



**University of
Zurich**^{UZH}

Design and Implementation of Systems Interfaces for a Decentralized Remote Electronic Voting System

*Claudio Brasser
Zurich, Switzerland
Student ID: 14-921-746*

Supervisor: Christian Killer, Muriel Franco
Date of Submission: June 21, 2021

Abstract

The right to vote for laws and political positions is a central part of modern democracy. While voting through post or at the voting site are still the most prevalent forms of submitting one's ballot, there exist recent efforts in taking this process online. Remote Electronic voting (REV) systems allow the voting population to submit their votes from the comfort of their own home, through internet-capable devices such as personal computers or smartphones. Casting a vote is a process of high severity, and for a lot of persons, bears emotional attachment. Recent literature and user studies on REV systems have uncovered concerns about security and privacy within the voters' mental models of casting their votes online. In most cases these concerns are covered by the software and thus don't actually form a threat to a vote's integrity. However, the system's user interface (UI) may not elicit the confidence and trust required for using the application properly while providing a positive experience. A blockchain (BC) offers several desirable properties that are striven for in a REV system such as immutability, decentralization, and transparency. However, BCs are also technical concepts for which a typical voter does not have a mental model. Thus for any REV system using BC technology, it is important to take special care with its UI design. Provotum is a BC-based REV system developed as a research project at the University of Zurich. Its latest version (3.0) received a technical upgrade and security audit, improving security, scalability, and introducing receipt-freeness. This version needs to be orchestrated in a headless fashion, meaning that there is no graphical user interface (GUI) to operate it.

This thesis analyses literature in the fields of UI and user experience (UX) design both within the domain of REV systems as well as in the general realm of modern software systems. Furthermore, the usability, and lack thereof, of Provotum 3.0 is analyzed. Based on these findings, UI designs for the vote administration software and the voting software are proposed. The designs are prototypically implemented using state-of-the-art technology. Both applications are evaluated based on well-established heuristic rules as well as based on the coverage of usage scenarios.

Zusammenfassung

Das Recht, um für Gesetzgebung und politische Ämter zu wählen, ist ein zentraler Teil der modernen Demokratie. Das Wählen durch traditionelle Kanäle wie die Post oder an der Wahlstation ist noch immer am meisten verbreitet. Es gibt jedoch seit geraumer Zeit Bemühungen, diesen Prozess durch das Internet zu bewältigen. Elektronische Wahlsysteme erlauben der wahlberechtigten Bevölkerung, ihre Stimmzettel durch internetfähige Geräte wie Smartphones oder Computer abzugeben. Das Abgeben eines Wahl- oder Stimmzettels birgt große Bedeutung und kommt bei vielen Wählern auch mit Emotionen einher. Literatur und Nutzerstudien im Bereich der elektronischen Stimmabgabe zeigen typische Sorgen der Wähler beim Abgeben der Stimmzettel durch das Internet. Viele dieser Sorgen sind unberechtigt und werden von der Software behandelt, dies wird jedoch oftmals nicht von der Benutzeroberfläche des Systems reflektiert. Dies führt dazu, dass der Benutzer weniger Zuversicht in das System hat als angemessen. Eine Blockchain bringt viele wünschenswerte Eigenschaften eines online Wahlsystems. Diese sind beispielsweise Unveränderbarkeit, Transparenz und Dezentralisierung. Blockchains bringen jedoch auch sehr technische Konzepte mit sich, welche beim typischen Wähler noch unbekannt sind. Dementsprechend ist es wichtig, dass Blockchain basierte Wahlsysteme speziell auf das Design der Benutzeroberfläche achten. Provotum ist solch ein Blockchain basiertes Wahlsystem, welches als Forschungsprojekt entwickelt wurde. Seine neueste Version (3.0) wurde technisch überarbeitet und auf seine Sicherheit überprüft. Dadurch konnte das System im Bereich der Sicherheit und Skalierbarkeit verbessert werden und enthält seit neuestem Maßnahmen gegen den Handel mit Stimmzetteln. Diese Version von Provotum wird jedoch noch immer ohne grafische Benutzeroberflächen bedient.

Diese Arbeit untersucht Literatur in den Bereichen des *User Interface* (UI) und der *User Experience* (UX) Designs. Diese Bereiche werden sowohl in der Domäne der elektronischen Stimmabgabe als auch in Hinsicht auf generelle Designrichtlinien untersucht. Des Weiteren wird Provotum 3.0 auf seine Benutzbarkeit untersucht. Basierend auf diesen Resultaten werden UI Designs für die Administrationssoftware des Wahlsystems und für die Wähler Applikation vorgeschlagen. Die Designs werden prototypisch mit modernen Technologien implementiert. Beide Applikationen werden schlussendlich anhand etablierter Heuristiken und Benutzungsszenarios evaluiert.

Acknowledgments

I would like to express my sincerest gratitude to anyone who supported me in writing this thesis. First and foremost I'd like to thank Christian Killer for providing invaluable feedback and guidance. He helped shaping this thesis but left enough room for my own ideas and visions. Furthermore I'd also like to thank Prof. Dr. Burkhard Stiller for allowing me to write this thesis within the Communication Systems Group.

Finally, I want to thank my parents for their continuous support throughout my demanding, yet rewarding time at the University of Zurich. This would not have been possible if it weren't for them.

Contents

Abstract	i
Zusammenfassung	iii
Acknowledgments	v
1 Introduction	1
1.1 Thesis Outline	2
2 Background	3
2.1 Blockchains and Distributed Ledgers	3
2.2 Remote Electronic Voting	4
2.2.1 Traditional Voting	4
2.2.2 The Swiss Internet Voting Case	5
2.3 User Interfaces & User Experience	6
2.3.1 User Interface Design	6
2.3.2 User Experience Design	7
2.3.3 Heuristic Evaluation	8
2.3.4 Dashboard design	9
2.4 Accessible Software Design	9
2.4.1 Partially impaired Eyesight	10

3	Related Work	13
3.1	Provotum	13
3.1.1	Provotum & Provotum 2.0	13
3.1.2	Provotum 3.0	16
3.2	Research in UX and UI Design for Remote Electronic Voting	17
3.2.1	General REV UI & UX	17
3.2.2	Display of Security Information	18
4	Design	21
4.1	Provotum 3.0 System Overview	21
4.2	Open Areas of Provotum 3.0	22
4.2.1	Technologies	24
4.3	Voting Authority Frontend	26
4.3.1	System Bootstrapping	26
4.3.2	Generating Votes	26
4.3.3	Tallying	27
4.4	Sealer Frontend	27
4.5	Voter Application	27
4.5.1	Registration	27
4.5.2	Vote Browsing	28
5	Implementation	31
5.1	Data Storage Model	31
5.2	VA Application	32
5.2.1	React Hooks	35
5.3	Voter Application	36
5.3.1	Caching	37
5.3.2	Cryptography Libraries	37

<i>CONTENTS</i>	ix
6 Evaluation	39
6.1 Heuristic Evaluation	39
6.1.1 VA frontend	39
6.1.2 Voter Application	41
6.2 Use Case Analysis	43
6.2.1 VA Frontend	43
6.2.2 Voter Application	48
7 Summary and Conclusions	53
7.1 Final Considerations	53
7.2 Future Work	54
7.2.1 User Guidance	54
7.2.2 Language & Terminology support	55
7.2.3 Further accessibility improvements	55
7.2.4 Enhanced information modes	55
7.2.5 State and Information Modelling	55
Abbreviations	61
Glossary	63
List of Figures	64
List of Tables	66
List of Listings	67
A Installation Guidelines	71
B Contents of the CD	73

Chapter 1

Introduction

The ability to vote for federal laws and the occupation of official government positions is an important part of modern democracy. There have been several propositions and pilot projects for taking this process online even in Switzerland [3]. It is needless to state that a software system designed for this task must be highly secure and ideally have shown to withstand professional security screenings. For this reason, the Swiss post has conducted a public intrusion test for their remote electronic voting (REV) system. Although the ballot box could not be manipulated, the system was nonetheless taken down due to problems that have been found in the source code [7]. While these measures aim to improve the security of the system, no public attention has been brought to the usability thereof. Existing literature suggests that perceived security is an important factor in whether or not a user is inclined to install and use a piece of software [15]. Moreover, literature in the specific field of REV suggests that careful evaluation of what information to display to the user may have significant impact on the system's usability [20]. A quantitative user study concerning the Neuchatel REV system, which has been in use for several years, found that a redesigned user interface (UI) improved the user's perceived security of the application [35]. These findings suggest that efforts into the design and evaluation of the UI of a REV system are justified as well.

Provotum is a REV system developed by the Communication Systems Group (CSG). It originated from and has been adapted across several research projects and publications [26][8][36]. The latest version of Provotum (3.0) has been proposed in late 2020 and has improved the system both in its technical security aspects as well as in its ability to be run in larger-scale scenarios [26]. Similar to the Swiss REV systems, Provotum has yet to receive any attention for its usability. In its current version it has no graphical user interfaces (GUI) at all, leaving operation only to developers or at least requiring extensive user guidance.

The general goal of this thesis is to design and implement user interface applications for Provotum. More specifically, this goal may be divided into the implementation of two major software systems: An administration dashboard for managing a vote on the internet and an application for voting through the internet on a smartphone. Each of these applications need to be considered in terms of UI/UX design and software design. All software developed should plug into the Provotum ecosystem with as little modification

as possible. From these objectives, a third sub-goal for this thesis is formed, that is, the review of literature and studies on the subject of UI/UX design in order to form a functional and appealing design system.

In order to achieve this goal, bibliographic reviews in the domain of UI/UX design and REV must be performed. The findings from the literature should then be incorporated into the final design of the proposed applications. Finally, the applications are evaluated in order to detect whether they comply with their respective use-cases and the discussed UI design findings.

1.1 Thesis Outline

This thesis is structured as follows. Chapter 2 introduces required concepts for reading this thesis effectively. This includes an introduction into blockchain and distributed ledger technology and technicalities of remote electronic voting. The chapter ends with the introduction of UI and UX design concepts and evaluation techniques. Chapter 3 discusses related research in the hybrid field of REV and UI/UX. This means an in-depth introduction to the history and development of Provotum and the review of literature on the UX of REV systems. Chapter 4 discusses the software and architecture design of the applications proposed in this work. This chapter lays out the current version of Provotum, its software components and design, and where its limitations lie. Afterwards the proposed additions are introduced based on the identified shortcomings. In Chapter 5, implementation-specific information for the aforementioned applications are provided. Chapter 6 evaluates both applications through use-case scenarios and heuristic evaluation. Finally, chapter 7 culminates this thesis by summarizing achievements, final considerations, and possible openings for future work.

Chapter 2

Background

The following chapter introduces relevant concepts required for reading this thesis. At first, BC technology is briefly explained. After that, the chapter covers the basic ideas of REV, UI design, and accessibility within software design.

2.1 Blockchains and Distributed Ledgers

A blockchain (BC) is a distributed data structure consisting of backward linked batches of transactions, so called blocks. Starting from the first created *Genesis Block*, each block contains a link to the previous one, as well as a set of transactions as its payload and some form of proof of correctness of its content. Each participating peer may maintain a full copy of the chain or only a subset. An important property of a BC is the *Consensus Mechanism* it uses for determining whether a submitted block is valid and should be attached to the chain. Another important characterization of BCs regards who is allowed to write data onto the chain. In a *permissionless* BC, anyone adhering to the protocol rules may submit a new block. The most popular example for permissionless BC is Bitcoin¹. A typical consensus mechanism for permissionless BCs is Proof of Work (PoW). PoW requires anyone who wants to submit a new block to solve a computationally intensive mathematical problem. When solved, a block may be submitted and the solution to the mathematical problem may be easily verified by all other peers. This mechanism protects the chain of information from changing its history, as for each changed block a valid solution to the problem must be provided as well. The counterpart to permissionless BCs are *permissioned* BCs, often referred to as distributed ledgers (DL). In a DL, there exists a central authority which is responsible for granting access to the blockchain. In a *public DL* read access is granted to anyone. Here, the central authority only manages write-access. Whereas in a *private DL* the authority also grants read accesses. For DLs, a typical consensus mechanism is proof of authority (PoA). PoA relies on a set of trusted authorities (sealers) that are allowed to build and submit new blocks and if the majority of the sealers accept a new block, a consensus is reached, and the block is attached to the chain. In REV-related literature, the concept of a public bulletin board (PBB) is

¹<https://bitcoin.org/en/bitcoin-paper>

often mentioned [28]. The PBB acts as a data structure to store all submitted ballots and must be public and append-only. This means that anyone must be able to verify the content of the board, and no one must be permitted to change it [28]. A public DL fits these criteria by nature of the technology and thus is a sensible choice for REV systems. The transparency provided by DLs and BCs also pose certain challenges in the context of REV. For instance, the publicly inspectable ballots must be encrypted in a way that allows verification of the ballot for legitimacy without revealing the content of the ballot.

2.2 Remote Electronic Voting

Expressing one's opinion on a political matter through casting a vote is a cornerstone of modern democracy. As the integration of technology into our everyday life progresses, it branches out into more and more aspects thereof. As more and more public services are provided digitally, the process of voting is currently also facing the prospect of change. The following section first conceptualizes the concept of voting independently of the platform and voting procedure used. Afterward a brief history of REV in Switzerland is provided. The regulatory jurisdiction for REV projects in Switzerland finally leads to security and privacy concerns that come with REV.

2.2.1 Traditional Voting

The process of vote casting has evolved over time. In Switzerland, the most common ways of voting are by postal services or submission of the ballot at the voting site. However, due to the federal nature of the political system in Switzerland, there exist still some regions in which casting a vote by raising one's hand at the public *Landsgemeinde* (a public place) is actively used for direct democracy [4]. This goes to show that the way people are used to casting their vote may differ significantly depending on local and regional traditions. In this work, *paper-based voting* refers to all ways of voting that are not supported by the digital recording of votes. Submitting a ballot by letter is referred to as *postal voting* and voting by submission at the ballot site as *on-site voting*. Regardless of the chosen way of ballot casting, the procedure of orchestrating a vote can be broken down into steps (based on [32]) in order to formalize the process:

1. Technical setup (Pre-voting)
2. Voting
3. Tallying & Publishing (Post-voting)

The pre-voting phase includes all actions required in order to enable eligible citizens to submit their votes. This may include but is not limited to the definition of the vote content, digitization of the vote content, management of eligibility, and definition of the voting time frame. The central authority responsible for the management of these tasks is generally referred to as the voting authority (VA) [26]. During the voting phase, the

voters may cast their votes accordingly to the chosen voting procedure. In traditional (paper-based) voting, this introduces additional stakeholders such as the postal office in the case of postal voting or the voting site and its administrators in case of on-site voting. Finally, in the post-voting phase, the ballots are tallied and the results are published if there have not been any complications. While these steps are simplified versions of what the actual processes may look like, it provides a rough framework thereof. This conceptual model for the voting process is refined later on in this work when it is applied to REV and Provotum.

2.2.2 The Swiss Internet Voting Case

The Swiss federal chancellery published a report on several pilot projects in REV within Switzerland, which also covered the legal basis required for the deployment of such a project [18]. Specifically, the document states that a REV system must provide at least the privacy and security given by non-electronic voting. Since 2004, 15 out of 26 cantons in Switzerland have offered a REV solution to voting and ballot casting to their residents, both local and abroad [3]. Depending on the canton, the residents would have to use one of two systems. One system was developed and maintained by the canton of Geneva. The other one was developed on behalf of the Swiss post in conjunction with a company based in Spain [11]. The Swiss jurisdiction originally only allowed for 10% of all ballots to be cast through REV means, which was later on raised to 30%. Additional laws then allowed certification of REV systems for higher percentages of votes that can be cast through the system. For instance, a system may be allowed to record 50% or 100% of electoral votes given that it fulfills a specific set of technical requirements [22]. The following three criteria after Galindo et al. [22] are used for the classification:

- *Cast-as-intended* verifiability ensures that the voter has the option to verify that the ballot after encryption on the client device contains the options they selected and has not been altered.
- *Recorded-as-cast* verifiability enables the voter to guarantee that their vote has been received and stored by the remote voting system exactly as it has been cast.
- *Count-as-recorded* verifiability enables the voter to verify whether their vote has been count in the election result the way it was cast.

Systems are eligible to record up 50% of the votes if they provide *Cast-as-Intended* verifiability. Systems that additionally provide *Recorded-as-cast* as well as *Count-as-recorded* verifiability may record up to 100% of votes [22]. The system developed on behalf of the Swiss post enabled *Cast-as-intended* verifiability through the use of so-called *return codes*. Prior systems also relied on the comparison of codes. The voter would receive a code from the remote system that could be used to challenge the vote and verify that it has not been tampered with. If a voter would detect a problem with the returned code, they would have to cast their vote again. This approach requires that casting multiple votes is allowed by the countries jurisdiction. The Swiss Post system introduced a *confirmation phase* into

this scheme, which enables the voter to verify the codes before the vote has been cast. Therefore the proposed scheme does not require multiple vote casting to be allowed [22].

In late 2018 the canton of Geneva announced that their system will no longer be available with immediate effect and in July 2019 the Swiss post followed their lead [2]. The Swiss post system underwent multiple public screenings and penetration tests. While there was no evidence suggesting a problem that could have lead to corrupted elections, some critical problems were found with the system [12]. The officials for the Swiss post system announced that they are working on a system with universal verifiability, in contrast to their current system only offering individual verifiability [13]. Thus as of early 2021 there is currently no way of voting electronically for Swiss citizens.

2.3 User Interfaces & User Experience

User interface (UI) design and user experience (UX) design are both highly active fields of research, often referred to as parts of *Human Computer Interaction* (HCI) research. Commercially successful software products and their interfaces are often designed, evaluated, and repeatedly improved through research-based methodology and metrics in cooperation with HCI experts. In the following section, definitions for UI and UX are provided. Commonly used methods and metrics for evaluating both areas are discussed in order to provide a baseline for exploring the UI/UX design for Provotum proposed in this work.

2.3.1 User Interface Design

HCI is a collective term for actions and research related to studying and designing the interactions of computers with technologies, especially personal computers [23]. The *user interface* is the part of software that allows a user to interact with the computer system in order to achieve their goal [44]. In other words, it is the part of a computer system that the user can see, interact, hear, or talk to [23]. Different technologies require specialized user interfaces that may differ in looks and ways of interacting with it [44]. Good UI designs may also adapt over time. For example, UIs for the web in the past had to be designed for interaction through keyboard and mouse peripherals. Today even traditional computers often have touch inputs, posing new challenges and requirements to UI designers and developers. Poorly designed UIs may cause frustration and dissatisfaction of the user and thus severely decrease productivity in a professional environment [44]. One of the key tools in iterative UI design is *prototyping* and especially *low-fidelity prototyping* [44]. A low-fidelity prototype is usually either realized on paper or through specialized software, the key being that it is quick and time-efficient. This way ideas and concepts can be evaluated without the need for programming. A low-fidelity prototype may also help stakeholders to express their needs for the system better and realize what might be missing or incomplete [44]. Typically such a prototype may thus allow soliciting early end-user involvement and feedback that would not have been possible otherwise. The quality of a UI is often expressed in terms of *usability*. A multitude of definitions of usability exists

in literature. In the following paragraph, a set of properties that are often mentioned in the context of usability is provided based on Galitz [23]:

Efficiency expresses how quickly and accurately a user can achieve their goal through the software. **Learnability** measures how hard using the software for the first time is as well as the challenges of mastering the software through gaining a deep understanding of its capabilities. **Error management** concerns the occurrence of errors caused by incorrect actions by the user. Thus on one hand, it is a metric that directly concerns the UI design in terms of how likely a user is to make incorrect actions. On the other hand, it also expresses how well the system may recover from errors. **Effectiveness** measures if the goal of the software may be completely and accurately achieved through normal use. **Satisfaction, Engaging** summarizes how pleasant using the UI is and how much it engages the user to unfold the software's full potential. Evaluation of a UI should be done both qualitatively and quantitatively. Qualitative measures may be taken by the observation of people using the system or through active conversation with the user [23]. Thus either through interviews (e.g. in [35]) or questionnaires (e.g. [20]) feedback from users may be gathered that should describe the UI's qualities in the respective categories of usability. Objective measurements for a UI design require the definition of relevant metrics. These metrics historically often include time required to achieve defined goals or the amount of user actions needed for completing a task [23]. There also exist more recent efforts in the quantification of design and aesthetics that explore concepts such as the proximity of UI elements, word counts in paragraphs, color systems, and go as far as using physics-based gravitational and mass formula in order to evaluate the balance of UI elements [48].

2.3.2 User Experience Design

The concept of user experience (UX) design is used in much broader fields than UI design. It is not limited to software or even computers and technologies in particular. Each product that is designed to fulfill a certain task may elicit an experience from the user. The way a particular user experiences a product may vary based on their cultural provenance [34] and on past experiences with similar products [25]. In the context of user applications, Hassenzahl [25] states that the UI is to be considered part of the UX, accompanied by other factors such as the device on which the software is running, the means of interaction, and how information is communicated to the user. In [25], a model for *Experience* is proposed, whose goal is to conceptualize the term *Experience* in the context of interaction with a product. The proposed model consists of three levels of interaction: *What*, *How*, and *Why*.

The **What** level is concerned with what can be achieved through the intended use of the product. A pocket knife for instance may bring functionality for cutting, sewing, and screwing. A messaging software on the other hand may enable a user to send and receive messages. The **How** is closely coupled to the actual product and its design. The goal of the designer on this level is to enable the *what* of the product while making it accessible and aesthetically pleasing to as many users as possible. Hassenzahl [25] accentuates that even the exact same action may feel better with one product over another. In terms of UI design, this may be simple things such as haptic or visual feedback received after pressing

a button. Lastly, the **Why** regards the reasoning behind using a product. It captures possible emotions related to using a product and should, according to Hassenzahl [25], be considered at the very start of the design process in order to determine the *tone* and *feel* of the experience.

2.3.3 Heuristic Evaluation

Heuristic evaluation is the process of evaluating a UI based on a set of well-defined heuristics [23]. Nielsen [38] originally proposed a set of ten heuristics, which have been slightly reworked in a second iteration [39]. Nielsen states [38] that these heuristics are broad usability principles on topics known to possibly cause problems when ignored. Heuristic evaluation is an inexpensive method of usability engineering that has the ability to find many problems [38][23]. In 2020 a modernized version of the original ten heuristics was published [6]. According to the author, the updated heuristics have been changed to more adequately represent today's technologies, however conceptually they still represent the same principles of usability. In the following section, these heuristics are introduced and summarized concisely after Nielsen [6][39]. These heuristics were used to evaluate the UIs built in the context of this thesis.

- **H1: Visibility of system status**

The user should be made aware of what is happening at all times. All actions should have a clear reaction and the system should not take implicit actions without letting the user know about it.

- **H2: Match between system and real world**

The system's design, language, and imagery should be adapted to the intended user. Wherever applicable, text should be in natural language rather than technical terms. Natural mental models and background knowledge should be used to improve usability.

- **H3: User control and freedom**

The system should easily allow the user to undo and redo actions. The user should feel in control over the application flow at all times. Exit and cancel actions must be clearly labeled.

- **H4: Consistency and standards**

Similar items in the application should look similar and similar actions should be followed by similar reactions.

- **H5: Error prevention**

The system should be designed to prevent errors as much as possible.

- **H6: Recognition rather than recall**

The system should not require the user to remember information from one screen in another screen. Minimizing the user's memory load should be a goal of the application.

- **H7: Flexibility and efficiency of use**

Shortcuts and accelerators for proficient users should be provided while still maintaining intuitive use for inexperienced users.

- **H8: Aesthetic and minimalist design**

The UI should only contain information that is relevant for the usage of the system.

- **H9: Help users recognize, diagnose, and recover from errors**

Errors and error messages should be communicated to the user in their natural language. The user should not have to deal with error codes. Error messages should offer solutions to prevent confusion.

- **H10: Help and documentation**

Ideally, the system should be self-contained. However, if there is necessary information that may not be presented in the system itself, it is important to provide it in an easily accessible way.

2.3.4 Dashboard design

Developing a sensible design for professional applications such as administration dashboards requires attention. In the following section, crucial guidelines specific to dashboard design after Few[21] that have been followed throughout the development of this thesis are discussed. An important aspect of information dashboard design is the spatial distribution of blocks of information. Few argues [21], that fragmenting data that should be seen together is a mistake. Thus typical elements seen in modern UIs such as tabs and different views must be considered carefully in this scenario. Even scrolling down a view in order to gain access to further down information may carry implicit information, such as decreasing importance with increasing scroll distance [21]. In order to support the human's capabilities in information detection optimally, a dashboard should support chunking together information so that it may be perceived optimally. This also goes in line with the recommendation to support the customization of a dashboard to the user's desires and needs. An administration dashboard should also not contain flashy animations. While pleasing and aesthetic design is important, coherent and informative display of data is even more so. Such a dashboard should serve functionality to condense, aggregate, and summarize data according to the domain-specific data and the user's preferences [21].

2.4 Accessible Software Design

An important factor when building and designing any public service or infrastructure is accessibility. Accessibility in the context of design generally refers to preventing any hindrances for persons with disabilities while using the designed product [16]. In the same way, as a public voting site needs to allow people in wheelchairs to enter the building and cast their ballots, a voting application must provide assistance for the widest practically

possible spectrum of abilities. Pavlov & Nikolay [41], as well as Keates *et al.* [30], suggest that rather than looking at specific sectors of the population (e.g., the elderly), one should focus on a set of impairments and possible hindrances which could make using the UI challenging. The following section lists common symptoms a user could potentially have and how literature suggests that they can be addressed through accessible software and GUI design.

2.4.1 Partially impaired Eyesight

According to the world health organization (WHO), at least 2.2 billion people have either a near or a distance vision impairment [47]. Implementing a typeface system that is easily readable for everyone is impossible. However literature suggests that through only minor software adaptations and feature additions, many more people may be included and assisted in using an application [41][30][31][21].

Most modern web browsers offer several accessibility features, such as text-to-speech synthesis and visual zoom levels. The World Wide Web Consortium (W3C)² provides instructions and help for web developers on how to program their websites in a way that supports these tools. This is important as they can only provide real benefits if the website is built around it. Text to speech synthesis may help individuals with impaired eyesight in order to understand the content of the screen they are presented with. The quality of this feature may be drastically improved by providing meaningful labels and metadata. In web forms, a user input typically consists of an input field and an associated textual label indicating what type of input is expected from the user, e.g., their email address. Generally, the label may be constructed using any generic HTML element such as the multi-purpose *div* element, the *p* element used for text bodies, or even a heading type. This provides the browser with no semantic information about which block of markup content belongs to which input field. Thus even though the text to speech system may still read the content of the label to the user, it cannot provide any information on if the corresponding input field to the read label is actually selected. If the programmer has considered this problem and used the appropriate *label* markup block as well as supplied all necessary metadata for connecting the form input element to the label, the browser may exactly tell the user when an input field is selected, what the content should be and if the provided input is valid, e.g., in the case of a specific required format [14].

Another important aspect of usability in the context of eyesight is typewriting. Typically this is defined by a combination of font face, size, and style. Choosing well-informed defaults for font face and size while paying respect to different pixel densities on different devices allows to provide access to a large proportion of the population [41][42] and choosing a readable font for any important application is critical to its success [21]. Research results suggest that there is no one size fits all approach to typesetting in software design but rather that software needs to support the adaptation of the content to the user's needs [41][42][30]. Modern web browsers typically offer functionality to increase text and font size. The W3C formulates standards and guidelines for developers and designers in order to make their websites adapt to these changes without breaking the UI and thus

²<https://www.w3.org>



Figure 2.1: Color scales, (a) offering no perception of relation between the steps, (b) a perceivable linear scale

drastically reduce the quality of experience [14]. Mobile operating systems such as Android³ and IOS⁴ also include settings for preferred text size. While most users might not need to change these default settings or simply are not aware of them, it is important to respect the ones who take the time to explicitly change their preferred font size by making sure that the UI does support their decision.

The font family and style used in applications is typically chosen by the developer or the UI designer. While these may contribute significantly to the perceived style and identity of an application, it is important to prioritize the purpose of the software and the fonts applicability over developer-opinionated design choices [21]. A common disability that makes reading and textual perception harder is dyslexia, a condition that increases the difficulty of writing and decoding written information [42]. A study on people bearing this disability showed that choosing the right font type does impact their reading performance [42]. The study results suggest that *Sans serif*, *monospaced*, and *roman* font types specifically help while *italic* font styles decrease reading performance.

In Europe, around 8% of men and 0.4% of women have some form of inherited color deficiency [17]. Not all visual impairments are equal, and thus not all of them can be addressed in the same way through UX design. However, there exist some general design guidelines that have been shown to help with a large number of impairment levels. Few [21] suggests that using color intensity scales of the same color instead of different colors for labels, badges, or icons allows people with impairments on color-based vision to see differences better. Additionally, the human perception registers differences in intensity more linearly than differences in hue, making the former a better indicator of importance even for non-impaired people [21].

³<https://www.android.com>

⁴<https://www.apple.com/ios/ios-14/>

Chapter 3

Related Work

In order to build a design system that is easily usable while still communicating crucial security-related information to the user, it is important to have a clear understanding of the REV process. This chapter first introduces the *Provotum* REV system as the central piece of related work for this thesis. A brief history of Provotum is shown and followed by detailed explanations of its inner workings. Afterward, related research projects in the field of REV and UI/UX are discussed, together with their results and implications for this work.

3.1 Provotum

Provotum [32][26] is a decentralized REV system. Its latest iteration (Provotum 3.0) originated from a security audit and protocol overhaul of the initially published version [26]. This work builds upon Provotum 3.0 and focuses on its usability and UI design, as mentioned in Chapter 1. This section briefly discusses the original version of Provotum and then introduces the relevant changes and adaptations from Provotum 3.0 in more detail. The goal of this section is to gain a clear understanding of the processes involved in vote administration with Provotum, in order to conceptualize the requirements for a GUI for the latter.

3.1.1 Provotum & Provotum 2.0

The original Provotum system was based on the Ethereum¹ BC. It used Ethereum's capability of executing custom logic through smart contracts in a distributed manner to encode the voting logic into the BC. One important stakeholder that comes with Provotum and has not yet been introduced is the *Sealer*. As aforementioned in Section 2.1, sealers are trusted authorities responsible for validating blocks and running the consensus mechanism in a PoA BC. In the case of a federal-republic system such as Switzerland, there could exist one sealer for each of the 26 cantons [32][26].

¹<https://ethereum.org/en/>

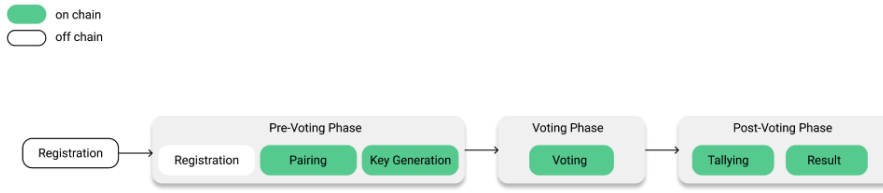


Figure 3.1: Provotum Phases, after [32]

The original Provotum system was improved upon in [32], resulting in Provotum 2.0. The new system was still based on the Ethereum BC and execution of custom code through smart contracts but improved upon some of the problems with the early prototype. According to the authors, the protocol of Provotum 2.0 can be split into the same three categories as the ones introduced in Section 2.2.1. The following paragraph summarizes the voting process with Provotum as described in the publication, putting emphasis on the conceptual sequence of steps and omitting technical detail which is not necessary for this work.

In the pre-voting phase, the VA starts the BC. If it is the first start of the system the genesis block is created as well as a specification document in the case of Provotum 3.0. This document contains all information required for peers to start their own BC node. The sealers may register themselves with the VA afterward and receive information on the BC in order to start their own node and join the peer-to-peer network. This process is shown in Figure 3.2.

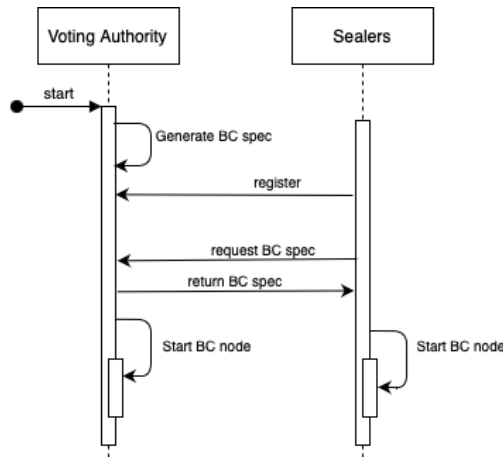


Figure 3.2: Bootstrapping sequence diagram

The VA now generates a vote with all required metadata and all peers partake in a distributed key generation in order to build the public key for the generated vote. Once the vote public key is built, the VA may open the vote for the public to submit their ballots. This leads the vote into the voting phase. Now the voter may choose one of the binary answer options *yes* or *no*. The selected answer is encrypted and sent to the BC, together with a non-interactive zero-knowledge proof (NIZKP), which guarantees that

the vote contains one of the two legitimate answers and nothing else [32]. Once the vote

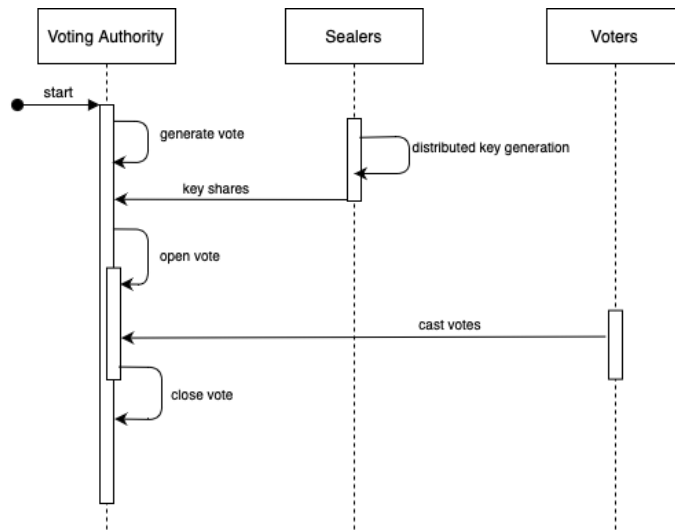


Figure 3.3: Voting sequence diagram

has been included in the BC, the voter receives a confirmation transaction through which they may verify that their vote has been in fact included. Figure 3.3 displays the voting phase. When the voting time window has ended, the VA may close the vote for public submission and thus introduce the post-voting phase.

In this final phase, the sealers collaboratively decrypt all of the votes and store the results in the form of shares on the BC. In order publish the results all of the shares are combined and then the final result is once again written onto the BC, which is described by Figure 3.4.

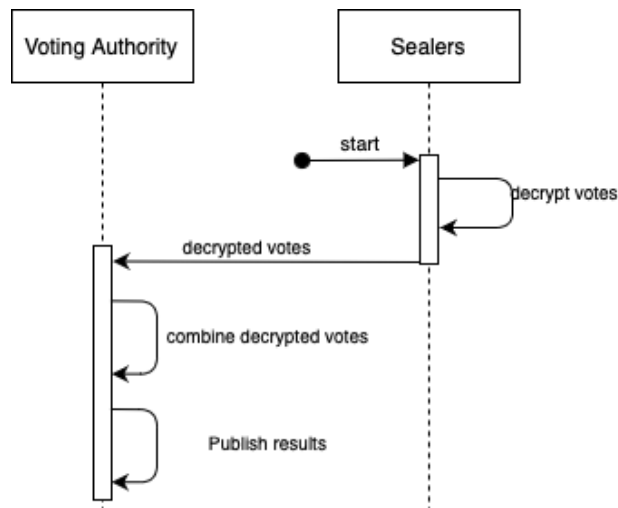


Figure 3.4: Finalization sequence diagram

3.1.2 Provotum 3.0

Provotum 3.0 is the latest iteration in the Provotum ecosystem. During its development, a security analysis for the previous Provotum version was performed with the goal of designing a new iteration of Provotum. The new and improved iteration should address some of the limitations that have already been identified by the authors of Provotum 2.0 [32][26]. The threats identified with Provotum 2.0 include problems on protocol level and implementation-specific problems. E.g., The fact that if a voter would register for electronic voting but would not actually vote, the identity provider could vote through the user's unused token [32][26]. Another identified threat concerned the possibility for a voter to participate in the distributed key generation process and block the tallying of a vote, due to implementation problems [26]. In order to address some threats and limitations posed by the technology stack and Ethereum's smart contracts, the authors chose a different technological base for Provotum 3.0. The new system is built with Substrate², which is a framework for building modular and customized BC technologies.

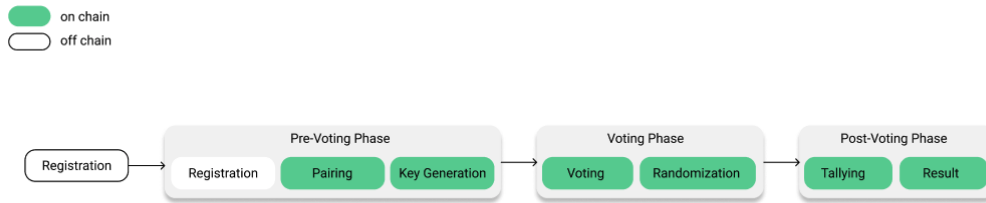


Figure 3.5: Provotum 3.0 Phases, after [26]

Figure 3.5 displays the phases for the re-designed Provotum 3.0 version. Even though the technical base for the implementation changed completely, the overarching phases of the voting process did not change as these are inherent to the voting protocol and implementation agnostic. One of the major identified problems with Provotum 2.0 was missing Receipt-Freeness. A REV scheme may be referred to as *Receipt-Free* if a voter is under no circumstances capable of proving how they voted [19]. As such, Receipt-Freeness is an important property of REV schemes to mitigate the possibility of vote buying or coercion [19]. Provotum 3.0 introduced the *Randomizer* in order to elevate Provotum to a Receipt-Free REV scheme [26]. The Randomizer's task is to blind each submitted vote through a non-deterministic factor that is out of the control of the voter. Previously a similar approach was used to encrypt the votes. However, as the voter had insights into the random value that was used for blinding the vote, they could simply re-use the same value once again in order to prove to anyone how they voted [26].

²<https://substrate.dev>

3.2 Research in UX and UI Design for Remote Electronic Voting

REV has been the subject of numerous research projects and has, in some cases, seen the light of real-world applicability already. Most REV projects strive for a similar set of theoretical properties in order to achieve a security and privacy level that is at least equivalent to traditional, paper-based voting. Different approaches are distinguishable by e.g., the underlying technology, their main focus of research, and the voting protocol used. There exists a plethora of literature and research on the topic of theoretical security, protocol design, and privacy of REV systems. However, even the most secure protocol is hardly of value, if it is not adopted and used by the public. A user study on the adoption of REV suggests that although security is the major concern when evaluated through a quantitative questionnaire, complexity and ease-of-use are within the most prevalent topics in qualitative interviews [24]. The following section discusses a set of existing REV research publications which are concerned with the UI and UX aspects of REV systems.

3.2.1 General REV UI & UX

In [35], the authors assess the usability of the Swiss Post internet voting interface [33]. The authors first introduced the Swiss post system to a group of UX experts from various backgrounds (computer science, design, psychology) and let them each go through the process of casting and challenging a vote. Later on the UX experts were questioned in a semi-structured interview on their findings on the UI. The authors summarized the findings into six usability weaknesses. While some of these weaknesses are specific to the voting protocol and especially to the challenging mechanism, there are some general remarks that can be drawn from the experts' statements [35]. As the Swiss post REV system uses a *return code scheme* for vote challenging, the voter has to submit and compare at least four codes per vote cast in order to ensure *Cast-as-intended* verifiability. This may lead to errors due to fatigue or lack of motivation, which in turn can cause an incorrect vote not to be detected. The voter can also lose their motivation for casting the ballot if something goes wrong and they dread the outlook of going through the process again [35]. Also mentioned repeatedly is the incoherent display of data and information between the code sheet and the software, which can cause confusion and reduce trust in the software. The experts also found some crucial information to be missing or not as prominently displayed as their importance may suggest. One instance thereof is missing instructions about the voting procedure on the code sheet, or hardly detectable information on what to do in case of errors. The REV system was also evaluated through a user study, comparing the original Swiss post system user interface to a version redesigned by the authors. In the following paragraph, some of the main takeaways from the user study by Marky *et al.* [35] are summarized.

The application should make it clear how to act in case of an incorrect vote. When a user realizes that their vote has not been registered by the system in the way they intended to cast it, potential fears and confusion can be prevented by providing a clear idea of how to act in this case in advance. Only one task should be put onto the user

per view of the application. Simplifying each separate view to one conceptual mental task allows the user to focus more on the specific task at hand. By providing clear labels and instructions, confusion about the task themselves may be averted. The user should be asked directly for the outcome of each mental task. The study results suggested that through asking directly for the output of a mental task, even though it may not be strictly relevant to the voting protocol, can help in improving the quality and correctness of the outcome. Thus the authors suggest providing buttons and input fields for tasks such as, e.g., code comparisons in a *return code scheme*. The application should not make the user memorize codes throughout different views. This again goes hand in hand with the concept of asking directly for the outcome of mental tasks, such as code comparisons. The verification process should be explained carefully and completely. The authors refer to *vote verification* as the process of checking for *Cast-as-intended* verifiability. It is argued that this step needs further explanation and special care in terms of UI design, as it is not part of traditional paper-based voting schemes and thus the user does not have a mental model previously built around it. Even if a *Cast-as-intended* verifiability mechanism is provided by the REV system, it can only fulfill its purpose and add value if the user is aware of its importance. Important data used for vote verification and vote challenging should be labeled appropriately and in the user's language. For instance, the study results suggested that the term *finalization code* was perceived to technical by the users and led to confusion. Generally, the authors emphasize on providing clear instructions and step-by-step guides for all scenarios [35].

In a literature review [40], Olembo and Volkamer compiled a list of recommendations for designing ballots and ballot interactions in REV systems. The authors emphasize designing the ballot in a way that is familiar to the voter, e.g., by imitating traditional paper ballot design. As there is no global standard in paper ballot design, this may lead to large differences per voting system. Additionally, this may be technically difficult to achieve, depending on the ballot to be imitated. This idea also stands in conflict with some of the heuristics presented in Chapter 2, as a ballot designed for paper-based voting most likely will not fit usability criteria for digital interfaces. The authors also mention that the system should clearly indicate when a vote has been cast and when an invalid vote was submitted, which goes in line with traditional generic UI design guidelines [39].

3.2.2 Display of Security Information

A crucial aspect of designing the UI/UX of security relevant applications is how much information should be displayed to the user. A trade-off between simplicity and transparency is unavoidable. The question that should be raised within this context is whether or not displaying certain information can improve the user's confidence in the application and thus, improve their experience. In [20], the authors aimed to research the impact of displaying security information on UX in the context of REV. Two versions of a voting application were tested: one showing security mechanisms (labeled version *D*) and one not showing them (labeled version *ND*). Among the findings of the study were a set of actionable guidelines to support the design of secure systems and the identification of the key UX factors that would impact the perceived security. Within the study, two groups of participants were formed and each group was shown one of the versions of the system.

Most participants who were shown version D did not actively notice the displayed security information, such as encryption notices. Although those who noticed the messages felt reassured by them. Some mentioned that they would pay attention to whether or not websites had a *HTTPS* certificate on the web and that there is no such equivalent on mobile applications. Although few participants that were shown the ND version proactively mentioned the lack of security information, some mentioned the process to be seamless. Few participants specifically mentioned a lack of feedback and that they would have liked to be informed about what was currently going on [20].

The REV procedure used by the mock applications included a verification phase, much like the Swiss post system [22]. The goal of the verification phase is to enable the user to verify that their vote has been recorded [20]. In the second iteration of feedback aggregation, the authors conducted interviews about the verification procedure. In alignment with the results from Marky *et al.* [35], the concept of vote verification is new to most participants. The application included a list of encrypted votes from other users, which were completely anonymous. This was explained to the participants through the application in advance. The impressions and statements about the verification phase were mixed. Although the additional step was perceived as positive and reassuring by some participants, others even stated that the additional step made them less confident in the application. They feared that their vote was not confidential just from seeing a list of votes from other people. Some participants also stated that they did not think the verification phase was necessary and that a simple message showing that the vote has been recorded would suffice.

The application also contained a verification phase that included the entry of login codes, which were received by letter in advance. The participants specifically mentioned this letter-based registration procedure to improve their trust in the application. Some participants mentioned that some type of biometric verification procedure, such as fingerprint scanning, would improve their perceived security even further [20].

Based on the results of the user studies, a list of recommendations for the design of such applications was compiled by Distler *et al.* [20]. At the core, these recommendations revolve around awareness of the user's security *concerns* and security *knowledge*. The authors suggest that most users do not have any concrete knowledge about technical security. They usually cannot identify application properties that would make the application more secure or which may prompt room for attacks. However, the results suggest that most users do have a general sense of awareness over security risks coming with new technologies [20]. Thus it is recommended for designers of such security-relevant applications to be aware of this presumption and explore how it may be taken on by UX/UI design [20]. The results of the study also showed that the users gained most of their trust in the application from the authentication phase, in which they were required to register using the codes they received through physical paper. The participants showed a higher degree of motivation for spending time and effort towards security in the authentication phase. This could be taken advantage of by including as many of the necessary security mechanisms as possible within the registration process. Finally, the core question of the study was to assess whether or not displaying security-relevant information to the user is beneficial to the experience of using the technology. Distler *et al.* [20] suggest that doing so may be successfully used to add a sense of importance to a part of the experience or to get more attention from the user onto some part where awareness of security is critical. Also

mentioned is the use of appropriate UI evaluation scales which allow for evaluation on a more holistic level than what is supported by traditional ones [20].

Chapter 4

Design

In this Chapter the architectural & software design for the software proposed in this thesis is discussed. For this purpose, the architecture of Provotum 3.0 is introduced, including all relevant software components with their responsibilities. After having established what is currently existing with Provotum 3.0, the next section explores open areas that are addressed within this thesis. Finally, each of the three programs proposed within this work is discussed in detail throughout their respective sections.

4.1 Provotum 3.0 System Overview

This section describes the architecture of Provotum 3.0 [26] after Hofmann. The involved stakeholders are introduced, alongside their responsibilities in the REV process with Provotum. Additionally, this section mentions all parts of Provotum 3.0 where this work ties in with the system or adds functionality.

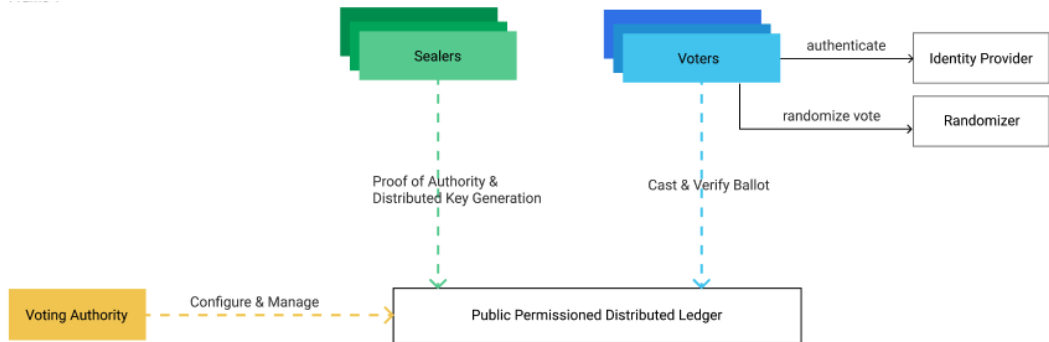


Figure 4.1: Provotum 3.0 Architecture

Figure 4.1 depicts a broad overview of the participating stakeholders of Provotum 3.0. The dashed lines represent interactions with the public permissioned distributed ledger (DL). The VA is the central entity in the voting protocol. They are responsible for the configuration of the system and are the first participant to start a BC node. This is a multi-step process and will later on be referred to *Bootstrapping*. Once the bootstrapping

process is over, the VA may then manage the REV capabilities of Provotum 3.0. Most importantly, they may create new votes with any number of topics and store them on the blockchain. The VA is also responsible for managing the life cycle of their votes. This includes opening the vote to the public when the voting period has started and closing the vote when it has ended. The VA also manages the participating *Sealers* and keeps a local list of them in order to have a complete view of the system.

The **Sealers** each run a BC node and are responsible for the proof of authority (PoA) consensus mechanism. This means that they will check each submitted block and try to find a consensus on whether the block is valid and should be integrated into the BC or not. The sealers also each must submit a public-key share for each vote. When the VA opens an vote for the public to submit their ballots, all public-key shares are then combined into the vote public key. This key is used to encrypt the individual votes. When the VA has closed a vote, the Sealers may collaboratively decrypt and tally the final vote results. Finally, the VA may then publish the final result by combining the tally shares from all sealers.

The **Voters** are members of the population who are eligible to vote. They first prove their identity with the *Identity Provider* in order to prove to the public that they are eligible to vote. Afterward, the Voters may cast votes through direct interaction with the BC. In order to achieve *Receipt Freeness*, Provotum 3.0 introduced a non-deterministic random factor to the ballot encryption, which is handled by the *Randomizer*. A vote will only be accepted by the BC if it has been signed by the Randomizer, thus requiring all ballots to be blinded by this random factor.

4.2 Open Areas of Provotum 3.0

Provotum 3.0 forms a working prototype for a blockchain-based REV system that is by itself fully operable. However, the operation of Provotum 3.0 in its current state is prototypical in many ways. There does not exist any graphical user interfaces (GUI) neither for the Sealers nor for the VA. Thus all responsibilities of the stakeholders as mentioned in Section 4.1 must be triggered directly through communication with a node.js server, the *VA Backend*.

In the Provotum 3.0 demo, this is done via HTTP requests. Even though there exists a request collection for Postman¹ that can be used through the Postman GUI, the system is hardly usable for any user without a very specific technical background. Figure 4.2 shows this interaction. Of course, the interaction via command line interface (CLI) is hardly viable for anyone not coming from the Provotum development community. Even a user familiar with the technicalities of making HTTP requests via CLI tools would have to do a significant amount of research into the domain of REV and Provotum. The operation of the system through a HTTP request manager such as Postman also has several problems and shortcomings. Crucially it still requires technical knowledge and a core understanding of how the communication works. Even configuring the system in a way that the requests

¹<https://www.postman.com>

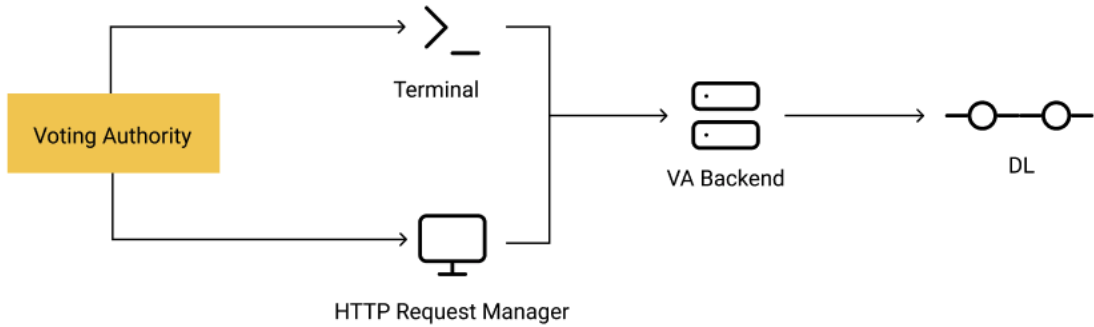


Figure 4.2: Provotum 3.0 User Interaction Flow

could be run requires knowledge of multiple concepts of HTTP communication, such as POST, GET, and request parameters.

Thus this is where this work ties into Provotum 3.0. Firstly the administration and bootstrapping of Provotum is made significantly easier and more accessible by providing a GUI for the VA. The proposed interface is built with web technologies and focuses on adhering to well-established design guidelines for administration software and dashboards. This allows a non-technical person to operate Provotum on their own, leveraging the security and privacy benefits of the system with minimal required domain knowledge about the BC, distributed computing, and REV. Figure 4.3 displays the new way of user interaction with Provotum. Another benefit of the proposed new architecture is the ability to connect directly to the BC from the GUI. This means that for trivial interactions such as data fetching, there is no communication to the VA server, thus reducing communication overhead as well as possibly penetrable connections.

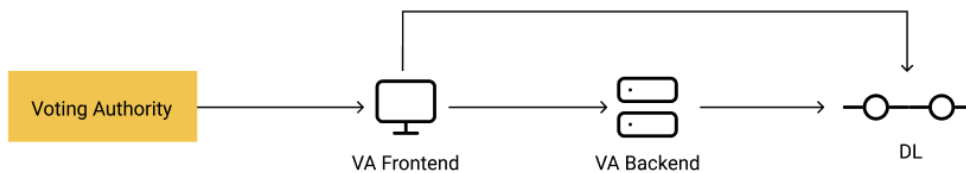


Figure 4.3: New User Interaction Flow

Additionally proposed is also a GUI for the sealers. As their responsibilities can be summarized within a few actions, this interface is highly simplistic in design. The most important factor in this part is to draw attention to the elections which require some manually triggered action from the sealer. Thus the research focus for the Sealer GUI lies in differentiating on what information can be safely omitted in order to improve visibility of the more crucial information at a point in time. The only user interface provided with Provotum 3.0 was a web-based application for the voters. This GUI allows logging in with

the identity provider, casting ballots, and viewing vote results. All data displayed in this voter application is fetched directly from the BC, thus removing any single trusted party. As this GUI was not a central part of Provotum 3.0, it also did not receive the same amount of care and attention as the rest of the system did. Thus this thesis reviews literature on application design with specific focus on REV applications, and on a broader level, security-relevant applications. From this related literature, a redesigned voter application is then proposed. The application will no longer be served as a website but rather built into a native mobile phone application.

Figure 4.4 highlights the addition of GUIs into the Provotum ecosystem by replacing the direct communication of the VA and the Sealers to the Servers. Both of these entities may now communicate their intentions to the system through a GUI.

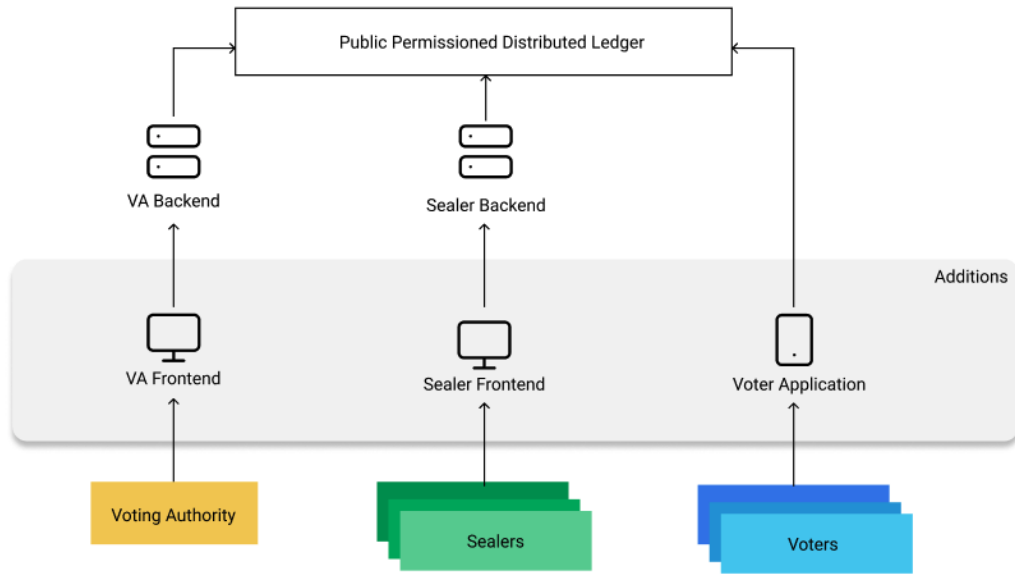


Figure 4.4: Provotum 3.0 Architecture updated

4.2.1 Technologies

The BC for Provotum 3.0 is based on Substrate. Substrate is an ecosystem for building modular and extensible BCs [45]. Custom implementations with Substrate are implemented as Rust modules. Thus all of the protocol implementations and the voting logic are implemented in Rust for Provotum 3.0. These additions to Substrate are referred to as *Pellets*. All parts of Provotum 3.0 which are not in the BC are built with conventional web technologies. They make use of the well-established node.js² framework for building server applications with expressjs³. The VA server application as well as the Sealer server both use the *Substrate Client* library in order to communicate with the BC. This library is written in JavaScript and simplifies the execution of remote procedure calls (RPC) from the server on the BC. The second important helper library is called *evote-crypto-ts*. It

²<https://nodejs.org/en/>

³<https://expressjs.com>

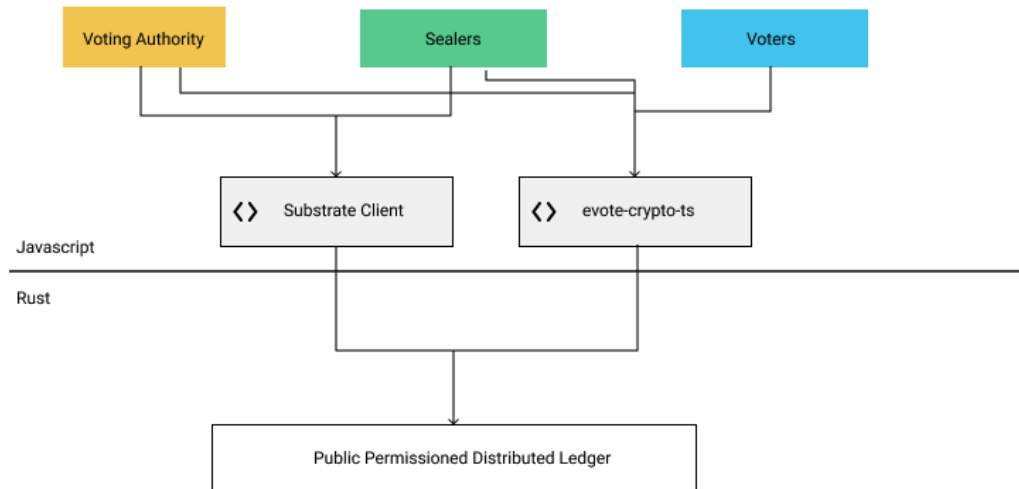


Figure 4.5: Provotum 3.0 Technologies

contains helper functionality for cryptographic functions required by the voting protocol. Figure 4.5 shows this architecture with the technology split in the vertical middle.

For this work, the chosen options for technology stacks were guided by the existing code base. As the architecture of Provotum 3.0 is heavily relying on web technologies, staying within this realm is a sensible choice. Additionally, this technology stack is well suited for rapid prototyping of UIs and provides arguably the best developer experience in terms of UI building. As REV is a highly security-relevant area it is important to pay respect to security when evaluating technology choices. Any addition to Provotum 3.0 must not make the system less secure than it was previously. As the VA is responsible for managing a Provotum based voting system, their interface to the latter must be responsive in both design and function. One of the most widespread technologies in the world of interactive web applications are JavaScript frameworks, which are often capable of a large amount of tasks. The core concept of a JavaScript framework is to make websites more dynamic and reactive, as well as helping with application state management and data flow in websites. The most popular JavaScript framework as of 2021 is *React*⁴ [46][43]. React is an open source project maintained by Facebook⁵. Provotum 3.0 uses React for the Voter web application, which makes it a sensible choice for this work as well since interoperability with the BC has already been tested. Another benefit of React is the availability of *React Native*⁶. React Native is a Framework that allows building native Mobile applications by using web technologies such as JavaScript. Since the goal of this work is to propose a native mobile application for the voters, this makes the combination of React and React Native a good choice. Keeping all applications close in terms of used technology provides several benefits like code encapsulation which in turn allows for code reusability and eases

⁴<https://reactjs.org>

⁵<https://www.facebook.com>

⁶<https://reactnative.dev>

source maintenance and future collaboration. It also helps to keep the entry barrier into the Provotum ecosystem as low as possible for future developers by introducing as few new and different technologies as possible.

4.3 Voting Authority Frontend

The VA web application is the central interface used for operating the Provotum REV system. The VA must be able to perform all of their tasks from within the application. In the following section, these tasks are described in detail together with their implications and requirements for the GUI.

4.3.1 System Bootstrapping

In order for Provotum to be operable, a sequence of bootstrap steps needs to be performed. The only step that is not achievable through the VA GUI is the starting of the VA backend, which has to be already running in order for the system to work. Thus this process should be made as easy as possible through the chosen method of software distribution. The first step in the bootstrapping phase is for the VA to wait until all sealers have registered themselves. This means that the GUI needs to display the number of sealers that are currently registered. Since the system does not know the number of sealers that should participate within the system, the VA has to decide themselves at which point this number is correct. Afterward, the VA creates a BC specification in JSON format that is used to start a BC node. The VA then starts their BC node with the generated specification. From the VA's point of view, the bootstrap process is now over. The sealers are now able to retrieve the BC specification from the VA and start their own BC node. The specification contains and describes all custom data types and settings, of which the BC needs to be aware of. Finally, the sealers insert their validator keys into the BC to finish the bootstrap phase.

4.3.2 Generating Votes

The data model for a single vote in Provotum consists of a vote title and one-to-many vote topics. Each vote topic describes a question that will be the subject of voting for the voting population along with two possible answers, yes or no. Thus the VA GUI must provide a form for submitting the vote with the relevant information. After a vote has been submitted, the sealers have to submit their public key share for the vote onto the blockchain. This is required for the voting protocol because it is using a distributed key generation mechanism. Finally, the VA combines all submitted public key shares for the vote, moving it into the *voting* phase. This will immediately open the vote to the public to submit their ballots. Thus it is crucial to inform the VA clearly about the implications of this step.

4.3.3 Tallying

Once the voting period for a certain vote is over, the VA may close it from ballot submission by starting the *tallying* phase. Next to closing the vote from the public, this also has the effect of communicating to the sealers that they may now start tallying their share of votes. Once the sealers have finished tallying, the VA may conclude the vote by publishing the results on the blockchain. This effectively makes the results publicly available to anyone.

4.4 Sealer Frontend

The sealers play a central role in the Provotum REV system. They partake in the distributed key generation and are responsible for the PoA consensus mechanism used by the BC. Despite their integral role in the protocol, the sealers have the least amount of actions that need manual user involvement. During the bootstrapping phase, they need to register themselves with the VA and then start their BC node. During normal operation, this only needs to be done once. Afterward, the sealers must submit their key share of the distributed key. This needs to be done once for each vote issued by the VA. Finally, the sealers must be able to detect closed votes and tally the results. Each of these actions can be achieved within a few mouse clicks, leaving the main challenge for the sealer frontend the coherent display of information in a way that makes is easily recognizable where action needs to be taken.

4.5 Voter Application

The voter application is the heart piece of any REV system. It is the piece of the ecosystem that will be interacted with by the largest amount of people. Additionally, the user group of the voter application is very heterogeneous, as little to no assumptions can be taken about the population of a voting area. Thus UI and UX considerations are necessary in order to make the application accessible to as many people as possible. In the following sections, the requirements for the voter application are discussed.

4.5.1 Registration

In a user study, Distler *et al.* [20] concluded that users gain a large amount of their trust in an application from the registration process. They seemed to gain confidence from the process of receiving a registration code in physical form and signing up on the application through that. As registration and eligibility and identity management is still an open area of research with Provotum, the registration process cannot be set in stone in this work either [26]. In order to emphasize the effect of the paper-based registration, a similar approach was used in the design of the voter application.

Assumption: The UX of the voter application assumes that all voters receive the private key for their voter wallet from the identity provisioning service prior to using the application. The design also assumes that the voting officials and the identity provider provide some explanation on the importance of a private key in a BC environment.

The application supports key submission either through typing in an input field or by scanning a QR code. Scanning the code simplifies the process and additionally removes the stress of potentially making typing mistakes in an obscured input field. This also goes in hand with suggestions from literature to not making the user memorize codes [35].



Figure 4.6: Password field with obscured letters, the code may easily be misspelled without detection

Upon the first usage, the user may enter their code through their preferred method in order to access their voter wallet. The application will then ask for permission to store the code in a secure storage on the device. This is a native dialog and most users will have seen this prompt in other applications, further improving confidence in the application. When the application is accessed in the future, a prompt for biometric access through fingerprint or facial detection will be presented instead of the initial screen. If the user decides to not (or is not able to) unlock the biometric prompt they are then redirected to the initial registration screen that was presented on first use. This use of biometric security measures has also shown to improve trust in REV applications [20]. After successful registration, the user is notified that the procedure has taken place as intended and that they are registered accordingly.

4.5.2 Vote Browsing

From a user’s perspective, a vote may be in one of three main conceptual states: Planned, currently open, and closed. In the following, all use cases concerned with votes in each of these states are listed with their implications on the application.

- **General browsing** The user may want to go through a list of votes and see for themselves what votes are currently ongoing, which are planned, or inspect the results of passed votes. The application allows this behavior through a simple list format with textual search and filtering by vote state.
- **Casting a ballot** The user may have a vote in mind that is currently open for ballot submission. They want to use the application to cast their vote. The application supports the user to find the desired vote either by filtering for only vote in the voting phase or by full-text search. When the vote is selected, the screen shows clearly labeled mutually exclusive buttons for *yes* and *no*. If the user is not registered yet, a label hints them to do so first. When the user presses the *submit* button, an

additional prompt will ask them to confirm that they want to submit their ballot, as this is an irreversible action. After the prompt has been accepted, a loading icon appears, letting the user know that the application is doing something and that they should not take any actions until further notification. Finally, when the application receives confirmation that the ballot has been included the user is notified and they now have a clear indication that their ballot has been cast on this vote, including technical information (e.g. block number of the ballot) if they are interested.

- **Result Inspection** The user may want to see results from a past vote. In traditional voting schemes, the only way of reliably inspecting vote results is through official channels of the VA. The protocol design of Provotum and the transparent nature of the BC allow displaying election results directly within the application. Next to the results, a link to the block containing the results is displayed. This allows the user to easily verify for themselves that the displayed information is accurate.

Chapter 5

Implementation

The concepts and designs introduced in Chapter 4 were implemented as a prototype in accordance with this thesis' goals. Additionally, some changes were made to the Provotum 3.0 source code [26] in order to enable compatibility and leverage certain use cases. In this chapter, relevant parts of the source code for both the VA web application and the voter mobile application are explained. The goal is to lead future developers onto the right tracks for making further changes and to clarify implicitly taken software design choices.

5.1 Data Storage Model

Traditionally, web and native development required completely separate technology stacks and toolchains. Web applications were only capable of running logic and interactive code through JavaScript¹. With the rise of client-side frameworks, such as ReactJS², VueJS³, and Angular⁴ web applications are becoming more sophisticated and may even resemble functionality typically expected in native desktop applications. Native mobile applications on the other hand have historically been built through special software for each platform. For Apple devices, a developer would have to use the Swift⁵ programming language and for Android-based devices⁶ Java (or Java derivatives) was required. React Native⁷ is a JavaScript framework that allows writing native mobile applications using technologies and paradigms usually only accessible on web-based applications. Both React and React Native may use the Redux⁸ state management library. A *state management library* is responsible for handling volatile data that is kept in memory while the application is running and is erased upon closing it. With web applications becoming more and more complex, so is the amount of data used and fetched from outside. Without state management, all

¹<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

²<https://reactjs.org>

³<https://vuejs.org>

⁴<https://angular.io>

⁵<https://developer.apple.com/swift/>

⁶<https://developer.android.com>

⁷<https://reactnative.dev>

⁸<https://redux.js.org>

data is lost upon page refresh or navigation. While in some cases (e.g., if the data is of minor volume) repeated fetching the data for each navigation instance may be feasible, this will often result in repeated waiting times for the user and put an increased toll on the network usage. Thus using a state management system allows for keeping consistent and persistent data across the application, as long as it is running on the client-side. Additionally, Redux manages typical storage problems such as asynchronous access and mutability out of the box. A Redux store may be separated into three parts (after the official Redux documentation [10]):

1. **Actions** are objects describing intentions to mutate data. An action object must include the type of action (e.g. *addTodo*) and may include a payload (e.g. *do laundry*). Actions may be *dispatched* from any component outside of the store.
2. **Reducers** are functions listening for certain actions. They receive the current application state and an action and return the new application state as a modified copy. These are responsible for the actual state modifications. Going with the previous example, the reducer for the *addTodo* action will create a copy of the list with the payload appended.
3. **Selectors** describe the interface for data queries from the applications state and may be invoked from outside of the store. For instance, a page may call *selectTodos* in order to display a list of todo items. It is considered best practice to query only the smallest amount of data necessary, which can be supported by the developer by implementing sophisticated filtering reducers to allow for more complex queries.

Both the VA and the voter application use Redux for their state management. According to the developer guidelines, it is good practice to separate the store into logical chunks [10], so-called *slices* [9]. The common piece of data used in both applications is the votes received from querying the Provotum BC. The *VotesSlice* offers all functionality for fetching and managing votes. The interface allows querying all or specific votes, filtered by their identifier or other criteria (e.g., their phase). The *VotesSlice* is consistent between both applications. This central piece of storage for the votes allows keeping the amount of data requested from the BC at a minimum.

Both applications have some additional slices responsible for handling functionality specific to them. The VA application has a *ChainSlice* that tracks the state and properties of the BC. The voter applications store contains additional slices for ballots and user configurations.

5.2 VA Application

As previously mentioned in Section 5.1, the VA application is built using the React framework. The UI design was built adhering to UI design guidelines specific for dashboard applications [21]. In contrast to typical websites, scrolling and page navigation to access information can be considered bad practice in information dashboard design because it

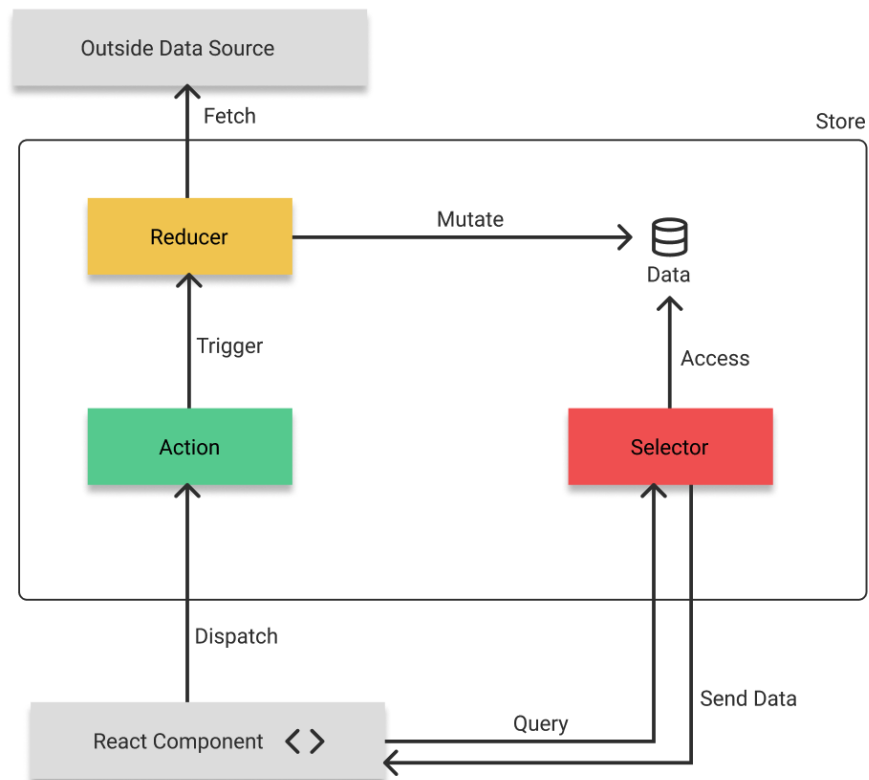


Figure 5.1: Redux Architecture Overview

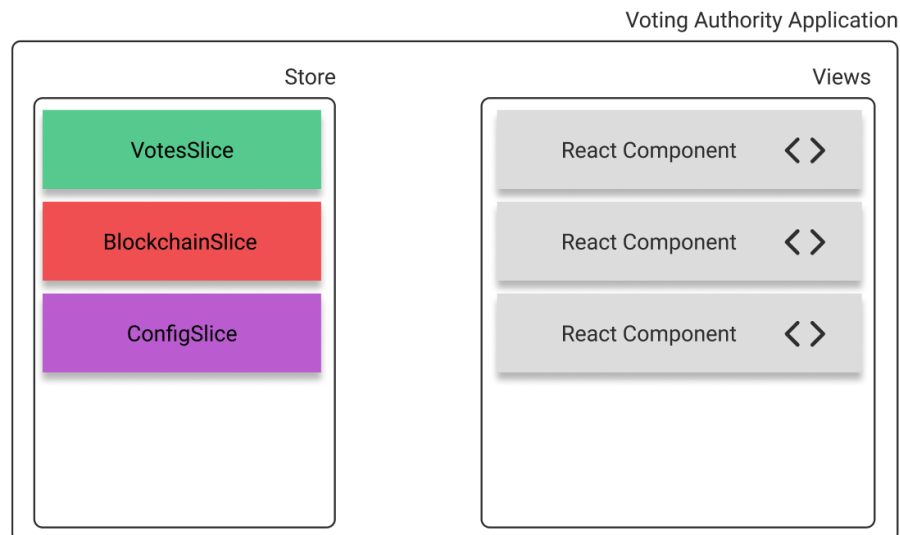


Figure 5.2: Voting Authority Application Store Design

requires the user to memorize certain information that is out of their view [21]. Thus the VA application was designed to keep all information in a single full-screen window. Another guideline for dashboards is *modularity* and *configurability*[21]. This means the user should have the option to enable and disable certain information, as well as rearranging the information on display. Both of these properties are respected in the VA application on a UI design and software design level. On the UI side of things, information is packed

into coherent components called *modules*. Except from a set of modules required for the operation of the system, all modules may be enabled or disabled through associated checkboxes. Modules may be rearranged and re-sized through drag-and-drop operation. Each module may have a minimum required size in the layout grid depending on its content. For instance, a module only showing the number of votes currently live may fit into a one-by-one grid slot, while a module showing voter participation statistics in the form of graphs may need a larger slot.

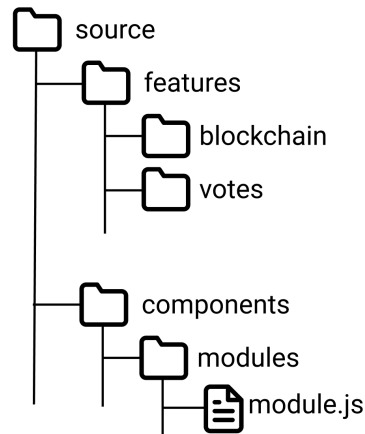


Figure 5.3: Voting Authority Application folder structure

On the software side, the application is designed to minimize the efforts of adding new modules to the dashboard. A developer may simply make a copy of the blueprint component, specify size requirements, and provide their custom markup for the new module. Listing 5.1 shows the code for a minimal module displaying the number of total votes. A module may query all the data it needs from the central store through the *useSelector* hook. By using the provided classes for *module* and (optionally) *simple-module*, the styling and spacing is automatically applied. This helps in providing a consistent experience across all modules. The module developer is supposed to provide the markup and logic for the content into the *module-content* block. Additional styling classes like *module-icon* and *module-information* may optionally be used for even more convenience. The module developer may place their module in the corresponding directory within the software project. The folder structure can be seen in Figure 5.3. The *features* folder contains separated parts of the application state, in accordance with the react developer guidelines and best practices for global application state⁹. These *features* are separate and complete blocks containing the aforementioned slices. All *view* components, which contain actual markup and UI code are located in the *components* directory and the dashboard modules should be located in the *modules* folder.

```

1
2 export function NumVotes(props) {
3   const votes = useSelector(selectVotes);
4
5   return (
6     <div className={`module simple-module num-votes`} >
7       <div className="module-content">

```

⁹<https://redux.js.org/style-guide/style-guide>

```

8         <div className="module-icon">
9             <GroupWork fontSize="large" color="primary" />
10        </div>
11        <div className="module-information">
12            <div className="value">{votes.length}</div>
13            <div className="label">Total Votes</div>
14        </div>
15    </div>
16
17    </div >
18  )
19 }

```

Listing 5.1: Example UI Module

Listing 5.2 shows the *uiBuilder* state slice, which maintains and persists all UI-related information such as the layout of the modules and which modules are enabled. A module developer may register their module here both in the *layout*, as well as in the *states* arrays. These are the data structures responsible for storing the location, size, and activeness of the modules.

```

1  export const slice = createSlice({
2    name: 'uiBuilder',
3    initialState: {
4      // ...
5      layout: [
6        {
7          name: 'module name',
8          i: 'module_id', // unique module id
9          x: 0, // x position
10         y: 0, // y position
11         w: 1, // width
12         h: 1, // height
13         // min & max configurable width/height
14         minW: 1,
15         maxW: 2,
16         minH: 1,
17         maxH: 2,
18       },
19     ],
20     states: [
21       {
22         i: 'module_id',
23         active: true, // if the module is enabled
24         optional: true // if the module may be disabled
25       },
26     ],
27     events: [],
28   },
29   // ...
30 });

```

Listing 5.2: Module Registration

5.2.1 React Hooks

Section 5.1 describes how data is stored within the central state of React applications and, more specifically both the VA application and the voter application. Figure 5.1 shows how the *Selectors* provide an interface to the data to any react component, e.g., a graphical view component. *React Hooks* are recent additions to the React framework that simplify the use of local component state in function components [10]. By using the *useSelector*

hook, a component may access data in the global application state. Listing 5.3 shows an example function component using the *selectVotes* selector through the *useSelector* hook. An advantage of accessing data through this method is that a subscription to the selected data is built. This means that if the data in the global state changes, the component is re-rendered with the new data without any user interaction required and without a visible page refresh. The component displayed in Listing 5.3 for instance would render a text field if no votes are registered in the global state. This would be the case upon starting the application before the votes are loaded from the remote BC. As soon as the votes are loaded, any component using this selector is newly rendered. In this case, the *renderedVotes* function would map each vote onto a visual component and render it.

```

1
2  const MyComponent = ( { } ) => {
3      const votes = useSelector(selectVotes);
4
5      const renderedVotes = votes.map(vote => (
6          <Vote key={vote.electionId} title={vote.title} />
7      ));
8
9      return (
10         <View style={{}>
11             {votes && votes.length > 0 ?
12                 (
13                     renderedVotes
14                 ) :
15                 (
16                     <Text>No votes found yet!</Text>
17                 )}
18         </View>
19     );
20 }
21

```

Listing 5.3: Function component using a React Hook

5.3 Voter Application

The voter application being built with native technologies for mobile devices introduces new technologies into the Provotum ecosystem. Through choosing react native as the technology, the technological dept was attempted to be kept as low as possible. Figure 5.4 shows the design of the global application state, which is the core logic that spans all across the application. The division into parts for votes, the identity provider, ballots, and the voter was somewhat semantically present in the current iteration of Provotum. However, the source code for all parts was interwoven together into a monolith store. This means that the code was difficult to analyze, maintain, or extend for collaborators. With the aforementioned separation of concerns, each part may be extended freely and easily. For instance, if at some point the core data model of Provotum votes changes, only the *VotesSlice* must be adapted in order to store the newly designed votes.

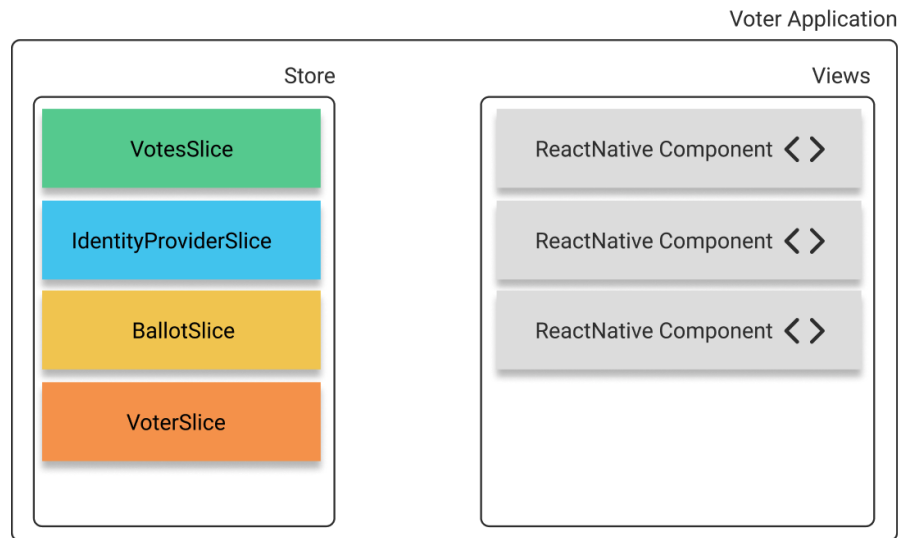


Figure 5.4: Voter Application Store Design

5.3.1 Caching

The voter application supports local caching of some information. Specifically, the identifiers of the blocks (block hashes) containing the user's previously cast ballots are stored on the client device. On one hand this enables UX benefits such as displaying for which votes the user has already cast a ballot or providing a direct link to the block in which a ballot was included. On the other hand this provides a receipt that a user has voted for a certain topic. It is important to note that nothing that is stored on the client device contains information about the content of the vote. This means that the receipt-freeness of Provotum is not challenged by the addition of these features. The application provides single-click access to deleting all of the cached data including the wallet's private key. However, the proposed version of the voter application does not guide the user as to when and why they should do so. This leaves room for possible future iterations of the application, where additional focus on user guidance and briefing may be set.

5.3.2 Cryptography Libraries

In order to interact with the BC, the voter application needs to perform certain cryptographic functionality. There exists a JavaScript library that handles most of these interactions, or at least simplifies the invocation for the developer. This library was however built for systems running on the *nodejs* JavaScript interpreter, which contains some additional libraries. The *Crypto*¹⁰ module implements functionality such as e.g., cryptographically strong random number generation and common hashing algorithms. However, the *Crypto* module is not available on the JavaScript interpreter used for React Native applications. Thus the aforementioned library, that handles BC interactions is not strictly compatible with React Native applications by default. It is however, required for

¹⁰<https://nodejs.org/api/crypto.html>

communication with ProvoTum. This problem was solved by the use of *Shims* and *Polyfills*. A Shim is code that changes calls to any API automatically to support the interface required by the latter [5]. Polyfills are additions to the code that allow to support certain functionality that would not have been supported otherwise [5]. Listing 5.4 displays parts of the *shim* file for the voter application. On lines one and two, some issues are fixed by simply providing definitions for global variables that are not present in the React Native JavaScript runtime. These do not introduce new functionality and code. On the next lines, some of the libraries required for the interaction with ProvoTum and the substrate BC are imported and registered withing the *global* name space.

```

1  if (typeof __dirname === 'undefined') global.__dirname = '/'
2  if (typeof __filename === 'undefined') global.__filename = ''
3
4  if (typeof process === 'undefined') {
5    global.process = require('process')
6  } else {
7    const bProcess = require('process')
8    for (var p in bProcess) {
9      if (!(p in process)) {
10         process[p] = bProcess[p]
11      }
12    }
13  }
14
15  process.browser = false
16  if (typeof Buffer === 'undefined') global.Buffer = require('buffer').Buffer
17  // ...

```

Listing 5.4: Enabling additional functionality

Chapter 6

Evaluation

Evaluation of GUIs and client applications may be done through many different techniques like *heuristic evaluation* or *user testing* [37][27]. The software produced in the context of this thesis was qualitatively evaluated through a heuristic evaluation. This technique has been shown [27] to capture many typical UI problems while being inexpensive, as it does not require additional human participants. Additionally, a use case evaluation was performed in order to show how the use cases defined in Chapter 4 have been covered by the implementation.

6.1 Heuristic Evaluation

This section discusses the results of the heuristic evaluation, as introduced in Chapter 1. Firstly, the VA frontend is discussed, followed by the voter application.

6.1.1 VA frontend

Table 6.1 shows the summary of the heuristic evaluation of the VA web application. In the following paragraphs, the findings are summarized and explained where further context is necessary.

H1 The system’s status is always presented through the *System Status* module displaying whether or not the Provotum BC is currently running. When the dashboard is in rearrangement mode, all modules receive a resize indicator on the bottom right and the hand icon responsible for enabling the mode turns green, indicating the status. Potential problems have been found with creating new votes and changing a vote’s phase, where loading icons could improve the display of the current status and confirmation messages are missing for some actions. **H2** As the VA frontend is intended for educated and trained administrators, the usage of technical language can be considered justifiable. The VA is supposed to be familiar with the phases of a vote and know the rough purpose of a sealer. If the system were to be adapted for a real-world scenario, the language and terms might











ID	Heuristic	Addressed with	Fulfilled
H1	Visibility of system status	BC indicator, mode indicator	
H2	Match between system and the real world	Intended technical language	
H3	User control and freedom	By design	
H4	Consistency and standards	Consistent color palette & design language	
H5	Error prevention	General software design, client-side error detection	
H6	Recognition rather than recall	Single-page design	
H7	Flexibility and efficiency of use	Highly flexible UI arrangement, customizable dashboard	
H8	Aesthetic and minimalist design	Option of disabling UI modules	
H9	Help users recognize, diagnose, and recover from errors	BC restart	
H10	Help and documentation	-	

Table 6.1: Heuristic evaluation keypoints for VA frontend

need adaptation as well. For instance, it could be a subject of future work to rename the sealers into *cantons* for deployment in the context of Swiss national votes.

H3 Nielsen [6] mentions the ability to cancel or undo actions in context with this heuristic. This is problematic with Provotum as the inherent nature of the BC does not allow removing data and neither does the Provotum protocol. **H5** The application is designed to capture most errors and faulty actions already on the client. There are however still errors where the problem could not have been found, potentially lies within the VA backend, or the inner workings of the Provotum BC. **H6** This heuristic is respected by the nature of the dashboard design approach after Few [21].

H7 & **H8** are both addressed through the customizable UI design pattern. By allowing the administrator to arrange their preferred layout and arrangement of modules the application is designed to reduce UI clutter and cater to both experienced as well as new users. **H9** Error recovery may be problematic for a user without the required domain knowledge in BC and programming. While some problems may be temporarily fixed by restarting the BC node, additional support will be required for long-term solutions. **H10** The topic of documentation and user training was determined out of the scope for this work.

6.1.2 Voter Application

Table 6.2 shows the summary of the heuristic evaluation of the voter mobile application. In the following section, the findings are summarized and explained where further context is necessary.

H1 The voter application displays animated loading screens and icons for each process. This addresses the main problems that occur with this heuristic. Nielsen [6] also states that a system should not take any actions without informing the user. The registration process with Provotum technically contains multiple sub-processes (e.g., blinding the user's address through the IDP). In order to reduce the mental load on the user and improve usability, these sub-processes are automated. As recommended by Distler et al. [20], security-relevant information should ideally be shown to the user as it has shown to improve trust. The application displays loading screens for each step. The loading screens are displayed for a minimum amount of time even if the actual action may be finished earlier in order to prevent rapid flashing of unreadable content. **H2** The voter application uses natural language equivalents for the voting phases in order to make use of the mental model of voting that is present with most people. **H3** Even though actions taken on the BC are irreversible, the application aims to maximize user control by requiring confirmation for all irreversible actions, such as casting a ballot.

H4 The voter application uses native UI elements and a consistent color palette to provide a consistent feeling and respect platform standards. The color palette has also been evaluated in terms of accessibility using the adobe color accessibility checker¹. **H5** The application is designed to capture user-caused errors on the client-side. While most errors directly caused within the application should be captured, little can be done from the client application to prevent backend errors. **H6** The application does not require the user to memorize any codes, which has also been found to be problematic with one of the REV systems previously deployed in Switzerland [35]. In certain situations, the user is required to memorize their location within the application. This has been addressed by providing titles on each screen, e.g., for the selected vote. If the user chooses to clear their locally cached data as described in Section 6.2, they are required to memorize the votes for which they have already cast their ballot.

H7 There are no shortcuts provided other than what is given by the native operating system, which may slow down an experienced user. **H8** The application does not display any information or UI elements that do not serve the purpose of either displaying information or acting as a control element. **H9** The most important way of recognizing errors for the user is to check if their ballot has been successfully recorded by the BC. They are encouraged to do so through a button appearing after ballot submission. In the case of severe problems with the application on the client-side, there are no built-in mechanisms other than a hard reset. **H10** is considered out of the scope for this work and has not been respected in the voter application either.

¹<https://color.adobe.com/create/color-wheel>











ID	Heuristic	Addressed with	Fulfilled
H1	Visibility of system status	animated loading icons, status icons	
H2	Match between system and the real world	Natural language, native icon language	
H3	User control and freedom	Always available navigation, confirmation dialogues	
H4	Consistency and standards	Consistent color palette, native UI elements & buttons	
H5	Error prevention	General software design, client-side error detection	
H6	Recognition rather than recall	No memorization of codes	
H7	Flexibility and efficiency of use	General design	
H8	Aesthetic and minimalist design	No unnecessary information	
H9	Help users recognize, diagnose, and recover from errors	Block inclusion link	
H10	Help and documentation	-	

Table 6.2: Heuristic evaluation keypoints for voter application

6.2 Use Case Analysis

In the following paragraphs, a list of use case scenarios for stakeholders in the role of the (i) VA, and (ii) the voter is presented. With each scenario, a workflow from the software shows how the goal may be achieved. Additional information on potential problems and variations in the use case is provided as well, in order to cover as many situations as possible.

6.2.1 VA Frontend

Bootstrapping

Provotum requires two major software components to be running in order to operate: The VA backend and the Provotum BC. As everything is controlled via the VA backend, it must be running in advance. The Provotum BC may then be started through the VA Frontend by clicking the labeled button. Figure 6.1 shows the UI for this process. The entire bootstrapping process from the VA's point of view is handled by this singular action, in order to simplify it as much as possible. If so desired, the VA may consult additional modules, e.g., the *peer information module* for more information before completing this step. This may help the VA to decide whether all dedicated sealers have registered themselves already.

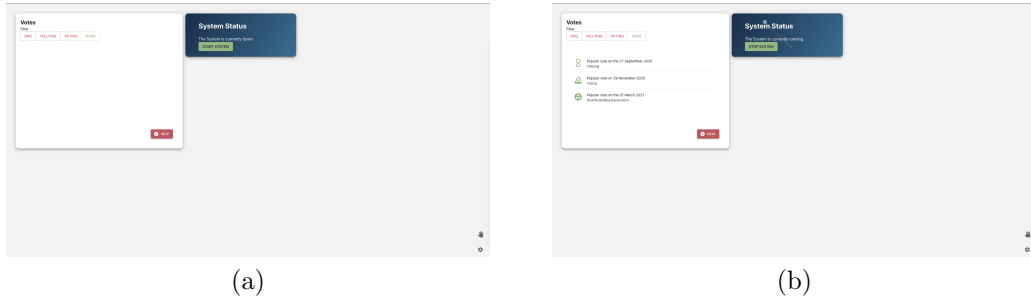


Figure 6.1: Starting the Provotum BC through the UI.

Inspecting Votes

A defining property of the BC is immutability, meaning that the list of votes will grow over time with more and more votes being created and administered. With that comes the need for filtering and searching for specific votes. Thus one use case is defined as manipulating and selecting the displayed votes. Figure 6.2 shows that this is easily achievable through the mutually exclusive filter buttons on top of the vote list. In comparison to Figure 6.1 (b), now only the votes currently in the *voting phase* are displayed. In future work, this could be improved by additional options for full text searching and sorting options.

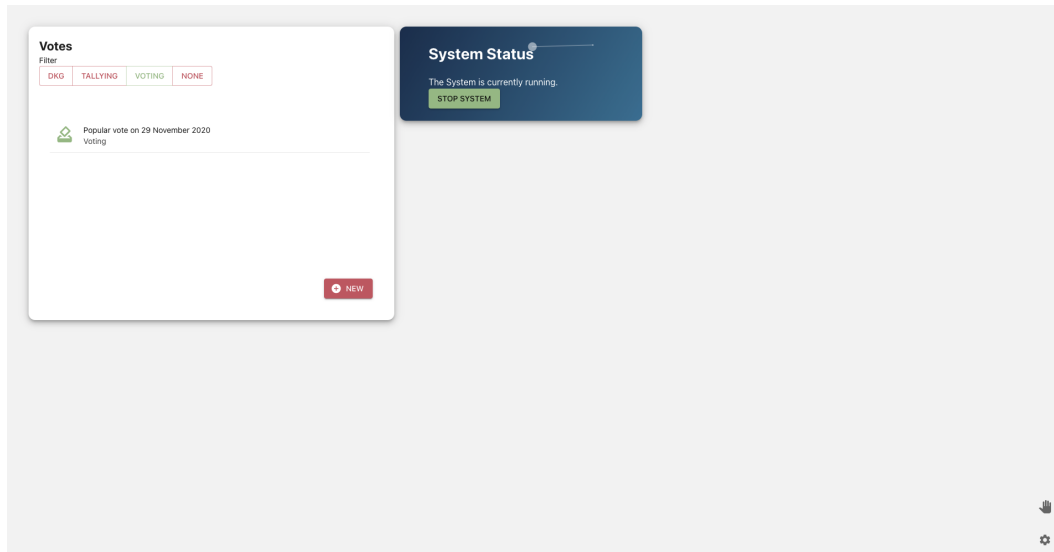


Figure 6.2: Filtering by phase of a vote

Creating Votes

Creating new votes is one of the central tasks of the VA. The data model used in the Provotum protocol allows each vote to have multiple topics that may be voted for. Thus the UI must support the creation of new votes and the assignment of the vote topics. Figure 6.3 shows the form offered for this scenario. The vote's title, as well as one topic, has been supplied here, leaving the field for more topics open again. Finally, the vote may be submitted to the Provotum BC and thus published.

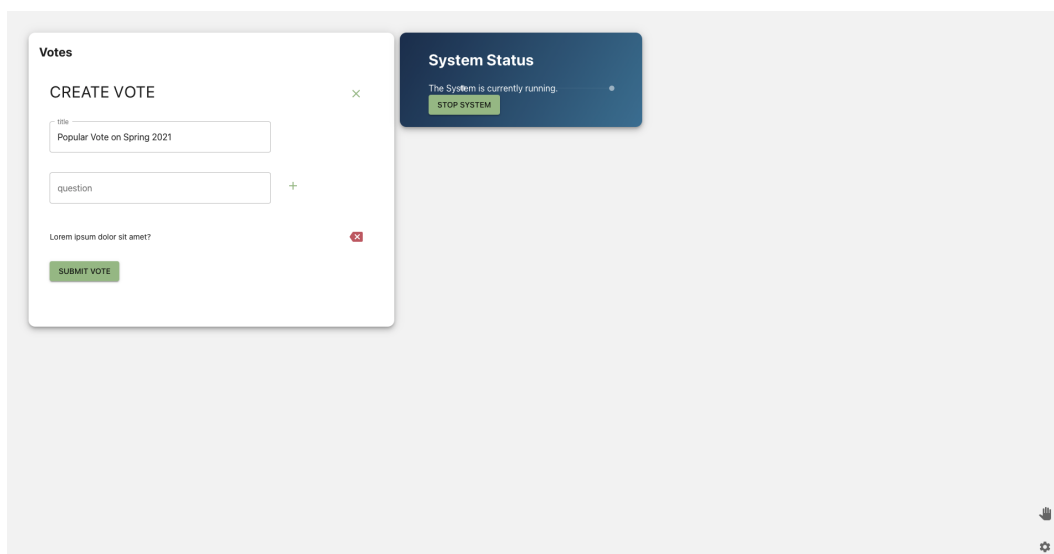


Figure 6.3: Creation of a new vote

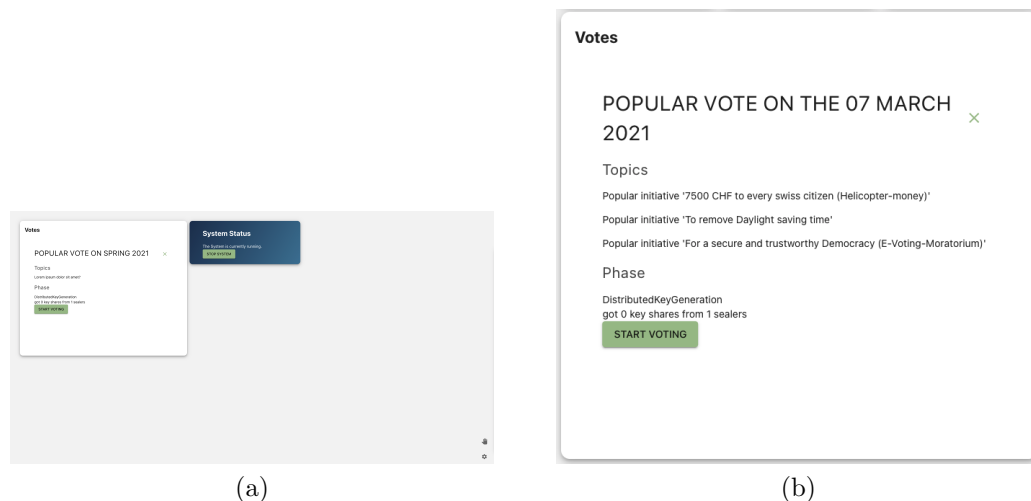


Figure 6.4: Opening a vote. (a) shows the detail screen embedded in the dashboard and (b) zoomed in on the details.

Opening Votes

After a vote has been created the VA is tasked with managing the vote's lifecycle. Initially, the vote is in the *distributed key generation phase*. During this phase, the sealers must provide their share of the distributed vote key. Previously with Provotum, the VA had no way of telling if and by how many of the sealers this has been done other than manual communication outside of the system. With the VA frontend, a vote's details may be accessed by clicking on it with the cursor. Figure 6.4 displays the vote details. The system displays to the VA, how many sealers are registered with them and how many of the registered sealers have already participated in the DKG mechanism. If the VA is satisfied with the number of sealers and the time is right, they may open the vote for the voters to cast their ballots.

Closing Votes

When the official voting period has ended, Provotum requires the VA to manually close the vote. As Provotum does not have a concept of time in votes in its current iteration, this must be manually tracked by the VA outside of the system. Figure 6.5 shows the detail screen for a vote currently in the *voting phase*. As this is the next logical step in the voting procedure as described in 2, the vote may be closed through the aptly labeled button.

Result Inspection

With the current version of Provotum, there was no user-friendly way of inspecting vote results. The VA had to either use the voter platform or manually inspect the content of the vote finalization block on the Provotum BC. Figure 6.6 shows the vote detail screen

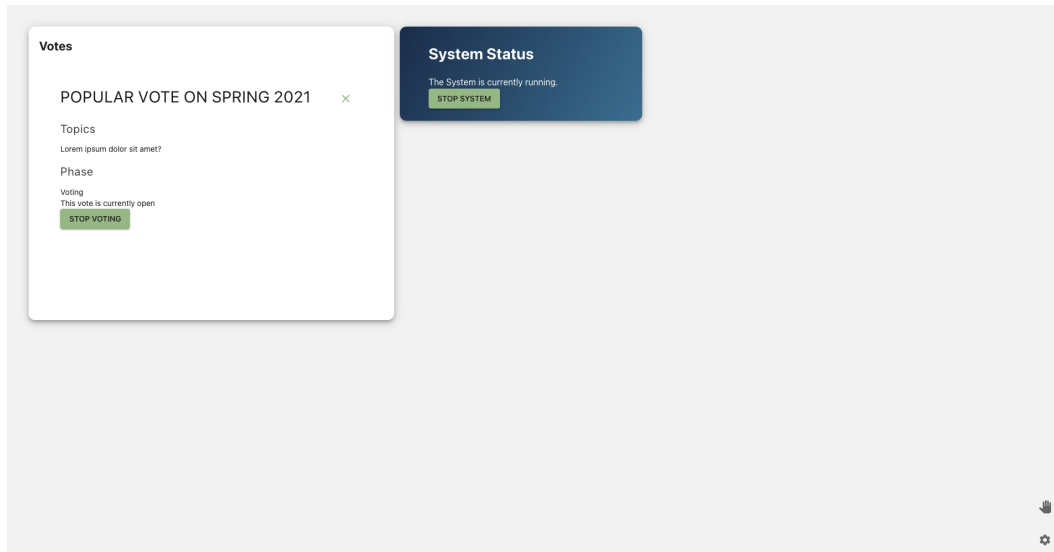


Figure 6.5: Stopping and finishing a vote

for a finished vote, showing the exact number of votes for each answer on each topic. As this vote is in its final form, there are no more actions to be taken here.

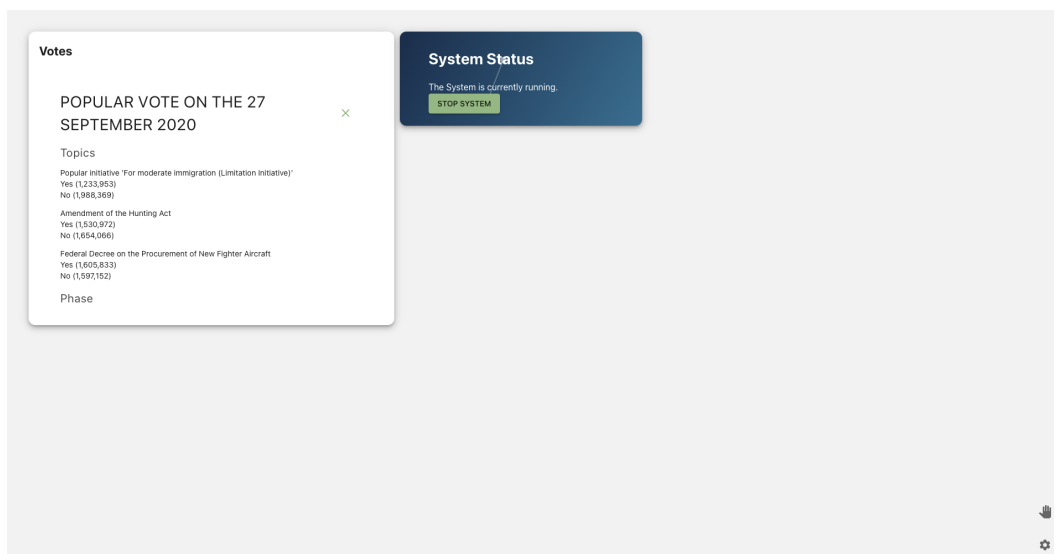


Figure 6.6: Inspecting vote results

Customization

As described in Chapter 4 and Chapter 5, the VA dashboard is designed to be customized in accordance with administration dashboard design guidelines [21]. The system is designed to accept arbitrary modules that can be plugged into the layout. A developer of an additional module may specify minimum and maximum sizes of their module in the grid and define if the module is optional. Figure 6.7 shows the dialog for selecting active modules. The core module required for managing the votes may not be disabled in order

to prevent any confusion for the operator. The position and size of each module may be customized by the VA in *layout* mode, which can be toggled through the hand icon on the bottom right. The hand icon, as well as a layout containing a multitude of modules, can be seen in Figure 6.8. As described in Chapter 5, each module has minimum size constraints defined by the developer in order to maintain its usability at all times. For instance, although the *voter participation* module will responsively scale down with its size to some extent, it will be rendered useless when scaled to small as nothing will be readable.

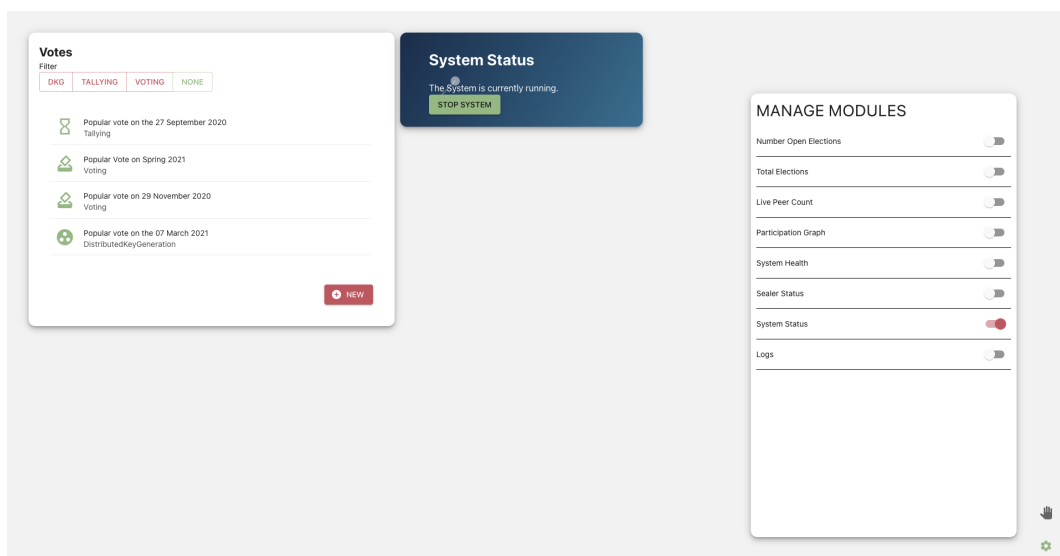


Figure 6.7: Choosing which modules should be enabled

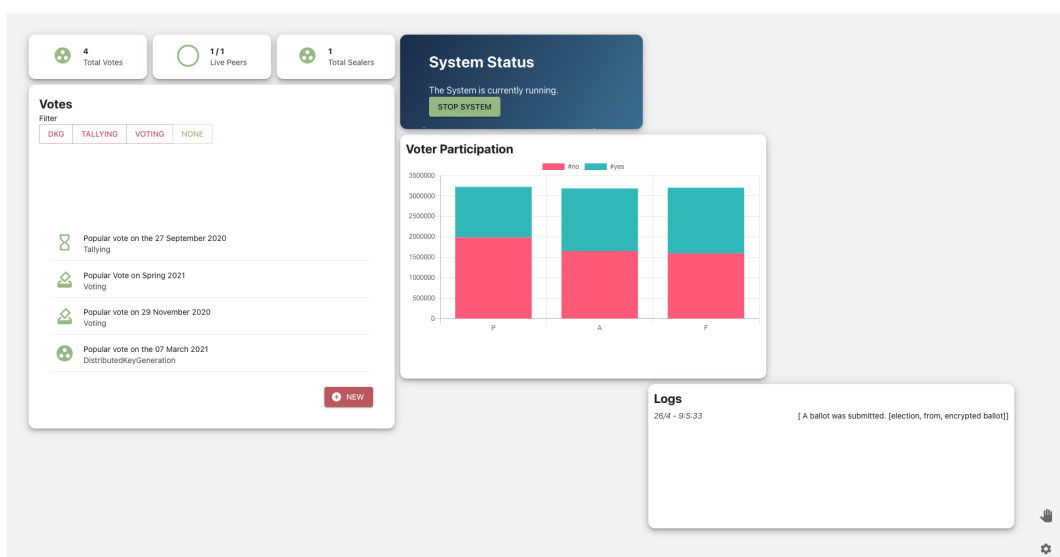


Figure 6.8: Multiple modules arranged in a custom order

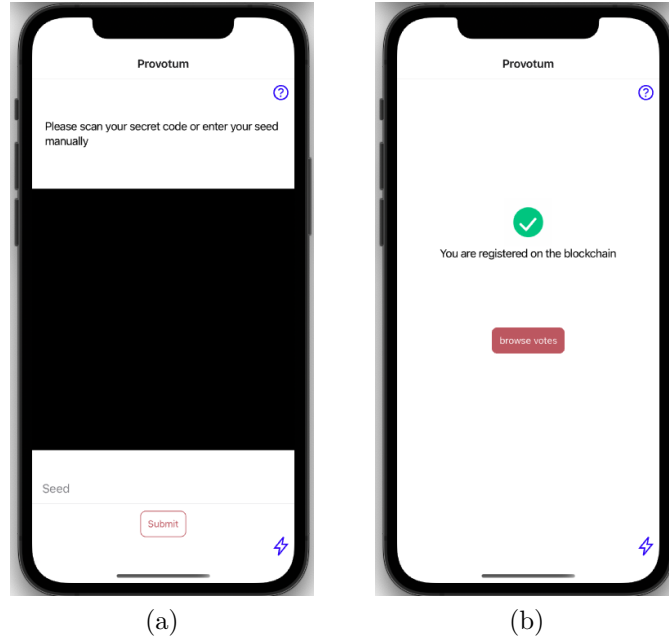


Figure 6.9: Registering the user’s wallet (a) and the message showing a successful registration (b).

6.2.2 Voter Application

Registration

Registration with Provotum works by generating a BC wallet through a provided private key and registering the wallet with the VA and the IDP. In the current state of Provotum, the wallet’s private key is stored in a centralized database on the VA server, accessed by the voter through a conventional login system with email and password. The voter application supports direct entering of the private key by the user, either through a text field or scanning a QR code. Due to development constraints with the iOS emulation software for testing applications [1], the text entry method was used during development. Figure 6.9 shows the camera for scanning the QR code and the text field for manual entering on the left side. On the right side, the screen for a successful registration is displayed. The application supports local storage of the wallet’s private key in an encrypted secure-storage of the device. Upon opening the application a second time, the user is prompted with biometric authentication for accessing the wallet, which could also not be extensively tested during development due to emulator constraints [1].

Browsing votes

Similarly to the VA, the voter must have a convenient way of browsing and finding votes. This is managed in a very similar way to the VA dashboard, although adapted to mobile-specific UI elements. As seen in Figure 6.10, the list of votes may be filtered by the voting phase. This way, a voter can easily navigate to votes that are currently open for ballot submission or votes that have results available. The application also supports a full text

search which incorporates both vote title as well as titles of a votes' topics. The vote phases are given names in natural language instead of the phase titles used for the VA dashboard (e.g., *upcoming* instead of *Distributed key generation* for votes that are created but not opened for voting yet).

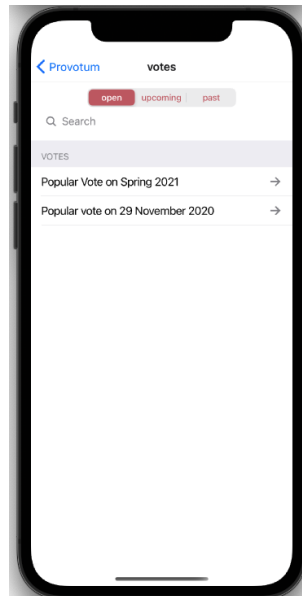


Figure 6.10: Vote list display with filtering and searching

Casting a ballot

The most central use case for the voter application is casting a ballot. When a vote that is currently in the voting phase is selected, the user is prompted with a mutually exclusive selection for each topic. Upon selecting the submit button, the user must confirm their intention to cast their vote through an additional prompt, as displayed in Figure 6.11. After successful submission, the application shows a clearly labeled card, indicating that the ballot is stored on the BC according to protocol. As discussed in Figure 3, encouraging the user to verify the vote's legitimacy is important. Thus a direct link to the block containing the vote is provided, as seen in Figure 6.12. Finally the application locally stores the information for which votes a ballot has been cast. Note that only the hash of the including block is stored and thus the local storage does not introduce potential leakage of vote contents. This storage enables the application to display the votes that have been cast by the user in the vote list.

Inspecting vote results

The high amount of transparency gained from a BC-based system like Provotum enables anyone to inspect vote results. Either by manually inspecting the contents of the publishing block or through any third-party GUI. The voter application contains very minimal vote results, providing just the information on which topics have been accepted

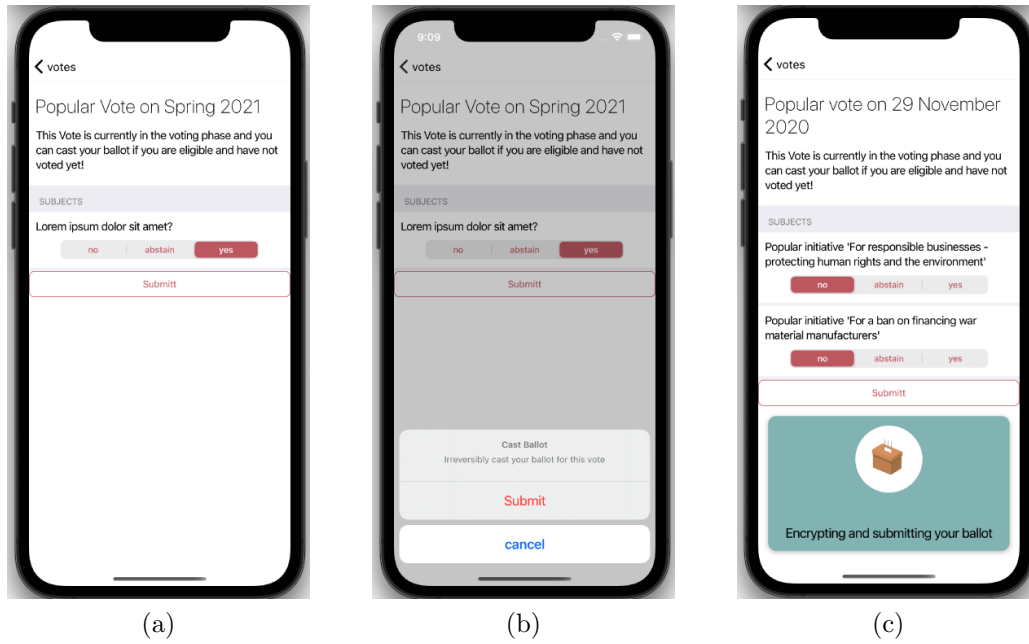


Figure 6.11: (a) Selecting an answer for a vote, (b) an additional prompt before ballot submission, and (c) a loading screen while the ballot is processed

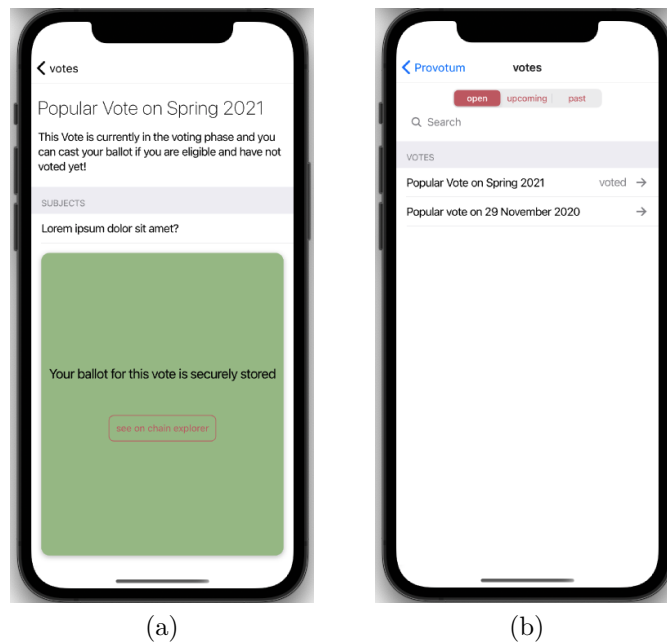


Figure 6.12: Confirmation that the ballot is recorded (a) and new indication in the vote list that a ballot for this vote has been cast (b)

or declined, as well as basic graphics that provide a rough image of the magnitude of the result. These screens are shown in Figure 6.13.

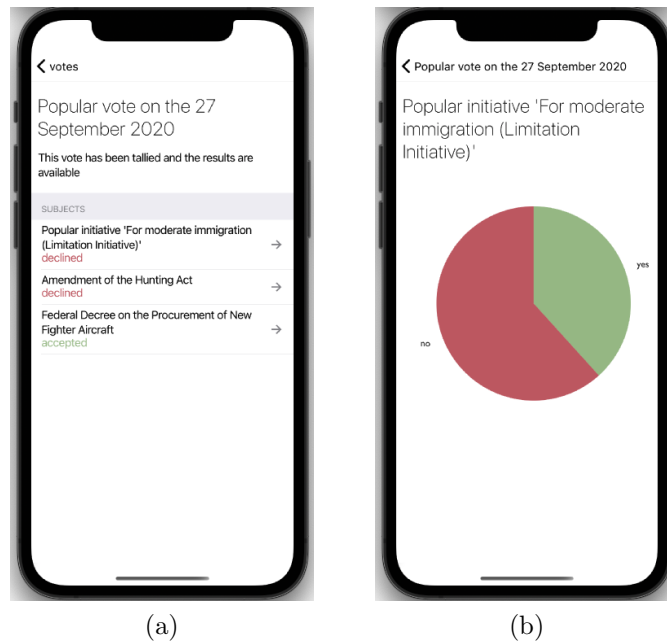


Figure 6.13: A vote that has been closed with it's topics and their results (a), and a pie chart with the results for a selected topic (b)

Removing data

The user should be in control of their local data and be easily capable of removing everything from their device after casting a vote. This is supported by the application through the flash icon on the lower right corner of the successful registration screen, seen in Figure 6.9. Upon tapping the button, an additional prompt is shown to the user in order to prevent unwanted deletion.

Chapter 7

Summary and Conclusions

The goal of this thesis was to improve Provotum 3.0's usability by the prototypical implementation of system interfaces for the VA and the voters. Initially, this meant gathering and compiling information from research in UI/UX design, REV systems, and dashboard applications. One of the contributions of this work is the compilation of information in these topics in the context of a REV system. From various sources from within the REV research field, as well as from outside, recommendations and design guidelines were gathered and summarized. With this information, a prototypical attempt at developing a design system for the VA application was made. The proposed application adheres to administration dashboard design guidelines and plugs seamlessly into the Provotum 3.0 ecosystem. It enhances the REV system's usability significantly by removing the need for technical and domain-specific knowledge from the user. Additionally, the voter mobile application was proposed. By incorporating UI design guidelines and findings from REV research, it was designed to be usable by and accessible to as many people as possible. Like the VA application, it may be integrated into Provotum 3.0 easily as it does not introduce new limitations and dependencies. The voter application acts independently of the VA application, only communicating with the BC itself, the Randomizer, and the IDP. Finally, an evaluation of the proposed UI designs based on the heuristics introduced in Chapter 2 and the use-cases introduced in Chapter 4 concludes this thesis. The following section discusses the goals achieved and the challenges faced during this thesis. Finally, this thesis is concluded with a prospect into possible scenarios for future work on this usability-enhanced version of Provotum 3.0.

7.1 Final Considerations

In order to decide whether the goals of this thesis have been achieved, they should be broken down into the two major parts: The VA application and the voter application. As shown in 6, the VA application has been implemented as a working prototype, covering all use-cases required for operation of Provotum 3.0. The main motivation for the VA application was to improve the ease-of-use for Provotum 3.0 to a degree where a non-technical person is able to operate it. This has been achieved through the VA dashboard

as well. The implementation is of prototypical nature and as such, still has some issues that could not be addressed during this thesis. Since the applications plugged into the large existing code-base of Provotum 3.0, resolving some of these issues would have required serious changes to the existing implementation. The second aforementioned goal was the voter application. The goal of bringing Provotum to the popular platform of smartphones as a stand-alone application has been achieved. Chapter 6 shows how the use-cases for the application are addressed and implemented in the software. An objective for the voter application was also to emphasize the fact that the system is running on a BC by displaying as much information about it to the user as possible. To some extent, this was omitted due to the UI design guidelines discussed in Chapter 3. However, it also turned out to pose more implementation effort than expected to get the relevant data from a BC system, which was not designed for delivering this data.

Upon starting this thesis, the main difficulties were posed by getting a hold in the Provotum ecosystem. Provotum was developed over multiple versions and student projects and, as of time writing this thesis, had multiple active branches of work. Later on, while getting a better understanding of Provotum and REV systems in general, the difficulties moved from conceptual understanding onto technical challenges. Although Provotum 3.0 was well prepared and documented for deployment and large scale application, less information about development was available. Thus during implementation, technical challenges appeared from software design choices in Provotum 3.0. These challenges usually originated from use-cases mismatches between Provotum 3.0 and this thesis. Overcoming these challenges either meant significant changes to the Provotum source code, which was not always feasible, or compromises within the new applications. However to some extent being faced with these challenges also helped with the produced code. The applications were designed with future modifications and adaptations in mind, trying to ease this process for future developers. The deployment instructions provided with the code also put more emphasis on a development workflow, rather than a production scenario.

7.2 Future Work

With any prototypical implementation, there is room for improvements and additions. This final section lists areas of the proposed software where potential for additions and improvements has been identified, or certain aspects had to be declared out-of-scope.

7.2.1 User Guidance

The voter application tries to be as self-contained and easy to use as possible. However, with a topic as complex as REV, additional guidance and instructions may be unavoidable. For instance, the voter application still operates under the assumption that the user has been briefed by the VA on how the REV system works and on what parts are important for the voter. The application supports and already contains functionality for a tutorial upon opening the application for the first time. This could be expanded with more relevant information and a more in-depth explanation in a future iteration. Additionally, a tutorial

video could be produced and distributed alongside the application. The idea of creating an explanation video has also been thrown up in a study that evaluated the UI of the *Helios Voting System* [29].

7.2.2 Language & Terminology support

Both applications are currently only available in the English language and all of the informational text is hard-coded. In future efforts, support for multiple languages and substitution of text strings could be implemented in order to improve the flexibility of the applications.

7.2.3 Further accessibility improvements

Although much effort has been put into making the voter application accessible to as many people as possible, there is still room for improvement. In order for people with heavy visual impairments that hinder their ability to read, text-to-speech synthesis and specific input patterns would be required, which could not be implemented within the scope of this thesis. Sensible defaults for typesetting and colors have been chosen, adhering to accessibility guidelines. This could be further improved by providing options for customizability.

7.2.4 Enhanced information modes

The voter application currently only displays a minimum of information that is intended for display by the Provotum protocol. However, more information may be aggregated from the public BC such as e.g., block confirmation times and block hashes. This information could be included within the voter application in order to enable educated users to verify the information for themselves.

7.2.5 State and Information Modelling

Multiple features intended to make it into the proposed applications had to be omitted due to missing support from the Provotum 3.0 core. Examples thereof are missing interfaces for additional data, minimal data models, and no system state information. This is no critique for Provotum 3.0, as it originates from its intended usage through HTTP requests and simply has not been in the projects scope. Nonetheless, addressing these issues on the backend side of the REV system could greatly improve the capabilities and possibilities of the respective UIs.

Bibliography

- [1] Apple simulator documentation. <https://help.apple.com/simulator/mac/11.0/>. Accessed: 2021-06-16.
- [2] E-government. <https://www.egovernment.ch/en/umsetzung/schwerpunktplan/vote-electronique/>. Accessed: 2021-06-03.
- [3] E-voting switzerland. <https://www.bk.admin.ch/bk/en/home/politische-rechte/e-voting.html>. Accessed: 2021-03-09.
- [4] Landsgemeinde appenzell. <https://www.ai.ch/politik/landsgemeinde>. Accessed: 2021-04-12.
- [5] Mozilla glossary. <https://developer.mozilla.org/en-US/docs/Glossary/>. Accessed: 2021-06-16.
- [6] Nielsen ten usability heuristics. <https://www.nngroup.com/articles/ten-usability-heuristics/>. Accessed: 2021-05-18.
- [7] Post press release on suspension. <https://www.post.ch/en/about-us/media/press-releases/2019/swiss-post-temporarily-suspends-its-e-voting-system>. Accessed: 2021-03-09.
- [8] Provotum. <http://provotum.ch>. Accessed: 2021-03-09.
- [9] Redux toolkit website. <https://redux-toolkit.js.org/>. Accessed: 2021-05-19.
- [10] Redux website. <https://redux.js.org/>. Accessed: 2021-05-19.
- [11] Swiss post e-voting system. <https://www.evoting.ch/en>. Accessed: 2021-02-24.
- [12] Swiss post suspended press release. <https://www.post.ch/en/about-us/media/press-releases/2019/swiss-post-temporarily-suspends-its-e-voting-system>. Accessed: 2021-05-10.
- [13] The swiss authorities online e-voting. <https://www.ch.ch/en/demokratie/voting-online/what-is-e-voting/>. Accessed: 2021-02-24.
- [14] W3 accessibility principles. <https://www.w3.org/WA>. Accessed: 2021-05-07.

- [15] Ali Balapour, Hamid Reza Nikkhah, and Rajiv Sabherwal. Mobile application security: Role of perceived privacy as the predictor of security perceptions. *International Journal of Information Management*, 52:102063, 2020.
- [16] Eric Bergman and Earl Johnson. Towards accessible human-computer interaction. *Sun Microsystems Laboratories The First Ten Years*, 2001.
- [17] Jennifer Birch. Worldwide prevalence of red-green color deficiency. *Journal of the Optical Society of America. A, Optics, image science, and vision*, 29:313–20, 03 2012.
- [18] Nadja Braun and Daniel Brändli. Swiss e-voting pilot projects: Evaluation, situation analysis and how to proceed. In Robert Krimmer, editor, *Electronic Voting 2006 – 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC*, pages 27–36, Bonn, 2006. Gesellschaft für Informatik e.V.
- [19] S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop (CSFW’06)*, pages 12 pp.–42, 2006.
- [20] Verena Distler, Marie-Laure Zollinger, Carine Lallemand, Peter B. Roenne, Peter Y. A. Ryan, and Vincent Koenig. Security - visible, yet unseen? In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI ’19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery.
- [21] Stephen Few. *Information dashboard design: The effective visual communication of data*, volume 2. O’reilly Sebastopol, CA, 2006.
- [22] David Galindo, Sandra Guasch, and Jordi Puiggali. 2015 neuchâtel’s cast-as-intended verification mechanism. In *International Conference on E-Voting and Identity*, pages 3–18. Springer, 2015.
- [23] Wilbert O Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.
- [24] Hillary Goretta, Betty Purwandari, Larastri Kumaralalita, and Oldyson Tri 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)*, pages 143–149, 2018.
- [25] Marc Hassenzahl. User experience and experience design. *The encyclopedia of human-computer interaction*, 2, 2013.
- [26] Alex Hofmann. Security analysis and improvements of a blockchain-based remote electronic voting system. 2020.
- [27] Robin Jeffries, James R Miller, Cathleen Wharton, and Kathy Uyeda. User interface evaluation in the real world: a comparison of four techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 119–124, 1991.
- [28] Hugo Jonker and Jun Pang. Bulletin boards in voting systems: Modelling and measuring privacy. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 294–300, 2011.

- [29] Fatih Karayumak, Michaela Kauer, M Maina Olembo, Tobias Volk, and Melanie Volkamer. User study of the improved helios voting system interfaces. In *2011 1st Workshop on Socio-Technical Aspects in Security and Trust (STAST)*, pages 37–44. IEEE, 2011.
- [30] Simeon Keates, P. John Clarkson, Lee-Anne Harrison, and Peter Robinson. Towards a practical inclusive design approach. In *Proceedings on the 2000 Conference on Universal Usability*, CUU '00, page 45–52, New York, NY, USA, 2000. Association for Computing Machinery.
- [31] Akif Khan, Shah Khusro, and Iftikhar Alam. Blindsense: An accessibility-inclusive universal user interface for blind people. *Engineering, Technology & Applied Science Research*, 8(2):2775–2784, 2018.
- [32] C. Killer, B. Rodrigues, E. J. Scheid, M. Franco, M. Eck, N. Zaugg, A. Scheitlin, and B. Stiller. Provotum: A blockchain-based and end-to-end verifiable remote electronic voting system. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 172–183, 2020.
- [33] Post CH Ltd. Post e-voting: system documentation. <https://www.post.ch/en/business-solutions/e-voting>, 2019. Accessed: 2020-02-16.
- [34] Aaron Marcus. Cross-cultural user-experience design. In Dave Barker-Plummer, Richard Cox, and Nik Swoboda, editors, *Diagrammatic Representation and Inference*, pages 16–24, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [35] Karola Marky, Verena Zimmermann, Markus Funk, Jörg Daubert, Kira Bleck, and Max 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*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.
- [36] Raphael Matile and Christian Killer. Privacy, verifiability, and auditability in blockchain-based e-voting, 2018.
- [37] J. Nielsen. Iterative user-interface design. *Computer*, 26(11):32–41, 1993.
- [38] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158, 1994.
- [39] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems*, pages 413–414, 1994.
- [40] M Maina Olembo and Melanie Volkamer. E-voting system usability: Lessons for interface design, user studies, and usability criteria. *Human-Centered System Design for Electronic Governance*, pages 172–201, 2013.
- [41] Nikolay Pavlov. User interface for people with autism spectrum disorders. *Journal of Software Engineering and Applications*, 2014, 2014.

- [42] Luz Rello and Ricardo Baeza-Yates. Good fonts for dyslexia. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13, New York, NY, USA, 2013. Association for Computing Machinery.
- [43] statistica. web framework statistics. <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>. Accessed: 2020-04-07.
- [44] Debbie Stone, Caroline Jarrett, Mark Woodroffe, and Shailey Minocha. *User interface design and evaluation*. Elsevier, 2005.
- [45] Substrate. substrate. <https://substrate.dev/docs/>. Accessed: 2020-04-07.
- [46] towardsdatascience. popular web frameworks. <https://towardsdatascience.com/top-10-in-demand-web-development-frameworks-in-2021-8a5b668be0d6>. Accessed: 2020-04-07.
- [47] WHO. Who on visual impairment. <https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment>. Accessed: 2021-04-01.
- [48] Mathieu Zen and Jean Vanderdonckt. Towards an evaluation of graphical user interfaces aesthetics based on metrics. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12. IEEE, 2014.

Abbreviations

BC	Blockchain
CLI	Command line interface
DL	Distributed ledger
GUI	Graphical user interface
HCI	Human computer interaction
JSON	JavaScript Object Notation
NIZKP	Non-interactive zero knowledge proof
PBB	Public bulletin board
PoA	Proof of authority
PoW	Proof of work
REV	Remote electronic voting
RPC	Remote procedure call
UI	User interface
UX	User experience
VA	Voting authority
WHO	World health organization

Glossary

Backend The parts of an application that are hidden from the user.

Ballot A way of submitting a vote secretly, usually through obscuring.

Block A data structure that contains a set of information and a link to the previous block in a blockchain.

Blockchain (BC) An append-only distributed data structure.

Cast-as-intended Voting property that ensures that the voter may verify that their ballot contains the same information after encryption.

Client-Side Code that is executed on the users device.

Command Line Interface (CLI) A software that requires user interaction through the use of textual commands.

Consensus Algorithm The algorithm that dictates how it is decided which blocks are appended to a blockchain.

Count-as-recorded Voting property that enables the voter to verify that their vote has been count as it was cast.

Distributed key generation (DKG) A process in which multiple stakeholders generate a combined key from individual key shares.

Efficiency A UI property measuring how quickly a user can achieve their goal through using a software interface.

Effectiveness A UI property measuring if a user can achieve their goal completely using a software.

Frontend An application that contains the UIs of a software system.

Graphical User Interface (GUI) A piece of software that allows interaction through graphical patterns such as buttons, menus, and windows.

Identity Provider (IDP) Stakeholder in a REV system that manages eligibility to vote.

JSON Data transmission format, often used for communication of data in web applications.

Learnability A UI property measuring how difficult using a software is to learn.

Proof of Authority (PoA) A consensus algorithm relying on trusted entities in a peer-to-peer network.

Recorded-as-cast Voting property that enables the voter to verify whether their ballot has been received and recorded by a REV system as it was cast.

Remote Electronic Voting (REV) The Process of holding a vote or election (partially) through the internet.

Satisfaction A UI property measuring how pleasant using a software is and how much it engages a user to unfold the software's full potential.

Sealer An participant that validates new blocks in a PoA blockchain.

Smart contract (SC) Code logic that is executed in a distributed manner by the use of BC technology.

Substrate A development kit for custom BCs.

User Interface (UI) A piece of software that is intended to be interacted with by the user.

Verification The process of verifying whether a ballot has been processed correctly.

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

List of Figures

2.1	Color scales, (a) offering no perception of relation between the steps, (b) a perceivable linear scale	11
3.1	Provotum Phases, after [32]	14
3.2	Bootstrapping sequence diagram	14
3.3	Voting sequence diagram	15
3.4	Finalization sequence diagram	15
3.5	Provotum 3.0 Phases, after [26]	16
4.1	Provotum 3.0 Architecture	21
4.2	Provotum 3.0 User Interaction Flow	23
4.3	New User Interaction Flow	23
4.4	Provotum 3.0 Architecture updated	24
4.5	Provotum 3.0 Technologies	25
4.6	Password field with obscured letters, the code may easily be misspelled without detection	28
5.1	Redux Architecture Overview	33
5.2	Voting Authority Application Store Design	33
5.3	Voting Authority Application folder structure	34
5.4	Voter Application Store Design	37
6.1	Starting the Provotum BC through the UI.	43
6.2	Filtering by phase of a vote	44

6.3	Creation of a new vote	44
6.4	Opening a vote. (a) shows the detail screen embedded in the dashboard and (b) zoomed in on the details.	45
6.5	Stopping and finishing a vote	46
6.6	Inspecting vote results	46
6.7	Choosing which modules should be enabled	47
6.8	Multiple modules arranged in a custom order	47
6.9	Registering the user's wallet (a) and the message showing a successfull registration (b).	48
6.10	Vote list display with filtering and searching	49
6.11	(a) Selecting an answer for a vote, (b) an additional prompt before ballot submission, and (c) a loading screen while the ballot is processed	50
6.12	Confirmation that the ballot is recorded (a) and new indication in the vote list that a ballot for this vote has been cast (b)	50
6.13	A vote that has been closed with it's topics and their results (a), and a pie chart with the results for a selected topic (b)	51

List of Tables

6.1	Heuristic evaluation keypoints for VA frontend	40
6.2	Heuristic evaluation keypoints for voter application	42

Listings

5.1	Example UI Module	34
5.2	Module Registration	35
5.3	Function component using a React Hook	36
5.4	Enabling additional functionality	38

Appendix A

Installation Guidelines

All of the source code, accompanied by instructions for installation and development environment setup can be found at <https://github.com/provotum/provotum-infrastructure-claudio>. The README file of the repository also contains information about hardware and software used throughout development.

Appendix B

Contents of the CD

In addition to this report in PDF format, an archive containing the following items is submitted:

- The source of the report in Latex format.
- All images and graphs shown in the report.
- The source code of all applications implemented for this prototype.
- The raw files for the sequence diagrams.