A DID wallet implementation of did:web

Panagiotis-Christos Kyrmpatsos *September, 2022*

Athens University of Economics and Business

School of Information Sciences and Technology
Department of Informatics
Athens, Greece

Bachelor Thesis
in
Informatics

A DID wallet implementation of did:web

Panagiotis-Christos Kyrmpatsos
3180226

Supervisor: Professor George C. Polyzos

September, 2022

Panagiotis-Christos Kyrmpatsos
A DID wallet implementation of did:web
September, 2022
Supervisor: Prof. George C. Polyzos

Athens University of Economics and Business
School of Information Sciences and Technology
Department of Informatics
Mobile and Multimedia Laboratory
Athens, Greece

Abstract

Currently, the world relies on identifiers from intermediaries such as Facebook, Google, email providers, and government agencies to connect us. This reality holds a number of concerns for our privacy and data gathered by those parties. In most cases, the handling and collection of data generated by those interactions are beyond the control of individuals. Decentralised Identifiers (DIDs) are globally, unique and persistent identifiers, which aim to provide a solution to those challenges. Since their formulation, a number of innovations have allowed the creation of private and secure peer-to-peer connection between two parties in a decentralised manner utilising diverse DID methods. With DIDs, it is possible for each party -individual or organisation- to create different DID documents depending on the context and digital relationship. This prevents data correlation on each interaction.

Possibly the most important aspect of DIDs lies in the fact that they are entirely controlled by the identity owner (controller). DIDs can be used independent of centralised registries, authorities, or identity providers. The aim of this project is to implement a new DID method (did:web) proposed in 2021 which attempts to bootstrap trust using a web domain's existing reputation. We design and implement a web server capable of creating, storing, and resolving DID credentials utilising the did:web method.

Περίληψη

Από τα πρώτα βήματα του Ίντερνετ η διαχείριση της ψηφιακής μας ταυτότητας έχει αποτελέσει μια συνεχή πρόκληση. Σήμερα, βασιζόμαστε κυριώς σε διαμεσολαβητές για την διαχείριση της, μεταξύ των οποίων μεγάλες εταιρείες (πχ Facebook, Google), πάροχοι email και κυβερνητικές υπηρεσίες. Αυτή η πραγματικότητα συνοδεύεται από μια σειρά δυνητικών προβλημάτων αναφορικά με την διαχείριση των προσωπικών δεδομένων των χρηστών. Η τεχνολογία των αποκεντρωμένων αναγνωριστικών (decentralised identifiers) δίνει απάντηση στους προβληματισμούς που προκύπτουν από αυτή την πραγματικότητα. Από την πρώτη υλοποίηση τους μια σειρά από καινοτομίες έχουν καταστήσει δυνατή την ύπαρξη αποκεντρωμένης και ασφαλούς ομότιμης επικοινωνίας μεταξύ δύο μερών χρησιμοποιώντας διάφορες μεθόδους. Με τη χρήση των αποκεντρωμένων αναγνωριστικών είναι εφικτή η δημιουργία ξεχωριστών εγγράφων για κάθε μέρος -άτομα ή οργανισμοί- ανάλογα με τις απαιτήσεις της εκάστοτε ψηφιακής σχέσης.

Η σημαντικότερη ίσως πτυχή των αποκεντρωμένων αναγνωριστικών αφορά την δυνατότητα καθολικού ελέγχου που δίνουν στον ιδιοκτήτη τους, δίχως την εξάρτηση από κάποια κεντρική οντότητα. Αυτό σημαίνει πως με την χρήση εγγράφων που βασίζονται σε αποκεντρωμένα αναγνωριστικά ο κάτοχός τους είναι εξόλοκλήρου υπεύθυνος για τον διαμοιρασμό, την κατοχή και τη διαγραφή τους δίχως την εμπλοκή κάποιου τρίτου μέρους. Στόχος του έργου μας είναι η υλοποίηση ενός συστήματος έκδοσης και διαχείρισης τέτοιων αναγνωριστικών που χρησιμοποιούν τη μέθοδο did:web η οποία προτάθηκε το 2021. Βασικός σκοπός της μεθόδου είναι η αξιοποίηση της υπάρχουσας εμπιστοσύνης σε συγκεκριμένους ιστοτόπους ώστε να επιταχυνθεί η υιοθέτηση των αποκεντρωμένων αναγνωριστικών ως μέσο ψηφιακής ταυτότητας.

Acknowledgement

Firstly, I would like to express my sincere gratitude to my supervisor, Professor George C. Polyzos. His support, guidance and overall insight have made this an inspiring experience for me.

Furthermore, I would like to thank PhD Candidate Iakovos Pittaras for the technical advice and knowledge he provided me with ,from the accurate choice of framework to the formatting of the final thesis. Lastly, would like to extend my sincere thanks to Researcher Dr. Nikos Fotiou for his punctual remarks and valuable knowledge regarding DIDs.

Contents

1 Introduction

A new era regarding identity management, digital claims, and authentication is currently underway by harnessing the power of Decentralised Identifiers (DIDs). A DID is a new type of identifier that enables verifiable, decentralised digital identity. Although DIDs are at their heart Universal Resource Identifiers (URI), nonetheless they have enabled a new paradigm to take hold, that of Self-Sovereign Identity (SSI). It is our firm belief that the impact of SSI technology will be profound. The importance of SSI infrastructure will be accompanied by a shift in control from a centralised/federated model to a Self-Sovereign model regarding digital identity.

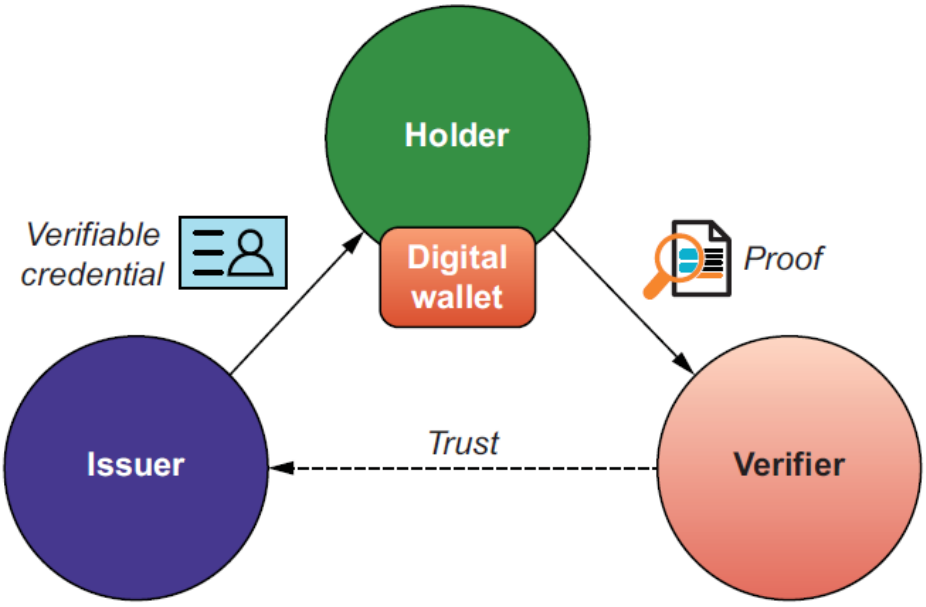

Fig. 1.1: The three core roles in the Verifiable Credentials trust triangle [Ale21, p. 41].

In this thesis, we explore how a recently proposed type of DID, did:web, can be used to permit the issuance and management of digital claims. It is the purpose of this thesis to explore the benefits offered by the new type of DID and to build a platform that further enables interaction and management regarding digital identity, thereby facilitating the transition to a Self-sovereign model of digital identity.

1.1 Motivation and Problem Statement

As of September 2022, there exist only a handful of implementations that can handle the did:web type of identifier. Since this method is newly proposed it would be both of technical and practical interest to explore the strengths and weaknesses presented in a digital identity management system that implements it. We also carefully examined the implementation of a did:self method presented here: [Fot+22].

1.2 Thesis Structure

Chapter 2

In this chapter, we turn our focus on the background information and relevant technologies that were used in our project. There exist three sections each elaborating on a different aspect of our work. Section 2.1 briefly explains the idea and motivation behind DIDs. Section 2.3 touches on the did:web method, the basis of our implementation. Certain strengths and weaknesses of this method are also analysed. Lastly, Section 2.3 presents and justifies the choice of the tools we used in our implementation regarding web technologies.

Chapter 3

In chapter 3, we introduce our did:web wallet architecture and also supply relevant UML diagrams. We present the system with its various functionalities. In this chapter, we also justify how our system complies with the operations defined in the did:web method specification.

Chapter 4

The focus of this chapter lies in evaluating the proposed implementation (see 4.1). Security strengths and possible weaknesses are mentioned and compared with other DID methods (see 4.2).Lastly,a series of possible future improvements are explored regarding the support of more DID methods alongside upgrades relevant to its security status (see 4.3).

Chapter 5

This chapter concludes our work.

2 Background and Related Work

In this section, several relevant technologies will be presented in brief to facilitate the understanding regarding the implementation of our system. We will focus on the use and significance of DIDs and on some other tools we used.

2.1 Decentralised Identifiers

Decentralised Identifiers (DIDs) are a new type of identifier under standardization by W3C that enables verifiable, decentralised digital identity. A DID can refer to any subject. A DID subject can refer to any entity. Examples include persons, organizations, things end even data model or abstract entities [22a]. The main attribute differentiating DIDs from other proposed solutions to identity management rests on their design which enables decoupling from centralised registries or identity providers. Particularly, DIDs enable their controller to "prove control over it without requiring permission from any other party" [22a].

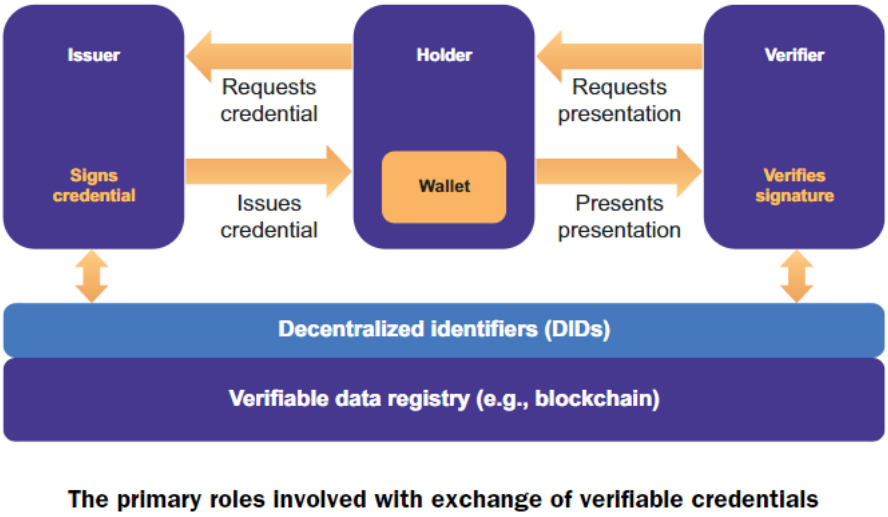

Fig. 2.1: DID overview [Ale21].

URIs are used by individuals and organizations in a variety of contexts. For example, URLs are a type of URI used every time we access a Web page. However, usage of a

URI differs from sovereignity over it. The majority of URIs that we use are not under our control. They are issued by external authorities and can be revoked at any time. Likewise, they may cease to be valid or used and distributed without our consent. In more extreme cases, they can be deliberately replicated and asserted by a malicious third-party (identity theft).

DIDs are a new type of globally unique identifiers. Their generation and usage is entity-controlled and authenticated using cryptographic proofs, such as digital signatures. Each entity can have an unlimited amount of DIDs enabling efficient management of multiple identities based on the context of each interaction. In simpler terms, DIDs allow interactions with other people, institutions, and systems that "require entities to identify themselves...while providing control over how much personal data should be revealed, all without depending on a central authority to guarantee the continued existence of the identifier" [22a].

DIDs therefore include all the information necessary to ensure efficient identification solely relying on the information available in them. DID resolution is responsible for the transformation of the DID to a DID Document.

There exist two types of DIDs: Private and Public. Private DIDs can be exchanged between two parties to create a secure channel that no one else is privy to. This means no third party has knowledge of what happens across that channel or who is behind it. It is possible to create as many separate DIDs for as many separate relationships as one sees fit to prevent correlation of one's private information, without relying on a single central authority. Public DIDs on the other hand are used when a subject wants to be publicly identifiable (e.g., a government office issuing passports). Public DIDs are a good way to initiate the interaction between private parties, who afterwards utilize private DIDs.

The did:web:example.com simple DID is resolved to a DID Document shown in the listing below.

```
{
    "@context": [
      "https://www.w3.org/ns/did/v1",
      "https://w3id.org/security/suites/jws-2020/v1"
    ],
    "id": "did:web:example.com",
    "verificationMethod": [
      {
        "id": "did:web:example.com#key-0",
        "type": "JsonWebKey2020",
```

```
11            "controller": "did:web:example.com",
12            "publicKeyJwk": {
13              "kty": "OKP",
14  ad "crv": "Ed25519",
15              "x": "0-e2i2_Ua1S5HbTYnVB0lj2Z2ytXu2-tYmDFf8f5NjU"
16            }
17          },
18          {
19            "id": "did:web:example.com#key-1",
20            "type": "JsonWebKey2020",
21            "controller": "did:web:example.com",
22            "publicKeyJwk": {
23              "kty": "OKP",
24              "crv": "X25519",
25              "x": "9GXjPGGvmRq9F6Ng5dQQ_s31mfhxrcNZxRGONrmH30k"
26            }
27          },
28          {
29            "id": "did:web:example.com#key-2",
30            "type": "JsonWebKey2020",
31            "controller": "did:web:example.com",
32            "publicKeyJwk": {
33              "kty": "EC",
34              "crv": "P-256",
35              "x": "38M1FDts7Oea7urmseiugGW7tWc3mLpJh6rKe7xINZ8",
36              "y": "nDQW6XZ7b_u2Sy9slofYLlG03sOEoug3I0aAPQ0exs4"
37            }
38          }
39        ],
40        "authentication": [
41          "did:web:example.com#key-0",
42          "did:web:example.com#key-2"
43        ],
44        "assertionMethod": [
45          "did:web:example.com#key-0",
46          "did:web:example.com#key-2"
47        ],
48        "keyAgreement": [
49          "did:web:example.com#key-1",
50          "did:web:example.com#key-2"
51        ]
```

```
52      }
```

Listing 2.1: The DID document of the did:web:example.com.

A basic overview of the major components of a DID are discussed below.

DIDs and URLs

A DID consists of three parts: the scheme did:, a method identifier. e.g. ethr, key, ion or web, and a unique method-specific identifier as specified by the did method.

Fig. 2.2: An example of a DID.

DID subject

Each DID identifies an entity. This entity is known as the DID subject. In most cases , the DID subject is also the DID controller.

DID controller

The DID controller is the entity responsible for creating and modifying the DID. This ability is usually asserted through the control of a set of cryptographic keys. A DID can have more than one controllers.

DID Document

A DID Document contains information associated with a DID. At the very least, the DID Document contains the ID of the document. Commonly verification methods, such as cryptographic public keys, and services relevant to interaction with the DID subject are also included. A DID Document can be serialised into a byte stream. The use of choice in our project is that of a JSON file accessible through a URL.

DID Resolver

A DID resolver is a system that takes a DID as input and produces the relevant DID Document. Our system allows DID resolution of did:web DIDs. The steps for resolving a DID to its respective DID Document are defined in each did method specification.

2.2 The did:web method

One major issue regarding the adoption of DIDs lies in "bootstrapping enough meaningful trusted data around identities to incentivize mass adoption" [22d]. The did:web method proposes a new way to utilise DIDs allowing them to bootstrap trust using a web domain's existing reputation.

Target system

The target system of the web:did method is the web host that the domain name described by the DID resolves to when queried through the DNS. A DID must use the prefix did:web so as to conform to the specification.

Method specific identifier

The method specific identifier must be a fully qualified domain name that is also secured by a TLS/SSL certificate with an optional path to the DID document. The identifier shall match the common name used in the TLS/SSL certificate and shall not include IP addresses. The identifier can include a port by using a percent encoded colon to prevent conflict with paths. Directories can also be targeted and delimited by colons rather than slashes. Some examples include:

- did:web:w3c-ccg.github.io
- did:web:w3c-ccg.github.io:user:alice
- did:web:example.com%3A3000

2.3 Web development tools

Flask web framework

Flask is a micro web framework written in Python. It does not have a database abstraction layer as is the case with other web frameworks. However, Flask does support extensions to provide such functionality. It depends on Jinja and Werkzeug and has become one of the most popular web application frameworks. Flask, Jinja and Werkzeug are all part of The Pallets Project.

Wekzeug

Werkzeug is an utility library for the Python programming language for Web Server Gateway Interface applications. Werkzeug is used for instantiating objects, which are subsequently used in requests, responses, and some utility functions.

Jinja

Jinja2 is a full-featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed. It also features automatic HTML escaping system for preventing cross site scripting attacks. Jinja provides Sandboxed execution mode. Every aspect of the template execution is monitored and explicitly whitelisted or blacklisted, whatever is preferred.

SQLAlchemy

SQLAlchemy is an open-source SQL toolkit and object-relational mapper (ORM) for Python. It is most famous for its object-relational mapper (ORM), an optional component that provides the data mapper pattern, where classes can be mapped to the database in open ended, multiple ways - allowing the object model and database schema to develop in a cleanly decoupled way from the beginning. Our project utilizes this feature.

Bulma

For the front-end component of our work the Bulma library was used. Bulma is a free, open-source CSS library. This means that Bulma provides the developer with CSS classes to help with creating a responsive and aesthetically pleasing website. Since Bulma comprises solely of CSS classes, the HTML code one writes has no impact on the styling of the page.

SQLite

SQLite is a database engine written in the C programming language. It is not a standalone app; rather, it is a library that software developers embed in their apps. As such, it belongs to the family of embedded databases. It is the most widely deployed database engine, as it is used by several of the top web browsers, operating systems, mobile phones, and other embedded systems. It generally follows PostgreSQL syntax, but does not enforce type checking by default. This means that one can, for example, insert a string into a column defined as an integer.

SQLite stores the whole database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine, allowing several processes or threads to access the same database concurrently. It implements this simple design by locking the database file during writing [SQL].

3 System Design and Implementation

In this section, we introduce our did:web credentials system. We shall present the basic architecture of the system and provide reasons for choosing the specific scheme and tools. Subsequently, we provide a detailed workflow of our implementation of the system, which comply to the Method specification requirements [22d].

3.1 Design

Our implementation is comprised of two basic components. The front-end enables the controller to create, modify, and resolve did documents with specific public keys. The front-end was built using Bulma framework to handle the CSS part. The back-end ,in conjuction with the database, handles the storage and authentication. For the back-end Flask and SQLite were used. The particular choices were made after taking into consideration the time requirements of our project, as well as to facilitate its maintenance and re-usability to potentially other projects. For this reason our project utilises Flask's Blueprints.

Flask uses a concept of blueprints for making application components and supporting common patterns within an application or across applications. A Blueprint object works similarly to a Flask application object, but it is not actually an application. Rather, it is a blueprint of how to construct or extend an application. In our application, we have separated the main functionality from the authentication functionality as two separate blueprints making it easier to upgrade the login options using OAuth 2.0 [22e] later in time.

A use case diagram demonstrating the basic functionality of our implementation is shown in Figure 3.1. In the use case diagram the separation between the authentication and the main module can be observed.

Our SQLite DB is constructed with two tables holding the necessary information for our project. One Table stores the credential and personal information of each user (Table User). A second table is responsible for storing and updating the JSON

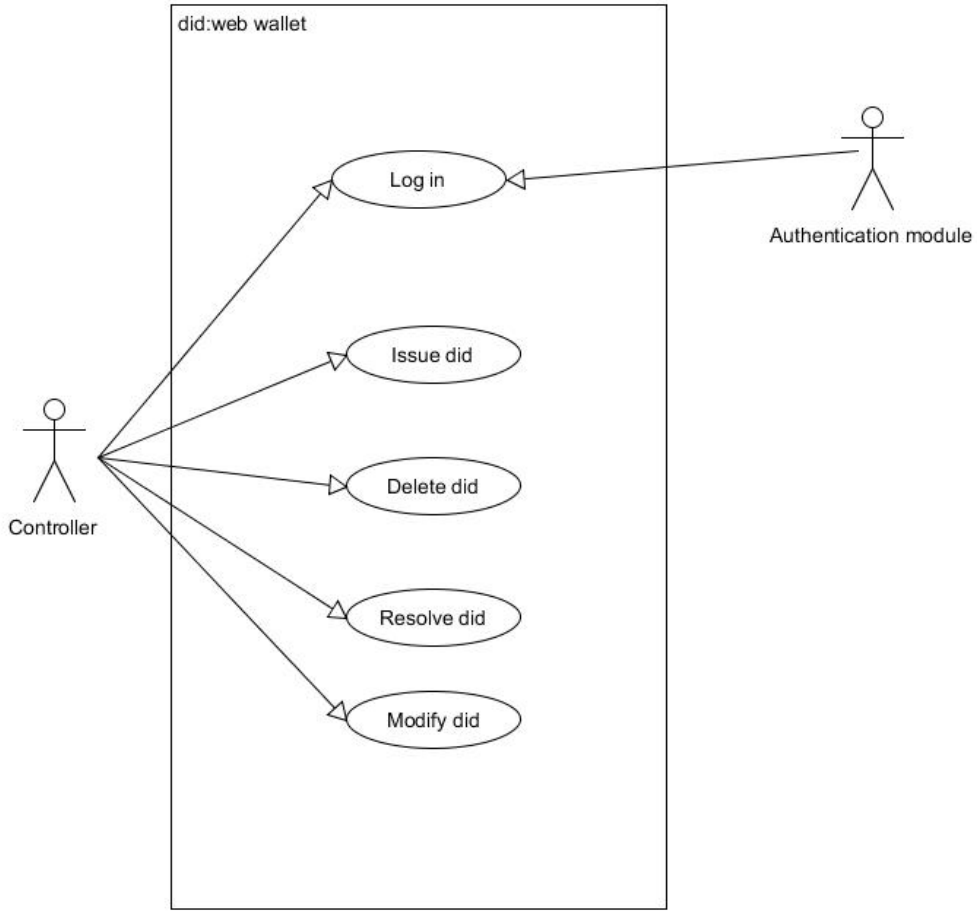

Fig. 3.1: Use case diagram.

documents used as DID documents. An overview of the class relationships is shown in Figure 3.3.

3.2 Implementation

3.2.1 The DID document

Our DID Document in did:web can include any of the properties defined by the DID specifications, and it is encoded using JSON.We follow the did:web method specification requirements for the following operations[22d]. A DID Document in our system includes the following properties:

1. *@context*: A list of relevant URLs dictating how the document data need to be read.

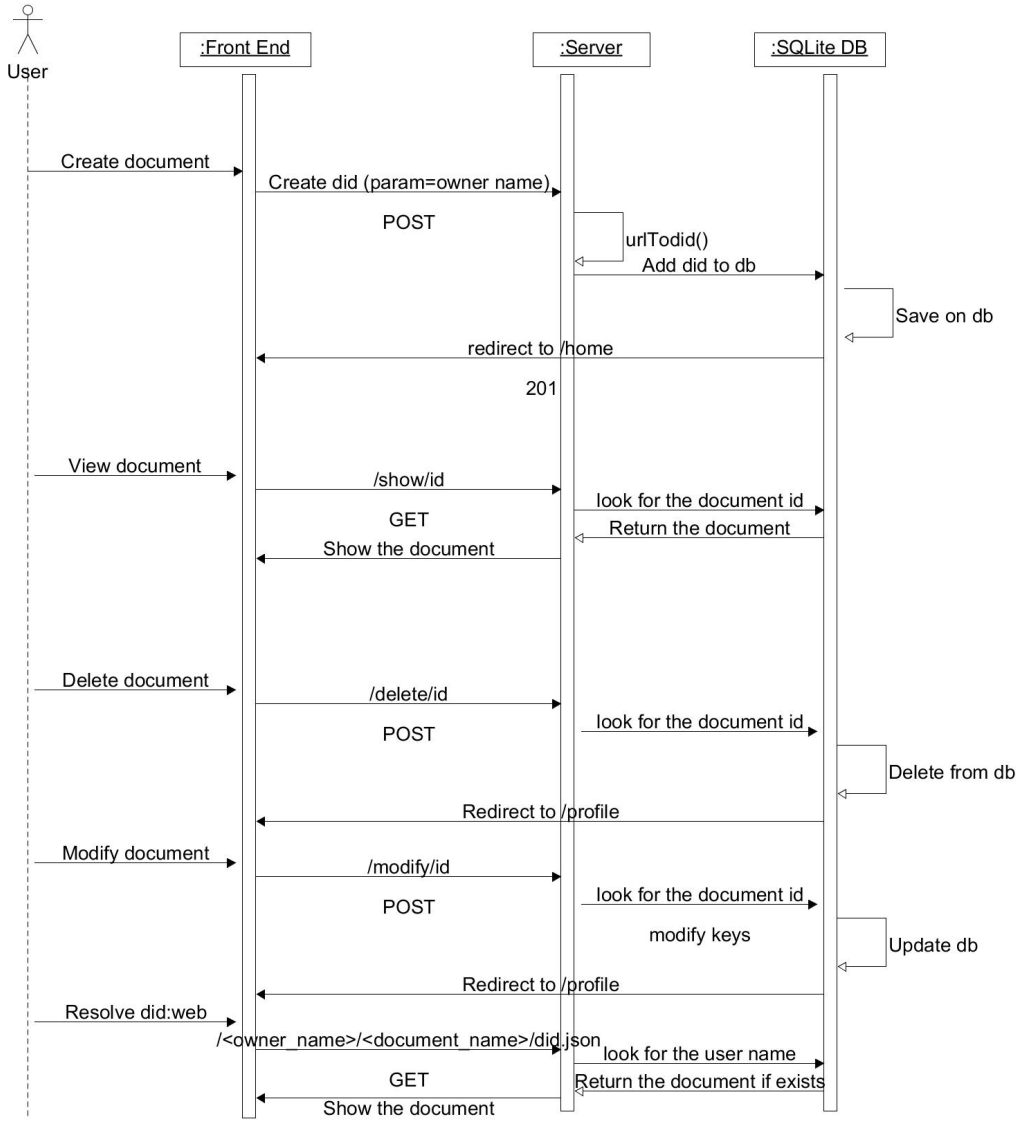

Fig. 3.2: Sequence diagram demonstrating basic actions.

2. *id*: The DID which the document concerns.

3. *verificationMethod*: A list of public keys expressed using the "JsonWebKey2020" notation. Each key in the list is identified by an ID.

4. *authentication*: A list of public keys by which an entity can cryptographically prove that they are associated with the DID.

Each public key included in the verificationMethod property is identified by a unique id. Consequently, there cannot be two keys with the same ID for the same did:web DID document.

3.2.2 Create(Register)

Since there is no HTTP API specified for the did:web method [22d] each implementation has the freedom to choose how to manage the creation of a DID document. Generally, three steps are involved:

1. Applying at a domain name registrar for use of a domain name.

2. Storing the location of a hosting service, the IP address at a DNS lookup service.

3. Creating the DID document including a suitable key-pair and storing the did.json file under the well-known URL to represent the entire domain.

For example the domain name example.com:user:mike, the did.json will be available under the following URL, https://example.com/user/mike/did.json .

3.2.3 Read(Resolve)

The following steps MUST be executed to resolve the DID document from a Web DID:

1. Replace ":" with "/" in the method specific identifier to obtain the fully qualified domain name and optional path.

2. If the domain contains a port percent decode the colon.

3. Generate an HTTPS URL to the expected location of the DID document by prepending https://.

4. If no path has been specified in the URL, append /.well-known.

5. Append /did.json to complete the URL.

6. Perform an HTTP GET request to the URL using an agent that can successfully negotiate a secure HTTPS connection

Our implementation utilises the resolve.py Python file to perform this sequence.

3.2.4 Update

To update the did.json Document the controller can modify the public keys utilised. It is also possible to choose which keys we want to utilize for authentication.

3.2.5 Deactivate(Revoke)

To delete the DID document, the did.json can be removed through deletion of the relevant JSON document from the database. This also results in the removal of the relevant URL. Only the controller of the Document after the login process is able to delete the relevant Document.

3.2.6 Authentication and Authorization

The did:web method specification does not specify any authentication or authorization mechanism for writing, removing, or creating the DID Document. Our implementation utilises the Flask-Login module for user session management. With the aid of Flask-Login, we handle the common tasks of logging in, logging out, and remembering users' sessions over extended periods of time. Flask-Login stores the active user's ID in the Flask Session. We also utilize it to restrict views to logged-in (or logged-out) users. In more detail , while the did.json file is accessible to everyone, operations on the DID Document can only be performed by the Document controller. Moreover, one needs to be logged in to create or delete a document. Our implementation also supports the "remember me" functionality.

The main properties utilised and implemented in our project include:

- is_authenticated: This property returns True, if the user is authenticated i.e. they have provided valid credentials.
- is _active: This property should return True, if this is an active user- in addition to being authenticated, they also have activated their account, not been suspended, or any condition your application has for rejecting an account. Inactive accounts may not log in.
- get_id(): This method must return a str that uniquely identifies this user and can be used to load the user.

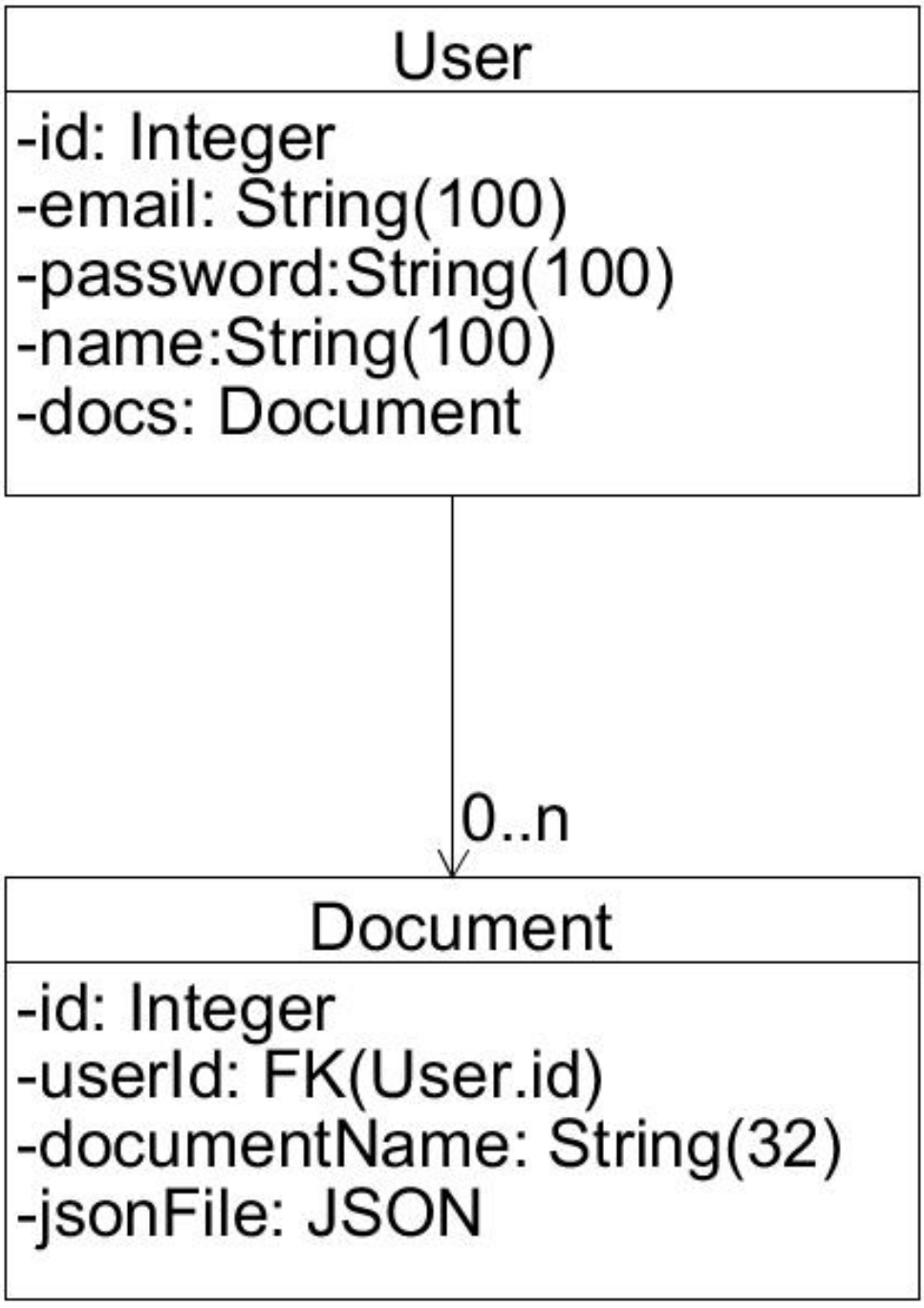

Fig. 3.3: Class diagram of our application.

4 Evaluation and Future Work

In this chapter, we evaluate the prototype system.

4.1 Qualitative Evaluation

Our system provides the basic functionality outlined in the did:web Method specification. It can handle sufficiently the creation and modification of a series of DID documents. Moreover, it provides each user with the possibility of having active and modifying a series of documents under a URL prepended by his username and followed by the document name.

In terms of security, our application relies on the security measures put in place by Flask-Login as described earlier. This provides sufficient protection to the user. Response times of our application are satisfactory even when multiple users are logged in simultaneously.

4.2 Security evaluation

4.2.1 Security comparison over other DID methods

When compared with other DID methods, did:web implementations differ from a security aspect since they rely inherently on the security of the website the DID Document is hosted on. In contrast,did:ethr , for example, relies on the ethr-did-registry and its security is secured by the decentralised application responsible for the creation and management of DID documents[22b]. did:self is another DID method recently explored by [Fot+22] which assures that a " DID Document is correct even if it is retrieved over an insecure channel". The did:key method [22c] also does not rely upon a centralised registry or Decentralised Ledger Technologies to transmit information regarding the relevant DID Document. This is achieved by the expansion of a cryptographic public key into a DID Document therefore protecting it from security issues which arise from the utilisation of the did:web method.

4.2.2 DNS Security Considerations

Overall, DNS may present some attack vectors that enable active security and privacy attacks on the did:web method [22d]. To mitigate those risks proper configuration of the DNS is needed. An example of such an attack-if DNS over HTTPS is not enabled-would be the interception of the DNS resolution via a Man in the Middle attack thus returning an incorrect DID Document retrieved from a malicious server [22d].

Spoofed DNS records constitute another possible security issue. In this case, a record returned by a malicious DNS Server is compromised and allows the record to be pointed at a malicious server containing different DID Documents. A possible solution to mitigate this risk comes in the form of DNSSEC.

Lastly, since did:web relies on DNS in order to perform the resolution of the DID Document all resolutions can potentially be tracked by a DNS provider [22d]. Furthermore, since the DID Document is stored on a web-server, each time a DID Document is retrieved, the web server has the ability to track the origin of the GET request. To combat the possibility of the party requesting the DID Document being tracked, usage of a trusted universal resolver is strongly encouraged. Alternative solutions include utilizing a VPN service or performing a resolution over the TOR network[22d].

4.3 Future Work

A series of improvements can be incorporated into our implementation to extend its functionality and interoperability. In this spirit, enabling the resolution and creation of DID documents which utilise different methods seems a reasonable first step. These could include methods which utilise different types of verifiable data registries such as Ethereum or Hyperledger. For that reason, implementation of method-specific resolvers is necessary for handling the documents. It is worth mentioning that the handling of those credentials would require the introduction of new modules in our system.

Furthermore, it would be useful to implement account recovery features in case some user loses their login credentials. Enabling OAuth 2.0 [22e] would also be useful regarding user registration. This would permit users to create accounts without the need to create a separate account to use the services provided by our system.

Lastly, we look forward to hosting our project on the university lab server.This action will enable use of DNSSEC tools. After the initial configuration a series of tests will

take place to ensure that all features are enabled and delivery times of services are acceptable. It would also be a good idea to subject our implementation to certain penetration testing in order to gauge for potential issues with the security.

5 Conclusion

DIDs seem to be the turning point in identity management facilitating the transition from federated identity management systems to a decentralised identity model. The SSI paradigm shift happening right now is deeper than a technology shift-it is a shift in the underlying infrastructure and dynamics of the Internet itself. With data protection and identity management becoming an important concern for an increasing number of individuals, it is apparent that relying on third parties like Facebook and Google to handle our online identities comes with many thorny issues. DIDs may prove a valuable solution to the challenges involved.

By leveraging the familiarity of the average Internet user, we attempt to implement a platform that makes it easy to use and issue DID Documents. Our project aspires to create a platform capable of issuing and managing DID Documents after creating a user account. Our solution uses the did:web method, which is a DID method that does not impose particular technology for implementing the registry that maintains the DID Document. The current implementation uses a database as our data registry. Future work in this area includes the investigation of alternative DID methods as well as improvements in the security of the proposed system.

Bibliography

[22a] *Decentralized Identifiers (DIDs) v1.0 Core architecture, data model, and representations*. 2022. URL: https://www.w3.org/TR/did-core/ (visited on July 19, 2022).

[22b] *did:ethr Method Specification*. 2022. URL: https://github.com/uport-project/ethr-did (visited on Oct. 4, 2022).

[22c] *did:key Method Specification*. 2022. URL: https://w3c-ccg.github.io/did-method-key/ (visited on Oct. 4, 2022).

[22d] *did:web Method Specification*. 2022. URL: https://w3c-ccg.github.io/did-method-web/ (visited on Aug. 4, 2022).

[22e] *OAuth 2.0*. 2022. URL: https://oauth.net/2/ (visited on Oct. 1, 2022).

[Ale21] Alex Preukschat and Drummond Reed. *Self-Sovereign Identity*. Manning Publications Co., 2021.

[Fot+22] Nikos Fotiou, Vasilios A. Siris, George Xylomenos, and George C. Polyzos. "IoT Group Membership Management Using Decentralized Identifiers and Verifiable Credentials". In: *Future Internet* 14.6 (2022).

[SQL] SQLite. *SQLite lemma — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-September-2022].

List of Acronyms

API Application Programming Interface

DID Decentralised Identifier

DLT Decentralised Ledger Technologies

DNSSEC Domain Name System Security Extensions

HTTP Hyper Text Transfer Protocol

JSON JavaScript Object Notation

SQL Structured Query Language

SSI Self-Sovereign Identity

URI Uniform Resource Identifier

URL Uniform Resource Locator

List of Figures