University of Zurich^UZH

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of Systems Interfaces for a Mixnet-based Voting System

Raphael Wäspi
St. Gallen, Switzerland
Student ID: 18-918-938

Supervisor: Christian Killer, Jan von der Assen
Date of Submission: August 25, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

It is of central importance for a democracy that all citizens entitled to vote have the opportunity to cast their ballots in votes and elections. The most common way to vote in Switzerland is to hand in a completed ballot directly at the urn or by postal services. However, an important part of Switzerland's e-government strategy is to enable voting and elections electronically. In this context, the Federal Council emphasizes that e-voting should not only be made available to the Swiss abroad, but to all citizens who are entitled to vote. For this reason, a new legal basis has come into force on 25.05.2022, which allows the cantons to offer e-voting as part of a trial operation. Numerous challenges must be overcome in the development of the digitalization of voting and elections. While there is already a large amount of scientific literature on this topic, most of it takes a centralized approach. The Provotum project, on the other hand, takes a different approach. Due to the use of distributed ledger technology, this electronic voting and election system is decentralized. This has desirable consequences as it allows the system to ensure transparency, integrity and robustness. In the latest version of Provotum, established cryptographic techniques have been included in the project, resulting in the enabling of elections and the shifting of computational effort to the election infrastructure. However, this version does not yet use a graphical user interface.

The thesis implements graphical user interfaces for vote casting and for vote administration. This means that relevant literature on user interfaces and user experiences in other electronic voting systems will also be examined, and the Provotum 3.0 Mixnet prototype will also be analyzed. Based on this analysis, three user interfaces will be created so that each stakeholder in Provotum can complete their tasks over the Internet. In addition, it creates a simple API that enables communication between the blockchain and the graphical user interfaces. These software components are implemented using modern technologies. Finally, user interfaces are evaluated using heuristics, use cases, a discussion and system usability scales.

Zusammenfassung

Für eine Demokratie ist es von zentraler Bedeutung, dass alle stimmberechtigten Bürgerinnen und Bürger die Möglichkeit haben, ihre Stimme bei Volksabstimmungen und Wahlen abzugeben. Die gängigste Art abzustimmen und zu wählen ist heute in der Schweiz die Abgabe des ausgefüllten Stimm- und Wahlzettels direkt an der Urne oder in brieflicher Form. Ein wichtiger Teil der E-Government-Strategie der Schweiz ist jedoch die Stimm- und Wahlabgabe elektronisch möglich zu machen. Der Bundesrat betont in diesen Zusammenhang, dass das E-Voting nicht nur den Auslandschweizerinnen und Auslandschweizer, sondern allen stimmberechtigten Bürgerinnen und Bürgern zur Verfügung gestellt werden soll. Aus diesem Grund trat am 25.05.2022 eine neue rechtliche Grundlage in Kraft, welche es Kantonen erlaubt im Rahmen des Versuchsbetriebes E-Voting anzubieten. Bei der Entwicklung der Digitalisierung von Abstimmungen und Wahlen müssen zahlreiche Herausforderungen überwunden werden. Zwar existieren bereits zahlreiche wissenschaftliche Literatur zu dieser Thematik, jedoch verfolgen die meistens einen zentralisierten Ansatz. Das Projekt Provotum hingegen verfolgt einen anderen Ansatz. Durch den Einsatz der Distributed-Ledger-Technologie ist das elektronische Stimm- und Wahlsystem dezentralisiert. Dies hat wünschenswerte Konsequenzen zur Folge, da dadurch das System Transparenz, Integrität und Robustheit garantieren kann. In der neuesten Version von Provotum wurde das Projekt um bewährte kryptografische Techniken erweitert, was zur Ermöglichung von Wahlen führte und die Rechenlast auf die Wahlinfrastruktur verlagerte. Allerdings verwendet diese Version noch immer keine grafische Benutzeroberfläche.

Die Bachelorarbeit implementiert graphische Benutzeroberflächen für Stimmabgaben und für die Verwaltung von Abstimmungen und Wahlen. Das bedeutet, dass auch die entsprechende Literatur über User Interface und User Experience in anderen elektronischen Wahlsystemen untersucht wird, sowie auch der Prototyp von Provotum 3.0 Mixnet analysiert wird. Basierend auf diese Analyse werden drei Benutzeroberflächen erstellt, damit jede Anspruchsgruppen in Provotum die Aufgaben über das Internet erledigen kann. Auf der Grundlage dieser Analyse werden drei Benutzeroberflächen entwickelt, damit jede Anspruchsgruppe in Provotum die Aufgaben über das Internet erledigen kann. Darüber hinaus wird eine schlichte API erstellt, die die Kommunikation zwischen der Blockchain und den Benutzeroberflächen ermöglicht. Diese Softwarekomponenten werden mit modernen Technologien implementiert. Schlussendlich werden die Benutzeroberflächen anhand von Heuristiken, Benutzungsszenarien, einer Diskussion und System Usability Scales evaluiert.

Acknowledgments

I would like to thank all of the people who took the time to complete the surveys or who supported me in any way in the writing of this paper. First and foremost, I want to thank Christian Killer, for his valuable input and for guiding me in the right direction. He always took time whenever I needed it and provided answers to every question I had. Furthermore, I would like to thank Prof. Dr. Burkhard Stiller for offering me the opportunity to write my bachelor thesis at Communication Systems Group.

Contents

Chapter 1

Introduction

In modern democracies e-voting is becoming more and more important. Therefore, the Swiss e-government includes e-voting as a part of their strategic plan [1]. This means that the federal government wants to give citizens the opportunity to participate in politics electronically, which includes Remote Electronic Voting (REV) systems. REV systems allow citizens to fill out ballots remotely in an uncontrolled environment, such as a web browser or cell phone [2].

In Switzerland, ten cantons have introduced an e-voting system by 2019. These cantons could choose between two e-voting systems: the system of the canton of Geneva and the one of the Swiss Post. In November 2018, the Canton of Geneva announced that they would no longer operate the system. In addition, the Swiss Post discovered errors in their source code that led to a temporary suspension of the e-voting system [3]. At the Federal Council meeting of April 28, 2021, it was decided to open the consultation procedure for the realignment of the trial operation of e-voting [4].

The Communication Systems Research Group (CSG) at the Department of Informatics of the University of Zurich is conducting a Provotum project that has been implemented in multiple research publications and projects [5]–[7]. Provotum is a blockchain-based REV system developed as a proof-of-concept. The project started in 2018 with Provotum 1.0, which had at first a central architecture but it already used the Distributed Ledger (DL) technology for storage [8]. Provotum 2.0 was the first fully decentralized version of the Provotum project, which also reduced the limitations for real world use. In 2020 Provotum 3.0 [9] was introduced, which replaced the DL technology with substrate to remedy technical limitations and improve the scalability. A further version of Provotum 3.0 with the addition of a mixnet is Provotum 3.0 Mixnet [10]. This version ensures that ballots cannot be associated with the voting choices. With Provotum 3.0's mixnet-based approach it is possible to conduct multi-way elections.

Compared to the pilots, which were introduced by the canton of Geneva and the Swiss Post, the Provotum project takes a different approach; while the other pilots had a central authority entrusted with voting, Provotum is decentralized. This means that trust is distributed among different stakeholders. A blockchain (BC) can therefore ensure that the electoral processes are not dependent on trust to one authority [8]. As Provotum is very advanced in terms of security, more attention must now be paid to usability. A study [11] that examined the Neuchâtel REV shows that the design of a REV system plays a more important role than assumed. It does not only reduce the frequency of incorrect votes, but also increases trust of the voters. For this reason, the effort for a well-designed and well-evaluated user interface (UI) is well justified.

1.1 Description of Work

The overarching goal of this thesis is to implement a suitable Graphical User Interface (GUI) for the Provotum 3.0 Mixnet version. It is important that the project includes a GUI for vote casting and vote administration. This new modular system interface will be evaluated in terms of usability. From an architectural point of view this work must determine which tools are necessary for the GUIs and then integrate the corresponding software libraries or implement them.

1.2 Thesis Outline

This thesis is structured as follows. Chapter 2 introduces several concepts that are relevant for reading this thesis. For clearer understanding, the terms distributed ledger technology, blockchain, remote electrical voting, UI and user experience (UX) are explained. At the end of this chapter, an introduction to the Swiss electoral system is given. Chapter 3 presents works that are related to this thesis. The Provotum developement is introduced and the most important concepts for this thesis are briefly recapitulated. In addition, this chapter discusses related research on UI and UX in existing REV systems. Chapter 4 shows the architecture of the prototype developed in this thesis. It shows which software components were added to Provotum 3.0 Mixnet to achieve the overall goal of this thesis. Chapter 5 then discusses the various software components in more detail by presenting the implementation of the underlying components and packages. To help future developers understand why the code is structured as described in this chapter, there are also brief introductions to the various technologies used to create the proposed prototype. In addition, this chapter also provides an introduction to the implementation of the elections. Chapter 6 then presents the evaluation of the system by performing a heuristic evaluation, use case analysis, discussion and system usability scale. The final chapter briefly summarizes this thesis, provides concluding thoughts, and identifies possibilities for future work.

Chapter 2

Background

This chapter introduces theoretical concepts that are required for reading this thesis. The first section briefly explains the DL technology and explains the consequences of it. The second part introduces BC and after that how they can be applied to REV systems. After that, the basics of UI and UX are shown and also advanced techniques in evaluation and accessibility are explained. Finally, the basics of Swiss federal elections are explained, which form the basis for the implementation of elections in the prototype.

2.1 Distributed Ledger Technology

The DL technology can be described as a database architecture that stores and shares data in a distributed and decentralized manner, while using cryptographic signatures to respect the integrity of the data [12]. Not so long ago, this technology was hardly possible, which is why centralized data storage was the primary option for a long time. Thanks to a combination of different ideas, as well as more powerful internet connections, this technology has now become possible [13].

The idea of DL is that there are several nodes, whereby each node has a synchronized copy of the data forming a distributed database [12]. DL technology forms an append-only chain with all blocks of data in which each block is cryptographically-linked to a previous one. This makes the system, and thus the data stored in it, safe from tampering, forgery and other types of malicious acts [14]. Since BC technology works pretty similarly to this description, there is a need to define the distinction between the DL technology and the BC technology. The fact is that DL technology is the umbrella term for all technologies that represent distributed transaction systems. This means that BC technology is only one implementation of distributed transaction systems. Another implementation would be, for example, Directed Acyclic Graph (DAG), which is also based on DL technology. However, the distinction between DAG and the BC technology is in the use of the different technologies and protocols [13].

As a consequence of the technical terms of this technology, DL technology brings some advantages as well as disadvantages. One of the biggest advantages compared to central data storage is reduced vulnerability. With central data storage, the entire system would be destroyed after the removal of the central instance. With distributed data storage, the system would continue to function despite the failure of a network node. However, distributed databases take longer to set up than centralized databases, whereby the latter are also very easy to maintain [13].

2.2 Blockchain

A BC is a specific development of DL technology, which was described in 2.1. Therefore, the key principles of a BC are very similar to those of a DL technology. It is also a public and distributed database that is used for storing or transferring data. In order for the BC to be protected from tampering and changes, cryptography is used [15]. As the name BC suggests, it is a chain of blocks that stores lists of transactions linked by cryptographic hashes [16]. The first block of a BC has no reference and is therefore called a genesis block [17]. To add a block to the chain, the majority of nodes in the network must agree on the validity of the transactions in a block and on the validity of the block itself through a consensus mechanism [18]. The most popular consensus mechanisms are Proof of Work (PoW), Proof of Stake (PoS) and Proof of Authority (PoA), which are explained in more detail in 2.2.1. However, there are also a number of other consensus mechanisms such as Proof of Importance, Practical Byzantine Fault Tolerance or Delegated PoS [13].

It is important to note that there are multiple types of BCs. They can be divided into permissioned and permissionless BCs and also in public and private BCs. This subdivision is crucial because it defines who has access to the general ledger or how the read and write permissions are distributed [13]. The majority of current BCs are permissionless like Bitcoin or Ethereum. In permissionless BCs, anyone with an internet connection can join and leave the network at any time. All transactions are transparent and can be read by anyone as long as the protocol is followed. The reason for this is that there is no central authority that manages the participants in the network. When a central authority like an organisation is managing the BC, for example by granting read and write access rights to authorized parties, it is called a permissioned BC. In permissioned BCs, a further distinction is possible with public and private definitions. A private permissioned BC restricts the reading rights to selected set, while a public-permissioned BC allows anyone to read the state. Examples of permissioned BCs include BCs with Hyperledger Fabric[1] created by developers using the Hyperledger toolset, or BCs with Substrate[2] [16] [19] [20].

2.2.1 Consensus Mechanism

A consensus mechanism is defined by the BC, which has the responsibility to check and add blocks to the chain. This is done by the nodes of a peer-to-peer network, which has to agree on the state of the database. Some consensus mechanisms are vulnerable to attack, which makes them unsuitable for BCs. An example of such consensus mechanisms are

[1]https://hyperledger-fabric.readthedocs.io/en/release-2.2/
[2]https://docs.substrate.io/main-docs/

Byzantine Agreement Protocols, which are considered traditional consensus mechanisms. For this reason, other mechanisms have been proposed to deal with BCs and DL types [20].

One of them is the PoW consensus, which is mainly used by permissionless or public BCs. In this consensus mechanism, record keepers must solve a complicated cryptographical puzzles to create a block to the chain and get a reward [15]. This process is called mining, and the record keepers are called miners. A mining reward would be for example tokens or cryptocurrencies. The difficulty of the mathematical problem depends on the number of participants or their hash power. This means that the more hash power used to solve the mathematical problem, the more difficult it must be. As a result, a lot of power is consumed, which is a major criticism of the PoW [13].

For this reason, attempts have been made to overcome the drawbacks of PoW by developing a different consensus mechanism. One such consensus mechanism is PoS, which leverages the capital of network participants rather than solving a mathematical problem. That means the more capital a person bets in PoS, the higher the probability of winning the reward by chance. Finally, the winner is allowed to add a new block to the BC. For this verification the winner gets the transaction costs of all transactions in the new block. This process is also known as staking or forging, but unlike mining, it does not create new tokens [13].

While PoW and PoS are mainly applied to permissionless BCs, PoA is used for permissioned BCs. PoA is so well suited to permissioned BCs because it does not rely on nodes to solve mathematical problems, but instead uses a set of authorities that are allowed to author and create the blocks [19]. These authorities are usually called sealers or validators of a BC. If the majority of authorities has approved the new state of the BC, it will become part of the permanent DL in the form of an additional block. The main advantage of this consensus mechanism is that it is less computationally intensive compared to the PoW mechanism [20].

2.2.2 Public Bulletin Board

In recent years, Public Bulletin Board (PBB) has evolved from physical settings to digital settings. In the physical setting it was the most simple and efficient way to advertise a product or information. This was sometimes done in a centralized way, where you needed permission to publish your information, or in a decentralized way, where you can publish your information without permission (e.g., on streets). This means that writing can be controlled by an authority, but the reading is public for a community. At the beginning of the Internet, when there were mainly static websites, an example for a PBB would be a news website, where the readers had to trust one writer. These PBB were always limited for writing but public or limited for reading. Then social networks were reinforced by dynamic websites, with forums playing an important role as the first example of an electronic PBB. These PBBs were mostly centralized and the readers had to trust single or multiple writers, which means that the writing and the reading were public and unrestricted. In the latest form of PBB, users are empowered by decentralized networks. They can be distributed, which for example is a DL, or decentralized, with blockchains being the best-known example. In these PBBs, readers trust selected writers or multiple writers through verifiable, transparent actions [21].

In Switzerland it is an obligation to present verifiable evidence of its executed process when a majority of the public uses the system [22]. In the e-voting literature, PBBs are therefore used wherever public access to information is desired [23]. So for a PBB in electronic voting would this mean that anyone who participates is able to verify the voting information [21]. Thus, the PBB has as its responsibility the two most important concerns, namely verifiability and auditability [24]. So the key properties of a PBB are public readability, immutability of information, and that it provides a consistent state to any viewer. Due to these listed characteristics, such a PBB in combination with a resilient BC system is of great importance for REV systems. For this reason, a public-permissioned BC with a PoA consensus mechanism is the best choice for PBBs [20].

2.3 Remote Electrical Voting

In today's democracies, it is very important to be able to freely express one's political opinion. Modern information and communication technologies offer a whole range of new possibilities to the organizers of elections and votes. One of these possibilities is the use of REV systems. The most important aspects of this system are that citizens' eligibility to vote can be established through an online channel and that they can then cast their vote from any laptop or cell phone [25]. In this section, the different types of voting are briefly explained. It starts with the traditional voting system, which is the only system in Switzerland today, and then with the electrical voting system. At the end the mix-net strategy is explained, which plays a very important role in the current Provotum version.

2.3.1 Traditional Voting

In Switzerland there are different forms of voting, which are determined by the cantons. One form that is very well known in Switzerland is the Landsgemeinde[3], where people vote by raising their hands. This original form of democracy still exists only in the cantons of Appenzell Innerrhoden and Glarus. The most common form of voting in Switzerland, however, is the postal vote or the casting of the ballot in the polling station. This form of voting can be divided into several stages:

1. Setup & Delivery
2. Casting
3. Storage
4. Tallying & Publishing

[3]https://www.appenzell.ch/de/kultur-und-braeuche/braeuche-und-traditionen/landsgemeinde.html

In the setup & delivery phase, all measures are carried out that enable citizens to cast their votes. At the beginning of this phase, the voting envelopes, paper ballots and signature card must be created, which are then are put together in a envelope. After that the postal service delivers these envelopes to each letterbox of the voters. Even at this stage, there are some security vulnerabilities. For example, one vulnerability would be the transition from creating the envelope to sending it to the post office. Here, the trusted person could steal an envelope and fill it out for his interests. Another security gap would be, for example, the databases in which all people are stored who are entitled to vote. In Switzerland, several companies have such databases, as some of them are working on a solution for identity management. If one database would be hacked, then this hacker could add or delete more voters. In the casting phase, the envelope with the vote can be sent to the municipality by the postal service or the vote can be cast personally in the urn of the municipality. After that, the vote is stored in an intermediate storage of the municipality. This storage is also not very secure, as usually a person appointed by the municipality has to remove the vote from this storage. In the tallying & publishing phase, the signature cards are validated and then the counting of the ballots is started. When the count is complete, the result is sent to the canton, which then forwards it to the federal government. When all facilities have the final result, it can be published. Of course, there are several security vulnerabilities in all phases. However, the most common one in the last phase is that the tallying and the final tally are manipulated [21].

So it is safe to state that the traditional voting system in Switzerland is not very secure and not without risks. However, the steps used in the traditional electoral system are not very different from those used in a REV system, and therefore this work will reuse these steps later, since they are also applied very similarly in Provotum.

2.3.2 Electrical Voting

In Switzerland, ten cantons have offered e-voting until 2019. There were two options for e-voting systems: the REV system of the Canton of Geneva and that of the Swiss Post. However, both systems are no longer available since 2018 and 2019 [3]. At its meeting on the 28 April 2021 the Federal Council decided to put the e-voting trial on a new, stable footing [4]. The legal framework[4,5] in Switzerland prescribes desirable properties for REV systems. The main desirable properties are privacy and verifiability.

The Swiss Federal Constitution[6] states that privacy is one of the most important properties of citizens when voting. The reason for this is that the privacy of the ballot prevents voters from being coerced or bribed. Therefore, ballot privacy for public elections became mandatory in most countries in the 19th century [26]. Privacy can be divided into the following subgroups: Ballot Secrecy (BS), Receipt-Freeness (RF) and Coercion-Resistance (CR). BS is defined as "no outside observer can determine for whom a voter voted" [27]. This means that voters cannot prove what they voted for or who they voted for. In other literature, BS is sometimes referred to as "ballot privacy," which can be used as a synonym

[4]https://www.fedlex.admin.ch/eli/cc/1978/688_688_688/de
[5]https://www.fedlex.admin.ch/eli/cc/1978/712_712_712/de
[6]https://www.fedlex.admin.ch/eli/cc/1999/404/en

for BS. RF means that ”a voter cannot prove after the election how she voted” [27]. This leads to the fact that it is not possible for voters to prove to a voter-buyer that they have voted as desired. In CR, ”a voter cannot interact with a coercer during the election to prove how he votes” [27]. The difference between RF and CR is that in RF the coercer only examines evidence obtained from observation of the voting process. Such evidence can only be obtained if the voter is cooperative and thus has bad intentions such as selling the vote. In CR, the voter does not need to be cooperative [28]. Therefore, CR is found to be a stronger and broader concept than RF. This means that an system with RF alone is not sufficient to fend off attacks. But it can be used to prevent the voter from presenting evidence how the vote was cast. In systems with CR, however, the coerced voter at least has the option of pretending to follow the coercer’s instructions. That is, coercers cannot control whether the voter voted as instructed [29].

In the traditional voting system, it is not possible for a voter to verify that the ballot has been correctly recorded. It depends on whether the postal service or election officials did their job properly. In a REV system, the trust required is even more important. This is because in a REV systems, it cannot observe the physical processes in a traditional system, such as the sending of the ballot or anonymization. Therefore, techniques are needed to allow the voter to verify that everything is being executed correctly. For this reason, verifiability is an important component in any type of voting system. It can also be divided into the following subgroups: Individual Verifiability (IV), Universal Verifiability (UV) and End-to-End Verifiability (E2E-V). The definition of a IV is that ”a voter can verify that the ballot containing her vote is in the published set of “all” (as claimed by the system) votes” [27]. IV alone is not enough for a REV system to provide verification. For this reason, there is the UV where ”anyone can verify that the result corresponds with the published set of “all” votes” [27]. IV is not sufficient, because it is trusted that the other voters will also verify that their ballot is correct in the set. That is why UV plays an important role in the verification process. However, there is another subgroup E2E-V, which comes with three additional properties:

Cast-as-Intended (CaI)	”Her choice was correctly denoted on the ballot by the system”
Recorded-as-Cast (RaC)	”Her ballot was received the way she cast it”
Counted-as-Recorded (CaR)	”Her ballot counts as received”

Table 2.1: Three properties of End-to-end Verifiability [27]

Considering that REV systems are in an uncontrolled environment, e.g., over the Internet with a cell phone or a laptop, CaI is very important [30]. With a CaI system, the voter would be able to know whether his vote was recorded correctly. In addition, the voter can also check in this way whether the vote was manipulated during the voting process [31]. However, it is very difficult for a voting system to perform verification while protecting his privacy [32]. For this reason, a good REV system should implement the main principles of privacy and verification, and find a compromise between the two.

2.3.3 Mixnet

The mixnet approach aims to protect the privacy of each voter by mixing ballots with a number of other ballots. This mixing makes it impossible to link the output to the input. A mixnet ensures that the link between input and output cannot be recovered by changing the appearance without changing the meaning [27]. With this approach, any verifier can be sure that a mixnet choice has been made, but without knowing the mixnet permutation [33]. Such a mixnet is often used in electronic voting, where the input for a mix consists of encrypted ballots ordered by a list of voters. If these ballots were not shuffled, it would be possible to determine how each voter voted after decryption, since the order of the ballots would match the order of the voter list [34].

2.4 User Interface and User Experience

The Internet contains countless websites that can be accessed from a variety of devices these days. It is easy to lose sight of the importance of interaction with these websites and how differently everyone interacts with these interfaces [35]. The research that focuses on human-computer interaction is known as Human Computer Interaction (HCI). UI design and UX design both have specific contributions to HCI. The following sections explain the terms UI and UX in more detail and what to look for in websites when it comes to accessibility.

2.4.1 UI

The UI takes a very significant role in HCI research, as it includes everything the user can see, hear, touch and interact with to achieve the goals [36]. It is very important for a good UI that the interaction comes across easily and naturally to the user. It is also the case that a good UI adapts over time. However, if the UI is poorly done, the user may get frustration and dissatisfaction and greatly affect productivity in a professional environment. To avoid this, UI designers usually work with prototypes. Especially low-fidelity prototypes are often used at the beginning. Low-fidelity prototypes are sketches that show the planned UI. Such sketches are sometimes drawn by hand, but can also be created using drawing programs such as Paint. In contrast to low-fidelity prototypes, there are also high-fidelity prototypes that already show the UI layout and its navigation. They already roughly show the look, layout and behavior of the final product. Nowadays, only software applications like Visual Basic or Figma[7] are used. A big advantage of high-fidelity prototypes is that they can be used as a marketing tool to present the first demo to a customer [37].

The quality attribute usability is often described as ensuring that UI's are easy and effective to use [36]. This term has several properties, which are briefly explained in the following table:

[7]https://www.figma.com/

Learnability	Shows how difficult it is to perform simple tasks when using a design for the first time
Efficiency	Indicates how quickly a user can complete a task after learning the design
Errors	Shows how the UI responds to human errors and how users recover from those errors
Memorability	How quickly can users regain all knowledge after an interruption of the use of the design
Satisfaction	Shows how pleasant the design is for the user

Table 2.2: Properties of the term usability [36]

2.4.2 UX

Unlike the UI, UX design can be more versatile because it defines how a person feels when interacting with a product [38]. This means that the product does not necessarily have to be software. At the same time, the UX may be distinct in different cultures, but it should be useful and appealing to each of them [39]. To be more specific about the term experience, there is a model that divides this term into three subcategories of interaction: What, How, and Why. The first category, the what level, is about showing what people can do with an interactive product. An example of this would be a phone call where the user can answer and make a call. The second category, the how level, is about making the features accessible and aesthetically pleasing to the user. Accessibility is discussed in more detail in the following section. The last category, the why level, deals with the motivation for using a product. Here, for example, emotions can play an important role. For example, when making a phone call to your beloved, the end is always sad because you can no longer communicate with your partner afterwards. To fulfill this need, there are text messages that no longer trigger this sad feeling. This would be a reason to use text messages [40].

2.4.3 Accessible Software Design

Accessibility as an inclusive development approach [41] is about ensuring that a product or service can be used by all. In this context, accessibility focuses on people with disabilities, and various laws exist for this purpose. However, accessibility does not only bring benefits to the UX of people with disabilities, but to all users. Because when a design is equipped for all ability levels, it is possible to create products and services that are appealing, or at least helpful or reassuring, to everyone [42]. Almost all web browsers provide some accessibility features like visual zoom levels or text-to-speech synthesis. However, these features can only be fully used if the website has taken them into account. For this reason, the World Wide Web Consortium[8] (W3C) provides instructions that can help developers implement these features [43]. Accessibility is broadly concerned with five different areas of the human body. These will be discussed in more detail in the following subsections [41].

[8]https://www.w3.org/

One of the five faculties of the human body is the **visual ability** of the eye. There are various weaknesses, such as "blindness, inability to see high-contrast colors, reduced vision at night, color-blindness, etc." [41]. These weaknesses do not always have to be permanent but can also be of a temporary nature. This can be, for example, when someone loses their glasses or bright sunlight is involved [41]. Thus, it can be seen that many products are not accessible to large parts of the population. The reason may be that designers do not know the needs of users with different abilities or how to address them [44]. Even small software adjustments can lead to a pleasant experience for many more users. An example of such small software adaptations are labels and metadata or alternative text. A big help for individuals with impaired eyesight is text to speech synthesis. This technology helps people to understand the written text by reading out the content of the screen [45]. However, with web forms the problem could arise that all labels of a form are read out, but the user does not know which input field belongs to which label. So, it can happen that the user knows which labels there are but does not know if the selected input field belongs to the correct label. This problem can be solved if the developer adds a attribute to the label, which must exactly match the id attribute in the input field. This way each label can be associated with the input field and the user with impaired eyesight now understands what has to be written in the selected input field [43]. For images, the alternative text can help. Here the user with impaired eyesight cannot see the image. However, if the developer has added an alternative text to each image, it will be read out by the text-to-speech synthesis. This is also an example that benefits not only the UX of people with disabilities, but also those who live in areas with expensive or low bandwidth. These people have images turned off on websites. If this is the case, the alternative text will still be displayed. But the alternative text does not just help people. Most technologies cannot see images either and that is why they use the alternative text. Good examples of this are search engines like Google [43]. Another important aspect is the font of the website. Most browsers, as well as Android and iOS, allow users to increase and decrease the size of the font, images, and other content with a zoom function. Some of these browsers even offer functions to adjust the font size and color in the default view [43]. This is not so easy with the font family. Here, the developer should carefully decide which font family to use on the website. It should be noted that there are disabilities such as dyslexia that make reading and writing difficult for those affected. In a study with dyslexics, it was found that sans-serif, monospaced and roman fonts are preferred and italic, serif and proportional fonts make reading difficult. This study should make it easier for developers to choose a font that offers a range of more accessible typefaces [46].

Hearing is also one of the five faculties of the human body. People with hearing disabilities are affected by deafness or decreased hearing in certain environments [41]. Therefore, it is important that its accessible website provides text transcripts for audio files, as these are not accessible to people with impaired hearing. This is also valuable for availability to search engines as well as other technologies that cannot hear. For websites, it is also possible to provide such transcripts without much effort and cost. Indeed, there are even transcription services that allow the creation of HTML text transcripts [43].

Cognitive disabilities are among the most common impairments [47]. In this case, processing instructions takes longer than usual [41]. This leads to limited comprehension and also a low tolerance for cognitive overload. The latter is the case, for example, when too many things happen at the same time. Therefore, it is central that processes are designed to be simple and straightforward. People with cognitive disabilities also have limited problem-solving skills. Therefore, for example, the use of a CAPTCHA is not suitable, since cognitive disabilities can lead to the further process not being processable. Another deficit of such persons concerns attention span. This leads to difficulties in concentrating on the process in question because attention falls on other things. This is the case, for example, with online advertising displayed on a website. In addition, many people with cognitive disabilities have difficulty reading. To assist these individuals, it is helpful if additional illustrations or even audio files are provided [47]. Furthermore, it is important that a website does not cause seizures. These can be caused by quick flashes of light, strobe lights or flickering [48].

When **speech disorders** occur, the affected person may stutter or be mute [41]. However, this disability also includes an accent that is difficult to understand. To provide barrier-free access for these people, text-based communication methods such as a chat, e-mail or other contact forms should be offered. This provides an alternative to speech-based communication [49].

In the case of reduced **mobility**, the performance of an action at screening takes longer than usual [41]. This also affects the use of the mouse, since it requires fine motor skills. Some people, for example older people, cannot use the mouse at all or only to a limited extent, which is why it is essential that all functions of an accessible website can also be operated via keyboard. This enables people with disabilities to use assistive technologies that imitate the keyboard. These include voice input, for example [43].

2.4.4 Heuristic Evaluation

In order to quickly identify usability problems, developers often perform a heuristic evaluation. Such an evaluation has the advantage that it is very easy to perform and that it reveals problems quickly [36]. The evaluators or the developers can examine the UI considering a set of usability principles established by Nielsen [50]. These principles were developed by Jakob Nielsen between 1990 and 1994 but are still used today. The top 10 heuristics [51] that are suitable for the evaluation of a UI and which have also been used for this thesis are:

1. Visibility of system status

With this principle, it is important that the user is always informed about the current state. This is done through visual cues and feedback after an action has been performed, e.g. when a key is pressed, something should happen so that the user realizes that the key has been pressed.

2. Match between system and real world

The UI should use the language of the user with the terms and concepts. This means that the information is represented with metaphors from the real world. An example of this would be a trash can for deleting files. Such models greatly improve the usability of a design.

3. **User control and freedom**
In this principle, it is very important that the user feels comfortable. If a user accidentally enters an undesirable state, for example by selecting the wrong button, they should have the option to exit that undesirable state by clicking undo or redo.

4. **Consistency and standards**
"Users should not have to wonder whether different words, situations, or actions mean the same thing." [51]

5. **Error prevention**
You should avoid the occurrence of error messages as much as possible. However, if an action should be performed that could have worse consequences, the user should be given a confirmation option before performing the action. A good example of this would be deleting data.

6. **Recognition rather than recall**
With this principle, it is important that the user does not have to remember information from one part of the system to another. Therefore, objects, actions and options should always be visible.

7. **Flexibility and efficiency of use**
With this principle, it should be remembered that accelerators and shortcuts are supported for experienced users.

8. **Aesthetic and minimalist design**
A UI should not make irrelevant information visible in such a way that it competes with relevant information for its relative visibility.

9. **Help users recognize, diagnose, and recover from errors**
Error messages should not be formulated in a technical way, so that the user also understands what went wrong and what to do now. A solution must always be suggested.

10. **Help and documentation**
If the system cannot be used without documentation, the UI should offer help. This documentation should then list only the most important steps for the user's task and should not be too extensive.

2.5 Swiss Federal Election

This section explains the basic principles of Swiss federal elections. The prototype of this work includes both the ability to create votes and the ability to create elections. These elections were developed to allow the election of the National Council in Switzerland. In the National Council elections, the 200 members are elected by the 26 cantons, with the respective population size playing an important role. The canton of Zurich [52] will be able to fill 36 seats from 2023, while six other cantons will only be allowed to hold one seat [53]. In these six cantons, the party with the most votes wins the seat [54]. In the twenty other cantons, party lists are provided on which as many candidates can be nominated as the canton has seats in the National Council. As a voter, you may cross off candidates on the lists or write them down twice. However, it is important that the total number of names on a list does not exceed the number of seats in the canton. In addition, there must be at least one candidate on each list. As a voter, you can also write a person on another list instead of a candidate. This approach is called panache. Also with this method it is possible to write a candidate twice.

However, it is very important to remember that there are party votes and candidate votes. If a voter has voted with a predefined list of candidates, each candidate receives one vote and the party receives the maximum number of votes (Canton Zurich 36 party votes). When you drop candidates from the list, the candidate votes disappear, but not the party votes themselves. This means that even if there is only one person on a list, the party will receive the maximum number of party votes. Panache is different, because the voter chooses a person who belongs to another party. In this case the party vote of the list party decreases by one and the party vote of the party in which the candidate runs with panashic votes increases by one. Of course, the party vote of the list party decreases even more if the voter panaches more often.

And then there are empty lists, where the voter does not choose a party. In this case, the elected candidates and their parties always receive the candidate vote and the party vote. It is independent of how many candidates are nominated by the same party. However, as with a predefined list, a candidate may be listed no more than twice. Futhermore, if there are no other candidates on an empty list, the unlisted positions are forfeited [55].

Chapter 3

Related Work

This chapter focuses on the existing approaches that have dealt with REV and are related to this work. In order to create a UI for a REV system, it is very important to understand the background processes of such a project. For this reason, the first part of this chapter deals with the Provotum project itself. Discussed are the development of Provotum, the stakeholders and the basic concept of Provotum. The second part of the chapter introduces related research that has already dealt with UI's for REV systems.

3.1 Provotum

Provotum is a proof-of-concept project that is being developed at the University of Zurich. The goal of this project is to provide a fully decentralized REV system that deploys a permissioned BC as PBB [20]. The latest version of Provotum is Provotum 3.0 with a mixnet-based approach (Provotum 3.0 Mixnet) [10], on which this thesis builds. To get an overview of the development of Provotum, the first part of this section presents the different versions of Provotum. In the second part, the stakeholders for the Provotum REV system are listed and explained. In the last part of this section, the version of Provotum 3.0 Mixnet is explained further and the inner functionalities are described in more detail.

3.1.1 Provotum Development

In 2018, the Provotum project started using DLT for storage only. However, the rest of the project was centralized, which led to security and privacy issues [8]. Provotum version 2.0 improved the prototype with the issues described in Provotum 1.0 in mind. In addition, this version was the first fully decentralized version with client-side encryption. However, this version also did not provide CR and RF [20]. In version 3.0, the prototype of Provtum was rebuilt using substrate. This prototype then solved problems that were encountered in version 2.0. For example, version 3.0 is RF, which is solved with a new entity called randomizer. In addition, the use of substrates has led to a number of other additions [9].

An example of this would then be Provotum 3.0 Mixnet [10], which is analyzed in more detail in the Subsection 3.1.3. This version of Provotum is the most current and also the one that is used for this thesis.

3.1.2 Stakeholders

In order to show how the inner workings of Provotum 3.0 Mixnet function, it is necessary to introduce the involved stakeholders. As illustrated in Figure 3.1, the Provotum REV system has several stakeholders. It always contains a voting authority (VA), a number of sealers and voters. Since version 2.0 there is also a randomizer as stakeholder. In the earlier versions, there was sometimes an identity provider who also acted as a stakeholder. However, in the version Provotum 3.0 Mixnet this identity provider is not yet implemented and therefore not listed in Figure 3.1. All stakeholders have different responsibilities, which are explained in more detail in the following sections.

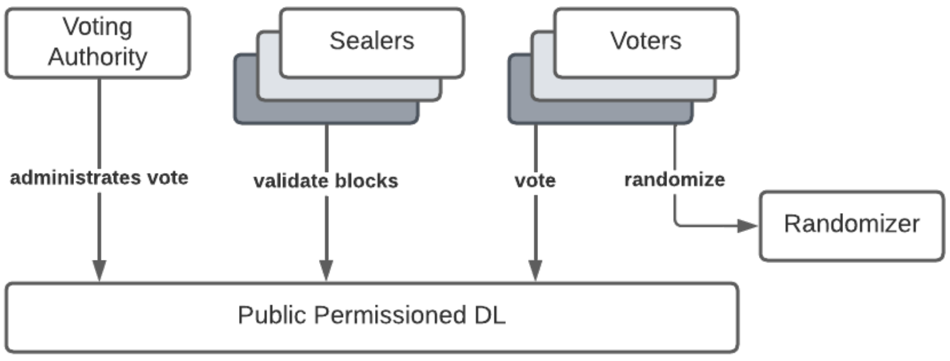


Figure 3.1: Provotum Stakeholders, after [10]

The **Public Permissioned DL** function like the PBBs of REV system. They consist of several nodes, some of which serve as validation nodes to validate new blocks. This means that the public-permissioned DL uses a PoA consensus mechanism for Provotum. All functions, such as creating a vote, are implemented in the DL. In addition, everything stored in the PBB is public [10].

The **Voting Authority** is the stakeholder responsible for coordinating the voting. Therefore, its tasks include creating the votes, combining the key shares to generate a public key for the vote, changing the phases of the vote and combining the decrypted shares. In the case of a vote in Switzerland, the VA would be the canton or national government, which would run the VA in a secure data center [10].

Sealers are also very important for the Provotum project. The reason is that Provotum uses a public-permissioned BC as PBB, where only validated entities can sign blocks. These authorities, named sealers, cannot only sign the blocks but also participate in the PoA consensus mechanism, which is explained in more detail in the Subsection 2.2.1. The main tasks of the sealers are to participate in the distributed key generation and in the decryption after the vote. In Switzerland, in the event of a government vote, this could mean that each canton would act as a sealer [20].

The **Voters** are the stakeholders who can cast their vote [20]. Since there is no identity provider in the Provotum 3.0 Mixnet version, the voters' only task is to cast their vote. In the Swiss government example, this would be all Swiss citizens who are allowed to vote.

The **Randomizer** is the reason why Provotum is RF since version 3.0. Its task is to encrypt the ballots again with a random value without knowing the plaintext of the ballot [9]. As a result, voters can no longer prove which ballot is theirs on the PBB [10].

3.1.3 Provotum 3.0 Mixnet

The goal of Provotum 3.0 Mixnet was to reduce ballot complexity and make Provotum election-ready. This allowed a nationwide scalability. To achieve this goal, a mixnet-based REV system was developed to ensure reliability, accountability, and robustness. This mixnet-based approach eliminated the constraint that a vote can only be either one or zero. In addition, the prototype of this version has been used to test and evaluate voting and elections with up to 1 million ballots. The ideas of Cramer et al. [56] also greatly influenced the voting protocol. As shown in Figure 3.2, the election protocol is divided into three phases. These phases are explained in more detail in the following subsections [10].

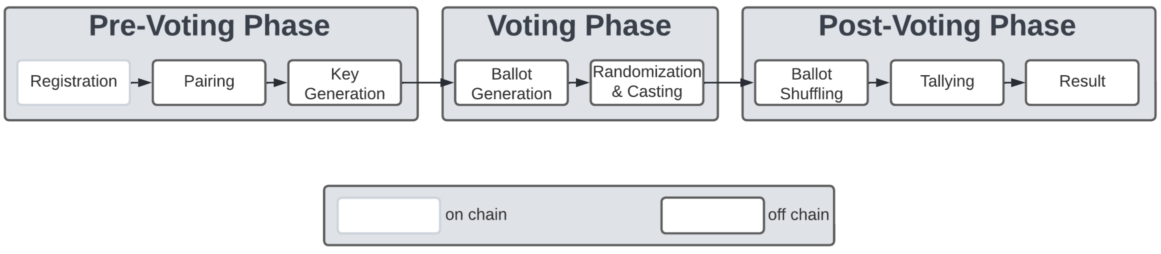


Figure 3.2: Provotum's Voting Protocol Phases, after [10]

Pre-Voting Phase

The pre-voting phase is very important, because this is where the vote is prepared. This phase is divided into three main tasks: registration, pairing and key generation. In the **registration** step, all sealers generate a public key and a private key. These keys are used to identify the sealers in the DL. Also, the chain has not started during the registration process and therefore this step is still off chain. This changes during the **pairing** step. Here the chain is started with a publication of the configuration file from the VA, which contains the public keys of the sealers and the public key of the VA. The reason for this is that the DL knows who is allowed to validate the blocks in the network. It is very important that the VA is only allowed to administrate the votes and therefore does not become too powerful as a network peer.

In the last step of the pre-voting phase, a series of events takes place called **key generation**. First, the VA creates a vote with the correct parameters and sends it to the PBB, which checks the parameters and saves the vote. After that, the VA can send more questions to the PBB, which will be part of the same voting. After the vote is created with all questions, all sealers must create a public/private key pair, which is then also sent to the PBB with a zero knowledge proof. If the PBB verifies the proof valid, then the public key share will be stored. Once all the sealers have created the key pair, the VA must trigger a key creation action that forces the PBB to combine all the key shares into a public key for the vote. This key is then later used to encrypt the votes of the voters [10].

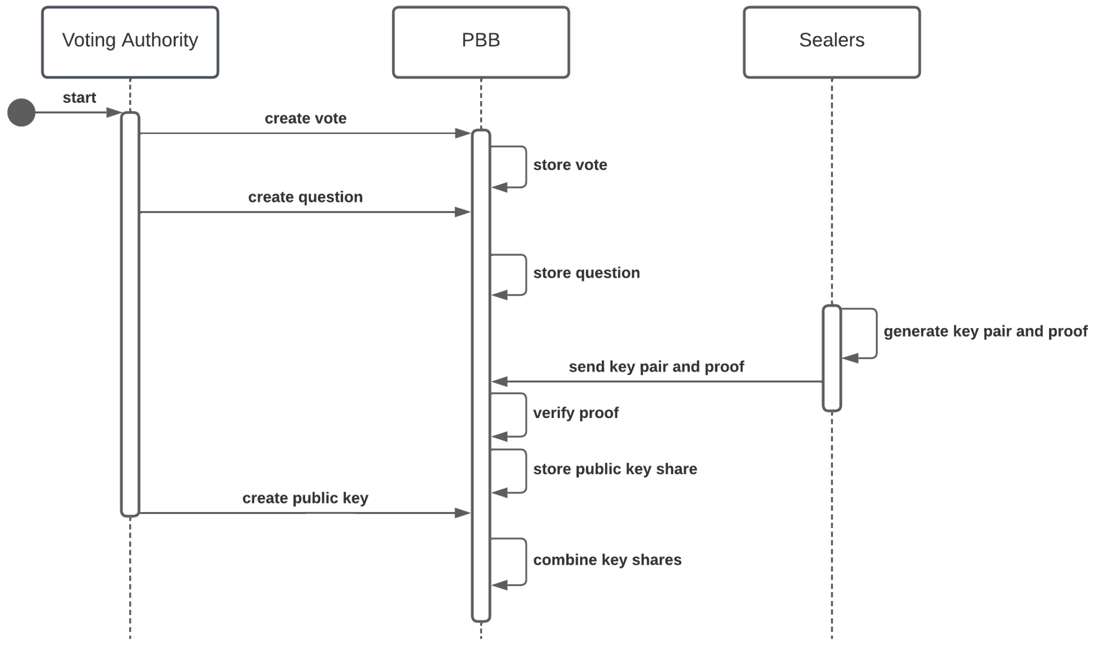


Figure 3.3: Pre-Voting Phase - Key Generation Step, after [10]

Voting Phase

The voting phase is divided into two steps: the ballot generation and ballot randomization & casting, which can be seen very well in Figure 3.4. This phase is actually the main phase when thinking about a REV system because voters can only do something in this phase. In the first step, **Ballot Generation**, voters answer the ballot questions. For each answer a random ballot is created, which is encrypted with the public key of the vote. This key was generated by the PBB in the key generation step in the pre-voting phase. In the second step, **Ballot Randomization & Casting**, the voter sends all ballots to the randomizer. This randomizer encrypts all the ballots again and signs them with the private key. It also proves that the re-encryption contains the same vote. Then the voter must verify that the proof is valid. If the proof is valid, then the ballot is sent directly to the PBB. When a new ballot reaches the PBB, it gets verified that the ballot is not a copy of another ballot and contains the signature of the randomizer. This ensures that the ballot is randomized. Finally, the ballot is stored in the PBB [10].

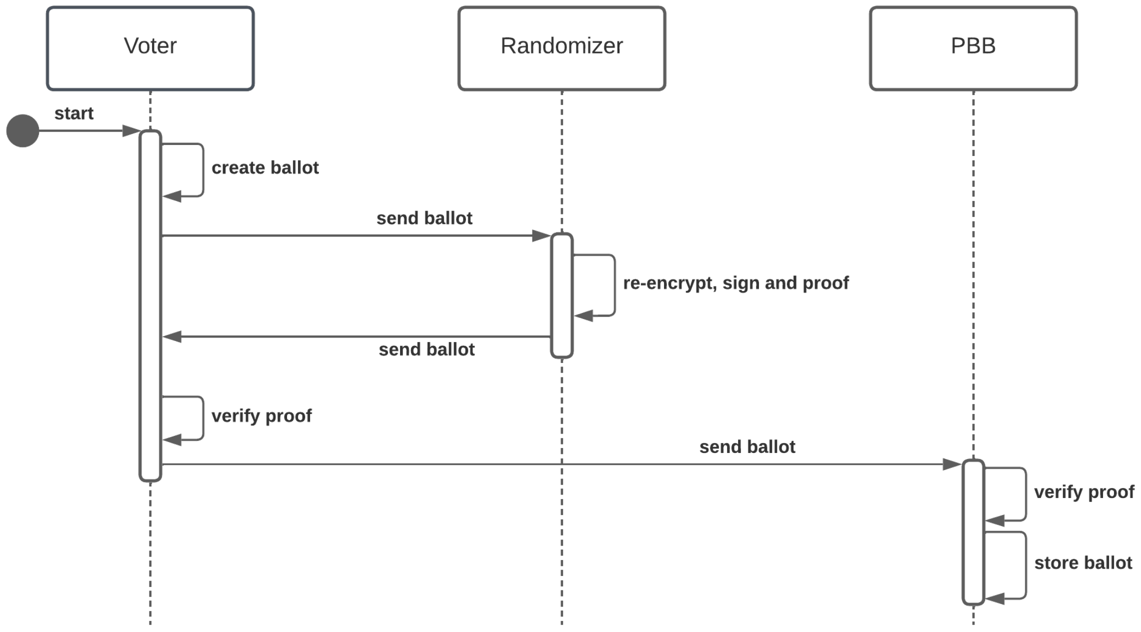


Figure 3.4: Voting Phase, after [10]

Post-Voting Phase

The post-voting phase is the phase that begins after the end of voting. All ballots sent in this phase will be rejected by PBB. If you compare the entire voting protocol with Provotum 2.0 and Provotum 3.0, you will notice that these versions did not yet have the ballot shuffling step. This was only added with the Provotum 3.0 Mixnet version. For this reason, the post-voting phase is divided into 3 steps: ballot shuffling, tallying and result. In the **Ballot Shuffling** step, all ballots are first shuffled so that they can no longer be associated with the creator. This is done by the sealers receiving a copy of the ballots from PBB, thus initiating the process. After shuffling, a proof and the shuffled ballots are returned to the PBB. There, the proof is verified and the ballots are stored if the proof is valid. After the ballots have been shuffled, the **Tallying** step begins, which initiates decryption by the VA. All sealers who were also involved in the key generation must decrypt the shuffled ballots. If only one sealer failed to do so, the vote could not be decrypted. This decryption is then submitted to PBB along with proof. If the proof is declared true by the PBB, the decryption is saved. The last step is the **Result** step. In this step, all decrypted ballots are first combined, which is triggered by the VA. Once combined, they are decrypted, which then leads to the revealing of the result [10].

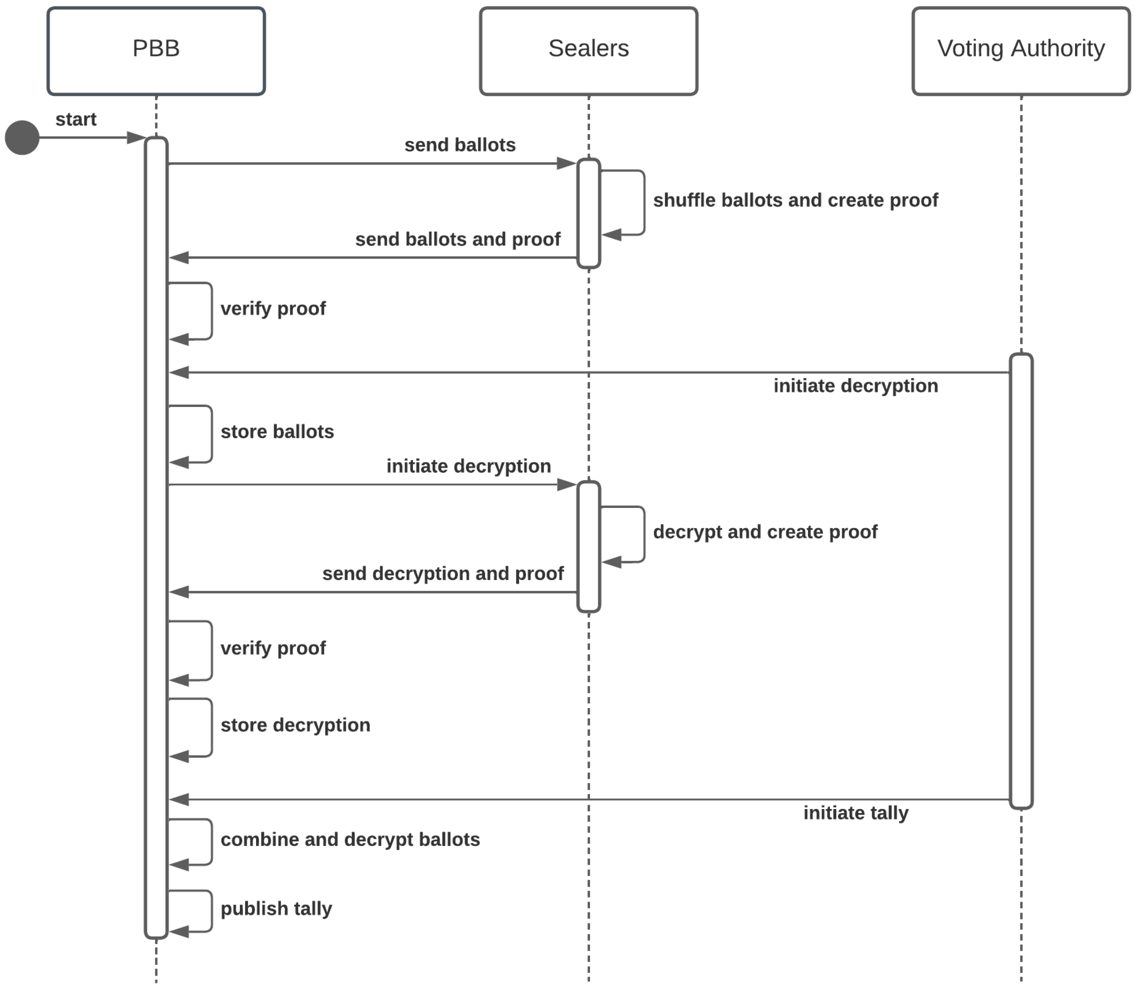


Figure 3.5: Post-Voting Phase, after [10]

3.2 UX and UI for REV Systems

In this section some existing research publications are presented that have focused on the UI and UX of REV systems. The reason is that there are many research projects and only some of them are already in use. However, the research has mainly focused on theoretical security, protocol design, and privacy of REV systems.

A research in Indonesia has investigated the factors that influence the adoption of e-voting technology. The result was that security plays an important role in the decision to determine an e-voting technology. However, for the adoption of the e-voting technology the attributes relative advantage, complexity and compatibility are of great importance. Relative advantage means that it is important that this innovation is considered better than the existing one. It also depends on how difficult it is to understand an innovation and that it also adheres the values, beliefs, experiences and needs of potential users [57].

Another study[11] has attempted to uncover the weaknesses of Swiss Internet voting and improve it. They commissioned twelve experts in HCI to find usability errors, which were then included in their evaluation. Based on this evaluation and a user study with 36 participants, the system was redesigned. After that, another user study was conducted with 49 participants on the redesign. The result was that 33% more incorrect votes were detected and an increase in trust and understanding from voters was also recognized. The reason for this were mainly the following 5 design adjustments:

1. **Know what to do in case of a wrong vote**
The purpose of the verification is to find incorrect votes. So if such a vote is found, the voter must be informed about it and he or she must know how to react to it.
2. **Only one task per step**
If a voter has to do several things in one step, it can be easy to overlook some tasks. For this reason, each step should contain only one task.
3. **Ask directly for the result of mental tasks**
It can increase effectiveness if a comparison is isolated to a specific step with buttons that ask directly for the result.
4. **Provide instructions and verification information**
There should be instructions at every step to ensure good guidance.
5. **Provide clear labels for verification data**
Any code used for verification must be clearly labeled to reflect the purpose.

Another study investigated the influence that the display of security mechanisms has on the UX in e-voting. Thereby two e-voting applications were developed; one with and one without displaying security protocols. Data for the UX was then collected for the two prototypes through semi-structured interviews and standardized rating scales. The result was that the prototype that presented security information performed better in terms of UX. For this reason, the authors of these studies have provided recommendations for the design of security-relevant technologies. These recommendations should help designers improve the UX for e-voting systems. One of the most important recommendations is to put a lot of emphasis on security during the authentication phase. The reason for this is that many users believe that the perceived security during the authentication phase is transferable to the overall security of the system. This means that many users cannot properly assess the security level of the system. Also, as a designer, one should not assume that users have major security concerns. Therefore, it should also be avoided that a security-priming bias is created by not asking participants to think about safety issues [58].

Chapter 4

Software Architecture

This chapter describes the software architecture of the prototype of this thesis. The first part of this chapter illustrates the structure of the Provotum 3.0 Mixnet version before this work supplemented the prototype. The second part of this chapter lists the open areas mentioned in the Provotum 3.0 Mixnet version. There it is also elucidated which open areas this thesis has addressed and how it was implemented.

4.1 Provotum 3.0 Mixnet System Overview

Much has already been said about the Provotum 3.0 Mixnet version in subsection 3.1.2 and 3.1.3. For this reason, this section will mainly deal with the architecture of this version. As can be seen in Figure 4.1, the relationships between some stakeholders and the packages are shown and it is also explained what these packages are responsible for. After that, it is shown how the Mixnet version 3.0 interacted with the nodes by listing all possible CLI commands. At the end of this section, the Docker architecture of this Provotum version is presented and explained.

4.1.1 Packages

As shown in Figure 4.1, the Provotum 3.0 Mixnet prototype is mainly divided into four different packages (client, node, crypto and randomizer). The arrows show how the packages interact with each other. It can be seen that the crypto package has no interaction with the other packages. However, it is very important because all the other packages depend on this crypto package. The main task of this crypto package is to implement the algorithms and proofs that occur in the voting protocol phases in Subsection 3.1.3. It is also packaged as a library so that it can be used and included by other packages (e.g. randomizer or client). These packages are willing to use the functionality of the crypto package. However, the main component of this prototype is the node package. The reason is that the PBB and the voting protocol that uses substrate are implemented

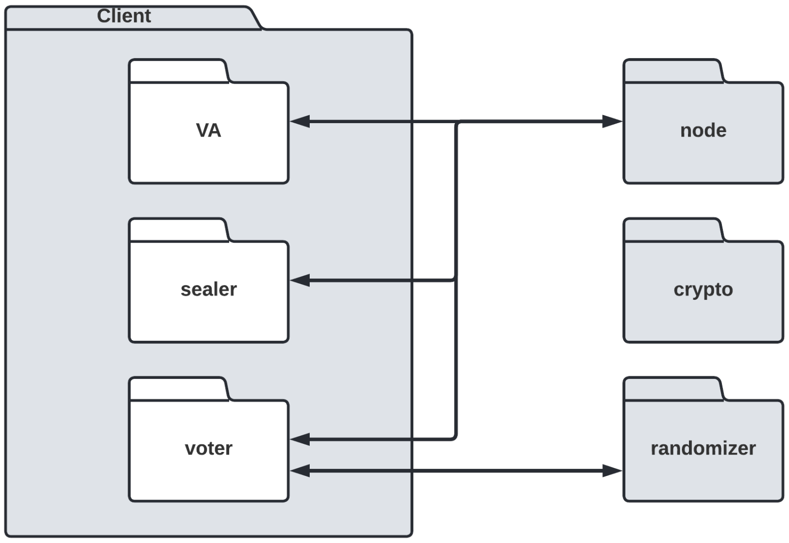


Figure 4.1: Provotum Prototype Packages, after [10]

in this component. To ensure good performance in this component, the Rust[1] programming language is used for this and all other packages. Rust is a modern low-level language that is type-safe and memory-safe. The randomizer is another package in this prototyp. Its task is to re-encrypt the ballots and generate the proofs. The proofs are created by the designated verifier and with the help of the crypto package. The last package is the client package. This package has the components VA, sealer and voter bundled into a single CLI. As the name of these components implies, they represent a stakeholder, which has already been described in Subsection 3.1.2. With the help of the crypto package, the client can interact with the node package and with the randomizer package. How this works exactly will be explained further in the next chapter [10].

4.1.2 CLI Commands

As explained in Subsection 4.1.1, the client package contains VA, sealer, and voter bundled into a single CLI that can then interact with the randomizer and the node package. This single CLI is called Provotum CLI and the goal is to impersonate the VA, voters and sealers. For this reason, each command in the CLI is preceded by a subcommand that specifies whether the command executes a VA, sealer or voter. Table 4.1 lists all possible commands that have been implemented in the Provotum 3.0 Mixnet version. The table is ordered in such a way that it follows the Provotum voting protocol. A command would be executed as follows: provotum-cli <subcommand> <command> <parameters>

[1]https://www.rust-lang.org/

Subcommand	**Command**	**Parameter**
va	setup	vote, question
va	store_question	vote, question
sealer	keygen	vote, sk, who
va	combine_pk_shares	vote
va	set_phase	vote, phase
voter	-	vote, question, nr-of-votes, votes
sealer	decrypt	vote, question, sk, who
va	tally_question	vote, question
va	result	question

Table 4.1: Provotum's CLI commands

Notice that there is no command for the voter. The reason for this is that the voter can only perform one action (vote), so there is no reason to define a command.

4.1.3 Docker

In the thesis, all packages were containerized using docker[2]. The reason for this is to simplify a project exploration and allow reproducibility. Furthermore, a docker-compose[3] script is provided. This script allows to start a predefined setup with a single command. Listing 4.1 shows the main parts of the docker-compose script, which means it shows four different containers (alice, bob, charlie, randomizer). The alice container is the VA, the bob and charlie container are the sealers, and the randomizer container is responsible for re-encrypting the voters' ballots. Lines 8, 21, 34, and 44 also show that only one docker image was created for this prototype. Therefore, the distribution of stakeholders is defined by commands in the docker-compose file (lines 9, 22, 35). Now if one were to run the command (docker-compose up) in this docker-compose file, then the Provotum prototype would start by launching the public-permissioned DL. Using a bash script or a terminal, it is then possible to run commands from the various stakeholders in the client package and simulate a vote.

[2]https://www.docker.com/

[3]https://docs.docker.com/compose/

```
services:
  alice:
    container_name: alice
    image: ghcr.io/provotum/provotum-mixnet:latest
    command: --chain=local --name Alice --base-path /tmp/alice --port 30333 --ws-port 9944 --rpc-port 9933 --execution Native
    network_mode: host
  bob:
    container_name: bob
    image: ghcr.io/provotum/provotum-mixnet:latest
    command: --chain=local --bob --validator --base-path /tmp/bob --port 30334 --ws-port 9945 --rpc-port 9934 --execution Native
    network_mode: host
  charlie:
    container_name: charlie
    image: ghcr.io/provotum/provotum-mixnet:latest
    command: --chain=local --charlie --validator --base-path /tmp/charlie --port 30335 --ws-port 9946 --rpc-port 9935 --execution Native
    network_mode: host
  randomizer:
    container_name: charlie
    image: ghcr.io/provotum/provotum-mixnet-randomizer:latest
    network_mode: host
```

Listing 4.1: Structure of docker-compose.yml File

4.2 Open Areas in Provotum 3.0 Mixnet

The prototype of the Provotum 3.0 Mixnet version works perfectly as described in Section 4.1. However, the prototype of this version still has basic features that need to be implemented. Crucial examples are the lack of CaI verifiability support, the lack of identity management, or the lack of GUIs. Although earlier versions of Provotum had an identity management, it was not implemented due to the limited amount of time [10]. This work extends the Provotum 3.0 Mixnet prototype with GUIs for casting and managing votes. The goal is to make all features offered by this version usable via a GUI.

The difficulty in this thesis was that although there were some client commands (Subsection 4.1.2), there was no Application Programming Interface (API) provided to execute them. This is obviously problematic when trying to provide all functionality via GUIs. For this reason, the implementation of an API was also part of the scope of this work. This API is quite simple, but it allows the execution of the commands provided by the 3.0 Mixnet version via REST API requests. This means that the API is communicating with the client package, which can be seen in Figure 4.2. However, another problem is that the prototype depends on these client commands. So the question arose how to get back information that should be presented in the GUI. For example, when the VA creates a vote, the information is sent to the public-permissioned DL, but this information can no longer be retrieved by the client commands. For this reason, a database was added to the prototype to store the information, which are relevant to the GUIs. It is clear that this contradicts the overarching goal that the Provotum project is decentralized, but due to the limited time frame and scope of this work, it was implemented this way. Since this database will have to be replaced in the future anyway, it can also bring other advantages. For example, users can now also be created, which makes a better impression during a demonstration. This method can be used to work around the lack of an identity management system. This chapter will show the architecture of the prototype and explain important elements (GUI, Database, API and Docker) in more detail.

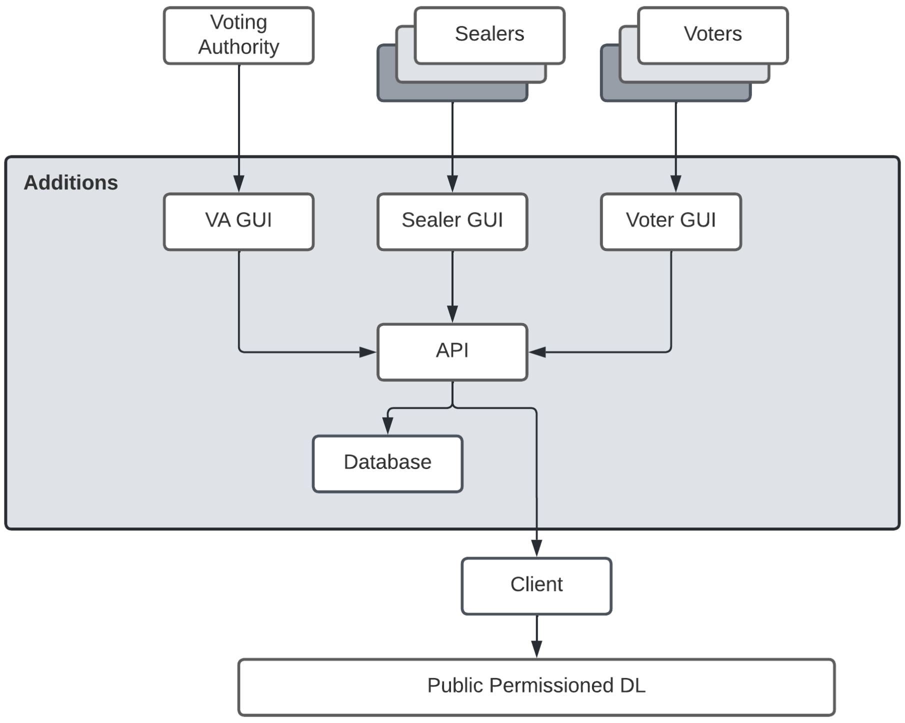


Figure 4.2: Provotum 3.0 Mixnet Architecture

4.2.1 Assumptions

It is very important to define assumptions before designing the system [59]. The reason is that these assumptions influence the design and implementation of the prototype [30]. The Table 4.2 illustrates the assumptions that were made to achieve the goal of this thesis.

ID	**Description**
A1	An identity management handles the authentication.
A2	The GUIs are only accessible to the respective stakeholders.
A3	API requests are secure and only possible via the respective GUI.
A4	Each voter is sincere and has no bad intentions.

Table 4.2: Assumptions

Considering *A1*, the system uses identity management to ensure that only users with voting rights have access to the system and that each voter is uniquely identifiable. *A2* implies that the access to the various GUIs is only available to certain people. This would mean, for example, that the VA GUI is only accessible to selected federal government employees. These employees are not interested in exploiting security vulnerabilities in the GUIs. Considering *A3*, the API only allows requests that come from the corresponding GUIs and are also encrypted. However, *A2* must be enabled for this assumption to serve its purpose. This prevents, for example, that an arbitrary person is able to create a vote or an election. Furthermore, the connection between the users' devices and the BC is confidential. *A4* means that the voter has no interest in manipulating the UI in such a way that a fraud or a potential failure of the entire system is possible.

4.2.2 Graphical User Interfaces

The main part of this thesis is to provide GUIs for vote casting and vote administration. For this reason, the prototype now offers three different GUIs; one for the VA, one for the sealers and one for the voters. The difference between them lies mainly in the different functionalities. This means, for example, that the VA GUI can only perform the functions for which it is intended. Another distinction is the importance of using a bash script in the sealer GUI. This is important because the sealer GUI works with environment variables to determine the port and the sealer's name. Unlike the VA, there can be multiple sealers. The definition of how many sealers there are and what they are named is done in the docker-compose.yml file. The name and port for one sealer is set there. This will be further discussed in the Subsection 4.2.5. To illustrate the voting process from a GUI perspective, the steps that have to be performed by the three GUIs are described below:

1. VA creates a vote
A voting starts with the creation of a vote. The VA must enter a name for the voting and at least one question. As can be seen in Subsections 3.1.3 and 4.1.2, the commands create vote and store question are separated. The VA using the GUI does not notice this, because both actions are executed after clicking on the create vote button.

2. Sealers generate keys
When a vote is created by the VA, all sealers can create the keys. The VA cannot proceed until all sealers have created the keys.

3. VA combines the key shares
When all sealers have created the keys it is possible for the VA to create the public key. This key is quite important because it is responsible for the encryption of the votes.

4. VA changes phase
By default, after a vote is created, the votes are in the key generation phase. However, the VA can now change this phase to the voting phase. On the voter GUI, this change would mean that the vote status is changed from closed to open.

5. Voters vote

Once the voting phase is reached, voters can begin filling out their ballots.

6. VA changes phase

If the VA decides that the time for voting has expired, the VA can change the phase again. In this case, the phase changes from voting to tallying, and the casting of ballots is no longer possible.

7. Sealers make decryption

In this step all sealers who were involved in the key generation must decrypt the shuffled ballots. This means that all sealers have to decrypt the ballot for each question in the vote. If only one sealer fails to decrypt the ballot, the result of the election will never be visible.

8. VA combines decrypted share key

If all sealers have decrypted the ballots, the VA can combine the partial decryptions. This must also be done for all questions. In the GUI perspective, the tallying is now complete.

9. VA gets result

When a question is tallied, it is theoretically possible to see the result. However, the VA GUI prohibits that a result of one question can be displayed. This means that all questions of a vote must be tallied before the VA can see the result.

4.2.3 Database

As described in Subsection 4.2.4, the API communicates not only with the Provotum BC over the client package, but also with a database. This interaction is very important because the database stores information that is needed and used by the GUIs. This is due to the problem that the data sent to the public-permissioned DL is stored, but the client package does not contain commands to retrieve it. The only exception is the result of the vote. Here the GUI gets the information directly from the DL. Storing data in a database contradicts the overarching goal of the Provotum project. The goal of the project is to provide a decentralized voting system. However, this means that part of the system is centralized again. Only information about the vote and the users will be stored in the database. The information about the vote should be available in the future via the public-permissioned DL. The reason is that this information is needed by the GUIs. In contrast, the information about the users is not urgently required. This information is only stored because the Provotum 3.0 Mixnet version does not have an identity management system. For this reason, the prototype developed in this thesis has created users that are best suited for demonstration purposes. Thus, it is possible for a spectator of a demonstration to log in and out with a username and password. Attached is a more detailed explanation of what and how exactly is stored in the database for a vote and for users.

In the case of voting information, it is important to mention that the information is stored in the database only if the action was successful in the public-permissioned DL. The most important element that is stored and used in all GUIs is the vote name. This is a string that is set when the VA has created a vote. Each vote also includes one of the three phases (key generation, voting or tallying). When creating a vote, key generation is the default phase, which is saved in the database. These phases are also stored as strings and Subsection 4.2.2 explains what effects a change of phase can have on the

GUIs. Furthermore, the number of sealers and the names of the sealers are also stored. With this information the GUI is able to see how many sealers have already created the key for this vote. Additionally, a boolean is stored that indicates whether the keys have already been combined for this vote. However, the information about the questions in a vote must not be missing, of course. The question itself is stored as a string, the sealers who have already decrypted the question as a list of strings and whether the decrypted shares have already been combined for the question.

The information about users stored in the database is only relevant for the voter GUI, unlike the vote information. First, of course, the names and passwords of the users are stored as stings. In addition, it is displayed whether a user is online or offline. This information is stored as a boolean value and is used only for logging in and logging out. This prevents multiple people from using the same user at the same time. Another important feature that is stored with each user are the questions that the user has successfully voted on. If an error occurred while submitting the ballot for a question, the voter GUI should also know this as well, so that this error can be corrected by resubmitting the vote for this question. The question is stored as a string, and whether the user voted correctly for this question is stored as a boolean value.

4.2.4 Application Programming Interface

The Provotum 3.0 Mixnet version requires an API so that commands (Subsection 4.1.2) can be executed by the client via the GUIs. For this purpose, REST API requests are used. This means that the API communicates not only with GUIs, but also with the client package of Provotum 3.0 Mixnet version and also with the database. The communication with the client is done in such a way that when a request is received by the GUI, the API moves to the client package with a bash command and there, executes the command for the public-permissioned DL. Some requests of the API communicate only with the client, others only with the database, and others with both, which can be seen perfectly in Figure 4.3. If the API communicates with both, it is important that communication with the database occurs only after communication with the public-permissioned DL has been successful. This is checked in the API with the response that the client receives after executing a command. The users that are created for demonstration purposes are created and stored in the database when the API is started for the first time. However, a check is made to see if the users already exist, and if so, this step is skipped. There are no REST requests to create users as they are only used for demonstration purposes. Therefore, they are created directly in the code, which can be seen in Subsection 5.2.1. So if one wants to add more users, they can be manually entered into the code. However, there are REST requests that are responsible for logging a user in and out.

Figure 4.3 shows an example of the procedure of a vote with the interaction between the API, the GUIs, the public-permissioned DL and the database. The term API in this example refers to the combination between the API and the client package. For simplicity, there is only one sealer and one question in this voting scenario. The first and, in this case, the only question is always created along with the vote. This means that the store_question command does not occur in this example. Furthermore, the responses from the API to the GUI and from the public-permissioned DL to the API are not displayed. Not only that, but the GET requests that the GUIs periodically make to check the status of the public-permissioned DL are also omitted. It should also be noted that logging in and out via the voter GUI is always possible. However, in this example, users only log in and out during the voting phase.

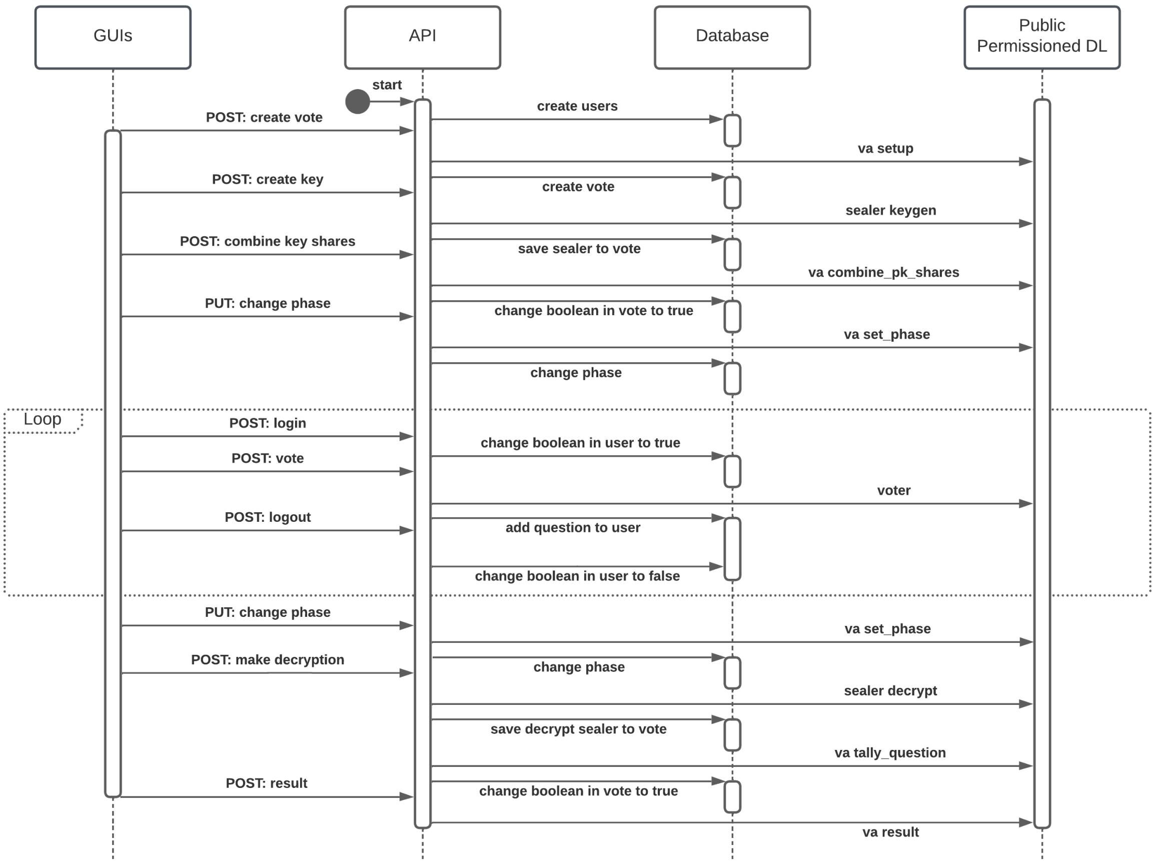


Figure 4.3: Interaction between API, GUIs, Public Permissioned DL and Database

4.2.5 Docker

Subsection 4.1.3 showed the architecture of docker containers and the docker-compose file in the prototype that existed prior to this thesis. It also explained why Docker is being used for the Provotum prototype. A goal of this thesis is to extend the prototype with more Docker containers. This addition can be seen visually in Figure 4.4. Besides the GUI containers, the MongoDB container and the API_CLI have been introduced. The functions of the MongoDB container have already been discussed in the Subsection 4.2.3. The API_CLI container has the peculiarity that this container contains both the API and the client. So it is possible to communicate with this container via requests with the public-permissioned DL (alice, bob, charlie and randomizer), because in this container the commands are executed via requests.

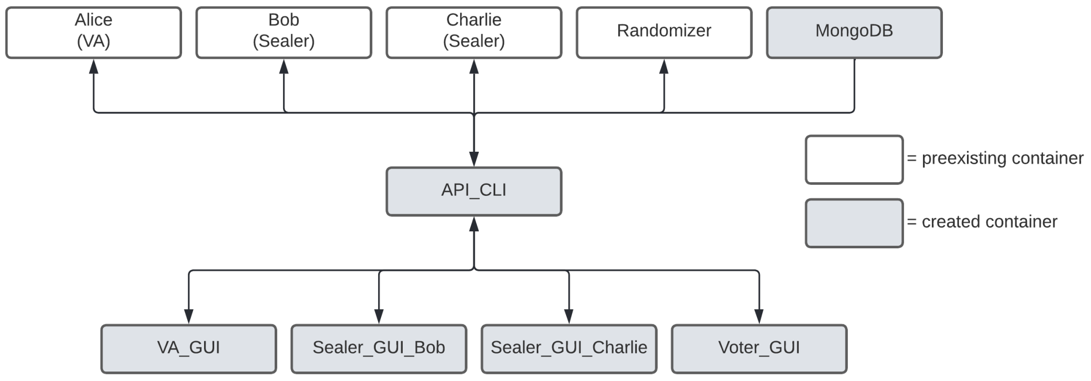


Figure 4.4: Interaction between Docker containers

As shown in Figure 4.4, there are six new containers and two of them are sealer GUI containers. Since the existing Provotum 3.0 Mixnet prototype provided two sealers, the GUIs were implemented as if there were only two sealers. This means that two sealer GUIs are defined in the docker-compose.yml file and in the source code of the API and the VA GUI the number of sealers is set to two. However, the number of containers in the Docker composition file could be increased, as it is possible to add as many sealer containers as desired. To make this possible, it is not only important to add more sealer GUIs, but also to add sealers to the public-permissioned DL. So if the number of sealers increases, the source code of the API and the VA GUI will have to be slightly modified. A possible solution is defined in the Subsection 7.2.1. However, adding a port and sealer name can be done via the docker-compose.yml file. This thesis has solved it in that way that each sealer container has a separate GUI and environment variables, which are used to define the name the sealer and the port of the GUI. Environment variables are set, because this way the GUI can also see what has been defined in the docker-compose file. This means that the sealer GUI knows what name it has been given in the docker-compose file and can display it in the UI. To make this possible, a bash script was created for the GUIs that reads the environment variables inside the containers. After that, the variables are saved in a file and imported into the React application with a script tag inside the head section of index.html [60].

Chapter 5

Implementation

Chapter 4 explained the architecture of the prototype in accordance with the goals of this thesis. To achieve these objectives, a lot of source code had to be added to the prototype. In this chapter relevant parts of the added source code are explained. This means that only the GUIs and the API will be examined in more detail in this chapter. The goal is to ensure that future developers know how the functionality was built in so they can make future changes to the prototype. For this reason, the first part of this chapter presents the technologies used to develop the prototype. The second part deals with the code structure, showing packages of the API and GUIs and explaining important code fragments in more detail. At the end, it is explained how the election is implemented in the prototype.

5.1 Technologies

The GUIs and the API use several technologies. This section serves to briefly describe the different technologies and explain what they are used for in the prototype. However, only the technology itself is explained here and no examples of the prototype's code are shown.

5.1.1 Typescript and Node.js

The API uses the Typescript[1] programming language developed by Microsoft and the open source JavaScript runtime environment Node.js[2]. Typescript is a superset of JavaScript, which means that Typescript is based on JavaScript and therefore any JavaScript code is also a valid TypeScript code. Typescript uses a compiler that converts the TypeScript code into plain JavaScript code. This has the advantage that the compiled code can be used in any environment with JavaScript. One of the most important reasons for using Typescript instead of JavaScript is that Typescript provides additional syntax for supporting types. These types increase productivity by helping to avoid runtime errors [61].

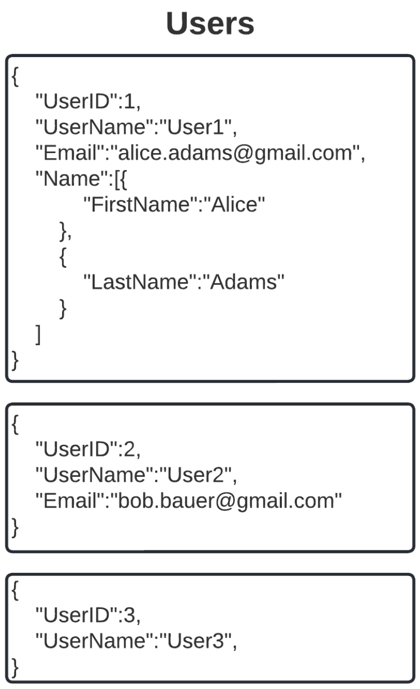
[1]https://www.typescriptlang.org/

[2]https://nodejs.org/en/

Node.js is popular for building APIs as it is known for fast and scalable network applications. This technology is the reason that client applications such as mobile applications or web applications can communicate with a backend service. The reason why Node.js has become so popular is that all APIs developed with Node.js are asynchronous and very fast in code execution. Moreover, these applications are highly scalable and they do not buffer data. However, it is also important to know that Node.js is not only a runtime environment, but also contains a library with multiple JavaScript modules. These modules facilitate the development of web applications [62].

5.1.2 MongoDB and Mongoose

MongoDB[3] is a document-oriented NoSQL database system that serves as the database in the Provotum prototype. A NoSQL document database means that, unlike an SQL database, it stores JSON documents that can vary in its structure. This means that, for example, something is saved for one user that is not displayed as a field for another user. An example of this is shown in the Figure 5.1. This would not be possible with an SQL database. There the fields, which occur only with some users, have to be stored in separate tables. For the example in Figure 5.1, this would mean that there must be two more separate tables, one storing the FirstName, LastName, and UserID, and the other storing the UserID and associated Email [63].



```
{
   "UserID":1,
   "UserName":"User1",
   "Email":"alice.adams@gmail.com",
   "Name":[{
         "FirstName":"Alice"
      },
      {
         "LastName":"Adams"
      }
   ]
}
```

```
{
   "UserID":2,
   "UserName":"User2",
   "Email":"bob.bauer@gmail.com"
}
```

```
{
   "UserID":3,
   "UserName":"User3",
}
```

Figure 5.1: Example for Collection in MongoDB

Mongoose[4] is a library for MongoDB and Node.js [63]. This library is responsible for

[3]https://www.mongodb.com/
[4]https://mongoosejs.com/

the translation between objects in the code and the correct representation in MongoDB. Since there is a communication between the API, which uses Node.js, and the database, which is a Mongo database, in the prototype, Mongoose is used inside the API package. There are several terminologies that keep appearing when working with Mongoose and MongoDB:

Collections	Collections in MongoDB are lists of multiple JSON documents. In the case of the example in Figure 5.1, users would be a collection.
Documents	Documents are the rows in an SQL database. In MongoDB, a document would be a JSON file.
Fields	Fields are the properties in a document. In a SQL database this would be a column.
Schema	A schema in Mongoose defines the data structure of a document. It defines what type (string, boolean, number, ...) a field should have.
Models	A model is the wrapper or constructor of a schema. You can use a model to create, read, or edit documents.

Table 5.1: Mongoose Terminologies, after [63]

5.1.3 React JS

React[5] was used to create the three GUIs in this thesis. React is a JavaScript library developed by Facebook that uses a component-based approach to create declarative UIs. The reason React is a library and not a framework is because React only deals with rendering UIs and leaves many other things to the discretion of individual projects [64].

The main advantages of React are the improved performance, flexibility and scalability of the application. The improved performance results from using a virtual JavaScript DOM instead of a regular DOM. A virtual DOM is a JavaScript object and is faster than the normal DOM. In addition, the use of components ensures the scalability and flexibility of React. This is because these components help developers maintain larger applications. In addition, the components themselves are very easy to expand. One reason for this is also that it is possible to further subdivide these components into classes and functions [65].

When working with React, it is obvious to use the React Router DOM for routing between components. In the case of the Provotum prototype, the BrowserRouter is used because Provotum's GUIs are developed as web applications. This BrowserRouter must be placed in such a way that it encloses the entire app component. In this way it is possible to define the paths and specify, which component must be loaded for a selected path. To prevent a user from accessing all pages via URLs, there exists a concept called private routing. Private Routes or Protected Routes are very easy to implement with React Router. This technology prevents the access of unwanted users to a certain page. Examples would be pages that should not be accessible without login or authetification [66].

With React version 16.8, React introduced hooks as a new feature for functional programming. These hooks allow the use of states and other React functions in a functional

[5]https://reactjs.org/

component. Before version 16.8, this was only possible via class components. There are numerous hooks in React, but only standard hooks were used in the development of the Provotum prototype. The hooks that are used are the state hook and the effect hook. The state hook is used to add local states to a component. When the component is re-rendered, React retains its state. As shown in Listing 5.1, the state hook returns two things, the current state value and the function used to update the state. By default, when declaring a state hook, the state value is given a chosen name and the function is given the same name preceded by a set. Calling the function will cause the state to be updated and the component to be re-rendered. The effect hook is used to perform a side effect. This effect is executed by default on every rendering. However, this hook lets the developer specify that the side effect occurs only when the component is rendered for the first time or when one or more states are updated [67].

```
function ProfilePage() {

    const [username, setUsername] = useState('user1');
    const [likes, setLikes] = useState(29);

    useEffect(() => {
        console.log("Page gets rendered or re-rendered");
    });

    return (
        <div>
            <p>Number of likes: {likes}</p>
            <button onClick={() => setLike(likes + 1)}>
                Like the profile
            </button>
        </div>
  );
}
```

Listing 5.1: React Hooks Example

5.1.4 Redux

As mentioned in Section 5.1.3, the states defined with a state hook are restricted to a local component. This means that all states are lost during a page navigation or refresh. To prevent this, state management libraries are used in web applications. For React, Redux[6] is such a state management library. The official introduction [68] to Redux recommends using the Redux Toolkit as the official approach. The reason for this is that Redux requires a lot of verbose and repetitive code. However, most of this is not necessary when using Redux and has also resulted in many developers having more opportunities to make mistakes. For this reason, the Redux Toolkit was developed, which eliminates exactly this weakness of Redux. The three most important parts in Redux are the store, the actions and the reducers:

[6]https://redux.js.org/

1. Store
Redux manages to make the states usable for all components because it uses a centralized store. This store can be accessed from any component. According to the developer tutorial [69] it is good practice in Redux Toolkit to separate the store into slices. These slices have the advantage that the initial state and the reducers are stored in one object. In addition, all actions are automatically generated based on the reducer functions.

2. Actions
Actions are JavaScript objects that describe that something has happened in the application, respectively the state. They only initiate the reducers and therefore they are not responsible for the change of the state [70].

3. Reducers
Reducers are responsible for changing the state defined by the reducer. They take the current state and an action as arguments, which then leads to a new state [70].

However, to be able to interact with the global store from a React component, React-Redux imports two additional hooks. The selector hook is for reading a global state from the store. The dispatch hook triggers an action in a component, which then causes the reducer in the store to change a state. In Figure 5.2 is an example sequence of how to change a global state from a React component. If you want to trigger an action in the React component, you need to import this action from the corresponding slice. After that, there is the possibility to change a global state by dispatching an action with the dispatch hook. The reducer defined in the slices receives the action and can then change the state in the store. Changing the state in the store updates the React component [71].

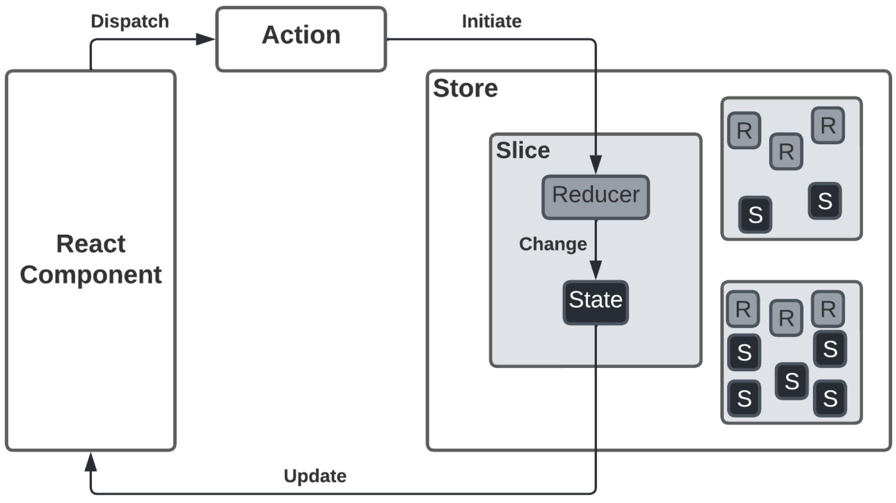


Figure 5.2: Redux Flow Diagram

Redux allows to store global states and display them in all React components using the selector hook. However, without further configuration, all states would be reset to their initial values in case of a page refresh. To prevent this, many web applications use the

Redux Persist Library. This library allows to save the store in the local storage of the browser. This way a user of the web application can refresh the page or even close and reopen it without losing the states. Redux Persistent was also used for the prototype of Provotum [72].

5.1.5 Axios

Axios[7] is a JavaScript library that allows you to make HTTP requests. These HTTP requests are based on promises and are made by the browser or Node.js. An important feature of Axios is that it is able to intercept requests and responses. The reason for using Axios in the Provotum prototype is that, unlike the Fetch API, Axios has a wide browser support [73].

5.1.6 Tailwind CSS

Tailwind CSS[8] is a utility-first CSS framework. This framework was released in May 2019 and has been popular among developers ever since. The reason for this is that, unlike other CSS libraries such as Bootstrap or Material UI, it retains complete control over styling. The reason for this is that Tailwind implements the CSS directly in the HTML under the class tag, as shown in Listing 5.2. This CSS is much simpler than the vanilla CSS because Tailwind provides thousands of built-in classes. Bootstrap, for example, also inserts its components directly in the HTML under the class tag. This will insert the component, but it is quite difficult and tedious to customize this component. So you lose control over styling if you use other well-known CSS libraries. However, Tailwind CSS offers also other advantages as well as disadvantages [74].

As mentioned earlier, control over styling is the biggest advantage of Tailwind CSS. So this is one of the few CSS frameworks that do not dictate how the project should be designed. Another advantage of Tailwind CSS is that the styling process is much faster. With the built-in classes that can be inserted directly into the HTML, the process to a good looking layout is pretty short. These pre-built classes also make it easy to create responsive designs. This also makes Tailwind CSS a mobile-friendly CSS framework [74].

However, direct styling in HTML also has disadvantages. This is because it can happen very quickly that the HTML code becomes quite confusing. Another disadvantage is that it is quite learning intensive. Since it is also quite similar to CSS, it is sometimes difficult for beginners to find the right built-in classes. In addition, Tailwind CSS does not offer styled components. This means that it is necessary to create components such as buttons or navigation bars by oneself. [74].

[7]https://axios-http.com/docs/intro

[8]https://tailwindcss.com/docs/installation

```html
<div class="container px-5 py-10 mx-auto flex">
    <div class="flex w-full">

        <div class="w-full p-10 py-6">
            <h1 class="text-5xl font-medium title-font text-logobrown
                -1000 tracking-wider">Test Title</h1>
            <p class="text-base py-7 text-logobrown-1000">Text</p>
        </div>

    </div>
</div>
```

Listing 5.2: HTML with Tailwind CSS

5.1.7 I18next

I18next [75] is an internationalization framework written and used in JavaScript. The goal of this framework is to overcome language barriers for users of the software. It includes integrations with any frontend frameworks like React, Angular, Vue.js and many more. It is even possible to use i18next on other platforms like Node.js, PHP and so on. I18next was released in 2011, making it an older framework. At the same time, this is also a great strength of i18next, as this framework has been available as open source for a very long time. For this reason, sustainability and maturity is a major advantage of i18next.

The goal of this thesis is to provide a foundation for this technology in the Provotum prototype so that future developers of Provotum can build upon it. For this thesis i18next was used to create a German translation of the text in the GUIs. The reason for this is, on the one hand, that in this way an example can be used to show what a translation can look like. On the other hand, it supports people who do not speak English well and are part of the evaluation (Section 6.4). However, it should be noted that some things have not been translated. For example, the data sent to and received from the API is provided in only one language. So, for example, if the question of a vote was written in English, this question will later be displayed to users in English as well. Also all error messages are in English, because the text of the error messages mostly comes directly from the API.

5.2 Code Structure

In this section the code structures and important code fragments of the Provotum prototype are presented. Only the packages and code fragments created for this thesis are illustrated. Figure 5.3 shows the high-level code structure of the Provotum Mixnet project. Important for this section are the marked packages, because these packages were created within the scope of this thesis.

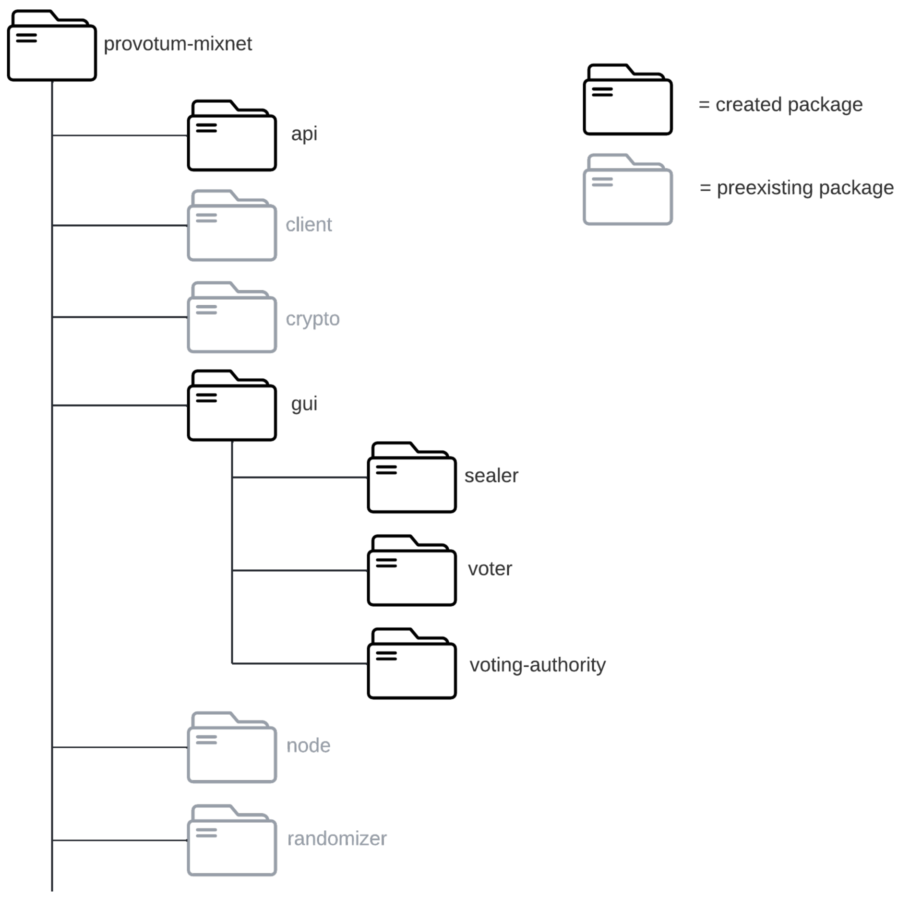


Figure 5.3: Code Structure of the Provotum Mixnet Project

5.2.1 API Packages

As shown in Figure 5.4, the API has five packages. Three of these five packages are named after the phases of the Provotum Voting Protocol (prevoting, voting and postvoting). The other two packages are the helpers package and the mongodb package. What tasks these packages have is explained in the next sections. All packages except the mongodb package consist only of an index.ts file. For this reason, they are not visualized in Figure 5.4. It is important to mention that the API also contains files that are not included in these five packages or in the src package. These packages are mainly for setting up the API. Also, it should be mentioned that some requests are already illustrated in Figure 4.3. This figure shows that the API only offers POST and PUT requests. The reason for this is that the GUIs make the requests with Axios and Axios does not support GET requests with content in the body.

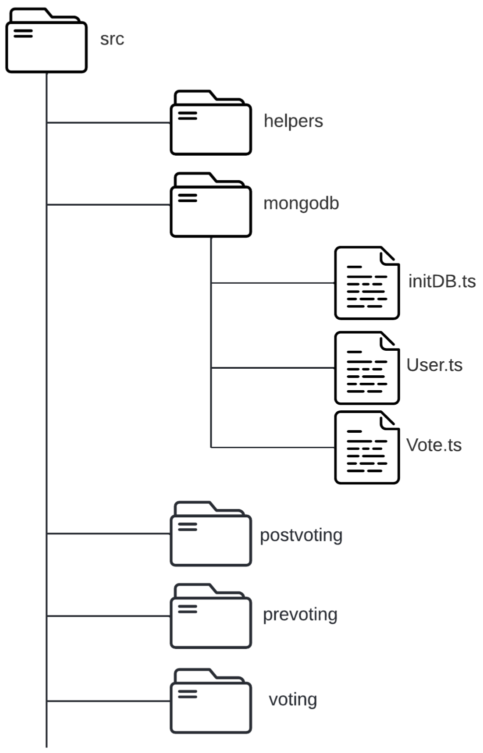


Figure 5.4: Code Structure of the api/src

```
prevotingRouter.post("/setup", (req, res) => {
    const { exec } = require('child_process');

    exec('cd .. && cd client && rustup run nightly-2022-05-20 cargo run
      --release -- va setup --vote "' + req.body.vote + '" --question "
      ' + req.body.question + '"', (error: any, stdout: String, stderr:
       any) => {
        else if (stdout.search("successfully created vote") > 0) {
            res.json(req.body);
            const Vote = require("../mongodb/Vote")

            // Save Vote in db
            const vote = new Vote({vote: req.body.vote, questions: [{
                questionName: req.body.question, decrypted_sealers: [],
                combined_decrypted_shares: false, yes_votes: 0, no_votes:
                 0}], phase: "KeyGeneration", status: "open",
                number_of_sealers: 2, combined_key_shares: false, sealers
                : []})
            vote.save().then(() => console.log("New vote is saved in
                mongoDB"))

        }
        else if (stdout.search("Connection refused") > 0){
            res.status(404);
            res.send("WsHandshake failed. Connection refused")
        }
```

Listing 5.3: Create Vote Request in API

In the **phases packages** (prevoting, voting and postvoting) almost all requests needed for voting have been created. The only exception that is not included in one of these packages is a PUT request that is responsible for changing phases. This request is defined outside the phase packages, as it must be performed in multiple phases. Otherwise, these packages contain all the requests that are required to communicate with the client and with the public-permissioned DL. An example of this would be the POST request ”Create vote”, which is included in the package prevoting, or the POST request vote, which is included in the package voting. In Listing 5.3 part of the source code of the ”Create vote” request can be found. This request is a POST request and it should be sent to the API with the path /prevoting/setup. On line 4 can be recognized the command, which creates the vote in the public-permissioned DL. After that, there are several if conditions that check if the response of the public-permissioned DL contains a certain string. Line 15 to 21 shows what happens when the Vote has been successfully created. In this example, the vote would be stored in MongoDB using the Mongoose technology.

The **helpers package** contains all requests that communicate only with the database. Here are all the requests that the GUIs require but do not receive from the public-permissioned DL. Examples of such requests are ”GetAllVotes”, ”GetAllUsers” or ”GetUserWithUsername”. Some of these request are GET requests, since some requests do not need to have content in the body. In addition, all requests used for demonstration purposes are also located in this package, for example the ”login” and ”logout” requests.

In the **mongodb package** all Mongoose schemas and also the file for initializing MongoDB are implemented. The initialization file starts immediately when the API is launched and tries to connect to the mongo database by Mongoose. If this is successful, all users will be created immediately. It is important to consider that the number of users in this file can be adjusted by adding more users manually in the source code. This thesis created no requests for this procedure, as one of the next goals of the Provotum 3.0 Mixnet is to implement an identity management system. There are also two other files stored in the mongodb package, User.ts and Vote.ts. These files are the Mongoose schemes for a vote and a user. The explanation of a schema is contained in the Table 5.1. Listing 5.4 shows the source code of User.ts. Thereby it is easy to see with which types the fields are stored in the mongo database. Also, it shows the export of the model in line 15.

```
1  import mongoose from "mongoose"
2
3  const votedSchema = new mongoose.Schema({
4      questionName: String,
5      voted: Boolean
6  })
7
8  const userSchema = new mongoose.Schema({
9      name: String,
10     password: String,
11     votedQuestions: [votedSchema],
12     logged_in: Boolean,
13 })
14
15 module.exports = mongoose.model("User", userSchema)
```

Listing 5.4: Source Code of User.ts

5.2.2 GUI Packages

The GUI package in the Provotum Mixnet project is divided into three different packages (sealer, voter and VA). Each of these packages represents a GUI in the Provotum Mixnet prototype. Although the GUIs have different functions and different looks, the code structure in the src folder is the same for all three of them. As can be seen in Figure 5.5, the GUIs have three different packages (components, helpers and redux). The tasks of these packages are explained in more detail in the next sections.

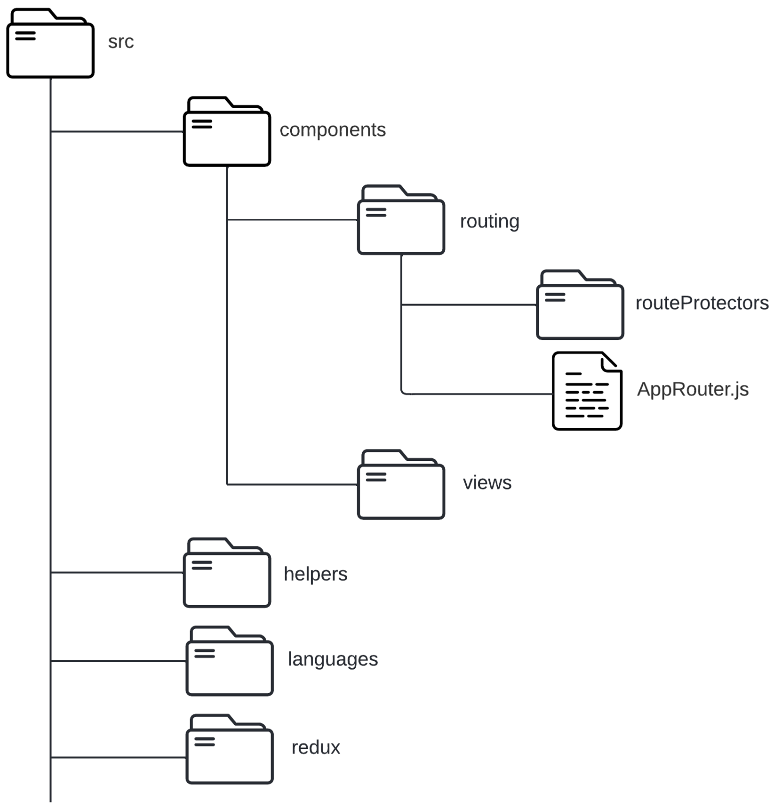


Figure 5.5: Code Structure of the gui/sealer/src

As you can see in Figure 5.5, the **component package** is divided into two more packages. The views package is actually the most important package of the GUIs. The reason is that these packages contain all important React components, which provide almost all functionalities. These package determine, for example, how to create a vote or how to cast a ballot in the specific GUI. Also, almost all Axios requests that interact with the API are defined in this package. It is important to mention that in this thesis all components of the GUI package were developed as function components and not as class components. For this reason, the React hooks are often used in the prototype. Also, all the CSS is defined in this package. Since the GUIs proposed in this thesis use the Tailwind CSS framework, the CSS is not separate from the individual React components. This means that the design, which is heavily influenced by the Swiss postal voting system and older Provotum GUI systems, is implemented in this package. It should also be mentioned that all GUIs are designed to be responsive. Tailwind CSS uses five breakpoints that have a minimum width of 640px, 768px, 1024px, 1280px and 1536px [76]. This means that Tailwind CSS, like other frameworks, uses a mobile-first system, meaning that if a breakpoint is implemented for a particular design, it will only take effect from the specified width and above. The prototype is designed for being used on devices with a screen size up to a minimum width of 320px. For all devices that have a larger width, the predefined breakpoints of Tailwind CSS are used to ensure a suitable design of the GUI. Then there is the routing package, which contains another package and a file. The AppRouter.js file is responsible for routing the user to the correct components. The file is shown in Listing 5.5. The lines 17-20 show how the different components are protected by guards. This principle is called privateRouting and is explained in more detail in the Section 5.1.3. These guards are implemented in the routeProtectors package. For example, they control in which step the state in the Redux store is. This step is of course also dependent on the phase of the vote, which is stored in the database as mentioned. If the gloabl state has stored a specific step in the Redux store, it is not possible to access a component form another step via the URL. This prevents a user from accessing a component that should not be visible yet.

```
12 const AppRouter = () => {
13     return (
14       <BrowserRouter>
15         <Routes>
16           <Route path="/" element={<Navigate to="/voting" />} />
17           <Route path="/createVote" element={<VoteCreationGuard><
                 CreateVote/></VoteCreationGuard>} />
18           <Route path="/keyGen" element={<KeyGenGuard><KeyGeneration
                 /></KeyGenGuard>} />
19           <Route path="/voting" element={<VotingGuard><Voting /></
                 VotingGuard>} />
20           <Route path="/tallying" element={<TallyingGuard><Tallying/></
                 TallyingGuard>} />
21           <Route path="*" element={<Navigate to="/" />}/>
22         </Routes>
23       </BrowserRouter>
24     );
25   };
26
27 export default AppRouter;
```

Listing 5.5: Source Code of AppRouter.js

In the **helpers package** there are all files and components that should support the main components in the view package. For example, the Provotum and Github logos are stored in this package and used in the header. Furthermore, the step displays are also defined in the helpers package. The reason for this is that this way a better overview can be obtained in the main components in the view packages. For this reason, header and footer are also implemented in the helpers package, which are the same in each component of a GUI. This means, for example, that all components of the sealer GUI have the same header, but a different header than the voter GUI. The header also indicates if the GUI has a connection to the public-permissioned DL, which is checked with an interval. Since each time a request must be made anyway, the phase of the votes is also always retrieved. This has the advantage that it is not possible to have a different phase in the GUIs than in the public-permissioned DL.

The **languages package** contains all the JSON files that store all the translations for the GUIs. This means that there is a separate JSON file for each language. With i18next, which is explained in Section 5.1.7, it is the case that all translations must be stored in the index.js file. However, to keep it organized, the documentation of react-i18next[9] recommends creating a separate i18next.js file beside the index.js. This file contains the configuration of i18next and all translations. The documentation also provides the advice that all translations should be moved to separate JSON files and then be imported. This is also the way how the Provotum prototype was implemented. The language package shown in Figure 5.5 contains exactly these JSON files.

The **redux package** contains all relevant components of the Redux technology. This means that the store and all associated slices are located in this package. This also gives an overview of which states are stored globally and which are only stored locally. In Listing 5.6 there is an example of the source code of a slice. The tasks of a slice are described in the Section 5.1.4. Line 6 shows an instance of a global state. This state is changed by the reducer, which can be found in lines 9-11. On line 16 it is possible to observe that the function of the reducer is exported as an action. The reason for this is that it must be possible to dispatch the actions in the other components by the dispatch hook from Redux.

```
3  export const ChainSlice = createSlice({
4    name: 'chain',
5    initialState: {
6      status: "OFF CHAIN"
7    },
8    reducers: {
9      setChainStatus: (state, action) => {
10         state.status = action.payload;
11     },
12   }
13 })
14
15 // Action creators are generated for each case reducer function
16 export const { setChainStatus } = ChainSlice.actions
17
18 export default ChainSlice.reducer
```

Listing 5.6: Source Code of ChainSlice.js

5.3 Elections

The Provotum 3.0 Mixnet version also allows to create elections. The reason for this is that in this version the HE-based tallying component has been replaced by a Mixnet-based approach [10]. This means that a vote is no longer limited to zero or one. However, this

[9]https://react.i18next.com/guides/quick-start

change was made to allow elections, but the entire prototype of Provotum is designed for voting. This can be observed, for example, in the naming of the phases (pre-voting phase, voting phase and post-voting phase) or in the parameters for the creation of a vote or election (vote, question). There is no command that creates an election and stores it in the public-permissioned DL. Thus, even if the VA creates an election, it is treated as a vote by the public-permissioned DL. For this reason, the chapters of this thesis mainly describe the architecture and implementation of the prototype for the use of votes. However, this section is dedicated only to the election and the way it was implemented in the prototype of this thesis.

The election was implemented in such a way that a Swiss parliamentary election would be possible, which was described in Section 2.5. One thing that is not yet implemented in the Provotum 3.0 Mixnet prototype are elections that include a handwritten name of a candidate. This would be technically possible according to Eck's thesis [10], but has not yet been implemented. For this reason, this is not yet present in the prototype of this thesis. In addition, apparentment was also not implemented in the prototype. Apparentment would allow parties to establish electoral alliances. However, everything else like panachage and cumulative voting was implemented. In the process, three new fields were stored in the database and a new request was added to the API. An additional field in the database is an array with all participants in a party list. This is submitted to the API along with the number of seats, which must be provided by each canton. This request interacts only with the database and not with the public-permissioned DL, which means that it can also be found in the helpers package in the API. Another field in the database indicates whether a user has already participated in the election. This field is not changed by a request, but it is checked for each voting request to see if it is the last possible vote. If this is the case, the field will be changed. In the React code, the names of the data types were not changed. The reason is that in the public-permissioned DL it makes no difference whether it is a vote or an election. Therefore the election name is stored under the data type name ”vote” and a party name under the data type name ”question”. In addition, there is also a list of 36 different numbers. Each candidate has assigned such a number on the party list, with which the request for voting is made. These numbers are important because in this way it is possible to find the right candidate in the party, if the result is received. These numbers are also tested for the client commands. Some numbers do not work with these commands and the requests will therefore never reach the public-permissioned DL. Furthermore, there are only 36 numbers because the canton of Zurich is the most populated canton and can only send 36 people to the parliament. Therefore, parties in the canton of Zurich are also only allowed to have a maximum of 36 party members nominated. It should also be noted that the GUI can only display six-digit numbers as a result. So if a candidate gets more than 9’999’999 votes, the result cannot be displayed correctly. However, since the system was introduced for the Swiss electoral system, this will not be the case.

Chapter 6

Evaluation

The main focus of this thesis is the implementation of GUIs for vote administration and vote casting. For this reason, this chapter evaluates these GUIs. This chapter begins with a heuristic evaluation that has a reputation for finding the most serious usability problems [77]. In addition, a use case analysis is performed to explain the process within the GUIs from the perspective of the different stakeholders. Subsequently, a discussion is presented that shows the privacy and verifiability properties in the prototype and how the GUIs affected them. At the end of this chapter, the results of the system usability scales are presented. Then it was within the scope of this thesis to survey 15 people in order to be able to evaluate the usability of the GUIs.

6.1 Heuristic Evaluation

In this section, the usability properties of the prototype are evaluated. This evaluation is performed with a heuristic evaluation (Subsection 2.4.4) for each GUI. There are also tables with the most important points and a score between one and five, which indicates if the heuristic is fulfilled in the specific GUI. A score of one means that this heuristic is not fulfilled at all, and a score of five indicates that this heuristic is completely fulfilled.

6.1.1 Voting Authority GUI

H1 There are several reasons why users are always informed about the current state of the UI. First, there is a vertical stepper that always displays the current phase in which the user is in. In addition, the header contains the status of the chain itself. This allows the user to check at any time whether the UI is connected to the Provotum BC. The user also always receives visual feedback immediately after pressing a button. This can be, for example, a disabled load button or a modal that appears. The effect that was desired should also be directly visible after loading. This could be, for example, a message that the creation was successful.
H2 The VA GUI is a UI that should only be used by trained administrators [78]. For this

ID	Heuristic	Score
H1	Visibility of system status	5
H2	Match between system and real world	4
H3	User control and freedom	3
H4	Consistency and standards	5
H5	Error prevention	4
H6	Recognition rather than recall	5
H7	Flexibility and efficiency of use	2
H8	Aesthetic and minimalist design	4
H9	Help users recognize, diagnose and recover from errors	3
H10	Help and documentation	3

Table 6.1: Heuristic Evaluation for VA GUI

reason, there are no explanations of what, for example, the purpose of a sealer is or how the Swiss voting system works when creating an election. The UI also uses real-world icons, such as trash cans, lists, or keys, which makes the UX slightly easier. However, since it can be assumed that only trained administrators use this GUI, the language of the UIs could be more technical.
H3 It is quite difficult to give complete freedom to the VA, because the UI also works with Provotum's Voting Protocol Phases. This means, for example, that the user cannot return to the CreateVote component after creating a vote in the GUI. However, a warning is displayed before each phase change so that the user knows that it cannot be undone. Besides that, the user has complete freedom inside a phase. So up to now, it is the case that one button starts and another button ends the voting phase. This could be a problem in a real-life scenario, as it would be possible to unintentionally start or end the vote too early or too late. One solution would be to require a start and end date and time when creating a vote at the beginning.
H4 The same design language and color palette were used for all UIs. This is also the reason why the different GUIs do not differ much. However, to quickly identify which GUI a person is interacting with, different colors have been used. For this the same color palette was used as for the Provotum logo. The colors red and brown were used for the VA GUI, dark blue for the sealer GUI and light blue for the voter GUI.
H5 Any action that can have more severe consequences must be confirmed by the user. However, the system is designed to avoid most errors. However, there are still errors that can occur because of the API or the inner workings of the Provotum BC.
H6 The voter does not have to remember any information from one part of the system to another. All important information, that is still used in the system, is stored either in the global store or in the database.
H7 An attempt has been made to increase the efficiency of the UI by using not only input fields, but also simple dropdowns. The standard shortcuts can also be used, as these are given by the respective operating system. However there is no flexible and customizable UI arrangement.
H8 The most important information of a content is usually highlighted in a colored box in the middle of the page. However, what can be a bit confusing is the fact that the instructions are always visible beneath the title.
H9 Most error messages suggest a solution. However, some of these messages come directly

from the API, which means that they are partially written in technical terms. Moreover, for this reason, they are written only in English.
H10 Each component has a guide underneath the page title with the most important information on how to use the UI. However, this guide is written in text form, which makes it a bit more difficult for the user to follow the instructions. A possible improvement would be a guide that shows step by step what to do. One component explains not only how to use the UI, but also why this action needs to be performed. This could be perceived as unnecessary information and the documentation could therefore be considered too extensive.

6.1.2 Sealer GUI

In terms of design, the sealer GUI is very similar to the VA GUI. Therefore, there are many similarities in the heuristics. In the following section, only the heuristics that differ from the heuristic evaluation by the VA are explained.

ID	**Heuristic**	**Score**
H1	Visibility of system status	5
H2	Match between system and real world	4
H3	User control and freedom	1
H4	Consistency and standards	5
H5	Error prevention	1
H6	Recognition rather than recall	5
H7	Flexibility and efficiency of use	3
H8	Aesthetic and minimalist design	4
H9	Help users recognize, diagnose and recover from errors	3
H10	Help and documentation	3

Table 6.2: Heuristic Evaluation for sealer GUI

H3 Sealers have no freedom, as they are bound by the phases of the vote and therefore by the VA. This means that the VA is responsible for a change to another phase. In this case, the sealer GUI automatically changes the phase without the user being informed beforehand. In addition, all actions performed by the sealers interact with the public-permissioned DL. This means that such an action cannot be undone, as the Provotum BC does not allow deleting data [78].
H5 In the sealer GUI it happens very often that an error message appears. This occurs mostly during decryption. The reason for this are the inner workings of the Provotum BC. The BC is designed in such a way that a considerable time must elapse between decryptions. But not only between the decryption, but also between the phase change and the first decryption. For this reason, a timeout of 6 seconds has been implemented in the GUI when the user starts a decryption. This is also the reason why it takes a long time to decrypt all questions. But even with these six seconds, the problem is not completely solved. This issue will have to be solved in a future work on the inner workings of the Provotum BC.

H7 As with the VA GUI, there is no flexible and customizable UI arrangement. However, this GUI is optimal for efficiency because it has no input fields; only buttons that must be pressed.
H10 Since only buttons have to be pressed in the sealer GUI and the use of the UI is therefore self-explanatory, the text under the page title only explains why the actions have to be performed. The reason for mentioning the reason is that the SUS respondents know why they need to activate this button. This should lead them to not perceive the system as complex. Furthermore, this information can be helpful in a demonstration.

6.1.3 Voter GUI

The heuristics **H4**, **H6**, **H8** and **H9** do not differ from the heuristic evaluation of the VA GUI. Therefore, these will not be discussed further in the next section.

ID	**Heuristic**	**Score**
H1	Visibility of system status	5
H2	Match between system and real world	3
H3	User control and freedom	5
H4	Consistency and standards	5
H5	Error prevention	3
H6	Recognition rather than recall	5
H7	Flexibility and efficiency of use	3
H8	Aesthetic and minimalist design	4
H9	Help users recognize, diagnose and recover from errors	3
H10	Help and documentation	4

Table 6.3: Heuristic Evaluation for voter GUI

H1 The voter GUI is also similar to the VA GUI in that it also includes a chain status in the header or buttons to load. Unlike the other GUIs, the Voter GUI can only perform actions related to the Provotum BC in the voting phase. This means that the status of the vote is also displayed. For orientation during a vote, the voter GUI also provides a vertical stepping.
H2 This UI uses clear titles and a natural language. This is because, unlike the other GUIs, this UI is intended for users who have not yet worked with a REV. However, no real-world icons are used.
H3 This GUI gives users complete freedom and control. The users can undo any action. There is one exception, however, and that is when casting a vote. Once the vote has been submitted, the user can no longer change it. The reason for this is that, according to Swiss law [79], each person eligible to vote may only vote once. In addition, the voter cannot fill out a ballot if Provotum is not in the voting phase. However, these limitations are intentional.
H5 As with the VA GUI, any action that can have more fatal consequences must be confirmed by the user. However, the system should also be designed in such a way that no error messages occur. However, as with the sealer GUI, there is also a problem due to

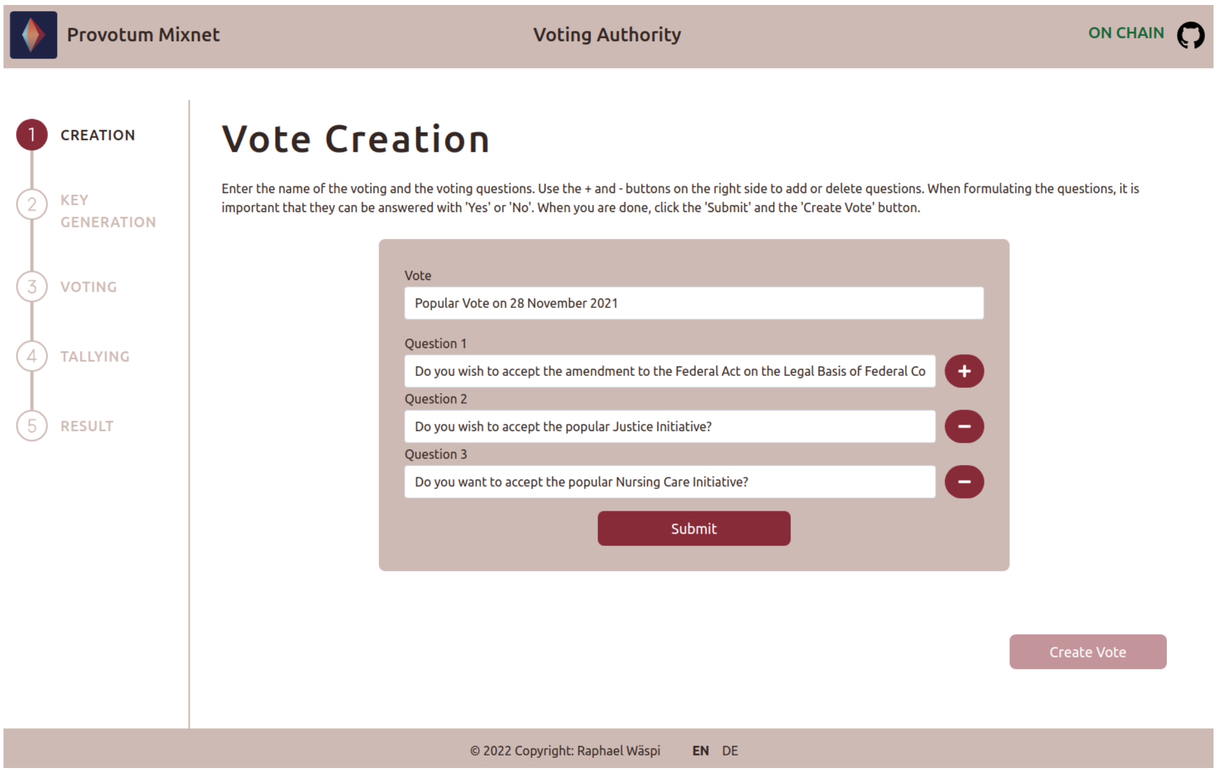
the inner workings of Provotum BC. The problem is that multiple votes cannot be carried out at the same time. For this reason, a timeout of 6 seconds was also implemented in the voter GUI when the user casts a vote. With this method, the error message occurs very rarely during a demonstration. However, this method can only be used to solve a demonstration, not a real-world scenario. If more than one voter tries to cast his vote at the same time, this can lead to major issues. This problem will have to be solved in a future work on the inner workings of the Provotum BC.
H7 To make ballot filling more efficient, the voter GUI uses checkboxes instead of input fields for votes and dropdowns for elections. Also, a keydown event has been added to the sign in function, which allows the user to log in with a click on the enter key. This reduces the susceptibility to invalid ballots. But even in this GUI there is no flexible and customizable UI design.
H10 Each component has an instruction with the most important information under the page title, which is also written in text form. However, unlike the others, the voter GUI does not provide any additional information about the reasons for performing the actions.

6.2 Use Case Analysis

This section presents use case scenarios involving the VA, the sealers, and the voters as stakeholders. Each scenario describes how the process in the software proceeds until the goal is reached. Various figures of the GUIs are shown for further illustration. In addition, the main existing problems of the proposed prototype that could not be solved yet are mentioned in the respective scenarios.

6.2.1 Voting Authority

Creating Votes: The creation of a vote is one of the most important tasks of the VA. In this process, the Provotum BC allows to add any number of questions to each vote. As shown in Figure 6.1, this is also possible via the VA GUI. After entering the name of the vote and the questions, they can be submitted in the GUI. Technically, this means that the entries are stored in Redux, but not yet in the public-permissioned DL. The reason for this is that the users can check the entries afterwards and also edit them if necessary, as illustrated in Figure 6.2. However, if users are satisfied with the input, they can create the vote. A warning is displayed during this creation and the VA must confirm that he really wants to create the vote with these entries. The reason for this is that the entries cannot be changed once the vote has been created, as they are stored in the public-permissioned DL.



Vote Creation

Enter the name of the voting and the voting questions. Use the + and - buttons on the right side to add or delete questions. When formulating the questions, it is important that they can be answered with 'Yes' or 'No'. When you are done, click the 'Submit' and the 'Create Vote' button.

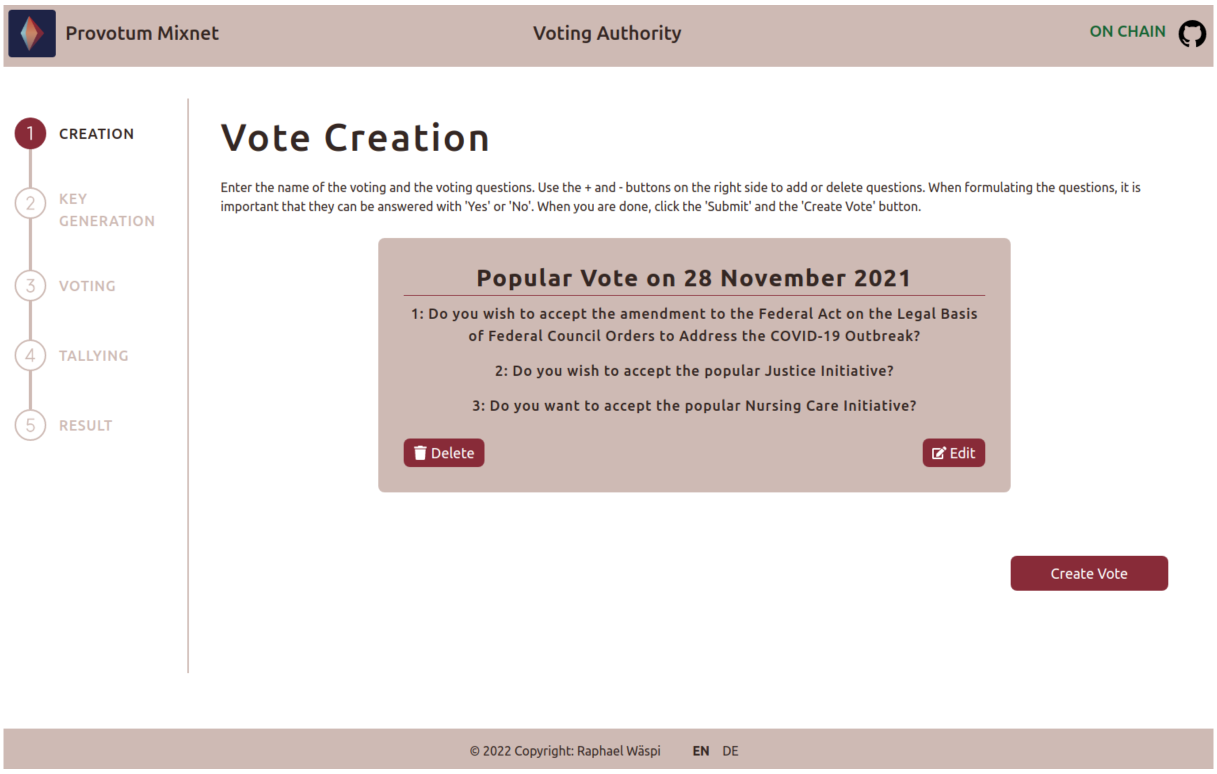
Vote: Popular Vote on 28 November 2021

Question 1: Do you wish to accept the amendment to the Federal Act on the Legal Basis of Federal Co

Question 2: Do you wish to accept the popular Justice Initiative?

Question 3: Do you want to accept the popular Nursing Care Initiative?

Figure 6.1: Creation of a vote



Vote Creation

Enter the name of the voting and the voting questions. Use the + and - buttons on the right side to add or delete questions. When formulating the questions, it is important that they can be answered with 'Yes' or 'No'. When you are done, click the 'Submit' and the 'Create Vote' button.

Popular Vote on 28 November 2021

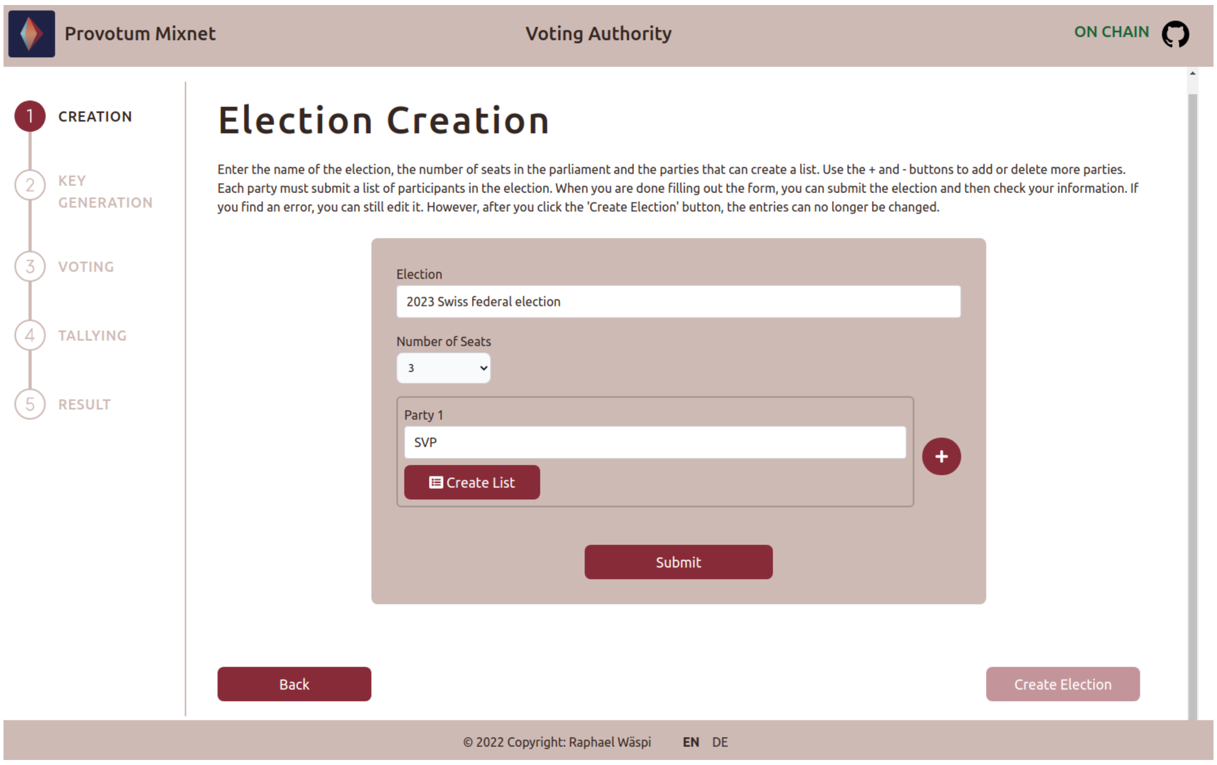
1: Do you wish to accept the amendment to the Federal Act on the Legal Basis of Federal Council Orders to Address the COVID-19 Outbreak?

2: Do you wish to accept the popular Justice Initiative?

3: Do you want to accept the popular Nursing Care Initiative?

Figure 6.2: Verification of inputs when creating a vote

Creating Elections: Since the Provotum 3.0 Mixnet version, multi-way elections are possible. For this reason, elections were introduced in the GUIs. It was implemented in such a way that a Swiss parliamentary election is possible. As shown in Figure 6.3, the VA can use the UI to enter an election title, the number of seats, the parties, and the parties' candidate lists. The idea is that the VA represents a canton that has a certain number of seats in the National Council. This allows the VA to write as many participants on a list for each party as it has seats in the National Council. After filling in the mandatory fields, the VA can check the entries and edit them again in case of mistakes. However, if the VA is satisfied with all the input, the election can be held. As with the creation of the vote, a warning appears.



Election Creation

Enter the name of the election, the number of seats in the parliament and the parties that can create a list. Use the + and - buttons to add or delete more parties. Each party must submit a list of participants in the election. When you are done filling out the form, you can submit the election and then check your information. If you find an error, you can still edit it. However, after you click the 'Create Election' button, the entries can no longer be changed.

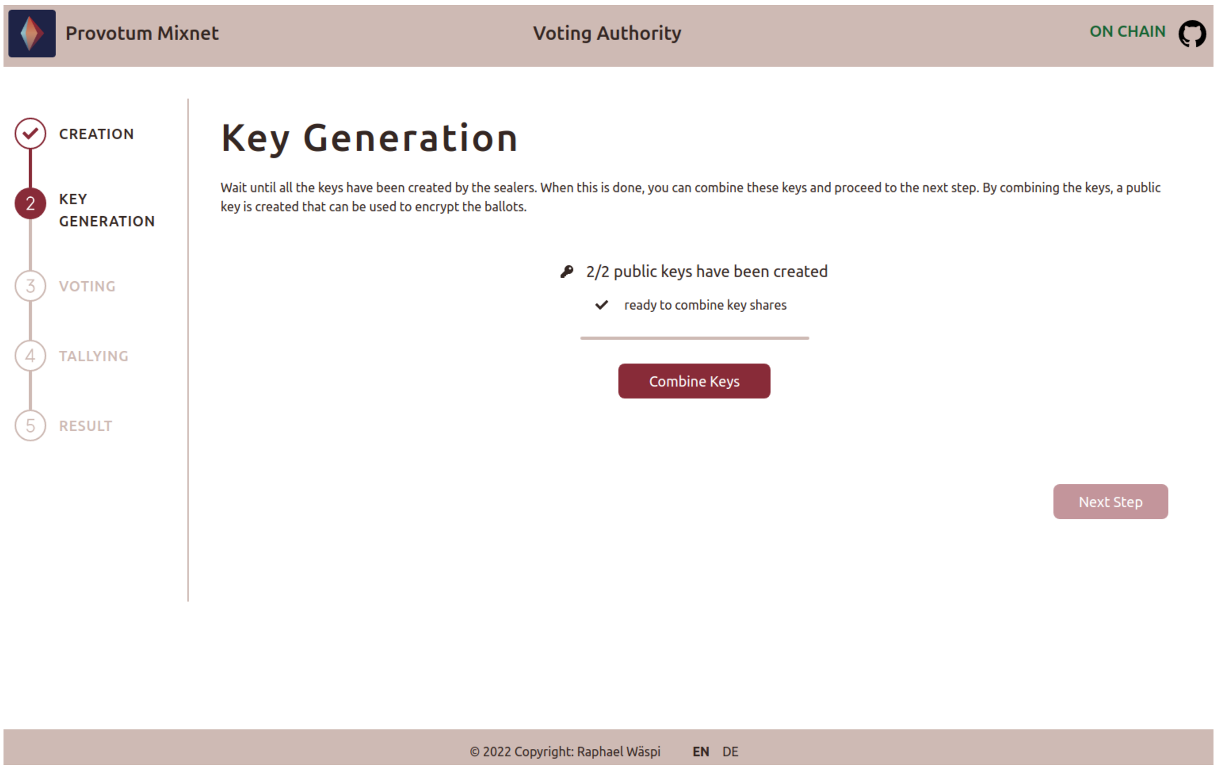
Election: 2023 Swiss federal election

Number of Seats: 3

Party 1: SVP

Figure 6.3: Creation of an election

Opening Votes: After the vote has been created, it is time to open the voting. The VA must first wait until all sealers have created their keys. Once the sealers have generated their keys, the VA can combine the key shares into a public key for voting. This process would now be feasible in the example of the Figure 6.4. After the key shares have been correctly combined, the VA GUI allows the VA to proceed to the next step. In the process, the phase in the Provotum BC changes from the Key Generation phase to the voting phase.



Key Generation

Wait until all the keys have been created by the sealers. When this is done, you can combine these keys and proceed to the next step. By combining the keys, a public key is created that can be used to encrypt the ballots.

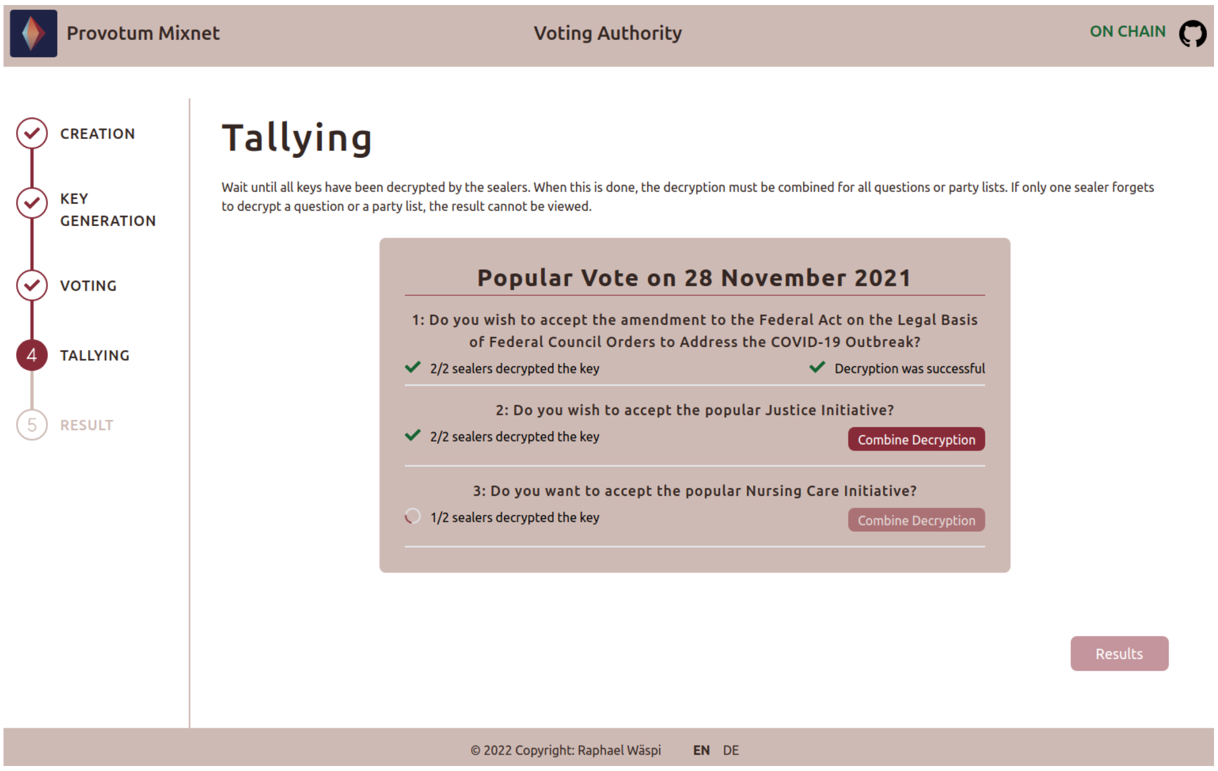
2/2 public keys have been created

ready to combine key shares

© 2022 Copyright: Raphael Wäspi

Figure 6.4: Combining key shares to one public key

Closing Votes and Tallying: During the voting phase, voters can cast their ballots. However, the VA may terminate the vote at any time, which means that the phase changes from the voting phase to the tallying phase. This process cannot be undone and the voters are not allowed to cast their ballots afterwards. For this reason, as with the creation of votes, a warning appears for the VA so that this action does not occur accidentally. In the tallying phase, the VA must first wait until all sealers have decrypted the questions with their keys. Once this is done, the VA can combine the decrypted ballots. In Figure 6.5 these steps can be seen well. In the first question, the ballots have already been correctly combined, and in the second question it is possible to start the combination. For the third question, the VA still waits until the last sealer has decrypted the question with the key. However, it is important to include at least one vote for each question (or for each party). If this is not the case, it could lead to problems in the tallying, since the inner workings of the Provotum BC do not allow decryption if a question does not contain a vote. In voting, this problem is solved if only one voter casts the ballot, because even if this user leaves all questions blank, the voter still casts a vote for each question in the BC. In an election, this is not enough because each voter can only cast a number of votes determined by the canton. Therefore, when demonstrating this prototype, it is important to note that each party received at least one vote.



Tallying

Wait until all keys have been decrypted by the sealers. When this is done, the decryption must be combined for all questions or party lists. If only one sealer forgets to decrypt a question or a party list, the result cannot be viewed.

Popular Vote on 28 November 2021

1: Do you wish to accept the amendment to the Federal Act on the Legal Basis of Federal Council Orders to Address the COVID-19 Outbreak?
✓ 2/2 sealers decrypted the key ✓ Decryption was successful

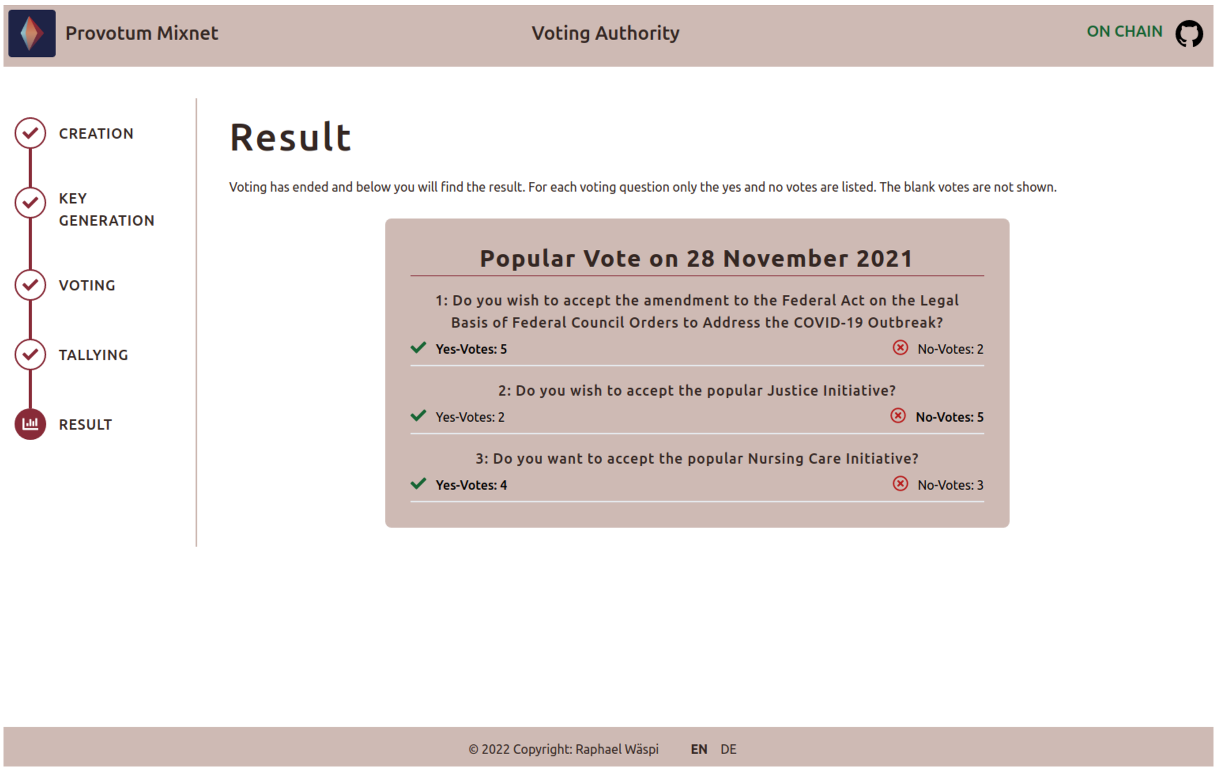
2: Do you wish to accept the popular Justice Initiative?
✓ 2/2 sealers decrypted the key

3: Do you want to accept the popular Nursing Care Initiative?
1/2 sealers decrypted the key

© 2022 Copyright: Raphael Wäspi

Figure 6.5: Combining decrypted ballots

Inspecting Results: Only when all questions have been decrypted by the sealer and the VA has combined the ballots, it is possible to view the results. As shown in Figure 6.6, the yes or no votes are highlighted in bold font to indicate what the majority voted for. In case of a tie, both are marked in bold. In an election, the results can also be viewed only after all ballots have been combined by the VA. As illustrated in Figure 6.7, the result lists all parties with the members that received a vote. The number in parentheses next to the party name is the number of party votes. Party members always have candidate votes next to them. How these votes result is explained in the Section 2.5.



Result

Voting has ended and below you will find the result. For each voting question only the yes and no votes are listed. The blank votes are not shown.

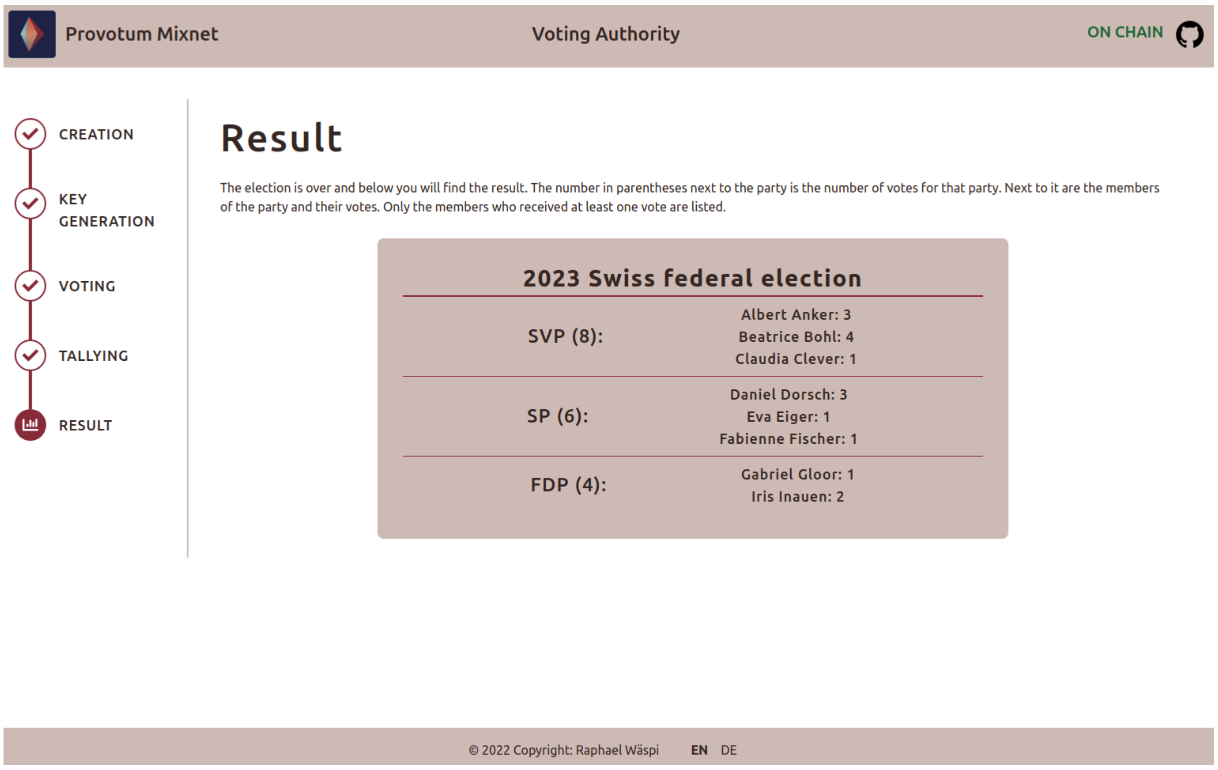
Popular Vote on 28 November 2021

1: Do you wish to accept the amendment to the Federal Act on the Legal Basis of Federal Council Orders to Address the COVID-19 Outbreak?
✓ **Yes-Votes: 5** ⊗ No-Votes: 2

2: Do you wish to accept the popular Justice Initiative?
✓ Yes-Votes: 2 ⊗ **No-Votes: 5**

3: Do you want to accept the popular Nursing Care Initiative?
✓ **Yes-Votes: 4** ⊗ No-Votes: 3

Figure 6.6: Inspecting result of a vote



Result

The election is over and below you will find the result. The number in parentheses next to the party is the number of votes for that party. Next to it are the members of the party and their votes. Only the members who received at least one vote are listed.

2023 Swiss federal election

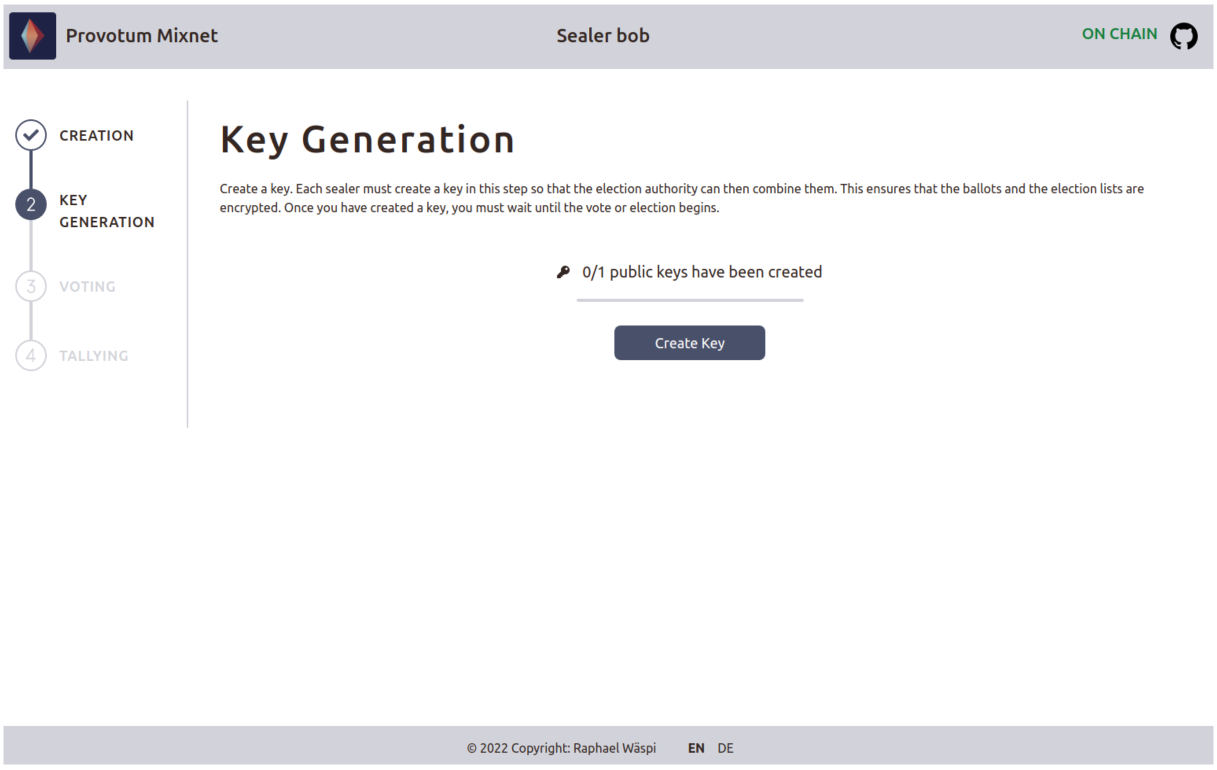
SVP (8):	Albert Anker: 3 Beatrice Bohl: 4 Claudia Clever: 1
SP (6):	Daniel Dorsch: 3 Eva Eiger: 1 Fabienne Fischer: 1
FDP (4):	Gabriel Gloor: 1 Iris Inauen: 2

© 2022 Copyright: Raphael Wäspi EN DE

Figure 6.7: Inspecting result of an election

6.2.2 Sealer

Generate Keys: After the VA created a vote or election, all sealers must generate a public key. As shown in Figure 6.8, only one button needs to be pressed for this action. If the button is disabled, it means that the vote or election has been created, but not all questions or parties have been prepared by the VA yet. This means that the sealers have to wait a few seconds before they are able to create the key. Furthermore, the sealer GUI has no function that allows to change a phase or to make a next step. It is totally dependent on the phases in the public-permissioned DL managed by the VA. This means that the sealer is automatically directed to the correct component.



Key Generation

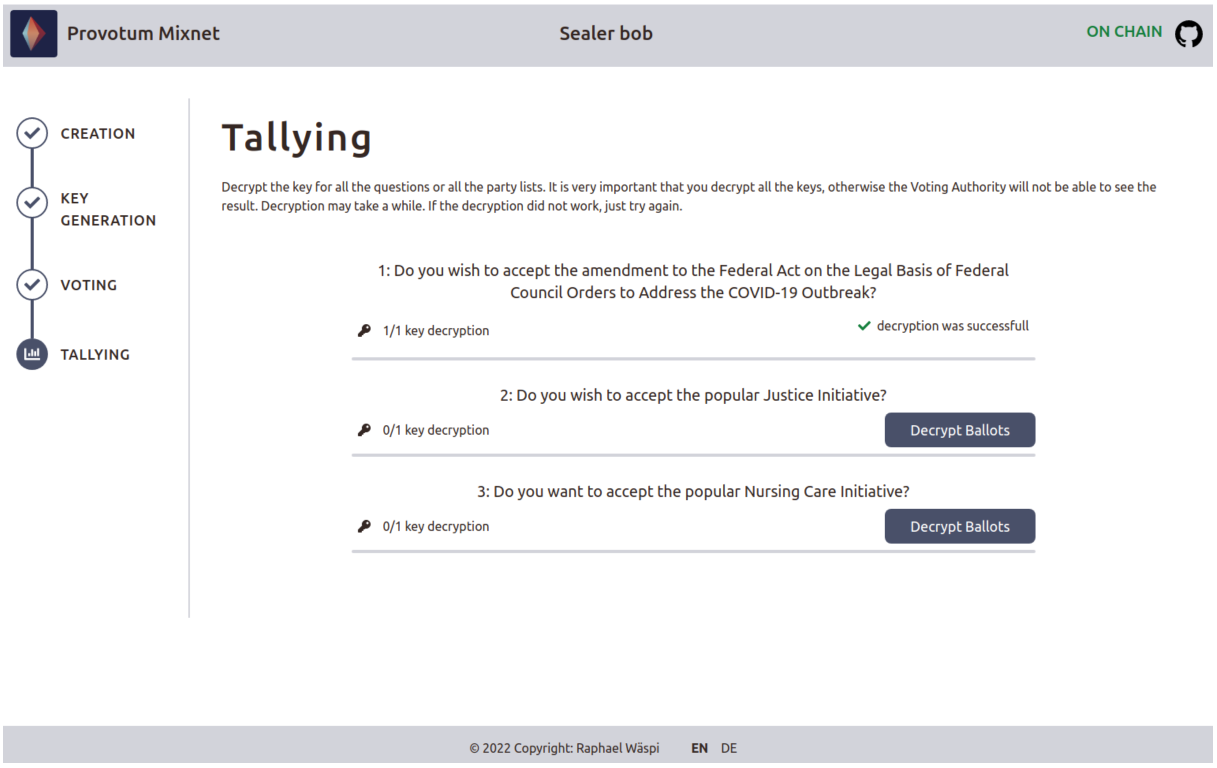
Create a key. Each sealer must create a key in this step so that the election authority can then combine them. This ensures that the ballots and the election lists are encrypted. Once you have created a key, you must wait until the vote or election begins.

0/1 public keys have been created

© 2022 Copyright: Raphael Wäspi

Figure 6.8: Key generation by the sealers

Decryption of Ballots: After the VA finishes the voting phase, all sealers who participated in the key generation must decrypt the shuffled ballots. As demonstrated in Figure 6.9, the sealer must press a button for all ballot questions, which then automatically decrypts the ballots. After that, all the tasks of the sealers are done. However, it should be noted that error messages often appear during decryption. The reason for this also lies in the inner workings of the Provotum BC. A certain amount of time must elapse between the decryptions, so that both can be processed by the BC. But time must pass not only between decryptions, but also between the phase change and the first decryption. For this reason, it is recommended to wait a few seconds after the end of voting before starting to decrypt the ballots. A waiting time of 6 seconds has been set up by the sealer GUI between the decryptions, so that fewer the error messages appear. However, depending on the number of questions, this may result in longer waiting times.



Tallying

Decrypt the key for all the questions or all the party lists. It is very important that you decrypt all the keys, otherwise the Voting Authority will not be able to see the result. Decryption may take a while. If the decryption did not work, just try again.

1: Do you wish to accept the amendment to the Federal Act on the Legal Basis of Federal Council Orders to Address the COVID-19 Outbreak?

1/1 key decryption ✓ decryption was successfull

2: Do you wish to accept the popular Justice Initiative?

0/1 key decryption Decrypt Ballots

3: Do you want to accept the popular Nursing Care Initiative?

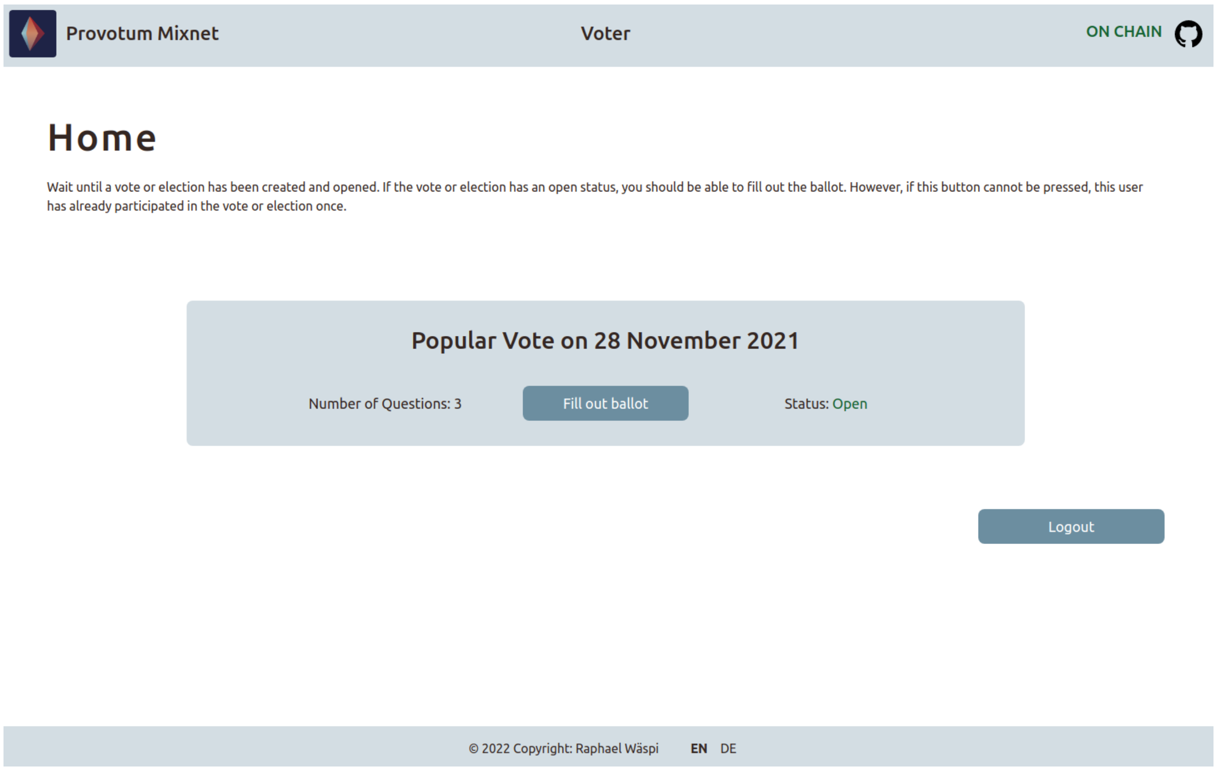
0/1 key decryption Decrypt Ballots

© 2022 Copyright: Raphael Wäspi EN DE

Figure 6.9: Ballot decryption by the sealers

6.2.3 Voter

Getting a Ballot: Unlike other versions, the Provotum 3.0 Mixnet version does not yet have an identity management. However, the voter GUI uses a login process. To log in, the voter must use a username and password that were created when the API was launched. Once a user is successfully logged in, the voter GUI home page will appear. On this component the votes and elections are displayed and also what status they have. If the status is open as in Figure 6.10, the user can open the ballot by pressing the button ”Fill out Ballot”.



Home

Wait until a vote or election has been created and opened. If the vote or election has an open status, you should be able to fill out the ballot. However, if this button cannot be pressed, this user has already participated in the vote or election once.

Popular Vote on 28 November 2021

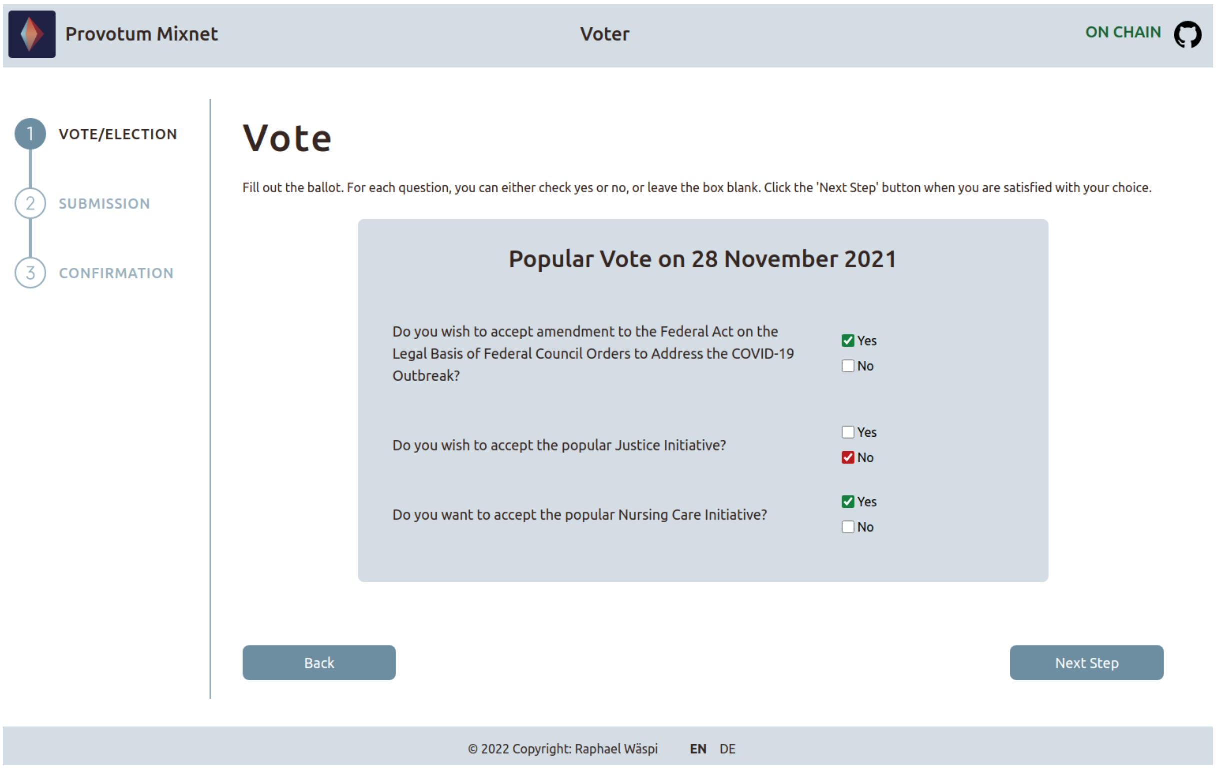
Number of Questions: 3

Status: Open

© 2022 Copyright: Raphael Wäspi

Figure 6.10: Homepage after logging in to the voter GUI

Filling out the Ballot: Once the voters have opened the ballots, they may directly start filling it out. There is a difference between whether it is a vote or an election. If it is a vote, the users can either vote yes or no on each question or leave the question empty. This is completed with checkboxes, as shown in Figure 6.11. In an election, as shown in Figure 6.12, it is possible to choose a party list and also candidates. It is important to note that the candidates of all parties are displayed, as it is possible in the Swiss electoral system to panache. However, each person can be selected no more than twice. The lists and candidates are displayed with dropdown menus. It is also possible to choose an empty list or an empty person. However, each voter must have at least one person on the list to proceed with the election.


Provotum Mixnet

Voter

ON CHAIN

1 VOTE/ELECTION

2 SUBMISSION

3 CONFIRMATION

Vote

Fill out the ballot. For each question, you can either check yes or no, or leave the box blank. Click the 'Next Step' button when you are satisfied with your choice.

Popular Vote on 28 November 2021

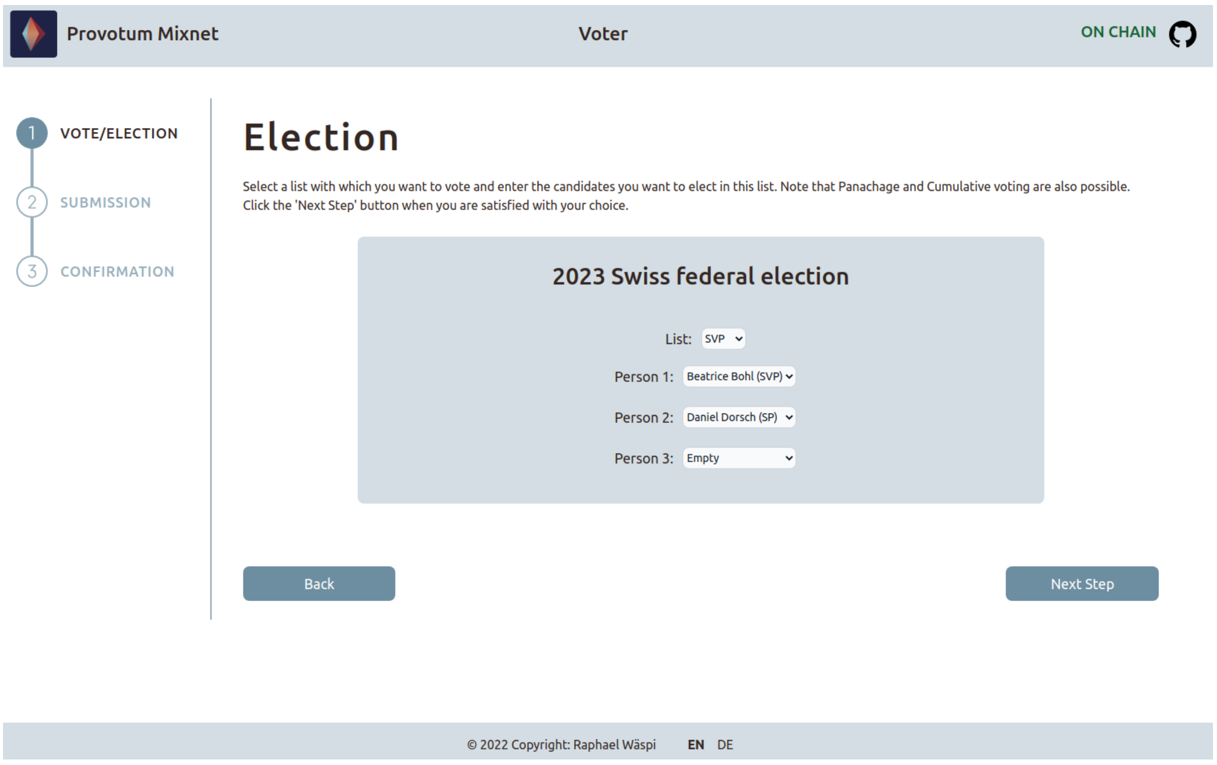
Do you wish to accept amendment to the Federal Act on the Legal Basis of Federal Council Orders to Address the COVID-19 Outbreak?	[✓ Yes] No
Do you wish to accept the popular Justice Initiative?	Yes [✓ No]
Do you want to accept the popular Nursing Care Initiative?	[✓ Yes] No

Back

Next Step

© 2022 Copyright: Raphael Wäspi EN DE

Figure 6.11: Ballot of a vote in the voter GUI


Provotum Mixnet

Voter

ON CHAIN

1 VOTE/ELECTION

2 SUBMISSION

3 CONFIRMATION

Election

Select a list with which you want to vote and enter the candidates you want to elect in this list. Note that Panachage and Cumulative voting are also possible.
Click the 'Next Step' button when you are satisfied with your choice.

2023 Swiss federal election

List:	SVP
Person 1:	Beatrice Bohl (SVP)
Person 2:	Daniel Dorsch (SP)
Person 3:	Empty

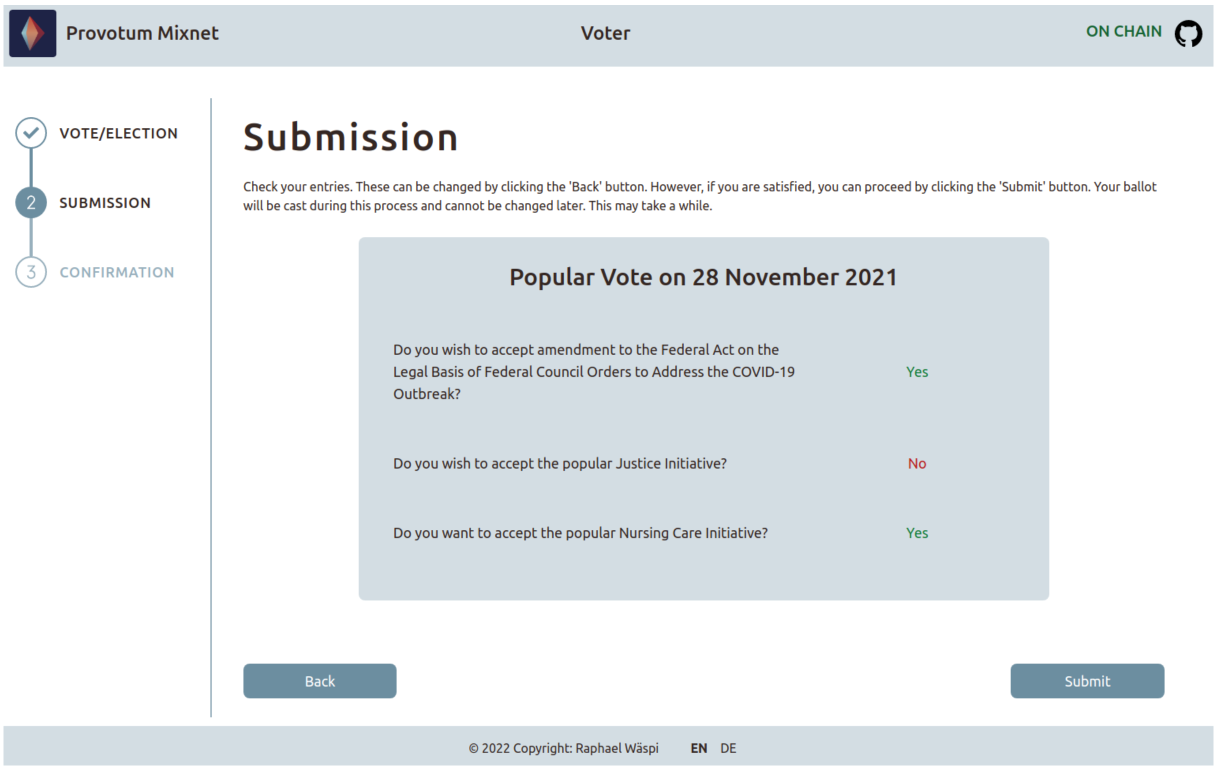
Back

Next Step

© 2022 Copyright: Raphael Wäspi EN DE

Figure 6.12: Ballot of an election in the voter GUI

Casting a Ballot: After the ballot has been filled out and the ”Next Step” button has been pressed, the voters can check their entries again, as shown in Figure 6.13. If the users are not satisfied with the choice, they can return to the ballot and edit it. However, if the voters are satisfied with their information, they can submit it. It may also take longer to submit the ballot, as the inner workings of Provotum BC can cause problems if several ballots are submitted in quick succession. For this reason, as with the sealer GUI, a waiting period between votes has been implemented in the voter GUI. This solves the problem if the number of voters is small, but not if several voters want to cast their vote at the same time. This issue could be solved in a future work.



Submission

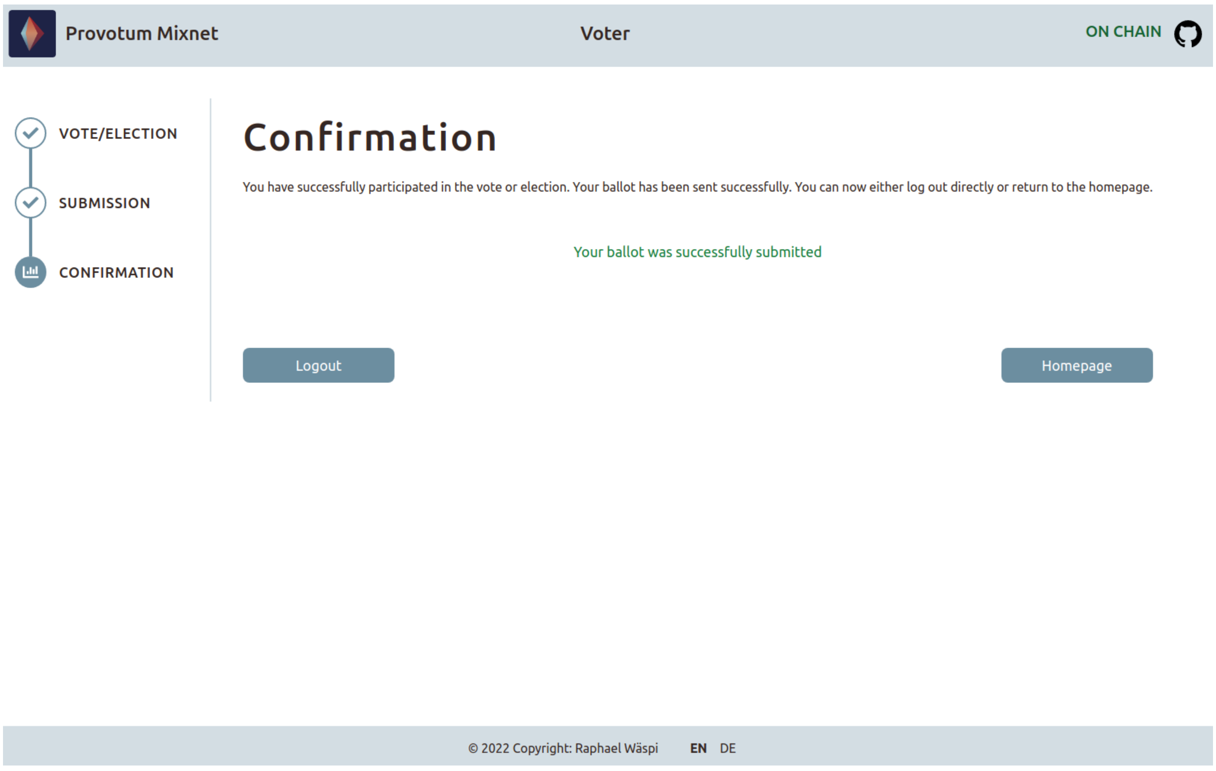
Check your entries. These can be changed by clicking the 'Back' button. However, if you are satisfied, you can proceed by clicking the 'Submit' button. Your ballot will be cast during this process and cannot be changed later. This may take a while.

Popular Vote on 28 November 2021

Do you wish to accept amendment to the Federal Act on the Legal Basis of Federal Council Orders to Address the COVID-19 Outbreak?	Yes
Do you wish to accept the popular Justice Initiative?	No
Do you want to accept the popular Nursing Care Initiative?	Yes

Figure 6.13: Verifying and casting the ballot

Closing the vote If everything worked with the casting of the vote, the user will receive a confirmation, as shown in Figure 6.14. From that moment on, that voter can no longer edit the ballot. For privacy reasons, the confirmation does not reveal what the voter has chosen. Since the voting for the voter GUI is then also finished, the user can then log out directly or return to home page.


Provotum Mixnet

Voter

ON CHAIN

VOTE/ELECTION

SUBMISSION

CONFIRMATION

Confirmation

You have successfully participated in the vote or election. Your ballot has been sent successfully. You can now either log out directly or return to the homepage.

Your ballot was successfully submitted

© 2022 Copyright: Raphael Wäspi EN DE

Figure 6.14: Confirmation of a correct vote

6.3 Discussion

This evaluation is mainly based on the privacy and verifiability properties that are explained in Section 2.3.2. This includes a discussion of the proposed prototype as well as the development of Provotum's previous prototypes.

6.3.1 Privacy

Throughout the various prototypes, the privacy properties of the Provotum REV system have continuously evolved. This evolution is shown in Table 6.4 below. The table shows that BS is already implemented in the proposed voting protocol as well as in the previous versions. The reason for this is the encryption of the ballots stored on the PBB [10]. In addition, no single entity is able to decrypt any part of the ballots. The reason for this is that the encryption of the ballots requires a public key that is generated together with the private keys of the sealers. This means that the approval of all sealers is needend for decryption. So it is not enough if only some or a majority of the sealers are malicious.

Table 6.4 also shows that since version Provotum 3.0 HE, the voting protocol is RF. To ensure that a voter cannot prove how the vote was cast, the vote is encrypted with a random value from the voter and then again with a random value from the randomizer. So it is impossible to show the filled ballot, because no person can know both random values [10]. Second, the voter receives proof from the designated verifier that the ballot was correctly re-encrypted [9]. Such proof is only meaningful to the voter and therefore not transferable to a vote buyer [10]. In the voter GUI it is implemented that the voter receives a confirmation after sending the ballot, but it does not indicate how the user voted.

However, the current Provotum prototype, like all previous versions, is not CR. This means that a coercer could force a voter to vote at his discretion. One solution would be to allow voters to vote multiple times, overriding the previous vote. However, Swiss law [79] dictates that each voter has only one vote and can only vote once [10].

	BS	**RF**	**CR**
Provotum 2.0 [20]	✓	✗	✗
Provotum 3.0 HE [9]	✓	✓	✗
Provotum 3.0 Mixnet [10]	✓	✓	✗

Table 6.4: Privacy properties for three latest Provotum prototypes, after [10]

6.3.2 Verifiability

Table 6.5 shows the evolution of the verifiability properties in the Provotum project. It states that nothing has changed from Provotum 2.0 to Provotum 3.0 Mixnet in terms of verifiability. The requirements for IV and RaC are fulfilled by allowing the voter to make a comparison between the ballot on their devices and the ballot on the PBB. The comparison can be made because the PBB returned a transaction hash after the encryptions of the voter and the randomizer. Once all peers have agreed to include the block, the voter can check if the voter's transaction hash is in the transaction log [10]. However, this is not possible in the GUIs.

In addition, the voting protocol is also CaR. This means that everyone, including voters, can check that everything has been done correctly [9]. Tampering, such as switching the ballot, would be noticed immediately by someone because the proofs would fail during the verification. UV is ensured in the proposed Provotum prototype as it is RaC and CaR [10].

However, the prototype is not a CaI for layerpeople. This means that layerpeople have no way of examining the contents of their ballot after it has been cast. However, there is a complicated procedure that makes this possible, but all users would have to understand the cryptographic primitives. This is not the case, as Provotum is intended for the masses and therefore any citizen eligible to vote may use the system. This implies that this procedure has not been implemented in the GUI. This also causes that the current REV system does not fulfill the requirements for E2E-V. For E2E-V, the three properties RaC, CaR, and CaI would have to be satisfied. Although RaC and CaR would be satisfied, the system lacks a more user-friendly audit capability to ensure that CaI is also fulfilled [10].

	IV	UV	E2E-V	RaC	CaR	CaI
Provotum 2.0 [20]	✓	✓	✗	✓	✓	✗
Provotum 3.0 HE [9]	✓	✓	✗	✓	✓	✗
Provotum 3.0 Mixnet [10]	✓	✓	✗	✓	✓	✗

Table 6.5: Verifiability properties for three latest Provotum prototypes, after [10]

6.4 System Usability Scale

The System Usability Scale (SUS) is a survey that can be used to quickly and easily evaluate the usability of a software. The SUS was developed by John Brooke in 1996 because there was a huge demand at the time for a tool that could quickly and easily record users' usability ratings of a product. A study [80] from 2008 shows that SUS has satisfied this need in the usability community and that it is a highly robust and versatile tool. The reason for this is that the SUS is technology-independent and can therefore be used in many areas. In addition, the survey is quite easy to use for the study participants and provides a score on a scale that can be easily understood by everybody. Respondents are provided with 10 questionnaires to rate on a scale of 1 (strongly disagree) to 5 (strongly agree) [81]. To obtain a respondent's SUS score, the odd and even questions must be scored differently. The points of the odd questions must be added and then subtracted with the value five. For the questions with even numbers, the values must also be added first, but then the sum of the addition is subtracted from the value 25. To obtain a respondent's final SUS score, the odd and even scores must be added together and then multiplied by the value 2.5. However, to obtain a representative value for the system, several users must be surveyed and then the average of all respondent's final SUS scores must be calculated. This average is then the final SUS score for the system and is also interpreted as follows:

SUS Score	Grade	Adjective Rating
>80.3	A	Excellent
68-80.3	B	Good
68	C	Okay
51-68	D	Poor
<51	F	Awful

Table 6.6: Interpretation of a SUS score, after [82] and [83]

For the GUIs created in this thesis, a SUS was also conducted with 15 people. These 15 people have an age between 19 and 67 years. In these surveys, the SUS template [81] was used, which contains the following questions:

1. ”I think that I would like to use this website frequently.”

2. ”I found this website unnecessarily complex.”

3. ”I thought this website was easy to use.”

4. ”I think that I would need assistance to be able to use this website.”

5. ”I found the various functions in this website were well integrated.”

6. ”I thought there was too much inconsistency in this website.”

7. ”I would imagine that most people would learn to use this website very quickly.”

8. ”I found this website very cumbersome/awkward to use.”

9. ”I felt very confident using this website.”

10. ”I needed to learn a lot of things before I could get going with this website.”

A separate SUS was created for all three GUIs. Starting with the VA GUI, this has a final SUS score of 88.8 (calculations in Appendix B). This means that the UI according to the Table 6.6 gets the grade A and it can be called as excellent. The biggest difficulty was entering the data to create a vote or election. For example, one person thought that the placeholder in the voting name was predefined and that nothing else needed to be entered in this input field. There were also people who clicked through the software without reading the description and the warnings. This was particularly evident when the voting phase began, as most of them had already clicked the ”End” button again by this point. This suggests that the testers did not fully understand the process and the tasks of a VA. As an improvement to the GUI, a time period could be entered in which the votes can be casted. This improvement is also mentioned in Subsection 6.1.1. For the sealer GUI, the final SUS score is 92.7. This means that this GUI receives also the grade A. The SUS score of this GUI is higher because it is less complicated than the VA UI. The reason for this is that there are neither input fields nor does the user have to navigate through components. However, as with the VA GUI, the users often do not know why they need to perform a particular action. For the voter GUI, the SUS final grade is 94.2. This means that the grade is also an A. Unlike the other GUIs, respondents knew why each action had to be performed. This means, for example, that a user does not have to wait for the VA to change phases or for the sealers to create their keys. In the voter GUI, users only see whether the vote or election is open or closed. Therefore, the voter does not see much of the inner workings of the Provotum BC. As a result, it is easier for respondents to understand all the features. However, it was suggested to make this GUI even more user-friendly. It is not too different from the other GUIs, which could be problematic since this GUI can be used by any person entitled to vote in a real-world scenario. The most frequently mentioned suggestion for improvement is a better user manual. Instead of plain text, step-by-step instructions should be provided. In addition, the functions should be better implemented in an election. One suggestion was that voters should first select only a list. After that, all people on the list should be visible and voters can cross off and cumulate from that list.

Chapter 7

Summary and Conclusions

The overarching goal of this work was to design and implement GUIs for voting and vote management in Provotum 3.0 Mixnet. This also means that relevant GUIs from other REV systems, such as the Swiss Post voting system, had to be researched. In addition, other related work of REV systems for usability respectively for UX and UI should be presented and documented. However, to implement suitable GUIs for Provotum 3.0 Mixnet, an understanding of the running system is required. Therefore, important information from these related works was gathered and summarized for this thesis. With all this information, three GUI prototypes were added to the Provotum 3.0 Mixnet version ecosystem. With these GUIs, it is possible for all stakeholders in the Provotum project to perform their tasks remotely via an interface over the Internet. Finally, an evaluation of the proposed UI was made in terms of usability. This includes evaluating whether Provotum 3.0 Mixnet can be used in a real-world scenario and what issues might arise in such a scenario. This chapter shows which goals were achieved and which problems were encountered during this thesis. Afterwards, suggestions for improvements and additions are presented, which should be considered in future work to improve Provotum 3.0 Mixnet.

7.1 Final Considerations

The implementation of this thesis allows vote casting and vote administration to be performed in Provotum 3.0 Mixnet using appropriate GUIs. The main motivation was to allow non-technical people to use the Provotum project without any prior knowledge. Before this thesis, only a CLI allowed to cast and administrate votes. It was also necessary to know what the responsibilities of each stakeholder are and how a vote process looks like in Provotum. With the proposed GUIs, vote casting and vote administration are now much easier to handle, and also the whole election system is visualized, which makes the project more comprehensible to the public. However, the prototype still has some issues that need to be addressed. First, this work developed an API for interacting with the BC that acts as a gateway between the BC and the GUIs. This API is very simple in design and uses the commands developed for the CLI. In addition, the proposed prototype uses a database to store the data, which are required in the GUIs. This is in contradiction to the overall goal of the Provotum project. Because by using a database, the Provotum project is no longer completely decentralized. In addition, the use of GUIs is not yet suitable for a real-world scenario. The most important reason for this is that the API uses the commands specified by Provotum 3.0 Mixnet. With these commands, it is for example not possible to send multiple votes in the same second. This would be very inconvenient, since in a real scenario millions of eligible voters would use this feature within a given time period. Another reason is that a REV system should allow all eligible voters to vote on the system. This prototype has implemented some accessible software design methods explained in Subsection 2.4.3, such as alternative texts for images or adding attributes to labels and input fields so that a connection can be established. However, these features have not been tested on affected people. Another reason is that the prototype was designed in such a way that there exist only two sealers.

7.2 Future Work

This section shows approaches for future work to improve the current prototype. Several areas are addressed in the REV that are either improvements or additions to the prototype.

7.2.1 API improvement

The focus of this thesis was on the development of the different GUIs. However, in order for these GUIs to work, a simple API had to be developed in the scope of this thesis, which allows interaction between the GUIs BC. It uses only the CLI commands of the Provotum 3.0 Mixnet version. However, in order to be able to create requests adapted to the GUIs, an API should be developed that can work directly with the data in the BC. This would already solve many problems. On the one hand, this would allow data currently stored in the database to be retrieved from the BC. This would mean that the created database would no longer be required and Provotum would therefore be fully decentralized again. On the other hand, it would also be very likely that this would eliminate the main reason for unsuitability in real-world scenarios. By adapting the requests to the GUIs, the error message would no longer occur for commands that follow in quick succession during decryption and vote casting. This would of course also increase the performance in the GUIs, as currently some commands are quite time consuming because of the timeouts, which are set to avoid those problems. In addition, a new API could directly indicate to the GUIs how many sealers are defined in the BC. In addition, the secret key of the sealer would no longer need to be stored as an environment variable, since the new API already knows the secret key of the respective sealer. However, small changes would have to be made in the source code of the GUI, since the number of sealers is stored as state in the VA GUI. In addition, the sealer GUI would no longer have to send the secret key as a content of a request.

7.2.2 Adjustments for elections

Provotum 3.0 Mixnet enables not only votes, but also elections. For this reason, the elections were also implemented in the GUIs. However, it can happen, that the decryption of the sealer may not work during the Tallying phase. The reason for this is that the inner workings of the Provotum BC do not allow the decryption for questions respectively parties that have not been voted on. Since there is a possibility that a party will not receive a vote, this problem should be solved. In addition, the parameters for the requests and the CLI commands are designed for voting. This may cause confusion for some future developers, since, for example, the name of a party is stored as a question in the inner workings of Provotum BC. Also the names of the participants on the lists are stored only in the database. Voters select a person by name, but in the background they only select a number stored in the BC that refers to a person in the database list. Also, the current Mixnet version [10] does not include the ability to handwrite a name in an election. In the Swiss electoral system, it is allowed to vote also for people who are not on the list of a party. The CHVote specification [84] could be a good starting point for further research on implementing such write-ins.

7.2.3 Further language support

In this thesis, the technology for additional language support was presented. The reason for this is that during the evaluation of the GUIs, people who do not speak English were also surveyed. Since the proposed GUIs are only prototypes that are not yet used in practice, the texts and explanations in the GUIs will change. However, once Provotum has a final version, these texts should be translated into all national languages.

7.2.4 Further accessibility design

Although the proposed prototype has implemented some accessible software design methods, they can still be extended and improved. One example would be the possibility of customization, which would allow all users to design the GUI according to their preferences. This would mean that, for instance, the font or color could be changed, which would help users with color blindness or dyslexia. Furthermore, these methods should be tested by affected users so that accessibility can be further improved.

7.2.5 Identity Management

The Provotum 3.0 Mixnet version [10] has no identity management. For this reason, some hardcoded users have been created for the voter GUI, which makes a demonstration of the REV look better in public. However, for Provotum to be used in practice, it should provide identity management. It should be noted that identity management already existed in previous Provotum projects [9][20]. However, these providers have always been trusted third parties, which is a problem because this provider must identify voters without affecting BS and ensure that it is not able to manipulate the result of a vote or election. For this reason, further research should be conducted in a future work.

Bibliography

[1] Swiss Federal Chancellery FCh, *E-voting Switzerland*, `https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting.html`, Accessed: 2022-06-24.

[2] C. Killer and B. Stiller, "The swiss postal voting process and its system and security analysis", in *Electronic Voting*, R. Krimmer, M. Volkamer, V. Cortier, *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 134–149.

[3] Swiss Post Ltd, *Ballot box not hacked, errors in the source code – Swiss Post temporarily suspends its e-voting system*, `https://www.post.ch/en/about-us/media/press-releases/2019/swiss-post-temporarily-suspends-its-e-voting-system`, Accessed: 2022-06-24.

[4] Federal Chancellery, *E-voting: current situation in Switzerland?*, `https://www.ch.ch/de/abstimmungen-und-wahlen/e-voting`, Accessed: 2022-08-20.

[5] Department of Informatics - Communication Systems Group, *E-Voting: Blockchain-based Remote Electronic Voting*, `http://www.csg.uzh.ch/csg/en/research/evoting.html`, Accessed: 2022-06-27.

[6] ——, *Provotum*, `http://provotum.ch/`, Accessed: 2022-06-27.

[7] ——, *Provotum Github*, `https://github.com/provotum`, Accessed: 2022-06-27.

[8] R. Matile and C. Killer, *Privacy, verifiability, and auditability in blockchain-based e-voting*, 2018.

[9] A. Hofmann, "Security analysis and improvements of a blockchain-based remote electronic voting system", M.S. thesis, University of Zurich, Nov. 2020.

[10] M. Eck, "Mixnets in a distributed ledger remote electronic voting system", M.S. thesis, University of Zurich, Mar. 2021.

[11] K. Marky, V. Zimmermann, M. Funk, J. Daubert, K. Bleck, and M. Mühlhäuser, "Improving the usability and ux of the swiss internet voting interface", in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–13, ISBN: 9781450367080. [Online]. Available: `https://doi.org/10.1145/3313831.3376769`.

[12] E. Benos, R. Garratt, and P. Gurrola-Perez, "The economics of distributed ledger technology for securities settlement", *Available at SSRN 3023779*, 2017.

[13] P. Egloff and E. Turnes, *Blockchain für die Praxis*. SKV Zürich, 2019.

[14] M. Rauchs, A. Glidden, B. Gordon, *et al.*, "Distributed ledger technology systems: A conceptual framework", *Available at SSRN 3230013*, 2018.

[15] L. W. Cong and Z. He, “Blockchain disruption and smart contracts”, *The Review of Financial Studies*, vol. 32, no. 5, pp. 1754–1797, 2019.

[16] K. Wüst and A. Gervais, “Do you need a blockchain?”, in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2018, pp. 45–54. DOI: `10.1109/CVCBT.2018.00011`.

[17] W. Wang, D. T. Hoang, P. Hu, *et al.*, “A survey on consensus mechanisms and mining strategy management in blockchain networks”, *IEEE Access*, vol. 7, pp. 22 328–22 370, 2019. DOI: `10.1109/ACCESS.2019.2896108`.

[18] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, “Blockchain”, *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 183–187, 2017.

[19] I. Scott, M. de Castro Neto, and F. L. Pinheiro, “Bringing trust and transparency to the opaque world of waste management with blockchain: A polkadot parathread application”, *Available at SSRN 3825072*, 2021.

[20] C. Killer, B. Rodrigues, E. J. Scheid, *et al.*, “Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system”, in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, 2020, pp. 172–183. DOI: `10.1109/LCN48667.2020.9314815`.

[21] Communication Systems Group, *IM 2021 - Tutorial on Blockchain-based Remote Electronic Voting from Theory to Practice*, `https://www.youtube.com/watch?v=Sflo0VMNQHk&t=14230s`, Accessed: 2022-06-27.

[22] The Swiss Federal Chancellery (FCh) (of 13 December 2013 (Status as of 1 July 2018)), *Federal Chancellery Ordinance on Electronic Voting*, `https://www.fedlex.admin.ch/eli/cc/2013/859/en`, Accessed: 2022-07-19.

[23] H. Jonker and J. Pang, “Bulletin boards in voting systems: Modelling and measuring privacy”, in *2011 Sixth International Conference on Availability, Reliability and Security*, 2011, pp. 294–300. DOI: `10.1109/ARES.2011.50`.

[24] P. Agrawal, S. Sharma, and S. Banerjee, *Blockchain vs Public Bulletin Board for Integrity of Elections and Electoral Rolls*, `https://www.theindiaforum.in/article/blockchain-vs-public-bulletin-board-integrity-elections-and-electoral-rolls`, Accessed: 2022-07-19.

[25] R. Krimmer, “A structure for new voting technologies: What they are, how they are used and why”, in *The Art of Structuring*, Springer, 2019, pp. 421–426.

[26] E. Cuvelier, O. Pereira, and T. Peters, “Election verifiability or ballot privacy: Do we need to choose?”, in *European Symposium on Research in Computer Security*, Springer, 2013, pp. 481–498.

[27] H. Jonker, S. Mauw, and J. Pang, “Privacy and verifiability in voting systems: Methods, developments and trends”, *Computer Science Review*, vol. 10, pp. 1–30, 2013, ISSN: 1574-0137. DOI: `https://doi.org/10.1016/j.cosrev.2013.08.002`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1574013713000282`.

[28] S. Delaune, S. Kremer, and M. Ryan, “Coercion-resistance and receipt-freeness in electronic voting”, in *19th IEEE Computer Security Foundations Workshop (CSFW’06)*, IEEE, 2006, 12–pp.

[29] Y. Ruan and X. Zou, "Receipt-freeness and coercion resistance in remote e-voting systems", 2017.

[30] C. Killer, B. Rodrigues, R. Matile, E. Scheid, and B. Stiller, "Design and implementation of cast-as-intended verifiability for a blockchain-based voting system", in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 286–293.

[31] O. Kulyk, J. Henzel, K. Renaud, and M. Volkamer, "Comparing "challenge-based" and "code-based" internet voting verification implementations", in *IFIP Conference on Human-Computer Interaction*, Springer, 2019, pp. 519–538.

[32] P. B. Rønne, P. Y. Ryan, and B. Smyth, "Cast-as-intended: A formal definition and case studies", in *International Conference on Financial Cryptography and Data Security*, Springer, 2021, pp. 251–262.

[33] W. D. Smith, "Cryptography meets voting", *September*, vol. 10, p. 80, 2005.

[34] T. Haines, R. Goré, and B. Sharma, "Did you mix me? formally verifying verifiable mix nets in electronic voting", in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 1748–1765.

[35] H. Joo, "A study on understanding of ui and ux, and understanding of design according to user interface change", *International Journal of Applied Engineering Research*, vol. 12, no. 20, pp. 9931–9935, 2017.

[36] W. O. Galitz, *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.

[37] D. Stone, C. Jarrett, M. Woodroffe, and S. Minocha, *User interface design and evaluation*. Elsevier, 2005.

[38] M. Mandal, *User Experience (UX) is not Limited to UI – A Designer's Perspective*, `https://www.evoketechnologies.com/blog/user-experience-ux-designer-perspective`, Accessed: 2022-07-20.

[39] A. Marcus, "Cross-cultural user-experience design", in *International Conference on Theory and Application of Diagrams*, Springer, 2006, pp. 16–24.

[40] M. Hassenzahl, "User experience and experience design", *The encyclopedia of human-computer interaction*, vol. 2, 2013.

[41] A. Mishra, *How to Incorporate Accessibility in Software Development*, `https://adevait.com/software/accessibility-in-software-development`, Accessed: 2022-07-10.

[42] Interaction Design Foundation, *Accessibility*, `https://www.interaction-design.org/literature/topics/accessibility`, Accessed: 2022-07-10.

[43] S. Lawton, Henry, and L. McGee, *Accessibility*, `https://www.w3.org/standards/webdesign/accessibility`, Accessed: 2022-07-10.

[44] S. Keates, P. J. Clarkson, L.-A. Harrison, and P. Robinson, "Towards a practical inclusive design approach", in *Proceedings on the 2000 conference on Universal Usability*, 2000, pp. 45–52.

[45] A. Khan, S. Khusro, and I. Alam, “Blindsense: An accessibility-inclusive universal user interface for blind people”, *Engineering, Technology & Applied Science Research*, vol. 8, no. 2, pp. 2775–2784, 2018.

[46] L. Rello and R. Baeza-Yates, “Good fonts for dyslexia”, in *Proceedings of the 15th international ACM SIGACCESS conference on computers and accessibility*, 2013, pp. 1–8.

[47] 2022 Deque Systems, *Cognitive Disabilities*, `https://dequeuniversity.com/class/archive/basic-concepts1/types-of-disabilities/cognitive-disabilities`, Accessed: 2022-07-10.

[48] ——, *Seizure Disorders*, `https://dequeuniversity.com/class/archive/basic-concepts1/types-of-disabilities/seizure-disorders`, Accessed: 2022-07-10.

[49] ——, *Speech Disabilities*, `https://dequeuniversity.com/class/archive/basic-concepts1/types-of-disabilities/speech`, Accessed: 2022-07-10.

[50] J. Nielsen, “Enhancing the explanatory power of usability heuristics”, in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, 1994, pp. 152–158.

[51] A. Jain, *10 Heuristic Principles – Jakob Nielsen’s (Usability Heuristics)*, `https://www.uxness.in/2015/02/10-heuristic-principles-jakob-nielsens.html`, Accessed: 2022-07-20.

[52] The Federal Council, *Nationalratswahlen 2023: Änderung bei der Sitzverteilung auf die Kantone*, `https://www.admin.ch/gov/de/start/dokumentation/medienmitteilungen.msg-id-84941.html`, Accessed: 2022-08-05.

[53] Statista Research Department, *Anzahl der Sitze im Nationalrat in der Schweiz nach Kantonen ab der Nationalratswahl vom 20. Oktober 2019*, `https://de.statista.com/statistik/daten/studie/471858/umfrage/sitzverteilung-im-nationalrat-in-der-schweiz-nach-kantonen`, Accessed: 2022-08-05.

[54] ACE Electoral Knowledge Network, *Electoral Systems*, `https://aceproject.org/ace-en/topics/es/annex/esy/esy_ch`, Accessed: 2022-08-05.

[55] SRF Schweizer Radio und Fernsehen, *So wählt man mit Listen*, `https://www.srf.ch/news/schweiz/wahlen-2019/panaschieren-und-kumulieren-so-waehlt-man-mit-listen`, Accessed: 2022-08-05.

[56] R. Cramer, R. Gennaro, and B. Schoenmakers, “A secure and optimally efficient multi-authority election scheme”, *European transactions on Telecommunications*, vol. 8, no. 5, pp. 481–490, 1997.

[57] H. Goretta, B. Purwandari, L. Kumaralalita, and O. T. Anggoro, “Technology criteria analysis and e-voting adoption factors in the 2019 indonesian presidential election”, in *2018 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, IEEE, 2018, pp. 143–149.

[58] V. Distler, M.-L. Zollinger, C. Lallemand, P. B. Roenne, P. Y. Ryan, and V. Koenig, “Security-visible, yet unseen?”, in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.

[59] J. Farris, *Assumptions*, `https://npdbook.com/problem-definition/assumptions/`, Accessed: 2022-08-13.

[60] K. Banovac, *How to implement runtime environment variables with create-react-app, Docker, and Nginx*, https://www.freecodecamp.org/news/how-to-implement-runtime-environment-variables-with-create-react-app-docker-and-nginx-7f9d42a91d70, Accessed: 2022-07-29.

[61] TypeScript Tutorial, *What is TypeScript*, https://www.typescripttutorial.net/typescript-tutorial/what-is-typescript, Accessed: 2022-07-27.

[62] Tutorials Point, *Node.js - Quick Guide*, https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm, Accessed: 2022-07-27.

[63] N. Karnik, *Introduction to Mongoose for MongoDB*, https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/, Accessed: 2022-07-27.

[64] A. Sirotka, *What is React?*, https://flatlogic.com/blog/what-is-react, Accessed: 2022-07-29.

[65] Tutorials Point, *ReactJS - Overview*, https://www.tutorialspoint.com/reactjs/reactjs_overview.htm, Accessed: 2022-07-29.

[66] R. Wieruch, *React Router 6: Private Routes (alias Protected Routes)*, https://www.robinwieruch.de/react-router-private-routes/, Accessed: 2022-07-29.

[67] Meta Platforms, *Hooks at a Glance*, https://reactjs.org/docs/hooks-overview.html, Accessed: 2022-07-29.

[68] D. Abramov, *Redux - Introduction*, https://redux.js.org/introduction, Accessed: 2022-07-29.

[69] ——, *Redux - Writing Slices*, https://redux.js.org/tutorials/fundamentals/part-8-modern-redux#writing-slices, Accessed: 2022-07-29.

[70] ——, *Redux - Redux Fundamentals, Part 3: State, Actions, and Reducers*, https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers#project-setup, Accessed: 2022-07-29.

[71] B. Golubovic, *Redux Toolkit: Redux with Less Code Part I*, https://symphony.is/blog/redux-toolkit-redux-with-less-code-part-i, Accessed: 2022-07-29.

[72] T. Briggs, *Persist state with Redux Persist using Redux Toolkit in React*, https://blog.logrocket.com/persist-state-redux-persist-redux-toolkit-react, Accessed: 2022-07-29.

[73] A. Verma, *Difference between Fetch and Axios.js for making http requests*, https://www.geeksforgeeks.org/difference-between-fetch-and-axios-js-for-making-http-requests, Accessed: 2022-07-29.

[74] N. Young, *The pros and cons of Tailwind CSS*, https://www.webdesignerdepot.com/2021/09/the-pros-and-cons-of-tailwind-css, Accessed: 2022-07-29.

[75] I18next documentation, *Introduction*, https://www.i18next.com, Accessed: 2022-08-05.

[76] TailwindCSS, *Responsive Design*, https://tailwindcss.com/docs/responsive-design, Accessed: 2022-08-17.

[77] R. Jeffries, J. R. Miller, C. Wharton, and K. M. Uyeda, “User interface evaluation in the real world: A comparison of four techniques”, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1991.

[78] C. Brasser, “Design and implementation of systems interfaces for a decentralized remote electronic voting system”, M.S. thesis, University of Zurich, Jun. 2021.

[79] Der Schweizerische Bundesrat (vom 24. Mai 1978 (Stand am 1. Juli 2022)), *Verordnung über die politischen Rechte*, `https://www.fedlex.admin.ch/eli/cc/1978/712_712_712/de`, Accessed: 2022-08-08.

[80] A. Bangor, P. T. Kortum, and J. T. Miller, “An empirical evaluation of the system usability scale”, *International Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008. DOI: `10.1080/10447310802205776`. [Online]. Available: `https://doi.org/10.1080/10447310802205776`.

[81] A. Chinn, *What’s the System Usability Scale (SUS) & How Can You Use It?*, `https://blog.hubspot.com/service/system-usability-scale-sus`, Accessed: 2022-08-10.

[82] H. Alathas, *How to Measure Product Usability with the System Usability Scale (SUS) Score*, `https://uxplanet.org/how-to-measure-product-usability-with-the-system-usability-scale-sus-score-69f3875b858f`, Accessed: 2022-08-10.

[83] W. T, *Measuring and Interpreting System Usability Scale (SUS)*, `https://uiuxtrend.com/measuring-system-usability-scale-sus`, Accessed: 2022-08-10.

[84] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, “Chvote system specification.”, *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 325, 2017.

Abbreviations

BC	Blockchain
BS	Ballot Secrecy
CaI	Cast-as-Intended
CaR	Counted-as-Recorded
CLI	Command-Line Interface
CR	Coercion-Resistance
DL	Distributed Ledger
DOM	Document Object Model
E2E-V	End-to-End Verifiability
GUI	Graphical User Interface
HCI	Human Computer Interaction
IV	Individual Verifiability
PBB	Public Bulletin Board
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
RaC	Recorded-as-Cast
REV	Remote Electronic Voting
RF	Receipt-Freeness
SQL	Structured Query Language
SUS	System Usability Scale
UI	User Interface
URL	Uniform Resource Locator
UV	Universal Verifiability
UX	User Experience
VA	Voting Authority
W3C	World Wide Web Consortium

Glossary

Ballot A piece of paper on which you make your choice in a secret ballot.

Blockchain (BC) An append-only distributed storing system which is managed by a peer-to-peer network.

Ballot-Secrecy (BS) Property in a REV system that guarantees that ballots remain private.

Cast-as-Intended (CaI) Property of a REV system that allows the voter to verify the information on the ballot after it has been encrypted.

Command Line Interface (CLI) A text-based user interface through which programs can be executed.

Count-as-Recorded (CaR) Property of a REV system that allows the voter to verify if the vote was added to the system count.

Graphical User Interface (GUI) Graphical interface of a software to facilitate the operation and interaction with an application

Identity Provider Responsible for ensuring that only eligible voters are able to vote.

JSON Data transfer format responsible for exchanging data between applications in an easily readable text form

Proof-of-Authority (PoA) Consensus algorithm that gives a number of trusted actors in the blockchain the ability to validate transactions.

Proof-of-Stake (PoS) Consensus algorithm in which the influence of an entity determines whether a block can be produced.

Proof-of-Work (PoW) Consensus algorithm in which the solving of a calculation determines whether an entity can produce the block.

Receipt-Freeness (RF) Property of a REV that makes it impossible for the voters to prove how they voted.

Recorded-as-cast (RaC) Property of a REV system that proves whether the ballot was received as casted.

Remote Electronic Voting (REV) The conduct of a vote or an election via the Internet.

Sealer The entity that is able to add the blocks in a PoA BC.

User Interface (UI) A software designed to interact with a user.

Voting Authority (VA) The entity responsible for the administration of a vote or an election in a REV system.

List of Figures

List of Tables

Listings

Appendix A

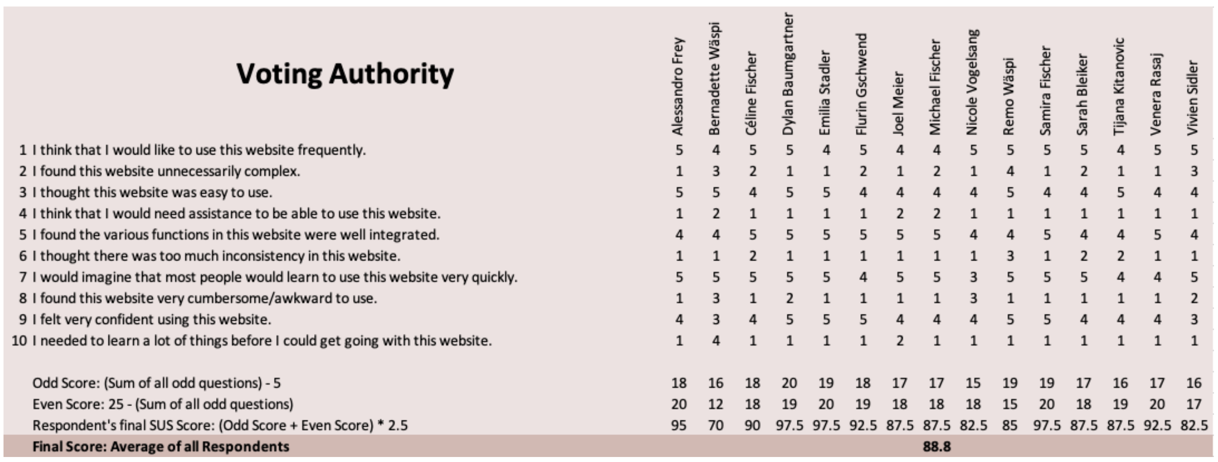
Installation Guidelines

All source code developed as part of this thesis can be found on GitHub. A README is provided for each package, which always includes instructions for installation and local deployment. There is also a Docker Compose script in the infrastructure repository that simplifies running a demonstration of the prototype. To run this demonstration, follow the instructions in the README of this repository.

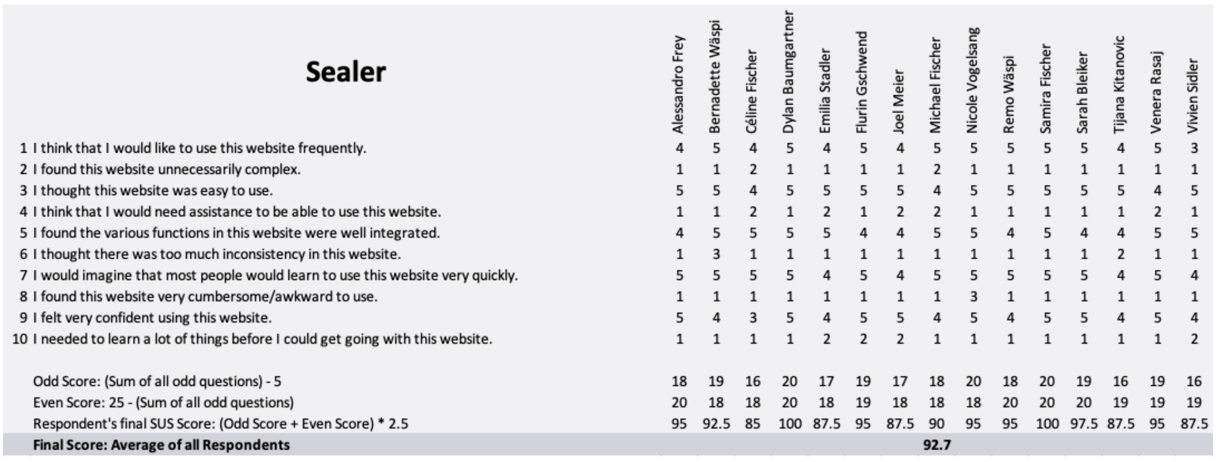
To ensure compatibility, Ubuntu version 20.04, Docker version 20.10 and Docker Compose version 1.26 or higher are required. It may work on older versions, but the application has been tested for these versions, so it is strongly recommended to use them.

Appendix B

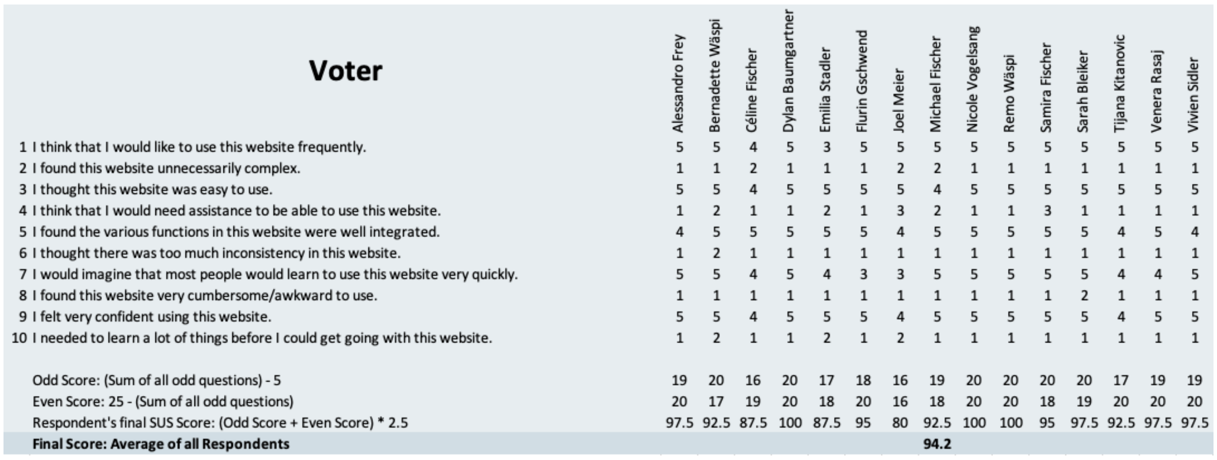
System Usability Scale



	Voting Authority	Alessandro Frey	Bernadette Wäspi	Céline Fischer	Dylan Baumgartner	Emilia Stadler	Flurin Gschwend	Joel Meier	Michael Fischer	Nicole Vogelsang	Remo Wäspi	Samira Fischer	Sarah Bleiker	Tijana Kitanovic	Venera Rasaj	Vivien Sidler
1	I think that I would like to use this website frequently.	5	4	5	5	4	5	4	4	5	5	5	5	4	5	5
2	I found this website unnecessarily complex.	1	3	2	1	1	2	1	2	1	4	1	2	1	1	3
3	I thought this website was easy to use.	5	5	4	5	5	4	4	4	4	5	4	4	5	4	4
4	I think that I would need assistance to be able to use this website.	1	2	1	1	1	1	2	2	1	1	1	1	1	1	1
5	I found the various functions in this website were well integrated.	4	4	5	5	5	5	5	5	4	4	5	4	4	5	4
6	I thought there was too much inconsistency in this website.	1	1	2	1	1	1	1	1	1	3	1	2	2	1	1
7	I would imagine that most people would learn to use this website very quickly.	5	5	5	5	5	4	5	5	3	5	5	5	4	4	5
8	I found this website very cumbersome/awkward to use.	1	3	1	2	1	1	1	1	3	1	1	1	1	1	2
9	I felt very confident using this website.	4	3	4	5	5	5	4	4	4	5	5	4	4	4	3
10	I needed to learn a lot of things before I could get going with this website.	1	4	1	1	1	1	2	1	1	1	1	1	1	1	1
	Odd Score: (Sum of all odd questions) - 5	18	16	18	20	19	18	17	17	15	19	19	17	16	17	16
	Even Score: 25 - (Sum of all odd questions)	20	12	18	19	20	19	18	18	18	15	20	18	19	20	17
	Respondent's final SUS Score: (Odd Score + Even Score) * 2.5	95	70	90	97.5	97.5	92.5	87.5	87.5	82.5	85	97.5	87.5	87.5	92.5	82.5
	Final Score: Average of all Respondents								**88.8**							



	Sealer	Alessandro Frey	Bernadette Wäspi	Céline Fischer	Dylan Baumgartner	Emilia Stadler	Flurin Gschwend	Joel Meier	Michael Fischer	Nicole Vogelsang	Remo Wäspi	Samira Fischer	Sarah Bleiker	Tijana Kitanovic	Venera Rasaj	Vivien Sidler
1	I think that I would like to use this website frequently.	4	5	4	5	4	5	4	5	5	5	5	5	4	5	3
2	I found this website unnecessarily complex.	1	1	2	1	1	1	1	2	1	1	1	1	1	1	1
3	I thought this website was easy to use.	5	5	4	5	5	5	5	4	5	5	5	5	5	4	5
4	I think that I would need assistance to be able to use this website.	1	1	2	1	2	1	2	2	1	1	1	1	1	2	1
5	I found the various functions in this website were well integrated.	4	5	5	5	5	4	4	5	5	4	5	4	4	5	5
6	I thought there was too much inconsistency in this website.	1	3	1	1	1	1	1	1	1	1	1	1	2	1	1
7	I would imagine that most people would learn to use this website very quickly.	5	5	5	5	4	5	4	5	5	5	5	5	4	5	4
8	I found this website very cumbersome/awkward to use.	1	1	1	1	1	1	1	1	3	1	1	1	1	1	1
9	I felt very confident using this website.	5	4	3	5	4	5	5	4	5	4	5	5	4	5	4
10	I needed to learn a lot of things before I could get going with this website.	1	1	1	1	2	2	2	1	1	1	1	1	1	1	2
	Odd Score: (Sum of all odd questions) - 5	18	19	16	20	17	19	17	18	20	18	20	19	16	19	16
	Even Score: 25 - (Sum of all odd questions)	20	18	18	20	18	19	18	18	18	20	20	20	19	19	19
	Respondent's final SUS Score: (Odd Score + Even Score) * 2.5	95	92.5	85	100	87.5	95	87.5	90	95	95	100	97.5	87.5	95	87.5
	Final Score: Average of all Respondents								**92.7**							



	Voter	Alessandro Frey	Bernadette Wäspi	Céline Fischer	Dylan Baumgartner	Emilia Stadler	Flurin Gschwend	Joel Meier	Michael Fischer	Nicole Vogelsang	Remo Wäspi	Samira Fischer	Sarah Bleiker	Tijana Kitanovic	Venera Rasaj	Vivien Sidler
1	I think that I would like to use this website frequently.	5	5	4	5	3	5	5	5	5	5	5	5	5	5	5
2	I found this website unnecessarily complex.	1	1	2	1	1	1	2	2	1	1	1	1	1	1	1
3	I thought this website was easy to use.	5	5	4	5	5	5	5	4	5	5	5	5	5	5	5
4	I think that I would need assistance to be able to use this website.	1	2	1	1	2	1	3	2	1	1	3	1	1	1	1
5	I found the various functions in this website were well integrated.	4	5	5	5	5	5	4	5	5	5	5	5	4	5	4
6	I thought there was too much inconsistency in this website.	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1
7	I would imagine that most people would learn to use this website very quickly.	5	5	4	5	4	3	3	5	5	5	5	5	4	4	5
8	I found this website very cumbersome/awkward to use.	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1
9	I felt very confident using this website.	5	5	4	5	5	5	4	5	5	5	5	5	4	5	5
10	I needed to learn a lot of things before I could get going with this website.	1	2	1	1	2	1	2	1	1	1	1	1	1	1	1
	Odd Score: (Sum of all odd questions) - 5	19	20	16	20	17	18	16	19	20	20	20	20	17	19	19
	Even Score: 25 - (Sum of all odd questions)	20	17	19	20	18	20	16	18	20	20	18	19	20	20	20
	Respondent's final SUS Score: (Odd Score + Even Score) * 2.5	97.5	92.5	87.5	100	87.5	95	80	92.5	100	100	95	97.5	92.5	97.5	97.5
	Final Score: Average of all Respondents								**94.2**							