2.

6.3.2 Biometrics

The chosen implementation of the fuzzy vault contained several disadvantages to the performance of the desktop implementation to the system. The largest issue was the strictness of the verification, which proved a challenge. As the implementation only uses one image of the fingerprint for registration, it requires the fingerprint image to be identical to that image on verification – any slight change in angle or pressure results in a different set of minutiae being present, thus failing the verification.

Systems such as Android fix this by taking many images of the actual fingerprint and combining them, which allows a better registration – a system such as this would prevent these issues occurring in the framework.

Anonymity Method

The anonymity method chosen was an implementation of the fuzzy vault scheme, as discussed in the Literature Review and Design sections. This is effective in creating tolerant intra user data, allowing each user's biometric to be transformed into a new unique dataset. However, the fuzzy vault scheme struggles when matching due to the error correction schemes used to separate the chaff points from the real points, thus reducing the matching capabilities (Jain, 2007).

It may have been more profitable to implement an invertible transform due to ability to change up the transform in use, so sets of users could utilise different transforms. Each time a new biometric is registered a different transform is assigned to it (Jain et al, 2007). This would allow high diversity in the anonymous data, making it far harder to trace the biometric back.

Hybrid biometrics have been proposed to attempt to combine the best of all the anonymity methods. It take a set of different approaches, such as salting and fuzzy vault schemes in order to protect the biometric data. This has the obvious advantage of being more secure, as it combines already known methods, but takes far longer to compute and verify.

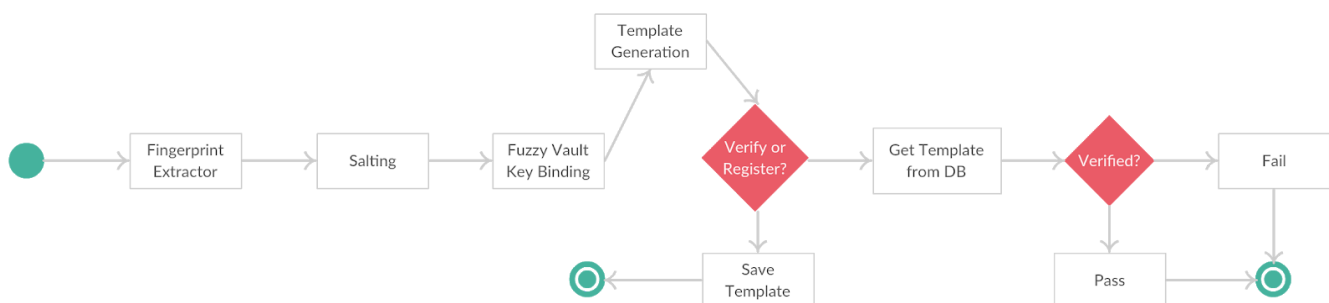


Figure 6.5.1: An example flow of a Hybrid System

Taking this idea further, it may be possible to increase efficiency by salting on the client side and sending the salted biometric, then performing the heavy computation of the vault on the server - maximising both security and efficiency.

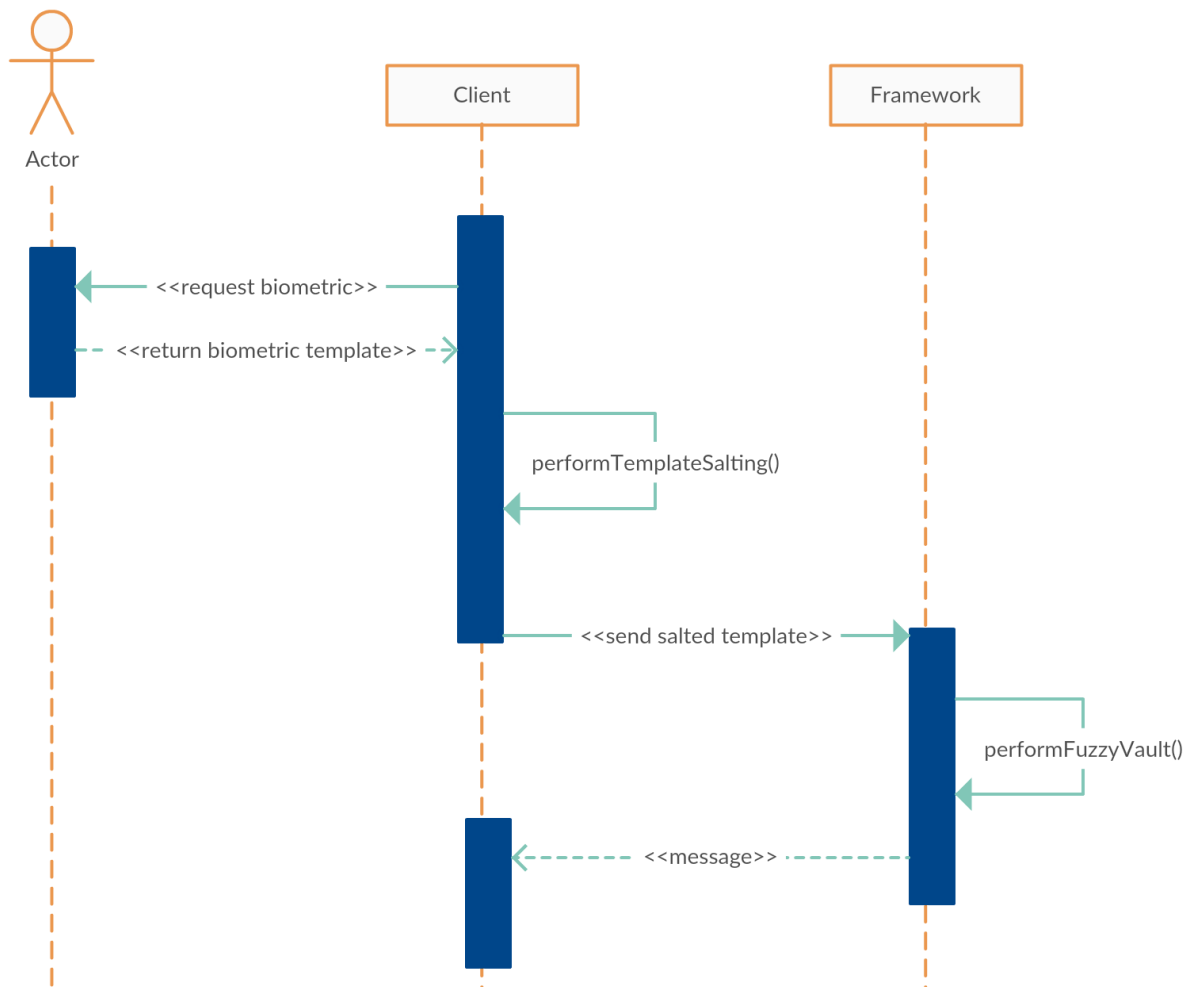


Figure 6.5.2: An example flow of a possible cloud based Hybrid System

6.4 User Testing Results

Task (Time in seconds)				
Participant No.	Creating Account	Registering	BioGate Login	Sussed Login
1	20	30	8	7
2	12	22	6	6
3	30	23	15	8
4	13	46	8	6
5	9	24	10	9
6	16	18	6	4
7	17	15	7	11

Figure 6.1.1: Raw Results for User Testing

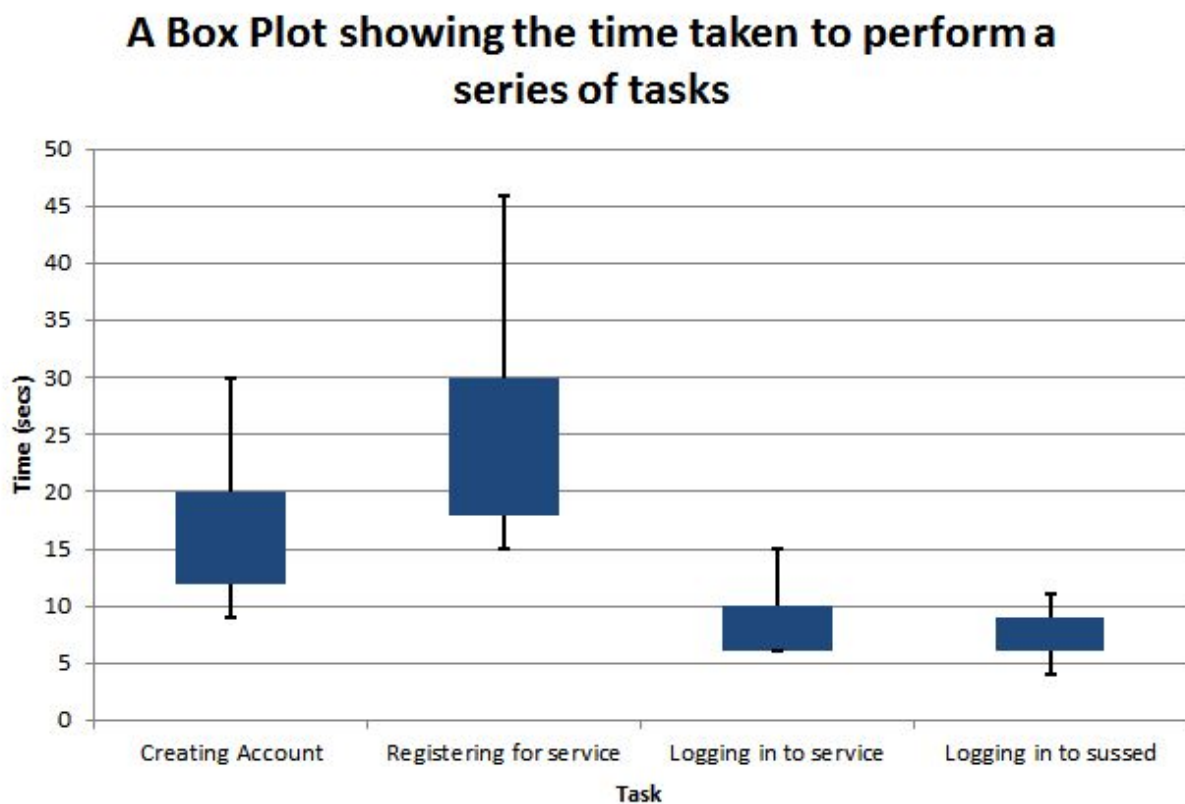


Figure 6.1.2: These box plots show the time taken for each task

From this it is clear to see that the BioGate Framework is very similar in login speed when compared to logging in to sussed, with the mean for BioGate being 8.5secs and for Sussed being 7.2secs. Furthermore, for the BioGate framework no information needs to be entered whereas logging in to sussed (student's university account) requires an email and password.

This is supported by the results from the questionnaires, which showed that 4/7 participants thought that the framework was either faster or possibly faster than their current methods of authentication, and two of the other participants had to wait longer than 7 seconds for the firebase notification - which skewed the data slightly.

However, speed doesn't necessarily imply usability. To gain the participants' views on this, they were asked to rate the usability of four different authentication measures before and after conducting the user testing (detailed results in Appendix I). This showed that more participants believed that fingerprint authentication was the most usable after using the framework than before using it.

The participants views on the security of using fingerprints was also assessed. It showed that the most secure method was one time passwords (OTP) followed by fingerprint authentication. This highlights the importance of OTP's, supporting the integration of them into the BioGate Framework - possibly making the system more attractive to users.

Interestingly, participants were more keen on having the data stored on a company's database as opposed to a government controlled database, suggesting users may be more keen on utilising this framework within the private sector as opposed to the public sector. However, the data also shows that users preferred having the biometrics stored locally (laptop/mobile) compared to anywhere online. This ties in with previous research for password managers; suggesting that the framework combines the best of local storage and usability to create an effective and secure authenticator.

Despite the evidence, these results lack external validity due to the size of the sample and the use of opportunity sampling, leading to an unrepresentative population sample. Further research should use stratified sampling on a larger population to fix this.

In conclusion, the user testing has shown the system to be a promising first step into mass online biometrics, with the users who tested the system suggesting that the system was usable - thus fulfilling Goal #3. It has also shown that the android implementation is likely to be more widely accepted due to its local storage and verification, but it also shows that users may be willing to use some form of cloud version if the biometric is securely anonymised.

Conclusion

To conclude, this paper has proposed an new utilisation for biometric authentication in order to allow users to securely authenticate themselves for an online service. The initial research highlighted the importance of security within the implementation - forming the basis for much of the design work. This has led to the development of a secure and usable system, completing all goals set for the project, thus showing that biometric authentication can be used to develop a secure online authenticator.

Further Work

As discussed, this implementation could be expanded by further research into several areas. These are:

- Research into hybrid biometrics - finding optimal methods to increase template security whilst reducing size for client anonymization.
- Further research into the usability of biometrics, analysing possible other metrics (such as amount of data needed to remember) which users may use to judge an authentication system.
- Implementing a better scheme to combine fingerprint images client side, to get an general image of a fingerprint.
- Develop methods to verify the framework from the client, to prevent framework spoofing.

References

- ISO/IEC 24745:2011. Information Technology - Security Techniques- Biometric Information Protection.
- Ratha, N.K., Connell, J.H. & Bolle, R.M., 2001. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3), pp.614–634.
- Bolle, R. M., Connell, J. H., & Ratha, N. K. (2002). Biometric perils and patches. *Pattern Recognition*, 35(12), 2727–2738. [https://doi.org/10.1016/S0031-3203\(01\)00247-3](https://doi.org/10.1016/S0031-3203(01)00247-3)
- Connell, J. (2006). Cancelable Biometrics : A Case Study in Fingerprints Cancelable Biometrics : A Case Study in Fingerprints, (May 2015). <https://doi.org/10.1109/ICPR.2006.353>
- Jain, A Nandakumar, K., 2007. Biometric Template Security.
- Topcu B. et al., “Biohashing with fingerprint spectral minutiae,” in *IEEE Int. Conf. of the Biometrics Special Interest Group* (2013).
- Tuyls, P., Akkermans, A. H. M., Kevenaar, T. A. M., Schrijen, G.-J., Bazen, A. M., & Veldhuis, R. N. J. (2005). Practical Biometric Authentication with Template Protection. In *Proceedings of the 5th international conference on Audio- and Video-Based Biometric Person Authentication* (pp. 436–446). Springer-Verlag. https://doi.org/10.1007/11527923_45
- Yager, N., & Amin, A. (2004). Fingerprint classification: a review. *Pattern Analysis & Applications*, 7(1), 77–93. <https://doi.org/10.1007/s10044-004-0204-7>
- Clancy, T. C., Kiyavash, N., & Lin, D. J. (2003). Secure smartcardbased fingerprint authentication. In *Proceedings of the 2003 ACM SIGMM workshop on Biometrics methods and applications - WBMA '03* (p. 45). New York, New York, USA: ACM Press. <https://doi.org/10.1145/982507.982516>
- Google App Engine: <https://cloud.google.com/>
- Chen, D. & Zhao, H., 2012. Data Security and Privacy Protection Issues in Cloud Computing. 2012 International Conference on Computer Science and Electronics Engineering (ICCSEE), 1(973), pp.647–651. Available at: files/443/Chen and Zhao - 2012 - Data Security and Privacy Protection Issues in Clo.html.
- Juels, A. & Sudan, M., 2006. A fuzzy vault scheme. *Designs, Codes, and Cryptography*, 38(2), pp.237–257.
- Sun, Y. et al., 2014. Data Security and Privacy in Cloud Computing. *International Journal of Distributed Sensor Networks*, 2014.
- Subramanyam, K., Frank, C.E. & Galli, D.H., 2003. Keyloggers: The Overlooked Threat to Computer Security. Available at:

<https://www.thevespiary.org/rhodium/Rhodium/Vespiary/talk/files/911-Keyloggers0283.pdf> [Accessed March 12, 2017].

Durumeric, Z. et al., 2014. The Matter of Heartbleed. Proceedings of the 2014 Conference on Internet Measurement Conference. Available at:
http://delivery.acm.org/10.1145/2670000/2663755/p475-durumeric.pdf?ip=152.78.38.23&id=2663755&acc=OA&key=BF07A2EE685417C5.A13CBF7F1C3C7DF4.4D4702B0C3E38B35.595DDC89FD3F921D&CFID=738243443&CFTOKEN=69642219&__acm__=1489340526_bf099005314ac47ca480a82a0b658329 [Accessed March 12, 2017].

Haque, M.S., 2016. Web Server Vulnerability Analysis in the context of Transport Layer Security (TLS). IJCSI International Journal of Computer Science Issues, 13(5).

Anil Kumar Meka, Andhra Pradesh, 2003. ENSURING THE INTEGRITY OF AN ELECTRONIC DOCUMENT. Available at:
<https://docs.google.com/viewer?url=patentimages.storage.googleapis.com/pdfs/US20030028774.pdf> [Accessed March 12, 2017].

Gentry, C., 2009. A FULLY HOMOMORPHIC ENCRYPTION SCHEME. Available at:
<https://crypto.stanford.edu/craig/craig-thesis.pdf> [Accessed March 12, 2017].

Lauter, K., Naehrig, M. & Vaikuntanathan, V., 2011. Can Homomorphic Encryption be Practical? Available at:
<http://delivery.acm.org/10.1145/2050000/2046682/p113-lauter.pdf?ip=152.78.38.23&id=2046682&acc=ACTIVE>
[SERVICE&key=BF07A2EE685417C5.A13CBF7F1C3C7DF4.4D4702B0C3E38B35.4D4702B0C3E38B35&CFID=738243443&CFTOKEN=69642219&__acm__=1489341244_5f6be3917a1f2b176058100a808e8258](http://delivery.acm.org/10.1145/2050000/2046682/p113-lauter.pdf?ip=152.78.38.23&id=2046682&acc=ACTIVE&key=BF07A2EE685417C5.A13CBF7F1C3C7DF4.4D4702B0C3E38B35.4D4702B0C3E38B35&CFID=738243443&CFTOKEN=69642219&__acm__=1489341244_5f6be3917a1f2b176058100a808e8258) [Accessed March 12, 2017].

Kaur, A. & Bhardwaj, M., 2012. HYBRID ENCRYPTION FOR CLOUD DATABASE SECURITY. IJESAT] INTERNATIONAL JOURNAL OF ENGINEERING SCIENCE & ADVANCED TECHNOLOGY, (23), pp.737-741. Available at:
http://ijesat.org/Volumes/2012_Vol_02_Iss_03/IJESAT_2012_02_03_56.pdf [Accessed March 12, 2017].

Hardt, E., 2012. The OAuth 2.0 Authorization Framework, RFC Editor. Available at:
<https://tools.ietf.org/html/rfc6749> [Accessed March 12, 2017].

Bansal, C., Bhargavan, K. & Delignat-Lavaud, A., 2013. Discovering Concrete Attacks on Website Authorization by Formal Analysis. , 46. Available at:
<https://hal.inria.fr/hal-00815834> [Accessed March 14, 2017].

Sun, S.-T. & Beznosov, K., 2012. An empirical analysis of OAuth SSO systems. In Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12. New York, New York, USA: ACM Press, p. 378. Available at:
<http://dl.acm.org/citation.cfm?doid=2382196.2382238> [Accessed March 14, 2017].

Yu, S., Lou, W. & Ren, K., Chapter 5.3: Data Security in Cloud Computing. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.714.8198&rep=rep1&type=pdf> [Accessed March 14, 2017].

Leinwand, A. (2016). Why Cloud Architecture Matters: The Multi-Instance Advantage over Multi-Tenant | Service Matters. Retrieved March 13, 2017, from <https://servicematters.servicenow.com/why-cloud-architecture-matters-the-multi-instance-advantage-over-multi-tenant/>

Gordon, D. (2009). The Advantages of a Multi-Tenant SaaS Architecture. Retrieved March 13, 2017, from <https://blog.samanage.com/cloud/the-advantages-of-a-multi-tenant-saas-architecture/>

The Benefits of SaaS Multi-Tenant Architecture | Signiant. (2016). Retrieved March 13, 2017, from <http://www.signiant.com/articles/the-benefits-of-saas-multi-tenant-architecture/>

ServiceNow. (2015). Advanced High Availability Architecture. Retrieved from www.servicenow.com

Chong Frederick, Carraro Gianpaolo, W. R. (2006). Multi-Tenant Data Architecture. Retrieved March 13, 2017, from <https://msdn.microsoft.com/en-us/library/aa479086.aspx>

Mell, P., & Grance, T. (n.d.). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-145>

Li, Z., Zhang, H., O'Brien, L., Cai, R., & Flint, S. (2013). On evaluating commercial Cloud services: A systematic review. *Journal of Systems and Software*, 86(9), 2371–2393. <https://doi.org/10.1016/j.jss.2013.04.021>

Galante, G., & De Bona, L. C. E. (2012). A survey on cloud computing elasticity. *Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012*, (1), 263–270. <https://doi.org/10.1109/UCC.2012.30>

Microsoft Azure, <https://azure.microsoft.com/en-us/?b=17.06> [Accessed 14/03/2016]

Amazon EC2, <https://aws.amazon.com/ec2/> [Accessed 14/03/2016]

Tams, B., 2016. Unlinkable minutiae-based fuzzy vault for multiple fingerprints. *IET Biometrics*, 5(3), pp.170–180. Available at: <http://digital-library.theiet.org/content/journals/10.1049/iet-bmt.2014.0093>.

Tams, B, THIMBLE. Available at: <http://www.stochastik.math.uni-goettingen.de/biometrics/index.php?id=22&language=en> [Accessed March 14, 2017].

FVC 2000 DB2-B, Accessed 2017 <http://bias.csr.unibo.it/fvc2000/download.asp>

Jain, A Nandakumar, K., 2007. Biometric Template Security.

Triggs, R., 2016. PSA: 34% of you aren't even using a lockscreen password |

AndroidAuthority. Available at:

<http://www.androidauthority.com/psa-use-a-lockscreen-password-668689/> [Accessed March 18, 2017].

Java Keystore, [Accessed 28/04/2017]

<https://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html>

Ahmed, A., Ahmad, S., Ehsan, N., Mirza, E., & Sarwar, S. Z. (2010). Agile Software Development : Impact on Productivity and Quality, 287–291.

Appendix

Appendix A - Hardware Costs & Design Archive

SecuGen Hamster Pro Fingerprint Scanner - £72

https://www.eyenetwork.com/secugen-hamster-pro-20.html?category_id=21

Design Archive Contains:

/biogate_framework

- All code from the framework (server side)

/biogate_android

- All code deployed on android

/biogate_desktop

- All code deployed for the desktop application

/biogate_otp

- All code + images for the chrome extension

/biogate_mock

- All code for the mock service

README

Appendix B - Initial Design Specification:

Component: Server Side Authentication for Clients Connecting

Description: Any client connecting to the server must be appropriately authenticated to utilise the server functions.

Functional Requirements:

- The system should be able to distinguish the client connecting
- The system should be able to verify the devices that they are connecting from.
- The system should be able to handle multiple connections at once.

Component: Storage Systems for storing user data

Description: The system must be able to store the data in a secure environment, which is accessible through the controller.

Functional Requirements:

- The system should be able to securely access the biometric and account information.
- The database must verify any connections to it to validate they are genuine.

Non-Functional Requirements:

- The system data cache for a user must be under n bytes.

Component: Securely receiving biometric data

Description: The server must be able to securely receive biometric data from a client and utilise it to authenticate the login.

Functional Requirements:

- The system must be able to receive the biometric data from the user.
- The system should be able to handle multiple connections at once.
- The system must be able to validate the user being connected.

Component: Verification of Biometric Data against stored data

Description: The biometric data must be stored in a secure area, and then verified against the stored data.

Functional Requirements:

- The system must be able to utilise a biometric SDK in order to verify the data
- The system must be able to securely connect to the BioDB.
- The system must securely obtain the result and respond to the client.

Non-Functional Requirements:

- The System must be able to handle verification of Fingerprint Biometrics.

Component: Generate Web Account API for the system to use

Description: The System must have an external web API which can alter account information such as password, usernames, devices and biometric data.

Functional Requirements:

- Must allow users to create/update devices for a particular account
- Must be able to edit account details, such as password/email.
- Must have account recovery information.

Component: Generate Web Verification API for the system to use

Description: The System must have an external web API, called the WVAPI which enables services to connect to the framework and utilise its functions.

Functional Requirements:

- Must allow a Service to attempt to verify a client.
- Must securely return a result to the Service in order to login.

Component: Client Applications

Description: This must be able to utilise the WAAPI to alter the data of the Account. It should also connect to aspects of the WVAPI.

Functional Requirements:

- Must be able to register new devices with the user accounts
- Must be able to capture fingerprints and transmit them to the server
- Must be able to notify the user when a login is needed.

Component: Accounts

Description: Must be able to store the Accounts database and allow verified connections through the WAAPI to access the data.

Functional Requirements:

- The system must be able to securely access the Accounts Information to create/update/delete accounts.
- Must give a secure link to the Biometrics data.

Component: Mock Service Implementation

Description: Must utilise the WVAPI and WAAPI to to create a mock service which shows the use of the framework.

Functional Requirements:

- Must be able to securely contact the Web Verification API and login a user depending on the result.
- Must be able to tell the client that is currently signed in to the browser

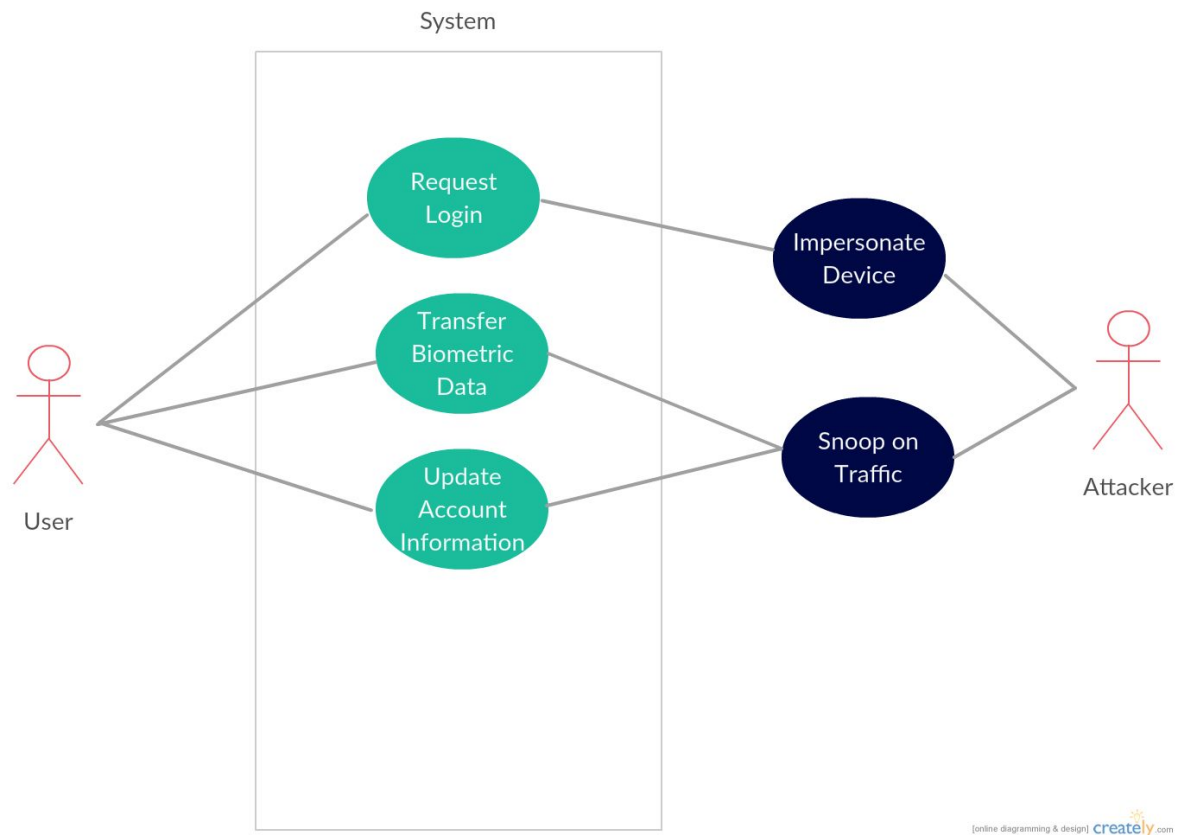
Appendix C - Asset Analysis:

Asset	Value	Exposure
Account Database	Medium as stores some information about the client.	High. Cost of restoring the system and loss of service to clients. Damaged Reputation.
Biometrics Database	High as stores all of the biometric data.	High. Cost of restoring the system and loss of service to clients. Damaged Reputation.
An Individual client's Biometric data	Low, but could be high for specific cases.	The direct losses would be low, but reputation would be damaged.
Validation Framework	High. Required for all logins the system processes, could give out invalid login validations.	Very High. A insecure system would give a very poor reputation.

Appendix D -Threat and Control Analysis:

Threat	Probability	Control	Feasibility
An unauthorized user attempts to spoof a message to the database in order to gain access to personal data.	Low	Require all communications to the database to be appropriately signed, so the location and validity can be verified.	Reasonably easy to implement into the system to counteract this, however more advanced attacks are far more difficult to defend against.
An unauthorised user snoops and intercepts communications from the server to a valid client.	High	All communications should be sent over HTTPS. It may also be necessary to encrypt the biometric data before sending.	Using HTTPS is trivial to implement, but extra encryption may cause computational cost and slow down the system.
An attacker attempts to spoof a valid login message from the server in an attempt to bypass verification	Medium	All validate login messages must be signed with appropriate information so the online service can verify the message.	May be quite complex to implement.
An attacker may gain physical access to the database.	Low	Only use trusted server hosting and host different databases in different locations	Using larger, reputable companies is preferred, but separating components may make the system too slow to be used effectively by users.
An attacker may attempt to imitate the SLD.	Medium	Secure keys should be used in order to verify a client's device identity.	Device verification is tricky to correctly implement but can be done. However, a sophisticated attacker may be able to get access to the CUK and imitate them.
An attacker may pose as the central framework and spoof the device into sending data to it	Low	Using specified and secure communication links in order to verify incoming messages with the server.	May be difficult to implement, as it requires significant method validation, however digital signatures should work as needed.

Appendix E - Misuse Cases



1. Use case: Login request

Actors: User, WVAPI, CF

Description: A user may request to log into the system. This notifies the CF, which then notifies the SLD to initiate biometric capture. This data is then sent to the server which verifies the biometric data passed and sends a response to the initial request source.

Data: User's Biometric data, User's account information.

Stimulus: Clicking a Login via BioID on a services login page.

Response: The boolean response to allow the service to log the client in.

Misuse case 1: Inject New Request

Actors: User, WVAPI, CF, Attacker

Description: A User attempts to login, an attacker attempts to inject a new request for a different service, so when a user verifies themselves they CF processes the attackers login request - logging the attacker in. Would require a user's account details to have been compromised.

Data (assets): User's Account on the Service

Attacks: An attacker **injects new requests** into the network (spoofing old requests sent by a user), getting the user to verify the attackers login request rather than their own. Would require a network monitor to notify the attacker when a user requests a login.

Implemented Security: The OAuth scheme for request authentication uses the **OAuth State variable**, which uniquely identifies each request. This prevents another login request using the verification message from a previous request.

Further Vulnerabilities: A User may not realise they are actually logging in an attacker and verify the attackers request

Misuse Case 2: Fake Services

Actors: User, WVAPI, CF, Attacker

Description: A User requests to log into a fake service, which would allow an attacker to gain legitimate access to users tokens, and could then use them to attempt to log the user into other services.

Data (assets): User's Account on the Service

Attacks: An attacker uses the **stolen tokens** to request a login to another service, which the user then accepts. This gives the attacker access to the users account for the requested service.

Implemented Security: The OAuth scheme for request authentication uses **IDs** and **Secrets** to verify the identity of any service making requests to the CF. The attacker would not be able to make requests to another service without knowing this secret information.

Misuse Case 3: Editing Token Redirects

Actors: User, WVAPI, CF, Attacker

Description: A User requests to log into a service, the service sends the request to the CF along with a redirect URL to post the tokens to once authenticated. If an attacker could edit this redirect, they would be able to post the access tokens to themselves, thus hijacking the request. This data could then be used later on to generate fake requests.

Data (assets): User's authentication tokens

Attacks: An attacker uses a network monitor to **capture** and **alter** requests coming from a service to the CF, **editing the redirects** to be able to steal the tokens.

Implemented Security: All Redirect URLs are set on Service creation, therefore they are not sent over the network - but extracted directly from the database.

Misuse Case 4: Cross Client Logins

Actors: User, WVAPI, CF, Attacker

Description: An attacker requests a login for an account (one they own), then edits the request using a tool such as **BurpSuite**, replacing some of the information with another user's credentials. As the verification request is for the attackers account, they can verify the request - but it logs in the targeted user due to the edited information.

Data (assets): User's authentication tokens

Attacks: An attacker sends login requests to the CF, replacing the id to that of the target user. The attacker then verifies the request using their own SLD, and the CF notifies the service that the target user has logged in - logging the attacker into the user's account on the service.

Implemented Security: Access and Authentication tokens are used to verify a user's authenticity, if both do not match what is registered to the user's account then the request will fail.

Further Vulnerabilities: None

Misuse Case 5: Intercepting Communications

Actors: User, WVAPI, CF, Attacker

Description: An attacker intercepts a login request from a client, stealing the data contained inside it. This could be tokens or biometric data.

Data (assets): User's authentication tokens, user's biometric data

Attacks: An attacker uses a tool such as **WireShark** to monitor the network traffic of a specific user, extracting the data from the unsecured request.

Implemented Security: All requests must use HTTPS, it is not possible to connect using HTTP.

Further Vulnerabilities: Any vulnerabilities in HTTPS could be exploited, leaving the request unsecured.

Misuse Case 6: Imitating Secure Login Devices

Actors: User, WVAPI, CF, Attacker

Description: An attacker requests a login for a specific user, and then imitates the users SLD to send a positive verification message back to the CF (android version) or stolen biometric data (desktop version), resulting in the attacker being logged into the user's account.

Data (assets): User's account for the service

Attacks: An attacker sends login requests to the CF, imitating a user. The attacker then poses as the SLD, and with the correct protocol the attacker would be able to send their own response to the CF - bypassing the biometric verification completely.

Implemented Security: As SLD's are registered, they are issued a client_id and client_secret. These are known only by the device, so the device cannot be spoofed as this information is required for authentication of the request.

Further Vulnerabilities: None

Appendix F - Biometrics Security Analysis

In order to ensure that all aspects of the project are secure, I will now use the Data Life Cycle (described in Literature Review) to choose specific security measures I can take in order to develop an effective prototype.

Generation

For general cloud services, this is not an important aspect - and generally cannot be controlled. However, for a system such as mine where I will have software running on the client machines it is of critical importance. Once the Biometric data has been captured on a user's PC, it is insecure. In order to protect it I will encrypt it using an RSA public key generated by the server and requested by the client. This will afford the biometrics some security, however I must also make sure that once the data is transmitted it is deleted from the system - using the gc call in Java to make sure the garbage collection removes the data from memory as quickly as possible.

Transfer

To protect the data in flight, all data must pass through HTTPS using TLSv1.2. In order to increase the security of the vulnerable data I will also encrypt all Biometric and password data with a secure RSA public key - so even if a flaw with TLSv1.2 is discovered the data will still be secure.

Use

Due to the nature of the Fuzzy Vault implementation the data does not need to be decrypted to be verified - meaning it is secure through the entire process. The same is also true of passwords, which are hashed upon arrival to the server, and can be compared whilst hashed - so they are never in cleartext at any point.

Also, SessionID's will be used to verify that the session being validated is the same as the session that was requested. This will mean that a session can only be verified if the sessions match - if they don't then the system is trying to verify the user for a session that wasn't requested.

Sharing

As the Framework is designed to be a third party authenticator it must have a secure method of sharing its verification capabilities. Because of this I have decided to implement a full OAuth2 scheme in order to allow other services to use resources presented by the framework, but keep a user's credentials secure.

In order to verify the devices clients are communicating from, I will use a scheme of sharing secrets - where the clients device is given a secret, which is used to verify the

identity of the device. This will prevent malicious users from stealing users credentials and spoofing the device.

Storage

In order to protect the data within the database, it will be encrypted with AES (including the biometric data, which is already stored in the encrypted Fuzzy Vault form).

Also, the database will only be able to be accessed through the container controller to prevent unauthorized connections.

Archival

No biometric data will be archived, accounts which have been inactive may be deactivated to free resources

Appendix G - Risk Analysis

Risk	Severity	Probability	Exposure (P*S)	Handling/Mitigation
Loss of code or documentation.	4	2	8	<p>Mitigation: Use Git and other Version Control to make sure local code is not lost.</p> <p>Handling: If code or documentation is lost, I will have to re-plan and re-prioritise work in order to complete all work needed.</p>
Unable to complete project within given deadline	5	2	10	<p>Mitigation: I will make sure all of the work is planned out and prioritised in order to verify I am on track to complete the project.</p> <p>Handling: If I cannot complete all the work, then the lower priority tasks will be omitted from the project.</p>
Supervisor is absent for a significant period of time	4	1	4	<p>Mitigation: Make sure I don't leave any work to the last minute, so I can get feedback as soon as possible.</p> <p>Handling: I can always contact my personal tutor for guidance.</p>
Illness and absence	3	2	6	<p>Mitigation: Attempt to be as healthy as I can throughout the length of the project, and not have 'time off' around deadlines.</p> <p>Handling: I will have to re-prioritise my work in an attempt to stay on track</p>
Inability to	3	3	9	<p>Mitigation: Verify all of the</p>

implement some goals of projects				goals can be implemented with current technologies and plan effectively to stay on track. Handling: I can omit some goals if they are unobtainable for any reason, and adapt the project as needed
Change of SDK API's	5	2	10	Mitigation: Check social media for products I'm using, and make sure there are no big updates coming. Handling: I can either use the older (working) version or attempt to re-write portions of my project as needed.

Appendix I - Raw Questionnaire Results

Appendix J - Project Brief