Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Mikkel Aas
Magnus Gluppe
Erlend Johan Ringen Vannebo

Using Self-Sovereign Identity and Verifiable Data to improve public services

Bachelor's thesis in Computer Science
Supervisor: Mariusz Nowostawski
May 2022

NTNU
Kunnskap for en bedre verden

Mikkel Aas
Magnus Gluppe
Erlend Johan Ringen Vannebo

Using Self-Sovereign Identity and Verifiable Data to improve public services

Bachelor's thesis in Computer Science
Supervisor: Mariusz Nowostawski
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Abstract

Symfoni, a Norwegian company specializing in verifiable data, wanted to make a demo for NAV that showcased how EU Digital Wallet could be used when applying for unemployment benefits. The end product was an application that runs on the web and a smartphone that allows users to control their verifiable data. It is essential to state that none of the sensitive verifiable data used in the application is real. Therefore it can not be used in a real-life scenario. It is, however, a good demo for showcasing the flow of data in the Self-Sovereign Identity system.

Sammendrag

Symfoni ønsket å lage en demo for NAV som kunne vise hvordan EU Digital Wallet kan bli tatt i bruk når én søker om dagpenger. Resultatet ble en applikasjon som kjører i nettleseren og på mobilen, der brukeren er i kontroll over sin egen verifiserbare data. Det er viktig å understreke at den sensitive verifiserbare dataen i prosjektet ikke er ekte, og kan dermed ikke brukes i en ekte situasjon. På en annen side er det en god demo for å demonstrere flyten av data i et egen-kontrollert identitet system.

Preface

This bachelor thesis is written by Mikkel Aas, Erlend Vannebo, and Magnus Gluppe for the Department of Computer Science at NTNU in Gjøvik.

First, we want to extend our sincere gratitude to our client and mentor, Jon Ramvi, CEO and founder of Symfoni. Jon and the other developers from Symfoni have provided invaluable input for this thesis, both from a theoretical and technical standpoint. He has put us in contact with NAV and different professionals to help during development. Second, we would like to thank Elin Vethe, our contact person, when collaborating with NAV, Norway's Work- and Welfare Management. She always showed great enthusiasm in our meetings and asked important questions that helped us improve the application. She provided an outsider's perspective and had the knowledge regarding the inner workings of NAV that the group required. Third, we would like to thank Mariusz Nowostawski, associate professor for NTNU, and our supervisor for the thesis. Mariusz has helped us traverse the university end of the thesis with the project's scope and writing the thesis itself.

Finally, we would like to thank our family, friends, and fellow students for being our support structure throughout this process. They have helped with proofreading the report and encouraging us along the way.

Repository

The code for this project can be found at NTNUs Git in a public repository. Inside the repository, there is a readme that will explain how to start the demo:
`https://git.gvk.idi.ntnu.no/Mikkelaa/bachelor`

Contents

Figures

Tables

Code Listings

Acronyms

API Application Programming Interface. 2, 13, 16, 18, 29, 32, 59, 61, 63, 66, 67, 78, 79, 83, 94

DID Decentralized Identifier. 8–14, 16, 17, 19–25, 32–34, 37, 38, 42, 44, 48, 50, 61–64, 68–71, 86, 88, 94, 96

EBSI European blockchain Service Infrastructure. 21

EU European Union. 2, 21, 35

GUI Graphical User Interface. 16, 30, 46

JSON JavaScript Object Notation. 67, 78

JSON-LD JavaScript Object Notation for Linked Data. 9, 32, 91

JWT JSON Web Token. 44, 57, 64, 66, 85

NAV Norway’s Work- and Welfare Management (Arbeids- og velferdsforvaltningen). 1, 2, 5, 8, 15, 19–21, 27, 31, 35, 42, 48, 53, 54, 61, 64, 74, 76, 77, 89–91, 93, 94, 96

NPM Node Package Manager. 59

NTNU Norwegian University of Science and Technology. vii, 2, 27

PKI Public Key Infrastructure. 13

SQL Structured Query Language. 59, 61, 74, 79

SSI Self-Sovereign Identity. xv, 2, 3, 5, 7, 8, 13–21, 24, 27, 30, 36, 40, 43, 44, 51–54, 57, 59, 61, 63, 78, 89, 91, 93–96

URL Uniform Resource Locator. 83

VC Verifiable Credential. 8, 13, 17, 19, 20, 24, 27–29, 31, 33, 37, 39, 42, 43, 47, 48, 52, 54, 64, 79, 91

VP Verifiable Presentation. 19, 20, 28, 37, 41–43, 47

Glossary

blockchain Is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network.. 1, 9–12, 21, 30

decentralized identifier Is a new type of identifier that is globally unique, resolvable with high availability, and cryptographically verifiable.. 8, 9

Ethereum Ethereum is a decentralized, open-source blockchain with smart contract functionality. 9, 88, 96

veramo Is a bundle of performant and modular APIs for Verifiable Data and SSI. 3, 13, 14, 16–18, 28–30, 59–61, 63, 64, 66, 67, 74, 78, 79, 92

verifiable credential Is a tamper-evident credential that has authorship that can be cryptographically verified.. 7, 10, 11, 13–16, 23, 24, 27–30, 59, 63, 68–70, 74, 76, 77, 79, 81, 92, 94–96

verifiable presentation Expresses data from one or more verifiable credentials, and is packaged in such a way that the authorship of the data is verifiable. . 41, 70, 71, 74, 90

Chapter 1

Introduction

1.1 Background

There will be greater demand for digital solutions in modern society as more and more of our information is stored and exchanged digitally. With increased globalization, both culturally and in the workspace, it is important to set standards for storing and exchanging information. The EU Digital Wallet initiative aims to achieve a secure unified digital identification system for Europe. Ursula von der Leyen, the President of the European Commission on the EU Digital wallet: "Every time an App or website asks us to create a new digital identity or to easily log on via a big platform, we have no idea what happens to our data in reality. That is why the Commission will propose a secure European e-identity. One that we trust and that any citizen can use anywhere in Europe to do anything from paying your taxes to renting a bicycle. A technology where we can control ourselves what data is used and how." [1]. This digital wallet will contain all of one's credentials, from passports to university degrees or military service, and is meant to be used by EU citizens across EU borders to identify themselves and to make use of services all over the EU.

The bachelor project is done in collaboration with Norway's Work- and Welfare Management(NAV) and Symfoni, a private company specializing in SSI and blockchain technology. Today many application processes at NAV are being automated, but many are still performed manually and are time-consuming. This is due to the applicant having to collect documentation from different institutions, and an employee at NAV has to verify these. This process can be automated using SSI technology. In collaboration with Symfoni, we created a proof of concept solution to show NAV the possibilities behind this technology and how their application processes can be automated in the future.

1.2 Project Description

Our goal was to create a proof of concept demonstration of the EU Digital wallet workflow through the application of unemployment benefits at NAV. The project aims to use the principles of the EU Digital Wallet to create a system for digitally storing credentials. These credentials are used in NAV's application process. Mocked counterparts of real-life institutions will issue the information stored in this wallet to simulate a real scenario. First, we have a user agent representing the EU digital wallet, where all issued credentials will be stored. It is also decoupled from large centrally run databases. Then we have an agent representing the state who issues an identification credential. We have an agent representing Symfoni, a business issuing employment and termination credentials. Lastly, we have a NAV agent who is requesting these credentials, verifying them, and deciding whether someone applies for unemployment benefits or not.

1.3 Target audience

The goal of the bachelor thesis is twofold, creating a product to show proof of concept for SSI and writing a bachelor's thesis for NTNU.

The product is mainly meant for NAV and Symfoni to see if technology has potential and if it improves upon the current system. However, we hope other institutions and businesses can take inspiration from this product, find other use cases for this technology, and continue development on the subject.

The report is written for people who have a basic understanding of central concepts within computer science, such as familiarity with coding, databases, and APIs. The report is targeted toward fellow students, the supervisor, and the examiner. Students who are interested in the subject can use this report to learn more about the subject matter.

1.4 Constraints

1.4.1 Time

The project is a bachelor thesis, so certain time constraints had to be considered. Being inexperienced in the subject matter was a point of concern as it would take time to learn the concepts of SSI and to learn the additional tools and programming languages required before we could start developing the product. Having a finite amount of time for the project meant we had to manage it well and prioritize the most important aspects. Early on We made a progress plan where we sat milestones for ourselves to hit (see Appendix B). We did this to ensure that we progressed and did not get stuck for too long on specific problems. This ensured that we were not spending too much time on one area, such as reading so that we had time to develop the application and write the report.

1.4.2 Physical

The group scheduled three physical work sessions at the campus a week. During these, we worked with the current tasks laid out in the project plan: pre-project planning, developing code, writing the report, and preparing for the final presentation. We had digital sessions over discord when physical meetings were not scheduled, and these consisted of the same tasks as the physical sessions. Later in the development stage, it was more open for individual work, both on the codebase and the report. In addition to having meetings with the group, we had regular meetings with our faculty supervisor and employer. These meetings were held digitally over discord or Microsoft teams.

1.4.3 Legal and ethical

The biggest ethical issue with this product is handling private and sensitive information. The intended function is for different government agencies to communicate and pass personal data. We must ensure this data is not compromised and used for nefarious means. This project uses dummy data, though a real-life implementation would have to consider privacy concerns.

Both Symfoni and our bachelor group are committed to the concept of open-source software, and the code is available to be studied and used by all. Our goal was to make a practical and secure solution so that others could build upon what we create.

1.5 Group background

The group consists of 3 computer engineering students with experience with several programming languages and system development. Both node and JavaScript frameworks are used, and the group has experience using these tools. None of the members had any experience with the SSI infrastructure, so a considerable amount of time was used to learn both the theoretical and practical applications of the technology. The main unknown software that needed to be learned were DIDComm and Veramo.io

1.5.1 Responsibilities and roles

The main parts of the project were split into different categories—the primary responsibility of each category was delegated to one group member. Every group member had responsibility for the entire project and was expected to work on each part of the project; however, the person with the primary responsibility was responsible for doing the final quality check of the content being produced. This made us use our different strengths to our advantage and distribute the workload evenly between us when performing quality assurance.

1.5.2 Code

Erlend was in charge of Code quality. We tried to uphold a high and consistent standard, using good practice throughout the project. Erlend ensured we shared the same principles like high cohesion, low coupling, and sensible variable names in our code. The plan was to have a code review every time a sizable part was merged into dev, and the code master had a greater responsibility to make sure this happened.

Being in charge of the code quality was by far the most significant responsibility, as this included: code conventions and standards, continuous integration, pipeline, code quality, and testing. This was a big task for one person, so the code master had to delegate some of these responsibilities to make sure not too much fell on one person.

1.5.3 Documentation

Mikkel was in charge of the overall documentation for the bachelor thesis. This included documentation for code, the main report, and all the necessary attachments required. All members were expected to document the work; the documentation master was not meant to micromanage everything, but to ensure that the overall style and standards were upheld.

1.5.4 SCRUM

The Scrum master for this team was Magnus. His job was to ensure that every team member was familiar with the Scrum process and followed the appropriate guidelines. With agile development, it was easy to exclusively focus on writing code and neglect the documentation and planning parts of the project. Scrum is lighter on the planning and documentation part than more traditional development methods, but it is still essential to use the issue tracker and sprint reviews. The Scrum master ensured that the development process did not slide into chaos.

1.5.5 Manager

In addition to his role of overseeing the documentation, Mikkel had the responsibilities of a manager. This included ensuring deadlines are upheld, organizing meetings, ensuring information flow between the team members and external stakeholders, and the general morale of the team. There is some overlap between the Scrum master and the manager, but there was no coordination issue considering the team was small.

1.5.6 Symfoni and supervisor

Symfoni played an essential role in the project. Early on, they gave us learning materials on the subject area and willingly wanted to discuss the subject to help us understand. We had weekly meetings during the development process to get feedback on our progress, and technical requirements were further discussed. Additionally, technical meetings were encouraged by them, where they helped us with coding and gaining insight into using the technologies within SSI. Symfoni also served as a bridge between us, NAV and the industry.

Our supervisor played a more distanced role in our project. We had meetings less frequently, mostly when more significant changes had been made to ensure we were on the right track regarding the academic expectations for the thesis.

1.6 Report structure

The report consists of ten chapters. In the first part, we will be discussing the thesis problem and the technologies we will be using to solve it. In the second part, we will talk about the initial design and how we implemented it. Lastly, we are going to discuss the result.

1.6.1 Introduction

Introduces the project to the reader and sets a baseline for the reader to follow through with the report.

1.6.2 Theory and technology

The theory and Technology chapter provides the background necessary to understand the technology used in this project. SSI is a reasonably new technology, and some knowledge is required to understand the scenario this bachelor wants to showcase

1.6.3 Requirement specifications

The Requirement specifications chapter contains the task given to the group by Symfoni, the employer issuing the bachelor problem statement.

1.6.4 Design

The Design chapter consists of the overall design choices for the bachelor project. Furthermore, it explains the project's overall structure and the scenario we build the application demo around.

1.6.5 Front-end design

The Front-end design chapter goes through how the application looks and the thought process behind the decisions that were made. Every page of the application is documented with an image and a short text giving additional context.

1.6.6 Development process

The development process chapter explains how the group worked to achieve the goal. It covers everything from internal code reviews to meetings with external partners like Symfoni.

1.6.7 Implementation coding production

The implementation and coding production chapter are about how the demo was made. It uses code examples from both the back and front end of our demo to describe how the demo was written.

1.6.8 Testing and quality assurance

The testing and quality assurance chapter are about how we tested and quality assured our code, and it also has some reflection on what we would have done differently.

1.6.9 Conclusion

In the concluding chapter, we discuss our thoughts on the project. The chapter ranges from our thought on team cooperation to the state of the final demo.

Chapter 2

Theory and Technology

This chapter is about the core technology and theory fundamental to understanding how the project was done. It explains the most important building blocks of the Self-Sovereign Identity system. Furthermore, it has an in-depth problem description that extends on the theories and technologies discussed.

2.1 Self-Sovereign Identity

Self-sovereign identity is a new model for digital identity on the internet. Its goal is to streamline the identification process for websites, services, and apps where a user needs to establish a trusted relationship to access or protect private information. It is driven by new technologies and standards in cryptography, distributed networks, cloud computing, and smartphones.

The main idea is to shift from a centralized to a decentralized system. The old identity models rely on the communication between an issuer and a verifier. The new model aims to bridge this connection through the user. In other words, they are letting the user be in control of their own identity. [2, p. 3]

2.1.1 Trust triangle

The three core parts of the SSI model are the issuer, holder, and verifier. The issuer is the source of all credentials, and every credential has an issuer. For the most part, the issuers are organizations such as government agencies, financial institutes, universities, corporations, and many more. However, individuals or objects can also be issuers. An individual could issue a credential for power of attorney, and a sensor could issue a credential for an approved product.

Holders request verifiable credentials from the issuers and keep them in the holder's digital wallet. In most cases, the holder of a digital wallet will be an individual, and the holder may then present these credentials to any verifier. This step makes trusted interactions possible, as the verifier checks if the credential is from a trusted issuer. A government agency has a high level of trust, while an unknown startup company has a lower level of trust.

Verifiers can be a person, organization, or object seeking trust assurance about the subject's credentials. The verifier requests proof of a credential or a claim from a holder's credential. If the holder approves this request, the holder's agent responds with proof that the verifier can verify. A crucial step in this process is the verification of the issuer's digital signature, typically accomplished using a DID, which will be discussed in the next section.[2, p. 24-26]

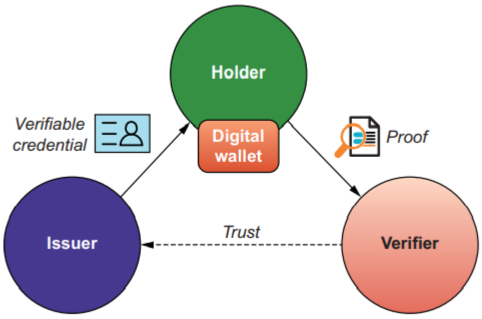


Figure 2.1: Trust triangle for SSI ecosystem. [2, p. 25]

The trust triangle is the relationship between issuers, holders, and verifiers, and it is fundamentally how human trust relationships are conveyed over a digital network. [2, p. 101]

VCs only convey trust if the verifier trusts the issuer. This does not mean that the verifier must have a direct business or legal relationship with the issuer. It means that the verifier is willing to make a business decision based on the level of trust assurance the verifier has in the issuer.

A trust triangle, as discussed above, only describes one side of a business transaction. It is possible that both parties request information and that both parties may play the roles of holder and verifier in a single transaction. This is not the case in our project, as we deal with government entities or private businesses. Here the trust in the user is established by sending an identity credential issued by the government. It would be less natural for NAV to send a credential proving their identity, as they are the established party.

2.2 Decentralized Identifiers

One of the essential building blocks of SSI is the decentralized identifier, commonly called a DID. A DID is a new type of identifier that can refer to any subject such as a person, organization, or an abstract entity. The innovative part of DID is that rather than being based on federated identifiers, they are entirely controlled by the DID's owner. This allows DIDs to be decoupled from centralized registries, identity providers, and certificate authorities and establishes trust-able interactions between subjects.[3]

2.2.1 DID methods

There are many different methods when it comes to creating a DID, both on and off the blockchain. A DID structure is quite similar to a URL; the scheme is defined as a DID, then the DID method is specified, and the last part is a DID method-specific string that serves as the ID. For reference see Figure 2.2. Two examples of DID methods are DID:ethr and DID:web.

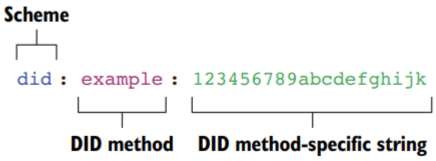

Scheme
did : example : 123456789abcdefghijk
DID method
DID method-specific string

Figure 2.2: Example layout for a DID. [2, p. 158]

DID:web is a decentralized identifier that is domain-based. Creating a DID:web consists of three steps

1. Register the use of a domain name.
2. Storing the IP address at DNS lookup service.
3. Create the DID document JSON-LD with a key pair and store the did.json at an URL.

The DID resolver can look up the URL to verify the did.json data. If several DIDs are stored on the same domain, use a specific path for the different DIDs. This method does not require a blockchain network to work and therefore is easier to understand for the general public. DID:web is a suitable option in a scenario where a business wants to create identifiers for themselves. Most businesses have a dedicated web page and can use that domain for verifying the DID:web identifier. Additionally, a malicious actor cannot impersonate the business, as they do not have access to the business' domain. One would look up their DID:web identifier and see that it is not issued from the correct domain and, therefore, fraudulent.[4]

DID:ethr is a DID method based on the Ethereum blockchain. It works very similarly to the DID:web method, but instead of storing the did document on a domain, one uses an Ethereum address. An Ethereum address can only be owned by one user at the time and has a public key. Decentralized identifiers are designed to be compatible with any distributed ledger or network. However, they are designed around the blockchain, using the peer-to-peer connection that the blockchain provides. To make changes to a DID-document on the blockchain you need to make an Ethereum transaction. To make a transaction requires gas; this means one needs to pay some Ethereum as a transaction fee. To avoid buying Ethereum, the group uses ethr:rinkeby, which is a free testing currency given to developers so they can work and experiment with Ethereum.

Ethereum is a decentralized global software platform powered by blockchain technology at its core. The first association people have with Ethereum is cryptocurrency. However, cryptocurrency is only one function of Ethereum's smart contracts. A smart contract is a program that runs on the Ethereum blockchain. Smart contracts define rules and regulations and automatically enforce them using code. A user interacts with the blockchain by submitting transactions to the smart contracts. Instead of a currency stored on the blockchain, the did:ethr stores a DID-document. [5]

2.2.2 DID document

A DID:method by itself is not very useful, just like a web address with only HTTPS is not very interesting. A web address needs a hyperlink to link to a resource; similarly, a DID needs a DID document. A DID document is a standardized data structure which contains metadata about the DID subject. These are typically cryptographic keys, authentication methods, and other metadata describing how to engage in trusted interactions. To retrieve a DID document, you need something called a DID resolver. The resolver takes the standardized data structure and does something useful, like using it to establish a trusted interaction. A DID document is not human-readable but a document intended for identity applications such as; digital wallets, agents, or encrypted data stores. [2, p.161]

2.3 Verifiable credentials

The concept of verifiable credentials is at the very core of self-sovereign identity. It is common to think of credentials as a piece of paper or plastic that fits in a wallet. Credentials can be used to prove someone's identity, like driving licenses, government IDs, and employment cards. The term credential extends to any tamper-resistant set of information that some authority claims to be true about the subject of the credential. This, in turn, enables the subject to convince others of the validity of their claim.

Every credential has a set of claims about the subject of the credential. These claims are made by a single authority, which is referred to as an issuer in self-sovereign identity. The person the credential is assigned to is commonly referred to as the credential holder. The claims made by the credential could be anything from a person's social security number, to their height and entitlements, like healthcare benefits. [2, p.139]

To qualify as a verifiable credential, the claims must be verifiable in one way or another. This means that a verifier must be able to determine who issued the credential, that it has not been tampered with, and that it has not been revoked or expired. With physical credentials, this is usually done with proofs of authenticity embedded directly in the credential, like watermarks and holograms. However, this manual verification process can be difficult and time-consuming.

Digital verifiable credentials save time and simplifies the verification process. The first part of a verifiable credential is a unique identifier, and the second part is metadata, like the expiration date. The third part is the claims contained in the credential, all the other data items like date of birth, gender, marital status, and similar claims. The fourth and last part is a digital signature made by the credential issuer using cryptography.

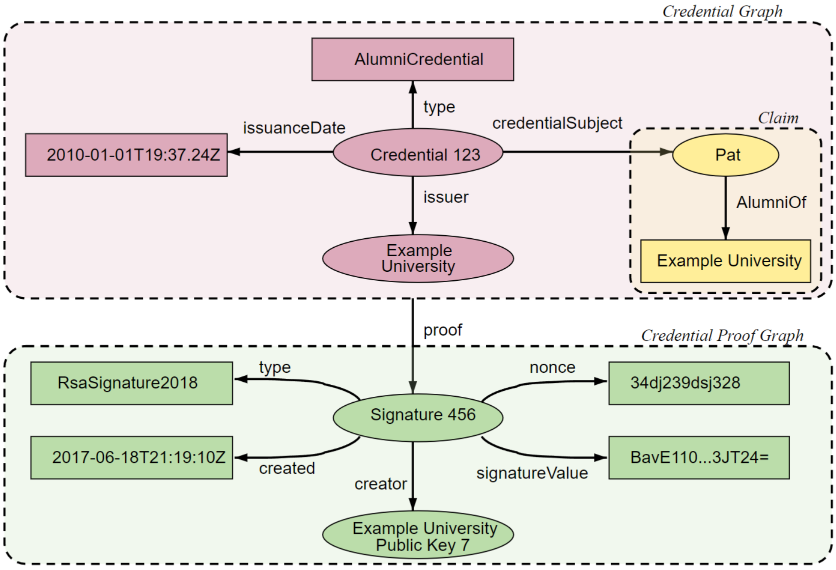


Figure 2.3: An information graph that shows the contents of a verifiable credential. [6]

Context - Schema

The credential schema is an essential part of the verifiable credential. It is a document used to guarantee the structure and, by extension, the semantics of a set of claims comprising a verifiable credential. The integrity of a verifiable credential heavily relies on how to structure the credential so that all the parties, issuer, holder, and verifier may have a singular trust mechanism provided by the credential schema. [7, p. 5.4]

2.3.1 Verifiable data registries

A DID can be registered with any decentralized network or verifiable data registry. For example, on a blockchain. A blockchain is a highly tamper-resistant transactional distributed database that no single party controls. In other words, it may provide an authoritative source of data that many different peers can trust without any single peer being in control. However, a blockchain trades off many features apparent in distributed transactional databases, like performance, efficiency, scalability, and searchability.

Blockchains accomplish the feat of removing central authority while maintaining high trust in data integrity by using a triple play of cryptography. Firstly, every transaction to a blockchain is digitally signed, and each peer manages their private

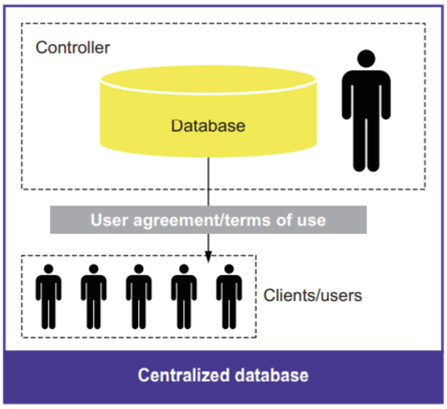


Figure 2.4: Architecture of a centralized database. [2, p. 33]

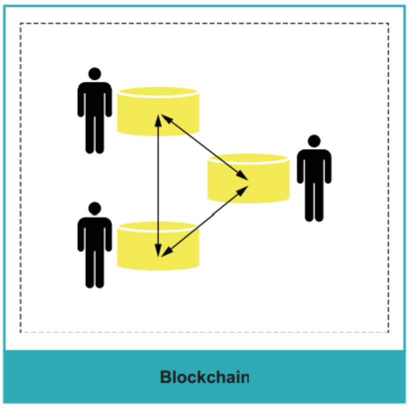


Figure 2.5: Peer-to-peer blockchain connection. [2, p. 33]

keys and digitally signs their transactions with the blockchain. This ensures that no new transaction is accepted unless the signature is verified. Secondly, all transactions are grouped into cryptographically hashed blocks linked to the previous block. This step ensures that the chain of transactions is immutable. Lastly, every new block is cryptographically replicated across all peer nodes on the blockchain network. This means that every peer ends up with a copy of the latest block, and as long as the majority agrees on the new addition, the blockchain may never be rewritten. [2, p. 33]

2.3.2 DIDComm

DIDComm messaging is a secure communication methodology built upon an exchange of DIDs to establish a communication channel. The exchange of keys is done by resolving the DIDs and finding the keys in their respective DID-documents. This constitutes an authentication channel, where one is given control of the DIDs private keys.

DIDComm allows for mutual authentication between two parties. Instead of a sign-in interaction that much of web communication requires, where the business vouches for one's identity, the DID belongs to the individual. This removes the power imbalance between the user and the business. The business does little to identify itself and gains a great deal of data from the user accounts. This data can be sold and used to manipulate advertisements or online behavior. Businesses properly identifying themselves adds another security layer that makes it harder to spoof websites intended for stealing personal data. [8]

DIDComm is a transport agnostic service. This means that the DIDComm transport layer does not care how the information is transported. One can transport the message over HTTP, BlueTooth, WebSockets, and more. In this project, we transport information over the local network. This is purely for creating a demo; a real-world implementation would have to use a different method. [8, p.8]

2.3.3 Trust anchors

A trust anchor in PKI is a certification authority that signs certificates that identify the other part in a communication channel. SSI does not remove trust anchors from the process. One can set up a cryptographically secure communication channel, but if one does not know whom one is communicating with, there are still possibilities for exploitation. There are different approaches to solving this issue, and it is almost impossible not to have a human involved in this process. [9]

It is possible to have an extensive registry with DIDs the user can look up to confirm they are connecting to a verified entity. The drawbacks to this are the loss of privacy and giving up some control to the people in charge of the registry. If a DID is connected to a person, and the person is identified in a centrally run registry, the DID can be used for tracking their online activity. In addition, the institution that controls the registry will have much power over the system. Will it be run by the state or a private corporation? These central databases are weak to exploitation by malicious actors and by the institutions that run them. [9]

Another way to solve this is an exchange of VCs after the DIDComm connection is established. One connects with an unknown entity and establishes their identity through an exchange of credentials. The drawback is that businesses and institutions need to have a credential proving who they are. It is more natural for the user to have a VC and the verification for a business to be more static. The benefit is that the user does not have to rely on a certification authority to have a centrally run registry.

2.3.4 Veramo

Veramo is a collection modular APIs that are designed for using verifiable data and SSI. Veramo makes it possible for anyone to use DIDs and verifiable credentials in their application. One of the drawbacks of using SSI is all the different standards. All the different components must have the same standards for the ecosystem to work. Veramo deals with this, so the developer does not have to. Veramo consists of different packages that deal with the SSI flow. Below is a list of the Veramo core functionality. [10]

- Key Manager - Creates keys for the DID documents.
- Data Store - Stores data in the wallet/database
- DID manager - Creates DIDs.
- Resolver - Resolves DIDs.
- Credential Issuer - Creates verifiable credentials.

If other Veramo packages are necessary, like DIDComm and message handler, they must be important in addition to the Veramo core.

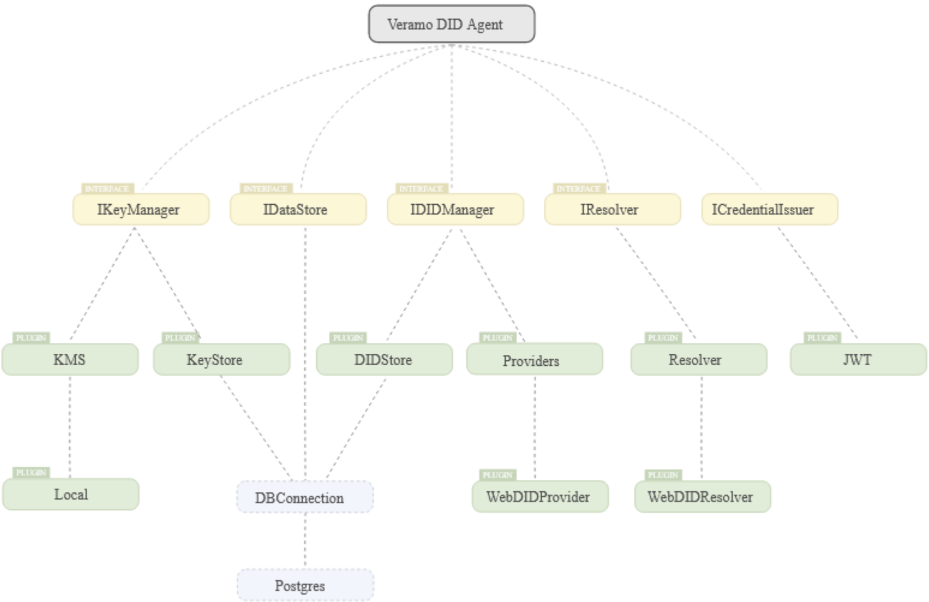


Figure 2.6: Veramo plugins architecture. [11]

2.3.5 Infura

One of the most important technologies we use for this project is Infura. Infura is a blockchain development suite with APIs and developer tools that grants easy and reliable access to the Ethereum network. In this project, it stores decentralized identifying documents on the Ethereum test network called Rinkeby. It is also used whenever we want to resolve a decentralized identifying document and send messages over DIDComm.

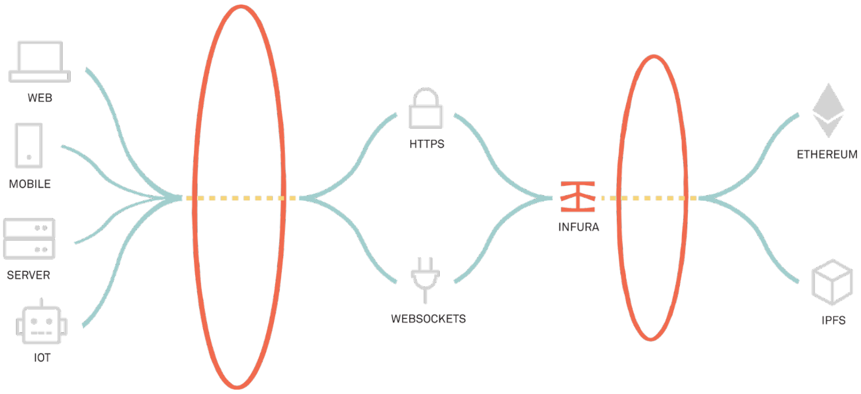


Figure 2.7: Infura connects to the Ethereum network. [12]

2.4 Problem description

One of the challenges with writing a bachelor thesis about SSI is that a lot of the infrastructure required does not exist yet. The task is to create a proof of concept, not a standalone application. When creating a proof of concept of something as significant as changing public services to use a SSI model, data and interactions need to be mocked. Creating the entire ecosystem would be outside the scope of our assignment and not possible with the human resources and time available.

This thesis focuses on the particular scenario of using verifiable credentials in the process of applying to unemployment benefits form NAV. For this interaction, we created four digital agents to make up the SSI ecosystem. The agents are the user agent, Symfoni agent, NAV agent, and state agent. The flow would go like this; the user requests a person's credential from the state to prove their identity. The user requests employment and termination credentials from their previous employer, who grants these based on the user's personal credential. Finally, NAV requests all three credentials and makes a decision based on the information they contain. This process can be made as simple or complicated as desired. With this framework, one can add more credential issuers and add their credentials as a requirement for NAV. An example could be NAV requiring information about an applicant's education, and the user sends an education credential issued by a university. In the same way, we could only have used an agent for the user and NAV and mocked the rest of the data.

One of the challenges for SSI is compatibility. Veramo is used in this project, and the Veramo API deals with many of the compatibility issues. All the digital agents in the project are created with Veramo. Compatibility is easy to achieve in a small self-contained project, but consistent data structures are required in a real-life scenario. This is why all verifiable credentials have a Schema they strictly follow. Creating schema ensures that the data issued from one issuer is the same as the verifier uses. Hence, a verifiable credential is the same regardless of which agents created it. These schemas do not exist and have to be made by the group.

In addition to the SSI elements of the project, we also have to develop an application with a GUI for the demo. This app must be able to communicate between the different agents so that they can send and verify credentials. There are many options for the transport layer as to how the information is exchanged, which is less important than the SSI elements. The final product uses DIDComm as a communication channel, as it is a part of the veramo plugins. It establishes DID-to-DID connections, which adds another layer of security and control for the user.

Chapter 3

Requirement specifications

This chapter is about the requirement specification set by Symfoni for this bachelor's thesis. The requirement specifications are important for understanding the state of the final product. These specifications will be illustrated by several diagrams with underlying text that explain the content.

3.1 Use case

The task was to create a proof of concept for SSI technology. To accomplish this, the application needed to contain certain elements. The requirements defined by Symfoni are as follows:

1. A VC issuer that issues credentials. Veramo is used as a framework for the creating of VCs and the DID.
2. A VC verifier that validates the credentials.
3. An application that exchanges the credentials between the Issuer and Verifier.

The students could have worked on the baseline Symfoni had already developed, but the group decided to develop an application from scratch. This was to gain a greater understanding of the entire SSI flow and increase their skills with app development. The workload and scope of programming the app with SSI concepts are appropriate for a three-member bachelor group. The original problem definition can be found in Appendix L

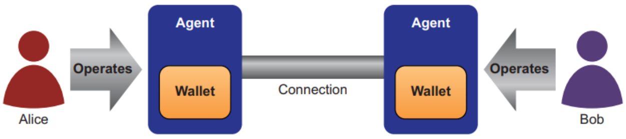


Figure 3.1: Digital agent. [2, p. 29]

We must have digital wallets to store verifiable credentials for this use case.

In contrast to a physical wallet, where a person directly manages credentials, a digital wallet needs a digital agent to operate. For this project we will be using Veramo agents, digital agents created by the Veramo JavaScript modular API for verifiable data and SSI.

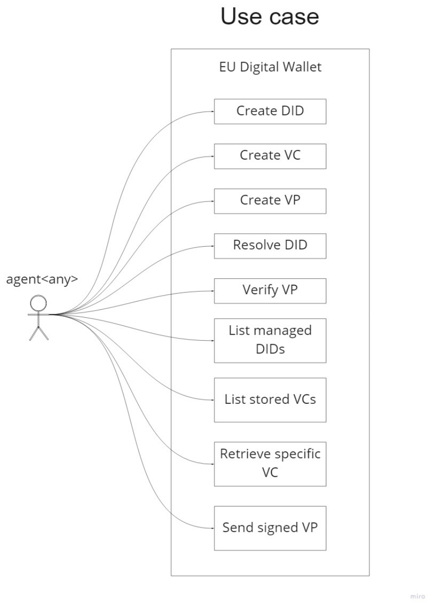


Figure 3.2: Simple use case for the actions an agent can perform.

The agents need to perform specific actions in relation to their role in the SSI ecosystem. All digital agents can perform these actions, but they only perform certain parts depending on whether they are the issuer, verifier, or holder.

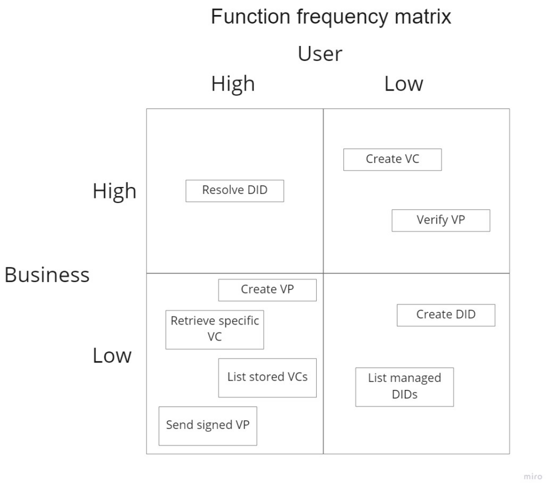


Figure 3.3: Frequency matrix for showing often actions are preformed.

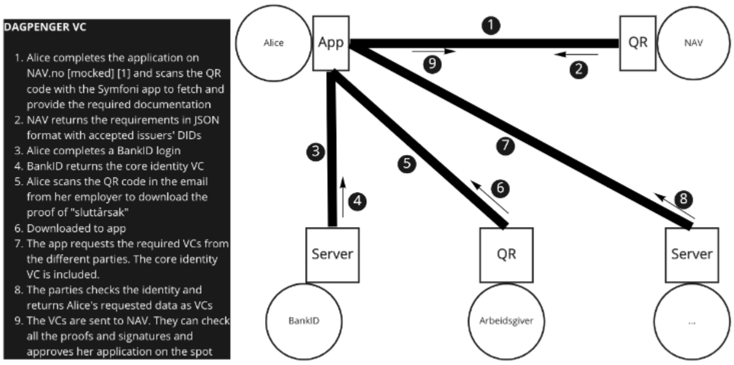
Figure 3.3 represents how often a digital agent performs an action based on their role as an employer or a user. A user agent that belongs to an individual would frequently retrieve VCs from their wallet but rarely create VCs. An agent belonging to a business or state institution would create VS but do not store VCs in the same way a user would. The frequency matrix illustrates how often a business and person perform different actions.

3.2 Flowchart

The flowchart can be found in Appendix I. We created a flowchart that describes the entire flow of the problem scenario from start to end. The scenario starts with the decision point that asks the user whether they have a DID or not. If not, the diagram explains how to obtain a DID, so one can proceed to the next decision. Then one is asked whether the user has a person VC; if not, it tells how to obtain one so that the user can proceed further. The same goes for the employment VC and termination VC. When all the VCs are collected, NAV is asked if they have a DID; if not, a description of how this happens is given. Then the user wants to apply for unemployment benefits, and NAV presents a QR code to do so. The user scans the code, and a new decision point comes, does the user trust NAVs DID? If not, a way to establish can trust NAV is presented. If the NAV DID is trusted the user sends a VP with the VCs requested from NAV. NAV then validates the VP and responds to the application with either a yes or a no, depending on whether the user meets the requirements for unemployment benefits.

The flowchart helped us during the development process, and it showed us clearly what steps were included in the scenario and helped us map out what steps we needed to implement in the development process. It also helped us to understand better the SSI flow in general.

3.3 Initial product outline



DAGPENGER VC

1. Alice completes the application on NAV.no [mocked] [1] and scans the QR code with the Symfoni app to fetch and provide the required documentation
2. NAV returns the requirements in JSON format with accepted issuers' DIDs
3. Alice completes a BankID login
4. BankID returns the core identity VC
5. Alice scans the QR code in the email from her employer to download the proof of "sluttårsak"
6. Downloaded to app
7. The app requests the required VCs from the different parties. The core identity VC is included.
8. The parties checks the identity and returns Alice's requested data as VCs
9. The VCs are sent to NAV. They can check all the proofs and signatures and approves her application on the spot

Figure 3.4: Proposed flow

The workflow in Figure 3.4 is what the project workflow is based on and was provided by Symfoni as a guideline. There have been some adaptions, like using a fake social security number instead of BankID, but the core concepts are the same. The group went with the philosophy of acquiring the necessary credentials before sending an application to NAV instead of establishing the connection to NAV first. In a society where SSI is the primary source of public services, it would be natural to receive VCs throughout the user's life. Instead of only acquiring them when the user applies needs them. In a less integrated system, the user would apply to NAV and be informed of what credentials are required.

3.4 Multistage transaction model

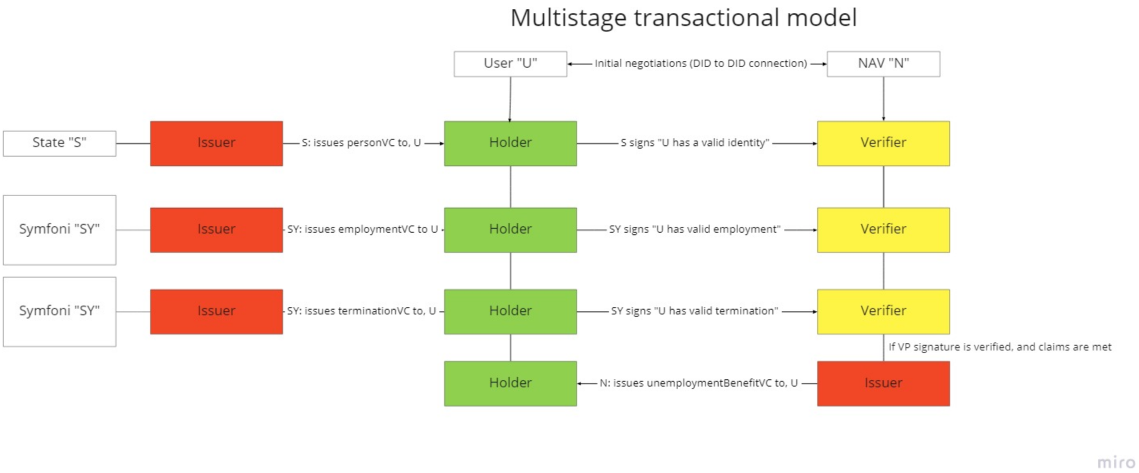

Multistage transactional model
User "U"
Initial negotiations (DID to DID connection)
NAV "N"
State "S"
Issuer
S: issues personVC to, U
Holder
S signs "U has a valid identity"
Verifier
Symfoni "SY"
Issuer
SY: issues employmentVC to U
Holder
SY signs "U has valid employment"
Verifier
Symfoni "SY"
Issuer
SY: issues terminationVC to, U
Holder
SY signs "U has valid termination"
Verifier
If VP signature is verified, and claims are met
Holder
N: issues unemploymentBenefitVC to, U
Issuer
miro

Figure 3.5: Shows the flow of credentials, and how the different actors play different roles in a transaction.

The first stage of the transaction is establishing a secure DID-to-DID connection where both parties have proof to show who they are. After the connection is established, the user opens his digital wallet and finds the VCs that the other party is requesting. If the user lacks one or more required credentials, the user has to ask for the missing VC from a trusted issuer. When all credentials are in the user's possession, they are formed into a VP, which is sent to the other party. The other party, in this case, NAV, verifies the VP payload and returns a credential with proof of unemployment benefits.

As displayed in Figure 3.5, the actors can play the role of holder, verifier, and issuer in the same transaction.

3.5 Swim lanes

This section describes the different Swimlane diagrams made to explain the flow of our application. These documents were prepared for the meeting with NAV, where we pitched the SSI concept for their application process. They are a combination of business diagrams that explain how a user would interact with the system and more technical diagrams of how the system logic works on a technical level. These diagrams were early drafts, and some changes have been made after they were created. Therefore, some of the information is outdated.

Initially, the plan was for businesses to use did:web and individuals to use did:ethr. The rationale was that every business has an official web page, and a user can look up the DID address on their website. An individual is less likely to have a personal web page, making it harder to use the did:web method. A user would need to get many different credentials from the state, passports, driving licenses, and alike. It would be simpler for the government if every person used the same DID-method, and it is distributed in a controlled way. This idea of using did:web was abandoned, as it seemed more logical for all the agents to use the same did:method. The group also wanted to utilize the blockchain, which did:web does not.

The group also changed the method for establishing DID-to-DID connections. The plan was to use Wallet Connect, but it was later changed to use DIDComm.

3.5.1 Business

For this section SSI will be referred to as the EU-digital wallet. EU digital wallet is part of the European Blockchain Service Infrastructure, EBSI. The EBSI aims to create a digital identity with trusted data sharing. When proposing this idea to NAV, we used the EBSI to gather interest, to show that large entities like the EU are interested in verifiable credentials and digital identity.

The business Swimlane diagram in Figure 3.6 shows a simplified view of the entire process. The first step is to download the EU digital wallet application. In this project, the user agent acts as the EU digital wallet app. Navigate to the national registry, which holds information for all Norwegian citizens, and log in. In the diagram, login requires BankID to make a secure connection. We did not have the opportunity to integrate BankID into our project, so the login is based on one's social security number. There are many possibilities for handling this identity verification, and BankID is our suggestion.

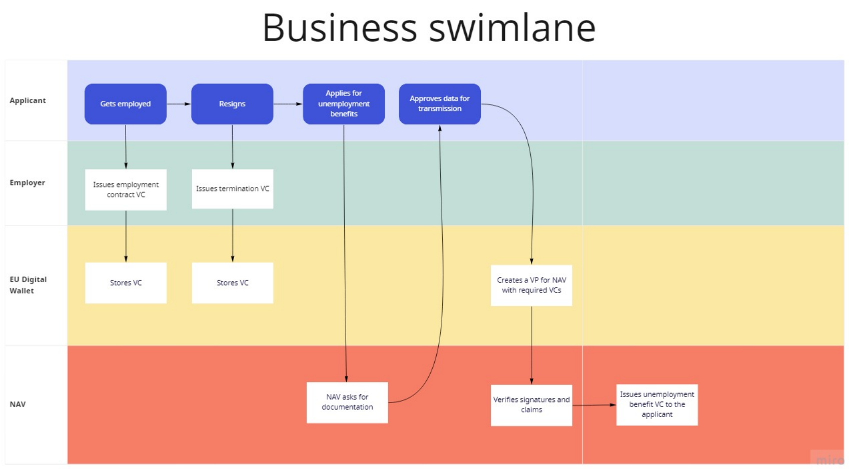

Business swimlane
Applicant
Employer
EU Digital Wallet
NAV
Gets employed
Resigns
Applies for unemployment benefits
Approves data for transmission
Issues employment contract VC
Issues termination VC
Stores VC
Stores VC
Creates a VP for NAV with required VCs
NAV asks for documentation
Verifies signatures and claims
Issues unemployment benefit VC to the applicant
miro

Figure 3.6: Shows a business flow how a user would proceed to apply for unemployment benefits

3.5.2 User making a DID

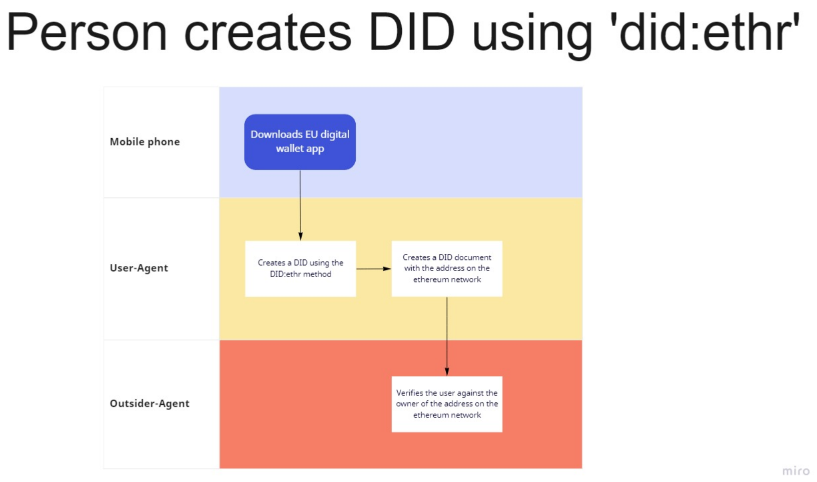

Person creates DID using 'did:ethr'
Mobile phone
User-Agent
Outsider-Agent
Downloads EU digital wallet app
Creates a DID using the DID:ethr method
Creates a DID document with the address on the ethereum network
Verifies the user against the owner of the address on the ethereum network
miro

Figure 3.7: A user creating a DID using did:ethr

This is the swimlane diagram for a user creating a DID. If a user do not have a DID they must first download the EU digital wallet app. In the app there will be a option to create a DID using the DID:ethr method. When the DID is being created a DID document is created with it, and contains its address on the Ethereum network. Now an outsider agent can verify the user against the owner of the address on the Ethereum network.

3.5.3 Company making a DID

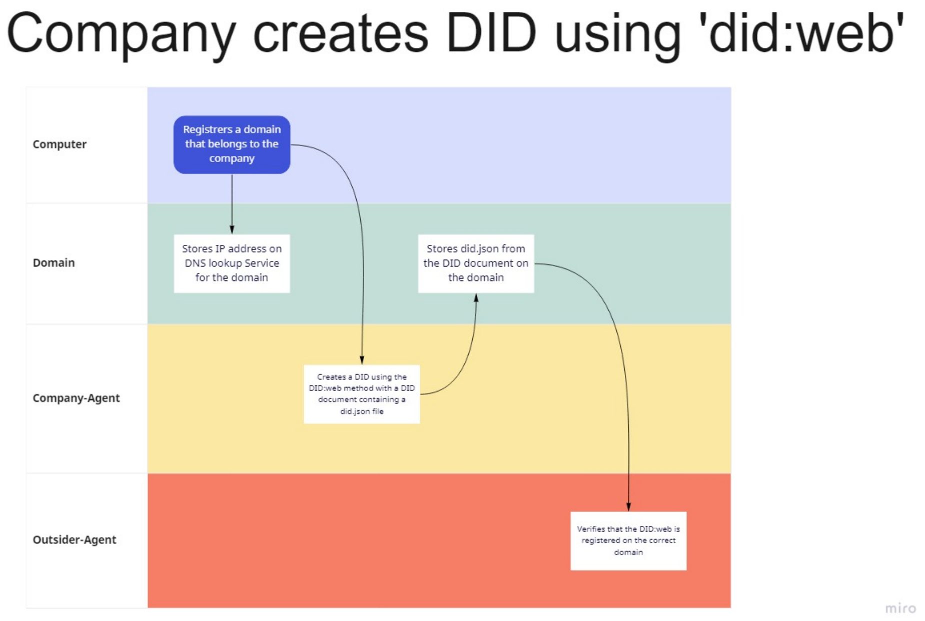

Company creates DID using 'did:web'
Computer
Domain
Company-Agent
Outsider-Agent
Registrers a domain that belongs to the company
Stores IP address on DNS lookup Service for the domain
Stores did.json from the DID document on the domain
Creates a DID using the DID:web method with a DID document containing a did.json file
Verifies that the DID:web is registered on the correct domain
miro

Figure 3.8: Company creates DID:web

As mentioned above, this diagram uses did:web for businesses. This figure is kept to show that it is possible to use different DID methods, and here is an example for creating a did:web. For did:ether, a business would use the same flow as the user in Figure 3.7.

First, a company registers a domain. They store their IP address on a DNS lookup service for the domain. The company agent creates a DID using the DID:web method with a DID document containing a did.json file. Then, the company stores the DID-document on their domain. Other agents can resolve the DID by looking up the DID-document on the company's domain. The DID-document can be stored on any website, a GitHub page if you do not have access to a domain.

3.5.4 User acquires personal ID VC

The following two diagrams are more technically oriented, as they describe how the digital agents interact with the system, not just the users. After the user has created a DID, they can request a person verifiable credential. This request would be to a government agency, and in this scenario, it is the national registry. The national registry request verification with BankID and establishes a connection.

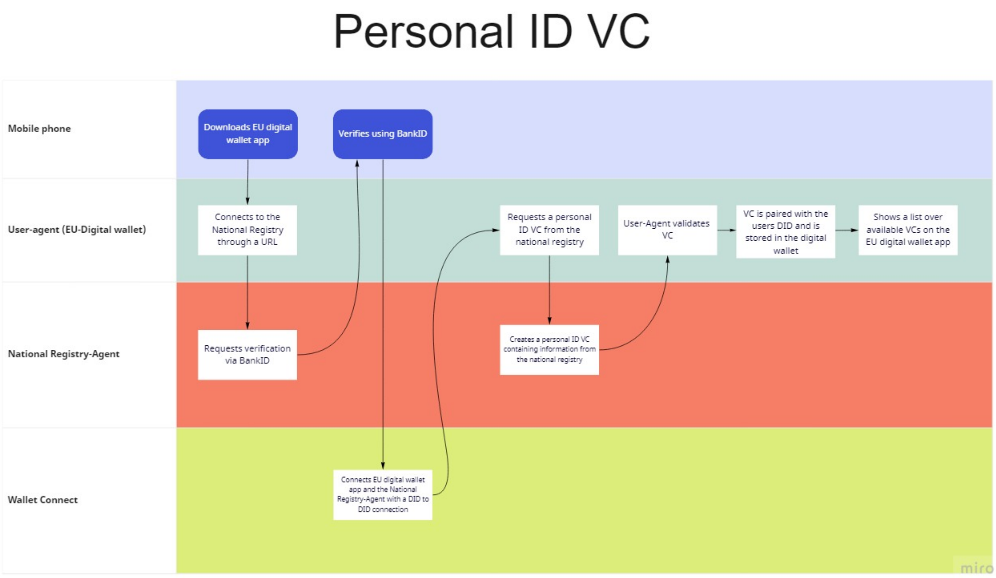

Personal ID VC
Mobile phone
User-agent (EU-Digital wallet)
National Registry-Agent
Wallet Connect
Downloads EU digital wallet app
Verifies using BankID
Connects to the National Registry through a URL
Requests a personal ID VC from the national registry
User-Agent validates VC
VC is paired with the users DID and is stored in the digital wallet
Shows a list over available VCs on the EU digital wallet app
Requests verification via BankID
Creates a personal ID VC containing information from the national registry
Connects EU digital wallet app and the National Registry-Agent with a DID to DID connection
miro

Figure 3.9: A user acquires a person credential from the national registry

3.5.5 User acquires employment VC

User acquires the employment VC is the use case we want to showcase. All the other diagrams are the steps required to obtain an employment VC. Go back to an earlier diagram to see how a person acquired the credential, or DID required to complete this workflow.

One of the goals of designing this system is to use as many existing systems as possible. When issuing a person credential, we use the data that already exists in the national registry and convert it into a verifiable credential. In the same way, when issuing employment credentials, one takes the information from an accounting system and converts it into a verifiable credential. Using the business registry, the government has to display the DID for all registered businesses. This way, the transition to using SSI is made easier when you can use existing systems. We want the system to be as unobtrusive as possible, requiring little technological knowledge.

Some steps need additional explanation. In the third step, the user agent verifying the business DID is not an active action. When using DIDComm, one establishes a connection before verifying the other party. "Verify business DID" is before establishing a connection because it is on the user to ensure they are connecting to the correct business. We imagine all Norwegian businesses need to have a business license and be registered in a government registry somewhere. Their DID should also be available on this registry. If it is just their DID address displayed, it would be hard to identify, but the concepts of identity spoofing and fraud are outside the scope of this project.

When the DID to DID connection is established, the employer asks for a personal credential from the user. An essential step of the EU digital wallet is that the user has more control over their data, and the user can decline to share information they do not wish to share. When the user consents, the employer agent verifies them, and if they are in their database, they issue a credential.

Entering the employee information is put towards the end of this diagram but would, in reality, be done asynchronously to this process. The employer hires a person, registries them in their accounting system, and then issue credentials based on the information in the accounting system. It is put at the end of the diagram because that is where the information is retrieved. It is a simplification but conveys the concept.

Due to space limitations see figure 3.10 in Appendix H for better quality.

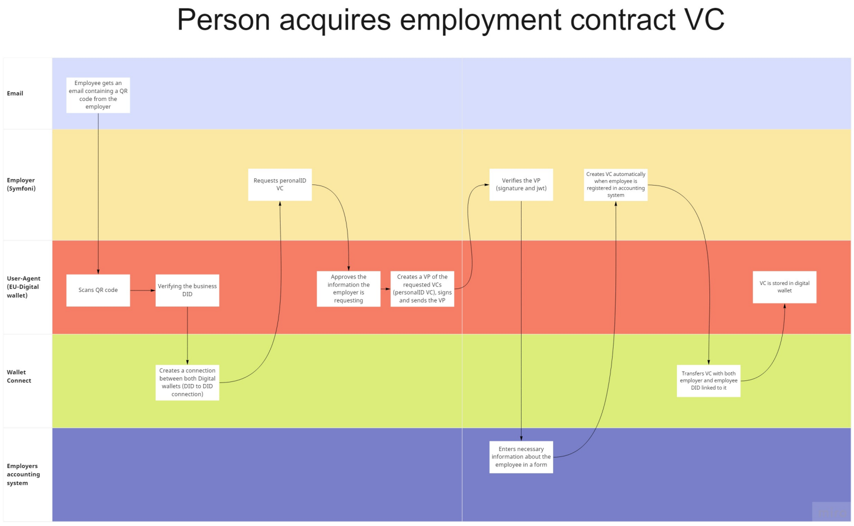

Person acquires employment contract VC
Email
Employer (Symfoni)
User-Agent (EU-Digital wallet)
Wallet Connect
Employers accounting system
Employee gets an email containing a QR code from the employer
Scans QR code
Verifying the business DID
Creates a connection between both Digital wallets (DID to DID connection)
Requests peronalID VC
Approves the information the employer is requesting
Creates a VP of the requested VCs (personalID VC), signs and sends the VP
Verifies the VP (signature and jwt)
Enters necessary information about the employee in a form
Creates VC automatically when employee is registered in accounting system
Transfers VC with both employer and employee DID linked to it
VC is stored in digital wallet

Figure 3.10: Shows the flow of credentials, and how the different actors play different roles in a transaction.

Chapter 4

Design

The design chapter showcases the final product's design and how it diverged or followed the requirements specifications. It does not cover how the design was implemented, as that will be discussed in the implementation chapter. However, it does describe design changes that were made during development.

4.1 Scenario

We have developed an application for creating, sending, and verifying verifiable credentials between different digital agents in this bachelor project. The problem statement was making a proof of concept showcasing SSI and how it could be used to improve public services. The public service we seek to improve is the application for unemployment benefits to NAV. This project is a cooperation between Symfoni, NAV, and NTNU. Having multiple respected institutions involved raises credibility for the SSI project and increases awareness of this new technology. Symfoni described this process as a long stair towards a future where public services are distributed using SSI, and this bachelor thesis was one step up the stairs.

This project is purely designed around creating a demo for showing the concepts behind SSI. Without major revisions, our application could not be deployed on the App Store or Google Play. This is due to the project structure and the government and businesses' lack of information and infrastructure. Several requirements do not exist yet for this scenario to work, and therefore, many of the requirements need to be mocked.

The missing infrastructure from the government comes from the lack of a verifiable credential for digital identity. Like a passport, this has to be issued by a state agency, as they are in charge of managing citizenship for a country. NAV needs to have the capabilities to receive VCs and process them to get the desired information. All the schema for the different credentials needs to match, something that is suitable for a government agency to regulate. Credentials of the same type need matching data structures. Therefore, they need to use the same schema. Organizing every entity to use the same schema type requires central planning and regulation, a task best suited for a government agency.

In the same way as a government agency, a business needs the capability to issue and verify credentials. If the user's workplace does not support VCs, then they cannot acquire one to send to NAV for verification. In this project, the employer takes information from their accounting software and creates a credential based on this information. The business's accounting software must be set up to accommodate the information required for issuing an employment credential. If the software input does not match the schema, one cannot create a verifiable credential that is compatible with other credentials of the same type.

The application for unemployment benefits is multi-faceted, and our case of employment and termination credentials only touch upon some of the requirements. One requirement not covered by this project is if one is actively searching for employment, and this is information not included in an employment or termination credential. However, one can create a verifiable credential with actively searching for employment as the subject using this project as a framework.

4.2 System architecture

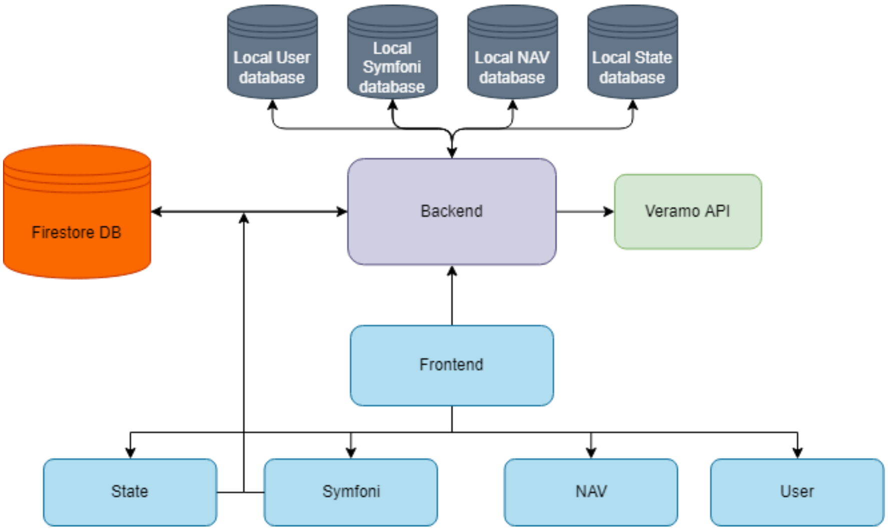


Figure 4.1: System architecture

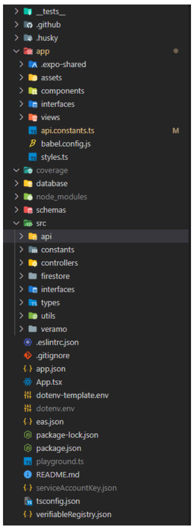
The system is divided into a separate backend and frontend where the backend interacts with the Firestore database, the local databases, and the Veramo API, and the frontend interacts with the backend and Firestore database.

In this application, the group has created four digital agents to represent the different entities involved with acquiring an employment credential.

1. The user agent represents an individual applying for unemployment benefits. They receive VCs from both the state and Symfoni. They store these credentials in their wallet and can create VPs to verify their information.

2. The Symfoni agent represents a business with the employer's role for this project. They add new employees in their accounting system and issue employment and termination credentials when the user requests them.
3. The NAV agents represent the Norwegian welfare agency. NAV requires information to process the application when a user applies for unemployment benefits. They request VCs from the user to verify that these criteria are filled.
4. The state agents represent the state agency in charge of distributing digital ID credentials, called person credentials, in this project. It takes information from a government registry and converts it to a verifiable credential.

4.2.1 File structure



```
> __tests__
> .github
> .husky
v app
  > .expo-shared
  > assets
  > components
  > interfaces
  > views
    api.constants.ts                M
    babel.config.js
    styles.ts
v coverage
> database
> node_modules
> schemas
v src
  > api
  > constants
  > controllers
  > firestore
  > interfaces
  > types
  > utils
  > veramo
  .eslintrc.json
  .gitignore
  app.json
  App.tsx
  dotenv-template.env
  dotenv.env
  eas.json
  package-lock.json
  package.json
  playground.ts
  README.md
  serviceAccountKey.json
  tsconfig.json
  verifiableRegistry.json
```

Figure 4.2: Src folder

One of the key concepts behind our file structure is modularisation. We split the code base into manageable chunks that are possible to understand. It is divided into two main folders, "Src" and "App.". In "Src" are all the files related to the back-end and the API, while the React Native code for the front-end is in the app folder. We have tests, databases, and schemas outside these, and they are not directly involved with the code base, just used by them.

The "Src" folder is modularised further, with the most prominent modules being the controllers and the API. The controller classes are split into one for each agent, with one parent class they all inherit from. Similarly, the API is split into controllers, where all the endpoints related to one agent are in that controller. To supplement these classes, we have folders for interfaces, types, utility functions, and the Veramo setup. Initially, the "src" folder was the only folder containing code; it should be renamed to "backend" as this is no longer the case. Due to avoid import problems, it has not been renamed.

The "App" folder containing the front-end is split into components, interfaces, and views. Views are where most of the code is located and are split into one folder for each digital agent. This way, we keep them separate and give the developer better control of what part of the GUI they are working on. Throughout the project, we have used interfaces for important classes. To give the developer a better understanding of what the class contains and make sure nothing important is removed unintentionally. The only outlier is "App.tsx", as it is located in the root folder. The reasoning behind this is node requires a particular path, and it was more consistent to move "App.tsx" than manually changing paths.

4.2.2 Limitations

The problem is that the functionality to have an integrated SSI system does not exist yet. In our scenario, an employer put the information about a new employee into an accounting software. This accounting software needs to have all the required information laid out in the employment schema; if not, the verifiable credential cannot be created. So a collaboration to use SSI needs both employers and accounting systems to participate. If the benefits are not immediately apparent, it will not be easy to convince them to adapt to this system.

Similarly, personal credentials need to be issued from existing information within the state. There already exists population registries, and the government gives out passports to its citizens, and this would have to be adapted to a person verifiable credential. This initiative is spearheaded by the EU EBSI commission, which aims to improve service infrastructure with blockchain technology. The benefit of this is having a large, respected institution as the EU support SSI implementation. However, the EU consists of 27 countries with different standards and methods, making it a huge undertaking to make all standards compatible. We mocked data based on existing standards from "datatilsynet" for this project. This infrastructure would need to be established first to make the app deployable.

4.3 Controller classes

For this project, we have different Veramo digital agents with functionality for issuing and verifying verifiable credentials. Each agent represents one entity interacting with the SSI system, so the "SymfoniAgent"represents an employer and all the actions an employer needs to be able to perform. All the agents have a controller class with functions related to their roles in the project.

The different agents share most of their functionality, so a parent class called "AgentController" was created. The main functionality is in this class, and the other agent controllers extend this class. This makes it possible to easily add new agents in the future and ensure that all agents have the same base functionality. It is also a time-saving measure, as there is no need for duplicate code between agent controllers.

With several developers working on this project, being consistent and using the correct data is important. A list of custom types has been made, so one has a predefined type as input instead of manually typing in variables. This is to reduce the number of errors made by mistyping and give a better overview of how the VCs are supposed to be set up. In addition an interface for the parent class "AgentController" has been made. This interface defines what the "AgentController" class consist of. This is for reproduce-ability and better control of all the functions in this fundamental class.

4.4 Schema

The first interactions of the employmentVC schema needed revisions to the structure. After meetings with NAV, changes were made to reflect the desired information better. Two important aspects not included in the first iteration were the last day an employee worked and the last day they received payment. This did not seem natural in an employment contract, so the employmentVC was split into an employmentVC and terminationCV. The employmentVC would include information included in a physical employment contract, while the terminationVC would include information related to the employee's termination. This includes the last day an employee worked, the last day they received payment, and if they were fired during their trial period.

```
{
    "@context": [
        "https://www.w3.org/2018/credentials/v1"
    ],
    "issuer": {
        "id": "did:ethr:rinkeby:[example_address]"
    },
    "type": [
        "VerifiableCredential",
        "BusinessVC"
    ],
    "credentialSubject": {
        "id": "did:ethr:rinkeby:[example_address]",
        "business": {
            "name": "name",
            "industrialCode": "12KJ",
            "organisationStructure": "AS",
            "organisationNumber": 12,
            "address": {
                "countryCode": "NO",
                "city": "Gjovik",
                "zipCode": 2827,
                "streetName": "gateveien",
                "streetNumber": 22,
                "floor": "2A"
            }
        }
    },
    "issuanceDate": "2022-02-18T10:02:18.000Z",
}
```

The students made the first iterations of the personVC and the businessVC from scratch. They were created using existing models of a person and a business by "Felles datakatalog" on "Data Norge" [13]. To create a custom schema for person and business was not desirable; it is better to link to an established schema from "Felles datakatalog." When this proved difficult, we consulted semantic web specialist Steinar Skagemo, but no usable schema was supplied from Data Norge. As there were no usable schema, the group created custom schema based on the information from "Felles datakatlog" [13].

Delving a little deeper into the semantic web, options for a further link between schema with JSON-LD was explored. JSON-LD is to increase machine readability for data on the web. With JSON-LD, one can provide context to fields on a webpage. If one signs up as a new user and requires a name field, one can link to a different website that defines a name. This way, a computer knows if it is a person's name or just a nickname. Unfortunately, this did not translate too well to the project. The JSON-LD linked to the different schema to define where we got the object from but did not enforce the types defined in the other schema. Because of this, it was decided not to use JSON-LD to give additional context to the schema.

4.5 API

To make an easy integration from the front end to the back end, we developed an API. This API is split into four different categories, one for each agent. The API works as a middleman between the controller class and where one wants to call certain functions in the application. This was used to great effect when developing the front end; simply call the API where desired. In addition, the API is handy for doing manual testing during the development cycle. Insomnia and Postman were used to test the API during development.

To make it easier to use, endpoint names have been standardized to simply use the correct name for the agent one wishes to use. Creating a DID for the user agent is done with the "/user/did" endpoint while creating a DID for Symfoni is "/symfoni/did". A colon in the endpoint means that specific user input is required. For example ":id" an id is required, and ":did" a DID-URL is required. There are tables with all the endpoints for each agent in the next few sections. Most share the same functionality, but there is some variation in functionality between the different agents.

4.5.1 user endpoints

The user agent has no database endpoints but has the most messaging endpoints, as they receive VCs through these messages. A user can create credentials but will rarely do so.

Table 4.1: Endpoints for the user agent

Endpoint name	relation	status codes	Request method	Description
/user/did	DID	201, 400	POST	Create a new DID
/user/did/:did	DID	200, 400	GET	Get a DID with DID:url
/user/dids	DID	200, 500	GET	Lists all the agent's DIDs
/user/resolve/:did	DID	200, 400	GET	Resolves DID with DID-url
/user/mainIdentifier	DID	200, 500	GET	Get the main identifier from the agent
/user/credential	CREDENTIAL	200, 400	POST	Add a new credential
/user/credentials	CREDENTIAL	200, 400	GET	Get a credential
/user/credential/:type	CREDENTIAL	200, 400	GET	Get credential by type
/user/credential/:hash	CREDENTIAL	200, 400	DELETE	Delete credential
/user/presentation	PRESENTATION	201, 400, 500	POST	Create a verifiable presentation
/user/verifyJWT	PRESENTATION	200, 400, 500	POST	Verify a JWT
/user/messaging	MESSAGE	200, 400	POST	DIDComm messaging
/user/sendMessage	MESSAGE	200, 400	POST	Send DIDComm message
/user/messages	MESSAGE	200, 400	GET	Get message
/user/message/:id	MESSAGE	200, 400	GET	Get message by ID
/user/handleMessageToken/:token	MESSAGE	200, 400	GET	Handle DIDComm message token

4.5.2 NAV endpoints

The NAV agent has the fewest endpoints as it does not create credentials or interact with a database, and NAV only needs to verify credentials in a presentation and send a message to the user.

Table 4.2: Endpoints for the NAV agent

Endpoint name	relation	status codes	Request method	Description
/nav/did	DID	201, 400	POST	Create a new DID
/nav/did/:did	DID	200, 400	GET	Get a DID with DID-url
/nav/dids	DID	200, 500	GET	Lists all the agent's DIDs
/nav/resolve/:did	DID	200, 400	GET	Resolves DID with DID-url
/nav/mainIdentifier	DID	200, 500	GET	Get the main identifier from the agent
/nav/credential	CREDENTIAL	200, 400	POST	Add a new credential
/nav/credentials	CREDENTIAL	200, 400	GET	Get a credential
/nav/credential/:type	CREDENTIAL	200, 400	GET	Get credential by type
/nav/presentation	PRESENTATION	201, 400, 500	POST	Create a verifiable presentation
/nav/verifyJWT	PRESENTATION	200, 400, 500	POST	Verify a JWT
/user/messaging	MESSAGE	200, 400, 500	POST	DIDComm messaging

4.5.3 symfoni endpoints

Symfoni has unique endpoints for creating employment and termination credentials. Symfoni also has several endpoints for retrieving and uploading data to an

Table 4.3: Endpoints for the Symfoni agent

Endpoint name	relation	status codes	Request method	Description
/symfoni/did	DID	201, 400	POST	Creates a new DID
/symfoni/did/:did	DID	200, 400	GET	Gets a DID with DID-url
/symfoni/dids	DID	200, 500	GET	Lists all the agent's DIDs
/symfoni/resolve/:did	DID	200, 400	GET	Resolves DID with DID-url
/symfoni/mainIdentifier	DID	200, 500	GET	Gets the main identifier from the agent
/symfoni/employmentCredential	CREDENTIAL	201, 400	POST	Creates employment credential
/symfoni/terminationCredential	CREDENTIAL	201, 400	POST	Creates termination credential
/symfoni/credential	CREDENTIAL	201, 400	POST	Adds a new credential
/symfoni/credentials	CREDENTIAL	200, 400	GET	Lists credentials
/symfoni/credential/:type	CREDENTIAL	200, 400	GET	Gets credential by type
/symfoni/presentation	PRESENTATION	201, 400, 500	POST	Creates a verifiable presentation
/symfoni/verifyJWT	PRESENTATION	200, 400, 500	POST	Verifies a JWT
/symfoni/employmentContract	DATABASE	201, 400, 500	POST	Adds employment contract to database
/symfoni/employmentContract/:id	DATABASE	200, 400, 500	GET	Gets an employment contract from database by id
/symfoni/terminationContract	DATABASE	201, 400, 500	POST	Adds termination contract to database
/symfoni/terminationContract/:id	DATABASE	200, 400, 500	GET	Gets a termination contract from database by id
/symfoni/terminationContract/:id	DATABASE	200, 400, 500	DELETE	Deletes a termination contract from database by id
/symfoni/employmentContract/:id	DATABASE	200, 400, 500	DELETE	Deletes an employment contract from database by id
/symfoni/messaging	MESSAGE	200, 400	POST	DIDComm messaging

external database. This is their employee database, where they store the information about their employees, which is used to create the credentials.

4.5.4 state endpoints

The state agent has additional endpoints for database management, as this agent needs to verify social security numbers from an external database. There are also endpoints for creating person and business credentials, as the state agent is the one who issues these in this project.

Table 4.4: Endpoints for the state agent

Endpoint name	relation	status codes	Request method	Description
/state/did	DID	201, 400	POST	Create a new DID
/state/did/:did	DID	200, 400	GET	Get a DID with DID-url
/state/dids	DID	200, 500	GET	Lists all the agent's DIDs
/state/resolve/:did	DID	200, 400	GET	Resolves DID with DID-url
/state/credential	CREDENTIAL	200, 400	POST	Add a new credential
/state/credentials	CREDENTIAL	200, 400	GET	Get a credential
/state/personCredential	CREDENTIAL	201, 400, 500	POST	Create person credential
/state/businessCredential	CREDENTIAL	201, 400, 500	POST	Create business credential
/state/credential/:type	CREDENTIAL	200, 400	GET	Get credential by type
/state/presentation	PRESENTATION	201, 400, 500	POST	Create a verifiable presentation
/state/verifyJWT	PRESENTATION	400, 500	POST	Verify a JWT
/state/person	DATABASE	201, 400, 500	POST	Create person in database
/state/person/:id	DATABASE	200, 400, 500	GET	Get person from database
/state/person/:id	DATABASE	200, 400, 500	DELETE	Delete person from database
/state/messaging	MESSAGE	200, 400, 500	POST	DIDComm messaging
/state/sendMessage	MESSAGE	200, 400	POST	Send DIDComm message

Chapter 5

Front end design

When designing the front end of the application, the goal was to be as practical and easy to navigate while still adhering to sound design principles. The different agents can be deployed on both android and the web, so the focus was on making it look like an android application. This is because the application must be deployed on both web and android, even if the agents would typically use a website. It is easier to deploy an app made for android on the web than the other way around.

All the buttons are centered in the middle of the screen to the better fit on a phone screen. The buttons have rounded edges, which is more common in mobile design. Instead of utilizing all the space on a computer screen, we opted for a scroll bar when the content needed more space. The application uses a simple black and white color scheme, with each agent having its distinct colors and styles on buttons. When choosing the agent-specific colors, we used colors representing the real-life counterparts of our agents used on their website. Blue was used for the EU, red for NAV, purple for the state, and the signature blue color for Symfoni.

The application is functionally four applications in one and is designed around that fact. All the agents have a page with the actions they perform in this scenario. This is to make the demo easy to use and understand. Not every page is designed around a real-life scenario but adapted to fit a demo for showcasing the concepts.

5.1 Login page

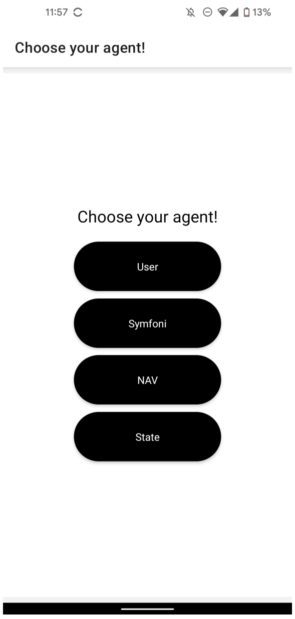


Figure 5.1: Login Page

The first page when one opens the application is the login page. It has four buttons, one for each agent. To log in to an agent, click the agent's respective button, and it navigates to that agent's homepage without any verification needed. This could have been a login with a username and password, which takes one to the right agent based on one's status. We decided against this, as it does not add anything to the SSI concept the project is demonstrating. Each agent would be an independent application in a real-world scenario, and we would not encounter this problem. The goal is to have four agents sending and verifying credentials between each other, so the login part is simplified for the demo.

5.2 User

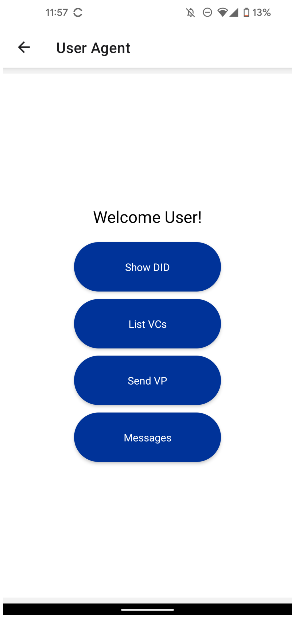
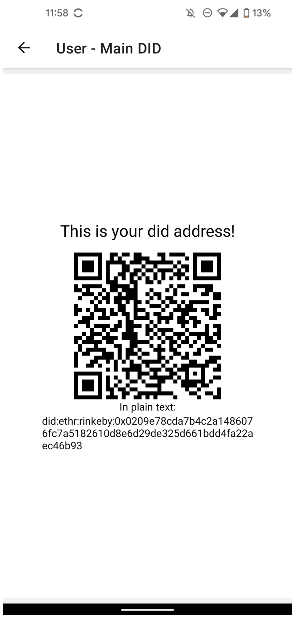


Figure 5.2: User home page

The user agent is the agent with the most options. It has four buttons: show DID, List VC, Send VP, and Messages. The user agent is designed for being used on a phone, not a computer. In several cases, one must scan QR codes to establish communication channels with other agents. This is most easily done with a phone and is a big part of why we designed with android in mind.

5.2.1 Show DID



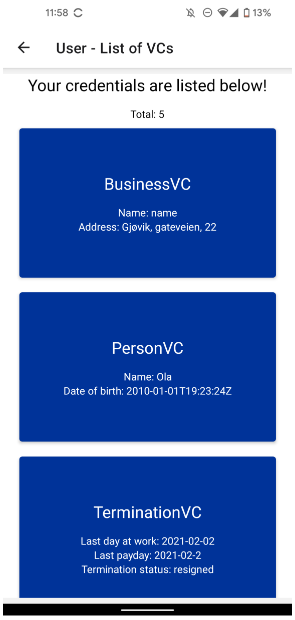
This is your did address!

In plain text:
did:ethr:rinkeby:0x0209e78cda7b4c2a148607
6fc7a5182610d8e6d29de325d661bdd4fa22a
ec46b93

Figure 5.3: User display decentralized identifier in a QR code and plain text

Show DID displays the main DID created by the user agent. It displays this in two ways. Firstly it is a QR code with the DID-URL as the payload. This is if the user wants to send their DID to someone else with QR code reading capabilities. Secondly, the DID-URL is shown in plain text, which makes it easy to copy and paste the DID-URL. It is impossible to remember a DID-URL, so it is necessary to have the option to copy the URL and paste it when a DID-URL is required input.

5.2.2 List VCs



Your credentials are listed below!

Total: 5

BusinessVC

Name: name
Address: Gjøvik, gateveien, 22

PersonVC

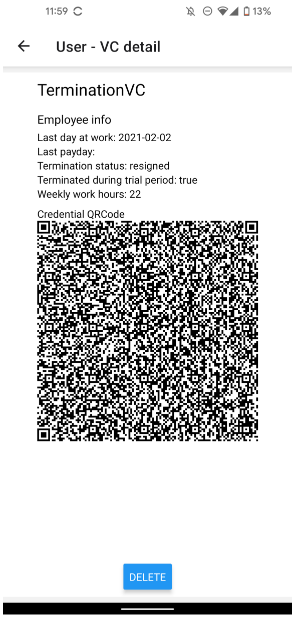
Name: Ola
Date of birth: 2010-01-01T19:23:24Z

TerminationVC

Last day at work: 2021-02-02
Last payday: 2021-02-2
Termination status: resigned

Figure 5.4: Stored verifiable credentials list

List VCs is a list of verifiable credentials that are stored in the user's wallet. The first page takes one to a list with cards representing different VCs. The cards display the credential type and have the essential information for identifying the credential. A person's credentials display name and date of birth; a business credential has the business name, address, and more. This is to distinguish different credentials of the same type and give context to what information the credential contains.



TerminationVC

Employee info

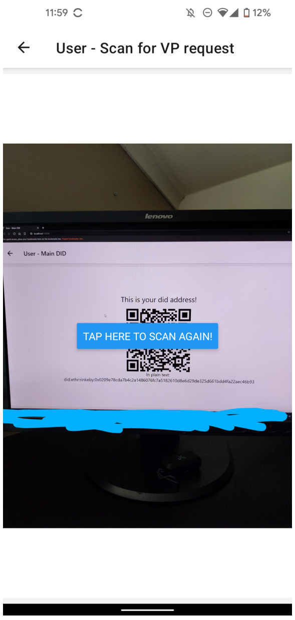
Last day at work: 2021-02-02
Last payday:
Termination status: resigned
Terminated during trial period: true
Weekly work hours: 22

Credential QRCode

Figure 5.5: Verifiable credential detail view

By clicking on one of the cards, it takes one to a detailed view of the credential. In this view, all the credential information is displayed. The information is structured like the schema used to create the credential. Like on previous pages, there is a QR code that contains the credential. At the bottom of the page is the delete button for removing the current credential. This has a popup with verification, as deleting credentials by mistake can have consequences. If one's person credential works as one's online identity, deleting it limits one's ability to interact with the SSI system.

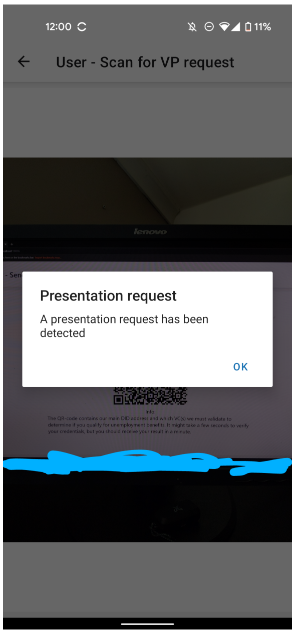
5.2.3 Send VP



[redacted]

Figure 5.6: Verifiable presentation request scanner

Send VP opens a QR reader, and the user scans a QR code provided by another agent. This is when another agent wants to set up a connection, and they display a QR with their request on their website. There is no website in this demo, just another page in the application, and this means one would need several devices to perform this action. After this, there is a popup that a verifiable presentation request has been detected.

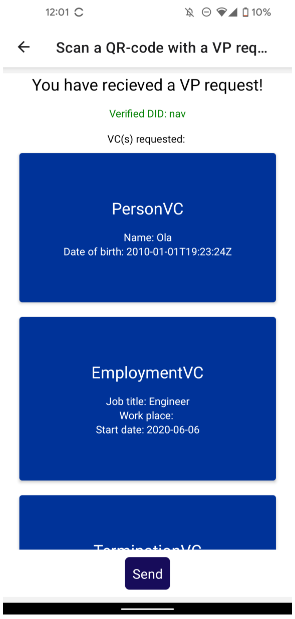


Presentation request

A presentation request has been detected

[redacted]

Figure 5.7: Verifiable presentation request alert



You have recieved a VP request!

Verified DID: nav

VC(s) requested:

PersonVC

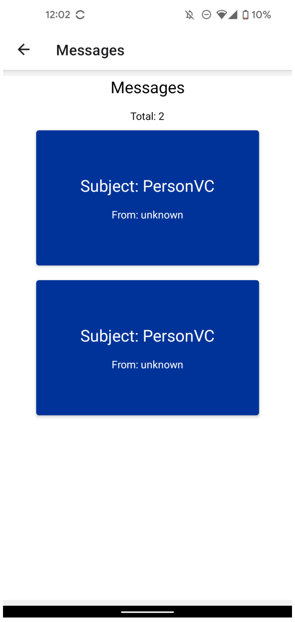
Name: Ola
Date of birth: 2010-01-01T19:23:24Z

EmploymentVC

Job title: Engineer
Work place:
Start date: 2020-06-06

Figure 5.8: User receive verifiable presentation request

The VP request uses the same design as the list VCs, as there are cards with basic information regarding the VCs that are requested. The green text above the credential list confirms if the DID persists in the verifiable registry. In this case, the DID is verified and belongs to NAV. By pressing "send," the user consents to share the requested information. By declining to send this information, the action one is trying to unlock with one's VP is declined. User control over sharing credentials is a core part of SSI and is included in our project.



Messages

Total: 2

Subject: PersonVC

From: unknown

Subject: PersonVC

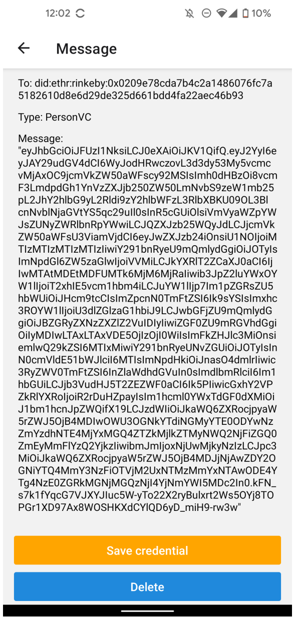
From: unknown

Figure 5.9: User message inbox

Finally, when one presses the send button, a popup appears confirming that the VP was sent.

5.2.4 Messages

The credentials are sent to the user agent as messages using DIDComm messaging. To save a credential, one needs to open the messages page. In the message inbox, messages are displayed in the same way credentials are listed in list VCs.



To: did:ethr:rinkeby:0x0209e78cda7b4c2a1486076fc7a5182610d8e6d29de325d661bdd4fa22aec46b93

Type: PersonVC

Message:

```
"eyJhbGciOiJFUzI1NksiLCJ0eXAiOiJKV1QifQ.eyJ2YyI6e
yJAY29udGV4dCI6WyJodHRwczovL3d3dy53My5vcmc
vMjAxOC9jcmVkZW50aWFscy92MSIsImh0dHBzOi8vcm
F3LmdpdGh1YnVzZXJjb250ZW50LmNvbS9zeW1mb25
pL2JhY2hlbG9yL2Rldi9zY2hlbWFzL3RlbXBKU09OL3Bl
cnNvblNjaGVtYS5qc29uIl0sInR5cGUiOlsiVmVyaWZpYW
JsZUNyZWRlbnRpYWwiLCJQZXJzb25WQyJdLCJjcmVk
ZW50aWFsU3ViamVjdCI6eyJwZXJzb24iOnsiU1NOIjoiM
TIzMTIzMTIzMTIzIiwiY291bnRyeU9mQmlydGgiOiJOTyIs
ImNpdGl6ZW5zaGlwIjoiVVMiLCJkYXRlT2ZCaXJ0aCI6Ij
IwMTAtMDEtMDFUMTk6MjM6MjRaIiwib3JpZ2luYWxOY
W1lIjoiT2xhIE5vcm1hbm4iLCJuYW1lIjp7Im1pZGRsZU5
hbWUiOiJHcm9tcCIsImZpcnN0TmFtZSI6Ik9sYSIsImxhc
3ROYW1lIjoiU3dlZGlzaG1hbiJ9LCJwbGFjZU9mQmlydG
giOiJBZGRyZXNzZXZlZ2VuIDIyIiwiZGF0ZU9mRGVhdGgi
OiIyMDIwLTAxLTAxVDE5OjIzOjI0WiIsImFkZHJlc3MiOnsi
emlwQ29kZSI6MTIxMiwiY291bnRyeUNvZGUiOiJOTyIsIn
N0cmVldE51bWJlciI6MTIsImNpdHkiOiJnasO4dmlrIiwic
3RyZWV0TmFtZSI6InZlaWdhdGVuIn0sImdlbmRlciI6Im1
hbGUiLCJjb3VudHJ5T2ZEZWF0aCI6Ik5PIiwicGxhY2VP
ZkRlYXRoIjoiR2rDuHZpayIsIm1hcml0YWxTdGF0dXMiOi
J1bm1hcnJpZWQifX19LCJzdWIiOiJkaWQ6ZXRocjpyaW
5rZWJ5OjB4MDIwOWU3OGNkYTdiNGMyYTE0ODYwNz
ZmYzdhNTE4MjYxMGQ4ZTZkMjlkZTMyNWQ2NjFiZGQ0
ZmEyMmFlYzQ2YjkzIiwibmJmIjoxNjUwMjkyNzIzLCJpc3
MiOiJkaWQ6ZXRocjpyaW5rZWJ5OjB4MDJjNjAwZDY2O
GNiYTQ4MmY3NzFiOTVjM2UxNTMzMmYxNTAwODE4Y
Tg4NzE0ZGRkMGNjMGQzNjI4YjNmYWI5MDc2In0.kFN_
s7k1fYqcG7VJXYJIuc5W-yTo22X2ryBulxrt2Ws5OYj8TO
PGr1XD97Ax8WOSHKXdCYlQD6yD_miH9-rw3w"
```

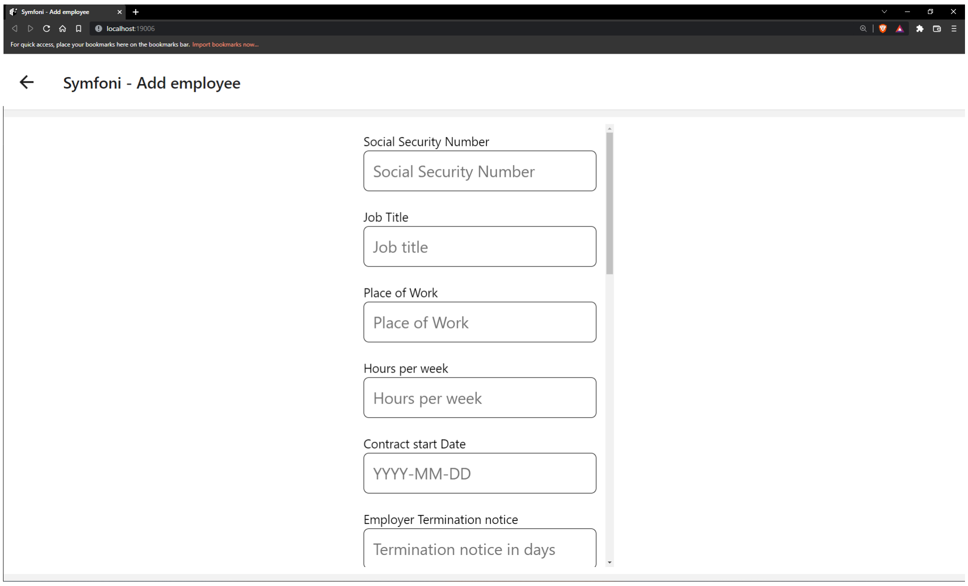
Figure 5.10: Message details

Clicking one of the cards takes one to the message details. At the top of the screen is the DID the message is addressed to, and the credential type in the message body. This is one area of the project that needs further work. The message body is the JWT containing all the information of the credentials the user requested. A JWT is not human-readable, and it does not look visually appealing having a large block of text in the message. There are two options with a message, save the credential and delete the message. If a user requests many credentials, it can quickly fill up the inbox. In addition, the messages are not thoroughly documented, so it can be challenging to see who sent the message and what it contains. Therefore, it is practical to delete messages when one has saved the credential. A popup appears when the credential is saved, confirming that the credential was saved.

5.3 Symfoni

Like the login page, the Symfoni page has been simplified for demo reasons. It is split into employee and employer actions. Usually, this would be a website, and when one logs in, the appropriate actions for one's role would be visible. Again this adds coding work that does not have any benefit for the SSI part of the bachelor. Therefore, both employer and employee actions are on the Symfoni page.

5.3.1 Add employee


Symfoni - Add employee
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Symfoni - Add employee

Social Security Number
Social Security Number

Job Title
Job title

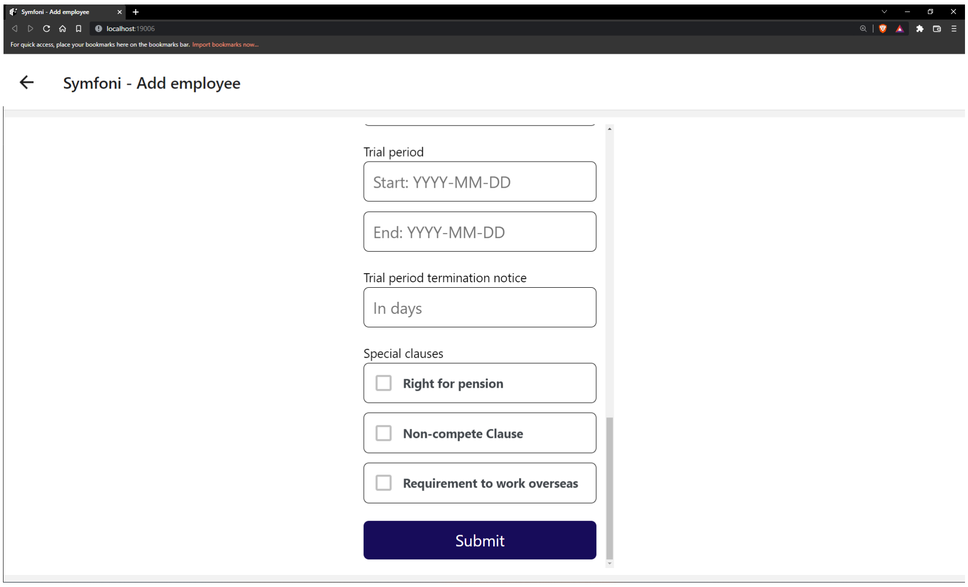
Place of Work
Place of Work

Hours per week
Hours per week

Contract start Date
YYYY-MM-DD

Employer Termination notice
Termination notice in days

Figure 5.11: Employment form part 1


Symfoni - Add employee
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Symfoni - Add employee

Trial period
Start: YYYY-MM-DD
End: YYYY-MM-DD

Trial period termination notice
In days

Special clauses
□ Right for pension
□ Non-compete Clause
□ Requirement to work overseas

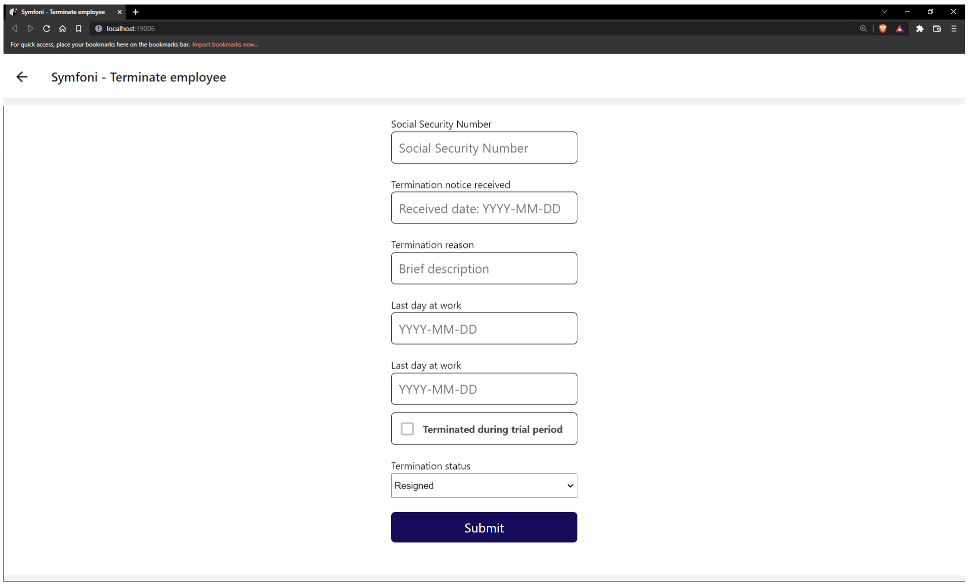
Submit

Figure 5.12: Employment form part 2

Add employee is an employer action and takes one to a form with employee information. Each field has a title and a placeholder to make it easier to fill in the correct information in the correct format. Employment status has a dropdown menu, as there are only two valid options for this field. The three last fields are yes or no questions, so they have checkboxes to make it more intuitive to fill in.

Add employee is one of the pages where the web and android design patterns conflict. It is a long form that needs to be scroll-able to fit on a phone screen. This could be avoided in web deployment, as more screen space can be utilized. One could argue that the Symfoni agent will usually be used on a computer and that this page should have been designed around a web deployment. However, the team decided that deploying on android was more important, and it gives the GUI a consistent look.

5.3.2 Terminate employee

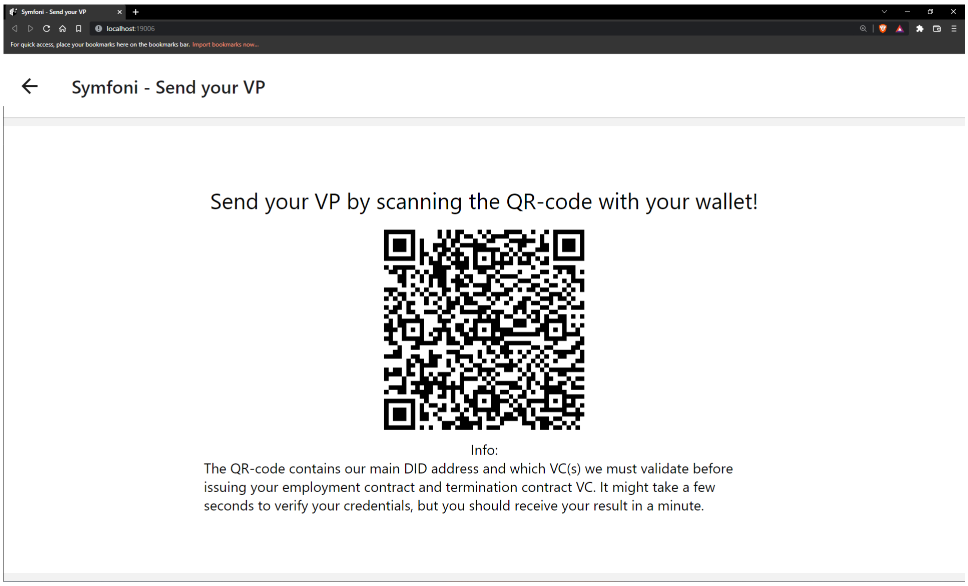

Symfoni - Terminate employee
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
Symfoni - Terminate employee
Social Security Number
Social Security Number
Termination notice received
Received date: YYYY-MM-DD
Termination reason
Brief description
Last day at work
YYYY-MM-DD
Last day at work
YYYY-MM-DD
□ Terminated during trial period
Submit

Termination status: Resigned

Figure 5.13: Termination form

Terminate employee is designed with the same principles as add employee. It is a form where each field has a title and a placeholder. There is one checkbox for yes/no questions and a drop-down menu for termination status. Error messages are displayed when the user clicks the submit button, as the form is not submitted until all the information is in the correct format. The form for terminating an employee is much smaller than the previous form, so the problem of needing to scroll is less prominent.

5.3.3 Request VCs


Symfoni - Send your VP
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Symfoni - Send your VP

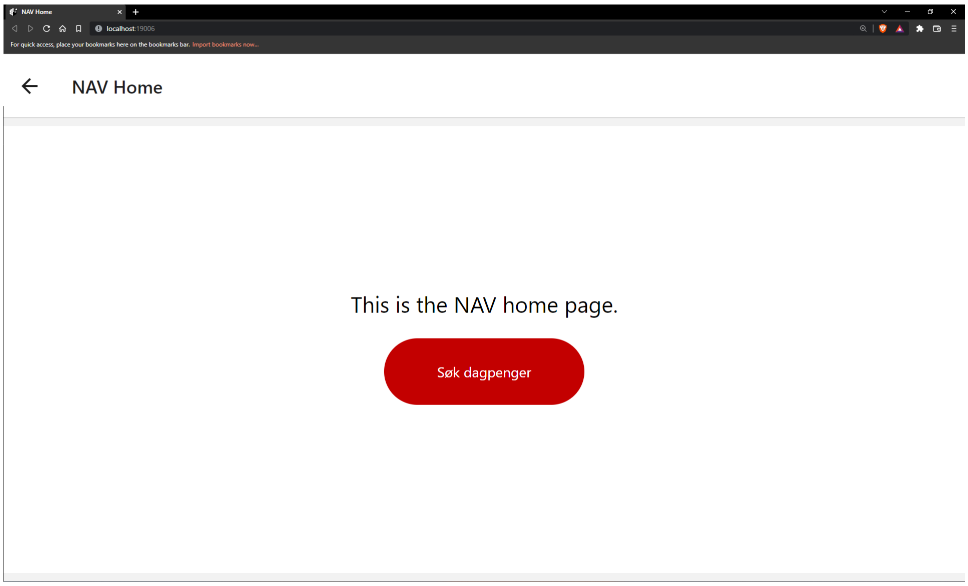
Send your VP by scanning the QR-code with your wallet!

Info:
The QR-code contains our main DID address and which VC(s) we must validate before issuing your employment contract and termination contract VC. It might take a few seconds to verify your credentials, but you should receive your result in a minute.

Figure 5.14: Symfoni request credentials

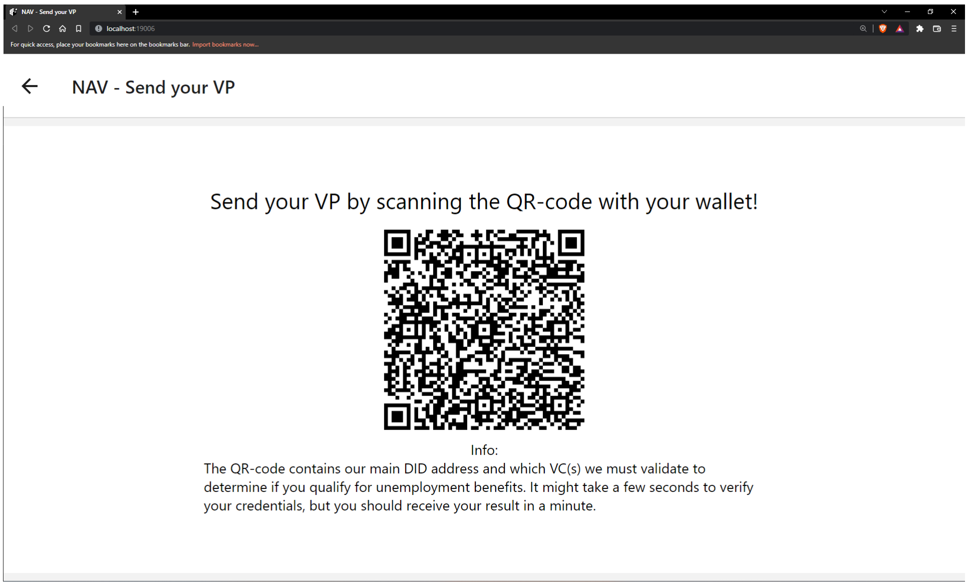
Request VC is an employee action and would normally be performed on the company website, not in the same application as the user agent. However, in this project, it is in the same application. By clicking the request VCs button, it takes the user to a new page with a QR code. This QR code requests a VP, so the user has to get to the send VP QR-scanner on their phone and scan this QR code. Then it goes through the send VP stages described in Figure 5.6.

5.4 NAV


NAV Home
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
NAV Home
Søk dagpenger

This is the NAV home page.

Figure 5.15: NAV home page


NAV - Send your VP
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
NAV - Send your VP

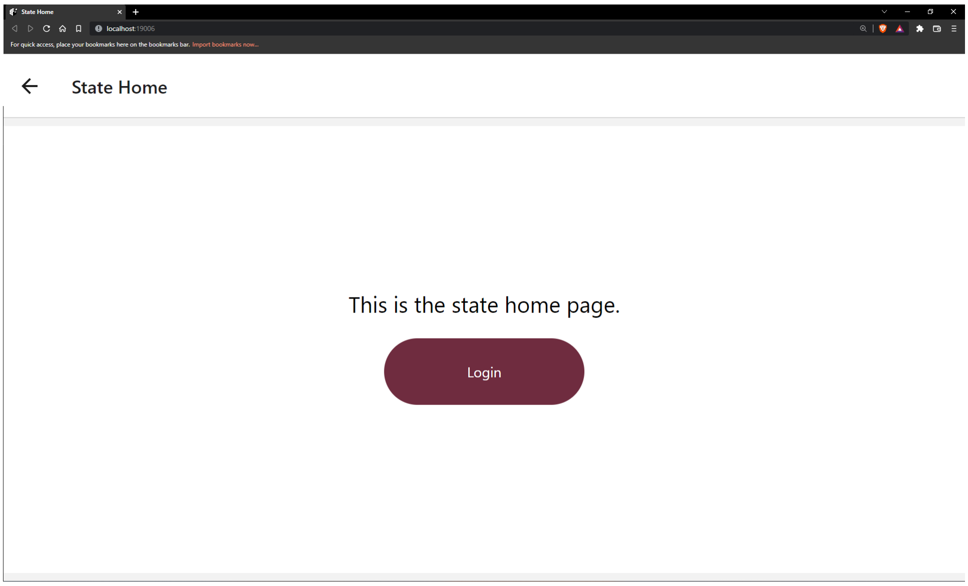
Send your VP by scanning the QR-code with your wallet!

Info:
The QR-code contains our main DID address and which VC(s) we must validate to determine if you qualify for unemployment benefits. It might take a few seconds to verify your credentials, but you should receive your result in a minute.

Figure 5.16: NAV verifiable presentation request

The NAV homepage only has one button, "søk dagpenger," which is Norwegian for "apply for unemployment benefits." This brings up a QR code that contains NAV's DID and a request for the VCs required to get unemployment benefits. The images are from the web build.

5.5 State


State Home
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

State Home

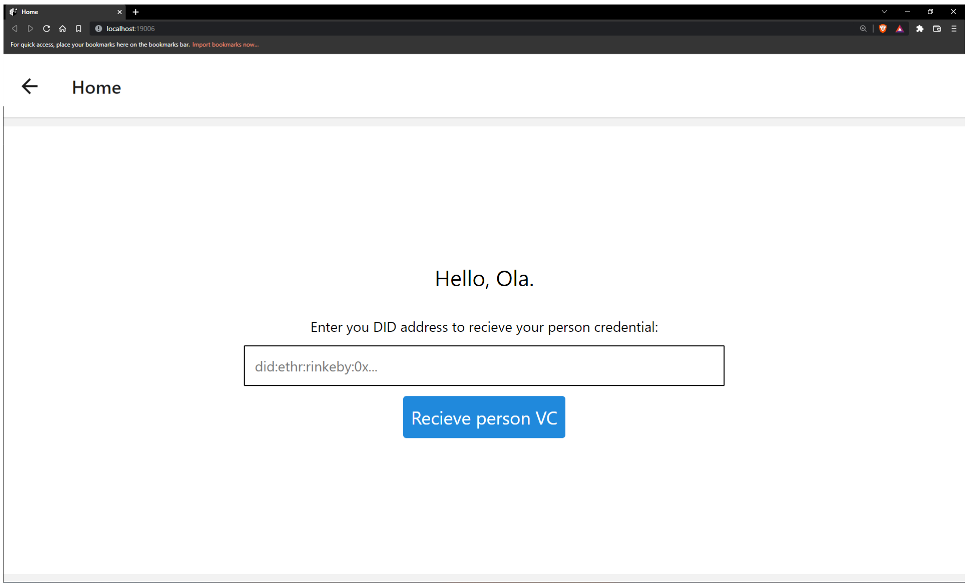
This is the state home page.

Figure 5.17: State home page

The state agent has also been made to suit a demo, not reflect a real-life scenario. Typically a user would go to a government website and request a person's credential. Signing in, one would need some identification; it would be natural to use Bank-ID in Norway. We did not have access to integrating Bank-ID into our application. Instead, we use a fake social security number saved in a database.

It was possible to simply make a website with minimal functionality and create credentials from data stored on the website. We preferred to keep everything in the same project. Therefore, the user request person credentials from the state in the same application. The core concept of requesting credentials from a government agency is still there, so it does not go against the project's aims.

The state login page requires a social security number to request a person's credential. It has error messages for wrong numbers and a placeholder that says what the input should be. A correct social security number takes one to the next page.


Home
localhost:19006
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
Home

Hello, Ola.

Enter you DID address to recieve your person credential:

did:ethr:rinkeby:0x...

Figure 5.18: State user page

This page has a personal greeting based on one's social security number because the individual is a person who exists in the government database. To receive a verifiable credential, the issuer needs one's DID-URL. This is where it is helpful to copy the DID URL from show DID in the user agent.

5.6 Wireframe

Before developing the front end, we created a low fidelity prototype in Figma. A low-fidelity prototype is for getting feedback on the structure and look of an application before starting development. Programming is expensive and time-consuming; gathering user feedback before writing code can save both time and money. A link to the prototype as well as images of the wireframe can be found in Appendix G.

The general look of the application stayed the same, but some changes were made. These changes were either because we changed the application's logic or removed because it was deemed unnecessary.

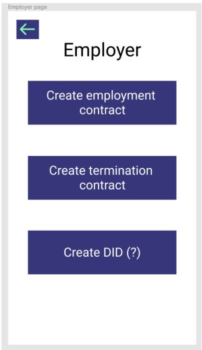


Figure 5.19: Figma employer page

The create DID view was no longer necessary, as it is handled in the back end. In the finished version, the contract buttons are the same. Instead of creating credentials directly, they add information into the database, which is later turned into credentials. Because of this, we needed to add a button for an employee to request their credentials.

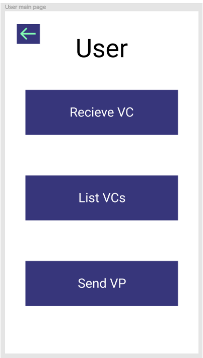


Figure 5.20: Figma user home page

The user page has gone through the most iterations. This is because we did not have the transport layer ready in the early stages when this diagram was made. By using DIDComm, we needed to be able to handle messages, and this led to the messaging endpoints being required. Having an option to show the user's DID was added after each agent got the main identifier. 5.2

5.7 Abandoned changes

A variety of features did not make it to the end product. Due to time constraints, features that did not contribute to showcasing the SSI ecosystem were deprioritized. This does not mean that the front end has no features, just that some quality of life features were not implemented.

The main page consists of four buttons that navigate to the different agents in the final product. We wanted to implement a login page with actual logins instead of just buttons to make the app more professional. One would automatically be taken to the correct page when logging in based on one's role. After consulting with Symfoni, this approach was abandoned, and the current design was chosen. Adding even more databases complicates the project unnecessarily. It also disrupts the flow when demonstrating the product, as one would need to log in every time one goes to a new agent.

Initially, we wanted to split the Symfoni page into two separate pages, one for employees and one for the employer. This would be handled with a login that takes one to the right page based on one's role. Employees would only see the "receive VC" and employers would see the "terminate employee" and "add employee." This was changed to the current implementation by using the same logic as for the main login page. It complicates the project and does not add anything regarding SSI.

Several interactions involve scanning QR codes to establish communication between agents. We debated who would have a QR code and who would scan them. The first iteration was that the user would supply the QR code when they started transactions. This design was inspired by observing other crypto wallets. Our application operates differently from crypto wallets, as the users interact with government agencies and businesses, not other users. It would be more natural for the user to scan the QR codes, not for a government employee to scan one. So we changed the design to the user scanning the QR codes and other agencies supplying them. With cryptocurrency, it is normal for users to trade with other users, but with SSI, it is uncommon for users to send credentials to each other.

Chapter 6

Development process

The development process chapter is about how the group worked to finalize the project. It presents the meetings that were arranged and the development framework used during development.

6.1 Meetings

During development, there were four types of meetings: weekly meetings with the employer Symfoni, guidance meetings with the faculty supervisor, business meetings with NAV, and internal meetings with the group.

6.1.1 Symfoni meetings

Every Monday at 15:00, we had a scheduled meeting with Symfoni, the private company that gave the task for the bachelor thesis. In these meetings, we go over what the group has accomplished this week, discuss problems we have encountered, and plot out what to do for next week.

In these meetings, we were either communicating with Jon Ramvi, the CEO and owner of Symfoni or Asbjørn, one of the developers working on SSI for Symfoni. In the earlier meetings, the discussion focused on the big picture, the project's overall design, and what methods and technologies to use. Jon Ramvi provided us with guidance on how to proceed and explained different concepts regarding SSI which was unfamiliar to the group.

Later in the project, there were more technical meetings with Asbjørn. Technical meetings were either scheduled where the students had prepared questions or more impromptu where the group reached out for assistance on a subject. In the technical meetings, Asbjørn provided examples we could study.

In addition to these scheduled meetings, there has also been asynchronous communication in the Symfoni discord channel. In this channel, both administrative and technical details were discussed.

Meeting notes from the meetings with Symfoni can be found in Appendix D.

6.1.2 NAV meetings

During the project, we have been collaborating with NAV to make our project's use case more in line with a real-life scenario. The VCs used in the project are based on information provided by NAV so that they can be used in their work managing unemployment benefits applications.

In addition to getting information from NAV, we have developed several diagrams for demonstrating the flow of an SSI ecosystem. These were used to demonstrate how a user would interact with all the different entities in the system. This worked as an introduction for someone who had no previous knowledge about SSI and how it could be implemented for NAV.

Meeting notes from the meetings with NAV can be found in Appendix E.

6.1.3 Supervisor meetings

Our supervisor is Mariusz Nowostawski from the faculty of information technology. Initially, we planned to have weekly meetings, but due to unavailability from Mariusz in the early stages, they became more infrequent. These meetings focused more on the University side of the bachelor thesis and questions about scope, the report, and project structure. Towards the end of the project, he provided guidance and feedback for writing the bachelor thesis.

Meeting notes from the meetings with our supervisor can be found in Appendix F.

6.2 Development model

For this project, we have been using the SCRUM framework. We have not strictly followed the SCRUM methods but adapted the concepts to fit our workflow.

6.2.1 Sprints and sprint review

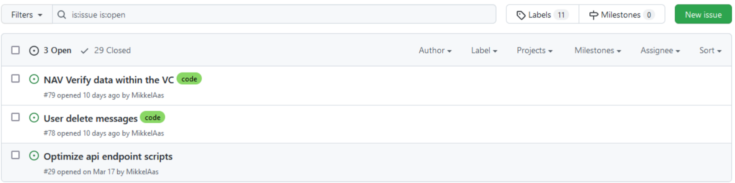
For this project, we have chosen to use the development method Scrum. Scrum is an agile development method where the work is done in short intervals called sprints. This could include coding, documentation, project planning, and more. Agile development is an iterative method where self-reliant teams develop software through a dynamic work process. Another essential feature is the kanban board or the Scrum board, where different tasks are defined and assigned for the coming sprint.

We chose Scrum for several reasons, mainly because it is a development method the team is familiar with and enjoys. The project is organized around weekly meetings with the employer, thus making it natural to organize weekly or bi-weekly sprints[14]. The kanban board allows a more fluid work environment, where one can assign new problems without the rigid structure of more traditional development methods.

The meetings with Symfoni worked as both sprint review, and sprint planning [14]. We went through what we had accomplished in the previous sprint and planned what to do in the next sprint. In retrospect, we should have split this process up into two parts, as it is easier to focus on one thing, and details may be overlooked. There were meeting notes taken during the meetings that worked as a sprint review, and these could have been written more thoroughly, and the team would rely less on oral information.

6.2.2 Issue tracker

For organizing the work, we have used a combination of the GitHub issue tracker and the meeting notes from the weekly Symfoni meetings. The issue tracker makes it easier to organize tasks into manageable chunks. It would also have been possible to assign issues to a developer, but our group usually did this orally.



□ ⊙ 3 Open ✓ 29 Closed

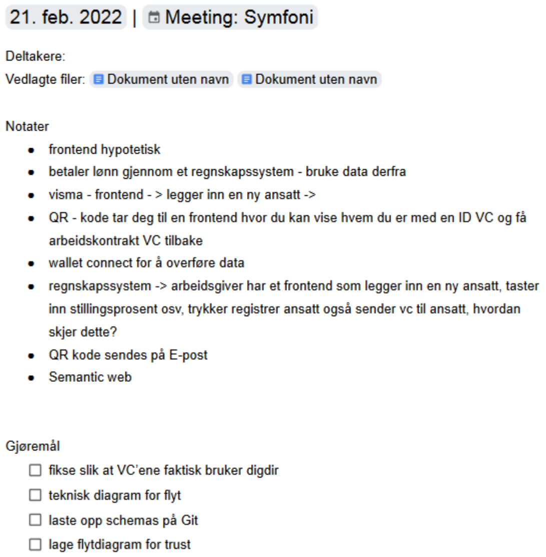
□ ⊙ NAV Verify data within the VC code
#79 opened 10 days ago by MikkelAas

□ ⊙ User delete messages code
#78 opened 10 days ago by MikkelAas

□ ⊙ Optimize api endpoint scripts
#29 opened on Mar 17 by MikkelAas

Figure 6.1: Issue tracker

The meeting notes are split into two parts: a summary and reflection of our conversation with Symfoni and bullet points on what to do for the next sprint. There is no section on what the group did in the last sprint, as that was the basis for the conversation with Symfoni. Adding a reflection on the last spring would have been valuable and something to consider for the next project. In Figure 6.2 is a typical example of meeting notes for one of the weekly sprints. In this particular meeting, we discuss the flow of adding new credentials. We had more code-oriented questions in later meetings which were not documented.



21. feb. 2022 | Meeting: Symfoni

Deltakere:

Vedlagte filer: Dokument uten navn Dokument uten navn

Notater

- frontend hypotetisk
- betaler lønn gjennom et regnskapssystem - bruke data derfra
- visma - frontend - > legger inn en ny ansatt ->
- QR - kode tar deg til en frontend hvor du kan vise hvem du er med en ID VC og få arbeidskontrakt VC tilbake
- wallet connect for å overføre data
- regnskapssystem -> arbeidsgiver har et frontend som legger inn en ny ansatt, taster inn stillingsprosent osv, trykker registrer ansatt også sender vc til ansatt, hvordan skjer dette?
- QR kode sendes på E-post
- Semantic web

Gjøremål

☐ fikse slik at VC'ene faktisk bruker digdir

☐ teknisk diagram for flyt

☐ laste opp schemas på Git

☐ lage flytdiagram for trust

Figure 6.2: Meeting notes

This is one of the aspects of the project that has not worked as intended. With a small development team and little oversight, it is easy to fall into the anti-pattern of not using the issue tracker and simply focusing on coding. This disrupts the scrum structure and makes it harder to document the work that has been done.

6.2.3 Code review

In more traditional versions of SCRUM, they use daily-scrum, which is everyone on the team briefly describes what they are working on every day [14]. Due to the size of our team, we chose not to use this approach. Instead, we keep each other apprised of what everyone is doing by using code-review. This is not optimal, as one gets the information when people are done instead of when people start. This was not a problem; due to the size and frequent communication between the team members, everyone had a good grasp of the different tasks being performed. Instead, one gets an in-depth review of the code produced when they are ready to submit.

To merge a branch into development, it has to be approved by a code review. Code review was done in two steps. First, the developer who has written the code goes through it step by explaining the functionality and showing how it interacts with the rest of the project. When work started on the front end, we ensured at least two developers could run the front end both on the web and on Android. After that, a different developer goes through all the code on GitHub, looking for errors or inconsistencies. If anything is brought up, the initial developer changes the files in the pull request based on feedback. The review is approved and finally merged.

This type of code review serves several purposes. It ensures code quality and that the logic is internally consistent with the rest of the project. It catches errors and ensures that only working code is merged into the dev branch. This way of doing code review makes sure all developers understand what everyone is doing and that they know what functionality is added to the program.

6.2.4 Reflection

In the later part of the project, we conducted a retrospective about the development process. The meeting notes were not as extensive as possible and very focused on tasks, not reflection. The reflection part was done in our conversations with Symfoni and internally in the group. Reflection was present, but there were few written notes on our thoughts. The rough outline is in our meeting notes, and that was enough for us to develop the project, but it is hard to go back and do a retrospective with a detailed accounting of our thoughts during development. If this was a larger project with a bigger team, more documentation would have been necessary for everyone to know what to do.

The group was very task-oriented and wanted to accomplish goals and milestones set early in the project. One example was the decision to use Wallet Connect or a different service to exchange credentials between wallets. The choice to use Wallet Connect was made very early, as that was what Symfoni was using. We set implementing Wallet connect as a goal without discussing possible alternatives. This was partly because the team was unfamiliar with the field. We did discuss that we only needed to transfer the JWT token, and how that was achieved was not important. However, we tried making wallet connect work for a long time before finding an alternative. Being more flexible is a takeaway, but one reason for the lack of flexibility is working in an unknown field. We could have decided on the transport layer faster if we had more knowledge.

One of the most invaluable contributions was the weekly meetings with Symfoni. Having an authority on SSI to guide us through the concepts and how the project would be structured was an immense contribution. Not only did they provide knowledge if we had questions, but we also had a dialog where we could pitch our ideas and get feedback that way. Jon always encouraged us to make our own solutions, as long as there was a rationale behind it.

Chapter 7

Implementation

The implementation chapter goes into detail on how the project was implemented. It consists of four parts; the Veramo digital agents, firestorm database, the API, and React Native. For each part, code examples will be explained in detail.

7.1 Tools

This section contains a list of tools used during the development process of the bachelor thesis.

- **Veramo.io** is a modular API for Verifiable Data and SSI. The API makes it possible to use verifiable credentials and Decentralised Identifiers without managing compatibility issues.
- **Node.js** is a run time environment used for traditional websites and back end APIs.
- **Express.js** is a web application framework for node.js that is used to create the API.
- **Firestore** is a NoSQL document-based database. It persists in the cloud, in contrast to the local databases used by the different agents.
- **SQLite** is a software library that provides a relational database system. SQLite is a server-less database and self-contained, so the database is part of the application.
- **GitHub** is a version control and code management system. The reason for choosing GitHub was mainly because we got a repository from our employer. This way, we uphold their open-source software standards, as it is a public repository on their GitHub page.
- **Husky** is a NPM package that allows custom scripts to be run against a repository. We used husky to ensure that our code was lint-free and that our tests passed on each commit.
- **Mocha** is a JavaScript testing framework used for unit testing in this project.
- **React Native** is an open-source JavaScript framework for developing mobile applications. This enables the developers to create an application for

multiple platforms.

- **Expo** is a framework for building react native apps. It comes with tools that simplify the creation and testing of react native apps. It also provides a more robust and convenient development workflow.
- **TypeScript** is the primary and only programming language used for this project. TypeScript is an extension of JavaScript. It is an object-oriented programming language rather than a pure scripting language. The only drawback is the lack of compatibility with other programs; one must perform extra steps to make it work. An example of this is Node.js requiring "ts-node" for running TypeScript scripts.

7.2 Building

The Expo framework was used to build the project. This was mainly because of two reasons, the first being that the application was supposed to be able to run on the web and a phone, and Expo can build to both. The second is that Expo allows rapid development as the developer does not have to write any native code. However, this comes at the price of using Expo-supported dependencies. For example, we could not use the standard React Native bar code scanning component but had to use Expos self-managed one.

Using the Expo framework with Veramo proved to be complicated. As mentioned earlier, the Expo framework only supported Expo-friendly dependencies, and Veramo was omitted. This was discovered late in the project, and it presented us with two options. We could eject out of Expo and begin writing native code, hoping to manage the dependencies through Gradle, or completely separate Veramo from our GUI by having the GUI interact with a server that interacts with Veramo. As none of us were comfortable with ejecting from Expo, we chose to have all Veramo functionality in an independent server.

Code listing 7.1: Continuous integration android build script

```
name: Android Build

on:
  push:
    branches: [ dev ]
  pull_request:
    branches: [ dev ]

jobs:
  build:
    name: Install and build
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: 16.x
          cache: npm
      - name: Setup Expo and EAS
```

```
        uses: expo/expo-github-action@v7
        with:
          expo-version: 5.x
          eas-version: latest
          token: ${{ secrets.EXPO_TOKEN }}
      - name: Install dependencies
        run: npm ci
      - name: Build on EAS for Android
        run: eas build --platform android --non-interactive
```

Implementing continuous integration with the Expo framework was very easy. We had to create a GitHub action for Node.js and add the client commands for building to both web and android. Though we could have built for both platforms in one script, we chose to have separate scripts as one build could fail while the succeeded. The web build is done locally, but the android build is a little trickier as it has to be sent to the Expo server for building. Access to this server is achieved by a secret token stored in GitHub secrets.

7.3 Veramo agents

The project is contains five different Veramo agents. The four agents represent different entities in this SSI ecosystem and one test agent. The different agents are the user agent, NAV agent, Symfoni agent (employer), and state agent.

7.3.1 Setup

All the digital agents are set up in the setup.ts file, inside the Veramo folder. The agents are created by importing libraries from the Veramo core. When creating a new agent with Veramo, different interfaces are defined. These interfaces are classes in the Veramo API with different functionality, either from the Veramo core or additional classes. This is to ensure that all the agents have the required functionality for the program to perform their actions in the SSI ecosystem. Each agent requires a SQLite database where the DIDs and credentials are stored. This database works as a digital wallet in our scenario. This is done five times, one for each agent with the correct name and database. Below is an example of creating a new DIDManager, one of the packages from the Veramo core. Here one manually sets what kind of DID-methods the manager can create. The default is ’did:ethr:rinkeby’, with the option to create ’did:web’ For the agent setup see appendix Appendix J

Code listing 7.2: Creating a new DIDManager

```
new DIDManager({
        store: new DIDStore(dbConnectionNAV),
        defaultProvider: 'did:ethr:rinkeby',
        providers: {
            'did:ethr:rinkeby': new EthrDIDProvider({
                defaultKms: 'local',
                network: 'rinkeby',
                rpcUrl: 'https://rinkeby.infura.io/v3/'
                + INFURA_ID,
            }),
            'did:web': new WebDIDProvider({
                defaultKms: 'local',
            }),
        },
    })
```

7.3.2 Main functionality

One of the central functions is "getMainIdentifier". The project supports having several DIDs for each agent, but to make it easier, we always use the main identifier. When a new agent is created, one of the inputs is the main identifier alias. By setting this up in the constructor, we control what the main identifier should be called. This becomes useful when we call the ’getMainIdentifier’ function.

```
export class StateAgentController extends AgentController
implements IStateAgentController{
    constructor(mainIdentifierAlias: string) {
        super(agentState, mainIdentifierAlias);
    }
```

This function takes no input, as the main identifier alias is predefined. First, it tries to get a request to see if the main identifier already exists. If it exists, it retrieves the main identifier from the agent where it was called. If it does not exist, it creates and returns a new main identifier. We ensure that an agent always has the main identifier using this function. As long as we use this consistently, we never encounter the problem of an agent not having a DID.

Code listing 7.3: The get main identifier function

```
async getMainIdentifier(): Promise<IIdentifier | Error> {
        try {
            return await this.agent.didManagerGetByAlias({
                alias: this.mainIdentifierAlias,
                provider: 'did:ethr:rinkeby'
            });
        } catch (error) {
            console.log('it␣did␣not␣exist');
            return await this.createDID(this.mainIdentifierAlias);
        }
    }
```

We wanted to create the main identifier in the constructor, but it caused problems as it is an async function. Instead, a workaround was created where the alias is defined in the constructor, and function calls are made with that alias. Veramo has a "didManagerGetOrCreate" which would have solved this problem. Unfortunately, the team was not aware of this function at the time. As we had already implemented our solution, we decided against changing it.

Code listing 7.4: The resolve DID function

```
async resolveDID(didUrl: string): Promise<DIDResolutionResult | Error> {
                try {
                        return await this.agent.resolveDid({
                                didUrl: didUrl
                        });
                } catch (error) {
                        console.error('was␣not␣able␣to␣resolve␣did', error);
                        return new Error('was␣not␣able␣to␣resolve␣did');
                }
        }
```

Resolving DIDs are at the core of the SSI pipeline. This function has one input, the DID-URL. It is not enough to have the alias; one needs the DID-URL to access the DID-document. ResolveDID is an async function with the promise of a "DIDResolutionResult". This promise is solved when the value is returned from the async function. If it is not a proper DID resolution, it throws an error, and the DID cannot be resolved.

Code listing 7.5: The create credential function

```
async createCredential(credentialData: any): Promise<VerifiableCredential | Error>
    {
        try {
                return this.agent.createVerifiableCredential({
                        credential: credentialData,
                        proofFormat: PROOF_FORMAT_JWT
                });
        } catch (error) {
                console.error('unable␣to␣create␣the␣verifiable␣credential', error);
                return new Error('unable␣to␣create␣the␣verifiable␣credential');
        }
}
```

The small "create credential" function is responsible for creating the verifiable credentials used in the project. This framework is handled by veramo, as the API can create verifiable credentials with the correct format. In a simpler project, the API could be called directly. However, as there are multiple agents in this project, it is put into a function using "this" for selecting the correct agent when the function is called.

7.3.3 Controllers

It is possible to call the Veramo functions directly, but we made custom functions to make it easier to customize precisely how the backend logic works. To do this, we made controller classes for the Veramo agents.

Agent Controller

The AgentController class is the primary controller class that the other agents inherit core functionality. Usually, a project would only have one agent, but we have them in the same project to make it easier to update and maintain. Since they share much of the same core functionality, we used inheritance to maintain consistency. This avoids code duplication and saves time with troubleshooting and development.

What all agents require is creating a main identifier, resolving DIDs, verifying JWTs, adding services to DID-documents, and sending messages through DIDComm. This is achieved through calling the current agent through "this.agent" and using the Veramo functions on the current agent. This way, one always calls the correct agent.

One of the key functions is 'getMainIdentifier'. This function either gets a main identifier with a premade alias or creates a DID with this alias. When using DIDs to exchange information, one wants to confirm that the other DID belongs to the right entity. Having multiple DIDs creates unnecessary confusion, and the 'getMainIdentifier' makes sure there is a main identifier and that this identifier is what is used.

NAV agent controller

The only unique action the NAV agent performs is to check if a user qualifies for unemployment benefits or not, this function can be found in Appendix K. The agent still needs to have a DID, resolve DIDs, verify JWT,s and such. Nevertheless, this is handled by inheriting from the AgentController.

The NAV agent receives a Token from a user through DIDComm. The NAV agent unbundles this token, and it needs to consist of three VCs: person VC, employment VC, and termination VC. To verify this, we use two utility functions verifySchema, and verifyIssuer. VerifySchema makes sure the VC follows the correct schema. If required information is missing from the VC, this throws an error. All agents must agree to use the same schema and naming conventions for VCs. VerifyIssuer makes sure that the agents issuing the VCs is the correct source. The state agent should always issue a person VC and Symfoni should always issue termination- and employment VCs.

When the VCs are validated, the content of the VCs needs to be validated. The NAV agent sends a message to the user if their application for unemployment benefits is approved or not. This last check can be as complex or simple as one wants, as long as the information is in the VCs. This implementation checks if the employee was terminated in the last three months and if they are below the age of 67.

State agent controller

Code listing 7.6: The create person credential

```
async createPersonCredential(issuerDid: string, credentialSubjectData:
    personVerifiableCredential['credentialSubject']){
                try {
                        const credential = await this.agent.
                            createVerifiableCredential({
                                credential: {
                                        '@context': [SCHEMA_W3_CREDENTIAL,
                                            SCHEMA_PERSON_CREDENTIAL],
                                        type: [TYPE_VERIFIABLE_CREDENTIAL,
                                            TYPE_PERSON_CREDENTIAL],
                                        issuer: {
                                                id: issuerDid
                                        },
                                        credentialSubject: credentialSubjectData
                                },
                                proofFormat: PROOF_FORMAT_JWT
                        });

                        return credential;

                } catch (error) {
                        console.error('unable␣to␣create␣the␣credential', error);
                        return new Error('unable␣to␣create␣the␣credential');
                }
        }
```

The unique action the State agents performs is to create a person credential. Custom types and a schema define this credential. The custom type defines what primitive datatype the different fields are and has enums where a specific input is required. The state agent can also create business credentials, which were an early iteration not used in the final product.

Symfoni agent controller

Code listing 7.7: The create employment credential

```
async createEmploymentCredential(issuer: string, credentialData: employmentVC['
    credentialSubject']) {
        try {
                const credential = await this.agent.createVerifiableCredential({
                        credential: {
                                '@context': [SCHEMA_W3_CREDENTIAL,
                                    SCHEMA_EMPLOYMENT_CONTRACT],
                                type: [TYPE_VERIFIABLE_CREDENTIAL,
                                    TYPE_EMPLOYMENT_CREDENTIAL],
                                issuer: {
                                        id: issuer
                                },
                                credentialSubject: credentialData
                        },
                        proofFormat: PROOF_FORMAT_JWT
                });
                return credential;
        } catch (error) {
                console.error('unable␣to␣create␣the␣employment␣credential',error);
                return new Error('unable␣to␣create␣the␣employment␣credential');
        }
}
```

The unique actions the Symfoni agent performs are to create termination and employment credentials. Similarly, as the state agent, it uses custom types and schema to ensure the correct information is put into the credential. How the employee information is gathered is covered under the database section.

The Symfoni agent also uses an external database to store employment information. When a user requests credentials, the Symfoni agent has a function to see if they qualify. The agent receives a JWT with a person credential, and if the social security number matches what is in their database, the credentials are issued.

7.4 API

The API section of the code is where most of the heavy lifting is done. It consists of a dozen endpoints spread across the four main agents. Many are almost identical, though some are unique to each agent. This section will describe how the most important shared endpoints work and describe the most important unique endpoints.

We decided to create an API for a couple of reasons. First and foremost, it was to decouple our front end from the back end. However, it was also necessary as many of the packages used in our back end did not cooperate well when running react native with Expo. The Veramo module, for example, used a couple of dependencies that refused to work with Expo.

Since we already used Node.js as our runtime environment and TypeScript as our main language, we used the Express.js framework to create our API.

Code listing 7.8: Express server setup

```
import http from 'http';
import express, { Express } from 'express';
import morgan from 'morgan';
import routes from './routes/routes';

const router: Express = express();

router.use(morgan('dev'));
router.use(express.urlencoded({ extended: false }));
router.use(express.json());

// The rules of the API
router.use((req, res, next) => {
        res.header('Access-Control-Allow-Origin', '*');
        res.header(
                'Access-Control-Allow-Headers',
                'origin,␣X-Requested-With,Content-Type,Accept,␣Authorization'
        );
        if (req.method === 'OPTIONS') {
                res.header('Access-Control-Allow-Methods', 'GET,␣POST');
                return res.status(200).json({});
        }
        next();
});

router.use('/', routes);

router.use((req, res) => {
        const error = new Error('not␣found');
        return res.status(404).json({
                message: error.message
        });
});

const httpServer = http.createServer(router);
const PORT: string | number = process.env.PORT ?? 6060;
httpServer.listen(
        PORT,
        () => console.log(`The server is running on port ${PORT}`)
);
```

7.4.1 Endpoint functionality descriptions

Code listing 7.9: The get main identifier endpoint function

```
const getMainIdentifier = async (req: Request, res: Response) => {
        const mainIdentifier = await navAgentController.getMainIdentifier();
        if (mainIdentifier instanceof Error) {
                return res.status(500).json({
                        error: mainIdentifier.message
                });
        }

        return res.status(200).json({
                mainIdentifier
        });
};
```

The most used endpoint across all the agents is the endpoint that retrieves the main identifier. The main identifier is the locally stored main identifying document containing the main DID address, services, and keys. It is one of the more straightforward endpoints and is not dependent on user input. Therefore the endpoint is not wrapped in a try-catch, and if anything goes wrong, it returns a status code of 500 because it would be the Veramo API's fault. However, if it successfully retrieves the main identifier, it will return the status code of 200 and the main identifier as JSON. To retrieve the main identifier, it uses the function created in the agent controller class that either retrieves an existing identifier or creates a new one when it is called for the first time.

Code listing 7.10: The create person credential endpoint function

```
const createPersonCredential = async (req: Request, res: Response) => {
        try {
                let issuer: string = req.body.issuer;
                const validationResult = validateSchema(PERSON_VC_SCHEMA_FILE_PATH,
                      req.body);

                if (validationResult !== true) {
                        return res.status(400).json({
                                error: 'unable␣to␣create␣VC',
                                errorMessage: validationResult
                        });
                }

                const credentialSubject: personVerifiableCredential['
                     credentialSubject'] = req.body.credentialSubject;

                if (typeof issuer === 'undefined') {
                        await stateAgentController.getMainIdentifier().then((
                             identifier) => {
                                if (identifier instanceof Error) {
                                        return res.status(500).json({
                                                error: identifier.message
                                        });
                                }
                                issuer = identifier.did;
                        });
                }

                await stateAgentController.createPersonCredential(issuer,
                     credentialSubject)
                        .then((credential) => {
                                if (credential instanceof Error) {
                                        return res.status(400).json({
                                                error: credential
                                        });
                                }
                                return res.status(201).json({
                                        credential
                                });
                        });
        } catch (error) {
                return res.status(500).json({
                        error: 'something␣went␣wrong,␣try␣again␣later.'
                });
        }
};
```

Another important set of endpoints is the ones used to issue credentials. Four endpoints issue credentials in total, each with a similar endpoint structure. These endpoints use a POST method with a body with two main parts. The first part is the DID responsible for issuing the credential, and the second is the subject data within the credential. The DID that issues the credential has to be managed by the agent, and if no DID is specified in the POST body, the agent's main identifier will be used. The data subject passed through the body is validated against a schema that we made to assure that the data is correctly formatted and that all required data is present. The schema file it is being validated against exists on the projects' GitHub page and will also be part of the verifiable credential to add context for other parties. If the subject data passes through the schema validation and the agent manages the DID, a credential with said subject data will be created and signed by the DID. The agent uses its create credential function from its agent controller class to create the verifiable credential. There is an example of the code behind the endpoint that creates a person credential in the code snippet above.

Code listing 7.11: The add credential to database endpoint function

```
const addCredential = async (req: Request, res: Response) => {
        if (typeof req.body.credential === 'undefined') {
                return res.status(400).json({
                        error: 'no␣credential␣object␣found,␣make␣sure␣that␣the␣
                            verifiable␣credential␣is␣wrapped␣in␣a␣credential␣object
                            .'
                });
        }

        const credential: VerifiableCredential = {
                '@context': req.body.credential['@context'],
                type: req.body.credential.type,
                issuer: {
                        id: req.body.credential.issuer.id
                },
                credentialSubject: req.body.credential.credentialSubject,
                proof: req.body.credential.proof,
                issuanceDate: req.body.credential.issuanceDate
        };

        await userAgentController.addCredential(credential).then((credentialHash)
            => {
                if (typeof credentialHash !== 'string') {
                        return res.status(400).json({
                                error: credentialHash.message
                        });
                }

                return res.status(201).json({
                        credentialHash
                });
        });
};
```

Naturally, if there is an endpoint for issuing credentials, there should also be an endpoint for storing credentials. The user agent is the only agent with an endpoint for storing credentials as it is the only agent with a use for them. Since the credential they are adding can be of any data, no specific schema validation is involved. However, the body received from the POST request is parsed into a verifiable credential object in the code. If it does not provide an object that matches the interface of a verifiable credential, the code should fail. When the request body is parsed to the verifiable credential object, the agent's controller class uses its add credential function to add the credential to its local database.

Code listing 7.12: The create verifiable presentation endpoint function

```
const createPresentation = async (req: Request, res: Response) => {
        try {
                const credentials: VerifiableCredential[] = [];
                let holder: string = req.body.holder;
                // if holder is not specified, use default DID
                if (typeof holder === 'undefined') {
                        await userAgentController.getMainIdentifier().then((
                            mainIdentifier)=>{

                                if (mainIdentifier instanceof Error) {
                                        return res.status(500).json({
                                                fatal_error: 'unable␣to␣find␣or␣
                                                    create␣the␣main␣identifier'
                                        });
                                }

                                holder = mainIdentifier.did;
                        });
                }

                if (req.body.listOfCredentials.length === 0) {
                        return res.status(400).json({
                                error: 'empty␣list␣of␣credentials'
                        });
                }

                req.body.listOfCredentials.forEach((credential: any) => {
                        credentials.push(credential['verifiableCredential']);
                });

                await userAgentController.createPresentation(holder, credentials).
                    then((presentation) => {
                        if (presentation instanceof Error) {
                                return res.status(400).json({
                                        error: presentation.message
                                });
                        }
                        return res.status(201).json({
                                presentation
                        });
                });

        } catch (error) {
                return res.status(400).json({
                        error: 'unable␣to␣create␣presenatation,␣make␣sure␣that␣the␣
                            credentials␣are␣inside␣an␣array.'
                });
        }
};
```

When users have saved many credentials to their database, they may want to combine them into a verifiable presentation, to verify themselves somewhere or to someone. There is, of course, an endpoint for this as well, and this endpoint is user-specific as the other agents verify themselves through a verifiable registry. It is a POST request with a body that consists of two parts, the first in the holder of the presentation, which defaults to the agent's main identifier. The second part is a list of verifiable credentials that should be combined into a verifiable presentation. If the user manages many DIDs, they may also specify another DID holder by adding it to the request body. Suppose the list of verifiable credentials is longer than one, and the holder has been specified. In that case, the endpoint calls the agent controllers to create a presentation function to create a verifiable presentation.

Code listing 7.13: The user send message endpoint function

```
const sendMessage = async (req: Request, res: Response) => {
        try {
                // get params from body
                const toDid: string = req.body.toDid;
                const mainIdentifier = await userAgentController.getMainIdentifier
                     ();

                if (mainIdentifier instanceof Error) {
                        return res.status(500).json({
                                error: mainIdentifier.message
                        });
                }

                const fromDid: string = mainIdentifier.did;
                const type: string = req.body.type;
                const message: object = req.body.message;

                // use params to send message
                const messageSent = await userAgentController.sendMessage(toDid,
                     type, message, fromDid);

                if (messageSent instanceof Error) {
                        return res.status(400).json({
                                error: messageSent.message
                        });
                }

                return res.status(200).json({
                        success: 'message␣sent'
                });
        // return positive / negative feedback
        } catch (error) {
                return res.status(500).json({
                        error: 'could␣not␣send␣message'
                });
        }
};
```

A verifiable presentation is only useful if shared with someone who can verify it. Therefore, we had to implement an endpoint that the user could use to send a verifiable presentation as a message. It is an endpoint with a POST method that requires a request body with three things: to whom the message will be sent, the message type, and the message itself. The DID address that sends the message will be the main identifier DID of the agent. These four parameters are passed into the send message function of the agent controller class, which packs the message and sends it over DIDComm.

Code listing 7.14: The symfoni handle messaging endpoint functionality

```
const handleMessaging = async (req: Request, res: Response) => {
        try {
                const generatedCredentials = [];

                // handle incoming message to retrieve the token from the encrypted
                     message body
                const message = await agentSymfoni.handleMessage({
                        raw: req.body as string,
                        metaData: [{type: 'message'}],
                        save: false
                });

                const messagePresentationToken = message.data.messageData;
                const senderDid = message.from;

                const mainIdentifier = await symfoniAgentController.
                    getMainIdentifier();

                if (mainIdentifier instanceof Error) {
                        return res.status(500).json({
                                error: mainIdentifier.message
                        });
                }

                const mainDid = mainIdentifier.did;
                const isQualified = await symfoniAgentController.
                    isQualifiedForContractVCs(messagePresentationToken);

                if (!isQualified) {
                        return res.status(400).json({
                                error: 'person␣does␣not␣have␣the␣right␣credentials'
                        });
                }

                const handledMessage = await symfoniAgentController.agent.
                    handleMessage({
                        raw: messagePresentationToken
                });

                const ssn: string = handledMessage.credentials?.at(0)?.
                    credentialSubject.person.SSN;
                const ssnHash = hashString(ssn);

                const terminationContractData = await dbGetTerminationContract(
                    ssnHash);
                const employmentContractData = await dbGetEmploymentContract(
                    ssnHash);

                // if any contract fetch from db throws an error, just end the
                    transaction completly
                if (terminationContractData instanceof Error ||
                    employmentContractData instanceof Error) {
                        return res.status(400).json({
                                error: 'something␣went␣wrong␣when␣trying␣to␣fetch␣
                                    from␣the␣database.'
                        });
                }
                // only make termination vc if symfoni has the termination data
                if (typeof terminationContractData !== 'undefined') {
                        const terminationVCObject: any = {
                                id: senderDid,
                                termination: terminationContractData['termination']
                        };
                        // make credential
                        const terminationVC = await symfoniAgentController.
                            createTerminationCredential(
                                mainDid,
                                terminationVCObject
                        );
                        if (!(terminationVC instanceof Error)) {
                                generatedCredentials.push(terminationVC.proof.jwt);
                        }
                }

                // only make the employment vc if symfoni has the employment data
                if (typeof employmentContractData !== 'undefined') {
                        const employmentVCObject: any = {
                                id: senderDid,
                                employment: employmentContractData['employment']
                        };
                        // make credential
                        const employmentVC = await symfoniAgentController.
                            createEmploymentCredential(
                                mainDid,
                                employmentVCObject
                        );

                        // as long as it is not an error, push it to the
                            credentials array.
                        if (!(employmentVC instanceof Error)) {
                                generatedCredentials.push(employmentVC.proof.jwt);

                        }
                }

                // as long as the list of generated credentials are not empty, send
                      the credentials
                if (generatedCredentials.length !== 0) {
                        for (let index = 0; index < generatedCredentials.length;
                            index++) {
                                await symfoniAgentController.sendMessage(
                                        senderDid,
                                        'SymfoniCredential',
                                        generatedCredentials[index]
                                );
                        }
                }

                return res.status(400).json({ Error: 'unable␣to␣handle␣the␣message'
                      });
        } catch (error) {
                return res.status(500).json({
                        error
                });
        }
};
```

What is considered a verified presentation is up to each agent. To determine whether a presentation is verified or not, we implemented endpoints that handle incoming messages. These endpoints are not limited to messages containing verifiable presentations, though they will not respond with anything meaningful unless they get one. Only the NAV and Symfoni agent has an endpoint for handling messages, as they are the only agents that need to verify presentations. Above is a code snippet showing how the Symfoni agents handle messages.

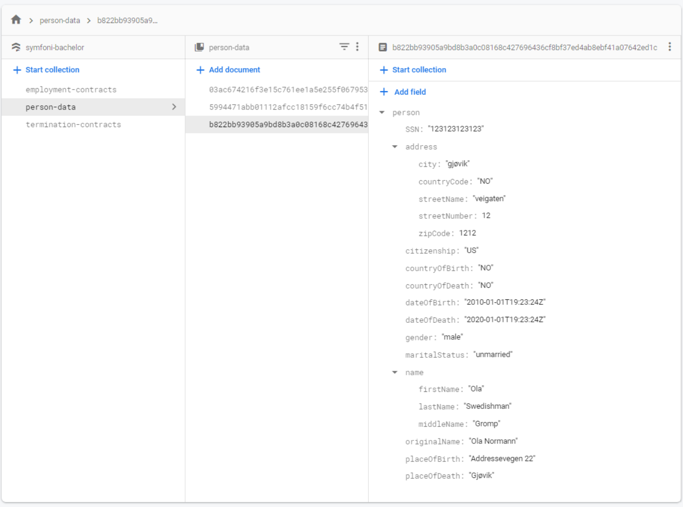
Symfoni's handle message endpoint is the most extensive endpoint we implemented, and there is most likely room for refactoring and optimization. It starts by calling the Veramo agents handle message functionality to compute a presentation token from the encrypted message. Then it runs the Symfoni agent controller class function to check if the presentation token is valid. If the token is valid, the presentation token is further handled to extract the verifiable credential tokens within the presentation. The data within the verifiable credential is then used to query the cloud database for contract data. If the person sending the presentation token has contract data in the database, contract credentials are created and sent back to the sender of the presentation.

7.5 Databases

The project utilizes a couple of databases. Firstly there are the five local databases that the Veramo agents handle. Secondly, there are the cloud databases used by our state and Symfoni agent to store information such as employment contract data, personal data, and termination contract data.

7.5.1 Firestore

For the cloud databases, we chose Firestore. Firestore is a NoSQL document-based database, which fits nicely with the purpose of our demo because it is very easy to install, maintain and use. We implemented these databases because we did not have access to real-world data like personal information that the state withholds, like a person's social security number. Therefore, we opted for a homemade personal data registry. In addition, we created a termination contract data and employment contract data database, which one would otherwise access through third-party vendors like Visma.



person-data > b822bb93905a9...

symfoni-bachelor

+ Start collection

employment-contracts
person-data
termination-contracts

person-data

+ Add document

03ac674216f3e15c761ee1a5e255f067953
5994471abb01112afcc18159f6cc74b4f51
b822bb93905a9bd8b3a0c08168c42769643

b822bb93905a9bd8b3a0c08168c427696436cf8bf37ed4ab8ebf41a07642ed1c

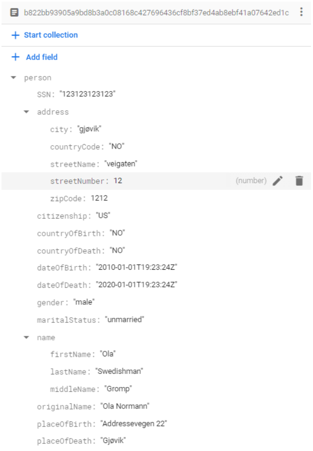
+ Start collection

+ Add field

```
person
    SSN: "123123123123"
    address
        city: "gjøvik"
        countryCode: "NO"
        streetName: "veigaten"
        streetNumber: 12
        zipCode: 1212
    citizenship: "US"
    countryOfBirth: "NO"
    countryOfDeath: "NO"
    dateOfBirth: "2010-01-01T19:23:24Z"
    dateOfDeath: "2020-01-01T19:23:24Z"
    gender: "male"
    maritalStatus: "unmarried"
    name
        firstName: "Ola"
        lastName: "Swedishman"
        middleName: "Gromp"
    originalName: "Ola Normann"
    placeOfBirth: "Addressevegen 22"
    placeOfDeath: "Gjøvik"
```

Figure 7.1: This is what the database looks like when directly interacting with it in the cloud.

To set up the database, we had to create a Firebase project. There is an option to create a Firestore database within the Firebase project. Within the Firestore database, three collections had to be created—one collection for personal data documents, one for employment contracts, and one for termination contracts. Each collection contains one document that represents one person. These documents have IDs based on a secret hash made by the secure hashing algorithm of 256 bits. These hashes are unique and can identify a specific person, and the same hash is used as an ID for the same person across all collections. In other words, a person's personal data document, and employment contract document, will have the same ID.



b822bb93905a9bd8b3a0c08168c427696436cf8bf37ed4ab8ebf41a07642ed1c

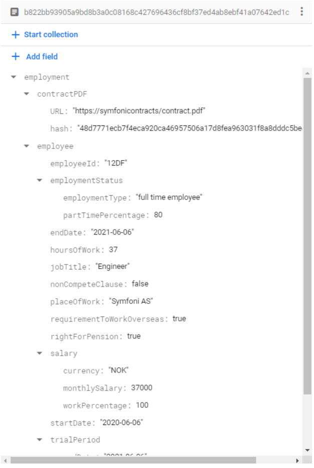
+ Start collection

+ Add field

```
person
    SSN: "123123123123"
    address
        city: "gjøvik"
        countryCode: "NO"
        streetName: "veigaten"
        streetNumber: 12                (number)
        zipCode: 1212
    citizenship: "US"
    countryOfBirth: "NO"
    countryOfDeath: "NO"
    dateOfBirth: "2010-01-01T19:23:24Z"
    dateOfDeath: "2020-01-01T19:23:24Z"
    gender: "male"
    maritalStatus: "unmarried"
    name
        firstName: "Ola"
        lastName: "Swedishman"
        middleName: "Gromp"
    originalName: "Ola Normann"
    placeOfBirth: "Addressevegen 22"
    placeOfDeath: "Gjøvik"
```

Figure 7.2: An example of a personal data document inside the Firestore database

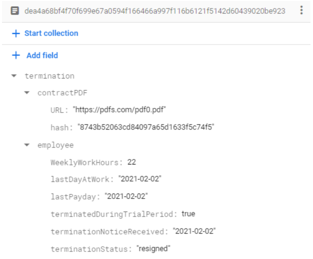
Many data points are used to describe a specific person within the personal data document. Most importantly, we have the person's social security number. The social security number is stored in plain text, which is bad practice for security reasons. It is kept in plain text because none of the social security numbers are real but mocked for the demo. The person's name and current address can also be found in the personal data document and their date of birth, gender, marital status, and much more. The personal data document is important because it is used by the state when they want to issue a verifiable credential of a person. All of the data within the personal document is also within the person's person-identifying credential.



```
b822bb93905a9bd8b3a0c08168c427696436cf8bf37ed4ab8ebf41a07642ed1c
employment
  contractPDF
    URL: "https://symfonicontracts/contract.pdf"
    hash: "48d7771ecb7f4eca920ca46957506a17d8fea963031f8a8dddc5be
  employee
    employeeId: "12DF"
    employmentStatus
      employmentType: "full time employee"
      partTimePercentage: 80
    endDate: "2021-06-06"
    hoursOfWork: 37
    jobTitle: "Engineer"
    nonCompeteClause: false
    placeOfWork: "Symfoni AS"
    requirementToWorkOverseas: true
    rightForPension: true
    salary
      currency: "NOK"
      monthlySalary: 37000
      workPercentage: 100
    startDate: "2020-06-06"
    trialPeriod
```

Figure 7.3: An example of an employment contract data document inside the Firestore database

The employment contract documents are used when Symfoni wants to issue an employment contract credential, and therefore it consists of all the data required to represent an employment contract. This data was thoroughly discussed with NAV, as mentioned earlier. Examples of such data are the start and end date, the expected salary, and the expected work amount.



```
dea4a68bf4f70f699e67a0594f166466a997f116b6121f5142d60439020be923
termination
  contractPDF
    URL: "https://pdfs.com/pdf0.pdf"
    hash: "8743b52063cd84097a65d1633f5c74f5"
  employee
    WeeklyWorkHours: 22
    lastDayAtWork: "2021-02-02"
    lastPayday: "2021-02-02"
    terminatedDuringTrialPeriod: true
    terminationNoticeReceived: "2021-02-02"
    terminationStatus: "resigned"
```

Figure 7.4: An example of a termination contract data document inside the Firestore database

The termination contract documents are much like the employment contract documents, although these are used when Symfoni wants to issue a termination credential. The data within the termination contract documents were also discussed with NAV. Examples of such data are how many hours the person worked on average, their last day at work, their last payday, and why they were terminated.

If these collections were not added to the project, the different agents would have to make and store the verifiable credentials simultaneously as their data was generated. The agents can store data without generating a verifiable credential by having these collections, and they only have to generate a credential whenever a person is asking for it.

Firestore code examples

Setting up the Firestore database in the code is simple. The first thing we had to do was to go to our Firebase project and download the service account key file. The service account key file is a way of verifying that one has the right credentials to be using the database. Secondly, we had to install the Firebase Firestore modules from the node package manager. Finally, we had to code the setup.

Code listing 7.15: Firestore database setup

```
import { initializeApp, cert } from 'firebase-admin/app';
import { getFirestore } from 'firebase-admin/firestore';

const PATH_TO_SERVICE_ACCOUNT_KEY = 'serviceAccountKey.json';

initializeApp({
        credential: cert(PATH_TO_SERVICE_ACCOUNT_KEY)
});

export const db = getFirestore();
```

The CRUD operations of the Firestore database are easy to write out in code. There is no relational mapping involved, and all files are stored as documents; all one has to know is which collection to search in and which document ID one wants to do operations on.

Code listing 7.16: Example of a read operation

```
export async function dbGetPersonData(id) {
        try {
                const docRef = db.collection(PERSON_DATA_COLLECTION).doc(id);
                const doc = await docRef.get();
                if (!doc.exists) {
                        return new Error('no␣document␣found');
                } else {
                        return doc.data();
                }
        } catch (error) {
                return new Error('unable␣to␣retrieve␣document␣from␣database');
        }
}
```

Above there is an example of how a read operation is carried out. For simplicity, we have removed the type declarations, as they are not important for the flow of the code. The first thing to notice is that the entire function is wrapped in a try-catch; this is in case there is some server-side error that results in an unexpected error. The second thing to notice is how we access the document we desire in just one line of code. Firstly, we specify the collection we want to search through and retrieve the document with whatever document ID we want. If the document does not exist, we return an error stating that we could not find any document. On the other hand, if it does exist, we return the data within the document.

Code listing 7.17: Example of a delete operation

```
export async function dbDeletePersonData(id:string) {
        try {
                const docRef = db.collection(PERSON_DATA_COLLECTION).doc(id);
                docRef.delete();
                return (await docRef.get()).data();
        } catch (error) {
                return new Error('unable␣to␣retrieve␣document␣from␣database');
        }
}
```

Deleting a document from the database is equally as simple as reading from the database. Like the read operation, it has a try-catch in case anything goes wrong on the server-side. Finding the document to delete is done the same way as when reading. If a document was found and deleted, the function returns the data of the deleted document. The idea behind returning the deleted documents data is to verify that a document was found and act as a security net if the wrong document was deleted.

Code listing 7.18: Example of a create and update operation

```
export async function dbAddPersonData(id, personData){
        try {
                const docRef = db.collection(PERSON_DATA_COLLECTION).doc(id);
                await docRef.set(personData);
        } catch (error) {
                return new Error('database␣insertion␣failed');
        }
}
```

Creating a new document and updating an old one are done the same way. Firstly, the function finds the document reference of the document it should create or update, then it sends some JSON data to that document reference. If the document did not exist beforehand, the function simply creates a new document. However, if the document did exist, the function will overwrite that document, thereby updating its data. The function is, of course, also wrapped in a try-catch.

7.5.2 Local

The local databases are handled by the Veramo API and are used for storing everything related to verifiable data and SSI. Each agent has its separate local database, although each is used a little differently. Each database is locked with its respective secret key, meaning that only the owner of the local database has access to it.

```
export const dbConnectionTest = createConnection({
        type: 'sqlite',
        database: TEST_DATABASE_FILE,
        synchronize: false,
        migrations,
        migrationsRun: true,
        logging: ['error', 'info', 'warn'],
        entities: Entities,
});
```

The user agent is the one that uses its local database the most. The user's identifiers, verifiable credentials, keys, and messages are stored within this database, and it is heavily accessed throughout the use of the application. For example, when retrieving the user's primary identifier, listing the stored VCs, reading the VC details, listing messages, and viewing message details.

Although the other agents also have use cases for their local database, it is minimal compared to the user. The other agents mainly use their local database to access their main identifier private key when signing credentials for issuing. This means that the security of this database is paramount, as anyone with access to it can impersonate that agent.

The databases use SQL for accessing, meaning that they are traversed with SQL queries. Fortunately, for simplicity, the Veramo framework comes preconfigured with an API that generates these queries. However, this can generate some difficult-to-understand error messages, which one can not directly fix within the queries themselves without speaking to the Veramo developers.

7.6 React Native

The project's different agents must be deployable on different platforms, as the different entities the agents represent have different workflows. It would be natural for the user agent to be used on a phone, while the Symfoni agent would be more commonly used on a computer. This requires our application to both run on the web and mobile. The best JavaScript framework for this purpose is React Native, as it is made for developing for mobile devices.

7.6.1 Navigation

Code listing 7.19: Navigation screen

```
export default function UserHomeView({navigation}: any): JSX.Element {
        return (
                <View style={styles.container}>
                        <Text style={styles.headingTextBlack}>Welcome User!</Text>

                        <NavigationButton
```

```
                                title='Show␣DID'
                                customStyle={buttonStyles.navigationButtonUser}
                                onPress={()=> navigation.navigate('
                                    UserMainIdentifier')}
                        ></NavigationButton>
                </View>
        );
}
```

React Native has built-in navigation functionality that keeps track of what screen is currently displayed and the previous screen. The entire app is wrapped in a navigation container, which is done in the entry file "app.tsx". Every screen in the app is put on the navigation stack. Each page has a navigation bar at the top. It states which page is displayed and has a back button leading to the previous page.

Code listing 7.20: pasing params to a different screen

```
export default function UserMessageDetailView({navigation, route}: any): JSX.
    Element {
        const {item} = route.params;
```

Using the stack navigator makes it easy to pass parameters between screens. An example of this is in the "UserMessageDetailView," which displays data from another screen using "route.params".

7.6.2 Components

When programming the application, we tried to split as much functionality into components as possible. Splitting into components makes it easier to understand, as different functionalities are isolated and imported where used. Using components also makes it possible to reuse code, as components are reusable and can be used in multiple places in the application.

Code listing 7.21: Credential card component

```
export function CredentialCard({item, navigation}: {item: VerifiableCredential,
    navigation: any}) {
else if (item.verifiableCredential.type[1] === TYPE_TERMINATION_CREDENTIAL) {
                return (
                        <TouchableOpacity style={styles.credentialCard}
                                onPress={()=> navigation.navigate('UserVCDetail', {
                                        item: item
                                })}
                        >
                                <Text style={styles.headingText}>{item.
                                    verifiableCredential.type[1]}</Text>
                                <Text style={styles.defaultText}>Last day at work:
                                    {item.verifiableCredential.credentialSubject.
                                    termination.employee.lastDayAtWork}</Text>
                                <Text style={styles.defaultText}>Last payday: {item
                                    .verifiableCredential.credentialSubject.
                                    termination.employee.lastPayday}</Text>
                                <Text style={styles.defaultText}>Termination status
                                    : {item.verifiableCredential.credentialSubject.
                                    termination.employee.terminationStatus}</Text>
                        </TouchableOpacity>
                );
```

Code listing 7.22: Credential detail component

```
TYPE_TERMINATION_CREDENTIAL) {
        return (
                <ScrollView>
                        <View style={styles.credentialDetailView}>
                                <Text style={styles.headingTextBlack}>{item.
                                    verifiableCredential.type[1]}</Text>

                                <Text style={styles.credentialDetailHeadingText}>
                                    Employee info</Text>
                                <Text>Last day at work: {item.verifiableCredential.
                                    credentialSubject.termination.employee.
                                    lastDayAtWork}</Text>
                                <Text>Last payday: {item.verifiableCredential.
                                    credentialSubject.termination.employee.
                                    lastPayDay}</Text>
                                <Text>Termination status: {item.
                                    verifiableCredential.credentialSubject.
                                    termination.employee.terminationStatus}</Text>
                                <Text>Terminated during trial period: {item.
                                    verifiableCredential.credentialSubject.
                                    termination.employee.
                                    terminatedDuringTrialPeriod.toString()}</Text>
                                <Text>Weekly work hours: {item.verifiableCredential
                                    .credentialSubject.termination.employee.
                                    WeeklyWorkHours}</Text>

                        </View>
                </ScrollView>
        );
```

Two major components are the "credentialCard" and "credentialDetail." One of the issues here is that there is no easy way to automate the process of displaying information from a verifiable credential. Every item in this credential has to be manually accessed by going into the credential subject and getting the correct field. This project only uses four different credentials, so while there is much manual labor, it is manageable. This becomes a bigger issue in a more extensive system with potentially hundreds of credentials.

7.6.3 Styles

Code listing 7.23: Example of button styles

```
export const buttonStyles = StyleSheet.create({
        submitButtonFormSymfoni: {
                margin: 10,
                backgroundColor: symfoniColor,
                borderRadius: 6,
                alignItems: 'center',
                padding: 10
        }...
}
```

Using the same principles of reusability and consistency as with components, a styles file defines all the different styles used in the app. By importing the predefined styles from this file, we ensure that everything has a consistent design. When different developers work on different parts of the app, they use the same style format to avoid design differences. All buttons for navigation use the navigation button styles, all forms use the form style, and so on. Every agent has a unique color scheme to make them more distinct and recognizable.

7.6.4 Forms

When developing the project, the development team tried to avoid using third-party libraries as much as possible. An exception to this was for creating the forms used in the Symfoni agent, as creating forms in react native is a challenge. We used Formik with yup validation to create the employment and termination forms.

Code listing 7.24: Yup validation

```
const EmploymentSchema = yup.object({
        socialSecurityNumber: yup.string().required(),
        jobTitle: yup.string().required('Required␣Field'),
        hoursOfWork: yup.number().required('Required␣Field').typeError('Invalid,␣
            must␣be␣a␣number'),
        startDate: yup.date().transform(parseDateString).required('Required␣Field')
            .typeError('Valid␣date␣required␣YYYY-MM-DD')
```

Yup provides form validation based on multiple factors, with built-in functionality for error messages. For this project, we used required fields and type guards. TypseScript already uses type guards, so using it twice has no purpose other than Yup requires it. Yup does have a "Date" type not included in TypeScript, which helps enforce the correct date format. The ability to have optional fields is good, as our credentials also have optional fields.

An alternative to using Formik would be using React Hook Form. This would have been a better choice as it is a newer tool utilizing react hooks. The difference is that React Hook Form isolates the input fields, preventing the form from rerendering from a single field change.

7.6.5 Function components

Code listing 7.25: UseState

```
export default function SymfoniRecieveVPView(): JSX.Element {
        const [isLoading, setLoading] = useState<boolean>(true);
        const [mainIdentifier, setMainIdentifier] = useState<IMainIdentifier>();
```

There are two ways to write React Native applications with components: functional and class components. We are using functional components for this project as they have a simpler syntax and generally less boilerplate. A functional component is just a plain TypeScript function that accepts props as an argument and returns a React Native element. React hooks makes it possible to add a state to a function component without converting it to a class. In the code snippet above, "SymfoniRecieveVPView" uses the hook useState to set the main identifier for this page.

Code listing 7.26: UseEffect

```
useEffect(() => {
                getMainIdentifier();
        }, []);
```

The Effect hook allows for performing side effects in a function component. When using an Effect hook with an empty array, we make sure the effect runs only on the first render. We only want to set the main identifier once, not every time the component renders.

Code listing 7.27: Get main identifier

```
const getMainIdentifier = async () => {
        try {
            const response = await fetch(SYMFONI_GET_MAIN_IDENTIFIER_URL);
            if (!response.ok) {
                alert('unable␣to␣generate␣QR-code␣with␣the␣main␣DID␣address');
                return;
            } else {
                const json = await response.json();
                setMainIdentifier(json.mainIdentifier.did);
            }
        } catch (error) {
            console.error(error);
            alert('something␣went␣wrong');
        } finally {
            setLoading(false);
        }
    };
```

The API is the bridge between the back-end and the front-end. React Native uses the fetchAPI, which works very similarly to "XMLHttpRequest". When using these functions, we cannot be sure when the fetch is completed, so we use async-await to ensure it is completed before continuing to the next step. Fetch takes two inputs, a URL and the optional argument request options, such as type of request. When receiving a response, it also receives a status code. We decide what to do with the data based on this status code. In the example, "getMainIdentifier" fetches a get request from a URL. This URL is a pre-defined constant to make it easier to always use the right URL for the right agent. Since this request is a GET request, there is no need to specify request options, as get requests are the default option.

Chapter 8

Testing and quality assurance

Having thorough testing and good quality assurance has been a priority for us since the beginning of the project. We were early in setting up a testing framework and adding git hooks to our project, ensuring that all tests passed and the project ran properly before any commits and pushes went through. The usage of branches in our repository was also important to us, and we had strict rules for them. We did not allow pushing changes to the main 'dev' branch directly. We always made pull requests from other branches, where those required a code review from another member to see if the code held a certain standard before we were allowed to merge. We did this to ensure that no harmful code or possibly project-breaking code would enter our 'dev' branch. This also led to us not spending a lot of time refactoring old code that would not suffice later on.

8.1 Unit testing

We use unit testing to verify that our functions follow our design and give expected results for different cases. For our unit testing, we have used the JavaScript framework Mocha. We chose Mocha because it runs on Node.js and supports testing for asynchronous functions. Not every function has a corresponding test, but the more important and frequently used functions do. For instance, we have written tests for our agent controller functions and set up a test database to confirm that the functions work and that the agent controller is able to meet our expectations of functionality. These are functions like creating DIDs, resolving DIDs, listing DIDs based on different parameters, creating credentials, verifying JWTs, creating presentations, and doing database operations on these, such as storing credentials in a database. We also have unit tests for our utility functions, such as our schema validation function. We have tests that ensure that good objects that follow the schema pass and tests that confirm that bad objects not following the schema will fail.

Code listing 8.1: Example of mocha before and after functions for tests

```
const testAgentController = new AgentController(agentTest, 'test');
const testJWT = 'jwt...';
let testDidUrl: string;
let testCredentialData: AnyObject;



before(async function () {
        // Makes and saves a did for the test cases.
        await testAgentController.createDID('test0').then((did)=>{
                if (did instanceof Error) {
                        return;
                }
                testDidUrl = did.did;
        });

        // Creates a couple more dids for testing.
        await testAgentController.createDID('test1', 'did:web');
        await testAgentController.createDID('test2');

        // Setting up a sample credential data for testing.
        testCredentialData = {
                type: ['VerifiableCredential', 'TestCredential'],
                issuer: {
                        id: testDidUrl
                },
                '@context': [SCHEMA_W3_CREDENTIAL],
                credentialSubject: {
                        'name': 'Ola',
                        'age': 25
                }
        };

        // Override console.log and console.error with empty functions to supress
             function logging when testing
        // making it easier to read the test result.
        // eslint-disable-next-line @typescript-eslint/no-empty-function
        console.log = function () {};
        // eslint-disable-next-line @typescript-eslint/no-empty-function
        console.error = function () {};

});

after(async function () {
        // Resets the test database after each run.
        await dbConnectionTest.then((testDb)=>{
                testDb.dropDatabase();
        });
});
```

This is an example of the mocha hooks, 'before' and 'after.' The 'before' function gets called before the test case is run; it initializes the databases with DIDs and credentials. The 'after' function gets called after the test case is finished and clears the database. These hooks ensure that the tests are persistent and run the same each time.

8.2 API testing

The API is a central part of our application and has many endpoints. Therefore it is important to make sure that they meet our expectations of how they function and their uses. Like unit testing, API testing works by sending a request to the endpoints, and the test passes if the correct status code or response body is returned from the endpoints. These tests can either be automatically completed through code or done manually through the use of API tools. For our project, we have chosen not to write automated API tests because the simplicity and familiarity of the API tools ’Postman’ and ’Insomnia’ have saved us time, and their functionality suffice our scope. We created a document with predefined reusable requests to all the endpoints in Postman. This made it easy to test our API and see that we got the status codes and responses we wanted.

8.3 GUI testing

During development, the GUI was tested manually as functionality was added. Every developer made sure it was deployed on both web and android before merging their changes into the development branch. We had strict procedures for developing new parts of the system, like code review, but they were manual, not automated. This manual testing is not to replace written GUI testing and is a weakness in our project. The lack of GUI testing is due to technological difficulties and time constraints. While this did not impact our project in a significant way, it is unsustainable in the long run. More developers and a more extensive codebase would inevitably lead to mistakes when GUI testing is not present.

In early development, we chose Mocha as our testing framework. This was based on research for the best testing frameworks for TypeScript, and the two options were Mocha and Jest. Based on the initial reaction, Mocha had better documentation, which is why Mocha was used. In retrospect Jest would have been a better option. The main reason is that Jest supports snapshot testing. This is when the testing framework renders a UI component and takes a snapshot. The UI can be compared to this previous snapshot in later testing to see any difference. This only fails when there are changes in the application, so if an update is made, the test needs to be rerun to update the snapshot. This type of testing requires some vigilance by the developer to not simply update the snapshot without checking if the intended results were achieved.

There are some solutions to this, using third-party libraries for Mocha snapshot testing, adding an additional framework to the project, and migrating from Mocha to Jest. We try to avoid using third-party libraries as much as possible; even if it works, it creates new problems with compatibility and updates in the future. Migrating to Jest was also possible, but it is unknown what consequences this would have for the existing unit tests, as there are some differences in syntax between the two. Adding Jest alongside Mocha proved to be challenging as we could not get Jest to work with TypeScript. In the end, we simply ran out of time.

With more time, we likely would make it work. While it does lack some polish, the application is operational now, and we did not want to make major changes this late in the process. We decided it was more important to focus on the end product than developing the desired GUI tests.

Snapshot testing is not the only way of GUI testing, as React Native also supports unit-testing, component-testing, and integration-testing. Unit testing tests small code snippets, like individual functions or classes. Integration testing is when multiple units are combined and are working together. Component testing is for testing individual components. Integration testing is a challenge because we use the Ethereum test network for our did:method, with its limited ability to provide Ethereum to run the application. This process has not been automated satisfactorily and is complicated to do manually. The tests are hard to accomplish because of the manual labor necessary to add the currency for transaction fees when updating the DID document for the required DIDs. Component testing is something that we should implement but was cut due to time constraints. [15]

Chapter 9

Conclusion

9.1 Discussion

9.1.1 SSI

The early parts of the bachelor's were heavily theoretical. SSI is cutting-edge technology and a subject matter unfamiliar to the group. Much time was dedicated to discussing how SSI works, how a user would interact with the system, the more considerable implications for public services, and how it would be implemented in the project.

One point of debate was if SSI would replace centrally controlled registries or supplement them. In the scenario where a person's credential replaces a passport, would there still be a database containing their personal information, or does this go against the concept of decentralization? After consulting with our employer, the group shifted focus towards creating a digital credential to replace physical credentials, not replacing all centralized registries. Removing all forms of registries and databases for public services would not be possible, as a central authority is required. A digital credential gives the user greater control of what information they send and the convenience of having the information digitally.

While the technology has much potential, a proof of concept with no basis, in reality, is of limited use. To increase the legitimacy and practical use of the thesis, the group has been in collaboration with NAV to get input on how a government service functions and its internal routines. This collaboration was made possible through Jon Ramvi, the CEO of Symfoni, as he organized the meetings between the different parties.

9.1.2 Collaboration with NAV

One of the benefits of working with NAV was explaining the project to someone unfamiliar with SSI. A considerable amount of time was spent creating different swim-lane diagrams to demonstrate how a user would interact with the system. Overall, this gave us a greater understanding, as we had to plan out and discuss every step a user would perform. More business-oriented diagrams were explained on a surface level, and more technical diagrams explained how the different agents interacted.

Another important factor for verifiable credentials is the schema. The group invented a schema based on what we thought would be useful to include in an employment contract. A group of students with limited working life experience is not ideally suited for this task. This first iteration was updated with input from NAV about what information they were interested in. One aspect was splitting the employment credential into two credentials, employment, and termination. Not all information relevant to an unemployment benefits application fits inside an employment credential.

The schema structure is created for the specific use case of unemployment benefits. An issue will occur if these schemas become too long and unmanageable. The idea is that an employer fills in the information in their accounting system, and if there are too many input fields, employers will be hesitant to use the system. A benefit with verifiable presentations is that you can have many smaller credentials for specific scenarios and put them together in a verifiable presentation. This way, there is no need for one large credential, but have several small ones.

9.1.3 Redundant code

Part of the codebase was made early in the development cycle and became obsolete when design changes were made for the project. In a real-life scenario, code that is not used should be removed. However, as this is a bachelor thesis, we decided to keep it in the project to show the thought process and work during development. Another reason for keeping everything in the project is that it can be used for further development.

An example of this is the business credential. Initially, we wanted a business to exchange a credential the same way a person does when establishing a DIDComm connection. After consultation with our employer, the flow of both parties presenting credentials was changed. A more static approach to verifying a business' identity was established, with a registry for businesses maintained by the government. While we do not use a business credential in this project iteration, it does not mean a business credential will never be used. Therefore it was kept in the project.

In the same way, as business credentials were not used in the final product, some of the API endpoints fall into the same category. Some endpoints were created and used during manual testing but did not serve the final product's purpose. In our project, a user does not create credentials, but that does not mean a user never will create a credential. One possibility we discussed with NAV was the user giving a power of attorney credential to their lawyer. It seemed wasteful to delete work that could be a foundation to build upon just for a cleaner repository. Neither the business credential nor the extra endpoints take much space, and they naturally fit into the project structure.

9.1.4 Schema

Early in the development process, the group spent a considerable amount of time on Schema for different VCs. This was to establish the baseline of verifiable information to build the project upon. Some of the work done was outside the project's scope, as they relate to large structural decisions on a government level. When creating the person credential, the group tried to find pre-existing similar data structures. The schema was based on the Norwegian Data Catalog, which has entries for defining a person. In the end, the structure of the person's credential did not matter a great deal, as the application does not use most of the information it contains. Receiving a person's credential is sufficient to verify an identity. This does not mean the work regarding schema was wasted, as it was useful for learning more about the concepts of SSI.

The struggle with deciding data for the Schema showcased the challenges with adopting SSI for public services. In this application, several layers of schema validation are utilized. This means only credentials using the correct Schema will be validated. To work with public services, every public service and business related to those services must agree to use the same standards for a credential schema. That is just on a national level; when collaborating with NAV, the idea was presented for using the EU Digital Wallet, where every member state of the European Union must use the same standards. This is a challenge, and the group debated using both data from the Norwegian Data Catalog [13] and the EBSI[16].

During this process, our employer recommended using JSON-LD. It was impossible to use the existing schema directly, but a new schema that referenced the existing schema could be possible. This way, the schema in the project would have a greater sense of legitimacy, as they are backed by real data, not just invented for this project. However, after consulting with a semantic web expert, it was revealed that this was not possible. The final schema for the application was created based on the data from "Data Norge" [13] and can be used outside our limited scope of unemployment benefits application.

9.1.5 Limited amount of issuers

Some changes were made to the original task description as the project evolved. The application is not deployed to multiple servers, and there are fewer credential issuers than initially intended. The multiple servers were reduced to fit the scope of the thesis better. Instead of having multiple credential issuers, the group utilized the collaboration with NAV to focus on the use case of applying for unemployment benefits. The main problem scenario was exchanging employment information. This upheld the requirement of using SSI but kept the scope more manageable for a bachelor's thesis.

Originally, the demo was supposed to showcase the SSI flow between up to eight actors. However, we only managed to demo the flow between four actors due to time constraints. This is a direct consequence of not building on the Symfoni baseline, and this required more coding work, and the number of issuers was reduced. Adding more actors would not further prove anything, but it would make the demo seem more realistic. Because of the project's structure, it is easy to create another actor through the controller classes, but it will be time-consuming as new endpoints, schema, and verification functions need to be made.

Symfoni proposed that we could use their already built proof of concept app and further build on it to make our demo. However, we thought that would take away a lot of our learning experience and make the bachelor thesis slim. Symfoni agreed with this, and we began building the project from scratch. If we were to use the pre-built app, we would most likely, have gotten a lot further in the demo, making a more realistic case.

9.1.6 Type guards in TypeScript

One of the core concepts of TypeScript is using types for protection from runtime errors and giving the developer assistance when writing code and general readability. The developers have held a high standard of utilizing types throughout the project. This has been a great assistance when programming, as types, gives a better idea of the content of some data. An example would be a function that requires a verifiable credential type; there is no mistaking what kind of input it needs.

This pattern was broken when developing the front end using React Native. When using React Native, the developers ran into issues with types. The developer simply used the "any" type when running into these issues. Using the "any" type defeats the point of having types, but it makes the code executable. The drawback is making it more difficult to understand the data flow between components. This becomes clear in react-navigation, where the development team failed to add types.

9.1.7 Alternatives

The most significant decisions made in the thesis were choosing to develop the application from scratch, the transport layer for the application, and the use case of unemployment benefits. These decisions have been discussed earlier, but this is a retrospective of what we have learned.

Developing our own application and not using the Symfoni baseline was the right decision. While it hurts the demo, as it is less complete than we wanted, the additional learning from developing an android application outweighs the negatives. It gave more substance to the thesis and helped the group reach their learning goals.

The choice was between using wallet connect or DIDComm. We chose DIDComm, as veramo has built-in DIDComm functionality, and we use veramo for the base of the digital agents in the project. This avoids adding more dependencies to the project and complicating it further. Wallet connect is more catered toward decentralized apps, which does not apply to our application. DIDComm was easier to use and fits better with our application.

In hindsight, the use case of applying for unemployment benefits was not the best choice. For someone unfamiliar with the technology, the benefits of switching to a SSI system are not clear. It would appear to gather information similarly to today's systems to an outside observer. Using verifiable credentials loses some usefulness when it is not integrated into society, but the user needs to gather all the credentials manually. This is not clear in the demo, but it can be reused once a credential is obtained.

9.2 Critique

The bachelor thesis is very conceptual, without a clear end goal. When we got our assignment, it was more a description of Symfoni and what kind of technology they work with than a well-defined task. After collaborating with Symfoni and NAV, we specialized in creating Verifiable Credentials for unemployment benefits applications. Before this, our only guideline was using the SSI concept to send and verify Verifiable Credentials. This did give the group a degree of freedom in choosing the direction of the thesis. However, it was not easy to comprehend the end goal and how to get there early on.

We are already working in a new field with limited documentation and examples. Using TypeScript alongside this also caused some problems. TypeScript has many benefits, and type guarding has been a great help when developing the application. However, TypeScript has compatibility issues with many programs and libraries. When programming in React Native, we frequently troubleshot a problem, only to find out it is a TypeScript issue. If we were to freely chose a programming language, we would probably still use TypeScript, but it has caused some issues.

The communication between the group and our university supervisor was less frequent than ideal. This was due to time constraints on the supervisor's end, as he has other responsibilities. This should be coordinated better with the university, as they are in charge of delegating responsibilities to the supervisors. We have been in contact with other bachelor groups that have been. This has not had the most prominent impact on the project, as it mostly filled the role of providing guidance. Towards the later stage of the bachelor's, it became more of an issue, as that is more university-related with writing the thesis itself.

9.3 Further work

This bachelor project had the scope of delivering a working demo that showcases the SSI ecosystem with a practical case. This baseline can be further developed and built upon.

An application would typically only contain one digital agent. Having four agents in the same project was to effortlessly update and debug all agents at once. For further work, all the digital agents need to be decoupled from the main repository and the necessary functionality implemented for each agent. The API handles the interaction between the user and the back-end. All the decoupled applications would require a new API.

This project has also been focused on one use case, automating the application of unemployment benefits for NAV. Specifically creating credentials containing employment information. This is an incomplete flow for unemployment benefits, as multiple factors are considered. The system needs to be extended to include military service, education, actively searching for employment, and other criteria. Credentials for these proofs can be created by using the same methodology as the employment credentials.

This scenario is only one possible use for verifiable credentials and the SSI infrastructure. Another use case could be transferring power of attorney to a lawyer or having a digital driver's license. The core concepts will be the same, for example, by having a secure DID-to-DID connection and sending credentials between digital agents. There are possibilities to explore by granting access of credentials to trusted parties or having autonomous sensors issuing credentials in a production line. SSI is a new technology, and this bachelor's thesis is only the tip of the iceberg.

9.4 Evaluation of group efforts

This section describes how the group worked together to solve the assignment and how the work was organized.

9.4.1 Organizing

A variety of factors disrupted the planned work schedule. During the first five sprints, this schedule with physical meetings was upheld strictly but was disrupted by additional schoolwork outside the bachelor's. Another mandatory subject was held simultaneously, requiring group work with other students and watching lectures. This made it difficult to have regular physical meetings, as a conference room is not well suited for watching lectures. After this subject was over, reserving rooms on campus was difficult as there was high demand.

Because of this, and that the group had gotten used to online collaboration, the majority of the remaining project was conducted online. There are both pros and cons to online work. The pros being working in an environment one is comfortable in and having access to all the amenities of a home office. It is also easy to share screens and follow along when another developer demonstrates code or pair programming.

9.4.2 Workload distribution

The workload distribution became a bit uneven. Most of the coding was done by one group member, for which there could be a couple of reasons. It could be the fault of a lack of good issue tracking. Although issues were made at the beginning of the project, they were never delegated to a specific person, meaning anyone could attack a given issue. As the project progressed, fewer issues were made, making the meeting notes the primary task source.

It might also have been a combination of not having a group leader and group members lacking personal motivation. The group did not have a leader that intervened when group members did not carry their load, and neither to motivate the group members to do their job. Lack of leadership meant that each member worked from personal motivation, which should not be taken for granted.

When we began documenting our project, the workload distribution evened out. All members contributed equally to proofreading, documenting, and making models. However, we did not change the group organization. Each member found something they wanted to do independently, notifying the rest of the group when they found something.

In retrospect, the group should have pointed out a group leader. The group leader should have held meetings each week to ensure everyone had a task to do and that the group members were making progress. These meetings should also have been used to help each other complete our tasks if we were stuck. We should also have delegated issues to the group members, as we found that assigning them to oneself made it easy to avoid issues and not finish them.

9.4.3 Project as a form of work

Working on a project as a team for the bachelor thesis has proven to be challenging yet rewarding. As a group, we had to learn about new technologies and how to work together to achieve a common goal. Being more than one student in a project and discussing the subjects helped tremendously during the learning process. Moreover, working together meant that we could achieve more than if we were working alone.

9.5 Conclusion

The goal of the bachelor's thesis was to create a demo with a proof of concept for SSI workflow for public services. This goal has been achieved as the group has developed an application for demonstrating a use case utilizing SSI. While the application lacks some polish and some aspects need further development, it is a working demo. As laid out in the problem description, the application has several digital agents with the capability to both issue and verify verifiable credentials. DIDComm handles the transport layer to ensure secure encrypted communication.

The application meets the requirements defined by Symfoni, but it is only a proof of concept. It cannot be used for any practical purpose outside of demonstrating an SSI workflow. For this application to be deployed on an app store, major changes would need to be made. This was the intention, but the final product is far removed from a real-life scenario. This makes the thesis a tool for showcasing SSI, or a baseline for other developers.

While the final product is only a proof of concept, the technology behind the concepts is how a real-life implementation would be created. Every digital agent has a unique DID that is stored on the Ethereum blockchain. These agents are entirely independent and can communicate through DID-to-DID connections. Through this communication channel, they exchange verifiable credentials with relevant data.

The data is split into three verifiable credentials developed in collaboration with NAV. A person credential for a digital identity, an employment credential with the data from an employment contract, and a termination credential containing the details when an employee exits a job. The group went to great lengths to have the verifiable credentials contain information relevant to the use case.

By combining these elements, the application creates cryptographically secure communication channels where the user has control of the exchanged data. The information uses the data structure of verifiable data, and the different entities use decentralized identifiers stored on the Ethereum blockchain.

Bibliography

[1] E. commission. ‘European digital identity.’ (), [Online]. Available: https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_en (visited on 16/05/2022).

[2] A. Preukschat, *Self Sovereign Identity: Decentralized digital identity and verifiable credentials*. Manning Publications, 2021.

[3] D. Reed, S. Manu, S. Markus, L. Dave and C. Allen. ‘Decentralized identifiers (dids) v1.0.’ (), [Online]. Available: https://www.w3.org/TR/did-core (visited on 19/05/2022).

[4] C. Gribneau, M. Prorock, O. Steele, O. Terbu, M. Xu and D. Zagidulin. ‘Did:web method specification.’ (), [Online]. Available: https://w3c-ccg.github.io/did-method-web/ (visited on 19/05/2022).

[5] Ethereum. ‘Introuction to smart contracts.’ (), [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/ (visited on 19/05/2022).

[6] M. Sporny, D. Longley and D. ChadWick, *Credential graph*, [Online; accessed 2022], 2022. [Online]. Available: https://www.w3.org/TR/vc-data-model/diagrams/credential-graph.svg.

[7] ‘Verifiable credentials data model v1.1.’ (), [Online]. Available: https://www.w3.org/TR/vc-data-model/ (visited on 19/05/2022).

[8] S. Curren, T. Looker and O. Terbu. ‘Didcomm messaging.’ [chapter 2, 5, 8]. (), [Online]. Available: https://identity.foundation/didcomm-messaging/spec/ (visited on 19/05/2022).

[9] IBM. ‘Trust anchors.’ (), [Online]. Available: https://www.ibm.com/docs/en/transparent-supply?topic=roles-trust-anchors.

[10] Veramo. ‘Pligins architecture.’ (), [Online]. Available: https://veramo.io/docs/veramo_agent/plugins (visited on 19/05/2022).

[11] Veramo, *Veramo plugins architecture*, [Online; accessed 2022], 2022. [Online]. Available: https://veramo.io/assets/images/veramo_plugins_simple-e61eae0077b0d914a305cb9750a6d300.svg.

[12] I. Liljeqvist, *Infura ethereum api*, [Online; accessed 2022], 2022. [Online]. Available: https://lh4.googleusercontent.com/M7m2-IOMnZDmHdrKz5aCH4Hbv3EIWNkUbK577Uci1 hg8t5eN3SoUz1lbRI6ex1KASESM1JsYCF3_6swavH_Zl-8ElkBAIcvd9DTLLKXLyd4gGc3Y.

[13] F. Datakatalogen. 'Felles informasjonsmodell for person og enhet.' (), [Online]. Available: https://data.norge.no/informationmodels/8bfa055a-9b0c-40df-a7c2-2e58173eb2e0.

[14] Scrum. 'Scrum glossary.' [Sprint],[Sprint review],[Scrum board]. (), [Online]. Available: https://www.scrum.org/resources/scrum-glossary (visited on 19/05/2022).

[15] React. 'Testing.' (), [Online]. Available: https://reactnative.dev/docs/testing-overview (visited on 19/05/2022).

[16] E. B. S. Infrastructure. 'What is ebsi?' (), [Online]. Available: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/What+is+ebsi.

Appendix A

Project Contract

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for datateknologi og informatikk
Veileder ved NTNU: Mariusz Nowostawski e-post og tlf.: mariusz.nowostawski@ntnu.no, +47 483 42 678
Ekstern virksomhet: Symfoni AS Ekstern virksomhet sin kontaktperson, e-post og tlf.: Jon Ramvi, jon@symfoni.id, +47 408 70 027
Student: Mikkel Aas Fødselsdato: 29.02.2000
Student: Magnus Johan Gluppe Fødselsdato: 05.04.1997
Student: Erlend Johan Vannebo Fødselsdato: 20.09.2000

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 11.01.2022
Sluttdato: 20.05.2022

Oppgavens arbeidstittel er: Sammenhengende tjenester: Bedre innbyggertjenester med Verifiable Credentials — Digitalisering av «Søknad om dagpenger»

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven: Ingen relevante utgifter

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

X	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato:	
Veileder ved NTNU: Dato:	
Ekstern virksomhet: Dato: 17.01.2022	Jon [illegible]
Student: Dato: 17.01.2022	Magnus Gluppe
Student: Dato: 17.01.2022	[illegible] Johan [illegible]
Student: Dato: 17.01.2022	Mikkel Aas

Appendix B

Project Plan

Project plan

IDATG2900 - Bacheloroppgave

Authors:

Mikkel Aas

Magnus Gluppe

Erlend Johan Vannebo

31. jan. 2022

Goals and constraints

Background

This bachelor's thesis aims to simplify applying for unemployment benefits from Nav, Norway's Work- and Welfare Management. Currently, your personal information is stored in big centrally controlled databases, and we aim to give people back control of their information. By utilizing SSI and the blockchain, we want to gather all this information in a digital wallet that the user controls. This gives the user greater control of their information and simplifies the process of verifying it.

Project goal

A bachelor thesis's primary goal is to study and work in-depth on a limited subject area. We have the opportunity to learn about SSI and implement a practical solution in a business setting. Working closely with a private company and learning about a developer's responsibilities and work environment is central to our project. We want the most authentic experience possible to prepare us for working life.

The bachelor thesis is the most significant undertaking we will have as students, and it provides valuable experience with working on larger projects. Completing such an extensive assignment requires planning, organizing, and consistent work routines. Our group will be focusing on the learning outcomes of the project. We will work hard to complete the application, but what we can learn from working on the application is the primary goal.

Constraints

Time

The project is a bachelor thesis, so certain time constraints need to be considered. This project represents two-thirds of a normal workweek, so roughly 25 hours per week or 1600 hours on the project in total. The students involved are relatively inexperienced in the subject matter and conducting a project on this scale, so learning new concepts and a new programming language will take time.

The goal is not only producing a product for a private business, so a substantial amount of time must be dedicated to writing the report and other tasks involved with writing a bachelor thesis. It is common among bachelor students to get excited and try to develop their product as far as possible and neglect the other aspects of a bachelor thesis.

Money

This project is mainly development-focused, where the only high cost is labor. Since this is a student project, there are no labor costs to speak of. The only other issue is servers, but the school can provide this or other free options available to students.

Legal and ethical

The biggest ethical issue with this product is handling private and sensitive information. The intended function is for different government agencies to communicate and pass personal data. We must ensure this data is not compromised and used for nefarious means. It is unlikely we will work with anything other than dummy data, and if we get involved, we have to take the necessary precautions.

We have signed a cooperation agreement between the faculty, students, and Symfoni, our employer. Any disagreements or problems will first be solved internally and, if they persist, taken up with the faculty. There is an additional contract for maintaining anonymity provided by the faculty, but it is unlikely this will be necessary.

Both Symfoni and our bachelor group are committed to the concept of open-source software. Our code will be available to be studied and used by all. Our goal is to make a practical and secure solution so that others can build upon what we create.

Scope

Subject area

The subject area for this thesis is SSI, which stands for Self-Sovereign Identity. To understand what SSI is, we need to study some of the fundamentals it is built on. This includes public-key cryptography, crypto wallets, and distributed ledgers.

SSIs goal is to establish trust in an interaction where an individual ID is required. To establish this trust, one party will present their credentials to the other parties involved in an exchange. The relying parties may verify that the credentials came from an issuer they trust. A verifier may then verify the signatures given by the issuer requesting a claim and the holder's identity.[1] It is generally recognized that the user, known as the holder, has complete control of the verifiable credentials for the system to be self-sovereign, and their consent is required to use those credentials.[2]

In most SSI systems, holders generate and control unique identifiers called decentralized identifiers. Decentralized identifiers are a type of identifier that enables a verifiable decentralized digital identity. The identifiers and credentials are managed using crypto wallets and verified using public-key cryptography anchored on a distributed ledger.[3]

Crypto wallets used in SSI, commonly referred to as SSI wallets, are wallets where users can store their digital identity. It can resemble a physical leather wallet used to store one's driver's license. The user manages it, and the user may choose which credentials they wish to show. In the context of self-sovereign identity, it is a software application and encrypted database that stores credentials, keys, and other secrets necessary to identify oneself. Each participant in the SSI ecosystem (issuer, holder, and verifier) needs an SSI wallet to issue, hold, and verify verifiable credentials.[4]

Public-key cryptography is used to ensure a secure connection between two entities. The usage of encryption and private-public key-pair makes for a secure data transfer. Encryption is done by a mathematical algorithm taking, for example, in SSI, the credentials as input and giving unreadable alphanumeric characters as output. It is important to have a safe data transfer to ensure that the credentials are only shared with trusted entities.[5]

A distributed ledger is a database that exists across several locations or among multiple participants. In contrast, most companies use a centralized database in a fixed location. A distributed ledger is decentralized to eliminate single point failure and the need for a central

[1] https://www.sciencedirect.com/science/article/pii/S1574013718301217?via%3Dihub
[2] The Path to Self-Sovereign Identity (lifewithalacrity.com)
[3] https://www.eublockchainforum.eu/sites/default/files/report_identity_v0.9.4.pdf
[4] Trinsic Basics: What Are SSI Digital Wallets? • Trinsic
[5] Role of Public Key Cryptography in Self-Sovereign Identity | by Affinidi | Affinidi

authority to validate and authenticate transactions.[6] In SSI, distributed ledgers establish immutable recordings of lifecycle events for globally unique decentralized identifiers.[7]

Limit

The scope of the project limits us. It is uncertain how much of Symfoni's technology we should build upon, how much we are developing ourselves, and how much data we will mock instead of fully implement. If we build things from scratch, then a considerable amount of time will be spent setting up the SSI wallets. If they are more or less ready, we can focus on the app's front end and back end.

Task description

When using different government agencies, the agencies require information across various issuers. Today, gathering the needed information is sometimes a manual process. The applicant serves as a mailman for the agency by collecting data from the different issuers manually and bringing it back to the agency. After the information is gathered, an agency worker must check the information and either approve or deny the applicant's request. This process for both parties can be automated using Verifiable Credentials.

Our task is to cooperate with the Norwegian Labor and Welfare Administration to create a demo for automating the process of applying for unemployment benefits. To complete this, we will create a Verifiable Credential issuer that can issue the necessary proof, a Verifiable Credential verifier that can verify the received proof, and an app that can exchange the proofs between the issuer and verifier. For the issuer and verifier, we will use the javascript framework: Veramo, and we will use Symfoni AS's proof-of-concept version of the app and further build on it for our needs.

Project organizing

Responsibilities and roles

We have split the main parts of the project into different categories and delegated the main responsibility for each category to one of the members. Every group member has

[6] https://marcopolonetwork.com/articles/distributed-ledger-technology/
[7] Self-sovereign identity: Why blockchain? IBM Supply Chain and Blockchain Blog

responsibility for the entire project and is expected to work on each part of the project; however, the person with the main responsibility is responsible for doing the final quality check of the content being produced. This makes us use our different strengths to our advantage and distribute the workload evenly between us when performing quality assurance.

Code

Erlend is in charge of Code quality. We will try to uphold a high and consistent standard, using good practice throughout the project. Erlend will ensure we share the same principles like high cohesion, low coupling, sensible variable names, among others. The plan is to have code review every time a sizable part is merged into dev, and the code master will have a greater responsibility to make sure this happens.

Being in charge of the code quality is by far the most significant responsibility, as this includes: code conventions and standards, continuous integration, pipeline, code quality, and testing. This is a big task for one person, so the code master has to delegate some of these responsibilities to make sure not too much falls on one person.

Documentation

Mikkel is in charge of the overall documentation for the bachelor thesis. This includes documentation for code, the main report, and all the necessary attachments required. All members are expected to document the work, and the documentation master is not meant to micromanage everything but ensure that the overall style and standard are upheld.

SCRUM

The Scrum master for this team is Magnus. His job is to ensure that every team member is familiar with the Scrum process and follows the appropriate guidelines. With agile development, it is easy to exclusively focus on writing code and neglect the documentation and planning parts of the project. Scrum is lighter on the planning and documentation part compared to more traditional development methods, but it is still essential to use the issue tracker and sprint reviews. The Scrum master will ensure that the development process does not slide into chaos.

Manager

In addition to his role of overseeing the documentation, Mikkel will have the responsibilities of a manager. This will include ensuring deadlines are upheld, organizing meetings, ensuring information flow between the team members and external stakeholders, and the general morale of the team. There is some overlap between the Scrum master and the manager, but there should not be a coordination issue considering the team is small. .

Routines and group rules

Attendance

We have three scheduled physical sessions per week, and it is expected that each group member is present at those meetings unless given notice of absence in a reasonable time frame prior to the meeting. In addition to these physical sessions, there is one weekly online meeting with the employer where attendance is mandatory. The development team consists of 3 members, so communication is conducted informally—no need for official meeting summons.

Git

To ensure that good code is being produced, lowering the risk of critical errors in our project, and making sure everyone is up to date with what is being done, we have a rule that every git-merge needs to be reviewed by each member. We are not pushing changes directly into the development/master-branch to further ensure this, and every change in the development/master-branch happens through merging with another branch.

Work ethic

We want to have a good work environment, to achieve this we have a few group rules regarding our working sessions:

Rule #1: Everyone shall contribute equally.
Rule #2: Everyone should respect deadlines imposed by the group or the faculty.
Rule #3: Everyone is responsible for ensuring everyone has a task to do.

Rule #4: Everyone should give a short minute presentation of what they have worked on at the end of the day. Especially when we have done individual work.
Rule #5: Everyone shall lend a helping hand to a fellow group member that is stuck.

Conflicts regarding the rules will be solved internally. The adviser will be contacted if it is impossible to solve the conflict internally.

Planning, follow-up and documenting

Main division of project

Scrum as software development model

For this project, we have chosen to use the development method Scrum. Scrum is an agile development method, where the work is done in short intervals called sprints. This could include coding, documentation, project planning, and more. Agile development is an iterative method where self-reliant teams develop software through a dynamic work process. Another essential feature is the kanban board or the Scrum board, where different tasks are defined and assigned for the coming sprint.

We chose Scrum for several reasons, mainly because it is a development method the team is familiar with and enjoys. The project is organized around weekly meetings with the employer, thus making it natural to organize weekly or bi-weekly sprints. The kanban board allows a more fluid work environment, where one can assign new problems without the rigid structure of more traditional development methods.

Scrum is more commonly used in teams with 3-8 members, so our group is on the lower end of the scale, but it was still the best fit for our group. Usually, one would have a sprint review within the team and then present the results to the product owner. However, we will work more closely with Symfoni, our employer, and the scheduled weekly session will work as sprint reviews.

Schedule for status meetings

We will organize three large status meetings during our project's development segment—one for each milestone.

The first big status meeting will be at the end of February, when we plan to finish the first milestone in our development stage. We will discuss our progress in creating our first issuer and verifier in this meeting.

The second meeting will take place at the end of March. Here we will discuss how far we have come in finishing the application's backend.

The last status meeting will be in mid-April. We will discuss how far we have come in implementing our backend to the Symfoni app.

Quality assurance

Standards

We have set standards for the different parts of the project to ensure everyone has the same agreement on how the project should be run. We also have set these standards to keep the quality consistent across the entire project.

Code standards

To ensure our code is high quality, we need some standards when writing code. All code will be run through a linter, code readability will be in focus, and all files will follow the same structure.

To lint our code, we will be using ESlinter. Linting is important because it improves overall code quality, and it can accelerate our development by finding errors at an early stage.

Communicating the code one has written is consequential, especially when working as a team. We are therefore going to value code readability rather than performance. Code readability best practices will be used actively. All variables will be given sensible names, functions will only do one thing, and comments will be used to explain our code if needed.

Documentation standards

To ensure that the documentation is high quality, we have to set some standards. Some documentation standard topics we will enforce are citing sources, grammar, and when to document.

Sources must be stated whenever information is gathered from external resources. We will be using the LaTex built-in citing tool to cite these sources, and the citing style that will be used is APA.

Grammar is important to improve readers' comprehension. Therefore, we will use Google Docs' built-in grammar check in unity with Grammarly. Furthermore, a fellow group member will proofread each written paragraph to further improve the text's readability.

Information is often lost during the process if not documented immediately. Hence, documentation will be done continuously during the process. Early-stage documentation does not have to be of high quality.

Meeting standards

The plan is to have two weekly meetings, one with Symfoni and one with the faculty advisor. If possible, all members should be present for both these meetings, but it is not essential. Meeting notes are taken at each meeting, describing what to do in the coming week. Meeting minutes will be taken at the more infrequent, more extensive status meetings. These status meetings should be organized around the project's milestones to discuss that part of the project in detail.

In the beginning, all meetings will be digital, but as the Covid situation develops, we will attempt to have meetings with the faculty advisor in person. The digital meetings will be held over Microsoft Teams or in Discord. Internal meetings with the student group will be less official and conducted in one of the scheduled work sessions, usually in person at the campus.

Configuration management

Tools

This segment contains all the relevant tools we plan to use in our bachelor thesis. We describe what each tool is and how we intend to use them.

GitHub

GitHub is a code hosting platform for version control and collaboration. In other words, it makes it easy for a group of developers to work together on the same project.[8]

We chose to use GitHub instead of the many other version control software because of our employer. Our employer's business exists on GitHub, and they opened a repository in their organization that we could utilize. Having our repository in their organization makes it easy for them to keep track of our progress and aid us if we are stuck on an issue. GitHub also offers other helpful tools like repository analytics and an issue board.

Google Drive

Google Drive is a file storage and synchronization service. It was developed by Google and launched on April 24, 2012.[9]

In our project, we will use Google Drive to share important documents. These documents will consist of diagrams, reading material, meeting notes, contracts, and many more that are not directly a part of the main report. Furthermore, Google Drive features neat documenting tools like google documents and spreadsheets that will come in handy.

Google documents and spreadsheets

In our project, we are using Google Docs. Google Docs is a part of Google Docs Editors as an in-browser word processing software[10]. It is software that lets us write simultaneously in the same document, updates it for everyone in real-time, and automatically saves it to the

[8] About wikis - GitHub Docs

[9] Official Google Blog: Introducing Google Drive... yes, really

[10] Google Docs: Free Online Document Editor | Google Workspace

cloud after every change. Google Docs is also very well integrated with Google Drive, which makes creating, sharing, and writing documents a seamless experience for us. Google Spreadsheets is also a software within the Google Docs Editors package and has the same benefits as Google Docs, but for spreadsheets instead.

We use Google Docs for documents related to, but not a part of the final report such as the project plan, meeting notes, etc. We use Google Spreadsheets for our Gantt-chart and timekeeping.

WebStorm and VScode

WebStorm and VScode are the IDEs we will be using for this project. An IDE is an integrated development environment used for source code editing, build automation and debugging. WebStorm is built upon IntelliJ with the programming language TypeScript in mind. VScode is flexible and supports many different languages, including typescript.[11][12]

We will mainly use WebStorm as it has many desirable features, mainly how easy it is to build projects. WebStorm requires quite a lot of power, so for smaller simpler coding tasks VScode is more suitable. By using GitHub for version control we can switch between IDE's without issue.

Discord

Discord is an online communications platform with both voice and text channels. It uses voice communications and multimedia sessions over IP networks. Discord was developed by Discord Inc., released in May 2015, and is one of the most commonly used platforms in the tech world.

Discord is the main communication between the students, the faculty supervisor and the employer. You can create different channels where you invite members of your choice. We have an internal discord chat just between the students, and a larger chat with Symfoni, our employer.[13]

[11] https://code.visualstudio.com/docs/editor/whyvscode
[12] https://www.jetbrains.com/webstorm/
[13] https://discord.com/company

Microsoft Teams

Microsoft teams is primarily a business communication platform. It was developed by Microsoft, and launched in March 2017 to join the Microsoft 365 family of products. People are organized into different teams with text channels, and you also have video conferences where you can share your screen to hold presentations.[14]

The official meetings with our employer are conducted through Microsoft teams. While more unofficial correspondence is done through discord, teams have a calendar function where you can schedule meetings, so it is a more organized way to hold meetings.

Overleaf

Overleaf is a collaborative cloud-based LaTeX editor that is used for writing, editing, and publishing, primarily scientific documents. The beta version of Overleaf launched on 16 January 2014.[15]

We will be using Overleaf to write our bachelor thesis, often referred to by us as the main report. LaTeX is widely acknowledged as one of the best text editors to produce scientific articles. However, its features often rely on the user to write LaTeX code. On the other hand, Overleaf provides the same features, but it is a rich-text editor, meaning that LaTeX code knowledge is not required. Using LaTeX in Overleaf makes the produced article look more refined and allows for complex designs.

Windows as an operating system during development

For our project development, we have decided to use Windows as the operating system. The reason behind this is that everyone in the group is familiar with using Windows, and to avoid complications with developing and building across different operating systems. If any systems require Linux to operate we can use WSL, which is Windows Subsystem for Linux. It is a compatibility layer that makes it possible to run Linux executables.

[14] https://www.compete366.com/blog-posts/microsoft-teams-what-is-it-and-should-we-be-using-it/

[15] About us - Overleaf, Online LaTeX Editor

Veramo

Veramo is a JavaScript framework that aims to make it easier for developers to use cryptographically verifiable data in their applications. It is designed to make it easy to use next-generation features like DIDs, verifiable credentials, and data-centric protocols.[16]

ESlint

ESLint is a static code analyzing tool created by Nicholas C. Zakas in 2013. It controls code quality and code style by analyzing code to see if it fulfills certain rules. Furthermore, it comes with many preset rules for common standards, but it also allows one to create and define rules that code shall follow. [17] In our project, we will use ESLint to help us keep our code clean and maintain good code quality.

TypeScript

TypeScript is a strongly typed programming language built on JavaScript, and it is better suited than JavaScript on larger projects due to it being a typed language. It allows for tighter integration with the editor, which can catch errors in the editor.[18]

We will be using TypeScript due to our employer's wish. It is the language used in the Veramo framework and our employer's company codebase.

[16] Introduction | Performant and modular APIs for Verifiable Data and SSI (veramo.io)
[17] https://eslint.org/docs/about/
[18] TypeScript: JavaScript With Syntax For Types. (typescriptlang.org)

Plan for inspection and testing

For our project, we will ideally use unit testing to test our code actively. The testing framework we have chosen is Mocha, and we will use ESlint to ensure the same code standards and conventions for the entire project.

We will do some simple user testing towards the end of the development phase. Once we have connected our backend to the Symfoni app, we can do some basic testing to see if the wallet is operational and the verifiable information is relevant. Our team will not be involved with the design decisions, so the user tests will solely focus on the technical aspects of the wallet.

Risk analysis at project level

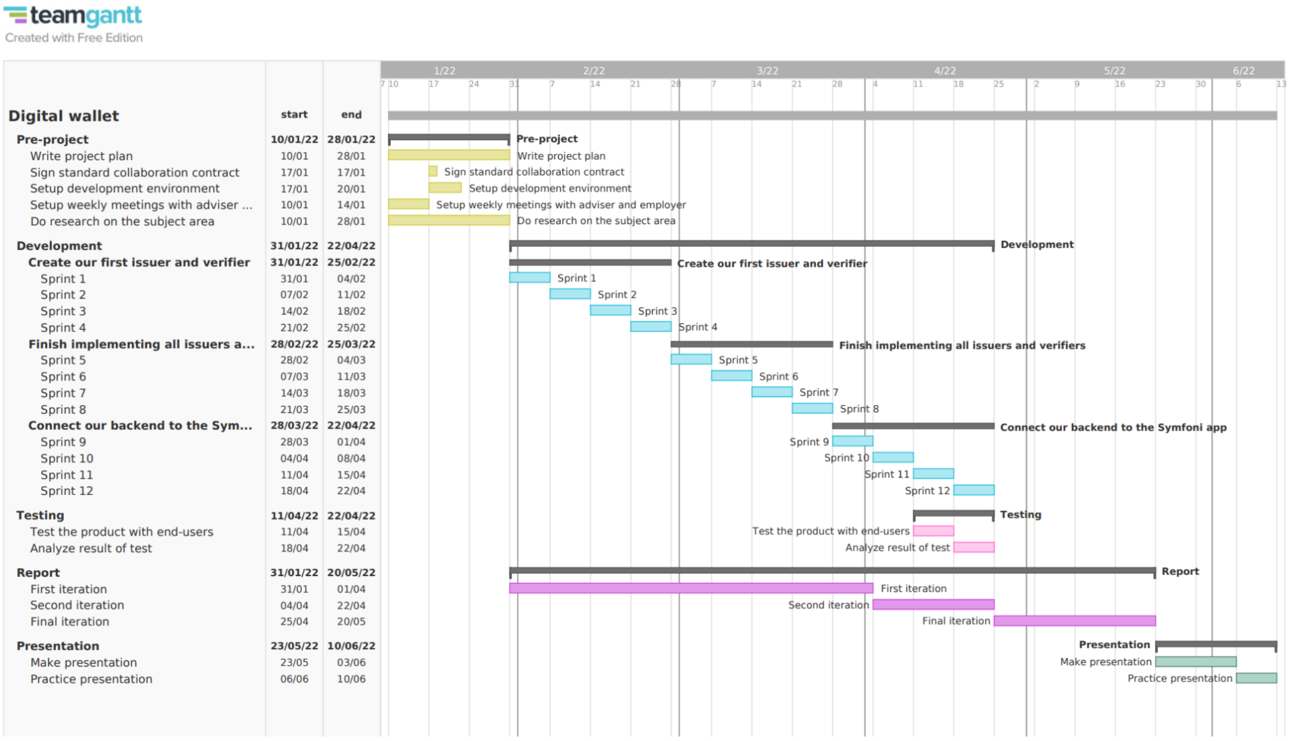
Risk Assessment

Risk	Likelihood	Impact
1 or more team members getting sick with covid	High	Low: Can continue with development digitally
Time constraints making it impossible to finish the product on time	Moderate	Moderate: Creates a crunch period, which makes a suboptimal product
Break in communication/loss of interest from employer	Low	High: Without input from the employer the product might not meet their desired expectations and specifications
Loss of communication with adviser	Low	High: Essential for guidance and feedback
Internal complications within the bachelor group	Low	High: Conflicts within the group is the biggest risk, as it would greatly impact all aspects of the project

Plan

Gantt chart

Gantt chart.pdf



Digital wallet	start	end
Pre-project	**10/01/22**	**28/01/22**
Write project plan	10/01	28/01
Sign standard collaboration contract	17/01	17/01
Setup development environment	17/01	20/01
Setup weekly meetings with adviser ...	10/01	14/01
Do research on the subject area	10/01	28/01
Development	**31/01/22**	**22/04/22**
Create our first issuer and verifier	**31/01/22**	**25/02/22**
Sprint 1	31/01	04/02
Sprint 2	07/02	11/02
Sprint 3	14/02	18/02
Sprint 4	21/02	25/02
Finish implementing all issuers a...	**28/02/22**	**25/03/22**
Sprint 5	28/02	04/03
Sprint 6	07/03	11/03
Sprint 7	14/03	18/03
Sprint 8	21/03	25/03
Connect our backend to the Sym...	**28/03/22**	**22/04/22**
Sprint 9	28/03	01/04
Sprint 10	04/04	08/04
Sprint 11	11/04	15/04
Sprint 12	18/04	22/04
Testing	**11/04/22**	**22/04/22**
Test the product with end-users	11/04	15/04
Analyze result of test	18/04	22/04
Report	**31/01/22**	**20/05/22**
First iteration	31/01	01/04
Second iteration	04/04	22/04
Final iteration	25/04	20/05
Presentation	**23/05/22**	**10/06/22**
Make presentation	23/05	03/06
Practice presentation	06/06	10/06

Activities

The group has scheduled three physical work sessions at the campus a week. During these we will work with the current tasks laid out in the project plan: pre-project planning, developing code, writing the rapport, and preparing for the final presentation. We will have digital sessions over discord when physical meetings are not scheduled, and these consist of the same tasks as the physical sessions. Later in the development stage, it is more open for individual work, both on the codebase and the rapport.

In addition to having meetings with the group, we have regular meetings with our faculty advisor and our employer. These meetings will be digital on discord or teams. We have also set aside some time for user testing, and ideally, this will be done in person, with a random sample of participants.

Milestones

Milestone 1: 31. jan 2022, finish the project plan and sign the standard agreement for academic collaboration.

Milestone 2: 28. feb 2022, create our first issuer and verifier.

Milestone 3: 31. mar 2022, finished implementing most of our issuers.

Milestone 4: 30. apr 2022, connect our back-end to the Symfoni app.

Milestone 5: 16. may 2022, finish the bachelor thesis report

Milestone 6: 6. jun 2022, finish our presentation

Appendix C

Task Description

Symfoni – blockchain

Denne oppgaven trenger en nærmere beskrivelse før den endelig godkjennes.

Kontaktperson : Jon Ramvi - jon@blockchangers.com

Symfoni AS is a startup with five people operating out of Oslo. We work for and with the Norwegian government, and Innovation Norway and the Ethereum Foundation fund us.

The technologies are the same, but there are different students can choose from.

The projects are all in the intersection between blockchain, Verifiable Credentials (VC), and government services.

Communication can be done in Norwegian or English.

Technologies:

- Blockchain, Ethereum, Smart Contracts, Solidity, Hardhat, Typescript
- Cloud, Azure, Docker, NestJS, Prisma, Postgresql
- Frontend: (React, React Native, HTML, CSS, JS, NPM…)
- WalletConnect, Veramo, DIDs, SSI, and eIDAS 2.0

Brønnøysundregistrene (BR), The Norwegian Business Registry
We are developing two solutions for BR.

1. BR Økosystem (BRØK), an infrastructure for data exchange and seamless services. It's built blockchain technology and is planned to be released on public Ethereum on a Layer 2 scaling solution (Arbitrum). The work is all open-source. There are problems to be solved within authentication, shared services, and privacy.
2. A cap table platform on top of BRØK. Unlisted companies can move their cap tables to BRØK, where the private sector utilizes them to offer customers insights, second-hand markets, issuance, etc. The work is all open-source. There are problems to be solved within data exchange (verifiable credentials), among others.

Patentstyret, the Norwegian Patents Office
We are part of a project to evaluate using BRØK for Trademarks and Licenses. Technically these can be represented as NFTs (Non-fungible Tokens). This project hasn't started, so there are infinite problems to frame and solve.

Symfoni ID
We are developing solutions for issuing VCs and verifying them. We are also developing an app for communication between these parties. Their work is part of the EU's ambition of a new digital wallet for the whole of Europe (eIDAS 2.0). The work is all open-source. The actual problems will have to be defined when the project starts.

Mvh
Jon Ramvi
Gründer av Symfoni AS

Appendix D

Symfoni Meeting Notes

12. jan. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, Asbjørn Riddervold, Jonas Johan Solsvik, og Erlend Johan Vannebo

Notater

- Faste møter mandag kl 15
- SSI-bok og andre ressurser
  - https://www.bookdepository.com/Self-Sovereign-Identity-Decentralized-digital-identity-verifiable-credentials-Alex-Preukschat/9781617296598?ref=grid-view&qid=1641993165127&sr=1-1
  - https://www.w3.org/TR/did-core/
  - https://www.w3.org/TR/vc-data-model/
  - https://identity.foundation/didcomm-messaging/spec/
- Sprint, kanban (sprint en uke av gangen, planlegge med oppdragsgiverne i de faste møtene på mandager)
- Gå gjennom kontrakt med veileder, fylle ut, gi til Symfoni før møtet
- Hvilken utfordring, forsøker å finne problemet mens vi jobber med det (bruker deres verktøy til å begynne med)
  - Trenger litt tid til å lage en oppgave
  - Bruke deres verktøy (ikke spesielt interessant ifølge dem) burde finne mer spennende utfordringer som kan drøftes
  - Utfordringer dukker kanskje opp underveis
- Se på GitHub for inspirasjon (app)

Gjøremål

- □ Gå gjennom kontrakt med Mariusz
- □ Lese på ressursene
- □ Sjekke ut Symfoni sin GitHub
- □ Utarbeide prosjektplan
- □ Videreformidle møtet til Mariusz (skal han være med på ukentlige møter?)

18. jan. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- Det finnes en APP som kan brukes
- Veramo har laget en ferdig SDK som gjør mye av VC arbeidet, Web resolving, og DIDs
    - Vi burde bruke Veramo til å utstede data (issuer) den er de forskjellige aktørene
- Mesteparten av dataen er mocked
    - Nav må ha data fra Symfoni (utesteder), for at NAV skal vite at det er Jon Ramvi fra Symfoni så må NAV slå opp i DID domene i Web for å se at det faktisk er Jon Ramvi fra Symfoni
- Holder har vi fra Symfoni
- Issuer og Verifier lager vi med Veramo
- Skatteetaten
    - NAV i utlandet som trenger bevis fra skatteetaten om at barnet bor i Norge
        - mocked
    - Det interessante er at det har verdi i samfunnet
- NAV kommer med QR kode for bevis, du bruker appen din
- Lager en VC med bankID som du sender til skatteetaten osv…, sender bevisene tilbake til NAV
- Strukturert data
    - Kontekst i VC for å definere nøkkelpunkter
- Lage en kontekst som er tilpasset NAV og høre med NAV om det inneholder
- Appen scanner en QR-kode og gir beskjed til en back-end om hvilken Issuer og hvilken data
- Symfoni har laget bankID ekte og test
- **Få data til å flytte seg fra issuer til verifier gjennom appen**
    - Bruke veramo til å opprette issuers og verifiers
- Ingen sterke meninger om studentlisens på IDE
- Håper at APP er forhåndsdefinert bra nok
- Symfoni mener at vi burde legge listen lavt og sette vanskeligere mål etterhvert

Gjøremål

- ☐ Lage diagrammer
- ☐ Lære Veramo

31. jan. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- Bare ansettelseskontrakten er nok
    - Som strukturert data
        - lønn
        - stillings..
    - sendes i flyten til nav
    - De vil gjerne hjelpe med å definere parametere
- Fokusere på arbeidsgiver og ansettelse
- Ingen preferanse rund did: metode
    - kan argumentere litt for begge deler
    - vi kan komme med anbefalinger
    - Det skal ikke være så viktig
    - Skal kunne samhandle basert på did:anything
- DID
    - hvilke koblinger
    - sqlite er en fin måte
- Trenger ikke server til å begynne med
- Kan prøve å spinne opp servere senere
- Bruk **EU Digital Wallet** som konsept
    - Det kan bety slik … for dagpenger
- Konkretisering av EU digital wallet
    - ENDGAME
        - mer desentralisering og mindre store
        - enklere å integrere med andre systemer
        - Du er sjef over din data og hvordan den flyter
- Både klienter og servere kan gjøre begge deler

10. feb. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- Slide about how to should be removed
- More focus on VC data
- find an employment agreement online
  - create a VC based on that word file
- create a context for the claims
  - explains the thought behind the context
- Transfer from contract to VC
- Create a hash of the pdf and put it in the VC
- Steg 5 is the main step, this is the end goal
- We need to create DIDS, VCS, and VC presentation
  - VC presentation is important as it can showcase data from many different
- Presentation should include a VP
- did:web:Symfoni, did:blockchangers.com:Symfoni
  - Should use blockchangers.com
  - need a private key from symfoni so that anyone
- well-known json
  - go to the signature and see that it works
- did method defines where the did document should be
- Universal resolver
- suggest that we use did web for nav
- read the spec
- Jump straight to how the sausage is made
  - Take off the fluff
  - When writing a thesis we have to take it back
  -
- To present, DID document for Symfoni, DID document NAV, and VC
- jwt checker
- Make schema for employment contracts

Gjøremål

- ☐ Create DID document for Symfoni
- ☐ Create VC for employment contract
- ☐ If we have time, create DID document for NAV

14. feb. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- Bryte opp VC til to person VC'er istedenfor å ha employer og employee i samme VC
- Lage context/schema i JSON
    - Lenker for referanse:
        - [Verifiable Credentials JSON Schema Specification (w3c-ccg.github.io)]
        - [https://www.w3.org/ns/did/v1]
- Lage VP
- Fargekode JSON dokument (low pri)

Gjøremål

☐ Se på verifiable presentation

21. feb. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- frontend hypotetisk
- betaler lønn gjennom et regnskapssystem - bruke data derfra
- visma - frontend - > legger inn en ny ansatt ->
- QR - kode tar deg til en frontend hvor du kan vise hvem du er med en ID VC og få arbeidskontrakt VC tilbake
- wallet connect for å overføre data
- regnskapssystem -> arbeidsgiver har et frontend som legger inn en ny ansatt, taster inn stillingsprosent osv, trykker registrer ansatt også sender vc til ansatt, hvordan skjer dette?
- QR kode sendes på E-post
- Semantic web

Gjøremål

- ☐ fikse slik at VC'ene faktisk bruker digdir
- ☐ teknisk diagram for flyt
- ☐ laste opp schemas på Git
- ☐ lage flytdiagram for trust

14. mar. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- bytte rekkefølge, først connecte, så lage VC
  - trenger motpartens did fra connection
- Slå sammen de tre øverste, front-end, back-end og mobiltelefon
- får epose med qr kode - fra hvem og til hvem
- ikke så farlig om vi bruker did:ethr eller did:web
- nav oppretter sin egen wallet, database register verifiserer NAV
  - ikke businessVC for NAV, heller et register
- list use case
- lage matrise brukes ofte vs brukes lite
- flowchart - user må kontrollere nav did'en
  - gjør det 'valgbart'
- ikke bry oss om terminologi
- vi lager eksempel skjemaer, eu kommer med virkelige skjemaer senere
- mulig å lage

21. mar. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Asbjørn Riddervold, og Erlend Johan Vannebo

Notater

- sende QR på e-post
- bytte om mellom arbeidsgiver og avsender (sende QR kode)
- wallet-connect
    - 2 enheter vil snakke sammen
    - må dele en hemmelighet
    - deles ved at man "off the band" scanner qr-kode til en kanal hvor man kan møtes
    - AES nøkkel
- register finnes ikke
- v1/v2 problem
- når dapp er koblet med wallet må de støtte samme RCP
- wallet har 1 agent
    - lager 4 dapper
        - bytter logo schema, generell funksjonalitet
        - 1 dapp for stat
        - 1 dapp for arbeidsgiver
        - 1 dapp for bruker
        - 1 dapp
        - 1 wallet-connect som alle bruker for å snakke sammen
- taster inn person id i front-end, front-end sender deg en

bruker er allerede registrert i firmaet

28. mar. 2022 | Meeting: Symfoni

Deltakere: Mikkel Aas, Magnus Johan Gluppe, Jon Ramvi, og Erlend Johan Vannebo

Notater

- async storage lagrer på enheten du er på
- fylle form for å lagre person data
  - generer senere når person vil ha VCen sin
    - genereres basert på personnummer
- express tungvint med mobil enhet
- express nødvendig for lagring av persondata
- react context for flyt av data mellom components
- bruke mer types i react native kan være lurt, men er ikke strengt nødvendig på top-level
- vurdere å bruke didComm

Gjøremål

- □ lage user database for både stat og symfoni
- □ lese om react context

Appendix E

NAV Meeting Notes

18. feb. 2022 | Vs: Verifiable credentials og dagpenger

Deltakere: Mikkel Aas Jon Ramvi magnusgluppe97@gmail.com ejvanneb@stud.ntnu.no magnujg@stud.ntnu.no andre.roaldseth@nav.no asbjorn@symfoni.solutions eivind.havnelid.royrvik@nav.no elin.vethe@nav.no mikkelaa@stud.ntnu.no

Notater

- Data NAV ønsker å ha med:
    - VC fra forsvaret
        - fra dato til dato
    - VC om personinformasjon
    - VC arbeidsførhet
    - VC om arbeidsforhold (viktig)
        - tidligere inntekt (har mye, men ikke i utlandet)
        - tidligere arbeidstid (har mye, men kvalitet ikke nødvendigvis god)
            - hvor mye du har jobbet
            - fast stilling / varierende (kan lage array med start og slutt dato)
            - bruker ukentlig arbeidstid, plusser sammen siste år eller 3 år og deler på 52 eller 156
        - status på arbeidsforholdet
        - oppsigelsestid (viktig)
            - ulikt om det er siste lønningsdag og siste arbeidsdag
            - Når man slutter å få lønn en ting
            - siste dag med lønn en annen ting
        - oppsigelsesdato
        - sluttårsak
        - avskjed
        - sluttavtale
        - Hvem har signert avtalen (ID)
        - Årsak til oppsigelse
    - Egen VC for oppsigelse (viktig)
        - siste betalingsdag
        - siste arbeidsdag
        -
        - oppsagt innad prøveperiode

        - oppsigelsesgrunnlag
    - VC om arbeids og forsikringsperioder i andre eøs-land (liten andel søkere)
        - lovvalg
        - grensearbeider
        - arbeidsland
        - bostedsland
        - utveksling av denne informasjonen skjer via sed - strukturerte elektroniske dokumenter
    - VC utdanning
        - hvor
        - når
        - grad? (se ppt)
- Annet
- 80% ordinære dagpenger
- 200 000-300 000 søknader i året
    - 21 dagers frist på svar
    - ekstremt mye manuelt arbeid
- Alle vilkår må være oppfylt
    - 40% får avslag
- bruker mye tid på å få tak i dokumentasjonen (nav må hjelpe?)
- mangelbrev i 1 av 3 saker
- trenger ikke et helt dokument, bare noe av informasjonen
- subsumsjon
    - ha regel og relevant fakta som backer regelen
    - 16 hovedvilkår
        - under her hovvedtiltak finnes det unntak
        - krever presise opplysninger
- minste arbeidsinntekt
    - minst 1,5g (150 000) siste 12mnd, minst 3 G siste 36 mnd
    - avmelding til skatteetaten, nav henter data fra skatteetaten
    - verneplikt er unntak (3mnd de siste 12mnd)
- når du ble agt opp
- Finnes unntak, f.eks hvis du ikke har tjent nok, men har vært i førstegangstjenesten i 3 måneder går det bra likevel.
- må ha 50% redusert arbeidstid
- utdanning

    - lite til ingen strukturert data

case arbeidsavtale:

- arbeidsgiver
- arbeidstaker
- arbeidssted
- arbeidsforholdstype
- Rammeavtaler
- yrkesbeskrivelse
- stillingstittel
- lønn
- lønningstype
- tariffavtale
- startdato
- eventuelt avtalt sluttdato

- avtale rundt oppsigelse
    - dato
    - tre måneder fra den første
    - minimum 1 mnd i prøveperioden
- arbeidstid
    - stillingsprosent
    - når
    - eventuell sesong
    - ikke skift, skift turnus, rotasjon
    - antall timer pr. uke som tilsvarer full stilling

- DIGDIR
    - Definerer verdiene

- Harmonisering av begreper er viktig
    - NAV bestemmer

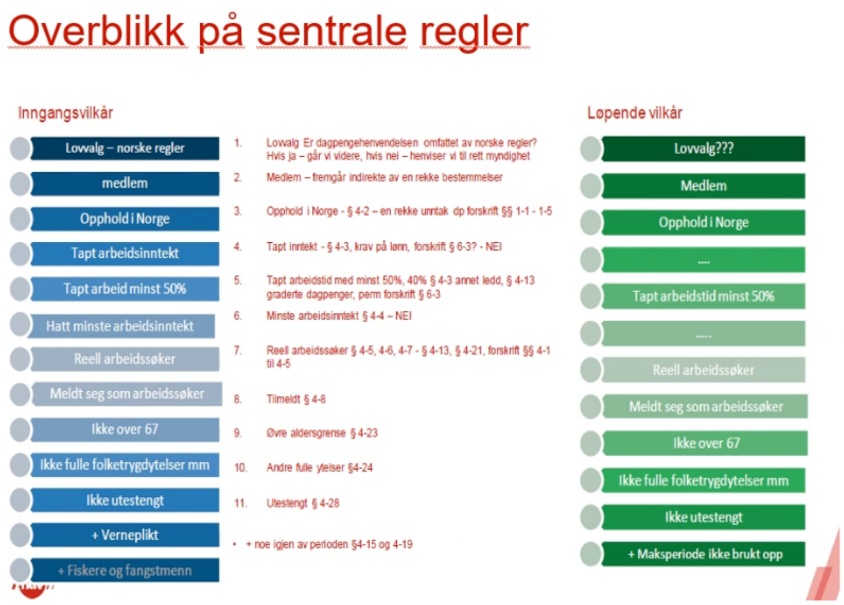
- A-ordning
    - issuer VC'er?
- Flytdiagram for NAV

        - brukerflyten
            - hvordan du forholder deg til de ulike aktørene
- Vise hvilke

- Viktig å skille mellom siste arbeidsdag og siste dag med lønn
- Ulike typer turnus osv. mer dybde i employmentStatus?
- Oppsigelse VC

Gjøremål

- ☐ lage et flytdiagram (forretning og teknisk)
- ☐ Lage en egen VC for oppsigelse
- ☐ Oppdatere arbeidskontrakt VC'en med data vi fikk fra nav



Overblikk på sentrale regler

Inngangsvilkår

- Lovvalg – norske regler
- medlem
- Opphold i Norge
- Tapt arbeidsinntekt
- Tapt arbeid minst 50%
- Hatt minste arbeidsinntekt
- Reell arbeidssøker
- Meldt seg som arbeidssøker
- Ikke over 67
- Ikke fulle folketrygdytelser mm
- Ikke utestengt
- + Verneplikt
- + Fiskere og fangstmenn

1. Lovvalg Er dagpengehenvendelsen omfattet av norske regler? Hvis ja – går vi videre, hvis nei – henviser vi til rett myndighet
2. Medlem – fremgår indirekte av en rekke bestemmelser
3. Opphold i Norge - § 4-2 – en rekke unntak dp forskrift §§ 1-1 - 1-5
4. Tapt inntekt - § 4-3, krav på lønn, forskrift § 6-3? - NEI
5. Tapt arbeidstid med minst 50%, 40% § 4-3 annet ledd, § 4-13 graderte dagpenger, perm forskrift § 6-3
6. Minste arbeidsinntekt § 4-4 – NEI
7. Reell arbeidssøker § 4-5, 4-6, 4-7 - § 4-13, § 4-21, forskrift §§ 4-1 til 4-5
8. Tilmeldt § 4-8
9. Øvre aldersgrense § 4-23
10. Andre fulle ytelser §4-24
11. Utestengt § 4-28

- + noe igjen av perioden §4-15 og 4-19

Løpende vilkår

- Lovvalg???
- Medlem
- Opphold i Norge
- ....
- Tapt arbeidstid minst 50%
- .....
- Reell arbeidssøker
- Meldt seg som arbeidssøker
- Ikke over 67
- Ikke fulle folketrygdytelser mm
- Ikke utestengt
- + Maksperiode ikke brukt opp

16. mar. 2022 | VC og dapenger

Deltakere: Mikkel Aas Jon Ramvi magnujg@stud.ntnu.no andre.roaldseth@nav.no asbjorn@symfoni.solutions eivind.havnelid.royrvik@nav.no

Notater

- Ønsker mer automatisering, ingen personer går igjennom søknaden
- Noen har ikke bank-ID, ikke så mange hos nav men flere hos IDI?
- Utfordringer med bedrifters regnskapssystemer og at de ikke gir tilstrekkelig med informasjon - schemas løser dette?
- Faktisk usikker på om det er nødvendig å utstede en dagpenger-VC
- Har en detaljert særordning for oppsigelsesgrunnlag?
- Varierende arbeidstid bruker gjennomsnitt siste år. Ellers fast ordning?
- ulike kategorier for vanlig ansatt, frilanser(oppdragstaker)/fast frilanser, næringsdrivende osv også har de egne kategorier for hvor mye man jobber under disse
- forskjell på fast stilling og heltid, man kan jobbe fast 50%. Medarbeid utover den faste stillingen, men usikker på hvordan en slik kontrakt ser ut?
- Dagpengeregelverket tar ikke hensyn til når man får penger og hvor ofte. De bruker A-ordningen som forholder seg til månedlig.
-
- Definere begreper for større gjennomgang

Gjøremål

☐ lage liste med data som NAV kan se over

8. apr. 2022 | VC, dagpenger og NAV

Deltakere: Mikkel Aas Jon Ramvi magnujg@stud.ntnu.no andre.roaldseth@nav.no hakon.rostad@nav.no petter.hafskjold@nav.no asbjorn@symfoni.solutions eivind.havnelid.royrvik@nav.no

Notater

- gi andre tilgang til å bruke VC'er er en stor business case
- Hvordan differensierer det seg fra andre signerings-dynamikker?
  - at brukeren er i sentrum
- godt case på fordeling av data brukeren burde ha selv og registerdata
- **FULLMAKT**
- må være et alternativ til den "vanlige" måten å gjøre det på.
- må integreres med lønns og personalsystemer
- samfunnsperspektiv sett i fra bølger av arbeidsledighet
- et alternativ hvis vi ikke skal bruke registerdata
  - rett og slett alternative kilder, hvor de foreligger etc.
- hva skal til for at dette systemet faktisk blir tatt i bruk
- nokios 3. oktober ish, 3 dagers workshop
- konkretisere case
- kan være nyttig i mange caser
- ikke konkurrere mot gode registerdata
- todelt
  - hvilken informasjon kan vi gi
  - hvilke data kan vi utstede

Appendix F

Adviser Meeting Notes

14. jan. 2022 | Meeting: Mariusz

Participants: Mikkel Aas, Magnus Johan Gluppe, Mariusz Nowostawski, and Erlend Johan Vannebo

Notes

- Redraw form based on what needs to be developed. Models and infrastructure for it to be run. Diagram for existing building blocks. UML
- Compatible SDK for different wallets
- Mocked or real, talking to the REST API, handshakes etc.
- Decide what data should be mocked (bankID, certain VC)
  - To narrow down or expand the scope
- Native mobile app vs web front-end
- Bigger status meetings: Organize by milestones (around once a month-ish)
- Include management plan in project plan
- Take notes on improvements for the thesis report
- Fill out the contract

Tasks

- ☐ Create a list of questions regarding which modules already exists and what data we are going to mock etc.. to further narrow down or expand the scope
- ☐ Create diagram of existing tools and protocols after questioning Symfoni
- ☐ Go through the project plan with Mariuszs input in mind
- ☐ Fill out the NTNU contract and present it to Symfoni

21. apr. 2022 | Bacheloroppgave-arbeid

Deltakere: Mariusz, Erlend, Mikkel, og Magnus

Notater

- add mariusz to overleaf report
- code snippets are good, but don't over do it
- If code snippets become too big, consider putting code snippet in the appendix
- working backwards is fine, not illegal
- writing a reflection summary from the notes we made in the meeting
- process adjustments
- don’t add all the weekly notes
- write about what went wrong, what we learned
- EVERYTHING that we did wrong and would not repeat.
- How would we re-do the process?
- How do we decompose our project
    - api design
    - dependencies
    - architecture
    - UML - diagram
    - quality insurance
    - CI
- Original design vs. end product
- split introduction
    - the issue at hand and the technologies at hand
    - another part that is the team
- conclusion AND future work
- maybe make a discussion part that is more chatty and reflecting than the conclusion part.
- Not an installation, more like how to run the demo.
    - Present and discuss the scenario.
        - How is the demo supposed to be run
- conceptual design paragraphs
    - user is in the center
- compare our scenario to bankID
- ping mariusz when we have a draft / when we are done with the chapter
- Who is the trust anchor?
    - How it works non SSI

    - How it works in SSI
- did:web vs. did:ethr
    - did:web trust anchor is domain name

Tasks

□ add Mariusz to the overleaf report

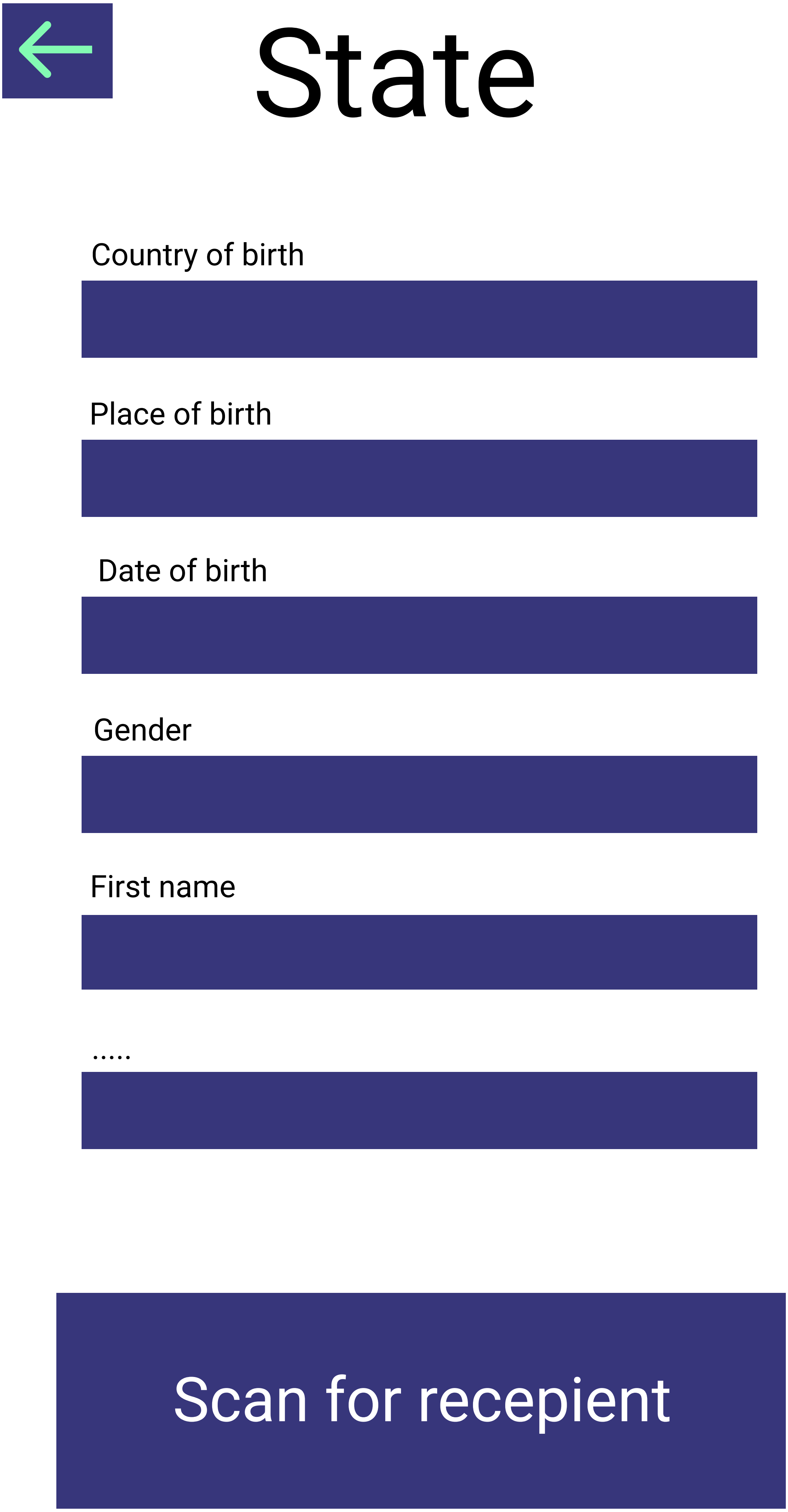
Appendix G

Low fidelity prototype

A fully functioning low-fidelity prototype can be found at: `https://www.figma.com/proto/Ytlv4tvPCH8dOiw2pd9LQN/EU-Digital-Wallet-DEMO?node-id=1%3A6&scaling=scale-down&page-id=0%3A1&starting-point-node-id=1%3A6&show-proto-sidebar=1`

State

Create personVC



State

Country of birth

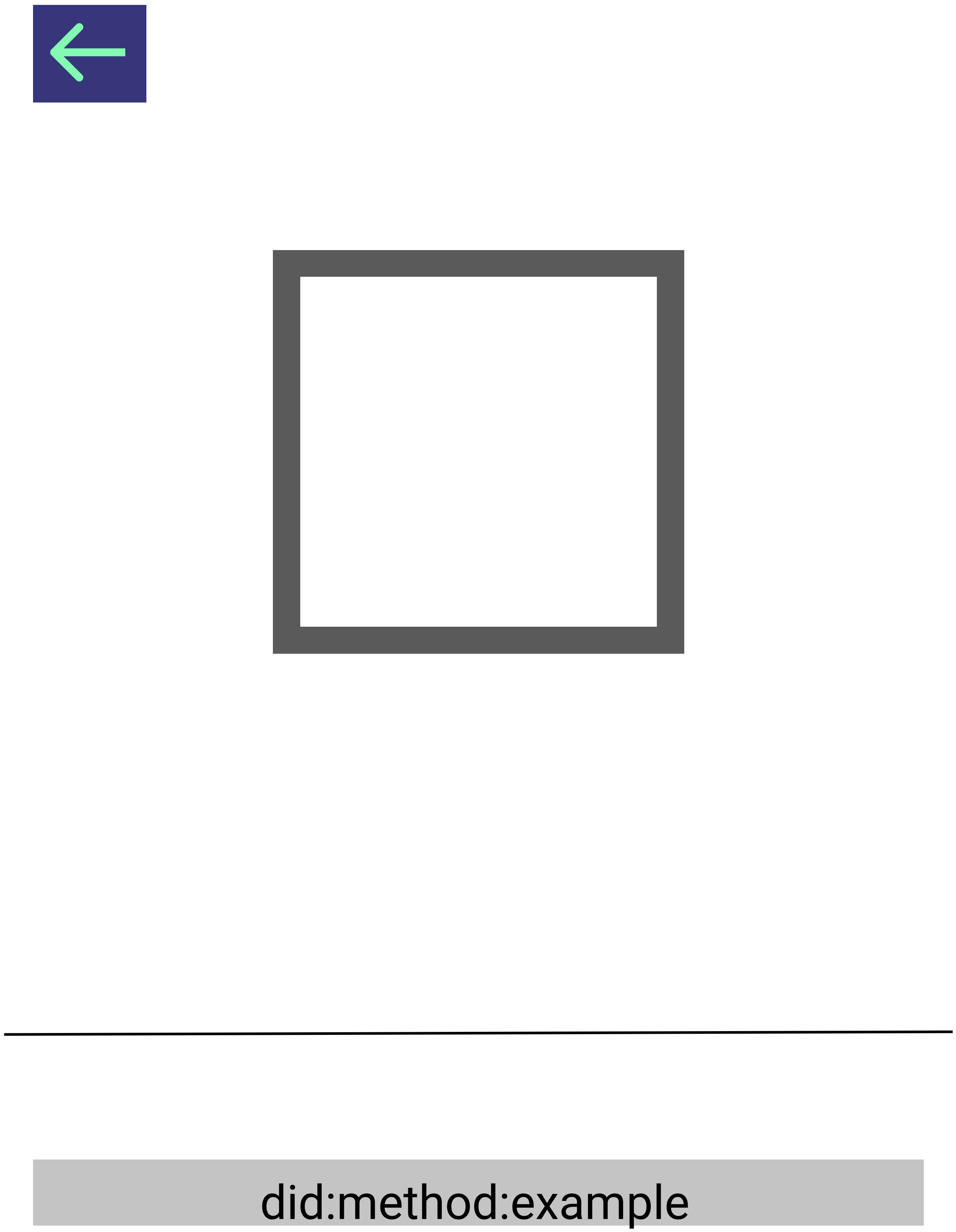
Place of birth

Date of birth

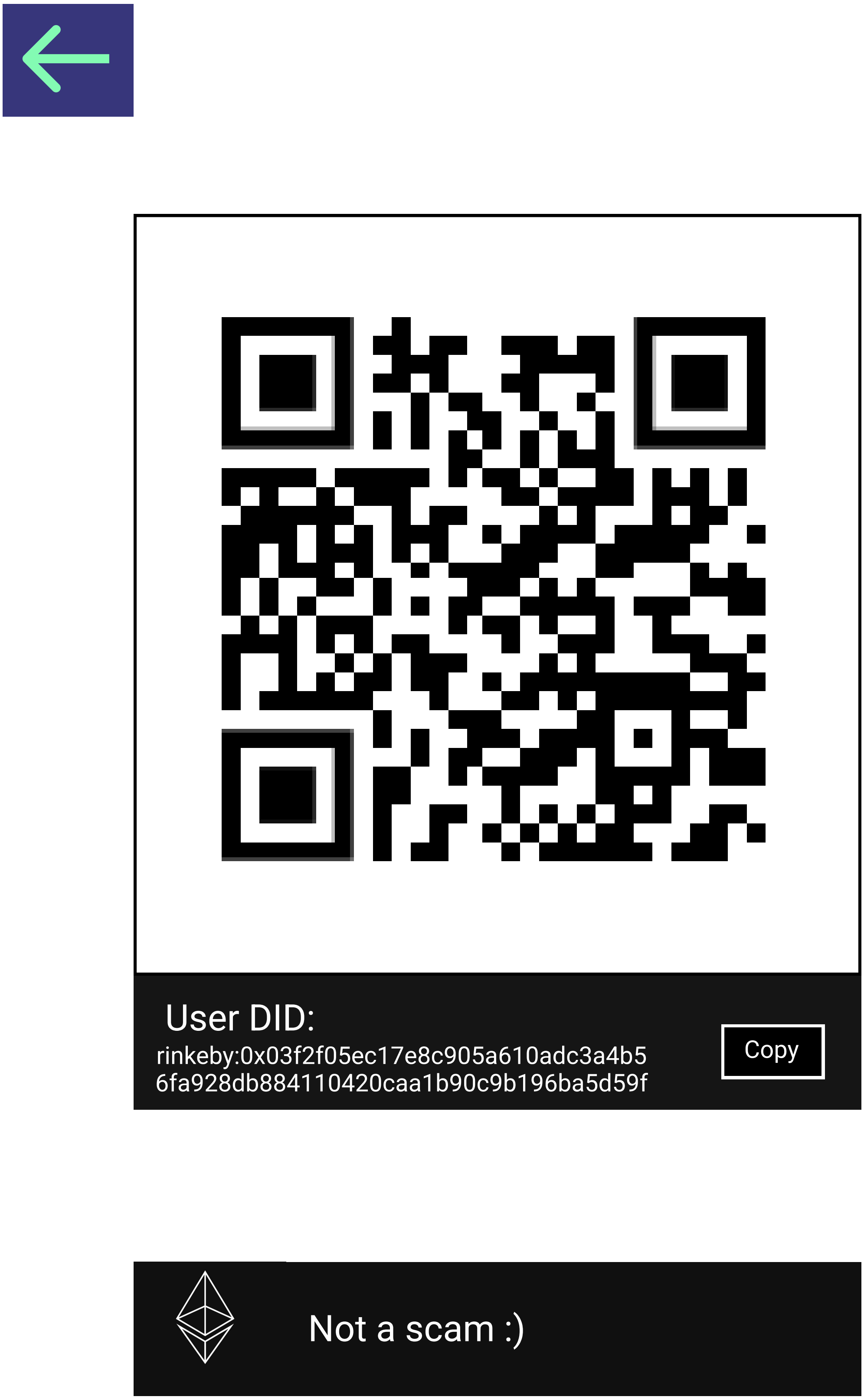
Gender

First name

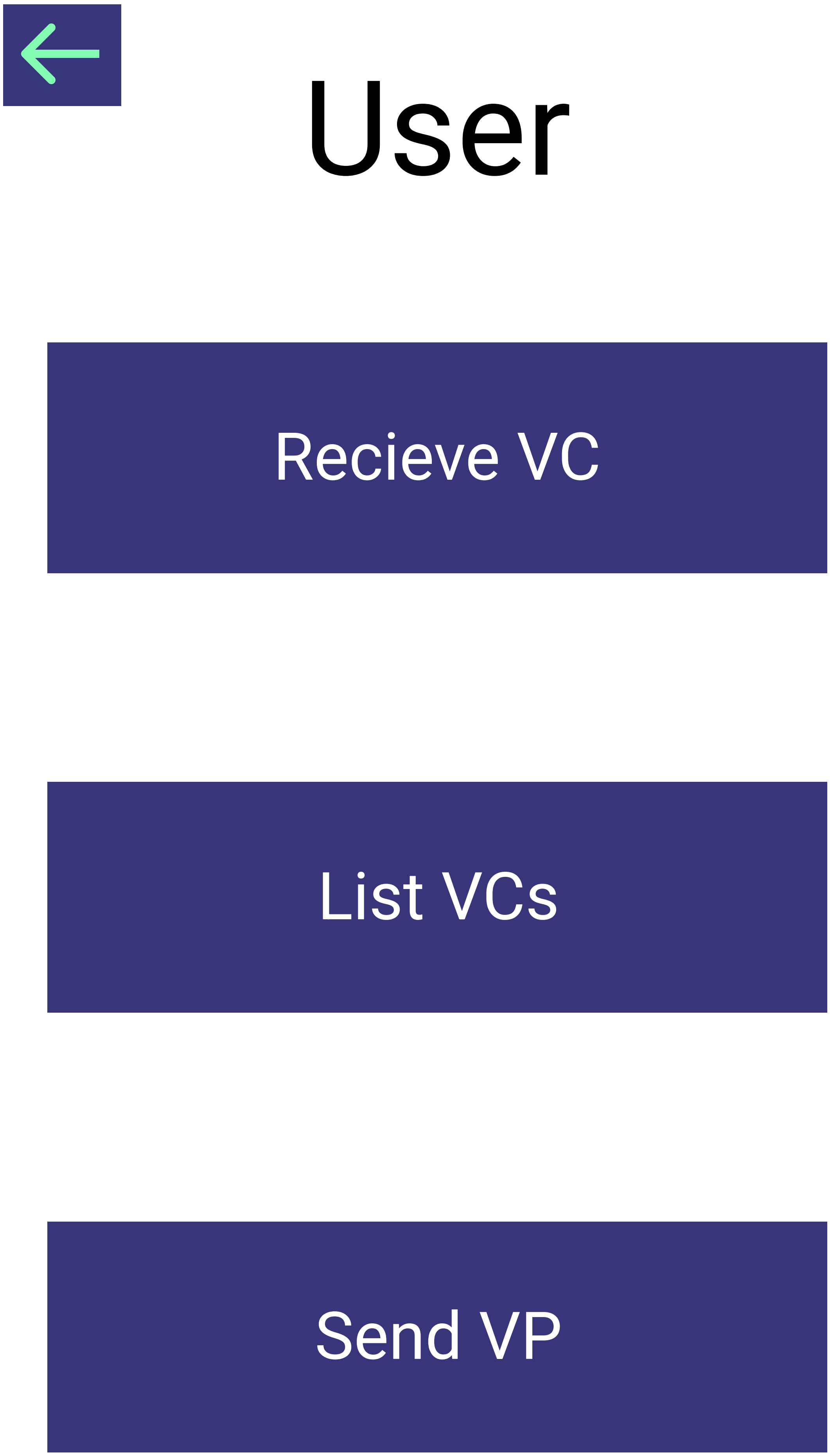
.....

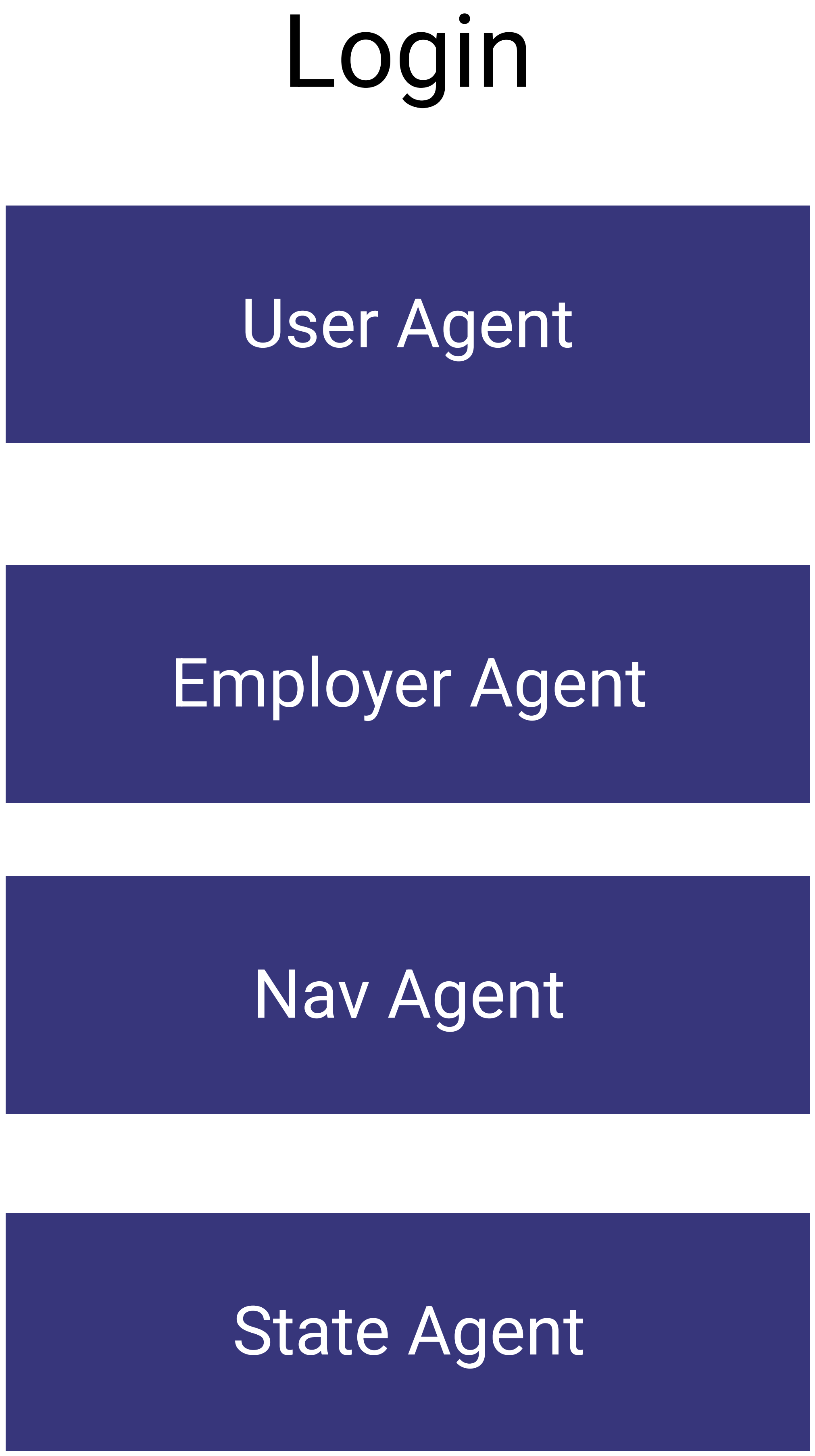


did:method:example

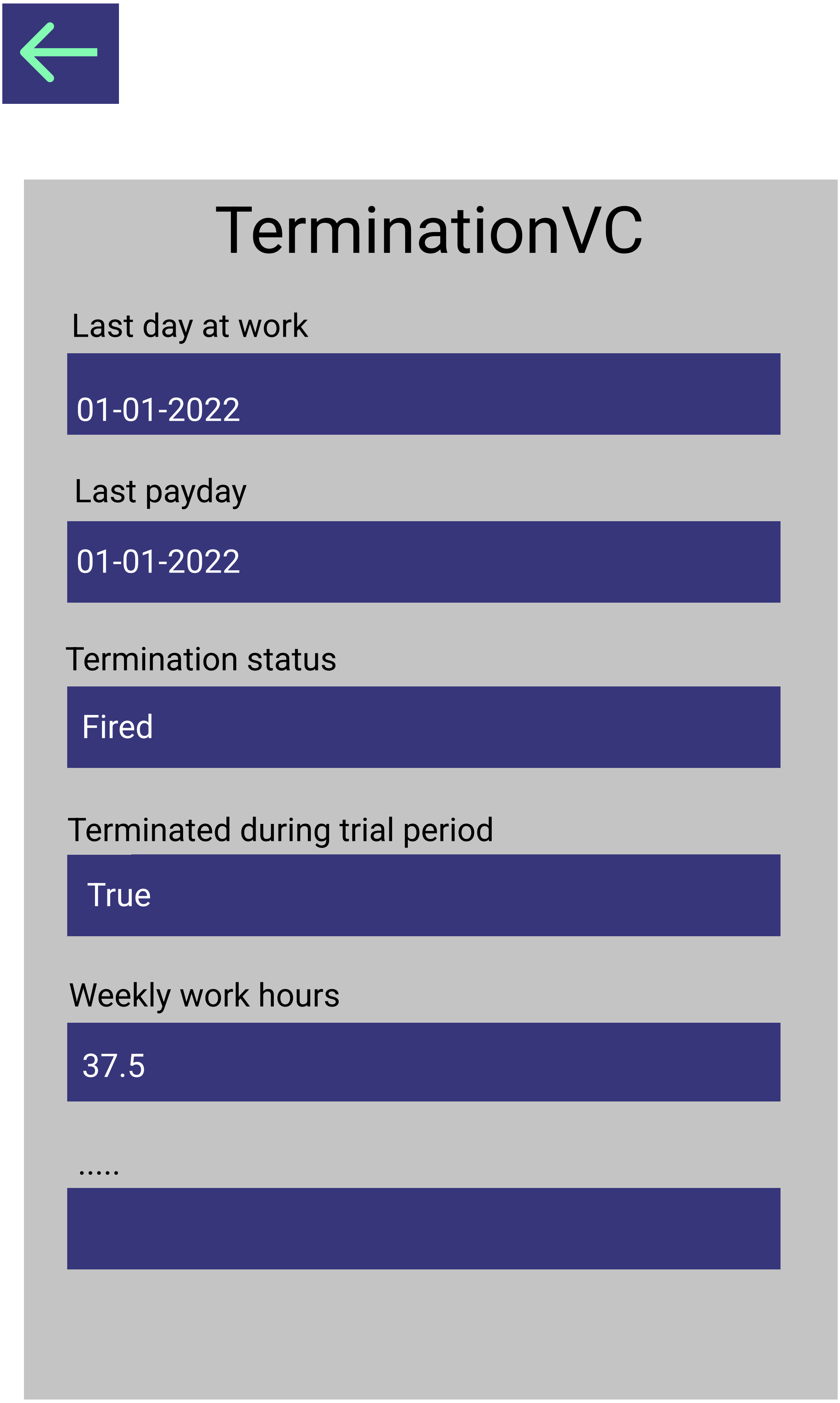


User DID:
rinkeby:0x03f2f05ec17e8c905a610adc3a4b5
6fa928db884110420caa1b90c9b196ba5d59f


User
Recieve VC
List VCs
Send VP



Login



TerminationVC

Last day at work

01-01-2022

Last payday

01-01-2022

Termination status

Fired

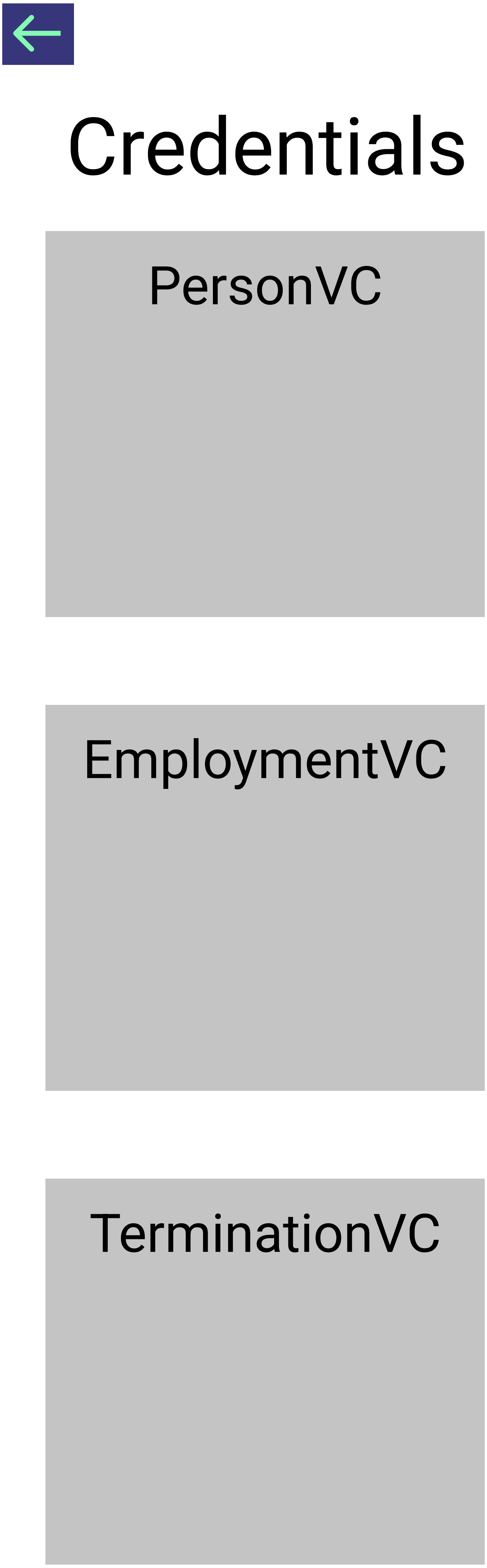
Terminated during trial period

True

Weekly work hours

37.5

.....



Credentials

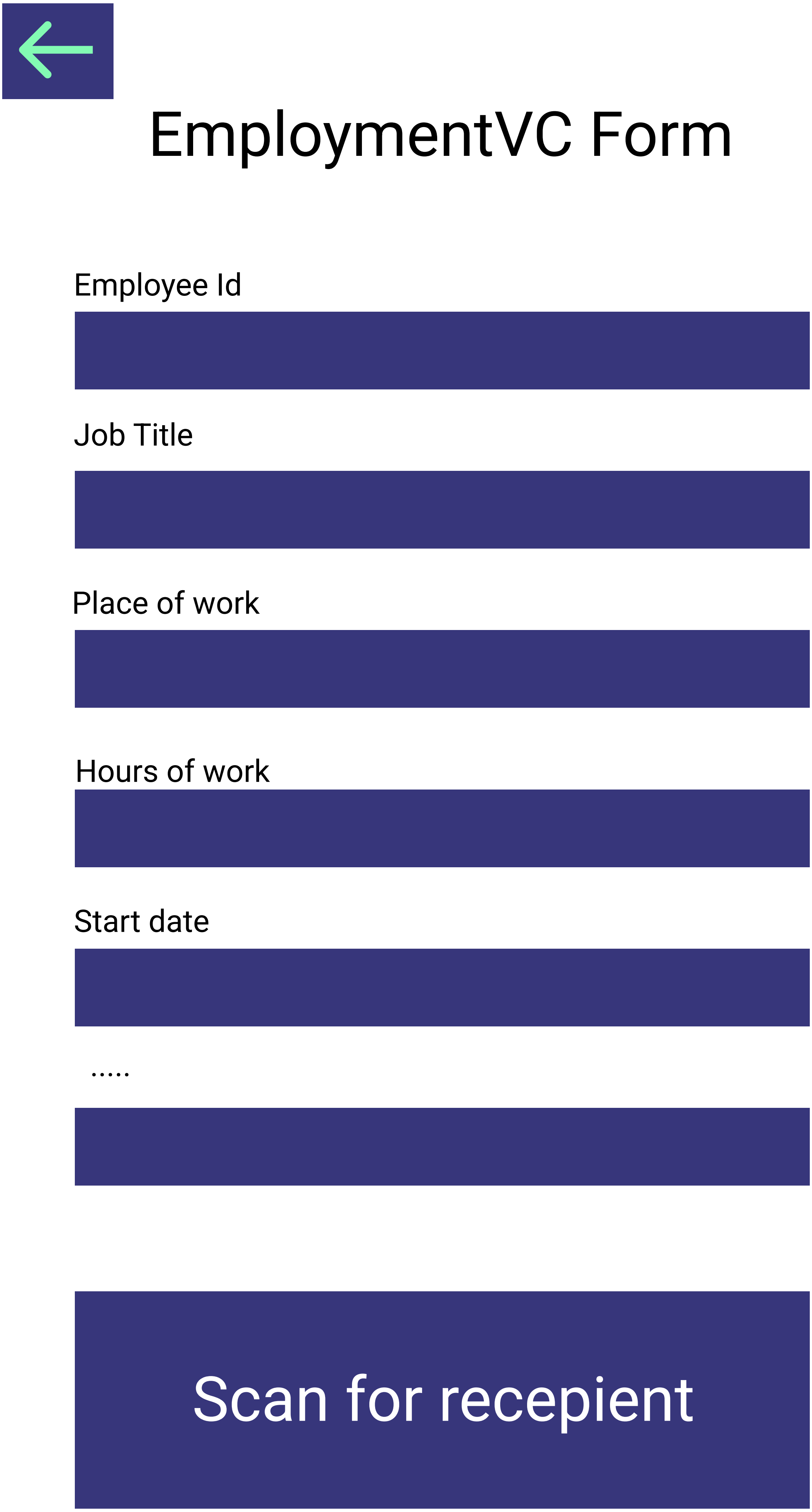
←

Employer

Create employment contract

Create termination contract

Create DID (?)



EmploymentVC Form

Employee Id

Job Title

Place of work

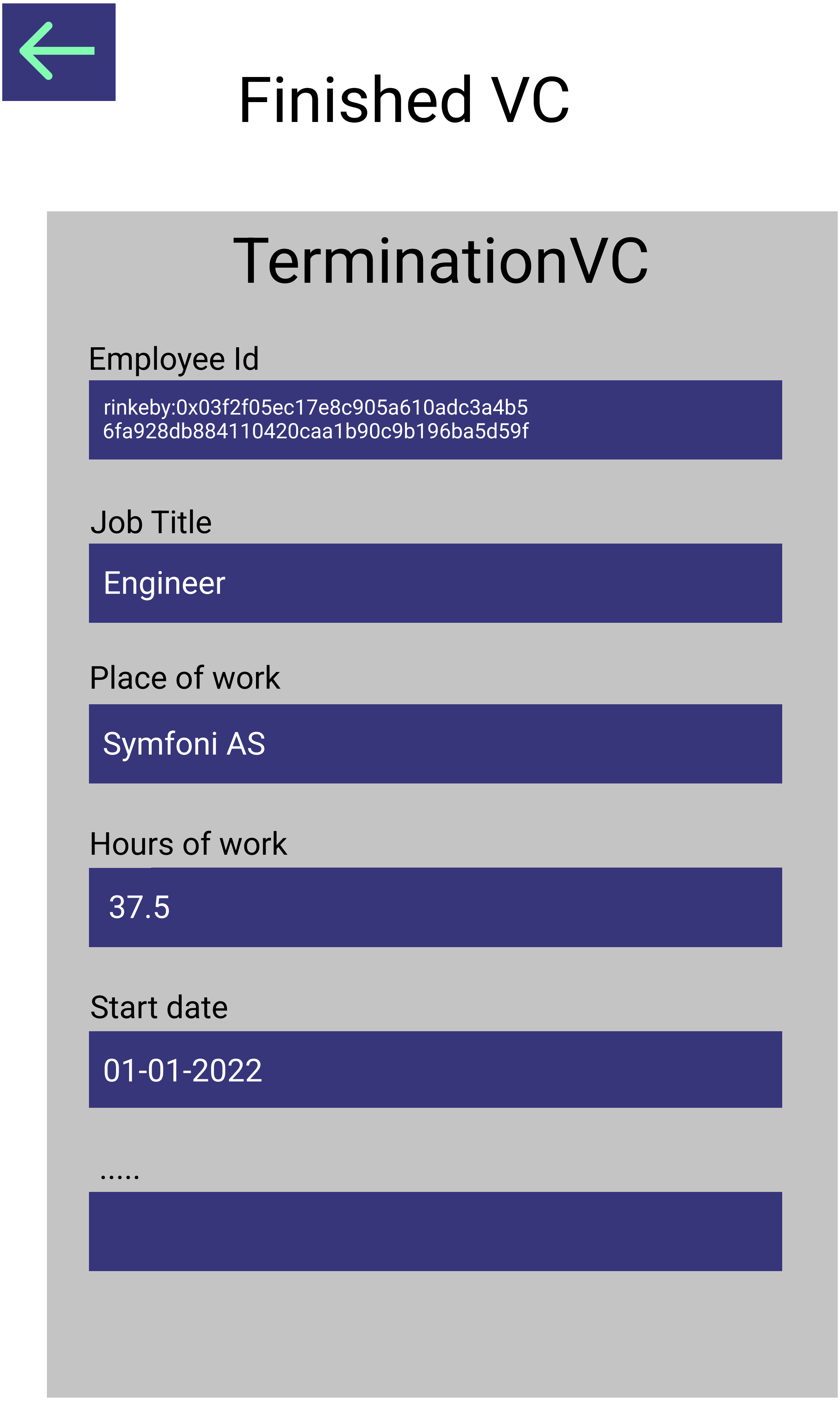
Hours of work

Start date

.....

←

did:method:example



Finished VC

TerminationVC

Employee Id

rinkeby:0x03f2f05ec17e8c905a610adc3a4b5
6fa928db884110420caa1b90c9b196ba5d59f

Job Title

Engineer

Place of work

Symfoni AS

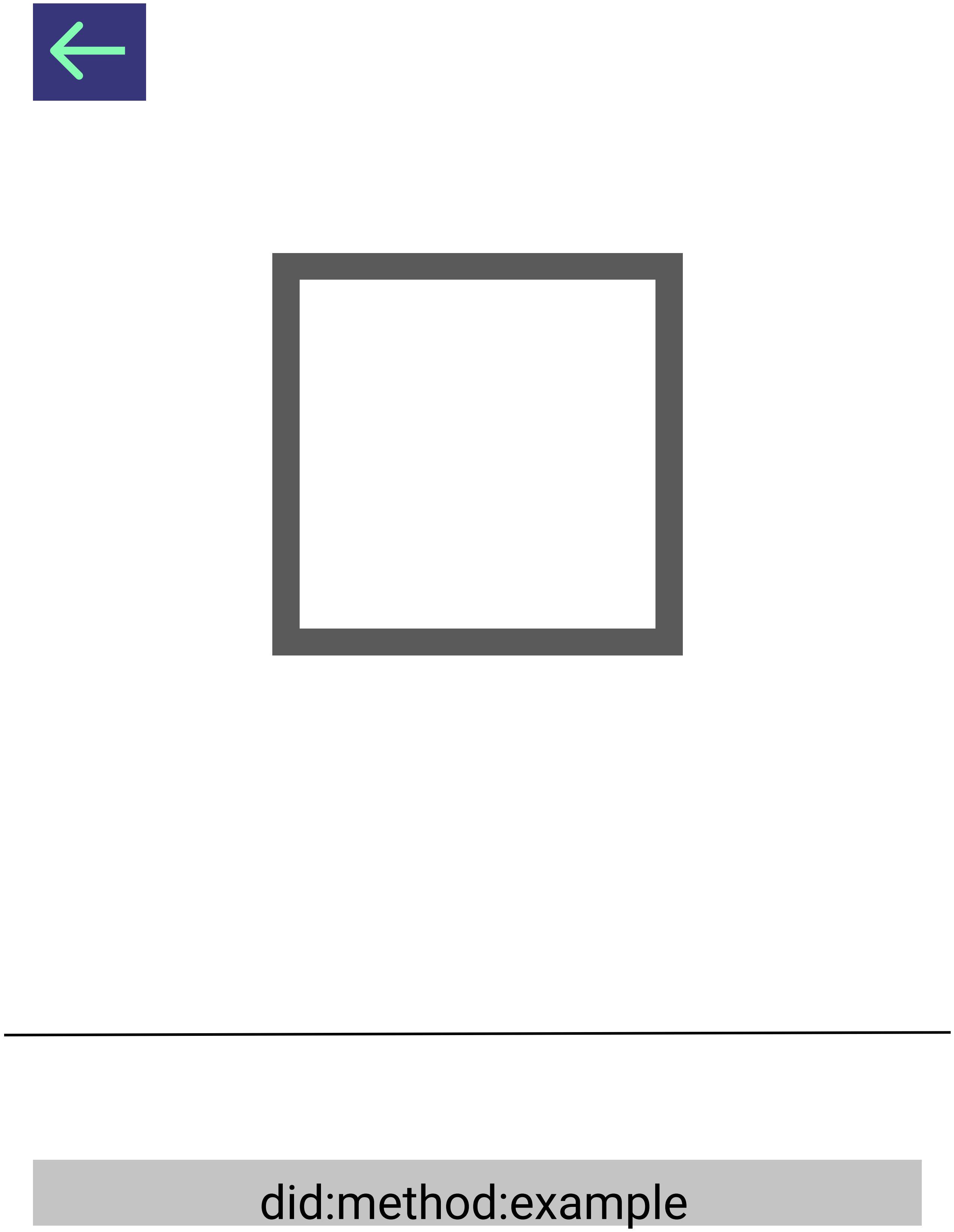
Hours of work

37.5

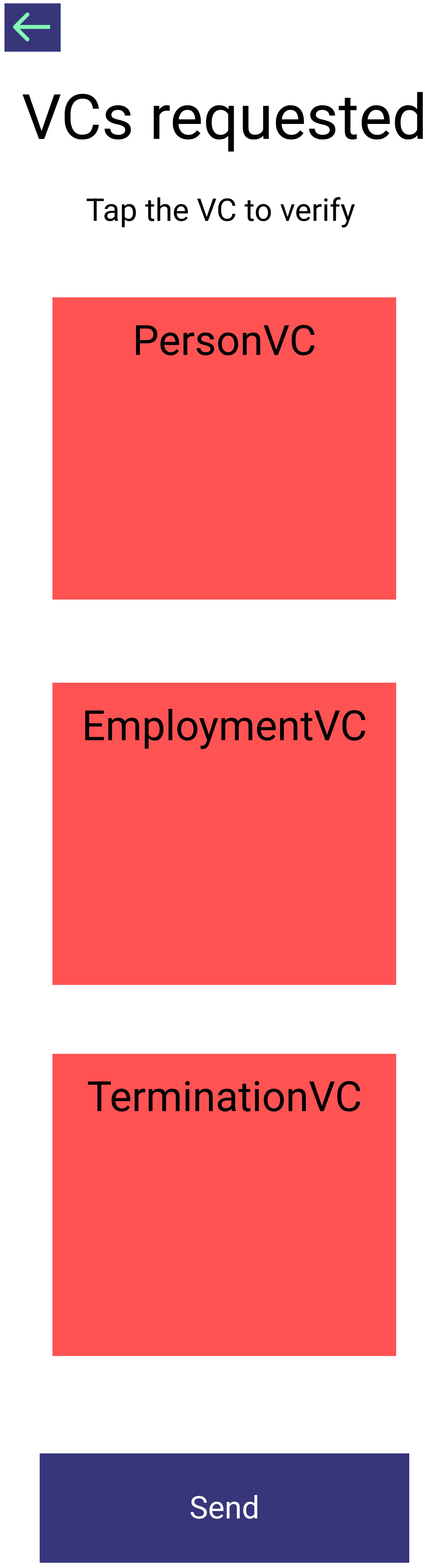
Start date

01-01-2022

.....

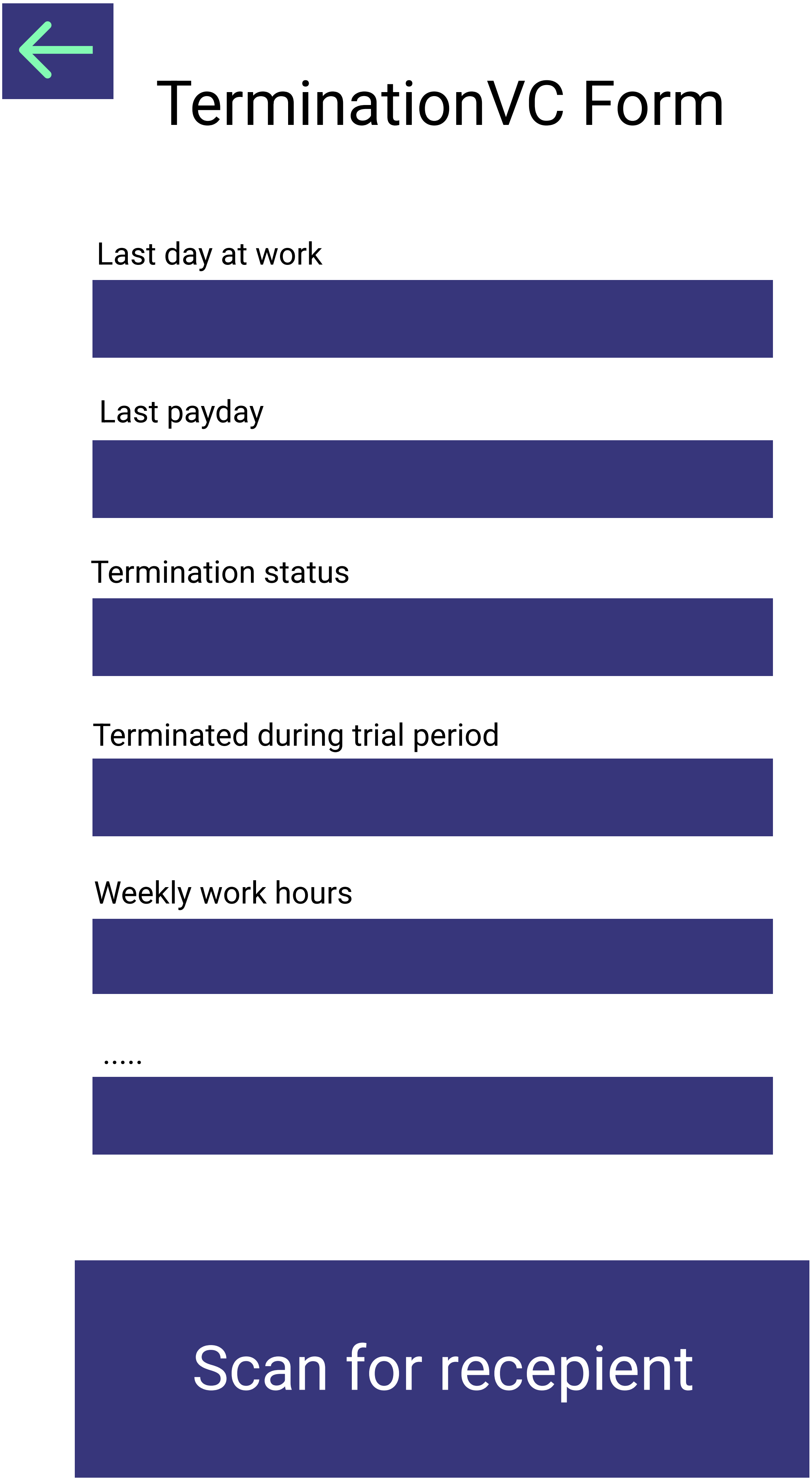


did:method:example



VCs requested

Tap the VC to verify



←

TerminationVC Form

Last day at work

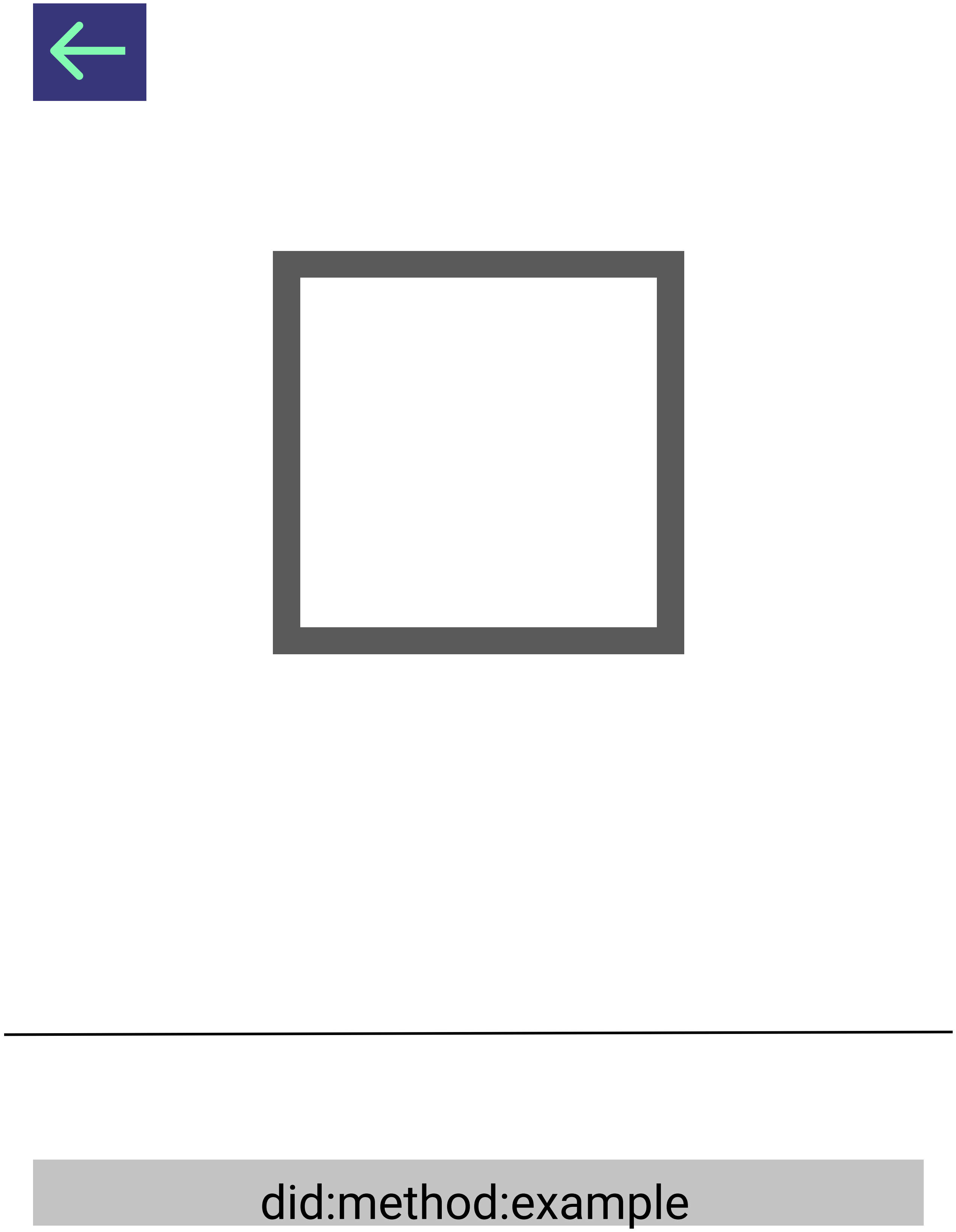
Last payday

Termination status

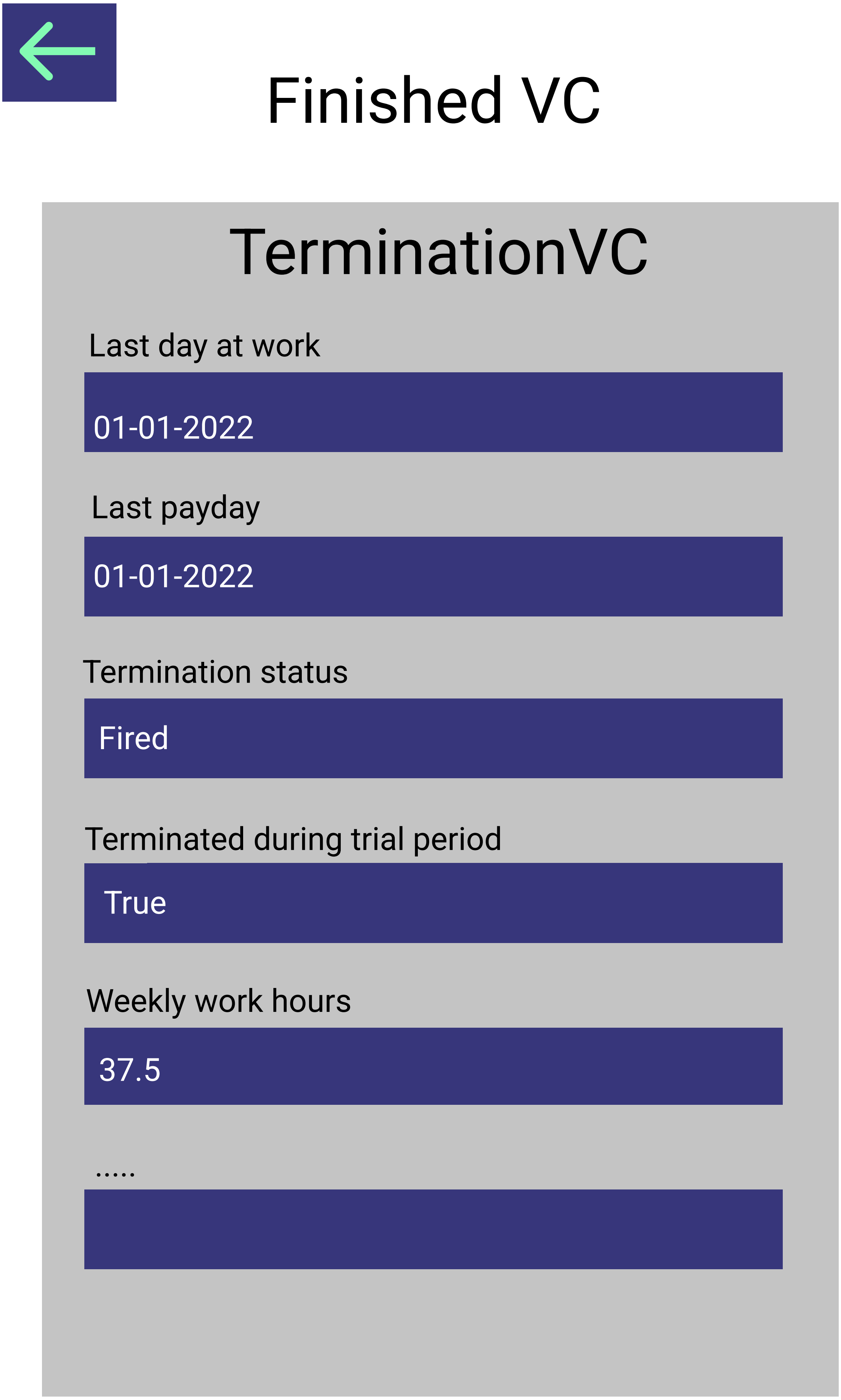
Terminated during trial period

Weekly work hours

.....



did:method:example



Finished VC

TerminationVC

Last day at work

01-01-2022

Last payday

01-01-2022

Termination status

Fired

Terminated during trial period

True

Weekly work hours

37.5

.....

←

NAV

Søk dagpenger

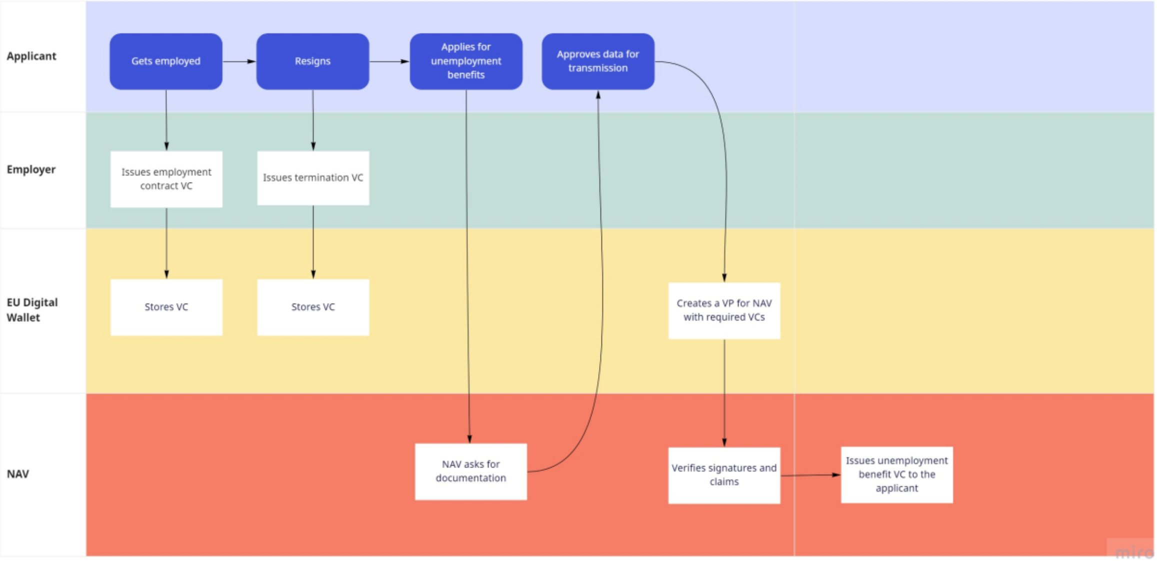
Søk dagpenger

Appendix H

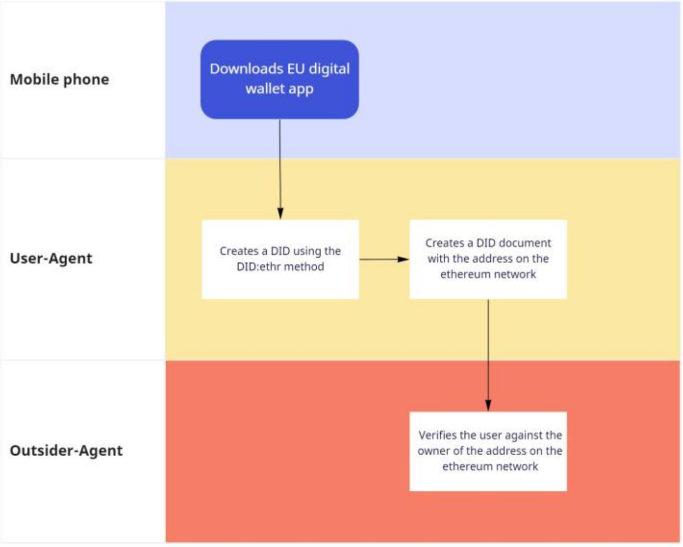
Swimlanes

This is a link to our Miro page where we have created most of our diagrams. The diagrams can be viewed in better quality inside the Miro page. `https://miro.com/app/board/uXjVOLsrucU=/?share_link_id=640869062584`

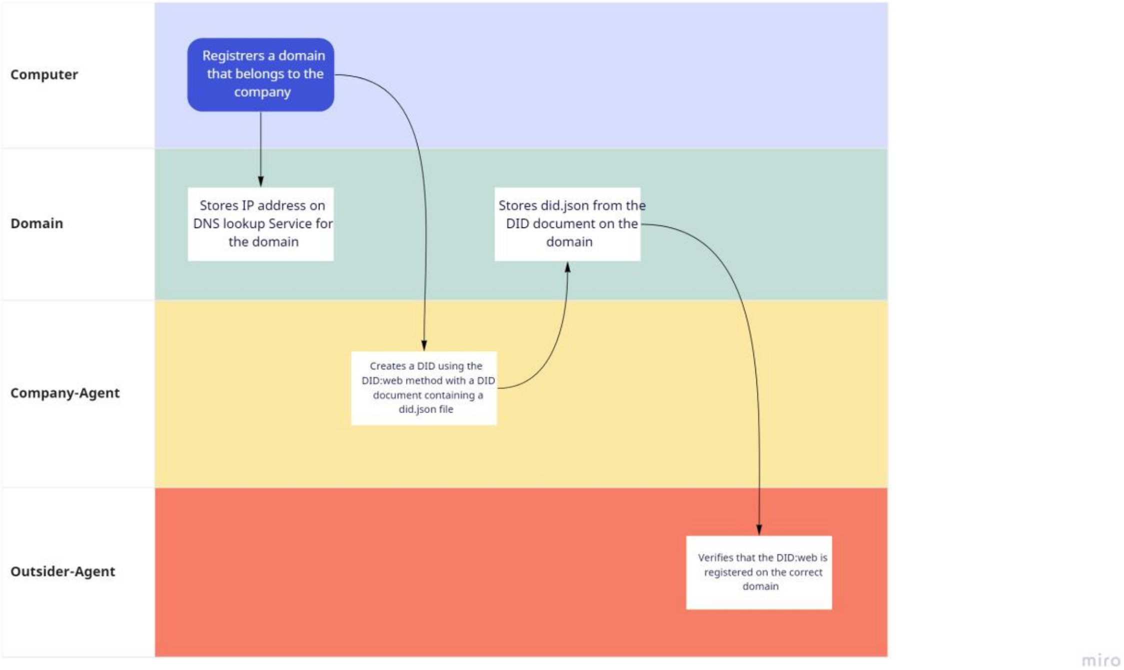
Business swim lane


Applicant
Employer
EU Digital Wallet
NAV
Gets employed
Resigns
Applies for unemployment benefits
Approves data for transmission
Issues employment contract VC
Issues termination VC
Stores VC
Stores VC
Creates a VP for NAV with required VCs
NAV asks for documentation
Verifies signatures and claims
Issues unemployment benefit VC to the applicant
miro

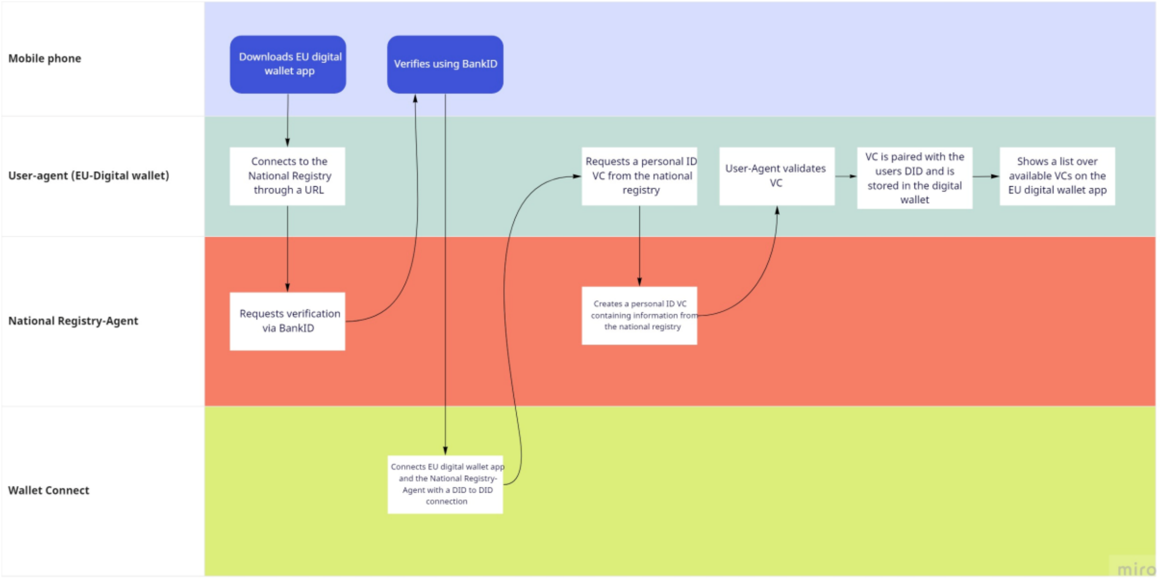
Person creates DID using 'did:ethr'


Mobile phone
User-Agent
Outsider-Agent
Downloads EU digital wallet app
Creates a DID using the DID:ethr method
Creates a DID document with the address on the ethereum network
Verifies the user against the owner of the address on the ethereum network

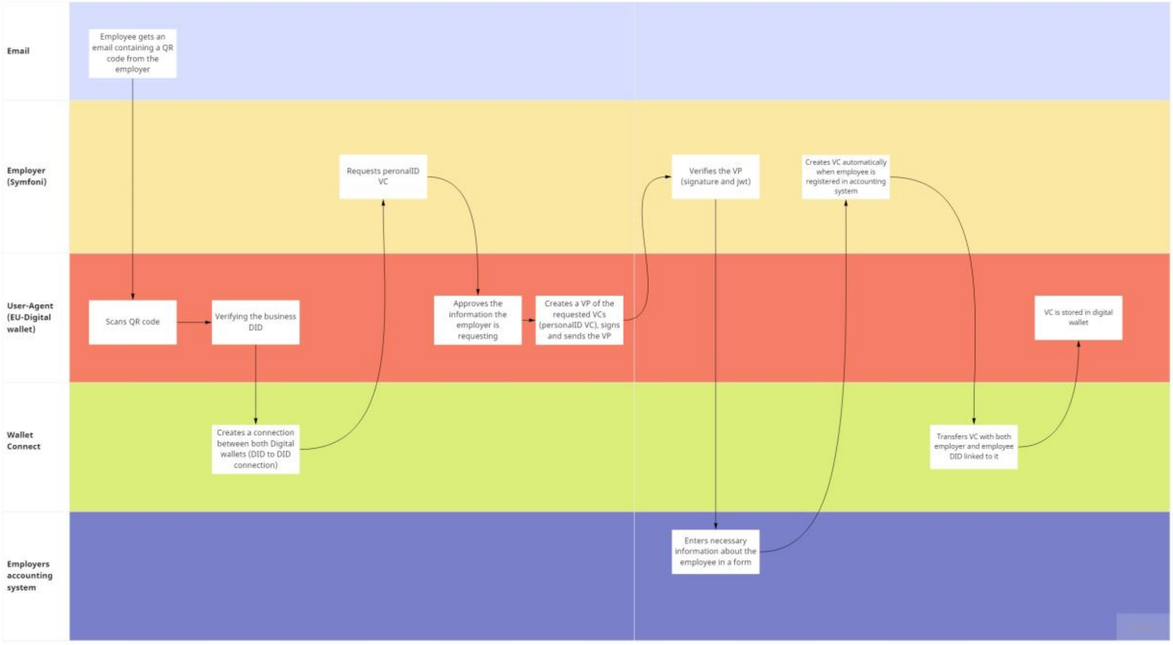
Company creates DID using 'did:web'


Computer
Registrers a domain that belongs to the company
Domain
Stores IP address on DNS lookup Service for the domain
Stores did.json from the DID document on the domain
Company-Agent
Creates a DID using the DID:web method with a DID document containing a did.json file
Outsider-Agent
Verifies that the DID:web is registered on the correct domain
miro

Person acquires personal ID VC


Mobile phone
Downloads EU digital wallet app
Verifies using BankID
User-agent (EU-Digital wallet)
Connects to the National Registry through a URL
Requests a personal ID VC from the national registry
User-Agent validates VC
VC is paired with the users DID and is stored in the digital wallet
Shows a list over available VCs on the EU digital wallet app
National Registry-Agent
Requests verification via BankID
Creates a personal ID VC containing information from the national registry
Wallet Connect
Connects EU digital wallet app and the National Registry-Agent with a DID to DID connection
miro

Person acquires employment contract VC

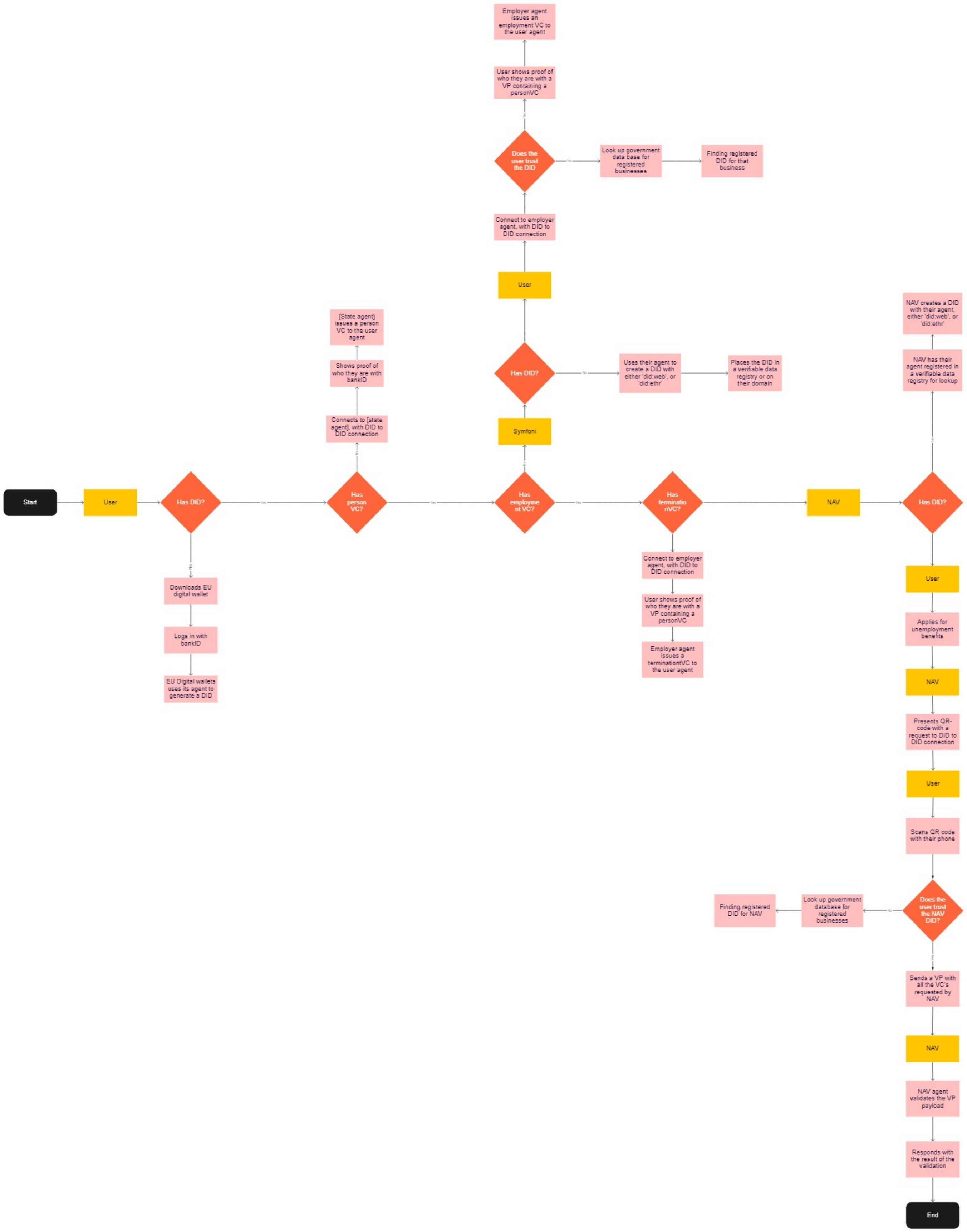


Email
Employer (Symfoni)
User-Agent (EU-Digital wallet)
Wallet Connect
Employers accounting system
Employee gets an email containing a QR code from the employer
Scans QR code
Verifying the business DID
Creates a connection between both Digital wallets (DID to DID connection)
Requests peronalID VC
Approves the information the employer is requesting
Creates a VP of the requested VCs (personalID VC), signs and sends the VP
Verifies the VP (signature and jwt)
Enters necessary information about the employee in a form
Creates VC automatically when employee is registered in accounting system
Transfers VC with both employer and employee DID linked to it
VC is stored in digital wallet

Appendix I

Flowchart

This is a link to the Miro page where you can find the flowchart in higher quality `https://miro.com/app/board/uXjVOJOoMd0=/?share_link_id=947488918471`


Start
User
Has DID?
No
Downloads EU digital wallet
Logs in with bankID
EU Digital wallets uses its agent to generate a DID
Yes
Has person VC?
No
Connects to [state agent], with DID to DID connection
Shows proof of who they are with bankID
[State agent] issues a person VC to the user agent
Yes
Has employment VC?
No
Symfoni
Has DID?
No
Uses their agent to create a DID with either 'did:web', or 'did:ethr'
Places the DID in a verifiable data registry or on their domain
User
Connect to employer agent, with DID to DID connection
Does the user trust the DID
No
Look up government data base for registered businesses
Finding registered DID for that business
Yes
User shows proof of who they are with a VP containing a personVC
Employer agent issues an employment VC to the user agent
Yes
Has terminationVC?
Connect to employer agent, with DID to DID connection
User shows proof of who they are with a VP containing a personVC
Employer agent issues a terminationtVC to the user agent
NAV
Has DID?
No
NAV has their agent registered in a verifiable data registry for lookup
NAV creates a DID with their agent, either 'did:web', or 'did:ethr'
User
Applies for unemployment benefits
NAV
Presents QR-code with a request to DID to DID connection
User
Scans QR code with their phone
Does the user trust the NAV DID?
No
Look up government database for registered businesses
Finding registered DID for NAV
Yes
Sends a VP with all the VC's requested by NAV
NAV
NAV agent validates the VP payload
Responds with the result of the validation
End
miro

Appendix J

User agent setup

Code listing J.1: Example of user agent setup

```
export const agentUser = createAgent<IDIDManager & IKeyManager & IDataStore &
    IDataStoreORM & IResolver & ICredentialIssuer>({
        plugins: [
                new DIDComm(),
                new KeyManager({
                        store: new KeyStore(dbConnectionUser),
                        kms: {
                                local: new KeyManagementSystem(new PrivateKeyStore(
                                    dbConnectionUser, new SecretBox(USER_KEY))),
                        },
                }),
                new DIDManager({
                        store: new DIDStore(dbConnectionUser),
                        defaultProvider: 'did:ethr:rinkeby',
                        providers: {
                                'did:ethr:rinkeby': new EthrDIDProvider({
                                        defaultKms: 'local',
                                        network: 'rinkeby',
                                        rpcUrl: 'https://rinkeby.infura.io/v3/' +
                                            INFURA_ID,
                                }),
                                'did:web': new WebDIDProvider({
                                        defaultKms: 'local',
                                }),
                        },
                }),
                new DIDResolverPlugin({
                        resolver: new Resolver({
                                ...ethrDidResolver({ infuraProjectId: INFURA_ID }),
                                ...webDidResolver(),
                        }),
                }),
                new CredentialIssuer(),
                new MessageHandler({
                        messageHandlers: [new JwtMessageHandler(), new
                            W3cMessageHandler(), new DIDCommMessageHandler()],
                }),
                new DataStore(dbConnectionUser),
                new DataStoreORM(dbConnectionUser)
        ],
});
```

Appendix K

isQualifiedForUnemploymentBenefits function

Code listing K.1: isQualifiedForUnemploymentBenefits function from NAVAgentController.ts

```
/**
         * isQualifiedForUnemploymentBenefits takes a presentation token and checks
              if the VCs within the presenation
         * token qualifies for unemployment benefits.
         * @param presentationToken a presentation token containing multiple VCs.
         * @returns a boolean stating whether the token qualifies or not.
         */
        async isQualifiedForUnemploymentBenefits(presentationToken: string):
             Promise<boolean | Error> {
                try {
                        let isVerifiedPersonVC = false;
                        let isVerifiedEmploymentVC = false;
                        let isVerifiedTerminationVC = false;

                        // handle message with token
                        const handledMessage: any = await this.agent.handleMessage
                             ({
                                raw: presentationToken
                        });

                        // get array of credential tokens
                        const credentials = handledMessage.credentials;

                        // handle each credential token
                        for (let index = 0; index < credentials.length; index++) {

                                // get credential message by handling credential
                                     token
                                const credentialMessage: any = await this.agent.
                                     handleMessage({
                                        raw: credentials[index].proof.jwt
                                });

                                // get credential message data
                                const credentialMessageData: any =
                                     credentialMessage.data;

                                switch (credentialMessageData.vc.type.at(1)) {
                                case TYPE_EMPLOYMENT_CREDENTIAL:
                                        if (
                                                ! await verifySchema(
                                                    credentialMessageData) ||
                                                ! await verifyIssuer(
                                                    credentialMessageData, issuers.
                                                    symfoni)
                                        ) {
                                                return false;
                                        }
                                        isVerifiedEmploymentVC = true;
                                        break;

                                case TYPE_TERMINATION_CREDENTIAL:
                                        if (
                                                ! await verifySchema(
                                                    credentialMessageData) ||
                                                ! await verifyIssuer(
                                                    credentialMessageData, issuers.
                                                    symfoni) ||
                                                ! this.verifyTerminationVCData(
                                                    credentialMessageData)
                                        ) {
                                                return false;
                                        }
                                        isVerifiedTerminationVC = true;
                                        break;

                                case TYPE_PERSON_CREDENTIAL:
                                        if (
                                                ! await verifySchema(
                                                    credentialMessageData) ||
                                                ! await verifyIssuer(
                                                    credentialMessageData, issuers.
                                                    state) ||
                                                ! this.verifyPersonVCData(
                                                    credentialMessageData)
                                        ) {
                                                return false;
                                        }
                                        isVerifiedPersonVC = true;
                                        break;

                                default:
                                        break;
                                }
                        }

                        // if all necessary credential are verified, return true.
                        if (isVerifiedEmploymentVC && isVerifiedTerminationVC &&
                             isVerifiedPersonVC) {
                                return true;
                        }

                        return false;

                } catch (error) {
                        console.error(error);
                        return new Error('unable␣to␣qualify');
                }
        }
```

Appendix L

Original Problem Definition

Sammenhengende tjenester: Bedre innbyggertjenester med Verifiable Credentials — Digitalisering av «Søknad om dagpenger»

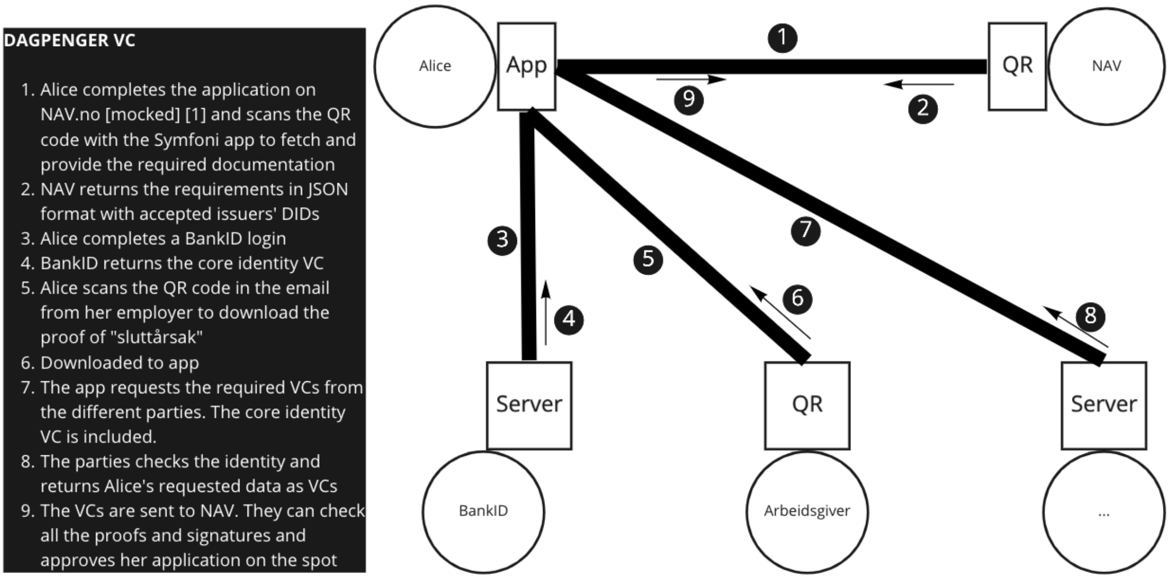
Skal man søke dagpenger fra NAV trenger man dokumentasjon fra diverse utstedere. Se tabellen. Dette er manuelle prosesser i dag hvor søkeren fungerer som «postbud for staten». Med Verifiable Credentials (VC) kan hele søknaden automatiseres. Både for søker og for godkjenning hos NAV.

	Utsteder
Bekreftelse på sluttårsak/nedsatt arbeidstid (ikke permittert) NAV 04-08.03	Arbeidsgiver
Har du forsørgeransvar eller bidragsplikt for barn under 18 år, og barna har ikke bodd i Norge? Fødselsattest/bostedsbevis for barn under 18 år	Skatteetaten i barnets bostedsland
Har du forsørgeransvar for barn eller barn under 18 år som er bosatt i et annet EØS-land? SED U006 Familieinformasjon	trygdemyndighetene i barnets bostedsland
Har du nylig avtjent verneplikt/siviltjeneste? Tjenestebevis	Forsvaret/Vernepliktsverket
Er du eller har du vært skoleelev eller student de siste seks månedene? Dokumentasjon av sluttdato	Skolen/universitetet
Tar du utdanning eller går på kurs? Elevdokumentasjon fra lærested	Skolen/universitetet/annet lærested
Får du sluttkompensasjon fra arbeidsgiveren din? Kopi av sluttavtale	Arbeidsgiver
Har du jobbet i et annet EU/EØS-land? U1 Perioder av betydning for retten til dagpenger	Attesten er dokumentasjon på arbeids- og trygdeperioder i annet EØS-land, og utstedes av myndighetene i det landet der har arbeidet.
Har du vært til sjøs? Sjøfartsbok/hyreavregning	Sjøfartsdirektoratet i det respektive landet?

Det som skal lages i denne bachelor-oppgaven er

- En VC Issuer som kan utstede det nødvendige beviset. «Veramo» brukes som rammeverk for opprettelse av DID og VC
- En VC Verifier som kan godkjenne mottatte bevis. «Veramo» brukes også her
- En app som utveklser bevisene mellom Issuer og Verifier. Istedenfor å utvikle også denne modulen fra bunn, har Symfoni AS har utviklet en proof-of-concept av appen. Studentene tilpasser denne til å løse oppgaven

Videre spinner studentene opp servere for de forskjellige utstederne, og gjør det mulig å gjennomføre en full demo av caset.



DAGPENGER VC

1. Alice completes the application on NAV.no [mocked] [1] and scans the QR code with the Symfoni app to fetch and provide the required documentation
2. NAV returns the requirements in JSON format with accepted issuers' DIDs
3. Alice completes a BankID login
4. BankID returns the core identity VC
5. Alice scans the QR code in the email from her employer to download the proof of "sluttårsak"
6. Downloaded to app
7. The app requests the required VCs from the different parties. The core identity VC is included.
8. The parties checks the identity and returns Alice's requested data as VCs
9. The VCs are sent to NAV. They can check all the proofs and signatures and approves her application on the spot

1: https://www.nav.no/soknader/nb/person/arbeid/dagpenger/NAV%2004-01.03/brev#15

Appendix M

Timesheet

Date	Mikkel	Magnus	Erlend	Notes
10.01.2022	240	240	90	- Created document template for the main report - Made typescript run on out computers - Downloaded WebStorm - Made and shared a google drive folder - Created a GitHub repository, made branches, and branch rules
11.01.2022	75	75	75	- Prepeared questions for the meeting with employer - Read the contract - Looked at earlier bachelor thesis' - Created important documents in google drive
12.01.2022	90	90	90	- Meeting with Symfoni - Discussed the meeting afterwards
13.01.2022	240	240	195	- Read about SSI - Wrote on the project plan
14.01.2022	300	300	300	- Organized weekly meetings with adviser - Worked on the project plan - Meeting with our advisor, Mariusz (60 min)
17.01.2022	240	240	240	- Prepared questions for meeting with symfoni - Filled out the contract - Wrote on the project plan
18.01.2022	240	240	240	- Started a Codecademy course on TypeScript - Meeting with Symfoni
19.01.2022	240	240	240	-Typescript Codecademy -Worked on project plan
20.01.2022	240	240	240	- Wrote on the project plan - Read the Veramo documentation
24.01.2022	210	210	210	- Read about SSI and discussed what we read
25.01.2022	180	180	180	- Made a model for our issuer, holder, and verifier scenario
27.01.2022	240	240	240	- Did the Veramo Cli Tutorial - Setup WSL - Revised the project plan - Creating async meeting document for Mariusz - Everyone got ESlint and Mocha working
31.01.2022	300	300	300	- Worked on setting up servers - Discussed the environmental effects of SSI
03.02.2022	240	240	240	- Started working on a presentation for NAV - Tried making a couple of VCs with the veramo client

04.02.2022	240	240	240	- Akademisk tekst (placeholder)
10.02.2022	360	360	360	- Meeting with symfoni and our advisour - Added different Agents for user, NAV and Symfoni
11.02.2022	240	240	240	- Revicing the VC to add more detailed meta data for previous employmet contract
14.02.2022	120	120	120	- Møter med symfoni
15.02.2022	120	120	120	- Made first attemt at a schema for a work contract
16.02.2022	180	180	180	- Fixing the schemas made yeasterday, and digging into meta data for did:web document
17.02.2022	150	150	150	- Added and troubleshooted a PDF hasher
18.02.2022	300	300	300	- Meeting with NAV - Revised VC's and made some new ones - Made a swim lane diagram for user interaction with the system
21.02.2022	120	120	120	-Further revised VC's -Meeting with Symfoni
23.02.2022	300	300	300	- Updated the technical swim lane diagram - Did some minor coding work
25.02.2022	0	0	90	- Individual work
28.02.2022	240	240	240	
02.03.2022	120	120	120	-rapport skrive kurs
03.03.2022	360	360	240	-Made a agent controller class that the other controllers will inherit
09.03.2022	360	360	360	- Working on JSON-LD
10.03.2022	360	360	360	- working on unit tests -fixing a default did for all agents
14.03.2022	300	300	300	- worked on API endpoints -meeting with symfoni
15.03.2022		120		- failed to make dapps
16.03.2022	360	360	360	- Meeting with NAV - Creating wireframe for app - revising some of the swimlane documents
17.03.2022	300	300	300	- further work on wireframe/figma
18.03.2022	120	120	0	
21.03.2022	90	90	90	-technical meeting with Asbjørn
24.03.2022	300	300	300	-working on front-end react native

25.03.2022	360	360	240	-working on front-end react native
28.03.2022	420	420	420	- worked on GUI forms - worked on GUI credential list view - technical meeting with Asbjørn
29.03.2022	420	420	420	- worked on view credential details and employment form - troubleshoot expo and android emulator
30.03.2022	360	360	360	
31.03.2022	360	360	360	- worked on walletconnect, styling, state agent and submitting forms to a database
01.04.2022	300	300	300	- Started on databases - implemented json validation
02.04.2022	360	360	360	- Updated schema - Made database and database endpoint for the symfoni agent - short technical meeting about wallet connect
04.04.2022	330	330	330	- Technical meeting with symfoni - Refactored schema
05.04.2022	240	0	0	- wrote unit tests
06.04.2022	360	360	360	- Started work on didcomm - Form to database and state database
07.04.2022	180	240		- Technical meeting with asbjørn, working on DIDComm - Adding gas to ethr did
08.04.2022	420	420	420	- Meeting with NAV - Pair programming setting up endpoints for DIDComm
11.04.2022	300	0	0	- worked on the nav recieve VP page
12.04.2022	300	0	0	- worked on the nav verify VP for unemployment benefit functionality
13.04.2022	300	0	0	- continued working on nav VP verification functionality - started working on user messaging system
14.04.2022	360	0	0	- continued working on the user messaging system - started working on symfoni VP verification functionality
15.04.2022	300	0	0	- finished symfoni recieve VP and issue VC
18.04.2022	420	420	420	- documentation, proofreading, etc.
19.04.2022	420	420	420	- documentation, proofreading, etc.
20.04.2022	420	420	420	- documentation, proofreading, etc.
21.04.2022	420	420	420	- documentation, proofreading, etc.
22.04.2022	420	420	420	- documentation, proofreading, etc.

25.04.2022	420	420	420	- documentation, proofreading, etc.
26.04.2022	420	420	420	- documentation, proofreading, etc.
27.04.2022	420	420	420	- documentation, proofreading, etc.
28.04.2022	420	420	420	- documentation, proofreading, etc.
29.04.2022	420	420	420	- documentation, proofreading, etc.
02.05.2022	420	420	420	- documentation, proofreading, etc.
03.05.2022	420	420	420	- documentation, proofreading, etc.
04.05.2022	420	420	420	- documentation, proofreading, etc.
05.05.2022	420	420	420	- documentation, proofreading, etc.
06.05.2022	420	420	420	- documentation, proofreading, etc.
09.05.2022	420	420	420	- documentation, proofreading, etc.
10.05.2022	300	300	300	- documentation, proofreading, etc.
11.05.2022	420	420	420	- documentation, proofreading, etc.
12.05.2022	420	420	420	- documentation, proofreading, etc.
13.05.2022	360	360	360	- documentation, proofreading, etc.
15.05.2022	420	420	420	- documentation, proofreading, etc.
16.05.2022	420	420	420	- documentation, proofreading, etc.
18.05.2022	420	420	420	- documentation, proofreading, etc.
19.05.2022	720	720	720	- documentation, proofreading, etc.

Aas, Gluppe, Vannebo

Self-Sovereign Identity and Verifiable Data

NTNU
Kunnskap for en bedre verden