APRIL 23, 2018

TECHNICAL MANUAL

MID – MY IDENTITY

CILLIAN MC NEILL - 14352621
SCHOOL OF COMPUTING
SUPERVISOR: GEOFF HAMILTON

Table of Contents

Introduction

This document will outline the system, its architecture, high-level design, as well as the problems encountered during the development of the application.

Overview

MiD is an identity engine that will aid in proving a person's identity. Using a mobile application, it stores information on the user locally and will only be sent out with the user's permission. This information is verified by respected institutions and the proof of that verification will be stored with a distributed, tamper-resistant solution called the "Blockchain".

This system aims to tackle the problem of identity fraud, where the need for someone to prove who they are in the modern, social-network driven society is incredibly difficult, as many people leave themselves vulnerable by voluntarily exposing very personal information. By streamlining this process into a safe and secure service will aid in providing not only peace of mind to users but will save companies, and even countries a lot of money in the long run.

Detailed documentation on the development can be found on the Gitlab blog or the Trello board that I used to create and monitor development tasks.

Motivations

This idea behind this project came about as a result of my time spent in Mastercard during my INTRA placement. My work there cantered primarily around authentication through biometrics. While there I noticed a need for a platform that would allow companies to quickly and easily authenticate their end-users.

There was also a personal drive behind the idea in this project. To have a universal form of identity, recognised by any institution saves time and effort for everyone, especially the issuing institutions. Having a service like this would allow them to work in conjunction, eventually phasing out and replacing different identity documents by a single "master" one. On a case by case basis the system is saving the user hundreds in processing fees and on the large scale it removes the need for numerous institutions, saving the government large quantities of money. I found the idea of this fascinating and I believe it would be the way forward for personal authentication around the world.

Glossary

- **Blockchain**
  - A continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block typically contains a link to a previous block along with any relevant data. Thanks to this and the overall design, blockchains are inherently resistant to modification of the data stored on them.
- **Distributed Ledger**
  - Replicated, synchronized and shared digital data that is spread across multiple locations.
- **Distributed Systems**
  - A model of computer networks in which the systems pass messages to one another to complete a task. The distribution of these systems allows for concurrent work to be done and without a single point of failure.
- **JSON**
  - A format that is easy for humans to read/write and computers to parse. Used in the transmission of messages across a network.
- **Hyperledger Fabric**
  - A branch of the Hyperledger Project (open source blockchain tools and distributed ledgers) originally created by IBM. It is a permissioned blockchain infrastructure using concepts such as roles between nodes and “smart contracts” to facilitate trading.
- **Node**
  - A JavaScript based development platform
- **Relational Database**
  - A digital database that divides data into tables with rows and columns. These tables can then be queries from the data within each table.
- **APK – Android Package Kit**
  - A package file format for the Android operating system.
- **API – Application Programming Interface**
  - A set of defined methods to allow for communication between various software components.
- **SAFe – Scaled Agile Framework**
  - A form of the agile methodology employed in large organisations. It promotes collaboration and scalability with numerous teams.
- **TDD – Test Driven Development**
  - Translating software requirements into short tests and developing a short repetitive cycle of adding to these tests as new requirements are created.
- **BDD – Behavioural Driven Development**
  - Principles expanded from Test Driven Development (TDD) to allow both developers and project managers to easily manage a project.
- **Identity Type**
  - A form of identity containing personal information (e.g. Passport)
- **Identifying Parties**

    - Authorities that will integrate MiD into their existing systems. They will validate submissions made to their identity type and create a certificate for accepted submissions.
- **Individual**
    - The end users that create and submit identity types of identifying parties to validate and answer requests for the verified information.

Research

Below details the research that went into the technology that makes up the infrastructure of MiD. It lists what I needed to do and what I decided on for the final implementation.

Blockchain

The biggest part of the project is storage and retrieval of a digital certificate on the blockchain that would allow anyone to see that your identity was valid. They would need to be easily created and queried by anyone at any time. Along with this, they need to be entirely secure in their creation so that only the identifying parties can create them.

To this end, I investigated different forms of blockchain implementations that I could use. The obvious versions of it would be some of the better-known ones, such as Ethereum but the problem with these is their use of a form of currency (Ether) to commit data to the chain. I wanted an implementation that I had control over and didn't need to worry about a currency or proof-of-work. Through research online and conferring with my colleagues in MasterCard I came across Hyperledger and Hyperledger Fabric. This is an open source blockchain implementation that takes on a permissioned structure. With this you can control who can what is updated to the ledger. This was perfect for my project as there needed to be a divide between user's and parties and Fabric would make this possible.

Backend Implementation

The backend would need to tie everything together, allowing every user to communicate with each other as needed and update their data. To allow for easy integration into any company's system I wanted to make all the functionality of MiD available through endpoints that users could contact. Each endpoint would carry out a specific task within MiD and allow any user to carry out the tasks required to create, verify and request identities.
There are many implementations of a server that I could use. As detailed in the project proposal I proposed to use spring on a tomcat server. There was research into libraries such as Node, but I felt I was more familiar with Java and the surrounding work that would be required to deploy it to tomcat. It would also allow me to implement libraries that would speed up the development process. I wanted to implement libraries such as:

- JDBC
    - Tie server into a relational database for easy data retrieval
- Swagger
    - Easy documentation of API endpoints in a user-friendly UI
- Google Firebase SDK
    - Library to tie into a Google Firebase project
- Cucumber
    - Easy implementation of Java BDD testing

There are implementations of these in other languages but, from working with them on other projects, I understand how to use them more through Java and due to the time constraint, I felt a Spring/Tomcat implementation would be the most time-effective solution.

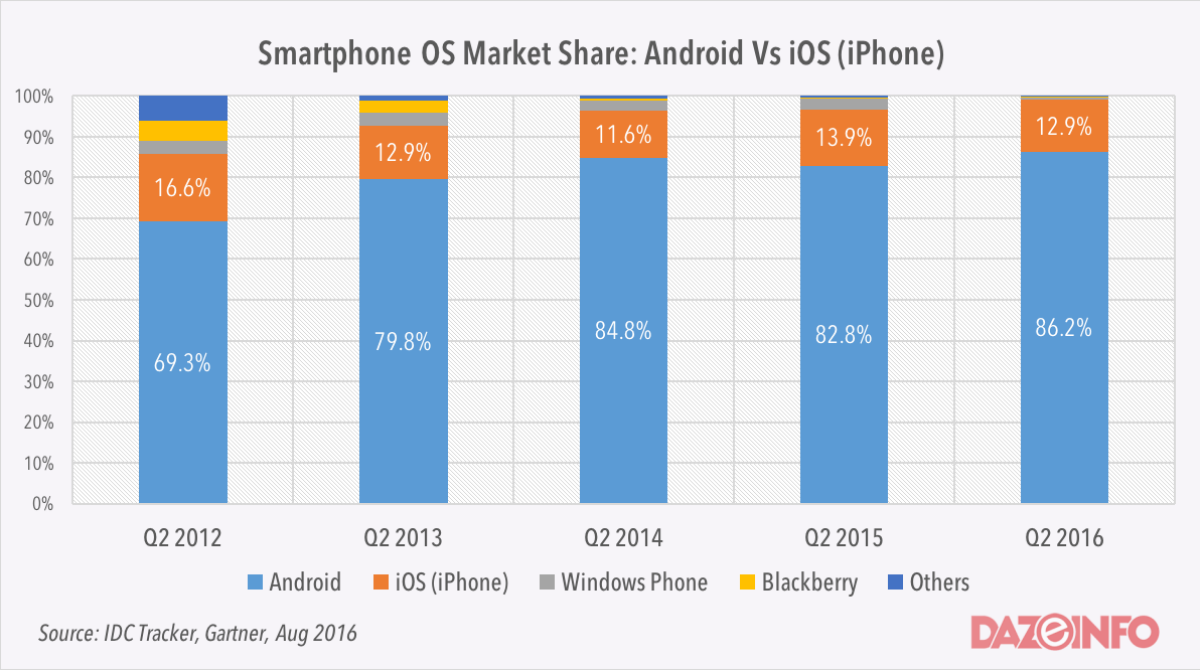
User Implementation

Below details what research went into choosing the platforms that each user would use to interact with MiD. It's important to note here that the backend of MiD is just a set of endpoints so any platform could have been used but I needed to choose one to demonstrate functionality. There is no reason that new platforms could be used to expand the ways a user could access MiD.

Individual

An individual wants to be able to access their information on the go so it was natural to assume that the user would want their information on a mobile application. From there I needed to decide on which platform I wanted to develop the application for.

There are several mobile operating systems I could make the application for, the main ones being IOS, Android and Windows. After consideration and research, I decided on developing it for Android.



Over the last few years the market share of Android has increased more and more. As I want to reach as many users as possible with the initial launch of the application, choosing android would be the safest bet.

Along with this, Android offers a plethora of free tools for a developer to use to develop and test a mobile application. Platforms such as Goole Firebase and the Google Play Store allow developers to quickly and easily deploy and test their application. Firebase will also allow a server to communicate with a specific mobile through notifications. This is OS independent but it's a very important feature for MiD as I will need to inform the user of updates within MiD and integrating it into Android is a "click of a button" process.

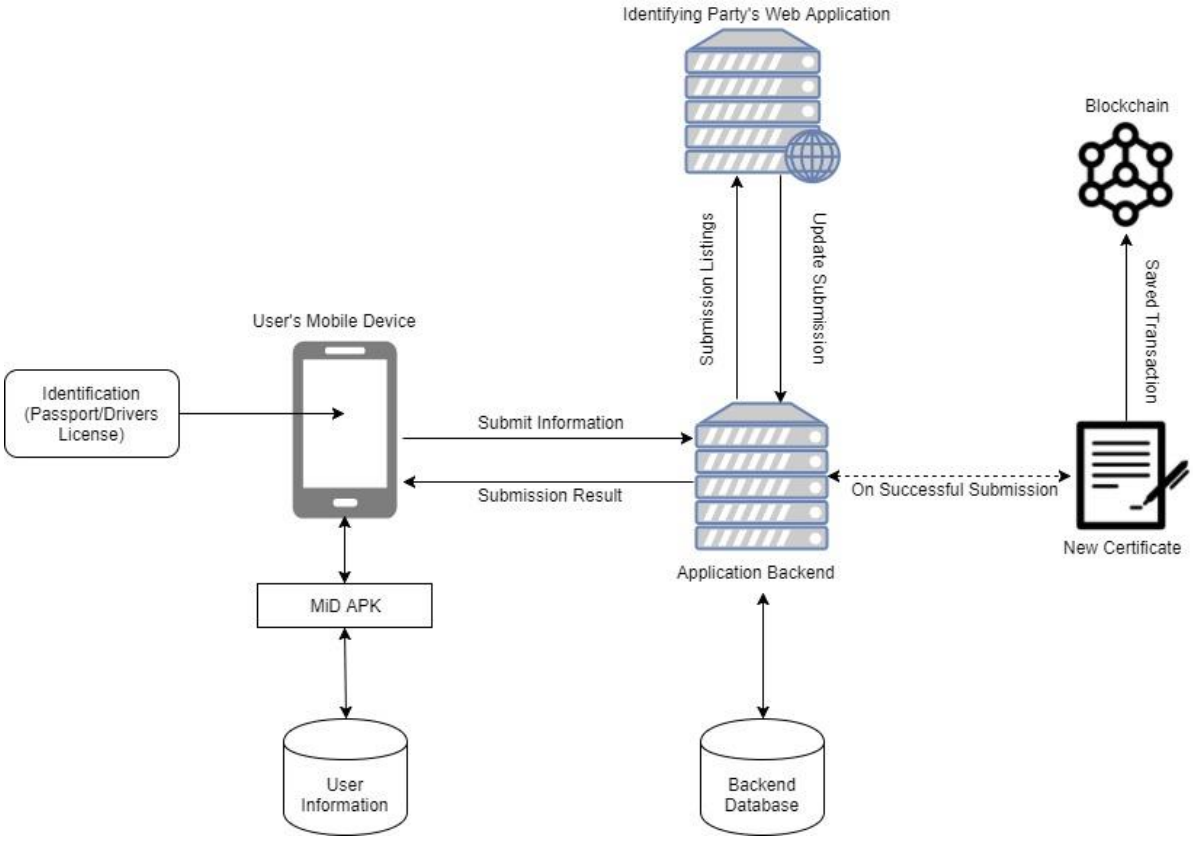
Identifying Party

As noted before, MiD is a set of endpoints that any service, given the correct steps, can use. This was done to allow it to be integrated into an existing application.

This means that an Identifying party shouldn't need a UI but will instead integrate the endpoints into their companies existing UI. A test UI was developed in Angular to demo the workflows that a party could work through, but it is by no means a complete example of what an authority should use.

System Architecture

High Level Design



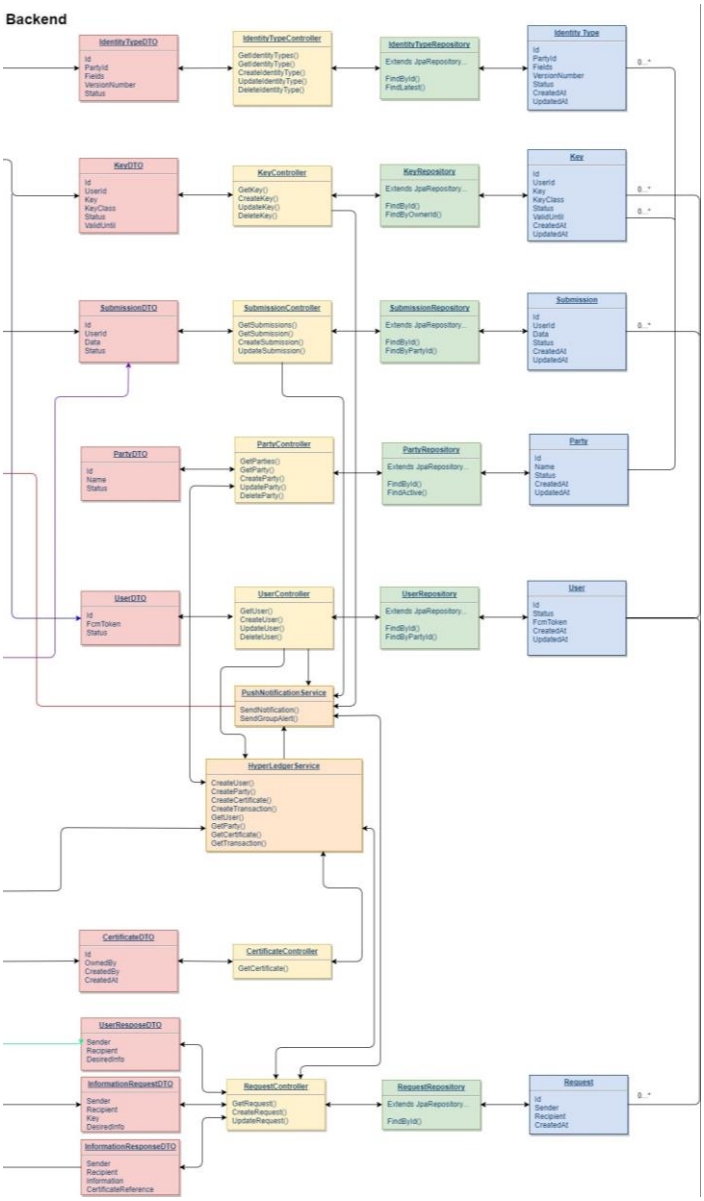
The overall design of the system has not changes since the functional specification. The diagram above draws out how the main components of the system function and how they communicate. A mobile device is the main channel through which an individual makes and answers requests.
An identifying party interacts with the system through their own application. They pull down any requests made to their service and evaluate them. Results of this evaluation are sent back to the system. Any successful submission is carried over to the blockchain in the form of a new certificate in the submitter's name.
Any other relevant information is stored on the application's backend. This includes information that will allow the system to communicate with an individual's device as well as information linking to an individual's currently pending submissions and active certificates on the blockchain.

Backend Design

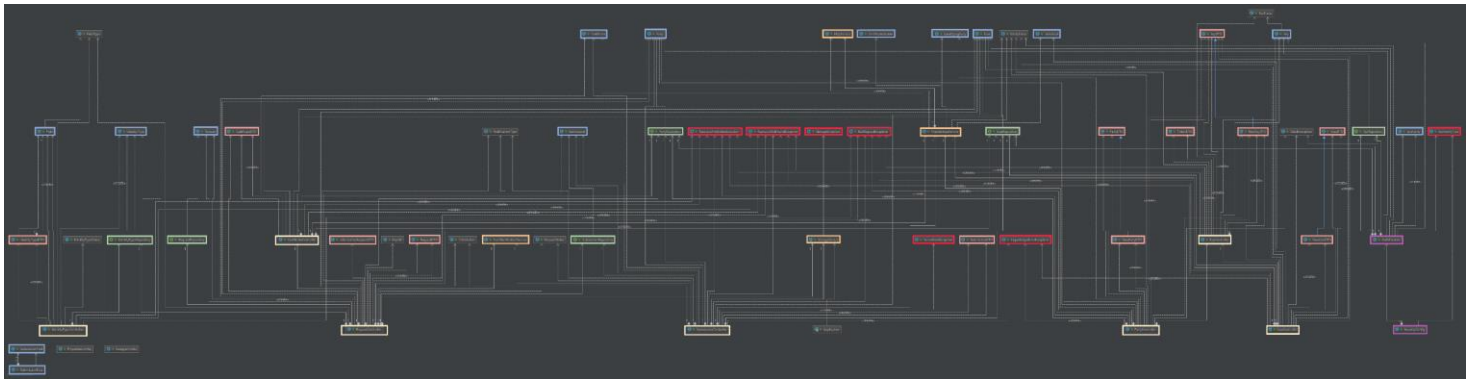
Below details the design of the backend of MiD. It will look at the overall design of this portion of the application as well as reasoning behind some of its design choices. For detailed documentation please consult the user manual or the API documentation.

Overview

During the initial design phase of the project this diagram was drawn up along with the rest of the components to show how they would communicate with each other.
Requests came in and were mapped as the java DTO objects. These objects were then handled by their respective controller. These controllers would then interface with JDBC repositories that used Java objects annotated as their respective database tables. With this configuration, all logic remained within the server and could be configured easily without having to worry about the knock-on effects it would have on connected components.
The database would be created entirely with JDBC annotations. This made for easy “drag and drop” deployment of the MiD application into tomcat. Any changes to tables, field properties or even database language could be configured directly through the configuration files in MiD.



The design has changed very little from the original diagram. New classes were created to facilitate new functionality (such as security related classes for authentication) but the general structure remains the same.



The overall complexity of the class diagram has increased but the basic log remains the same. (a higher resolution version can be found here).

The breakdown of the classes are as follows:

- Yellow
  - These are the server's controllers. Any request matching a path within the server is mapped to one of these controller's methods.
- Pink
  - Server's DTO java files to be used for marshalling and unmarshalling of JSON request data
- Green
  - JDBC repository interfaces to connect to the servers database and allow for easy data transmission
- Blue
  - Annotated Java classes for use with the JDBC interfaces
- Orange
  - Server service files for inter-server communication (eg. Communication with Google Firebase and Hyperledger instantiation)
- Purple
  - Security classes used for user authentication
- Red
  - Exception classes mapped to HTTP error codes
- Uncoloured
  - Configuration and enum classes used across the application

Design Choices

Below details the main design choices that were made when creating the backend portion of the application.

Data Storage

As noted before, the backend makes use of JDBC annotated Java classes to create and maintain its databases. This choice was made for two main reasons:

- Ease of deployment
    - When the application is deployed the server will run the JDBC initialization scripts. These will scan the application for annotated classes and repositories and set up the tables according to what is entered. This means that all a user needs to do is to enter their details into the configuration files within the application. The server will handle all database creation.
- Ease of maintainability
    - As the server creates the tables it means that any changes to the annotated classes will be reflected in the database tables. Name changes, column length, primary/ foreign keys and a variety of other annotation data can be entered and updated to allow for any sort of application configuration.

MiD makes use of a “StorageService” java class that handles the storing and retrieving and large data files that cannot be stored within the applications database. Once again for ease of use, the application makes use of the server’s local storage to store the files. There are security and data integrity concerns here but the ease of configuring a server file path and having it taken care of by this class currently outweighs this. By design, the files are also fully encrypted so the only main concern here is data-integrity. As this is only an initial version of the application there is no reason that the service can’t be updated to handle SFTP to store files offsite but for now this is a simple and effective solution.

Authentication

MiD makes use of token-based, basic authentication using HTTP headers. When a user makes a request, they must have the required token to make any sort of request. The workflow is as follows:

- The user registers with MiD (registering their public RSA key) and receives a token back
- When a user needs to make a request, they encrypt this token with the private key and send it as a header within the request
- The server unencrypts the token and verifies that it matches what is on file for that user.

This allows the server to verify the user and then decides on what calls they can make and what data they are allowed access to within those calls. This isn’t a bulletproof form of authentication. On a standard HTTP connection between a user and a server the key can be intercepted and reused by another user to impersonate the original owner using a replay attack. To this end, MiD makes use of full HTTPS for all communication so the traffic is fully encrypted to outside users.
There are third party authentication services that can be leveraged but I feel this allows me the ability to keep the application as a closed system while still offering a form of security to all users. This can of course be built upon for a more complete version of the application, but this demonstrates the security of the application with no issues.

Mobile Application Design

Overview

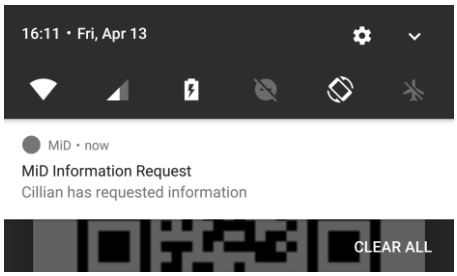
The mobile application acts as a wrapper for the calls that any user can make to the server. It allows anyone to interact with the server and view their data and view it in an easy to user system.

The application is broken down into three separate layers, the visual layer that the user interacts with, the local storage layer that stores all application relevant data and the server communication layer which handles all calls to the server and returns them back to be processed by the app.

Visual Layer

This layer handles all user interaction and uses the other two layers to display relevant information to the user. The primary function of the application is to allow users to view their identity cards. This is done using an edited version of Florent Champigny's MaterialViewPager to display each field of the card in an easy to use manner.

The application is tied into Google Firebase so it will handle notifications from the server and display them to the user. Everything within the application is locked behind a 4-digit pin that the user creates during the initial registration phase. To view any of the user's information they must first enter this pin.

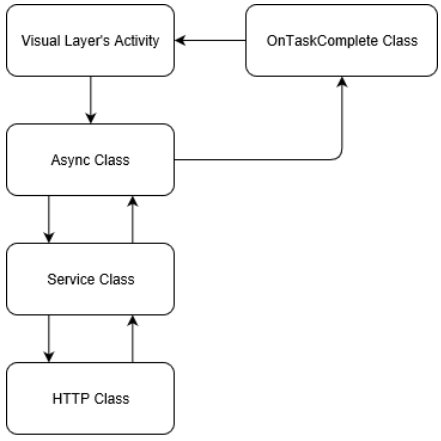


Local Storage Layer

This layer stores all local data for the application to call upon and display within the visual layer. This makes use of a custom "DatabaseHandler" class that exposes methods the application can use to retrieve Java classes back that represent the tables in the local database. This design came from my use of JDBC and has made local storage within the application very easy to work with.

Server Communication Layer

This layer handles all communication with the server using a layered structure. The visual layer calls upon the desired asynchronous class for the request. Each type of call has its own asynchronous class that returns different data.



Now off the main UI thread, This class wraps the request in the authentication headers required and passes it on the respective service to run in the background. This service then sets up the correct paths to the server and calls the base HTTP class to make the actual request. This generic class handles any call by each service and returns it back to the service. This request is passed back up to the async class which calls to an interface that the visual layer overrides to implement its own logic. When the request has finished in the background it will enter into the visual layers implementation of this method and carry out whatever logic is required on the data returned from the server.

Design choices

Android Version

The application supports Android 5.0 and upwards. The choice for this was the easier implementation of some of android's base functions (eg. creating dialogs) and the higher market share of this version. Android 5.0+ makes up over 80% of the current market using the android operating system and that number is only rising.

App Security

The application makes use of a 4-digit pin to secure the data within the application. A password isn't required to create an account on the server, only the user of an RSA public/private keypair which is generated by the application itself and not the user. This means that the user only needs to worry about securing the data on the local storage of the phone. Data on android is partitioned and is not accessible from another application without the use of "Rooting" tools which is not something an everyday user has access to. If a malicious attacker wanted another users phone then they must first break the screen lock on the phone which could be a mixture of numeric, alphanumeric or biometric locks.
Of course, Security can be further increased locally using biometrics or a longer alphanumeric password but as a proof-of concept application it is not currently necessary.

Blockchain

Overview

The blockchain is what is used to store and verify submissions made to a party within MiD. When a user has their identity reviewed and accepted by the corresponding identifying party they have a certificate created in their name on the blockchain. That certificate is referenced in any requests made to that user for information, so the receiving user can know that the information retrieved is valid.
The Hyperledger Fabric blockchain implementation was used for MiD. It allowed me to create an implementation with access control and no proof of work on the data being added to the blockchain. This made development a lot simpler. Rather than proof of work, Hyperledger makes use of levelled identities and varying consensus algorithms to commit transactions to the blockchain. It also allows for privatisation of data within the blockchain, allowing only certain parties to view parts of the blockchain without affecting the overall structure of everything else.

MiDs implementation makes use of 2 models, 1 asset and 1 transaction type:

- Models: → Ids of are referenced within MiD
    - Individual
        - individualId
    - Identifying Party
        - partyId
- Assets
    - Certificate
        - certId
        - SubmissionHash → comma separated hash of each field of the identity type
        - CreationDate
        - Trustee → The party that creates the certificate
        - Owner → The individual that owns the identity type
- Transactions
    - UpdateStatus → used to revoke a certificate if the need arises
        - Certificate → original certificate
        - New Status → status it is to be updated to

Design Choice

The primary design choice here was the decision to use Hyperledger over another implementation. As noted before I research other implementations and decided on this one. I feel the structure of this matches the exact structure I envision for a global blockchain identity network. A partitioned structure between countries and states that allows for access to lower levels from the higher ones (eg. A mayor can see their cities structure, but the president can see all states). The lack of proof of work and the inclusion of access control means that the structure I envision is entire possible and easily achieved using Hyperledger. It is easily implemented, scaled and added to at any point.

Problems and Resolution

This section details the problems encountered during the design and implementation of the system and the actions that were taken to resolve them.

Problem	Creating database friendly identity cards
Description	Ensure that the cards are easily stored and retrieved
Solution	By using comma/semicolon seperated data fields I was able to create a structure that would allow for easy storage and retrieval of the identity types Eg. FIRSTNAME:Firstname,BIRTHDAY:Date of birth…

Problem	Creating blockchain certificate storage
Description	Create the appropriate classes to create and reference certificates on the blockchain
Solution	Using simple models to identify an individual and party I can then create a certificate that can reference all these models. All three of the ids can then be stored within MiD for easy reference. When a user is created we simply call Hyperledger to create the appropriate user (Individual/Identifying Party) and store the returned Id along with the rest of the user data within MiD When a submission is accepted we create the certificate and store the returned id with the submission data.

Problem	Notifying Users on mobile device
Description	Be able to notify a specific mobile device anywhere in the world
Solution	Using Gooogle Firebase I was able to reference any user by their FCM token. This means that at any point I can notify the user of events within MiD. This saves a massive amount of overhead and complications that would ensue from implementing a communication structure myself

Problem	Authenticating Users
Description	Be able to authenticate users and ensure they have the correct role within MiD
Solution	Make use of the basic authentication header within HTTP requests to authenticate users. A user will send their server id and an encrypted token that will be decrypted and matched against what's stored on the server. This token can be updated at any point if needs be. Roles are assessed based on the user's server id, if their id matches a user within the user table then they have the role of a user and if it's a party id then it's a party role.

Problem	Securing user data
Description	Ensure that data is secure and only viewable by the correct users
Solution	Make user of RSA and AES encryption methods. Encrypt the data using a symmetric AES key, encrypt the AES key with the recipients public RSA key then send the data to the server to be retrieved by the user.

Problem	Storing user data
Description	Store user data in a logical location that can be easily retrieved at any point
Solution	Leverage the security of the application and store the encrypted data within a file. This file is then stored on the server in a location defined by a server configuration file. A storage class is used to save and load data to local storage. Save the path to the file within the database for easy retrieval. This is applicable to user submissions as the data submitted is quite large.

Problem	Ensuring Secure data transmission
Description	Ensure data transmitted between the server and user is secure and unreadable by malicious users
Solution	Leverage the HTTPS communication protocol to ensure all data is secured during transmission.

Problem	Demonstrate Party-side workflows
Description	Ensure there is a user UI and backend to demonstrate workflows while still sticking to MiD security policies
Solution	Create basic angular website with a service to call to a configured server. This server will wrap all the UI's calls with the correct authentication headers and store any necessary data for the UI.

Problem	Mobile-side communication with server
Description	Easily allow the mobile application to contact the server and retrieve data that can be used by the Android app
Solution	Create a generic HTTP class that services can use to call specific endpoints on the server. Asynchronous classes need to be created to leverage these services as calls to the server cannot be made on Androids main UI thread.

Problem	Server-side communication with blockchain
Description	Creation of a service that will allow for easy communication with the blockchain implementation created for MiD
Solution	Expose the blockchain through endpoints that are accessible by the server only. Make use of an edited version of Androids generic HTTP class to call on these endpoints for easy data transmission.

Problem	Easy Documentation of backend endpoints
Description	Be able to easy reference any endpoint within MiD and understand how it works
Solution	Import a swagger dependency and configure it to expose all of MiDs endpoints. This will rebuild every time the application is deployed so the documentation is always up to date.