Ammaar Ayub Patel

Decentralised Chain-of-trust Based Teaching Certification

Computer Science Tripos - Part II
Robinson College
2022

Declaration of Originality

I, Ammaar Ayub Patel of Robinson College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. I am content for my dissertation to be made available to the students and staff of the University.

Signed: Ammaar Ayub Patel

Date: 12/05/22

Proforma

Candidate Number:	2340C
Project Title:	Title
Examination:	Computer Science Tripos - Part II, 2022
Dissertation Word Count:	11910[1]
Code Line Count:	10214[2]
Project Originator:	The Dissertation Author
Project Supervisor:	Mansoor Ahmed-Rengers
High-level Supervisor:	Anil Madhavapeddy

Original Aims of the Project

The original aim of my project was to explore a digital and decentralised educational certification system, inspired by historical systems which used chain-of-trust-based certification. The intention was to produce some software that facilitates the issuing and verification of digital credentials to represent such certificates, and that decides the roots of trusts and the ontology of the subjects using a voting system.

Work Completed

The project successfully achieved its original aims. It builds on existing blockchain-based tools to represent educational certificates as self-sovereign credentials. It facilitates roots-of-trust (called masters) and verification of the credentials via chain-of-trust using the masters as valid endpoints. It facilitates a subject ontology, with each credential (including masters' credentials) having a subject, and the chain-of-trust verification taking this into account along with the ontology. It uses voting to adjust the set of masters and the ontology of subjects, and allows masters to vote if their master credentials are sufficient to vote on the matter, according to the subject ontology.

Special Difficulties

None.

[1] Computed using Google Docs' word count feature.

[2] Computed using cloc: https://github.com/AlDanial/cloc (v1.92). The breakdown is shown in Appendix III

Contents

1 Introduction

1.1 Overview

Educational certificates have become one of the most important things in our lives, and we spend years working towards earning them. The certificates are considered a major factor in career prospects and other aspects of life. My project explores a new solution for issuing, managing, and verifying educational certificates.

1.2 Background

1.2.1 Current Systems

Currently, institutions issue certificates to their students, and others can verify the legitimacy of the certificate with the institutions. The institutions are the source of truth, with the certificates meaning nothing on their own. This is an easy system to use, but the institutions are integral. The institutions are privy to the usage of the certificates, allowing them to gain information about people's private lives. If an institution is compromised, that information could be exposed, or the certificates could be lost. These issues are made worse with malicious external actors. For example, a dictatorship could monitor people's activities through their certificates, or take them away as they please. Clearly these are serious problems.

Another issue is institutions can issue certificates in anything. Excluding a few legal restrictions in some countries, they do not need to abide by any standards or common understandings, and rarely do people check what someone's certificate actually means.

1.2.2 Ijaza

Ijaza, which roughly translates to "permission", is a traditional certification system. It is decentralised, independent of institutions. It was predominantly used throughout Islamic history in various areas, from religious sciences, metaphysics, philosophy etc, to subjects like medicine and law. It works on a chain-of-trust model, so when someone is certified, it is their teacher that issues the certificate. A teacher can only issue certificates in subjects they are certified in.

Verification of someone's certification requires verifying it was issued by the teacher to the student, and that the teacher is certified. To verify that the teacher is certified has the same requirements, producing a recursive process which stops at some trusted points/teachers (roots-of-trust). The roots-of-trust are well known and accepted experts, meaning no further proof is needed of their authority in the subject.

A system like this does not have the issues caused by centralised institutions, but it makes verification complicated and still depends on common understandings, both for the subjects and the roots-of-trust. A point to note is in Ijaza being certified in a subject is equal to being qualified to teach it, as it follows a philosophy that you do not truly know something until you can teach it.

To clarify, Ijaza does not exist solely outside of institutions. Many institutions operated under the Ijaza model, acting as hubs of knowledge and learning, and there are still some such institutions today. Rather it meant teachers, or sometimes those in charge of teaching (e.g. heads of departments), issued the certificates rather than the institutions themselves.

1.2.3 Paper vs Digital Certification

Most educational certificates are represented by a piece of paper. This is not ideal, as the certificates can be forged, making them less trustworthy, or they can be lost or damaged. To mitigate this, most institutions keep records of certificates so that people can request another copy, or to allow others to check if a certificate is legitimate. Some offer a unique code on the certificate in order to verify it, usually using a digital platform which reveals the certificate's details.

Recently some institutions have also issued digital certificates, in various forms. However, these are often even more dependent on a centralised authority, as it is the institution, or an organisation tightly associated with them, that "hosts" the certificate, providing a link for the owner of the certificate to share. Some institutions work with platforms that act as middlemen, "hosting" the certificates, and associate them with accounts on their platforms. Their association with the institution confirms the certificates' legitimacy.

As far as I am aware, no significant efforts have been made to improve any Ijaza-based certificates. They are still paper-based, which is a particular issue with chain-of-trust-based certificates, since there is not a single authority or institution you can go to to verify the certificate. The only option would be to register the certificates with some third-party authority, but that goes against the concept of institution-independent certification.

1.3 My Project

1.3.1 A Digital Implementation of Ijaza (E-Ijaza)

My project builds a digital implementation of the Ijaza system, which I call e-Ijaza. It takes the Ijaza concept of chain-of-trust-based teaching certification, and implements it in a way that solves the verification and shared understanding of the subject ontology and roots-of-trust. It also overcomes some of the issues of current systems (section 1.2.1), and some of the issues with paper certificates (section 1.2.3). The chain-of-trust model brings the benefits of being institution-independent, as well as being useful in areas like crafts (e.g. carpentry) where a person's training involves working under someone skilful in the craft until they are able to work in the craft on their own. For clarity, I will use *certificates* to refer to the general concept of Ijaza-based educational certificates, and *credentials* to refer to the implementation of such certificates in e-Ijaza.

To be viable e-Ijaza must achieve certain things. It must define a structure between subjects, as requiring credentials to be issued in the exact subject of a held credential is inflexible. Allowing credentials to be issued in different subjects requires a shared subject ontology, an understanding of how subjects relate to each other. It must also have defined roots-of-trust, which act as universally recognised credentials, otherwise chain-of-trust verification is not possible. Having a shared set of roots-of-trust allows people to use credentials and know they will be accepted by others.

Finally, it must implement credentials that represent chain-of-trust-based teaching certificates. The verification of the credentials involves two parts. One is verifying that it was issued by the teacher to the student. The other is ensuring the teacher holds valid credentials that satisfy the subject according to the subject ontology. And for a teacher's credential to be valid, it must either be a root-of-trust, or it must be recursively verified.

1.3.2 Hyperledger Indy/Aries

Hyperledger Indy[3] is a blockchain-based ledger specifically designed for digital identity. It facilitates self-sovereign credentials, meaning if x issues y a credential, y holds and owns it, and can allow others to verify it without the involvement of x. By credentials here we do not mean the teaching credentials in this project, but something more general, which I call *Indy Credentials* (ICs). ICs prove something that the issuer asserts about the holder. This could be asserting that the issuer sent an email to the holder, where the IC is the email, and the holder can prove that it received it from the issuer. And Hyperledger Aries[4] provides frameworks for building agents to communicate with Indy deployments and other agents on the same Indy network.

E-Ijaza builds on Aries to share the roots-of-trust and the subject ontology, to adjust them via voting, and to produce and use credentials that are decentralised and institution-independent. E-Ijaza also implements a user-friendly interface.

[3] https://www.hyperledger.org/use/hyperledger-indy
[4] https://www.hyperledger.org/use/aries

2 Preparation

2.1 Starting Point

Before starting the project, I had read some of The Sovrin Foundation[5]'s whitepapers explaining how Sovrin/Indy work at a high-level, but had no other knowledge or experience of the technology or related technologies. I had no experience working with blockchain-based technologies or digital/cryptographic credentials.

I was comfortable with typescript[6] and had experience building non-professional web applications in Angular[7], although I had little to no experience with testing. I had some experience building API servers in NodeJS[8], written in typescript using the ExpressJS[9] framework.

I had no experience using Docker other than running scripts that produce docker containers already set up for use.

2.2 Requirements Analysis

The requirements of e-Ijaza are:

- Users will be able to view the set of master credentials, the subject ontology, their credentials, and the credentials they have issued.
- Users will be able to issue credentials in subjects that are reachable from their held credentials, according to the subject ontology, and will be able to revoke them at a later time.
- Users who hold master credentials will be able to create proposals to adjust the set of master credentials or the subject ontology. They will be able to vote on proposals if the subjects involved in the proposal are reachable from the subjects they have master credentials in, according to the subject ontology. A proposal will pass if more than half the possible voters are in favour.
- Users will be able to verify if another user is authorised in a subject. By authorised I mean they hold valid credentials in subjects that together can reach the target subject. By valid credential I mean it is either a master credential, or issued by someone who is authorised in the subject of the credential. This needs to be done whilst maintaining the privacy of users and the credentials they hold beyond what is necessary to reveal.

2.3 Hyperledger Indy (Indy)

2.3.1 The Ledger

The ledger used in Indy networks is a public-permissioned network, meaning everyone can read the ledger, but not everyone can write. *Transaction Endorsers* can write to the ledger, and can write on behalf of others.

An Indy network is run by *Stewards* who run the *validator nodes*. A validator node is one that takes part in *consensus*, which means being responsible for approving transactions and agreeing on the state of the ledger. Each validator node stores and operates on a copy of the ledger.

The ledger is purpose-built for digital, self-sovereign identity (mentioned in section 1.3.2), and only contains data related to that purpose. Privacy is ensured by design, by ensuring no private data is

[5] https://sovrin.org/
[6] https://www.typescriptlang.org/
[7] https://angular.io/
[8] https://nodejs.org/en/
[9] https://expressjs.com/

written on the ledger, even if encrypted or hashed, as such algorithms have a shelf-life, but data on ledger does not.

2.3.2 The Identity System

2.3.2.1 Decentralised Identifiers (DIDs)

DIDs contain 3 parts, an identifier, a DID Document and a method. The method specifies how the DID and DID Document are created, resolved, updated, and deactivated. The identifier is similar to a Universally Unique Identifier (UUID), since it is generated without using a centralised system and depends on a statistically low chance of collision in order to be considered unique. The DID Document contains data on the DID, including its public keys, with the owner of the DID holding the private keys.

Indy uses *public DIDs*, where the DID Document is stored in a central location (i.e. the Indy Ledger). Indy's public DIDs include the endpoint of the DID's owner. The public key can be used with this to establish communication. All writes to the Indy ledger are done using public DIDs, and Stewards and Transaction Endorsers have extra permissions associated with their DIDs.

Pairwise DIDs, also known as *Private DIDs or Peer DIDs*, are used in communication between agents on the Indy Network. They do not involve storing DID Documents in a central location. In every new communication the agents involved create new pairwise DIDs, providing privacy as communications cannot be correlated. Even if a connection is established using a public DID, the agents will still create private DIDs for the communication.

2.3.2.2 Schemas and Credential Definitions

Schemas define the attributes of an IC, and the owner can create multiple versions.

Credential Definitions are needed to issue ICs. They relate to a particular version of a schema, not necessarily owned by the owner of the credential definition. The credential definition contains a public key for each attribute of the schema, and the owner holds the associated private keys.

Schemas and Credential Definitions are written to the ledger.

2.3.2.3 Indy Credentials

ICs (mentioned in section 1.3.2) are what encapsulate digital identity on an Indy network. They are not written to the ledger as they contain private information. An agent can issue an IC using a credential definition they own, assigning values to its attributes, and signing it with the private keys from the creation of the credential definition.

It is worth noting that since an IC is associated with a credential definition, and a credential definition is associated with its owner's public DID, it is possible to find the public DID of an IC's issuer.

2.3.2.4 Revocation

Revocation of ICs involves a *revocation registry*, a *tails file*, and a *cryptographic accumulator*. The tails file contains a set of large numbers, and the cryptographic accumulator is a number produced using a subset of the tail file's numbers. A revocation registry relates to a particular credential definition. It contains information on how revocation is handled, the URI and hash of the tails file, and which cryptographic accumulator is used. The revocation registry and cryptographic accumulator are written to the ledger, and the tails file is hosted externally.

When issued, an IC is assigned an unused factor from an appropriate tails file, and the cryptographic accumulator is updated to include this factor. On revocation the accumulator is updated to exclude the factor. A *proof of non-revocation* uses a cryptographic technique known as Zero-Knowledge Proofs (ZKP) to prove whether a factor is used in an accumulator without revealing what other factors are used. Since nothing is deleted from blockchain ledgers, a proof of

non-revocation can use a past version of the accumulator to prove non-revocation at some past point in time.

2.3.2.5 Proofs

Proofs are sent from one agent to another, and can include attributes from multiple ICs, predicates on attributes from ICs, and self-attested attributes. ZKP is used to selectively reveal attributes from an IC and to prove predicates. Every attribute that is not self-attested includes the credential definition of its IC and a proof of non-revocation if the IC is revocable.

Proofs do not involve writing to the ledger. An agent uses a proof to communicate information to another agent and to prove assertions made by other agents.

2.3.2.6 Limitations

Indy's identity system has a few limitations. Firstly, a proof cannot prove whether two attributes came from the same IC, which is a problem if attributes in an IC are dependent on each other as their relationship cannot be proven.

Secondly, predicates on attributes in proofs are limited to greater or equal to ("≥") and the function of the predicate is not customisable or necessarily standardised. The example usage given is proving a minimum age using a date of birth from an official IC (like a driver's licence).

Finally attributes in ICs are untyped, and are generally assumed to be strings. It is possible to assign a (HTTP) MIME type to attributes in an IC, however it is not clear how supported this is. And the MIME type is asserted in the IC, so the types can vary between ICs with the same definition.

2.3.3 VON Network

VON Network[10] is a mature, development level Indy network that provides a web interface to view the ledger and an API to programmatically register public DIDs, which is not usually possible on production networks. I preferred this over making my own network as running my own as the latter would involve setting up my own Indy nodes and their configuration, and would not be as easy to debug. Setting up Indy Networks is not part of the project, so using a purpose-built and properly tested solution for the Indy Network in the development environment is better practice.

2.4 Hyperledger Aries (Aries)

2.4.1 Aries Frameworks

The Aries project replaced Indy-SDK and provides agent frameworks that can be built on to communicate on identity-focused networks, although they currently only work with Indy. I reviewed the available agent frameworks, and a table of their differences is in Appendix I.

2.4.2 Aries Cloud Agent - Python (ACA-Py)

2.4.2.1 Overview

I decided to use ACA-Py for e-Ijaza. It includes the key functionalities of Indy, including off-ledger functionality such as issuing and storing ICs, managing public/private keys, working with proofs and establishing connections. And it is also easy to set up revocation for ICs with ACA-Py.

ACA-Py is controlled via an API, which allows the application to be programmed in any language. Due to the asynchronous nature of communications, ACA-Py uses a webhook to provide updates on the state of active communications with other agents. It is the most mature of the available frameworks, with the only downside being it does not support mobile applications, but I had no intention of working with mobile applications in this project.

[10] https://github.com/bcgov/von-network

It is worth noting that the ACA-Py is capable of more than I use it for, such as multi-tenancy on one agent, or acting as a transaction endorser for other agents. It also implements protocols that are not accepted yet, which provide additional functionality but which I chose to avoid.

2.4.2.2 Documentation and Guides

The GitHub repository for ACA-Py contains some reasonably maintained documentation and a "new developer" guide, which proved useful in getting to grips with ACA-Py, but there are large sections missing or out-of-date, with some areas simply marked as "to do" and the guide noting that the content of the guides is based on out-of-date versions of the code.

Aside from using out-of-date code, the "new developer" guide has a separate script for starting pre-configured agents. And since documentation on the main scripts and its arguments was lacking, learning how to deploy ACA-Py involved some trial-and-error.

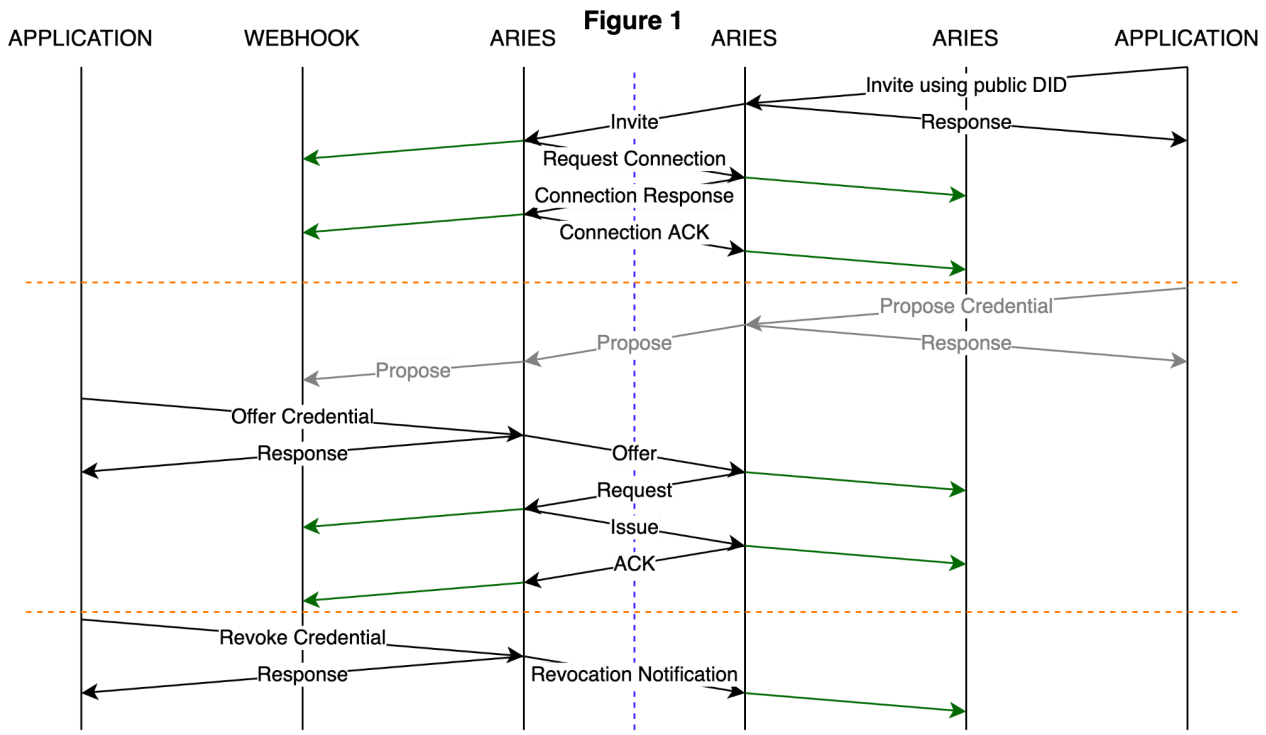
2.4.2.3 API

The API and webhook are not well documented. There is an Open API[11] specification, and ACA-Py exposes a Swagger UI[12] interface for making API calls. The interface was very useful, as I could experiment with the API calls before implementing them, and could implement e-Ijaza in stages whilst manually running the unimplemented aspects. Trial and error became necessary as I realised the Open API spec was lacking. In some places it is inaccurate and in others it is missing information, including which fields are required or what is definitely returned, and what the possible values of fields are.

2.4.3 Communication Protocols

ACA-Py implements a number of communication protocols, including for connections, ICs, and proofs. These protocols involve multiple API calls, although I configured Aries to automatically handle some of the steps. Below are diagrams showing the main protocols. Black is the necessary parts, grey is optional, and green is data passed to the webhook, which can be read by the application.

Figure 1



[11] https://www.openapis.org/
[12] https://swagger.io/tools/swagger-ui/

Figure 1 shows connecting using a public DID, issuing an IC, and revoking an IC.

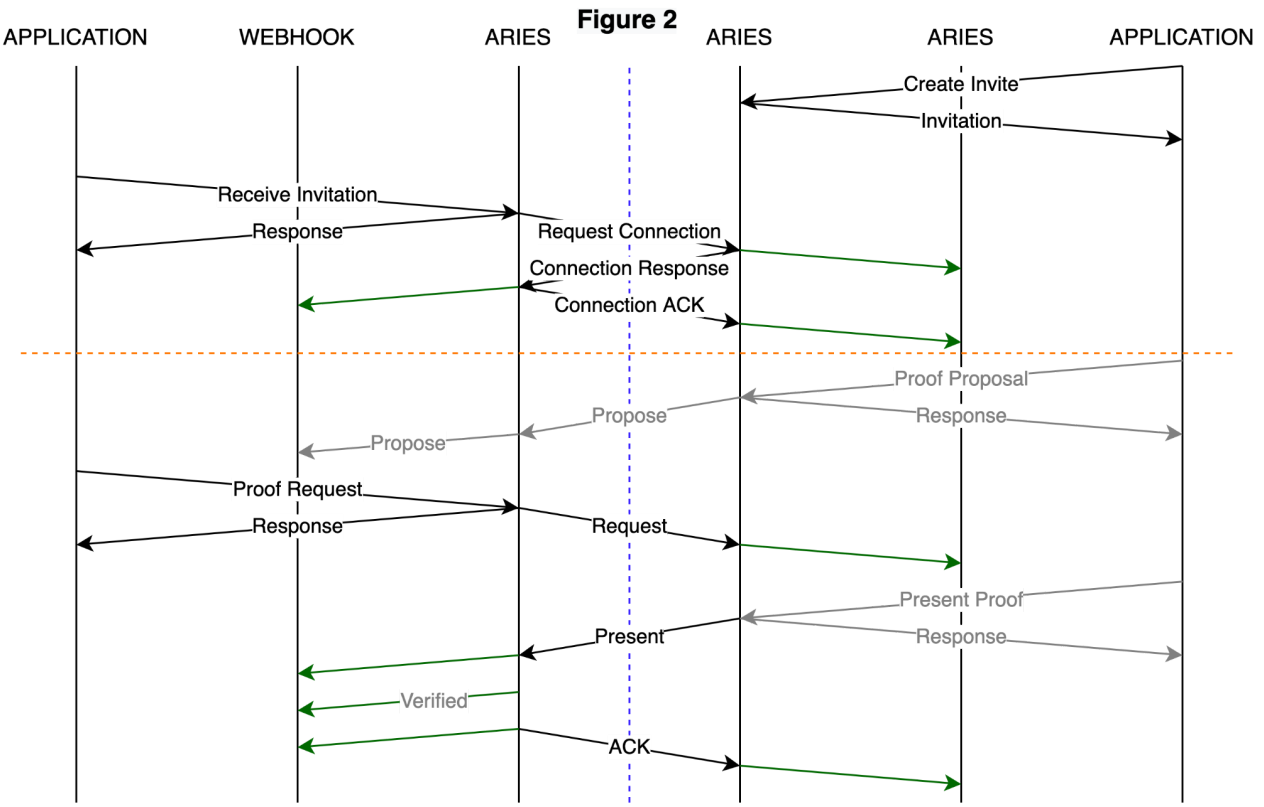


Figure 2 shows connecting using an off-ledger invitation (often passed as a QR code), and presenting a proof. The "Present Proof" edge is optional because if the proof protocol begins with a proposal, and the proposal contains no self-attested attributes and specifies all the ICs to be used, then the proof will be presented automatically. Otherwise, it needs to be manually presented.

2.4.4 Indy Tails Server

Indy Tails Server[13] is used to host tails files (section 2.3.2.4), and ACA-Py can automatically configure with it at start up.

2.5 Credentials

Using ICs to implement e-Ijaza credentials makes the credentials digital, self-sovereign, revocable and provable using Indy and ACA-Py.

In the project proposal I proposed being able to merge and narrow credentials, so as to only reveal a credential in the requested subject rather than revealing credentials in different subjects, as the latter implicitly reveals more information. This cannot be done by Indy or Aries as it depends on an external and variable component, the subject ontology. Doing this would require a third-party agent that issues credentials to users based on credentials they already hold. This agent needs to be trusted by users as it becomes the source of truth, and it would be responsible for revoking credentials when a user's held credentials, or the subject ontology, change. This third-party agent becomes a central authority, which weakens the "institution-independent" aspect of e-Ijaza.

Instead, users are able to choose whether or not to reveal credentials for each request. And to automate it, they can set some credentials as *public*. If authority in a subject can be proved using only public credentials, then the application will automatically respond to the request. Indy and ACA-Py make it possible to reveal multiple credentials in a single proof (section 2.3.2.5).

[13] https://github.com/bcgov/indy-tails-server

2.6 Subject Ontology

The subject ontology consists of connections between subjects. There are two types of connections. First, a subject can be the child of another, such that authority in a subject implies authority in its children. Second there are component sets, which work in the opposite direction, such that authority in all the subjects in the set implies authority in the parent subject. A subject's component sets must only consist of its descendants (i.e. its children, children's children etc).

The ontology starts with a root subject, and adjusting it only involves creating and removing connections. Subjects are created by making them a child of an existing subject, and they are deleted if they become unreachable from the root subject.

The subject ontology forms a directed graph, and operations on it are just reachability tests. We start with a set of subjects (such as the set of held credentials) and see if we can reach some goal subject(s). The standard algorithms for such operations are breadth-first search (BFS) and depth-first search (DFS). Both algorithms prevent cycles by not searching from nodes that are previously found, and can store the paths to nodes as they find them.

Both algorithms can be adapted to account for component set connections. This requires some extra state which keeps track of which subjects in each component set have been found. Only when all the subjects in the set have been found can the component set connection in the graph be used.

Standard BFS always finds nodes along the shortest path, since each iteration searches the paths one longer than those of the previous iteration. This is not true when using component sets, as multiple paths are combined to reach a single node. However, it can be useful to still reach subjects via the shortest path, as such paths rely on the least number of connections between subjects. To still reach via the shortest path, the use of component sets has to be delayed until all other paths up to the same length have been searched, which is essentially switching a first-in-first-out queue for a priority queue (while DFS uses a stack). This could be useful in certain circumstances, but it could take longer.

2.7 Voting

As mentioned in the requirements analysis (section 2.2), e-Ijaza allows proposals for changes to the set of master credentials and the subject ontology. The original intention had been that users could vote on any proposal if the subjects of their held credentials could reach all the subjects involved in the proposal. This can be abused if someone who can vote on a proposal creates additional agents, issues them credentials, and votes using them, effectively getting multiple votes. The solution is to decide if a user can vote using only master credentials, as this is far harder to abuse.

ICs can be used to represent a user's right to vote on a proposal. They can be issued to allow a user to vote, and revoked once they have voted. I call these *vote credentials*.

2.8 Storing State

2.8.1 Types of State

Indy and Aries provide most of the key functionalities necessary for e-Ijaza, but one thing missing is the storing of application state. There are three types of state in e-Ijaza:

1. That which is stored automatically by Aries, such as ICs.
2. That which is not stored automatically but is integral to the system, such as the subject ontology. This data is also shared amongst the users.
3. That which is not integral to the system, such as active proof requests, and is acceptable to lose in rare situations, such as system crashes.

2.8.2 Storing in a Database

One option is to use some form of database. If this is per-user, then care needs to be taken to keep it in sync with the Aries agent and the rest of the system, and a solution needs to be found to share data between users. If this is a shared database, such as one storing the subject ontology, then access needs to be controlled and it needs to update users when there is a change. This requires another application to manage the database. Either way it is extra work outside of the core purpose of the project.

2.8.3 Storing in Memory

Data could be stored in memory, or on a local hard drive by the application. This is simple and easy, but not resilient, and a solution needs to be found to share data between users. If data integral to e-Ijaza is stored like this, then a memory failure could bring down the entire system.

2.8.4 Storing as ICs

Data could be stored in ICs. It could be stored by issuing an IC, and edited by revoking and re-issuing, with revocation notifications alerting users of changes. It can be stored on an agent by issuing it to itself, and deleting old data rather than revoking. This requires one application to be responsible for the data, and if its ICs are compromised the data could be lost. ACA-Py can be backed up and recovered[14], but it is not covered in this project.

2.8.5 Storage in e-Ijaza

I opted to use ICs to store data that is integral to the system. The subject ontology and set of masters are shared as ICs, and proposal data (section 2.7) is included in the vote credentials. Other data is stored in memory. Using ICs allowed more time to be focused on learning and understanding Indy/Aries.

2.9 Controller Application

E-Ijaza requires an application to manage the state. This includes managing proposals/votes (section 2.7) and storing and sharing the set of master credentials and the subject ontology as ICs (section 2.8). It is also needed to issue and revoke master credentials. I will call this application the *controller*.

2.10 UI and Server Frameworks

The application has 2 parts, the server which contains the logic and communicates with ACA-Py, and the web app which is the interface. Given my experience (section 2.1) I chose to use Angular for the web app and ExpressJS in NodeJS for the server, both written in typescript.

I decided to use Angular Universal, which is a library for server-side rendering. This is definitely not necessary for the project, but it was convenient as it created an ExpressJS server that automatically included serving the web app, and meant the entire application is compiled by the Angular compiler. It also allowed sharing code easily between the server and the web app. Making a separate server and web app is trivial, but this approach was convenient for development.

2.11 Development Model

I used a spiral development model for this project. This allowed me to develop different parts of the application at different stages, and to start with simple working versions and then increase the complexity and features. Being able to redesign was key whilst exploring features, especially when they did not work as expected.

[14] https://github.com/hyperledger/aries-cloudagent-python/issues/575

This does mean that the line between preparation and implementation was somewhat blurred, as some aspects of preparation were done after aspects of the implementation had been completed.

3 Implementation

3.1 Techniques, Libraries and Tools

The use of VON Network, ACA-Py, Indy Tails Server, ExpressJS, NodeJS, typescript and Angular have already been mentioned in the Preparation Chapter.

3.1.1 Angular

I used a number of built-in Angular libraries, including Angular CLI (for generating boilerplate code and compilation), Angular Universal (section 2.10), Angular Forms (for creating and controlling forms) and Angular HTTPClient.

Angular uses a *Model-View-Controller* pattern. Angular Components consist of HTML, CSS and a typescript class, with the HTML and CSS forming the view, and the class being the controller (controlling the view). Angular Services are singleton classes shared between components and act as the model, containing data used by the controllers and methods to handle user actions.

I also used UI components from Angular Material[15] to build the interface.

3.1.2 RxJS (Observables)

Observables, provided by RxJS[16], are streams used for reactive and asynchronous programming. The streams can be passed through *operators* to produce new streams, encouraging a functional programming approach. The streams can be multicast (based on a single source), or unicast, where the stream is re-created and re-run for every subscription. I used Observables heavily in my project to handle both state and asynchronous actions, on both the server and the web app.

3.1.3 Object Oriented Programming (OOP)

OOP is a traditional programming paradigm for encapsulation. I used OOP throughout my code, by using Singleton classes (classes with a single shared instance) to encapsulate different aspects of the program, as well as, in some places. using classes to produce interacting objects to represent interacting components.

3.1.4 Axios

Axios[17] is a library I used for making HTTP calls on the server, particularly to the ACA-Py API.

3.1.5 Jupyter Notebooks

I used Jupyter[18] notebooks for writing and running python scripts, which I used to produce graphs from test results.

3.1.6 Docker

Von Network, ACA-Py and Indy Tails Server all run using Docker[19]. I packaged e-Ijaza as a Docker image based on ACA-Py's image. For the production image I used multi-stage builds, with one stage compiling the application and the second combining ACA-Py and NodeJS with the compiled

[15] https://material.angular.io/
[16] https://rxjs.dev/
[17] https://github.com/axios/axios
[18] https://jupyter.org/
[19] https://www.docker.com/

code (discarding the rest). The ACA-Py API is not exposed outside the Docker container. I use Docker volumes to mount folders in order to output logs.

I also produced a development image, which mounts the entire application codebase in a volume alongside ACA-Py, and uses the Angular CLI development compiler which updates on changes. This also exposes the ACA-Py Swagger UI (section 2.4.2.3) in order to manually access API calls..

3.1.8 Other tools

I used Git[20] to manage the e-Ijaza repository, and GitHub[21] to back it up. I used openapi-typescript[22] to produce types from the ACA-Py's OpenAPI specification (section 2.4.2.3), which I had to supplement with my own type definitions due to inaccuracies. I used ts-node[23] to run typescript files without compilation, eslint[24] for linting, webstorm[25] for programming, Google Docs[26] for writing the dissertation, and Code Blocks[27] and tree.nathanfriend.io[28] for formatting in the dissertation.

3.1.9 Licences

All the libraries and tools used in development are under either MIT or Apache 2.0 licences. Tools used to write and format the dissertation are freely available to use.

3.2 System Overview

Figure 3 (next page) shows the data flow in the overall system, and the key components involved. The Indy Network and Indy Tails Server are not involved in the work of the project but are essential to the system as a whole. The applications are the result of the project, and include ACA-Py. There can be multiple user applications, and multiple Indy Tails Servers (each used by one or more applications for revoking their issued ICs), but only one controller application and one Indy network. My development environment only runs one tail server for simplicity, but the use of them is managed by ACA-Py so using more would not affect e-Ijaza's code.

3.3 Application Overview

Figure 4 (next page) shows an overview of an e-Ijaza application. The Aries agent is ACA-Py, and the server and interface it is packaged with are the code produced by this project. The "Server-Side-Rendering" is produced by Angular Universal, which also handles the serving of the compiled web app files. I followed the practice of decoupling the client and server, so that the server does not affect the interface except through a HTTP API. The diagram also shows how the ACA-Py API is kept within the docker container and is not exposed. Communication with the rest of the system (other applications etc) is only through ACA-Py.

[20] https://git-scm.com/
[21] https://github.com/
[22] https://github.com/drwpow/openapi-typescript
[23] https://typestrong.org/ts-node/
[24] https://eslint.org/
[25] https://www.jetbrains.com/webstorm/
[26] https://www.google.co.uk/docs/about/
[27] https://workspace.google.com/marketplace/app/code_blocks/100740430168
[28] https://tree.nathanfriend.io/

Figure 3 - Full System Overview

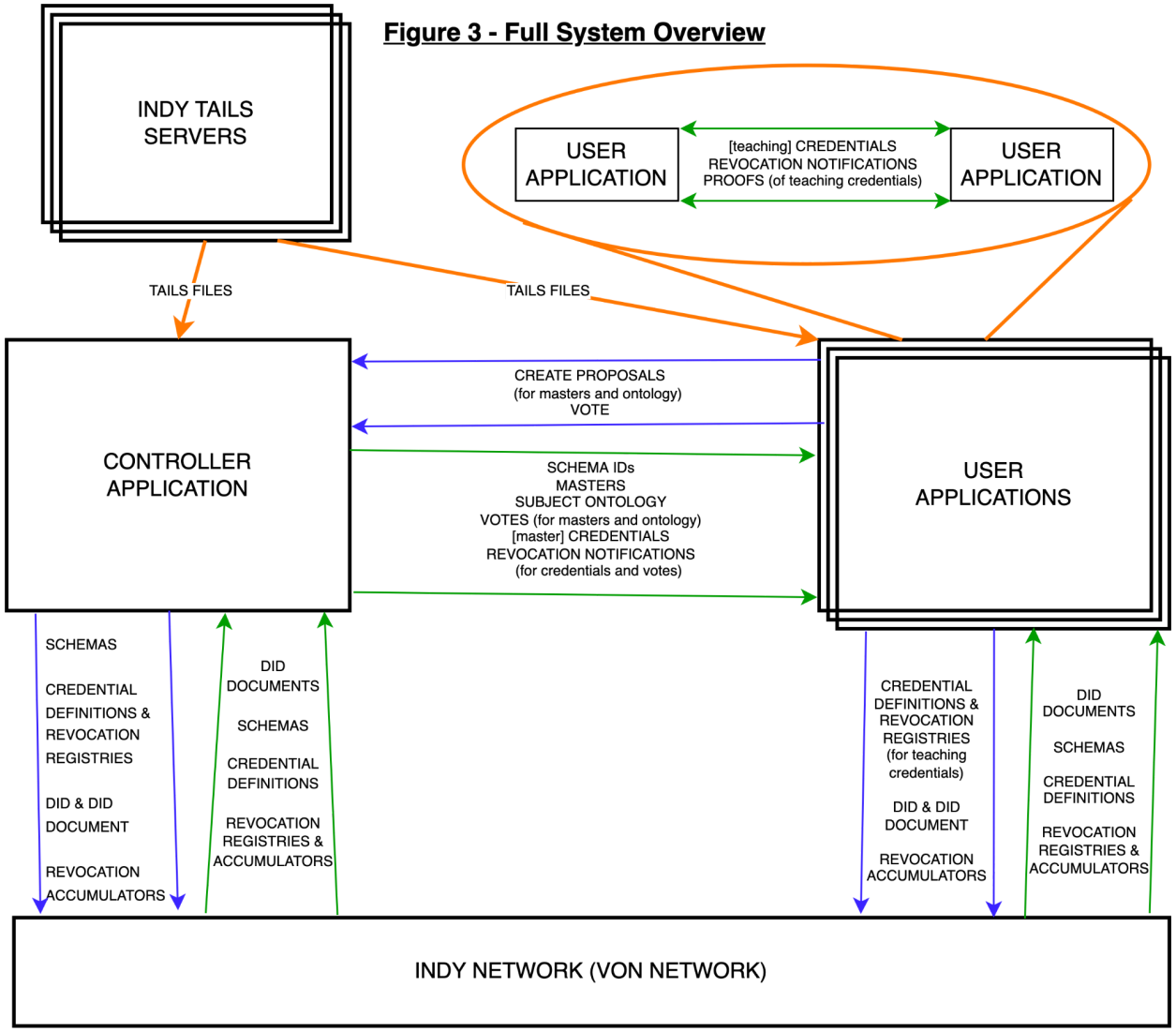
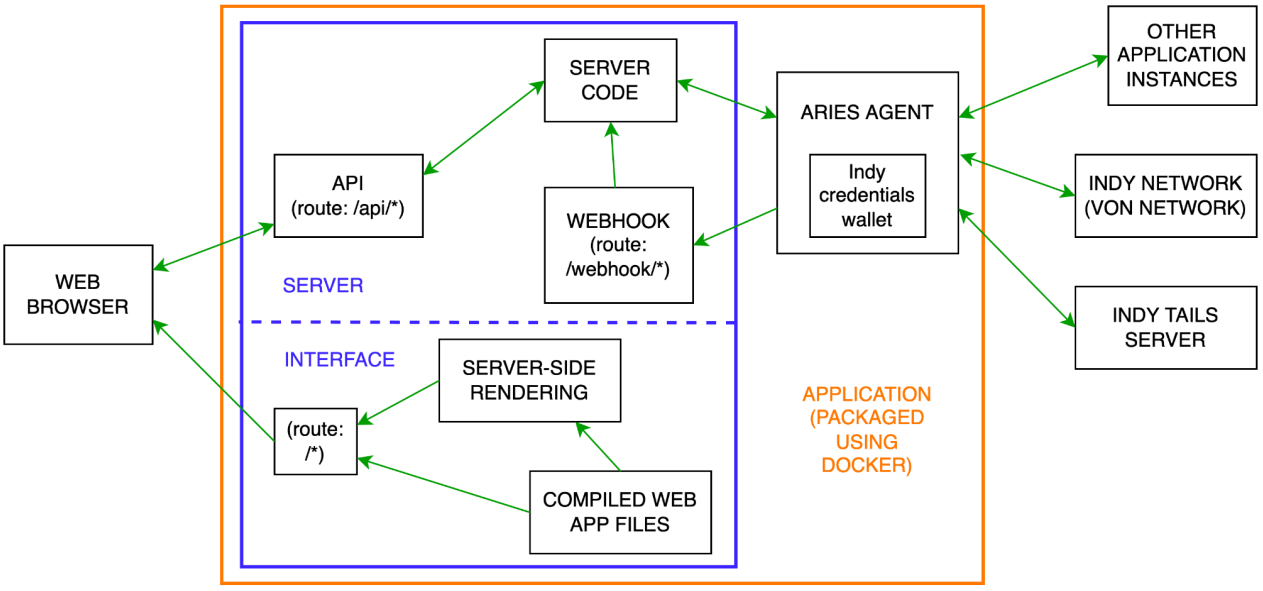


Figure 4 - Application Overview

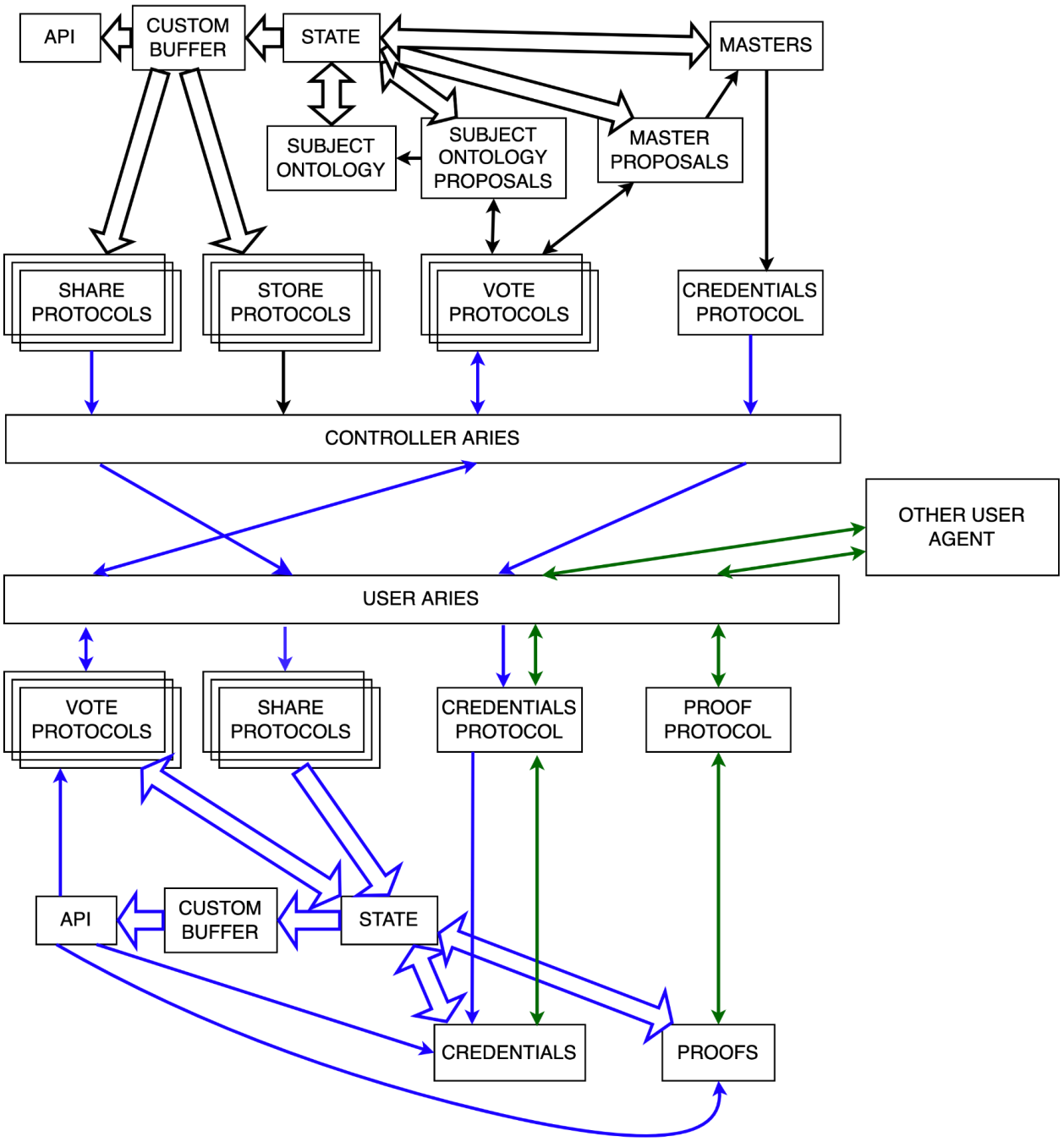


3.4 Controller/User Overview and Interactions

3.4.1 Diagram

Figure 5 shows an overview of the structure of both the controller application and user applications, as well as showing how they interact. The black arrows represent data transfer related to the controller, blue relates to the user, and green relates to transfers between users. The larger, outlined arrows represent observable streams (section 3.1.2). Sections 3.4.2 and 3.4.3 explain the core aspects of the diagram, the e-Ijaza protocols, and the state management.

Figure 5 - Controller/User Overview and Interactions (Data Flow)



3.4.2 Protocols

All interactions are through ACA-Py, and the diagram shows how different interactions are encapsulated with different e-Ijaza protocols (which are built from Aries protocols).

The Share Protocols share the set of master credentials and the subject ontology using ICs (section 2.8.4). With the subject ontology, there is one IC for the list of subjects, and then one for each of the subjects with their data (their children and component sets). This is done to prevent the data in one IC getting too large, however it means the user could have an inconsistent state whilst waiting for ICs, so the Share Protocols do not expose updated state (on the user) unless it is consistent.

The Store Protocols are used by the controller to store the set of master credentials and the subject ontology using ICs (section 2.8.4). To do this it first establishes a connection with itself via off-ledger invitations (section 2.4.3).

The Vote Protocols are used by users to send new proposals and votes, and by the controller to issue and revoke vote credentials. Voting is described in section 2.7. The API on the user directly calls the protocol to vote or create proposals. The protocol on the user exposes the current votes (and the associated proposals) to the state, and gets the held credentials' state. Creating proposals involves a proof with a self-attested attribute of the proposal, and the controller requesting a second proof for any master credential. Voting on a proposal involves a single proof with the vote credential, and a self-attested attribute representing the user's vote. The user side does not delete vote credentials once they are used, but waits for the revocation notification, so the controller side must check to ensure vote credentials are not used multiple times.

The Credentials Protocol is used by the controller to issue and revoke master credentials. It is used by the user to receive credentials (and revocation notifications), and issue (and revoke) credentials to other users.

The Proof Protocol is used to request credential proofs (section 2.5), and to receive and respond to credential proofs. It does not handle automatically responding to proofs. Credential proofs involve two stages. The first is the verifier requesting a proof for the target subject, and the prover self-attesting a list of credentials they hold, by their subject and issuer public DIDs, which prove the target subject. The prover could also reject the proof at this stage. The second stage involves the verifier requesting the credentials with the specified issuers DIDs. The prover waits for this proof request after responding to the first, and then automatically responds. If the proof is valid, then the verifier knows the prover is authorised in the subject if the provided credentials have valid chains-of-trust.

3.4.3 Global State Structure

Figure 5 also shows the use of a global state structure, where different *state components* encapsulate aspects of the state and expose them as streams to the global state, and the rest of the application access the streams through the global state. The state components handle actions on their states (such as creating a master credential), and issuing and revoking ICs (such as master credentials). They also encapsulate some actions related to their state, such as responding to a credential proof request. But state is never read from state components directly. The purpose of a custom buffer is explained in section 3.5.1.

3.5 Application State

3.5.1 State Components

Figure 6 - Application State Component

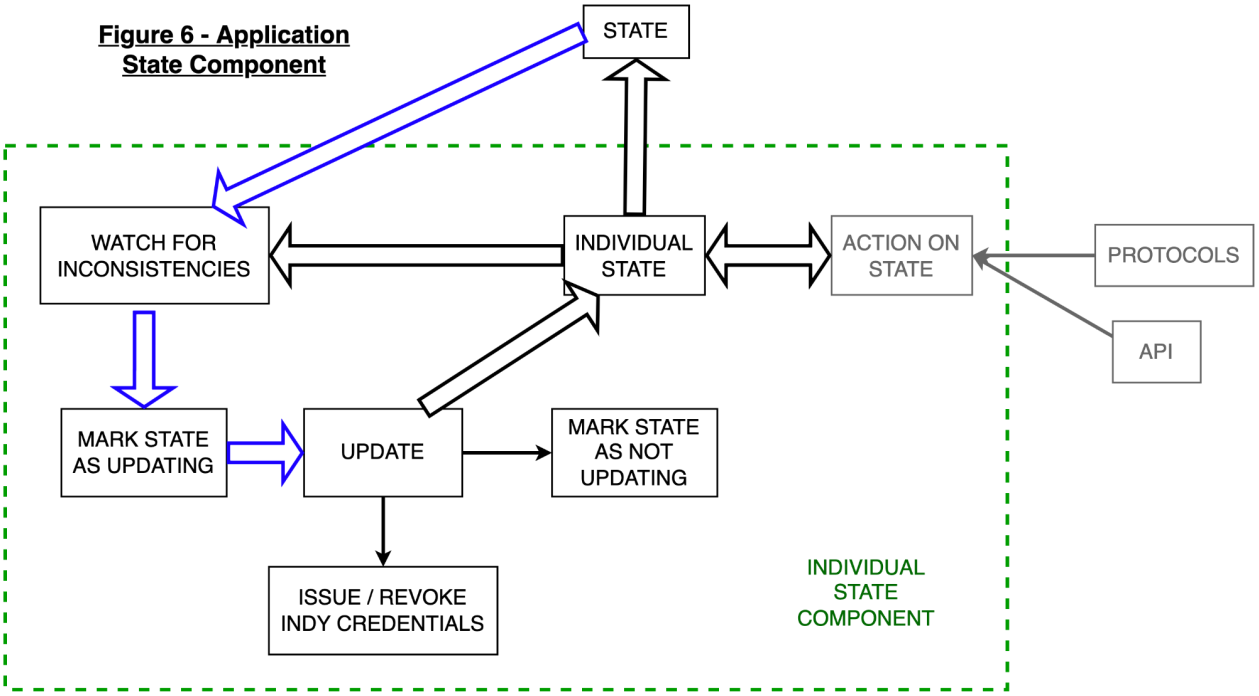
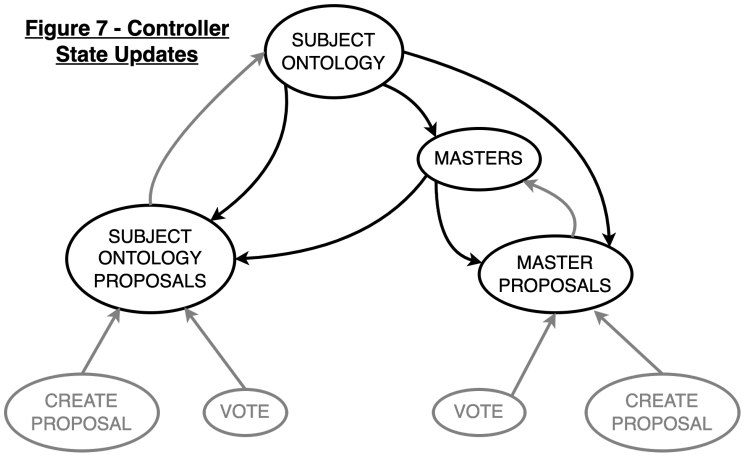


Figure 6 shows the workings of a state component. The black outlined arrows represent the state's observable, and the blue arrows represent it merged with other states' observables. State components watch for inconsistencies related to their own state. When there is an inconsistency, they update their state. When updating, a state component holds a shared mutex from the global state. The global state only exposes an updated state when the mutex has not been held by any state component for a certain amount of time, to ensure the rest of the application only sees a consistent state. The mutex must be not held for some time to allow state components to notice an inconsistency. This buffering of updates on the global state is done using the custom buffer shown in Figure 5 (section 3.4.1).

All work on the state is done on the update cycle. Actions on the state make minimal changes. For example, adding a new master credential simply adds the public DID and subject of the credential, and the actual issuing of the credential is done on the next update cycle. As such state components notice inconsistencies in their state with regards to other states (e.g. a master credential in a no longer existing subject), and inconsistencies in the state itself.

Figure 7 - Controller State Updates



3.5.2 Controller State

Figure 7 shows the controller's state, with each arrow showing a potential update, and grey arrows signifying external causes (i.e. not caused by an inconsistency).

A change in the subject ontology can cause changes due to a deleted subject in use elsewhere.

A change to the masters can cause a change if it changes the set of valid voters on a proposal.

A change in the masters/subject ontology can cause a change in proposals if they change the set of valid voters on a proposal, or if they make a proposal impossible.

The proposals can cause changes if a proposal passes, causing it to be actioned on the respective data.

The diagram includes cycles, however there is not a risk of endlessly updating, since all cycles go through a proposal state component. Proposals can only trigger changes if they complete, which means every cycle would reduce the number of proposals, so the cycle has to end eventually.

One thing missing is that the controller can create a master credential if none exist, as without a single master credential e-Ijaza cannot do anything. Also, the controller ensures the root subject ("knowledge") always exists, for a similar reason (subjects are deleted if they are not reachable from the root subject).

3.5.3 User State

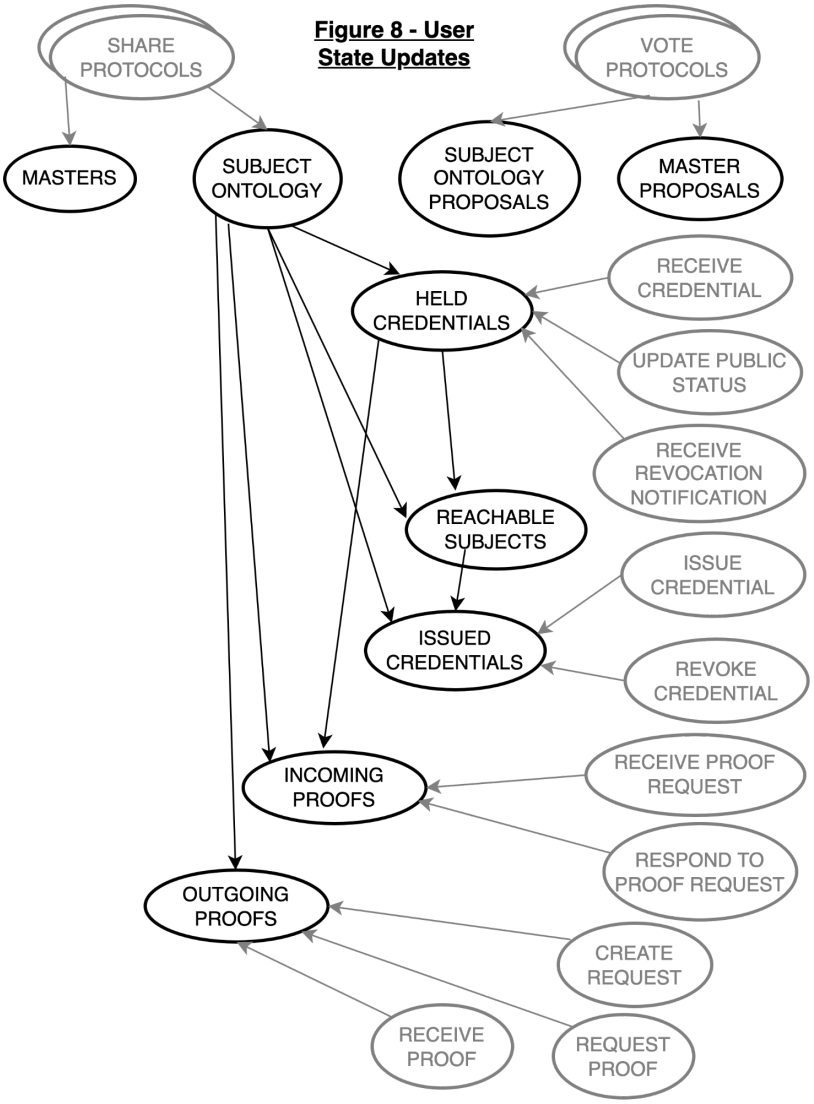


Figure 8 - User State Updates

Figure 8 shows a user's state, in a similar manner to Figure 7 above. Held Credentials, Reachable Subjects, and Issued Credentials are all maintained in one state component, as are Incoming Proofs and Outgoing Proofs. This was done due to how closely linked they are, but it would have been simple to have a separate state component for each one.

The proposal state components contain the votes issued to the user.

Some points to note. Nothing causes changes to Masters, Subject Ontology, or either of the Proposals. And creating proposals or voting does not change any state of the user. The reasons are mentioned in 3.4.2, which essentially is because the changes are triggered by the controller.

Also, the set of masters does not affect the held credentials, or the proofs. Both aspects use the credentials directly, and identify credentials as master credentials due to the issuer's public DID being the controller's.

The actions on the state include updating the public status of held credentials. This is used as described in section 2.5, to determine which credentials can be used to automatically respond to proof requests. This aspect of the state is stored in local memory, so on start up it is assumed all credentials are private. This is useful in order to prevent automatically responding to proofs on start

up, especially if recovering from a crash. Setting the public status also affects proof requests that have already been received, as changes to the held credentials can cause changes to the incoming proofs, by either triggering an automatic response or changing the credentials the user is asked to expose in the proof if automatic response is not possible.

3.5.4 API/UI State

Figure 9 shows how the web app updates its state using the API. The web app never changes its own state, even after making a state changing API call, but rather checks for changes to the state from the server. It works by sending the server a timestamp. The server caches the latest states with the timestamps of when they were updated, and replies with booleans representing what has changed, and the time on the server (as a timestamp). The web app can then request the individual states' data with separate API calls. The timestamp the web app sends to the server is always the previous one returned from the server, and in the first call it uses 0. Since all the timestamps are generated on the server there is no issue with time differences or clock skew.

The web app also periodically requests state updates. Figure 10 is a flow chart of how that works. It uses a client-side generated timestamp to track the time between updates, and waits for any current updates to finish before checking if it should request another update.

Figure 9 - Web App State Management

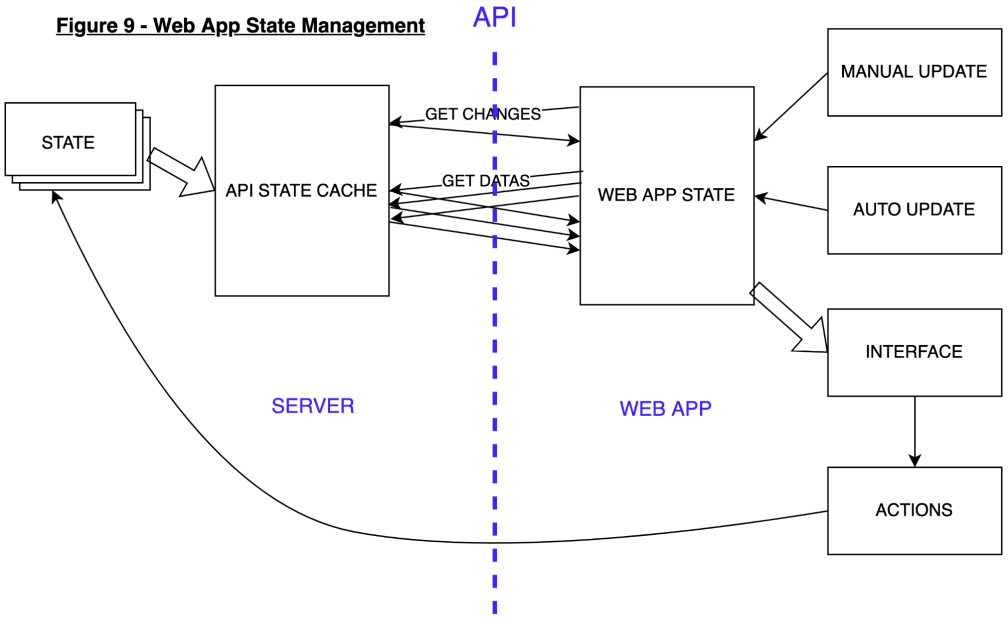
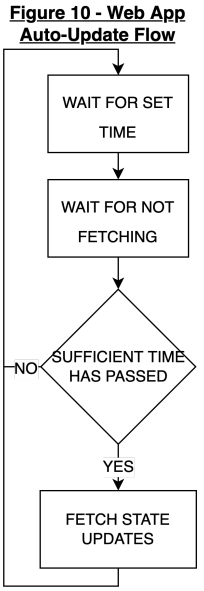


Figure 10 - Web App Auto-Update Flow



Manually triggered updates also do not occur if an update request is taking place. However, they queue a maximum of one update request after the currently active one. This is because a manual update is used when expecting a change after a user action, and the currently active update request may have been triggered before the expected change.

3.6 Application Initialisation

Figure 11 shows the process of initialisation. Initialisation is manually triggered after start up. This could have been part of the start up as command line options or something similar. But being able to start up without the application connecting back to the network and initialising can be useful when recovering from a crash. Also initialising after start up is more convenient for complex initialisation.

Initialisation includes determining if the application is a controller or user, since I programmed both functionalities into a single application. This was for simplicity, and ease of code reuse. It could be useful to have two separate applications using libraries to share code, especially if the codebase gets

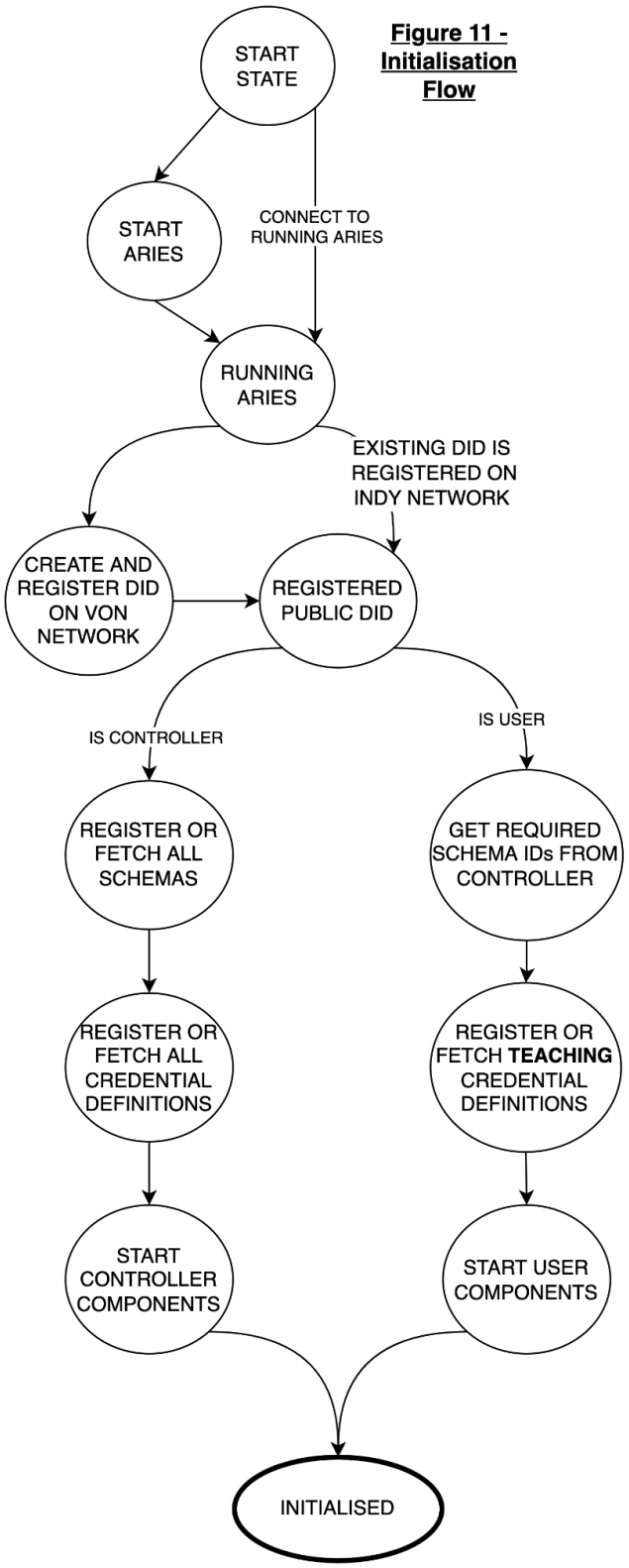


Figure 11 - Initialisation Flow

very large due to containing functionality for both types of applications. However, this is not an issue in this project.

Public DID registration involves creating a DID on ACA-Py, registering it on the Indy network, and then registering it as the public DID on ACA-Py. On production deployments of Indy DID registration is often manual and involves legal agreements, but VON Network is designed for development and allows registration through a HTTP call. E-Ijaza allows for manual registration, and does automatic registration for VON Networks.

Controller initialisation involves registering schemas and credential definitions, getting stored data, initialising data, and setting up listeners for incoming requests. User initialisation involves requesting data and schema IDs from the controller, registering a credential definition for e-Ijaza credentials, and setting up listeners. Both initialisations involve starting long-lives subscriptions to observables, such as watching webhooks, or state components watching the state for inconsistencies.

An initialisation state is exposed as part of the application state. It has 7 states; start, connecting to Aries, Aries ready, registering public did, public did registered, initialising and complete. It is self-explanatory where these states correlate to in the diagram. The reason for the states such as “connecting to Aries” is to ensure that the application does not duplicate work, such as trying to create the Aries agent twice at the same time. The initialisation state also exposes other data to the application state, such as the controller’s public DID.

Some initialization could be done lazily, like creating credential definitions when first issuing. However, performance is more consistent if the work is done at start-up, especially when some work takes a long time (such as creating credential definitions). It is also a useful assumption in the rest of the code that the initialisation is done.

3.7 Credential Proof Requests

The protocol for proof requests is mentioned in section 3.4.2, and the use of automated responses is mentioned in sections 3.5.3 and 2.5. The requesting of proofs, particularly in following the chain-of-trust, is managed in an OOP style. A class called `CredentialProof` is used, and its instances represent proof requests. It maintains proof state using the recursive interface below:

```
export interface Proof {
 did: string
 subject: string
 result: boolean | null
 proof: Proof[] | boolean | null
}
```

The `Proof` object starts with the original proof request, with `result` and `proof` set to null. It then makes the request using the proof protocol. If it is accepted then `proof` is set to a list of `Proof` objects representing the credentials (where `did` is the issuer's public DID, `subject` is the subject, and `result` and `proof` are `null`). And then the same algorithm is repeated on the outstanding `Proof`s (i.e. the received credentials). This then traces and verifies the chain-of-trust from the original proof request.

If a provided credential is a master credential, the `result` and `proof` attributes are set to true without any request. If a proof request is rejected, its `result` and `proof` attributes are set to false. The `result` attributes are updated according to their relative `proof` attribute. If any of the `Proof` objects in the `proof` attribute have `result` set to false, then the parent object's `result` attribute is set to false. And if all the `Proof` objects in the `proof` attribute have `result` set to true, then the parent object's `result` attribute is set to true.

When the root `Proof` object's `result` attribute is set to true or false, the proof request completes. It also completes if the proof request is cancelled/deleted.

`CredentialProof` instances encapsulate this functionality. The expose an observable stream that emits with the latest changes to the root `Proof` object. The functionality also ensures not to repeat the same proof requests. If this is not done, malicious actors could produce a circular chain for a credential that would never complete.

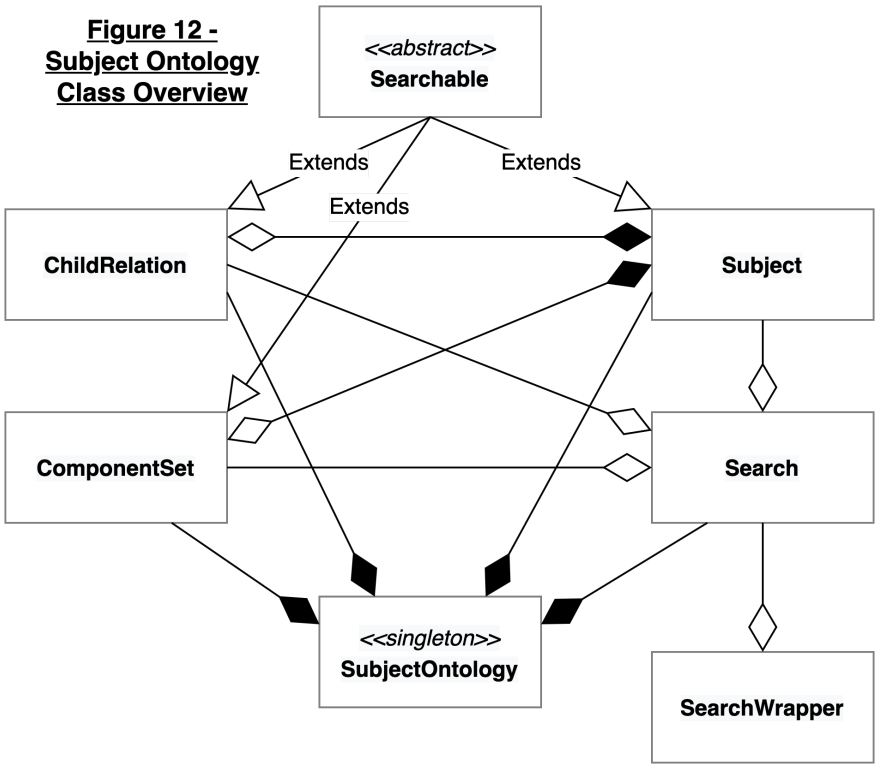
3.8 Subject Ontology

The subject ontology and operations on it are described in section 2.6. The subject ontology representation in the state components is as data, with a mapping between subjects and their children and component sets. This is not easy to run reachability tests on. Instead, I have implemented a parallel OOP style representation. A singleton class acts as the entry point to the representation. Each subject and each connection are represented as instances of classes that interact. Figure 12 is a simple class diagram of the representation, with a full diagram in Appendix II.

Multiple searches can be run on the ontology at the same time. The searches begin with a set of starting subjects, a set of ignored subjects/connections, a set of goal subjects. I use BFS, and each search includes a flag on whether the search queue should be first-in-first-out, or a priority queue (with the latter possibly taking longer but ensuring only shortest paths are found). Each subject/connection maintains a path to it per active search which is set once when it is reached. When a subject/connection is reached it adds the subjects/connections that are reachable from it to the search queue, if they have not been reached already. The search stops when either all the goal subjects are reached (if the set is not empty), or when the search queue is empty.

Component set connections do not get added to the search queue when they are found in the same way as subjects and child connections. Instead, they maintain a state per search on which subjects they have been found from, and get added to the search queue only when they have been found from the entire set (i.e. the entire set has been reached).

Figure 12 - Subject Ontology Class Overview



The set of ignored subjects/connections in a search are never added to the search queue, and never have a path. This is used to run tests on the subject ontology if things were removed, without actually editing the ontology. This is used to check if proposals are valid, i.e. do not create an invalid subject ontology. It is also used to find the descendants of a subject, by ignoring all the component sets.

This OOP representation of the subject ontology is updated before the subject ontology state components emit updated data, so that all other aspects of the application can rely on them being in sync. It uses a custom shared mutex to allow multiple readers, but ensure there are no readers when updating.

3.9 Logging Data

As mentioned in section 3.1.6, the application docker containers mount directories to store log files. This includes the standard and error outputs from the application and from ACA-Py. The application also logs the timing of credential proof requests in csv files, both when proving and when verifying/requesting.

3.10 Repository Overview

3.10.1 Structure

E-Ijaza's repository is large, with over 10,000 lines, including a server, a web application, type definitions and utilities, scripts for building and running the application and development environment, and code for running tests (chapter 4). The repository structure below only shows core files and directories written by me, and excludes auto-generated code. A more detailed breakdown of lines of code (and how they were calculated) is given in Appendix III.

The repository was generated by Angular CLI, which was used to generate the web app boilerplate code for Angular Components and Services. Angular CLI was also used to generate boilerplate Components from Angular Material Components as well, primarily for the tables and the tree structure. Files that are primarily generated and edited by Angular CLI are excluded from the repository structure below, and the calculations of the lines of code.

```
.
├── scripts/... [5 files; 200 LO; bash,Dockerfiles]
├── src/
│   ├── app [50 files; 2438 LOC; HTML,SCSS,Typescript]/
│   │   ├── credentials/...
│   │   ├── initialisation/...
│   │   ├── masters/...
│   │   ├── nav-container/...
│   │   ├── proofs/...
│   │   ├── services/
│   │   │   ├── api/...
│   │   │   ├── loading/...
│   │   │   └── state/...
│   │   ├── subjects/...
│   │   └── app.component.{html,scss,ts}
│   ├── server [99 files; 6224 LOC; Typescript]/
│   │   ├── api/...
│   │   ├── aries-api/...
│   │   ├── aries-based-protocols/...
│   │   ├── credentials/...
│   │   ├── initialisation/...
│   │   ├── logging/...
│   │   ├── master-credentials/...
│   │   ├── schemas/...
│   │   ├── subject-ontology/...
│   │   ├── webhook/...
│   │   ├── state.ts
│   │   └── index.ts
│   ├── types [12 files; 468 LOC; Typescript]/
│   │   ├── aries/...
│   │   ├── interface-api/...
│   │   ├── server/...
│   │   ├── schemas.ts
│   │   └── index.ts
│   └── utils/... [7 files; 131 LOC; Typescript]
└── test/... [11 files; 753 LOC; Typescript,Python]
```

3.10.2 Aries

`/src/server/aries-api/` is a wrapper around the Aries APIs. It wraps them as promises to match typescript asynchronous programming, and can be easily converted into observables. `/src/types/aries/` contains types for Aries, primarily generated with openapi-typescript, supplemented by my own type definitions to correct errors.

`/src/server/aries-based-protocols/` contains singleton classes for all the e-Ijaza protocols (section 3.4.2). Each class contains both sides of a protocol, which makes development and bug fixing easier.

3.10.3 State Components

The server code contains singleton classes which manage state components. Each manager initialises the related protocols, watches the global state (`/src/server/state.ts`), and exposes an observable which is re-exported by the global state. The global state is also a singleton class.

3.10.4 Webhook

`/src/server/webhook` includes a singleton class for watching the webhooks. It exposes observables for each of the webhooks. It also provides methods for watching webhooks for specific communications, which completes when the communication completes.

3.10.5 Scripts

`/scripts` contains bash scripts for managing the VON Network, Indy Tails Server, and applications packaged in Docker in development and production form. It downloads the repos from GitHub into `/temp`. It also contains the Dockerfiles for the application Docker images, which are based on the ACA-Py images (built from the ACA-Py repo in `/temp`).

`/logs` contains the logs from the application Docker images. Applications run as part of the tests store their logs in `/test`, which contains the files for running the tests.

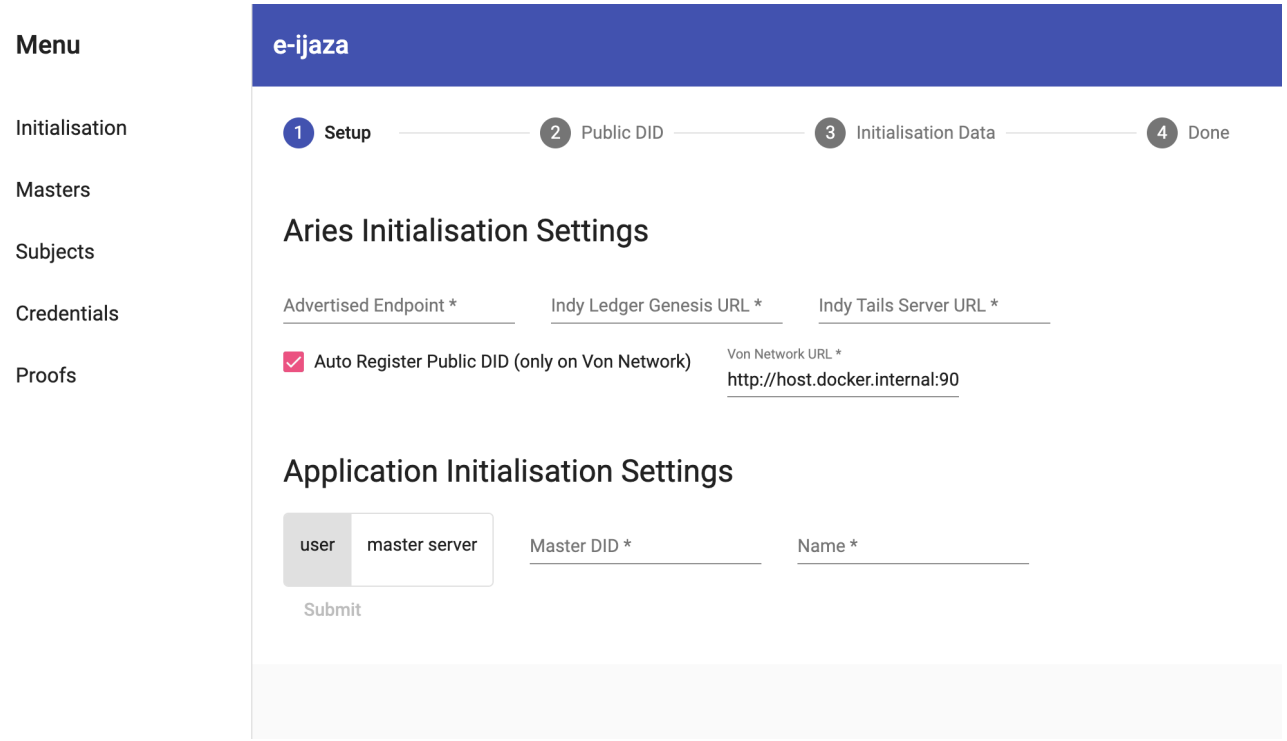
3.10.6 Utils

`/src/utils` contains utilities used across the application. This includes a number of simple things to avoid repeating code. It also includes a custom type utility that makes a given type immutable, changing sets to readonly sets etc, and changing object properties to readonly etc. it also includes implementations for a high-level shared mutex used in various places, including for the state (mentioned in section 3.5.1), and a shared read-write mutex used for the subject ontology OOP representation (section 3.8).

3.11 Interface

Below are screenshots from the user interface.

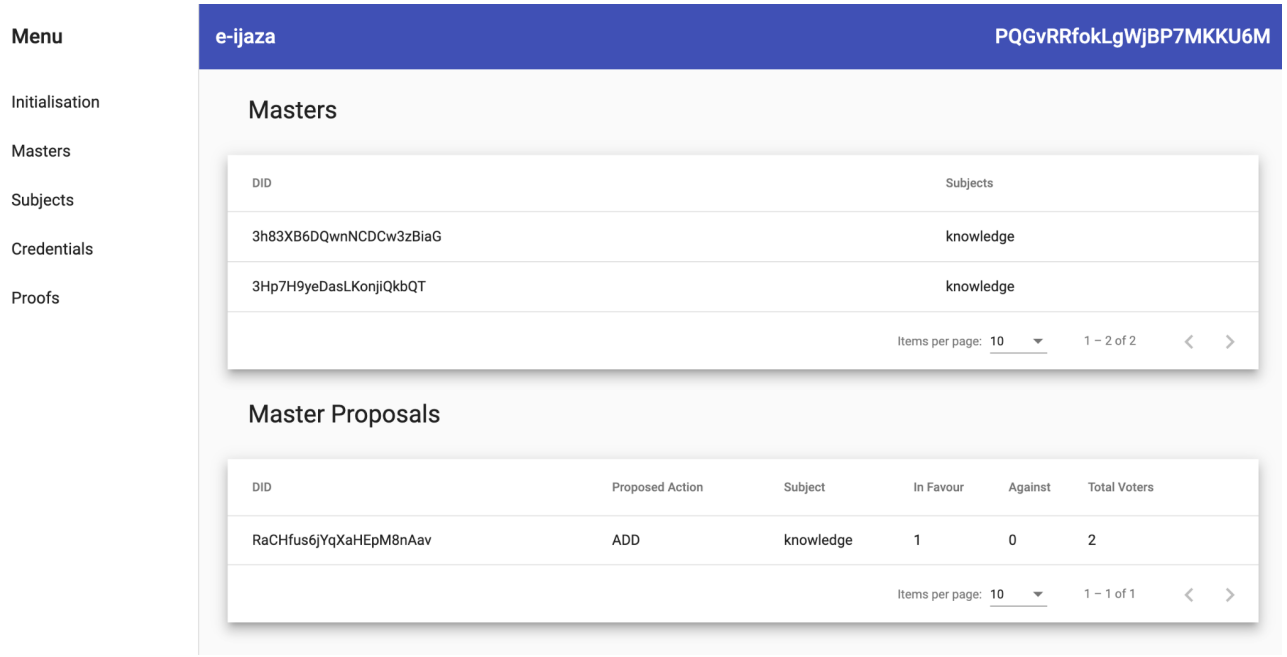
3.11.1 /initialisation



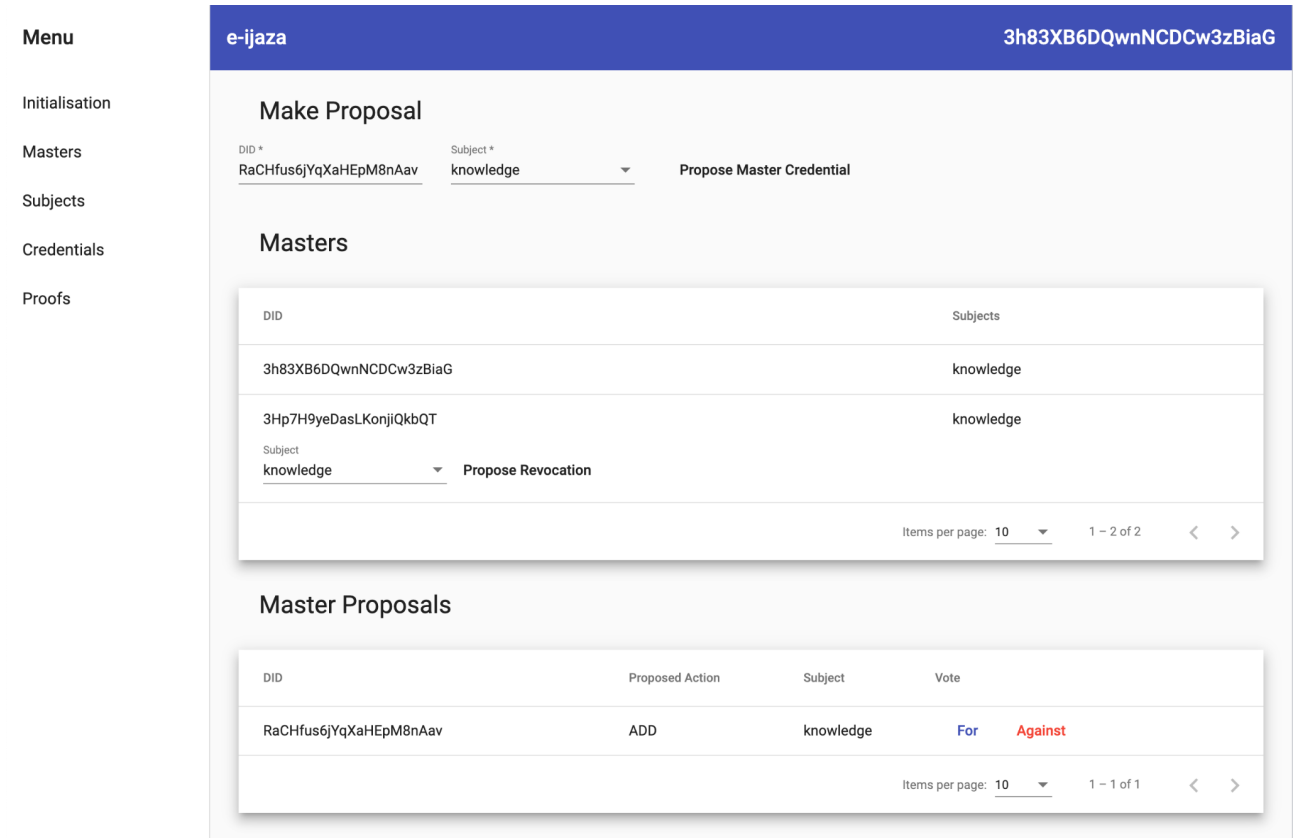
The initialisation page handles all initialisation paths, including recovering from partial initialisation. If everything is filled in on the Setup section and auto-registering the public DID is selected then none of the other sections will be used, rather it will just step through them.

3.11.2 /masters

Controller:

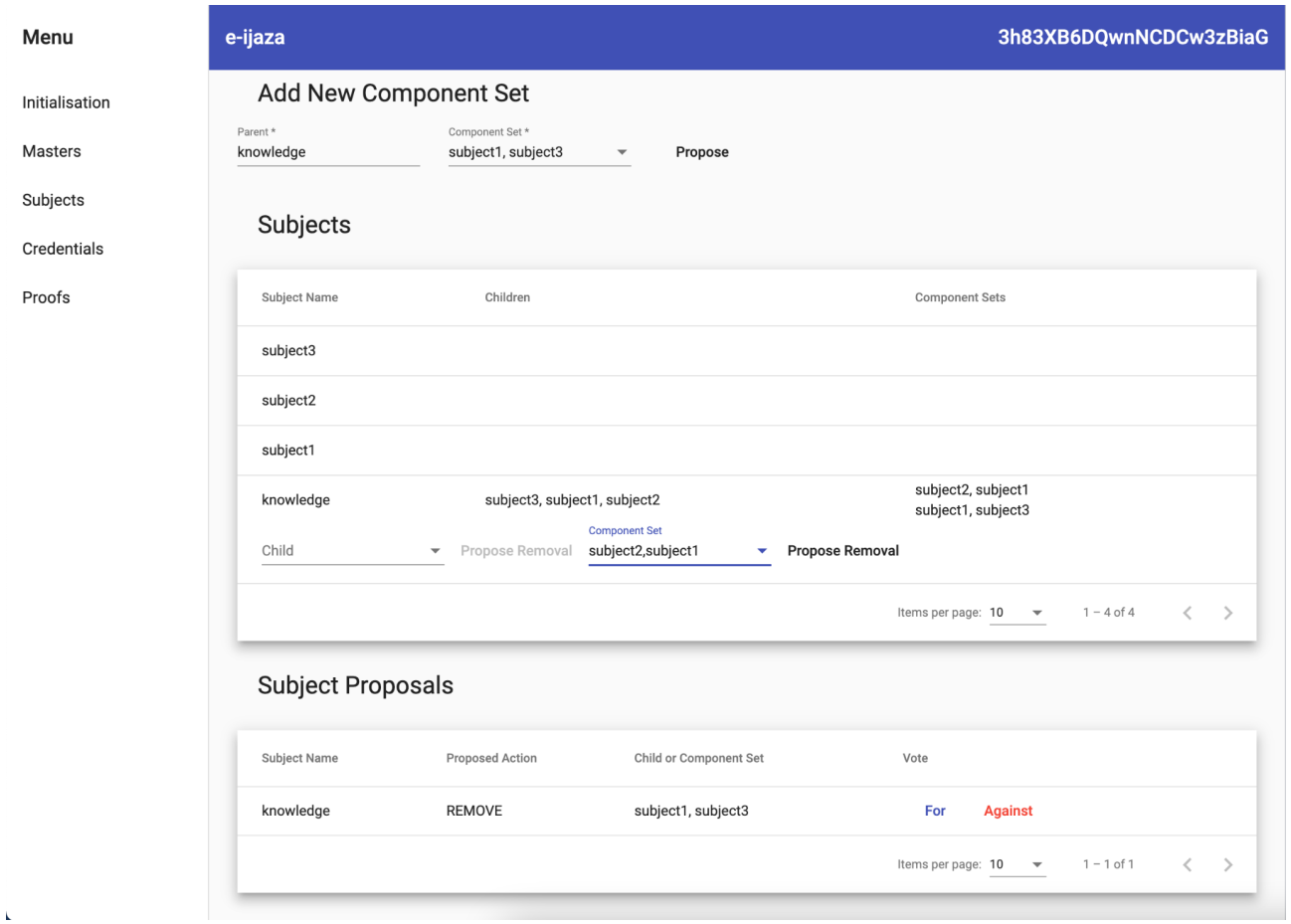


User:

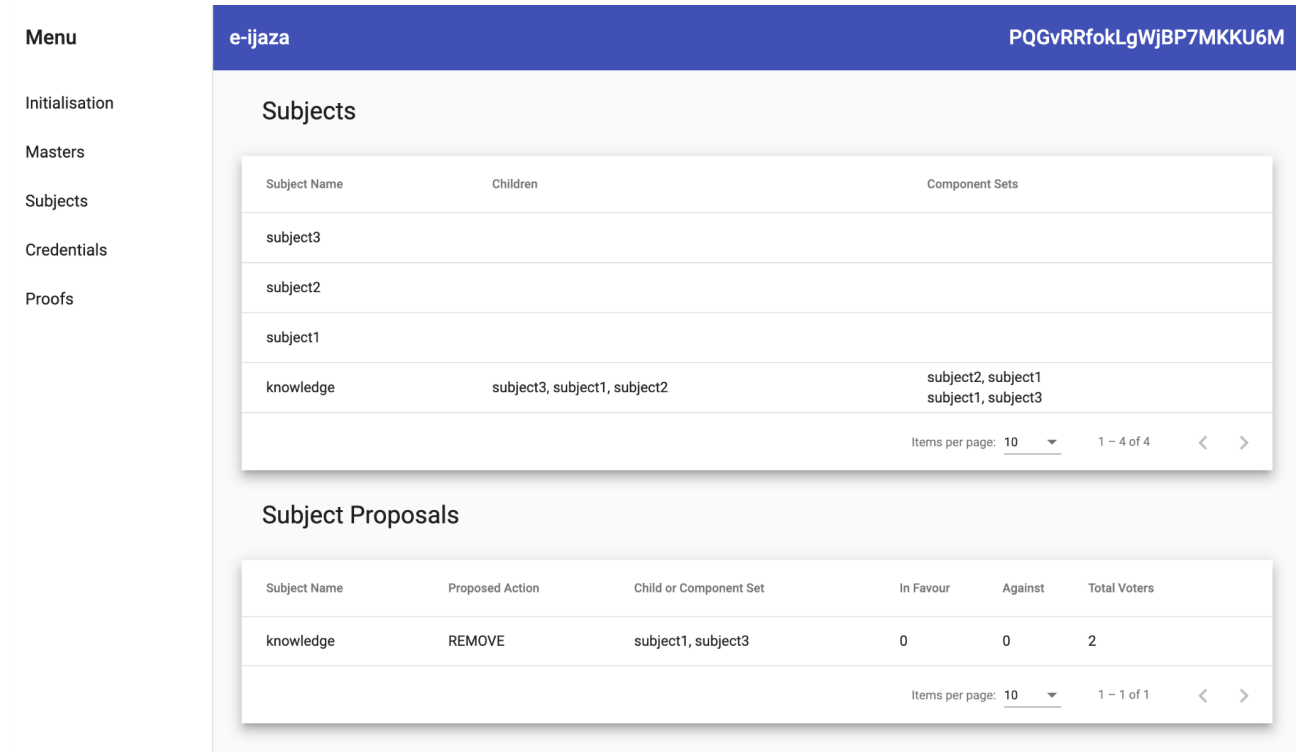


3.11.3 /subjects

User:

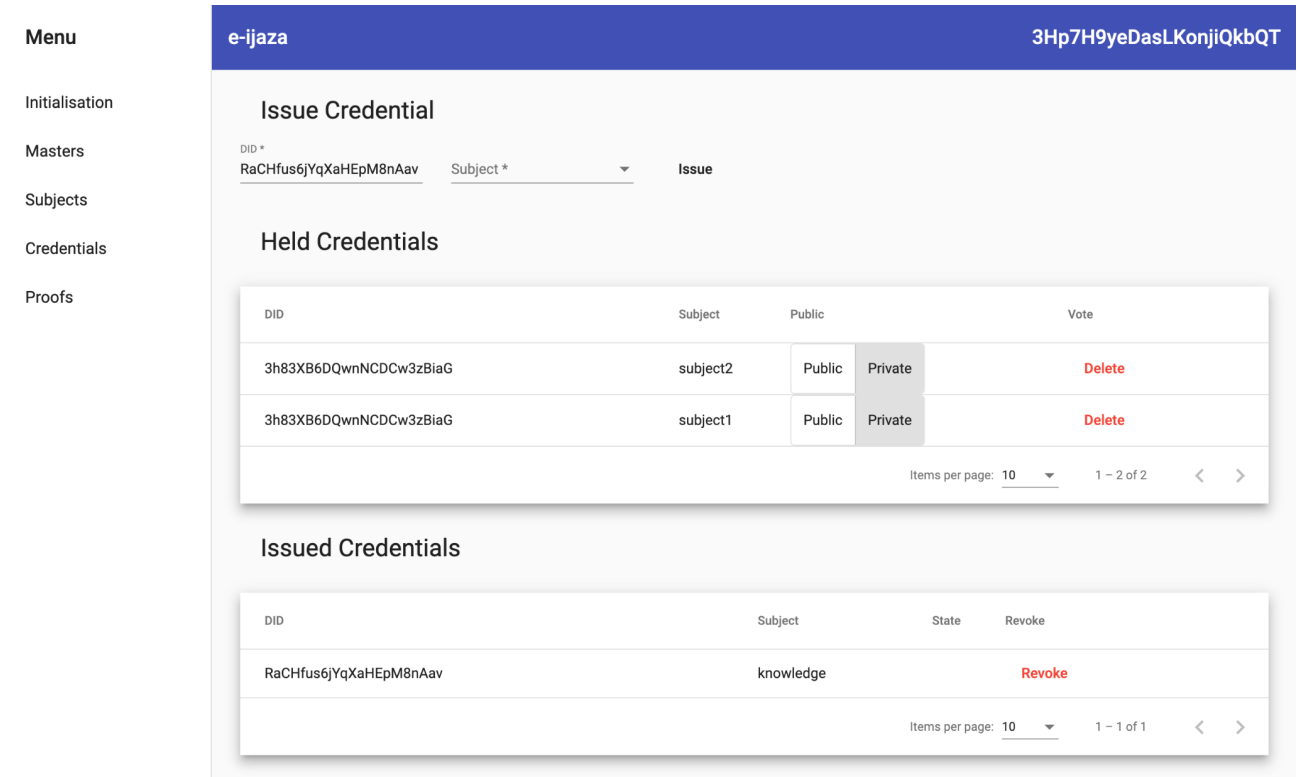


Controller:



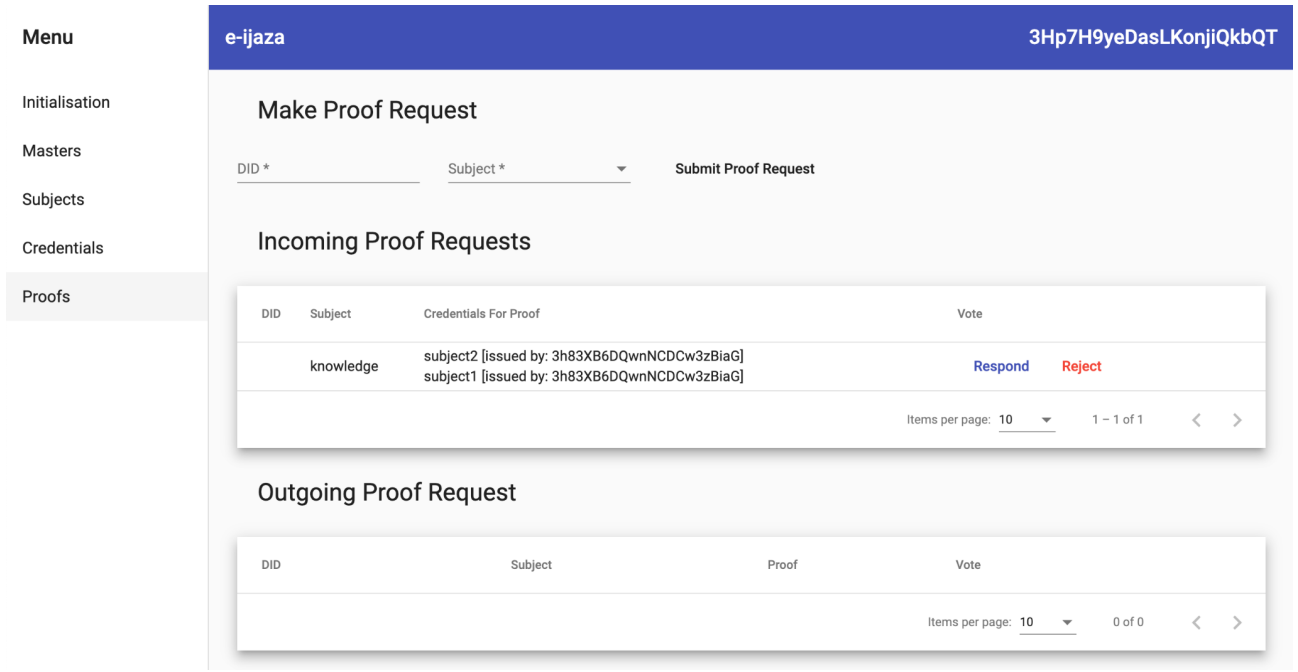
3.11.4 /credentials

User:

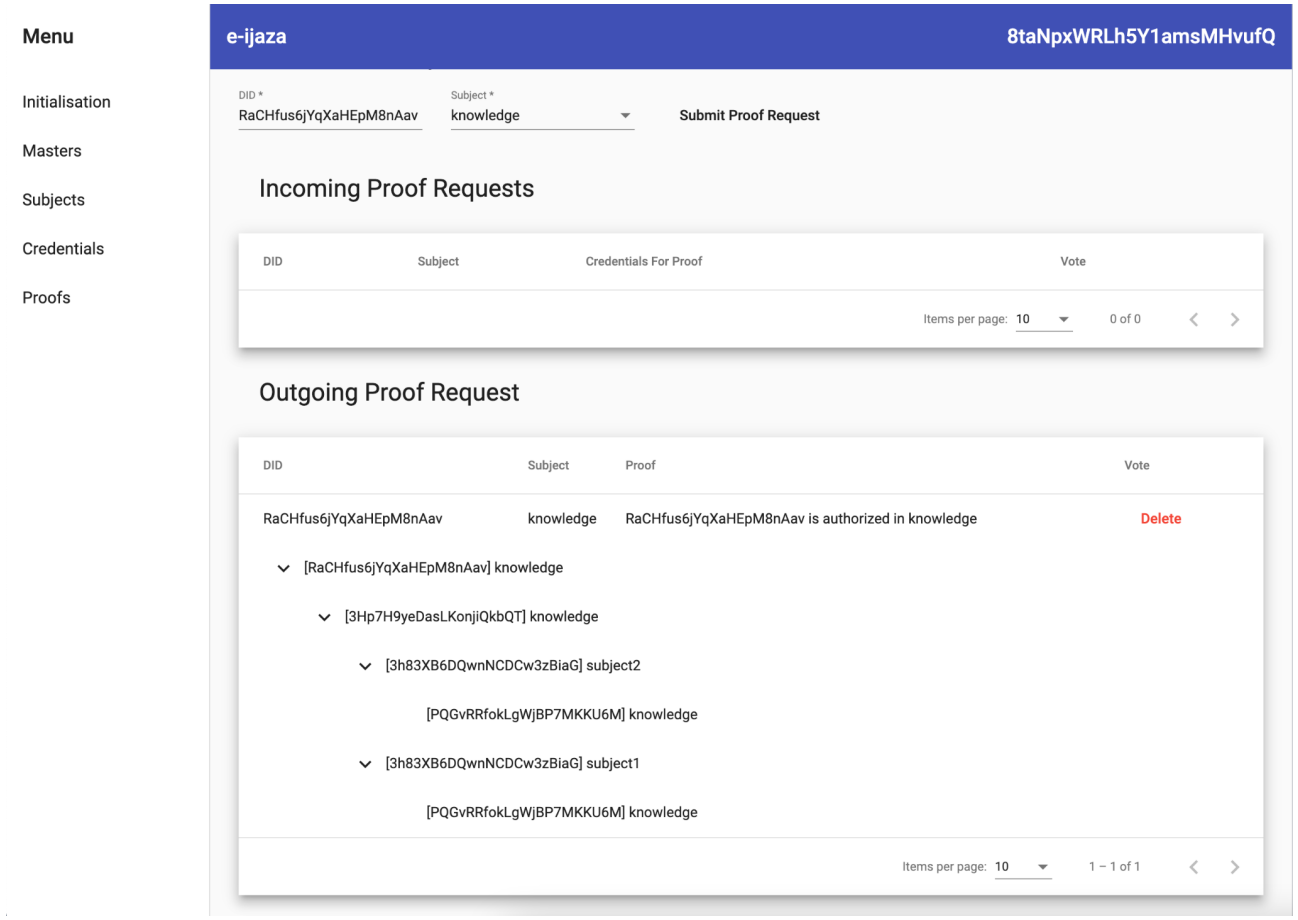


3.11.5 /proofs

Prover:



Verifier:



4 Evaluation

4.1 Test Process

For the evaluation I tested the performance of credential proofs when verifying chains-of-trust, using the timings logged on when proving and when verifying (section 3.9).

The data for the tests was in the format below:

```
interface TestData {
 users: string[]
 ontologyCommands: SubjectProposalData[]
 master: { name: string; subject: string }
 issueCreds: { issuer: string; receiver: string; subject: string }[]
 test: { user: string; subject: string }
}
```

The process for running a test is:

1. Create the applications (i.e. start the docker containers). This includes a controller application, an *ontology creator*, the users (from the test data), and 10 *verifiers*.
2. Initialise the controller.
3. Initialise the ontology creator (using the controller's public did).
4. Issue a master credential in the root subject to the ontology creator.
5. For each ontology command, have the ontology creator create the proposal, and then vote in favour of it. The commands are done in order.
6. Initialise the user referred to in the master attribute of the test data.
7. Issue a master credential from the ontology creator using the details from the master attribute of the test data. This involves first creating a proposal and then voting in favour of it.
8. Initialise the remaining applications.
9. Use a verifier to request a proof, using the details from the test attribute of the test data.
10. Repeat (9) with different numbers of verifiers. Use 1, 4, 7 and 10 verifiers.
11. Repeat 9 and 10 three times.

When running the test, the logs are written in `/test/logs`, and the directory is renamed to identify the exact test that produced the results. Also, the results of the proof requests are monitored, and each unique result is written to `/test/logs/proof_results.txt`. If e-Ijaza is functioning properly there should only be one unique proof result across all the verifiers and all the runs.

After running the tests, Docker is then purged. The VON Network is reconfigured to change the number of nodes, using the docs in the VON Network's GitHub repository, and then the test is re-run. This is done for von networks of 4, 7 and 10 nodes. This produces 6 sets of data, three for the Ijaza data and three for the custom data.

Following the style of batch processing, the code that runs the tests paused between operations during the setup of tests, and paused between repeats of the tests (and obviously I manually pausing between re-running the tests with different VON Network configurations as it involved manually adjusting the VON Network code).

4.2 Test Data

I used two test data sets, and ran the process described in section 4.1 on both of them. The first is an Ijaza chain from the dataset I got from Maan Al-Dabbagh as mentioned in the project proposal. The raw extract from the dataset, the translation and the extracted Ijaza chain are in Appendix IV. The

Ijaza chain has 19 people. It is all in one subject, as Ijaza does not have a concept of a shared subject ontology that allows issuing an Ijaza in something other than what you hold an Ijaza in.

The second dataset I used was a custom one, intended to run the tests using a subject ontology that used child connections and components sets. I used numbers for users and letters for subjects. Figure 13 shows the subject ontology, and Figure 14 shows the chain-of-trust for the credentials..

Figure 13 - Subject Ontology for Custom Test Data

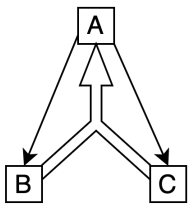
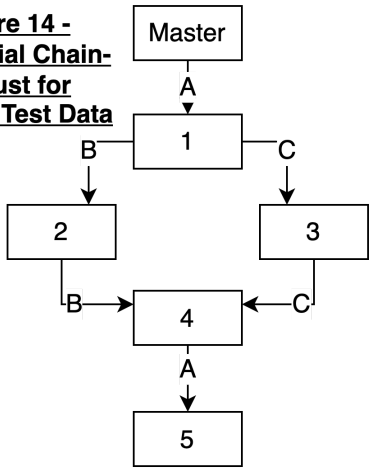


Figure 14 - Credential Chain-of-trust for Custom Test Data



This is a much smaller chain than the ijaza data, but it is sufficient to test the child connections and component set connections. Also, since I'm running the tests on my laptop, it could be useful to have some of the tests require less docker containers, in case computational limitations affect the results.

When running the tests, I set the docker resources to 12GB of memory, 8 CPU cores, and 120GB of storage. That is out of 16GB of memory, 12 CPU cores, and 512GB of storage.

4.3 Results

4.3.1 Correctness

For the 6 sets of tests, each only produced a single unique proof result, and they were all valid. The proof results are in Appendix V. The provers and verifiers also logged the correct number of requests as expected.

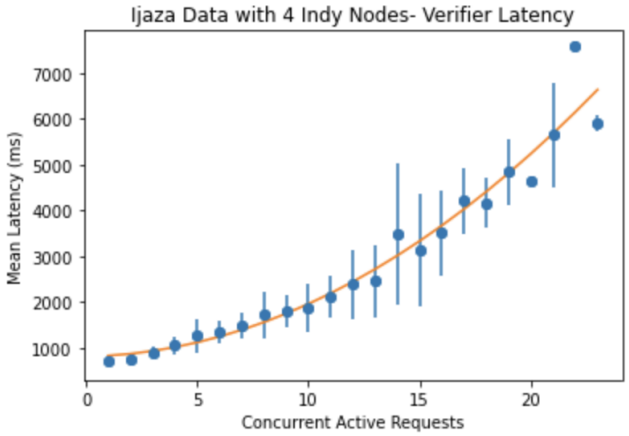
4.3.2 Verifier Latency

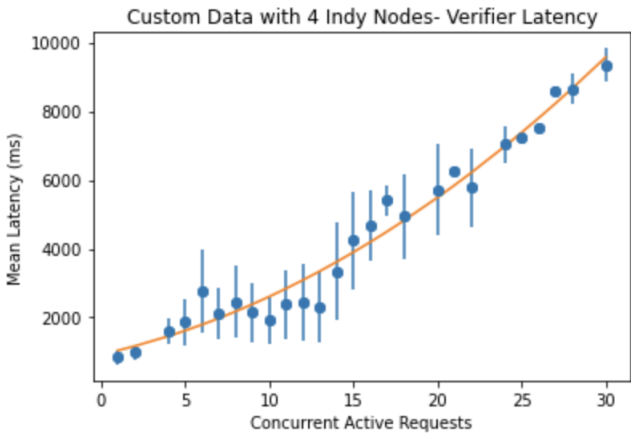
Below are graphs showing the mean latency of requests made by verifiers against the number of concurrent verifier requests on the network. The number of concurrent requests for a given request was determined by finding out how many other requests were active at some time during the time the request was active. The graphs are per dataset and per VON Network configuration. The error bars are 1 standard deviation in both directions. The graphs from the Ijaza data are made from 1254 data points. The other graphs are made from 396 data points, due to the lower number of users in the custom data, and therefore the shorter chains-of-trust.

All the graphs show a strong, greater than linear positive relationship. This could mean that e-Ijaza is not successful at handling lots of requests. However, there are some caveats. Running the entire system on my laptop means that the applications, VON Network and Indy Tails Server were all sharing the same resources, and the networking between them was handled on one machine. It would make sense for the relationship to become greater than linear at the point where there are no longer enough resources, but it is not clear how big a part that plays in the results.

Some of the error bars are very large. This shows an inconsistency in performance. Part of this could be due to the system being run on one machine. It could also be due to NodeJS being a single-threaded program, which means it may have been processing one request at a time causing the rest to have to wait. Therefore, the first request may go through quickly whilst the others could take increasing amounts of time. This would imply that the error bars would keep increasing, but that does not seem to be the case. However, this could simply be because of variation in the amount of data used to form each point. The point at 30 concurrent requests does not necessarily have 30

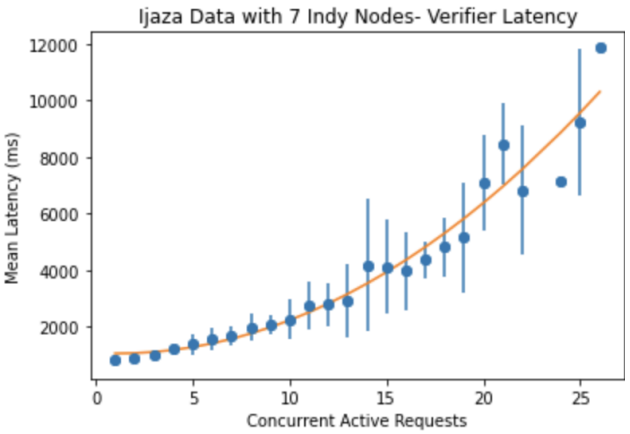
data points, due to how the set of concurrent requests is calculated as described at the start of this section.

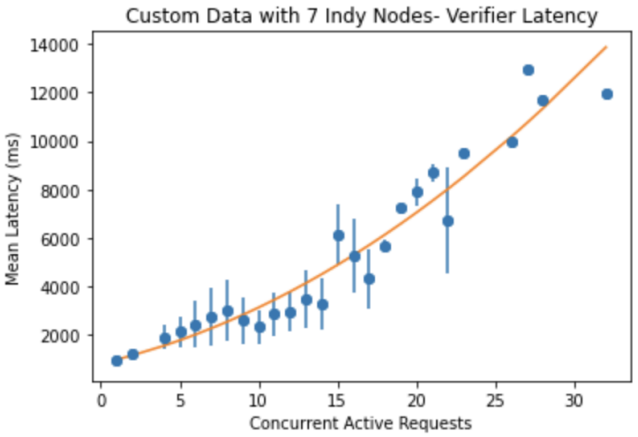





```
y = 10.635x^2 + 8.325x + 813.247
Data Points=1254
```

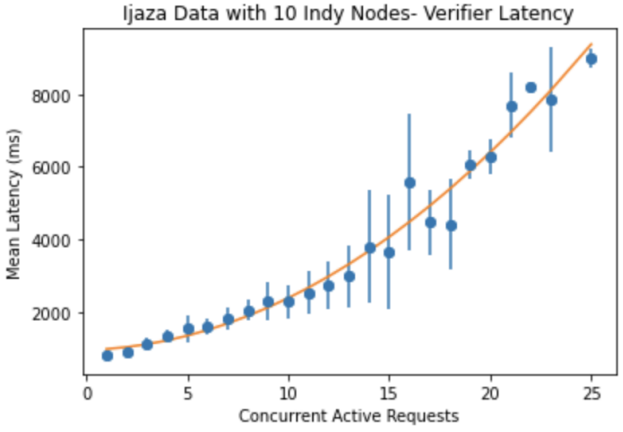
```
y = 5.997x^2 + 107.999x + 927.763
Data Points=396
```

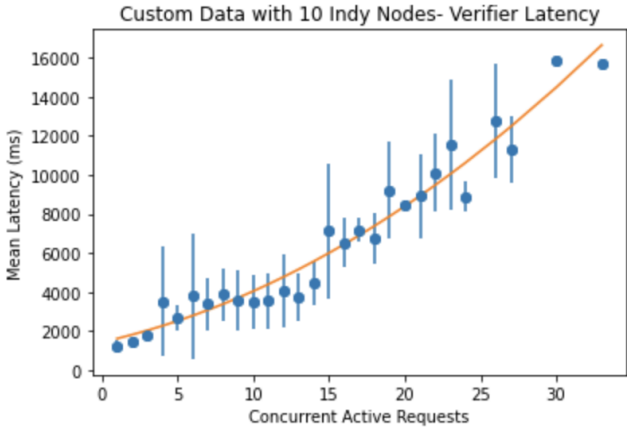





```
y = 14.959x^2 + -34.23x + 1081.34
Data Points=1254
```

```
y = 8.032x^2 + 150.544x + 821.014
Data Points=396
```






```
y = 12.907x^2 + 13.969x + 958.47
Data Points=1254
```

```
y = 8.7x^2 + 174.857x + 1421.048
Data Points=396
```

With the Ijaza data, the number of Indy nodes appears to have a weak, positive correlation with the latency, increasing the acceleration of the gradients on the lines of best fit. However, this correlation is not very clear. For the custom data sets, this correlation seems much clearer. The custom data involves far less users, and therefore far less applications, which means the limited resources may have a lesser effect on the results. This may explain why the difference is clearer with the custom

data. Another factor could be that tests with the custom data involve a user proving using multiple credentials. To verify credentials are not revoked the verifier must get the revocation accumulator from the Indy network, and the associated tails file. This could increase the time taken in verification if it has to be done for more credentials.

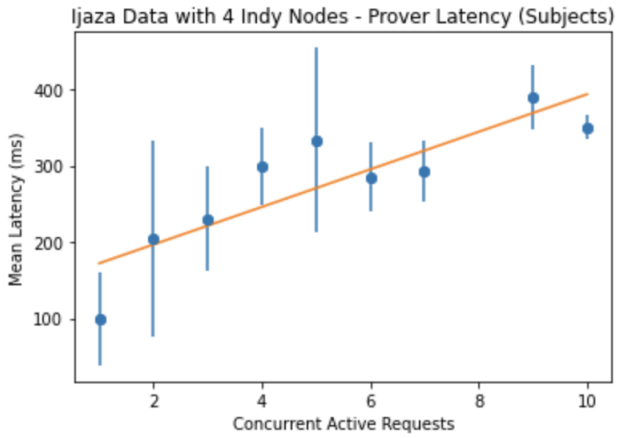
The custom data results generally seem to show a greater latency for the same number of concurrent requests compared to the ijaza data results. This could be related to proofs having multiple credentials. It could also be related to provers and verifiers needing to run tests on the subject ontology to find suitable credentials for the proof and validate that provided credentials satisfy the request. For the ijaza data this is simple as all the proofs are exactly one credential in the subject requested.

4.3.3 Latency When Replying to Request for Subjects

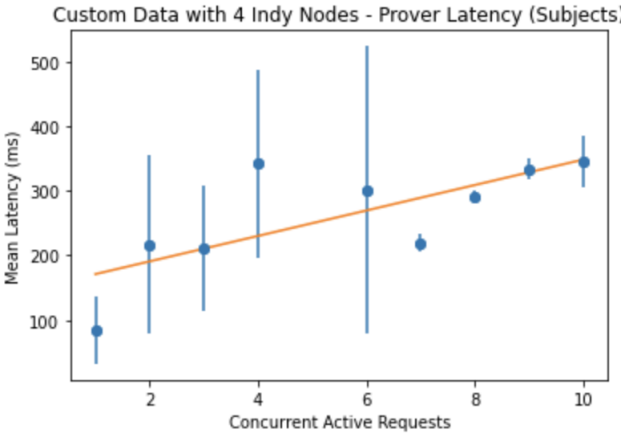
This relates to the first stage of proving credentials, and is the latency between the prover receiving the request and responding, against the number of concurrent requests. This means the number of concurrent requests in the same stage on the same application. The graphs have the same number of data points as the ones in the previous section.

These graphs show a more linear relationship than the previous section. This makes sense, as it involves no network communications or communication with other agents, just computation over its own data. However, the correlation, whilst clear, is not strong, and the error bars are very large. This again shows inconsistency in the performance.

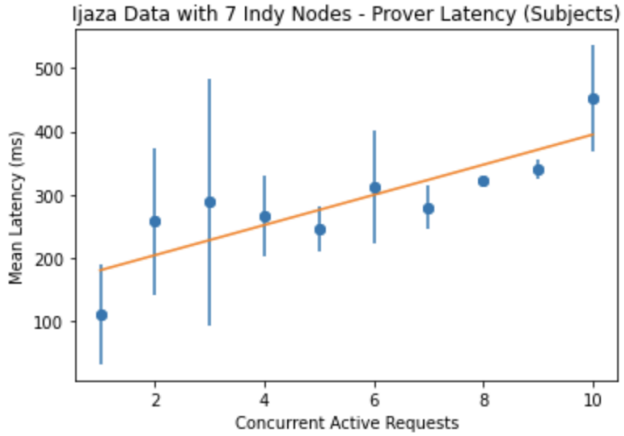
There is no clear difference caused by changing the number of Indy nodes. This makes sense as this stage does not involve any communication with the Indy network. There also is not a very clear difference caused by using the custom data, although looking at the equations the custom data consistently has a larger y-intercept. This could be the additional time caused by more complex searches on the subject ontology, but the inconsistency in the results makes it difficult to say if there is a difference or not.



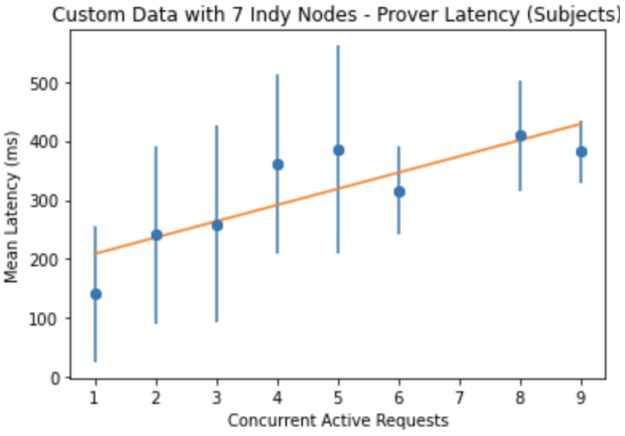
y = 24.597x + 147.486
Data Points=1254



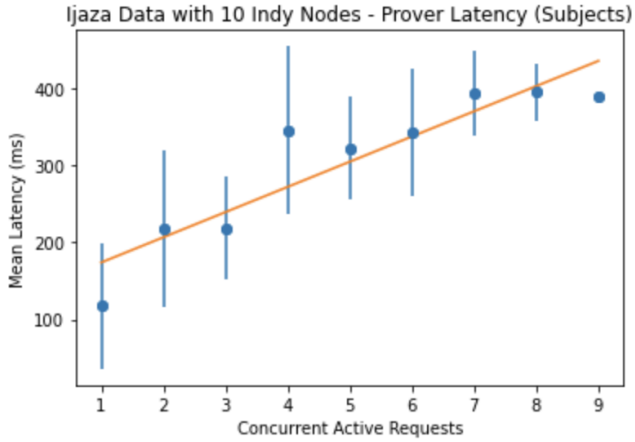
y = 19.685x + 151.52
Data Points=396



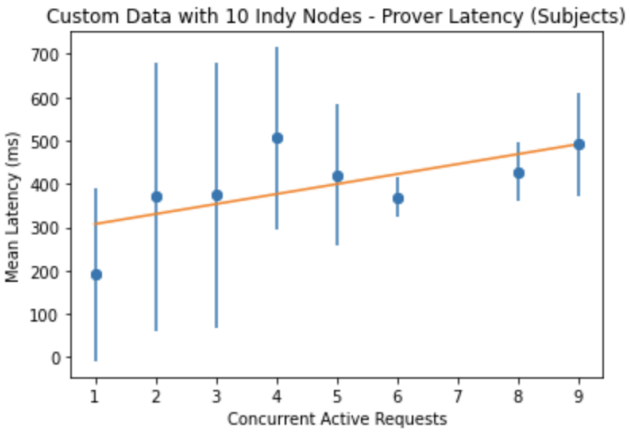
y = 23.805x + 156.985
Data Points=1254



y = 27.496x + 181.28
Data Points=396

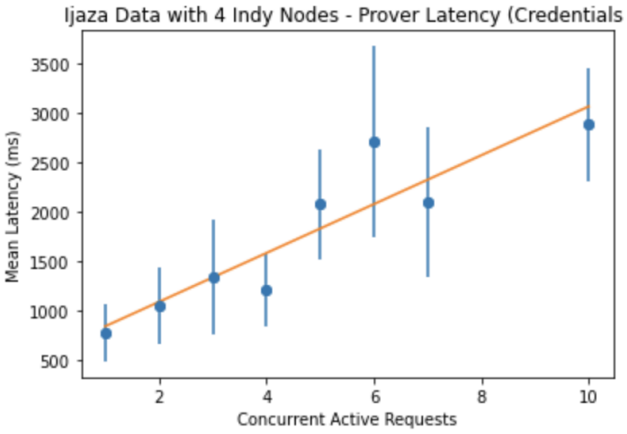


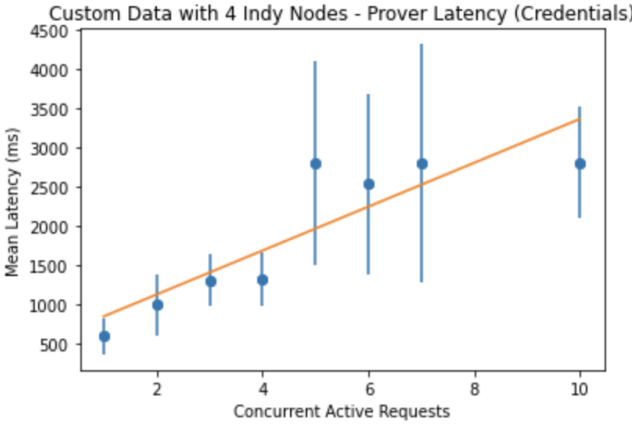
y = 32.727x + 141.061
Data Points=1254



y = 23.055x + 284.449
Data Points=396

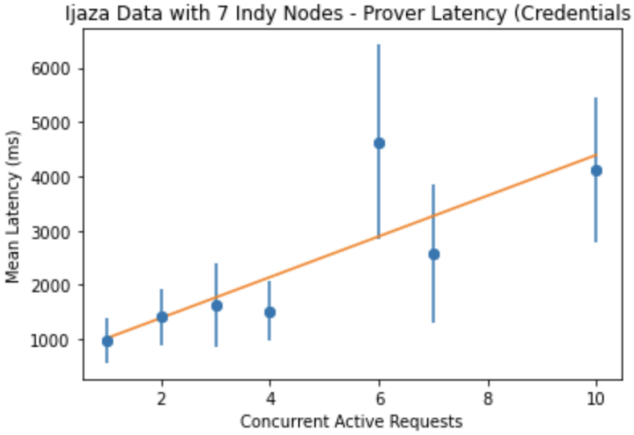
4.3.4 Latency When Replying to Request for Credentials

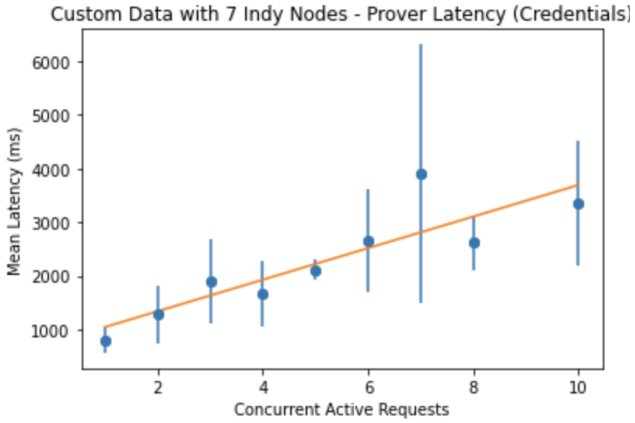





```
y = 247.022x + 594.992
Data Points=1254
```

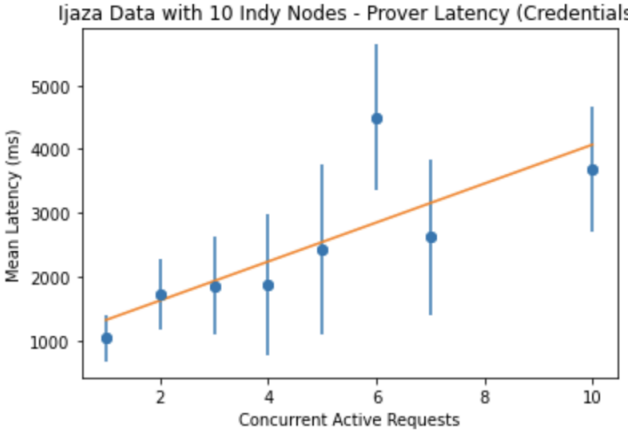
```
y = 279.978x + 570.852
Data Points=396
```

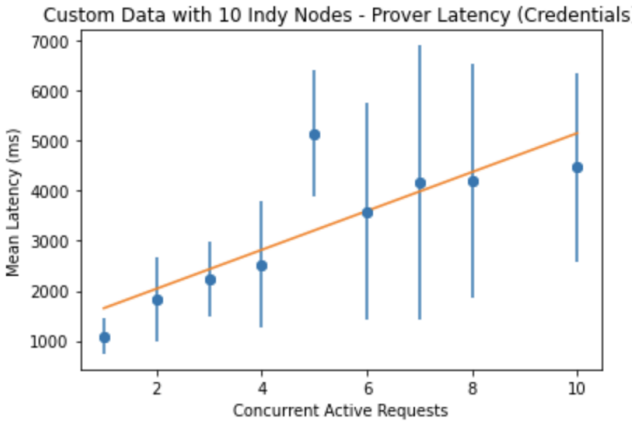





```
y = 375.37x + 638.358
Data Points=1254
```

```
y = 294.097x + 752.208
Data Points=396
```






```
y = 305.281x + 1017.602
Data Points=1254
```

```
y = 389.1x + 1263.534
Data Points=396
```

These graphs relate to the second/final stage of proving credentials. These are also linear correlations. However, the gradients and y-intercepts are significantly greater than the first stage of proving credentials. Proving credentials involves no logical work, but it does require cryptographic proofs to be formed with the credentials. This shows that this work on ACA-Py is substantially greater than the work on the application in the first stage of proving credentials. This makes the first stage seem reasonable.

The number of Indy nodes seems to have a minor effect on the gradients, and a significant effect on the y-intercepts. Part of producing the cryptographic proofs does involve getting the revocation accumulators, which may be the aspect that is slowing down. However again the error bars are very large, showing inconsistency in the performance results.

There is no clear difference caused using the custom data set vs the ijaza dataset.

4.3.5 Evaluation

Generally, the results show that increasing the number of concurrent requests increases latency, although performance is inconsistent. A number of reasons have been given, in this chapter, for some of the increases in latency, including the single-threaded nature of NodeJS and the limited resources on a laptop.

The results do show linear relationships on the prover (4.3.3 and 4.3.4) which is good, and shows the stage of creating a proof from credentials, a standard flow in Indy/Aries, has a greater latency and a greater rate of increasing latency than the first stage of proving the credentials (sending the subjects).

The results from the verifier show a greater than linear growth in latency, which is not ideal. Since the growth is linear on the prover, this could be caused by either the network, verifying credentials, or delayed processing of incoming requests on the prover, with the latter being a result of the application, not the network or Indy/Aries.

To know whether the latency is an issue, we would need to know the expected loads (or concurrent requests) on the applications. This is not clear, as it could be assumed that the average user would not have many concurrent requests, but if a user is in a lot of chains-of-trust then they could have a lot of proof requests. We cannot know how many chains-of-trust the average user is likely to be essential to without research, so assessing what the likely loads on user applications is outside the scope of this project. The tests would also need to be run in an environment more similar to production, with Indy nodes (and e-Ijaza applications) hosted on separate machines.

There are lots of limitations to this evaluation, some of which have been mentioned in the chapter. Measuring the performance of user applications is useful, but the performance of the controller application is critical, and it has to handle large numbers of requests of various types. It is unlikely the controller application I have produced would be capable of running a production scale e-Ijaza system. The evaluation also does not test the effect of the complexity of the subject ontologies on performance, which is important. And last but not least, it does not test how updating affects performance. To know whether the effects of updates on performance is critical, it would be essential to know how often the subject ontology and set of master credentials is likely to change, both in normal cases and extreme cases.

5 Conclusion

5.1 Success Criteria

Some changes were made since the project proposal. Firstly, in my project proposals I intended to use Sovrin. During my preparation I realised that Sovrin donated its code to the Linux Foundation to become Hyperledger Indy. Sovrin is a global deployment of Indy, run by The Sovrin Foundation which manages the legal agreements and requirements of actors on the Sovrin Network. Due to the complexities of using Sovrin, I opted to use the development-focussed deployment of Indy, VON Network, instead.

In my project proposal I intended to allow certificates to be combined and narrowed. As discussed in section 2.5 this was not possible, and I implemented a different way for users to prove authority in a subject using other credentials, and still have control over their privacy.

Finally, my proposal intended to allow voting based on a user's credentials. As described in section 2.7, this is problematic so e-Ijaza allows voting based on a user's master credentials instead, which alleviates the problems.

With these two caveats, the original success criteria were met. E-Ijaza implements Indy-based credentials that are verifiable via chain-of-trust, ending with master credentials. The set of master credentials and the subject ontology are adjustable via voting. Proofs of authority in subjects can be satisfied using other credentials, and e-Ijaza can represent the data provided by Maan Al-Dabbagh.

5.2 Reflections

In hindsight, it would have been better to have known the subject area better before I began. I would also have liked to spend more time on evaluation, testing across multiple machines and possibly even applying to use the Sovrin development network. I might have also explored storing data differently, as whilst storing as ICs helped me understand the technology better and meant I was not focussing on too many different things, it could have been interesting to use a database or some other technology which stored the data in a more intuitive manner, and that was optimised for writing and reading data rather than issuing, receiving and proving ICs.

I would have also liked to explore the protocols available on Aries more. I avoided any whose specifications were not accepted, or whose implementations were not complete, as using them would be risky and could lead to unnecessary complications. But there were some interesting ones, such as the introduction protocol, which allows agents to create a connection between two other agents it is connected to. This could be used in the chain-of-trust verification, where the prover connects the verifier with the users that issued the credentials used in the proof, and maybe the issuers can automatically reveal their proofs on the basis that it is to prove a credential they issued.

Another change would be using a language and framework more suited to handling concurrent requests and local state, or developing a greater understanding of how to optimise NodeJS applications for these circumstances. This was not the focus of the project, and therefore sticking with what I was more familiar with was appropriate. However, the application would be far more practical if built with performance in mind.

Finally, I would have liked to have displayed users' names rather than just public DIDs. Public DIDs are a unique identifier but they are not very useful for a human. However if the name is revealed as part of communications, agents could reveal different names to different people, making it an easy target for abuse. One solution would be to have users publicly expose a name, making it harder to abuse as they cannot have different names in different communications. Another option is to allow users to store associations between public DIDs and names, like a mobile phone's contacts list. Although the latter option is not very effective when unknown users communicate, whether by issuing credentials or requesting proofs.

5.3 Future Work

An interesting direction for future work would be to look at using credentials to control access to another blockchain-based technology. This could be used to store the set of masters, the subject ontology, and the proposals on both. It could be a public-permissioned blockchain, where anyone can read it but write access is limited (according to the user's held master credentials). Besides the more complex access control, work would have to be done in building the blockchain-based technology with an agent to communicate on Indy networks, which is needed for the access control but also for issuing master credentials and vote credentials, and doing the rest of what the controller application does.

Another direction is to modularise the project, turning it into a number of libraries that can be used in other Indy agents and applications. One of the key aspects of Indy-based digital identity is controlling all your identity in place in a way that maintains your privacy. E-Ijaza currently only handles e-Ijaza credentials, but if it was part of agents and applications that handle other ICs it would be far more practical. It would, for example, allow someone to provide their chain-of-trust-based credentials along with their passport or other ID, when applying for jobs etc.

Bibliography

de Jong, L. (2020). *Connecting ACA-py to Development Ledgers*. Laurence de Jong. https://ldej.nl/post/connecting-acapy-to-development-ledgers/

Developers. (n.d.). Sovrin. https://sovrin.org/developers/

Getting Started Aries Dev. (n.d.). GitHub. https://github.com/hyperledger/aries-cloudagent-python/tree/main/docs/GettingStartedAriesDev

Hardman, D. (2018). *How DIDs, Keys, Credentials, and Agents Work Together in Sovrin 131118*. Sovrin. https://sovrin.org/wp-content/uploads/2019/01/How-DIDs-Keys-Credentials-and-Agents-Work-Together-in-Sovrin-131118.pdf

Reed, D. (2019). *Hyperledger Aries: The Next Major Step Towards Interoperable SSI*. Evernym. https://www.evernym.com/blog/hyperledger-aries/

Sovrin Glossary V3. (2019). Sovrin. https://sovrin.org/wp-content/uploads/Sovrin-Glossary-V3.pdf

sovrin one pager. (n.d.). Sovrin. https://sovrin.org/wp-content/uploads/2018/08/sovrin-one-pager-8.5x11-200818.pdf

Tobin, A., & Evernym. (2018). *What Goes on the Ledger?* Sovrin. https://sovrin.org/wp-content/uploads/2018/10/What-Goes-On-The-Ledger.pdf

Tobin, A., & Reed, D. (2016). *The Inevitable Rise of Self-Sovereign Identity*. Sovrin. https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf

Windley, P. J. (2016). *How Sovrin Works*. Sovrin. https://sovrin.org/wp-content/uploads/2018/03/How-Sovrin-Works.pdf

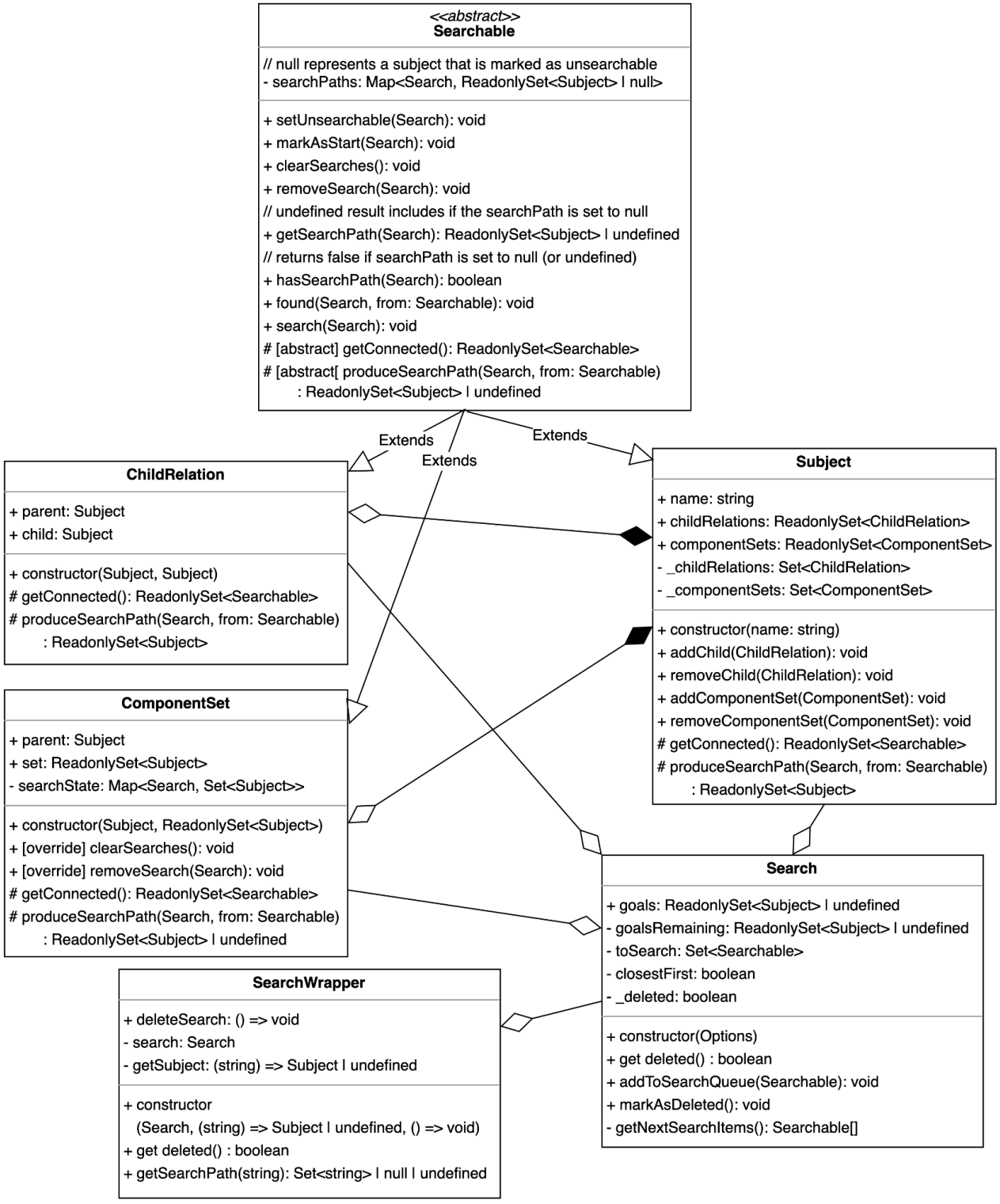
Appendix

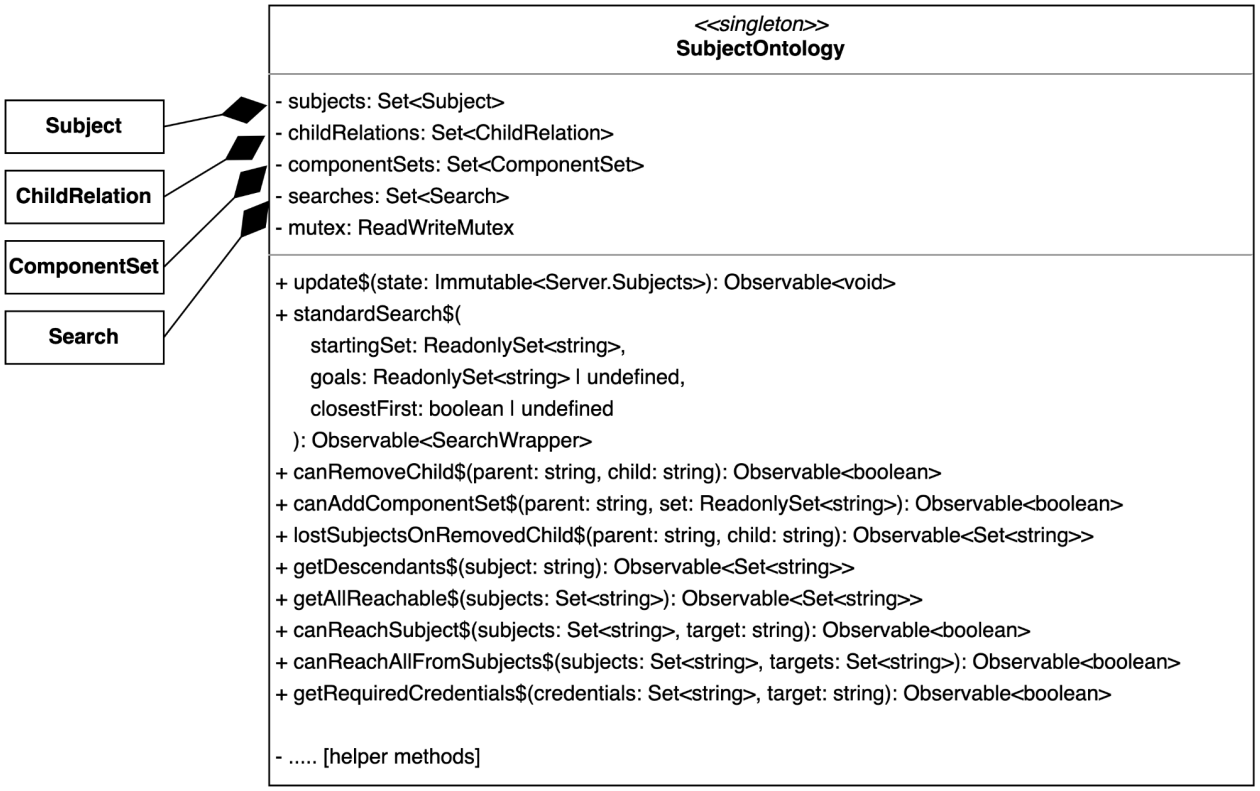
Appendix I - Aries Frameworks

Name	Overview	Development State	Application Code	Assessment
Aries Cloud Agent - Python	An always-on agent to be deployed in the cloud, not suitable for mobile-agent applications.	The project successfully embeds Indy-SDK and has production-level deployments. It also has the most consistently maintained guide for new developers.	The framework exposes a HTTP interface for an application/controller (written in any language) to use. It can also send updates to a HTTP webhook.	This is ideal to use as it gives flexibility in terms of the application code, and seems stable. Having guides for new developers is essential as I do not have time to explore repositories and manually discover features. The only down-side might be it cannot be used for mobile-agents.
Aries Framework - .NET	Can be used for building mobile and server-side agents.	The project successfully embeds Indy-SDK and has production-level deployments.	The application/controller can be written in any language that can embed the framework as a library.	This is a reasonable framework since it seems ready to use and gives flexibility in terms of application code. It is not clear the quality of its developer guides
Aries Static Agent - Python	A configurable agent that does not use persistent storage. All keys and other aspects must be statically configured.	The framework needs to be paired with a “full agent” that is capable of DIDComm (DID-based secure communication). The agent does not have a wallet for storing credentials.	The application/controller must be written in python and use the framework as a library.	This seems less than ideal as it requires additional components. It would not be smart to waste time learning how to put together a working agent when the project’s focus is building on top of

				the agent.
Aries Framework - Go Aries-sdk-ruby aries-framework-javascript	These are all under active development and not ready for “out of the box” use.	They do not all embed Indy-SDK, or do not fully implement it.		None of these are sensible solutions for this project, as they lack complete functionality and are not simple to use. This would mean simply running the agent will consume a lot of development time.

Appendix II - Subject Ontology Class Diagram




Subject
ChildRelation
ComponentSet
Search
<<singleton>>
SubjectOntology
- subjects: Set<Subject>
- childRelations: Set<ChildRelation>
- componentSets: Set<ComponentSet>
- searches: Set<Search>
- mutex: ReadWriteMutex
+ update$(state: Immutable<Server.Subjects>): Observable<void>
+ standardSearch$(
startingSet: ReadonlySet<string>,
goals: ReadonlySet<string> | undefined,
closestFirst: boolean | undefined
): Observable<SearchWrapper>
+ canRemoveChild$(parent: string, child: string): Observable<boolean>
+ canAddComponentSet$(parent: string, set: ReadonlySet<string>): Observable<boolean>
+ lostSubjectsOnRemovedChild$(parent: string, child: string): Observable<Set<string>>
+ getDescendants$(subject: string): Observable<Set<string>>
+ getAllReachable$(subjects: Set<string>): Observable<Set<string>>
+ canReachSubject$(subjects: Set<string>, target: string): Observable<boolean>
+ canReachAllFromSubjects$(subjects: Set<string>, targets: Set<string>): Observable<boolean>
+ getRequiredCredentials$(credentials: Set<string>, target: string): Observable<boolean>
- ..... [helper methods]

Appendix III - Lines of Code

I counted the lines of code using cloc (https://github.com/AlDanial/cloc). I only counted the files I had primarily written. There were some that were generated by Angular CLI, and I made some minor edits to, which I did not include.

cloc ignores blank files. However, the codebase has some that were generated as part of boilerplate by Angular CLI but I then did not use (specifically scss files).

/src/app

```
cloc --exclude-ext=spec.ts,module.ts app
```

**.spec.ts and *.module.ts were generated and edited by Angular CLI, and I did not edit them.*

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                      27            246            159           1783
HTML                            16             55             31            575
SCSS                             7             16              0             80
-------------------------------------------------------------------------------
SUM:                            50            317            190           2438
-------------------------------------------------------------------------------
```

/src/server

```
cloc server
```

The directory's contents were entirely written by me.

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                      99            585             27           6224
-------------------------------------------------------------------------------
SUM:                            99            585             27           6224
-------------------------------------------------------------------------------
```

/src/types

```
cloc types
```

I wrote everything in the directory except one file, which was generated by openapi-typescript..

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                      13            123           2461           4489
-------------------------------------------------------------------------------
SUM:                            13            123           2461           4489
-------------------------------------------------------------------------------
```

```
cloc types/aries/aries-autogen-types.ts
```

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                       1              4           2461           4021
-------------------------------------------------------------------------------
```

So the total lines of code in the directory is 4489 - 4021, which is 468 (across 12 files).

/src/utils

```
cloc utils
```

The directory's contents were entirely written by me.

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                       7             21              0            131
-------------------------------------------------------------------------------
SUM:                             7             21              0            131
-------------------------------------------------------------------------------
```

/scripts

```
cloc scripts
```

The directory's contents were entirely written by me.

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Bourne Again Shell               2             25              0            173
Dockerfile                       3              1              1             27
-------------------------------------------------------------------------------
SUM:                             5             26              1            200
-------------------------------------------------------------------------------
```

/test

```
cloc ./application-wrapper.ts ./controller.ts ./ijaza-data.ts
./custom-data.ts ./ontology-creator.ts ./proof-result-logger.ts
./run-tests.ts ./test-data-type.ts ./user.ts ./verifier.ts
```

The directory also contains the logs from the tests, so I had to manually choose the files I wrote.

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                      10             60             26            672
-------------------------------------------------------------------------------
SUM:                            10             60             26            672
-------------------------------------------------------------------------------
```

There are also 81 lines of code in the Jupyter notebook (process_test_results.ipynb). I manually counted this by copying the code sections into a file and removing the blank lines.

This total

Total

The total lines of code count is 10214, across 157 typescript files, 16 HTML files, 7 SCSS files, 2 BASH files, 3 Dockerfiles, and 1 (python) Jupyter notebook.

Appendix IV - Ijaza Data

Arabic

١ - صحيح البخاري:

سمعت «صحيح الإمام» الحافظ الجهبذ الناقد اللافظ أمير المومنين في الحديث، أبي عبد الله محمد بن إسماعيل البخاري عن شيخنا الشيخ محمد المذكور، وذلك عام مجاورته بمكة المشرفة سنة سبعين وألف من الهجرة النبوية، سماعاً لبعضه من أوله إلى قوله: «بوادره»[٢]

وذلك بقراءة شيخنا العلَّامة القدوة الهمام شيخ الإسلام الشيخ عيسى بن محمد بن محمد بن أحمد الجعفري المغربي المكي المالكي، وإجازة لسائره[٣].

وقد سمعت منه أيضاً في مجاورته الأولى أبوابًا من الصحيح بقراءة الشيخ علي الأيوبي الخطيب بمكة المشرفة[١]، وأجاز الحاضرين - ومنهم الفقير - به عن أبي النجا سالم بن محمد السنهوري سماعاً عليه لبعضه وإجازة لسائره قال: قرأته جميعًا على المسند النجم محمد بن أحمد الغَيْطِي، بقراءته لجميعه على شيخ الإسلام القاضي زكريا، بقراءته لجميعه على شيخ السُّنة أبي الفضل بن حجر[٢]، بسماعه لجميعه على الأستاذ إبراهيم بن أحمد التنوخي، بسماعه لجميعه على أبي العباس أحمد بن أبي طالب الحجار، بسماعه لجميعه على السراج الحسين بن [ق/ ٣/ أ] المبارك الزَّبيدي - بفتح الزاي - الحنبلي، سماعاً لجميعه، عن أبي الوقت عبد الأول بن عيسى بن شعيب السجزي الهروي سماعاً، عن أبي الحسن عبد الرحمن بن محمد الداودي سماعاً، عن أبي محمد عبد الله بن أحمد السَّرَخْسِي سماعاً، عن محمد بن يوسف بن مطر الفرَبْري سماعاً، عن أمير المؤمنين في الحديث محمد بن إسماعيل البخاري رحمه الله تعالى سماعاً، فذكره.

وبالسند قال الإمام البخاري:

حدثنا مكي بن إبراهيم قال: حدثنا يزيد بن أبي عبيد عن سلمة بن الأكوع رضي الله تعالى عنه قال:

سمعت النبي ﷺ يقول: «مَنْ يَقُلْ عَلَيَّ مَا لَمْ أَقُلْ فَلْيَتَبَوَّأْ مَقْعَدَهُ مِنَ النَّارِ»[٣].

أخرجه في كتاب العلم في باب إثم من كذب على النبي ﷺ، وهذا من ثُلاثِيَّاتِهِ، وَجُمْلَتُها خمسةٌ وعشرون حديثاً.

Translation

The translation is done using Google Translate's Scan feature. It is not completely accurate, but it is sufficient to extract an Ijaza chain for the purposes of testing e-Ijaza. The Ijaza chain is in a specific saying of the Prophet Mohammed (the final prophet in Islam).

1 - Sahih al-Bukhari:

I heard the hadith of "Sahih al-Imam," the hafiz, the critic, the emir of the believers in the hadith, Abu Abdullah Muhammad bin Ismail al-Bukhari on the authority of our sheikh, Sheikh Muhammad mentioned, and that was the year of his neighbourhood in Mecca in the year one thousand and seventy of the Prophet's migration, listening to some of it from the beginning to His saying:

"Abwadarh" by the reading of our Sheikh, the eminent role model, the great sheikh of Islam, Sheikh Issa bin Muhammad bin Muhammad bin Ahmed al-Jaafari, the Maghribi al-Makki al-Maliki, and an authorization for the rest of it.

I also heard from him, in his first neighbourhood, chapters of the Sahih by the recitation of Sheikh Ali Al-Ayoubi Al-Khatib in the honourable Mecca, and the attendees - including the poor - permitted it on the authority of Abu Al-Naga Salem bin Muhammad Al-Sanhouri, listening to some of them and authorizing the rest of them. All of them on the Sheikh of Islam Qadi Zakaria, with his recitation of all of them on the Sheikh of Shana Abi Al-Fadl bin, with his hearing on all of them on Professor Ibrahim bin Ahmed Al-Tanukhi, with his hearing on all of them on Abi Al-Abbas Ahmed bin Abi Talib Al-Hajjar, with his hearing on all of them on Siraj Al-Hussein bin Al-Mubarak Al-Zubaidi - Fath Zai - Hanbali, listening to all 5, on the authority of Abi al-Waqat Abdul Awal bin Issa bin Shuaib al-Sijzi al-Harawi, on the authority of Abu al-Hasan Abd al-Rahman bin Muhammad al-Dawdi, on the authority of Ahmad al-Sharkhasi, on the authority of Muhammad ibn Yusuf ibn Matar, Commander of the Faithful in hadith Muhammad ibn Ismail al-Bukhari Allah ibn Abi Muhammad Abd al-Farbari On hearing, may God Almighty have mercy on him, he mentioned it. And with the chain of transmission, Imam al-Bukhari said: Makki bin Ibrahim told us, he said: Yazid bin Abi Obaid told us on the authority of Salamah bin Al-Akwa', may God Almighty be pleased with him, he said: I heard the Prophet, peace and blessings be upon him, say: "He who says about me what I did not say, let him take his seat in the Fire."

He brought it out in the Book of Knowledge in the chapter on the sin of the one who lied about the Prophet, peace be upon him, and this is one of his triads, and the whole of it is twenty hadiths.

Ijaza Chain

Sheikh Muhammad

Sheikh Issa bin Muhammed bin Muhammed bin Ahmed al-Jaafari

Sheikh Ali Al-Ayoubi Al-Khatib

Abu Al-Naga Salem bin Muhammad Al-Sanhouri

Sheikh of Islam Qadi Zakaria

Sheikh of Shana Abi Al-Fadl bin

Professor Ibrahim bin Ahmed Al-Tanukhi

Abi Al-Abbas Ahmed bin Abi Talib Al-Hajjar

Siraj Al-Hussein bin Al-Mubarak Al-Zubaidi

Fath al-Zai al-hanbali

Abi al-Waqat Abdul Awal bin Issa bin Shuaib al-Sijzi al-Harawi

Abu al-Hasan Abd al-Rahman bin Muhammad al-Dawdi

Ahmad al-Sharkhasi

Muhammad ibn Yusuf ibn Matar

Muhammad ibn Ismail al-Bukhari Allah ibn Abi Muhammad Abd al-Farbari

Makki bin Ibrahim

Yazid bin Abi Obaid

Salamah bin Al-Akwa'

The prophet

Appendix V - Proof Results from Tests

Ijaza Data


```
{
 "name": "Sheikh_Muhammad",
 "subject": "ijaza_subject",
 "result": true,
 "proof": [
  {
   "name": "Sheikh_Issa_bin_Muhammed_bin_Muhammed_bin_Ahmed_al-Jaafari",
   "subject": "ijaza_subject",
   "result": true,
   "proof": [
    {
     "name": "Sheikh_Ali_Al-Ayoubi_Al-Khatib",
     "subject": "ijaza_subject",
     "result": true,
     "proof": [
      {
       "name": "Abu_Al-Naga_Salem_bin_Muhammad_Al-Sanhouri",
       "subject": "ijaza_subject",
       "result": true,
       "proof": [
        {
         "name": "Sheikh_of_Islam_Qadi_Zakaria",
         "subject": "ijaza_subject",
         "result": true,
         "proof": [
          {
           "name": "Sheikh_of_Shana_Abi_Al-Fadl_bin",
           "subject": "ijaza_subject",
           "result": true,
           "proof": [
            {
             "name": "Professor_Ibrahim_bin_Ahmed_Al-Tanukhi",
             "subject": "ijaza_subject",
             "result": true,
             "proof": [
              {
               "name": "Abi_Al-Abbas_Ahmed_bin_Abi_Talib_Al-Hajjar",
               "subject": "ijaza_subject",
               "result": true,
               "proof": [
                {
                 "name": "Siraj_Al-Hussein_bin_Al-Mubarak_Al-Zubaidi",
                 "subject": "ijaza_subject",
                 "result": true,
                 "proof": [
                  {
                   "name": "Fath_al-Zai_al-hanbali",
                   "subject": "ijaza_subject",
                   "result": true,
                   "proof": [
```

```
{
 "name": "Abi_al-Waqat_Abdul_Awal_bin_Issa_bin_Shuaib_al-Sijzi_al-Harawi",
 "subject": "ijaza_subject",
 "result": true,
 "proof": [
  {
   "name": "Abu_al-Hasan_Abd_al-Rahman_bin_Muhammad_al-Dawdi",
   "subject": "ijaza_subject",
   "result": true,
   "proof": [
    {
     "name": "Ahmad_al-Sharkhasi",
     "subject": "ijaza_subject",
     "result": true,
     "proof": [
      {
       "name": "Muhammad_ibn_Yusuf_ibn_Matar",
       "subject": "ijaza_subject",
       "result": true,
       "proof": [
        {
         "name":
"Muhammad_ibn_Ismail_al-Bukhari_Allah_ibn_Abi_Muhammad_Abd_al-Farbari",
         "subject": "ijaza_subject",
         "result": true,
         "proof": [
          {
           "name": "Makki_bin_Ibrahim",
           "subject": "ijaza_subject",
           "result": true,
           "proof": [
            {
             "name": "Yazid_bin_Abi_Obaid",
             "subject": "ijaza_subject",
             "result": true,
             "proof": [
              {
               "name": "Salamah_bin_Al-Akwa",
               "subject": "ijaza_subject",
               "result": true,
               "proof": [
                {
                 "name": "The_prophet",
                 "subject": "ijaza_subject",
                 "result": true,
                 "proof": [
                  {
                   "name": "controller",
                   "subject": "ijaza_subject",
                   "result": true,
                   "proof": true
                  }
                 ]
```

```
                                        }
                                      ]
                                    }
                                  ]
                                }
                              ]
                            }
                          ]
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Custom Data


```
{
 "name": "5",
 "subject": "A",
 "result": true,
 "proof": [
   {
     "name": "4",
     "subject": "A",
     "result": true,
     "proof": [
       {
         "name": "2",
         "subject": "B",
         "result": true,
         "proof": [
           {
             "name": "1",
             "subject": "B",
             "result": true,
             "proof": [
               {
                 "name": "controller",
                 "subject": "A",
                 "result": true,
                 "proof": true
               }
             ]
           }
         ]
       },
       {
         "name": "3",
         "subject": "C",
         "result": true,
         "proof": [
           {
             "name": "1",
             "subject": "C",
             "result": true,
             "proof": [
               {
                 "name": "controller",
                 "subject": "A",
                 "result": true,
                 "proof": true
               }
             ]
           }
         ]
       }
     ]
   }
 ]
}
```

Phase 2 Proposal

Decentralised Chain-of-trust Based Teaching Certification

Computing systems can be designed either to have centralised or decentralised authority. This can be seen in centralised banks vs blockchain currencies. Many modern education has taken the approach of a centralised system. However, this is not obviously the best approach in all circumstances, as it gives the educational institutions (and often by extension governments) full control over the system and the verification of students' knowledge and certification.

My project (provisionally called e-Ijaza) will implement a digital version of a system known as Ijaza to explore the potential of implementing decentralised educational credentials systems. In Ijaza a certificate asserts the owner's knowledge in a certain subject matter, and allows them to then issue certificates to others in that subject matter or a subset of it. The certificates are validated in a similar way to chains of trust, by validating the certificate and the teacher's certificate until reaching some trusted point. Without a centralised authority controlling the definition of courses and subjects, Ijaza certificates can be issued in any subject or in any subset of a subject. The key points to consider in this project are legitimacy of the certificates (ensuring they are not fake), verifiability of the certificates (ensuring their assertions can be validated through the chain of trust) and confidentiality (users should not have to reveal more about their studies than necessary).

E-Ijaza will use Sovrin, a system built on blockchain for decentralised identity. Sovrin is a public-permissioned network which is managed by Stewards who run validator nodes which read and write on the ledger, with the ledger and Stewards being the root of trust. Sovrin allows users to make *claims* about themselves and others, and through decentralized identifiers (DIDs) and asymmetric cryptography can allow others to verify those claims independently of the identity provider or a centralised register holding the claims. It also allows revocation of the claims. Sovrin will be used for the issuing and ensuring legitimacy of certificates (which will be implemented as claims). However Sovrin only verifies a claim is legitimate, and can't do complex or specific verifications such as verifying a chain of trust, which requires some additional functionality.

Sovrin also is not capable of ensuring the confidentiality of the certificates due to the flexibility of the certificates. For example, if I had an Ijaza certificate in theology but wanted to show my authority just in a specific school of thought, this would not be possible in Sovrin. Sovrin does have an implementation of Zero Knowledge Proofs (ZKP) which allows the hiding of attributes on its claims, but for this case it would require the claim to hold all the subsections of theology, which could make the claims very large (there would a lot of attributes) and would be unfeasible and impractical to update anytime the subject evolved in any way. E-Ijaza will require a ontology of subjects in order to allow the narrowing of certificates, which will need to be maintained separately from Sovrin. This will also need to work in the reverse direction for merging certificates (i.e. combining certificates in different schools of thought of theology into a certificate in theology). The system will need to be inherently open-ended to allow for subject evolution.

Starting Point

I have no experience using or building decentralised technologies. I have read a few of Sovrin's whitepapers explaining how it works, but have no other knowledge or experience of the technology.

I have some experience building non-professional websites and web apps, particularly in Angular, although I have little to no experience with testing.

Work to be Done

Learning to use Sovrin and Setting up the Development Environment

Before I can start building the e-Ijaza system I need to set up my development environment. Whilst Sovrin is a live public-permissioned network that would require appropriate access to use, I will be using a local version of it for development that is available via the code repository (it is an open-source project). I will need to become familiar with the appropriate sections of the code and API, and how issuing and verifying the certificates work, how I can add data to the ledger and what the restrictions are, and how to create users and identity providers.

After this I will have to ensure that my local environment is set up such that any code I write can access the Sovrin APIs in a similar way to how they would on a live implementation.

Chain of Trust Verification

Whilst Sovrin can verify a claim is not fake, it can not do complex or specific verification techniques such as required by e-Ijaza (specifically following and verifying the chain of trust). Part of solving this will require properly defining how the certificates will be implemented as claims to ensure the chain of trust can be followed and verified, and the other is actually doing the verification.

To follow the chain of trust e-Ijaza will need to be able to access the certificates of the issuer (recursively), or at least the certificate(s) that is appropriate for the one being verified. This can either be done by including the certificates in the certificate being verified, or the certificates in the chain being put on the public ledger. Putting certificates on the ledger would reduce privacy and make the chain harder to maintain, but would prevent the underlying Sovrin claims getting too large. e-Ijaza will seek to implement both solutions, therefore shortening the chain needed to be stored in the certificates if anyone chooses to make their Ijaza certificate(s) public.

Once it is possible to follow the chain, it needs to be verified. The core part of this is using e-Ijaza's ontology of subjects to ensure that each subject of each certificate is covered by those higher in the chain. It is possible that the chain will not be a single path, as multiple certificates may have been used to authorise the creation of one. The verification will need to account for this. It is also necessary for the chain to end somewhere, which can either be at some point that the verifier trusts or at some generally trusted point, which will be represented by a master certificate.

Master Certificates

For the chain of trust to work it is necessary to have master certificates (if a set of trusted teachers aren't provided) that act as the end of the chains of trust. This will require a system for producing master certificates, which includes deciding who gets them in what subject. The system will start with a root set of master certificates given out to a privileged set of people (provisionally called *Wardens*). Further master certificates will be issued (creating new Wardens) via a decentralised voting mechanism, with the exact mechanism to be decided in the project. A similar voting mechanism will be used for revoking master certificates. When issuing a master certificate, if the subject does not already exist then it will be created on the e-Ijaza ontology of subjects system.

The actual record keeping in e-Ijaza will be done on a blockchain ledger (seperate from Sovrin's). The system will need to allow voting using the Sovrin certificates and the e-Ijaza chain of trust verification system, whilst keeping a record of proposals, votes, and the current state. The system's API will also need to be registered on Sovrin as a public identity provider. This system will become a root of trust in e-Ijaza, as the issuing of master certificates is essential and not logically decentralised.

Ontology of Subjects

The ontology of subjects needs to be managed in an open and decentralised manner. In practice this will mean it is decided via a majority vote held for a minimum amount of time (similar to master certificates), such that people can vote or propose changes on subjects that they have a certificate in. This means they can vote on what a subject covers, and they can vote on what is needed to form the subject (when merging a certificate). These are not necessarily the same. The subjects are created one of two ways, either a master certificate is issued in it, therefore creating it, or it is voted into existence as a subpart of another subject. Each subject will have a unique ID to be used in the certificates. For the purpose of voting, someone's certificate in a subject is valid if the chain ends with a master certificate.

The actual voting and record keeping will be similar to master certificates in terms of technology. This system will also become a root of trust in e-Ijaza.

User Interface

Having a user interface to allow people to interact with e-Ijaza is essential. Users will have to login using their certificate(s), which is likely to be the most complicated part. It will likely be implemented as a web interface, and will need to interact with all the e-Ijaza systems and some of the Sovrin APIs (for issuing and revoking certificates).

Evaluation

I will evaluate e-Ijaza to ensure it has the required functionality to meet the success criteria. To do this I will use some data on historical Ijaza chains that Maan Al-Dabbagh from the Faculty of Asian and Middle Eastern Studies has collated during his PhD, and sent to me. The data is contained within a pdf so I will need to extract it and put it into a format that my system can use. The data includes a number of Ijazas with their subject, along with the chain of teachers which resulted in that Ijaza.

It would also be good to evaluate how long it takes the system to reach consensus, as this is often a core issue with decentralised systems. As the number of nodes increases, it is often the case that throughput increases and latency increases. E-Ijaza could be evaluated to see how the performance in these aspects vary as the number of nodes increases, and it could be compared to the performance of Sovrin on its own. The performance could also be compared with running e-Ijaza on a single computer.

Success Criteria

1. E-Ijaza should allow users to produce Sovrin-based certificates that are verifiable via a chain of trust, which may end with master certificates, with the set of master certificates being decided via a voting process.
2. E-Ijaza should allow certificates to be combined and narrowed.
3. E-Ijaza should allow users to vote on the ontology of subjects used in the certificates they hold.
4. E-Ijaza should be able to represent the evaluation data provided by Maan Al-Dabbagh.

Work Plan

Week 1-2 (1st - 14th November)

- Setup Sovrin locally and create users, agents and certificates.
- Investigate Sovrin's features, including how to issue certificates.

Milestone: Development environment ready

Week 3-4 (15th - 28th November)

- Add functionality for issuing and verifying certificates.
- Add functionality for managing master certificates.

Milestone: can verify certificates using chain of trust.

Week 5-6 (29th November - 12th December)

- Define a subject hierarchy, and use it for issuing certificates and managing master certificates.
- Start working on a system for narrowing and merging certificates.

Milestone: certificates have subjects/topics that are used in verification.

Week 7-9 (13th December - 2nd January)

- Complete system for narrowing and merging certificates.
- Build system for managing flexible subject hierarchy.
- Start extracting Ijaza data from the dataset, and put it into an appropriate format.

Milestone: Can narrow and merge certificates based on fixed subject hierarchy and can manage a flexible subject hierarchy.

Week 10-11 (3rd - 16th January)

- Adjust systems to use flexible subject hierarchy.
- Ensure the necessary Ijaza data is taken from the dataset and put into an appropriate format.

Milestone: Systems use flexible subject hierarchy. Ijaza data is ready for use during testing.

Week 12-13 (17th - 30th January)

- Allow some spare time to catch up if I've fallen behind.
- Do project review.

Milestone: Project review Complete

Week 14-15 (31st January - 13th February)

- Built user interface.
- Work on introduction to the dissertation.

Milestone: Core project complete.

Week 16-18 (14th February - 6th March)

- Add any missing tests.
- Build any tools and create and test data required to complete full testing of the system.
- Do testing with multiple computers and Ijaza data.
- Fix any bugs.
- Finish introduction and preparation chapters of the dissertation.

Milestone: Full test done on complete implementation

Week 19-21 (7th - 27th March)

- Complete draft of dissertation and send it to supervisors and Director of Studies.
- Repair and refine implementation.

- Work on any extensions (time permitting).

Milestone: Draft dissertation sent.

Week 22-25 (28th March - 24th April)

- Refine dissertation using feedback and keep re-sending and improving.

Milestone: Dissertation complete and ready to submit.

Week 26-27 (25th April - 8th May)

- Spare time.

Milestone: Submit dissertation and code 1 week early.

Resource Declaration

- My laptop: 2019 16” MacBook Pro - 2.6GHz intel core i7 CPU with 16GB of RAM and 512GB SSD.
    - All my work will either be in a private GitHub repository or on my University account’s Google Drive
    - I will use the MCS facilities if something goes wrong with my laptop
- I will use multiple computers (either on the MCS or in the intel lab) to test the distributed aspect of the produced systems.
- I will be using the open-sourced code for the Sovrin Network.
- I will be using the Ijaza data sent to me by Maan Al-Dabbagh in my evaluation.