



**BRNO UNIVERSITY OF TECHNOLOGY**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

**FACULTY OF INFORMATION TECHNOLOGY**

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

**DEPARTMENT OF INTELLIGENT SYSTEMS**

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

## **MOBILE APPLICATION FOR DECENTRALIZED ELECTIONS**

MOBILNÍ APLIKACE PRO DECENTRALIZOVANÉ VOLBY

**BACHELOR'S THESIS**

BAKALÁŘSKÁ PRÁCE

**AUTHOR**

AUTOR PRÁCE

**VLADISLAV PASTUSHENKO**

**SUPERVISOR**

VEDOUCÍ PRÁCE

**Ing. MAREK TAMAŠKOVIČ,**

**BRNO 2024**

## Abstract

This paper describes the development of a decentralized mobile application for the Android operating system, using the 1-out-of-k Blockchain-Based Boardroom Voting protocol. The paper describes the motivation for writing the work, the general concepts associated with the development of decentralized applications, describes the development plan, describes the selected technology, implementation and testing of application, peculiarities when working with Flutter when interacting with Solidity smart contracts.

## Abstrakt

Tato práce popisuje vývoj decentralizované mobilní aplikace pro operační systém Android, která využívá protokol hlasování v zasedací místnosti založený na blockchainu. Článek popisuje motivaci k napsání práce, obecné pojmy spojené s vývojem decentralizovaných aplikací, popisuje plán vývoje, popisuje zvolené technologie, implementace a testování aplikace, zvláštnosti při práci s Flutterem při interakci s chytrými kontrakty Solidity.

## Keywords

Blockchain, Decentralized application (dApp), Mobile application, Elections, Voting, Security, Transparency, Accessibility, Blockchain technology, Smart contracts, Ethereum, DApp development, Web3, Mobile app development, Decentralized systems, Peer-to-peer networking, P2P, Cryptography, Flutter, Metamask, Truffle, WalletConnect

## Klíčová slova

Decentralizované aplikace, volby, hlasování, bezpečnost, chytré smlouvy, peer-to-peer síť, vývoj mobilních aplikací

## Reference

PASTUSHENKO, Vladislav. *Mobile application for decentralized elections*. Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Marek Tamaškovič,

# Mobile application for decentralized elections

## Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Tamaškoviče Marka. Další informace mi poskytli Ivan Homoliak a Stančíková Ivana. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....  
Vladislav Pastushenko  
May 9, 2024

## Acknowledgements

I would like to thank Stančíková Ivana and Ivan Homoliak for providing materials and support in writing this work.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Theory</b>	<b>6</b>
2.1	Blockchain . . . . .	6
2.2	Ethereum . . . . .	8
2.3	Decentralized Applications . . . . .	10
2.4	Web3 . . . . .	10
2.5	Scalable Self-Tallying Blockchain-Based Voting . . . . .	12
<b>3</b>	<b>Concept and Requirements</b>	<b>17</b>
3.1	Basic Criteria . . . . .	17
3.2	Functional Requirements . . . . .	17
3.3	Non-Functional Requirements . . . . .	18
<b>4</b>	<b>Solution draft</b>	<b>20</b>
4.1	Selected Technologies . . . . .	20
4.2	Concept . . . . .	23
4.3	Pages . . . . .	24
<b>5</b>	<b>Implementation</b>	<b>26</b>
5.1	Architecture . . . . .	26
5.2	Client . . . . .	27
5.3	Authorization . . . . .	28
5.4	Home Page . . . . .	29
5.5	Election List . . . . .	31
5.6	Setup Stage . . . . .	31
5.7	Sign-Up Stage . . . . .	32
5.8	Pre-Voting . . . . .	32
5.9	Voting Stage . . . . .	33
5.10	Fault Repair . . . . .	35
5.11	Tally . . . . .	37
5.12	Important Aspects of Working with Smart Contracts from Dart application	38
<b>6</b>	<b>Testing</b>	<b>40</b>
6.1	Testing Process . . . . .	40
6.2	Poor Compatibility with Android 11 and Older Devices . . . . .	40
6.3	Unstable Operation of Metamask . . . . .	41
6.4	Lack of Functionality on the Home Page . . . . .	41

6.5	Unsorted List of Available Elections . . . . .	41
6.6	Visualization of Voting Results . . . . .	42
6.7	General Feedback . . . . .	42
<b>7</b>	<b>Changes Based on Testing</b>	<b>43</b>
7.1	Changed Header for ElectionDetailed Page . . . . .	43
7.2	Added Non-Voted Voters Count and Percentage of Participation in Elections	43
7.3	Added Doughnut Chart . . . . .	43
<b>8</b>	<b>Conclusion</b>	<b>45</b>
	<b>Bibliography</b>	<b>46</b>
<b>A</b>	<b>Contents of the attached DVD</b>	<b>48</b>

# List of Figures

2.1	Chain of states [20] . . . . .	7
2.2	Ethereum virtual machine [20] . . . . .	9
2.3	Voting phases of the SBBB-voting protocol [18] . . . . .	14
3.1	Use case diagram . . . . .	19
4.1	Authorization and Profile page wireframes . . . . .	25
4.2	Available Elections and Election pages wireframes . . . . .	25
5.1	Client-server communication . . . . .	27
5.2	Screenshots of authorization . . . . .	29
5.3	Activity diagram of user sign-in system . . . . .	30
5.4	Screenshots of <code>VotingDetailed</code> page in <code>VOTING</code> stage . . . . .	34
5.5	Screenshots of <code>VotingDetailed</code> page in <code>FAULT_REPAIR</code> stage . . . . .	36
7.1	Screenshots of <code>VotingDetailed</code> page in <code>TALLY</code> stage before and after changes	44

# Chapter 1

## Introduction

Elections are an important form of people's participation in public and political life. They are an indispensable public institution of the functioning of the political system, often being the most important criterion of legitimacy. However, traditional forms of elections are burdened with a number of serious drawbacks associated with their centralization.

The use of blockchain and decentralization in elections can provide a number of benefits, including:

- **Increased security:** By leveraging blockchain technology, it is possible to create a secure, tamper-evident record of votes that is resistant to fraud and manipulation. This can help to increase the integrity and trustworthiness of the electoral process.
- **Improved transparency:** Decentralized systems, such as those based on blockchain technology, can provide increased transparency as all participants in the system have access to the same information and can verify the validity of the data. This can help to build confidence in the electoral process and ensure that all votes are counted accurately.
- **Enhanced accessibility:** The use of blockchain and decentralization can make it easier for voters to participate in elections, particularly in cases where traditional voting methods may be impractical or difficult to access. For example, blockchain-based voting systems can enable individuals to cast their votes remotely, which can be particularly useful in cases where voters may be physically unable to travel to polling stations or may be located in areas with limited polling infrastructure.
- **Reduced costs:** The use of decentralized systems can help to reduce the costs associated with conducting elections, as they can eliminate the need for intermediaries to verify and validate votes, and can also reduce the need for paper-based voting systems, which can be expensive to print and manage.

Decentralized applications are a convenient way to interact with and use blockchain technology. They are Web, mobile, or desktop applications running without a server part, implemented in a decentralized computer system - for example, on a blockchain. Thus, the creation of a decentralized application with the ability to conduct elections will provide all the benefits of blockchain mentioned above: transparency, reliability, and immutability of data for the widest possible range of people, in a convenient and familiar form for the modern individual.

The goal of this work is to plan and implement an Android-based decentralized application that will implement a 1-out-of-k voting protocol. The final app will need to interact with an Ethereum blockchain-based smart contract implementing the 1-out-of-k voting protocol and provide an interaction interface for all phases of the protocol, including registration and voting. An important aspect of the project is also to not only provide correctly working functionality and logic but also to implement a user-friendly interface, test the application, perform a usability study, and incorporate its feedback into the implemented application.



# Chapter 2

## Theory

### 2.1 Blockchain

#### Basic Concept

Blockchain is a decentralized ledger where information is exchanged through secure channels. This *peer-to-peer* (**P2P**) technology organizes information into **blocks**, which are linked together using cryptographic hash functions that serve as unique identifiers [17].

Blockchain ensures data integrity by providing a unified and reliable source of information, preventing duplication, and enhancing data security.

The blockchain system eliminates the possibility of fraud and data manipulation, as any changes can only be made with the approval of the majority of participants. Data on the blockchain is **immutable**; it can be transferred and new data can be created, but existing data cannot be altered or deleted.

Blockchain was first practically implemented in 2009 as the underlying mechanism of the Bitcoin cryptocurrency [2].

#### Chain of Blocks

The process of saving data to the blockchain, during which information is transferred between network participants, is called a **transaction**. When a user (or smart contract) initiates a transaction, such as transferring a certain amount of cryptocurrency to another user, this transaction is broadcast across the network. For a transaction to be confirmed, each network node verifies its authenticity by analyzing digital signatures and other associated information.

When any transaction occurs on the blockchain network, information about it and other transactions is recorded in a new data **block**.

Hence the concept of *blockchain*. It is literally a chain of data blocks connected to each other using cryptographic methods. Each block is linked to the ones before and after it. Transactions iteratively record the new state of the network – this means that changes do not occur at the moment of the transaction, but at the moment the nearest data block is recorded after the transaction. Even if a transaction was made immediately after the mining of the previous block, it is necessary to wait for the mining of a new block for that transaction and all transactions made after the mining of the previous block to be confirmed. Once a new block is recorded, it cannot be deleted or altered, creating one of the main advantages of blockchain - **immutability**.

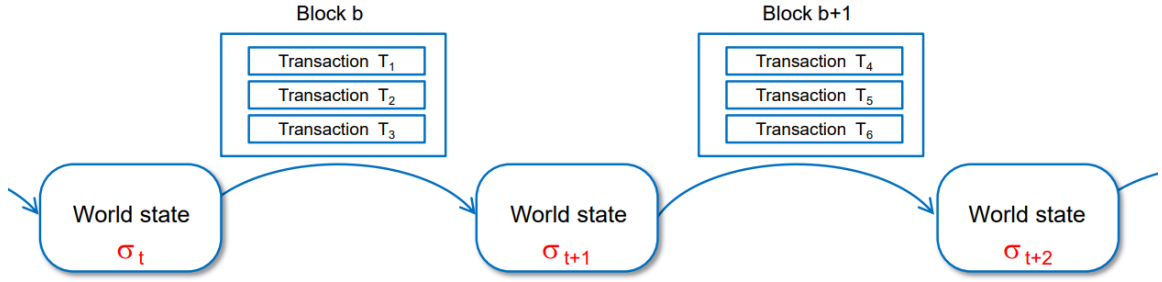


Figure 2.1: Chain of states [20]

Thus, a blockchain usually does not have information about its current state, but it has an initial state and a continuous chain of data with the histories of all transactions since its inception, which can be used to calculate the current state of the network.

Because of this, network participants usually have a full local copy of the entire blockchain.

## Mining

To record and confirm a new data block, it is necessary to verify the correctness of transactions, a role performed by *miners*. The main task of mining is to achieve **consensus** on which transactions to consider valid, preventing network participants from using already spent cryptocurrency coins. Mining processes vary across blockchains but typically involve collecting information about recent transactions, forming a new data block for the blockchain, and then solving complex mathematical problems that validate these transactions. Usually, this means finding a hash value that fits the transactions in the new block and allows for the derivation of a *secret key*. The hash that miners try to find is generated based on the hash of the previous block, a random number (*nonce*), and the sum of transaction hashes since the last transaction. The mining **difficulty** for each blockchain is calculated differently, but for example, in Bitcoin mining, the difficulty of finding the hash usually increases and is a variable quantity [5].

The calculation of such operations for key derivation is a non-trivial task requiring significant computational power. Typically, powerful computers specifically assembled for optimized mining tasks are used in real cryptocurrency networks. For this reason, mining in busy and large blockchains requires powerful equipment, which in turn requires a large amount of electricity, a cost of blockchains and cryptocurrencies. Although the difficulty of mining is not constant, on average it is growing every year [1].

However, for successfully finding a hash to record a block, the miner who mined that block receives a reward - a certain amount of newly issued cryptocurrency. Thus, miners have a financial incentive to invest in mining.

Cryptocurrencies like Bitcoin have a halving system. Bitcoin halving occurs every 210,000 blocks, roughly every four years. Halving cuts the reward miners receive for mining a block in half. The initial reward amount in Bitcoin was 50 BTC per found block, but by

2024, four halvings have occurred, and the current block reward is 3.125 BTC. This system exists to slow down inflation or even to ensure the deflation of the cryptocurrency.

Because all transactions are decentralizedly computed on a multitude of unrelated computers around the world, each of which has a copy of the entire blockchain, hacking and forging transactions become an improbable event. It is only possible if someone controls more than 50% of the computational power of the entire blockchain network, which is practically impossible for popular and widespread cryptocurrencies.

This gives blockchains a huge advantage, as blockchains and cryptocurrencies cannot be centrally controlled by any network participant. This ensures security, full control, and freedom over one's cryptocurrency wallet for network participants. However, it also increases the risks of erroneous transactions or involuntary transactions made due to the actions of fraudsters or created under pressure in other conditions. Since there is no party that could cancel or prevent a transaction, any actions taken in the blockchain network must be approached with utmost care and responsibility.

## 2.2 Ethereum

The implementation of the logic of this decentralized application is based on the **Ethereum blockchain** as one of the most popular platforms for creating **smart contracts**.

Ethereum is an open blockchain platform that allows users to create and use decentralized applications (**dApps**) based on smart contracts. Ethereum was proposed by programmer Vitalik Buterin in late 2013 and officially launched in July 2015. It was designed to expand the capabilities of blockchain beyond just transaction processing, as seen in Bitcoin. Ethereum is not just a cryptocurrency but an entire ecosystem that includes the integration of executable programs directly into the blockchain, the creation of new types of tokens, support for various token standards, and more. Ethereum operates on the **Ethereum Virtual Machine (EVM)**.

Once the dApp is deployed, users can interact with it and execute the rules and conditions specified in the smart contracts by sending transactions to the Ethereum blockchain. The Ethereum network will then validate and execute the transactions and update the state of the dApp accordingly.

### Ethereum Virtual Machine (EVM)

The EVM is a virtual computing environment distributed across computers in the blockchain network, responsible for executing smart contracts. It is Turing complete, meaning it can perform any computation that can be done on a computer, provided there are sufficient computing resources [6].

This is the central part of Ethereum's architecture, isolated from the rest of the network, which ensures the security of code execution. The concept of EVM was detailed in the Yellow Paper by Gavin Wood [23].

EVM serves as a platform for token operations within the Ethereum network, invoking smart contracts, altering application states, and calculating address balances.

It is a kind of decentralized computer. Code written in an Ethereum block is executed independently on all virtual machines.

Moreover, EVM is used not only in Ethereum but also in networks like Polygon, VeChain, and others. EVM's compatibility with other networks opens up opportunities for migration to other systems for scaling and testing.

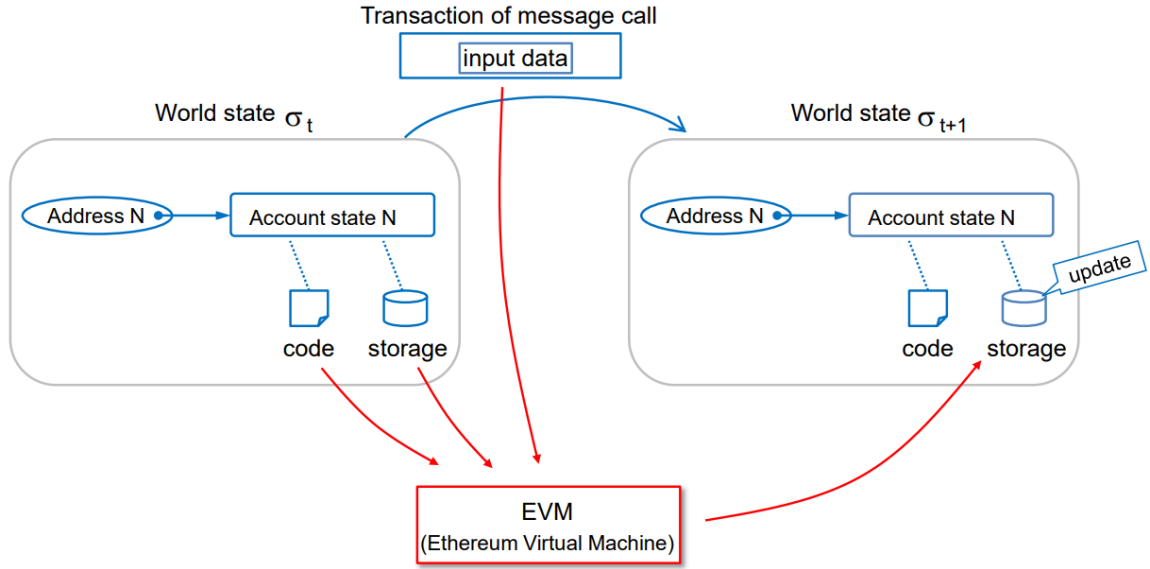


Figure 2.2: Ethereum virtual machine [20]

EVM uses a stack architecture to ensure security and determinism. Instructions are executed sequentially, with data being passed and retrieved from the stack.

## Types of Accounts

In Ethereum, there are two types of accounts:

- **Externally Owned Accounts (EOAs)** - controlled by users who possess the private keys that created the account,
- **Contract Accounts** - managed by smart contracts, whose code is stored on the blockchain and executed when this smart contract is called by a user or other smart contracts.

## ABI and Bytecode

The **Application Binary Interface (ABI)** is a standard for interaction between two binary software interfaces, used to describe interactions between smart contracts [24]. ABI is necessary to retrieve information for calling functions and obtaining data from smart contracts. **Bytecode** is the compiled code of a smart contract that can be executed by the EVM. Most smart contracts are created in **Solidity**, which is compiled using the Solidity Compiler to generate bytecode.

## Concept of Gas

1 Ether equals 1,000,000,000 **Gwei** or 1,000,000,000,000,000 **wei**. These units represent what is known as **gas**. Gas is a unit of measurement for the computational work of transactions or smart contracts on the Ethereum network. It is the currency used to pay for operations carried out on the EVM. Gas is necessary to prevent unnecessary cycles and abuse of smart contract usage. The more complex the operation, the more gas it requires.

There are priority fees for gas - the cost that must be paid to expedite the execution of transactions.

Network congestion determines the price of gas; the more congested the network, the more expensive the gas. This system was created to motivate miners when the **TPS** (transactions per second) rate is low.

## 2.3 Decentralized Applications

### Basic Terms

**Decentralized Applications (dApps)** are software applications that are built on top of decentralized networks, such as blockchain platforms. They are designed to operate in a decentralized manner, meaning that they do not rely on a central authority or server to function.

dApps are used for a wide range of purposes, including financial applications, social networks, marketplaces, and more. They can be used in various industries and sectors, including finance, healthcare, education, and government. They are built using various technologies, including blockchain, peer-to-peer networking, and cryptography.

dApps offer a number of benefits over traditional centralized applications, including increased security, transparency, and decentralization, which can help to enhance trust and make them more resistant to tampering and censorship.

### Common structure of dApps

Generally, a dApp will consist of a front-end interface that users interact with and a back-end decentralized network that powers the dApp. The front-end interface can be built using a variety of technologies, including web technologies and mobile development technologies.

The back-end of a dApp is powered by a decentralized network, such as a blockchain or decentralized file storage system. This network stores and manages the data and logic of the dApp in a decentralized manner, meaning that it is not controlled by a central authority or server.

The internal logic of a dApp is implemented using **smart contracts**. The term was coined in 1997 by Nick Szabo [19]. Smart contracts are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code. Smart contracts contain the rules and conditions that govern the operation of the dApp, and they are stored and enforced by the decentralized network that powers the dApp.

The structure and internal logic of a dApp are designed to enable the dApp to operate in a decentralized manner, without the need for a central authority or server to control its operation. dApps are fully transparent since their code is accessible to all network nodes.

## 2.4 Web3

### Basic terms

Web3 is a term that has become popular in recent years due to the broader adoption of blockchain, representing a new generation of the internet. Web3 is built on blockchain and decentralization.

The number 3 denotes the third generation, which was preceded by the first and second generations.

## Web1

Web1 (web 1.0) is the first generation of the Web, often referred to as read-only. This phase began in the 1990s. It focused on data retrieval without interaction. Internet speed and access were severely limited. It lacked visual elements and interactivity. Its main features were:

- Static HTML web pages,
- Limited user interaction,
- Content sourced not from databases, but from the server's filesystem,
- Content created and published by a limited number of people,

## Web2

Web2 (web 2.0) followed, as the internet became more widespread and web technologies were developed and refined. The main distinguishing feature of Web2 from its predecessor was interactivity. Users could not only read and retrieve content but also interact with it, create new content, and modify existing content. Web2 emerged in the early 2000s and represents the Internet as we know it today. Social networks, blogs, and video services emerged where each user could modify site content through a simple interface. With the proliferation of mobile devices and smartphones, Web2 experienced incredible growth and spread. Features of Web2 include:

- Dynamic content that users can freely create, modify, and share with others,
- Widespread and accessible internet,
- Page interaction and social network activity,
- Centralized platforms controlling large data sets and user interactions,

## Web3

Web3, in turn, is a new generation based on the decentralization of data storage and interaction. Web3 is often associated with the use of EVM for creating decentralized applications. The term „Web3“ was first proposed by Gavin Wood, including the idea of shifting control and ownership of data from centralized corporations to individual users.

Besides decentralization, Web3 is characterized by other features such as the widespread use and integration of artificial intelligence, the Internet of Things, semantic search relying on context rather than keywords, and more [21].

Despite the obvious advantages of Web3, such as greater user autonomy and control, the absence of a centralized point of failure, transparency, and the elimination of intermediaries, Web3 has several significant technical limitations and drawbacks. Currently, Web3 is difficult to scale. It is complex for the majority of ordinary users to operate, as the concept of blockchain is still not widely spread and has a relatively high entry threshold in terms of internet technology awareness. Web3 is difficult to regulate, often becoming a problem for the legitimization and legal status of decentralized technologies in many countries. Due to the freedom to use one's assets, Web3 requires a more responsible approach to vigilance and awareness when used. Utilizing Web3 requires sufficiently powerful hardware, limiting its accessibility.

## 2.5 Scalable Self-Tallying Blockchain-Based Voting

The application is based on a **1-out-of-k Scalable Self-Tallying Blockchain-Based Voting protocol**, derived from **BBB-Voting** [22]. This protocol is a modified 1-out-of-2 protocol with support for a choice of  $k$  available options, where

$$k \geq 2$$

This protocol used a fault-tolerant approach to ensure that voting is completed without restarting the protocol.

**The SBBB-voting protocol conforms to the following basic properties:**

- Privacy of the Vote
- Perfect Ballot Secrecy
- Fairness
- Verifiability
- Self-Tallying
- Dispute-Freeness
- Fault Tolerance
- Resistance to Serious Failures

These properties guarantee a high level of quality in the functioning of the application logic.

The SBBB-voting protocol includes several phases: **registration** (identity verification, key ownership verification, enrollment on Blockchain), **setup** (agreement on system parameters, submission of ephemeral public keys), **prevoting** (computation of MPC keys), **voting** (vote packing, blinding, and verification), **fault-recovery**, booth tallies, and the **final tally** phase [18].

It should be noted that not all phases of the protocol are implemented automatically by the blockchain. For example, it is expected that the Authority will notify users about the start and end of the elections, provide the address of the deployed election contract and the **FastEcMul** contract necessary for calculating decomposed scalars, and participants will create and submit data about their public keys.

### Participants

The protocol involves several different participants:

- **Voter** - Voting participants. They must be invited by the Authority. A voter must create and provide an ephemeral public key based on which an anonymous vote for an election candidate will be cast.
- **Authority** - The election administrator, who creates and deploys the election contract, invites and registers participants, changes some stages of the election, controls the division of voters into groups, and so on.

## Phases of the Protocol

The operation of the protocol is divided into several mandatory and optional phases, each of which may require certain actions from election participants.

### Registration

This is a preparatory phase that largely remains outside the protocol implementation. In this phase, voting participants create their wallets and provide information about their public keys to the Authority. Voters must also verify the authenticity of their wallet ownership, for example, using a personal signature. This phase also includes verifying the identity of voters, which is typically performed by third-party services, and linking the voter's identity with their wallet address.

Next, the Authority must determine settings, such as identifying election candidates, and compile and deploy all necessary smart contracts to the network. The Authority must provide addresses for at least two deployed smart contracts - **FastEcMul** and **MainVotingC**, as they are necessary for voters to participate in the elections.

### Setup

The Authority must enroll the public keys of all voters, after which it should invoke functions to divide voters into groups and deploy **Voting Booths** for each group. Dividing voters into groups is necessary to optimize the protocol operation and to administer participants more conveniently.

After deploying the Voting Booth smart contracts for each group of voters, the voters must generate **ephemeral keys** and send information about them to the contract. The creation of ephemeral keys is necessary to preserve the anonymity of voters. An ephemeral key should only be used within one election. All election participants have access to all confirmed ephemeral keys of their group through interaction with the Voting Booth smart contract, which will be needed in further steps. Once all voters submit their keys, the Voting Booth automatically enters the next phase.

### Pre-Voting

This phase requires no action from voters and depends only on the Authority. It includes the computation of **Multiparty Computation (MPC)** keys for synchronizing keys among all voters and consequently achieving a self-tallying property. To carry out this phase, the Authority must perform several sequential calls of different smart contract functions - **buildRightMarkers4MPC** for calculating right-side values for MPC key computation, **modInvCache4MPCBatched** for precomputation of modular inverses for MPC key computation, and ultimately the calculation of MPC keys using the obtained parameters and calling the **computeMPCKeys** function. MPC keys for each participant can be obtained using the following formula:

$$h = g^{y_i} = \frac{\prod_{j=1}^{i-1} g^{x_j}}{\prod_{j=i+1}^n g^{x_j}}$$

After successfully completing the function, the Voting Booth automatically moves to the next phase.



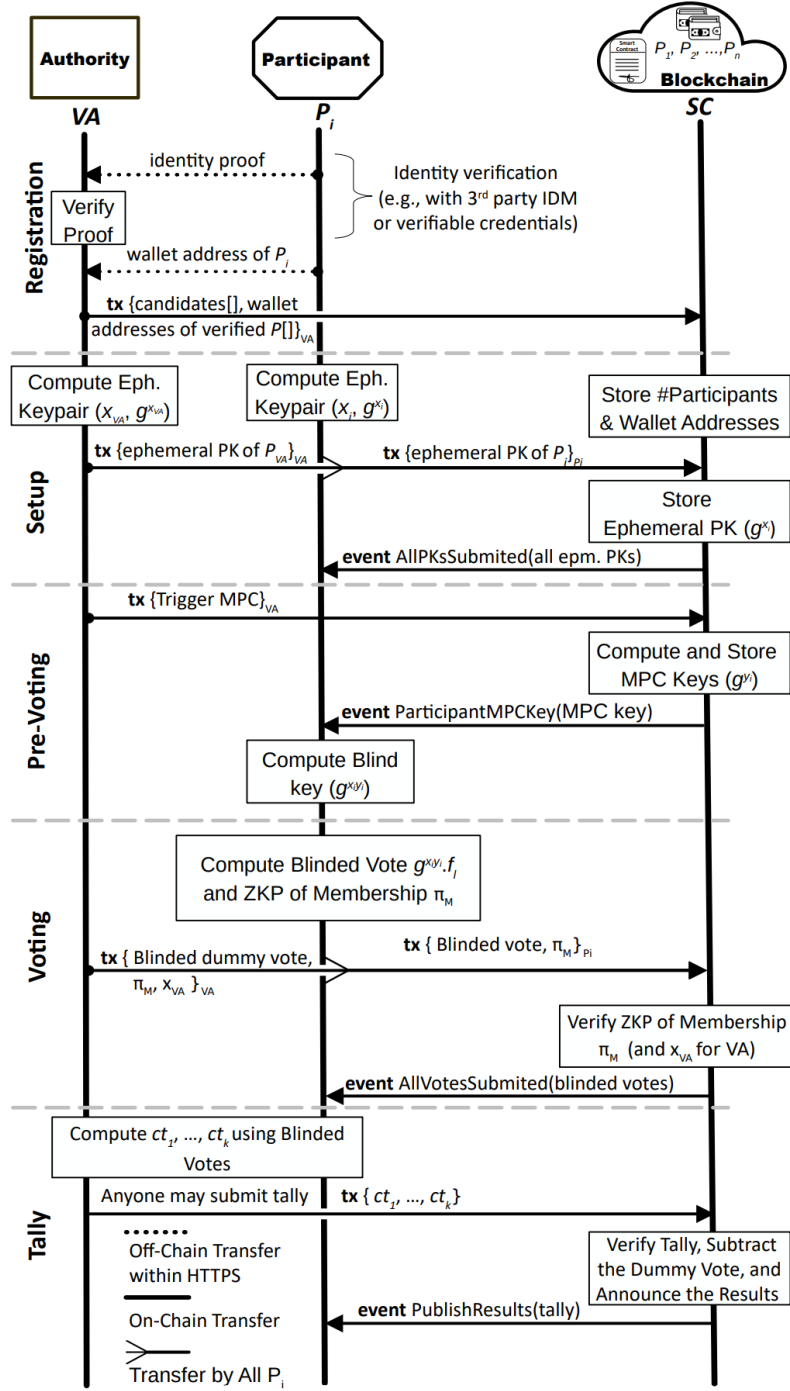


Figure 2.3: Voting phases of the SBBB-voting protocol [18]

## Voting

The next phase is the actual **voting**.

In this phase, each voter blinds and submits their vote to the contract. The voting participant's client must formulate and send the vote using the following formula:

$$B_i = \begin{cases} g^{x_i} y_i f_1 & \text{if } P_i \text{ votes for choice 1,} \\ g^{x_i} y_i f_2 & \text{if } P_i \text{ votes for choice 2,} \\ \vdots & \vdots \\ g^{x_i} y_i f_k & \text{if } P_i \text{ votes for choice } k. \end{cases}$$

Along with the encrypted vote, participants must also send what are known as **zero-knowledge proofs (ZKPs)**. ZKPs are a cryptographic method used to prove knowledge about a piece of data (in our case, data about the private key) without revealing the data itself.

Next, the contract analyzes the received data, and if they are correct, it stores the received vote from the participant.

## Tally

When the number of received votes corresponds to the number of participants, the election moves into the **Tally** phase. In this phase, no interaction with the contract from the voters is required again. All necessary actions are performed by the Authority. They must call the `computeBlindedVotesSum` function to compute the sum of all blinded votes. Then, using the data obtained thanks to `decomposeScalar`, call `modInvCache4Tally`, which initiates the precomputing of modular inverses for tally computation. The final step is calling `getFinalTally`, which retrieves the results of the count.

During the calculations, it is necessary to provide the correct final results of the election in the Voting Booth, which can be obtained exclusively by tallying votes, i.e., iterating through all possible outcomes. The number of tally attempts can be determined by the formula:

$$\binom{n+k-1}{k}$$

Final counts of votes  $c_i$ , where  $i \in \{1, \dots, k\}$  is calculated as follows:

$$\sum_{i=1}^n V_i = \sum_{i=1}^n x_{ij} G + F = c_1 F_1 + \dots + c_k F_k$$

## Fault Recovery

In case any of the registered voters fail to confirm their vote, counting the votes becomes impossible. For this, the protocol includes an optional **Fault Recovery** phase. To enter this phase, the Authority must call the function `changeStageToFaultRepair`. After transitioning to this phase, each voter must recover their vote, and to do this, each voter needs to generate corrective keys for each non-voting participant. Subsequently, each voter calls the function `repairBlindedVote`, which requires modular inverses for values compared in proof validation and elements of zero-knowledge proof for each key.

After the vote is recovered by each of the active voting participants, the voting moves to the Tally phase, as described in the previous subsection.

## Chapter 3

# Concept and Requirements

### 3.1 Basic Criteria

The application must ensure that all steps of the **SBBB-voting protocol** work for the voter. Some steps of this protocol do not require direct interaction from the user, while others can only be performed exclusively by the user.

This work does not involve handling the protocol from the Authority side, as it was initially intended to be implemented for voters only.

The application interface must meet certain quality standards. The UI of the mobile app should be easy to use and intuitive, with clear and concise instructions and a logical flow. The UI should be visually appealing and visually consistent, with a clear and cohesive design style. It should be responsive to different screen sizes and resolutions and should work well on different devices and platforms. Also, it should be acceptably fast and smooth, with minimal resource usage.

The app should have convenient, logical navigation between pages. Pages and their functionality should be logically separated from each other. When developing, frequently used elements should be separated into reusable components, which will be the basis for the stylistic homogeneity of the application.

### 3.2 Functional Requirements

**Functional requirements** define how a system should behave. They specify what the system must do to meet user needs. In this work, the functional requirements must minimally comply with all phases of the SBBB-voting protocol and ideally expand its functionality for a pleasant user experience. All actions must work reliably, as this is the main acceptance criterion of the work.

The list below describes the main actions that should be available to the user:

- User authentication and log out;
- Retrieval and display of a list of available elections;
- Retrieval of information about elections and their status Generation of ephemeral keys;
- Generation of ephemeral keys;
- Sending ephemeral keys to the smart contract, confirming participation in elections;

- Retrieval and display of all election candidates;
- Generation and sending of encrypted votes and zero-knowledge proofs;
- Generation of data for vote repairing, sending a request to repair the vote;
- Retrieval and display of election results;

### 3.3 Non-Functional Requirements

**Non-functional requirements** describe the system's constraints. A system can meet minimal functional requirements without fulfilling non-functional requirements; however, adhering to these makes the user experience more pleasant, so their fulfilment is very important. In this work, there are several important non-functional criteria, such as support for the Android OS and interaction with the blockchain and smart contract of the SBBB-voting protocol. Some other non-functional requirements are highly desirable, as they represent the minimum quality standards of a modern user application.

Here is the main list of non-functional requirements:

- Android application;
- Reactive components;
- Client-side communication with the Ethereum blockchain for participating in elections and retrieving data about them;
- Centralized server for managing metadata about elections;
- Consistent design;
- Correct error display;
- Correct display of loading states;

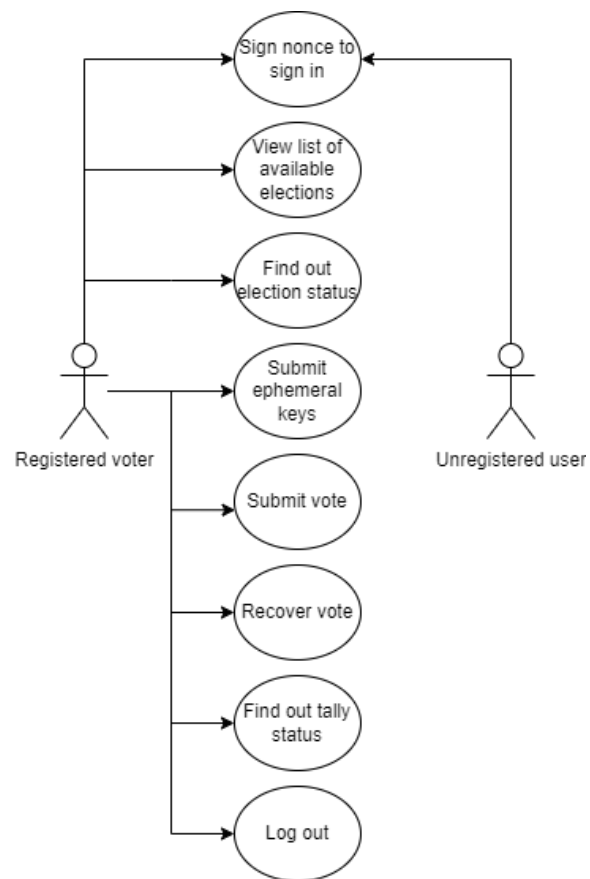


Figure 3.1: Use case diagram

# Chapter 4

## Solution draft

### 4.1 Selected Technologies

#### Flutter [9]

In modern mobile app development, various frameworks and libraries are utilized. These provide a software environment that lays the foundation for app development, offering a set of tools, usage templates, data structures, and more, which significantly facilitate and speed up the creation of apps and help organize the source code.

Frameworks can be divided into two subtypes: **native** and **cross-platform**. **Native frameworks** utilize the official SDK of the operating system for which development is undertaken. Using native frameworks in development offers greater control over the resources, capabilities, and functionality of the platform being developed for.

However, cross-platform frameworks are gaining popularity each year. These allow development for different operating systems using the same code base. One such framework is **Flutter**.

Flutter is an open-source framework by Google for creating applications on various platforms using a single code base. **Dart** [4], a programming language also developed by Google, is used for writing code in Flutter.

In Flutter, an application consists of many independent **widgets** (interface elements), each describing a component of the user interface. Thus, the component-based approach used by Flutter helps better decompose code, which in turn simplifies scaling the application.

Flutter has a robust ecosystem and a wealth of various libraries. This is particularly useful in the context of developing a decentralized application, as the functionality necessary for working with blockchain is quite specialized and not natively provided in most development environments.

In Flutter, a key principle is the **reactivity of components**. This means that components can change and behave differently depending on the state of the application. For instance, when an event occurs, such as a user pressing a button, the application re-renders the displayed elements without reloading the entire screen.

Flutter also has a rich library of basic widgets, which saves time in developing simple designs and components with typical functional tasks.

## Truffle Suite [13]

For building Ethereum dApps, it is necessary to choose a convenient smart contract development environment.

**Truffle Suite** is a smart contract development environment developed by ConsenSys Software. It simplifies the process of compiling smart contracts by automatically handling all necessary compilation and linking of contract artifacts. It also supports the convenient and reliable deployment of smart contracts to the blockchain.

Although the implementation and development of smart contracts for the SBBB-voting protocol are not covered in this work, analyzing the data received from the blockchain on the client side using debugging will be very useful during the application development process. Truffle not only has a **CLI** version but also a **VS Code extension**, which allows for convenient actions such as step-by-step debugging using a GUI.

## Ganache [10]

**Ganache** is part of the Truffle Suite and serves as a tool for creating a local blockchain, which developers use for testing and developing decentralized applications.

Ganache allows the creation of a local simulation of the Ethereum blockchain. Developers can execute instant transactions without waiting for confirmations and paying for gas as in the real network. It also allows you to configure network settings, create the desired number of accounts within the blockchain and adjust their balances, monitor events, mined network blocks, conducted transactions, and more.

Like Truffle, Ganache not only has a **CLI** but also a desktop **GUI** application, which facilitates convenient interaction with the test environment.

## Metamask [11]

**Metamask** is a software-based cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app. Metamask supports adding custom chains, which is necessary for interaction with Truffle.

In terms of dApp development, Metamask is used for interacting with dApps using their web browser or mobile application. It is compatible with **WalletConnect SDK**, which is described in the next section.

## WalletConnect [14]

The interaction of a client application (web or mobile) with the blockchain can be organized in various ways. **Web3Modal** from WalletConnect pertains to the type of interaction associated with cryptographic wallets and the convenient integration of providers from different blockchains.

**WalletConnect** is an open protocol that connects decentralized applications with cryptocurrency wallets. It operates over a secure channel and supports a variety of wallets, simplifying user interaction with blockchain applications. WalletConnect is supported by the most popular blockchain wallets, including Metamask.

The WalletConnect SDK is available for developing decentralized applications on different platforms, including Flutter. Part of the WalletConnect SDK is **Web3Modal SDK**. *„It provides a simple and intuitive interface for requesting actions such as signing trans-*



*actions and interacting with smart contracts on the blockchain“ [16]. When invoking functions, Web3Modal creates a pop-up window that offers a choice of cryptocurrency wallets to connect to. For mobile versions, when a user selects a wallet, they are redirected to the application of the chosen wallet. Initially, the user needs to connect it to the calling application. Once connected, a session object is created, which can be used to obtain the address of the connected wallet and make calls for reading or writing on the blockchain, known as **Remote Procedure Calls (RPC)**. If the initiated call requires interaction with the user’s private key, such as sending cryptocurrency or signing a message (using the `personal_sign` method), the user must confirm this action in the wallet application. This ensures the security of use.*

Any errors that occur during interactions with the blockchain can also be retrieved using an instance of Web3Modal. These can be handled by the client to ensure correct behaviour and to reflect the error in the client’s user interface.

Thus, **WalletConnect Web3Modal** acts as a link between the user application and the blockchain through the cryptocurrency wallet application.

Additionally, one of the advantages of WalletConnect Web3Modal is its compatibility with **iOS**, which opens up the potential for expansion onto another platform.

To use Web3Modal, it is necessary to obtain a project ID from the official WalletConnect website. This functionality is free and open for use according to their **Terms of Service** [15].

## Centralized Server

The backend is the server side of an application. The client usually interacts with the backend through an **API**, in which the **HyperText Transfer Protocol (HTTP)** plays a key role, serving as the main method of data transfer between the client and server.

When a user performs an action in a mobile application or on a website, the client application (frontend) sends a request to the server (backend) via HTTP. These requests can be of various types:

- **GET** to retrieve data,
- **POST** to send new data,
- **PUT** to update existing data,
- **DELETE** to remove data.

Next, the server processes the request, for example, by sending a request to retrieve or modify data from the database, and then sends the formulated response back to the client.

Compared to decentralized logic, the classic approach of using a centralized backend has a number of obvious advantages.

The development of centralized systems is generally easier. For the development of a classic backend, there are a large number of tools and different technologies available. Data storage using a centralized database is generally much cheaper. This has a significant impact on the choice of data to be stored on a blockchain because the high cost forces developers to minimize the amount of stored data as much as possible and only store on a blockchain what really needs the benefits of storage using decentralization technologies.

Also, one of blockchain’s major advantages—**immutability**—can be a serious disadvantage in agile development approaches. The resulting need to change even small pieces

of application logic after a perfect release can cause great difficulties when using purely decentralized technologies.

The SBBB-voting protocol itself does not provide all the necessary data to implement an acceptable user application. For example, in order to create and display a smart contract entity in the user interface of an application, it is necessary to first obtain its address. However, the blockchain cannot automatically obtain information about the deployed election contracts in which the user will have to participate. Blockchain also does not allow setting and transferring metadata that is insignificant for functionality but useful for displaying the user interface. For example, the blockchain does not transmit user-friendly labels for displaying a list of smart contracts. With a centralized server, it's possible to associate a smart contract entity with its address, assign a name to it, and link it to users for whom it is necessary to display this smart contract. A centralized backend is well-equipped to handle such tasks.

However, careful planning and implementation of the server and database are required to ensure that the principles of the SBBB-voting protocol are not violated. The server should never receive private data, such as private keys, associations between a vote and the voting user, and other information that could compromise the anonymity of the process.

Decentralization is often not required at all and will only be a complicating factor in development without any objective benefit. Therefore, it was decided to use a combined development approach. This means that some non-key data and logic will be implemented using classic centralized tools, but crucial aspects of the system will be decentralized.

## Flask and MySQL [7] [12]

In the implementation of this project, complex server-side application logic is not required. For this reason, the minimalist framework Flask was chosen. Flask is a lightweight and minimalist framework for the Python programming language. It is simple yet extendable. Flask has a large number of dependencies and does not have a heavy structure, which allows starting with very little and adding only the necessary components. Flask is convenient for creating RESTful APIs due to its built-in routing and request-handling capabilities.

As a solution for the database, MySQL was chosen. It is an open and accessible system, supported and developed by Oracle. MySQL is easy to install and use, while being free and sufficiently reliable, making it ideal for the requirements of this project.

## 4.2 Concept

### Basic Criteria

The application must ensure that all steps of the **SBBB-voting protocol** work for the voter. Some steps of this protocol do not require direct interaction from the user, while others can be performed exclusively by the user.

This work does not involve handling the protocol from the Authority side, as it was initially intended to be implemented for voters only.

The application interface must meet certain quality standards. The UI of the mobile app should be easy to use and intuitive, with clear and concise instructions and a logical flow. The UI should be visually appealing and visually consistent, with a clear and cohesive design style. It should be responsive to different screen sizes and resolutions and should

work well on different devices and platforms. Also, it should be acceptably fast and smooth, with minimal resource usage.

The app should have convenient, logical navigation between pages. Pages and their functionality should be logically separated from each other. When developing, frequently used elements should be separated into reusable components, which will be the basis for the stylistic homogeneity of the application.

## 4.3 Pages

### Authorization

The authorization page is the entry point to the application for all unauthorized and unregistered users. One of the key features of the dApp is the authorization and registration process. Unlike usual applications, where the user account is identified with a centralized service account (e.g., email) and a password, the accounts of decentralized applications have a **crypto-wallet** at their core. Only an active crypto wallet is required for the initial user registration. On the UI side, the authorization page will look like a blank page with an authorization button. First, the user will connect their wallet with this button, and the connecting button will be changed to the sign-in button. When clicking on this button, the user will be required to sign a **nonce** to confirm ownership of the wallet. After successful signing, the user will be authorized (in the case of registered users) or registered and authorized (in the case of unregistered users).

### Profile Settings

For the initial version of the application being developed, only minimal functionality is required - the ability to log out. In the future, as the functionality of the application expands, it may be possible to add user settings.

### Available Elections

This page will show all the elections in which an authorized user can participate or has already participated. Each election will redirect to an election details page.

### Election

This should be a dynamically changing page. Depending on the stage of the voting and user actions, this page will display either information about the election status, election results, or a form for performing necessary actions during an active election phase (sending an ephemeral key, voting, correcting a vote).

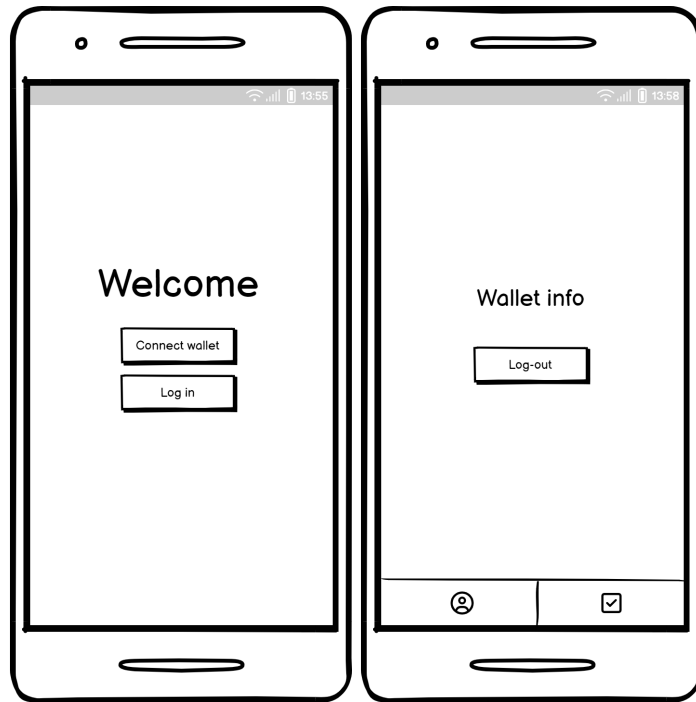


Figure 4.1: Authorization and Profile page wireframes

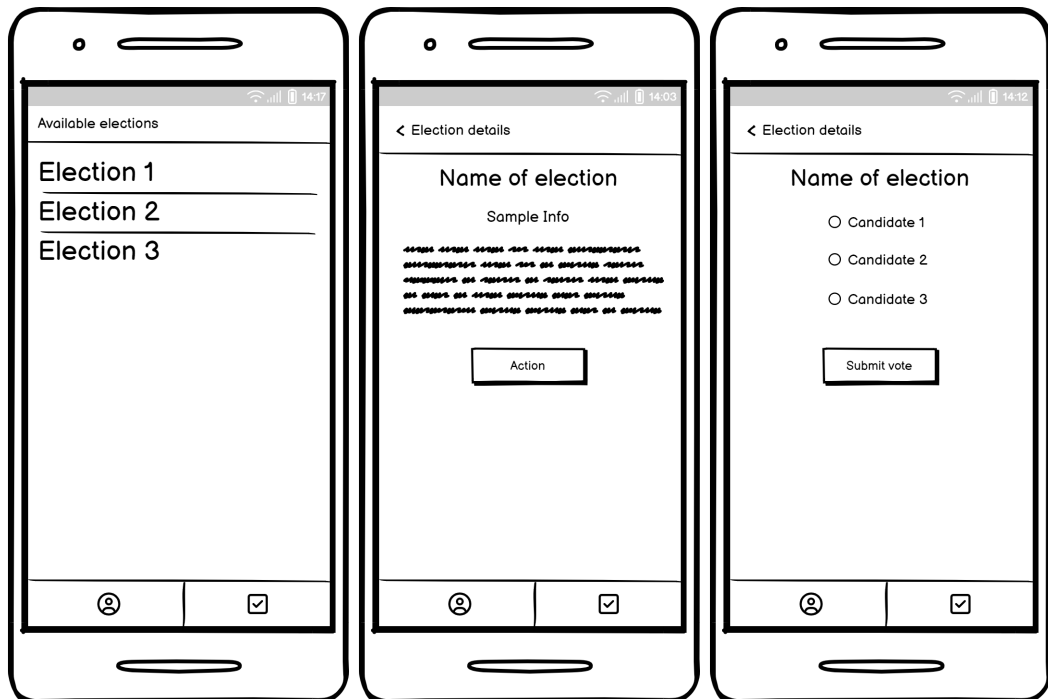


Figure 4.2: Available Elections and Election pages wireframes

## Chapter 5

# Implementation

### 5.1 Architecture

Since the application is based on two different types of servers—a centralized server using Flask connected to a centralized database, and a decentralized blockchain—different APIs and solutions are required for client communication. The final architecture of the application has been designed as a combination of **client-server architecture** for interacting with the blockchain and **Model View Controller (MVC)** architecture for communication with the centralized server.

#### Communication with the Blockchain

The file `/lib/src/utils/bc.dart` is responsible for communication and interaction with the blockchain. This file contains classes corresponding to three smart contracts with which the client interacts:

- **MainVotingContract** - The main election contract. It provides information about whether a user has already been enrolled in the elections, details about the candidates and their numbers, information about voter groups, the retrieval of the corresponding voting booth address for each group, and the results of the final vote count.
- **VotingBoothContract** - The contract for the voting booth. It is the most comprehensive contract for voter interaction, necessary for obtaining information and interacting with elections. It retrieves information about users' ephemeral keys, user IDs, the number of voters in the Voting Booth, sends requests for the calculation of modular inverses, retrieves candidate data, sends and confirms ephemeral keys, encrypted votes, requests vote recovery, and obtains information about the election stage within the booth.
- **FastEcMulContract** - An auxiliary contract for calculating decomposed scalars.

The **Web3modal** module handles connection and sending requests. Connection parameters are located in the file `/lib/src/app.dart`. Chain parameters are located in `/lib/src/screens/electionDetailed.dart`. The parameters with ABI paths are directly in the smart contract classes.

Function calls to the smart contracts are made by sending data to **Remote Procedure Call (RPC)** endpoints. RPC is a mechanism for calling procedures in another program (computer). With RPC, all necessary operations for communicating with the blockchain

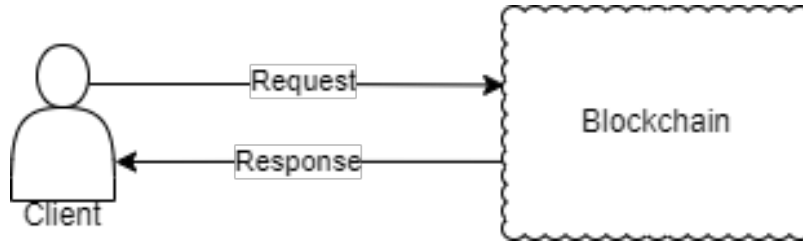


Figure 5.1: Client-server communication

are performed. Calls can be of two types: write and read. Write calls can change the state of the node, while read calls only retrieve data.

Communication with the blockchain is similar to the classic client-server architecture, with the difference that the server is decentralized. This design model, where data processing is divided between the server application and client devices, consists of two main components:

- **Client** - Operates on the user's device, either as code in a browser or as an application on a mobile device. The client sends requests to the server and displays the data received in response.
- **Server** - A remote computer (in this case, the blockchain network) that receives and processes requests from the client, changing its internal state and sending back a formulated response.

## Communication with the Centralized Server

In this project, the server is implemented in Python using the **Flask** framework. The server is based on a **REST API (Representational State Transfer Application Programming Interface)**, which means it consists of a set of endpoints accessible via standard HTTP methods. Each request from the client to the server must contain all the information necessary for its execution. The server does not store any client state. It processes each client request, handles data based on the request parameters, and returns a correct response or an error response with an error code and message.

The server's endpoints are located in the file `be/app.py`. Through these endpoints, users can obtain information about the elections they are participating in, addresses of voting contracts, and **FastEcMul**. The functionality can be expanded for use by the Authority client.

**MySQL** is responsible for the database. The class for connecting and sending requests to the database from the server is located in `be/db.py`. The settings for connecting to the database are also located there. Communication is facilitated using the **MySQL-python** package.

## 5.2 Client

### Structure

The client side consists of an Android application written on the cross-platform framework **Flutter**. All client components are contained in the folder named `flutter-application`.

The configuration file `pubspec.yaml` is located in the root of this folder. Resources used in the application, including files containing the **ABI** of contracts, are located in the `flutter-application/assets` folder.

The application implementation is located in the `flutter-application/lib` folder. The entry point to the application is the file `flutter-application/lib/main.dart`. Here, the application is launched, and the configuration saved in the device's memory is retrieved. Global **providers** are also created and passed around. In Flutter, a provider is a special package designed for convenient transmission and management of **state**. State refers to the data used to present the UI, which changes the dynamic content of the application depending on it. State management includes creating, transmitting, modifying, and deleting state.

Inside the subfolder `src`, you find the file `app.dart`. It creates the root widget of the application, configures routing within it, and also creates an instance of the class `W3MService`, obtained from the `web3modal` library, which acts as the main link between the application and the blockchain network. The configuration of `W3MService` is also contained in this file.

The `src` folder contains several subfolders:

- `/components` - reusable components of the application,
- `/screens` - screens or pages of the application,
- `/utils` - auxiliary classes and constants, such as APIs, utility functions, and more.

## 5.3 Authorization

### Connecting of Wallet

To start, in order to use the functionality of the `web3modal` library, the user needs to connect their wallet, which initiates the creation of a session. The user is greeted by a welcome screen with a button to connect the token. After clicking the button, a menu opens offering a choice of different cryptocurrency wallets for connection. During the development of the application, testing and verification were carried out using the Metamask wallet. After choosing a wallet, the user is redirected to the cryptocurrency wallet application, where they need to agree to connect. The user is then redirected back to our application, where two other buttons will now be displayed: one for signing in and another for disconnecting the wallet (in case the user has chosen the wrong wallet or wants to end the session).

### Authorization Mechanism

The logic of the authorization mechanism is as follows: The user model in the centralized database corresponds to the public address of the crypto-wallet. Also, a unique one-time identification number is generated for each user. This number is regenerated each time an authorization attempt is made on the account. When a user attempts to authorize, the Front-End sends a request to the Back-End with the public address of the user's crypto-wallet. If the user is not registered, the server creates a new non-activated user in the database based on the received wallet address. Then the authorization process for new and already registered users is the same.

The Backend responds to the received request with a one-time identification number. The response received will initiate functionality on Front-End. This causes a redirection

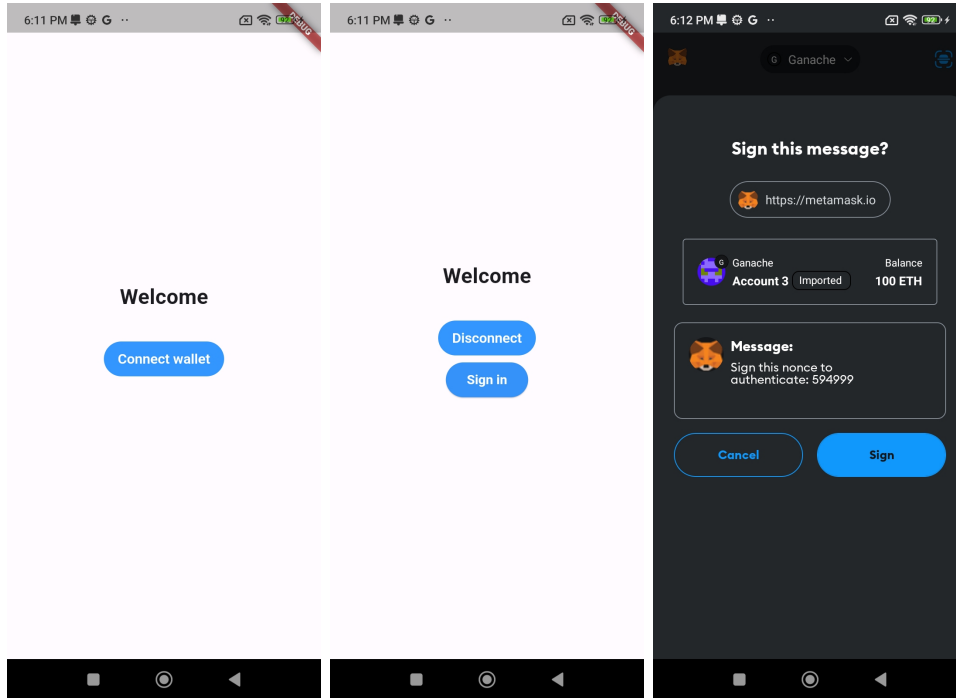


Figure 5.2: Screenshots of authorization

to the Metamask application, which waits for the user to sign or reject the signature of the one-time identification number. Once the user has signed or rejected the signature request, they are redirected back to the application. In case of a successful signature by the user, Front-End sends the next request to the server containing the signature. The Back-end then cryptographically validates the signature, in which case success means successful authorization of the user. The Backend then responds with a session token.

This token is sent through the provider to `main.dart` and is saved in long-term storage implemented using the `shared_preferences` package. Later, upon reopening the application, if the user has an authorization token stored, they automatically land on the home page, bypassing the authorization page.

If an error occurs during authentication or if the user rejects the signature of the one-time number, they are returned to the authentication page, where an error message is displayed (see Fig. 5.3).

## 5.4 Home Page

Upon entry, we are greeted by the profile page, which serves as the home page of the application. Here, you can see a button for logging out. Pressing this button removes the authentication token, updates the state, and redirects the user back to the authentication page. This page could be expanded in the future to include profile settings, such as setting personal data, configuring application settings, and more. This functionality is not anticipated in this work.

Additionally, this and subsequent pages display a navigation bar at the bottom of the screen, which allows switching between pages using the Navigator.



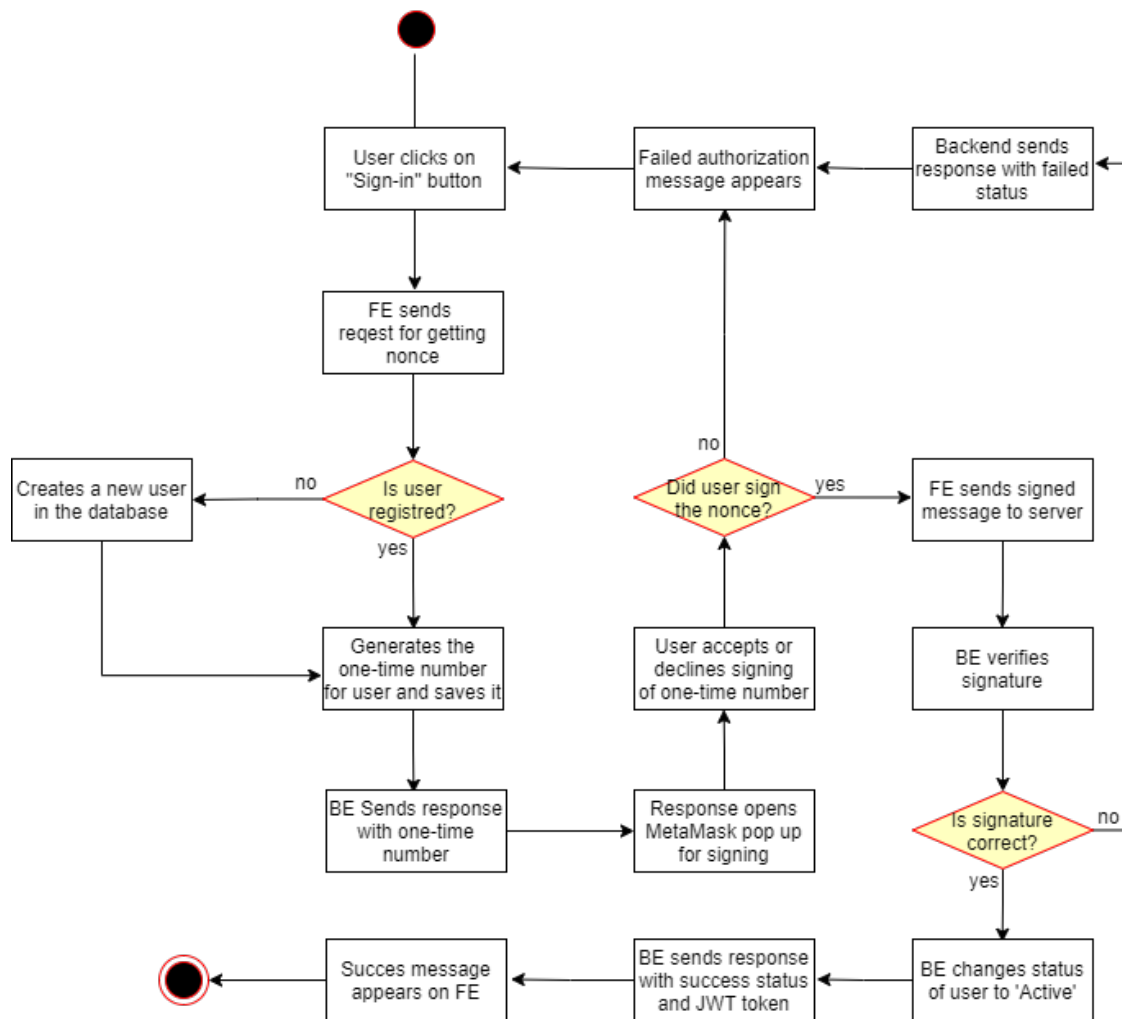


Figure 5.3: Activity diagram of user sign-in system

## 5.5 Election List

From the profile page, the user navigates to the Elections tab. Here, the user can view and enter various elections in which they have been registered. If the user has not yet been registered in any elections, an appropriate message is displayed. When this page is opened, Flutter sends a request to the centralized backend. The functionality for communicating with the backend is located in the file `flutter-application/lib/src/utls/api.dart`. In it, HTTP requests are formulated and sent using the `http` package. Using the function `getWalletData`, we obtain from the server all information about the user along with an array of `election_contracts` objects, containing data about all the election contracts the user is participating in, including details about the deployed smart contract address for the elections and `FastEcMul`, and the human-readable name of the elections set by the Authority.

All elections are displayed in a sequential list, which shows the title and contract address of each election. Each item in the list is a link that leads to the details page of the elections.

## 5.6 Setup Stage

### Description of Setup Stage

Processes for obtaining information described further occur automatically, displaying only the current state or a form for interaction with the user. If errors occur during information retrieval or interaction with the smart contract or cryptocurrency wallet, they will be displayed in the user interface.

After clicking on an item in the list of available elections, the user is redirected to the `ElectionDetailed` page. This page serves as a point of interaction and information retrieval about the elections for the user.

Initially, for the contract to appear in the voter's list, the Authority must add it to these elections in the database. After this, if the voter has been added to the database, but the Authority has not yet added the voter's address to the smart contract using the `enrollVoters` function, the voter sees a corresponding informational message. Communication at this phase, including this information, is handled through the `MainVotingContract` class.

The method `isVoterEligible` returns information on whether the user was enrolled in the elections. If the Authority has already added them, the message changes to a request to wait for the creation of Voting Booths and the start of the elections. After adding users, the Authority must divide and create groups of voters and for each group create a `VotingBooth` contract using the `splitGroups` and `deployBooths` methods, respectively. The creation of booths marks the transition to the next phase.

### Steps of Setup Stage

1. Retrieve election information from the centralized backend.
2. Create a `MainVoting` contract from the received address.
3. Make a request to `MainVoting.isVoterEligible` to determine if the voter is registered for these elections. If not registered, display a message. If successful, proceed to the next step.

## 5.7 Sign-Up Stage

### Obtaining Data for Sign-Up Stage

In this phase, voters need to create and send their ephemeral keys.

It is necessary to obtain the address of the **VotingBooth** smart contract corresponding to the user's wallet address. We can obtain the user's group by referring to the **votersGroup** function of the main contract with the wallet address as an argument, which returns the group ID. Then, using the **groupBoothAddr** function of the main contract with the obtained ID as an argument, we can obtain the desired address of the deployed **VotingBooth** smart contract.

Once we have the **VotingBooth** address, the interface displays a button for generating and confirming ephemeral keys. This occurs in the **generateAndSubmitPK** method and the **Voter** class. The **Voter** class is key in working with the protocol.

Upon initialization, it generates a unique private and public ephemeral key based on the **secp256k1** parameters for the elliptic curve used in the protocol. These parameters are widely used in cryptography, for example, they are used for Bitcoin. The **elliptic** library for Dart [3] is used for working with the elliptic curve, generating keys, and calculations.

### Submit Ephemeral Keys

Thus, when a **Voter** class object is created, ephemeral keys are generated within it, and when calling **generateAndSubmitPK**, it sends a request to the **VotingBooth** contract to call **submitVotersPK** with the parameter of the generated public ephemeral key. This redirects the user to the Metamask app, where they must confirm the action.

Subsequently, upon successful key confirmation, the user is redirected back to the app, where, if successful, they see a message about the successful key confirmation and a request to wait for the start of voting.

### Steps of Sign-Up Stage

1. Make a request to **MainVoting.votersGroup** to obtain the group ID. If unsuccessful, display a message asking to wait for the creation of **VotingBooth**.
2. Make a request to **MainVoting.groupBoothAddr** to obtain the **VotingBooth** address. If unsuccessful, display a message asking to wait for the creation of **VotingBooth**.
3. Create a **VotingBooth** contract. Display a form for generating and submitting ephemeral keys.
4. On button press, create a **Voter** class with ephemeral keys.
5. Confirm keys using **VotingBooth.submitVotersPK**. Display a message asking to wait for the start of voting.

## 5.8 Pre-Voting

This phase is conducted on the Authority side, thus it is not reflected in the app implementation. The Authority must run the **computeMPCKeys** function to calculate the MPC keys. For this, beforehand, they need to read the right markers and precomputed modular inverses for the right side of subtraction and for the result (MPC key).

## 5.9 Voting Stage

### Preparing for Voting

After the successful call of `computeMPCKeys`, the `VotingBooth` enters the `VOTING` stage.

All information about the election stage is obtained using the `getBoothStage` method of the `VotingBooth` contract.

If the elections have started, we obtain information on whether the user has already voted or not. This is implemented in the `getAndSetDidVote` method, which is based on the `getBlindedVote` method of the `VotingBooth` contract. For this, it is necessary to send the voter's user ID.

The voter's ID is obtained using the `votersPKidx` method based on the parameter containing the user's address. If the user has not voted, we display a form for choosing a candidate and a button to confirm the vote.

To get information about the candidates, it is first necessary to obtain the number of candidates by referring to the main contract through the `getCntOfCandidates` method. To get the name of a candidate, we need to iterate through all candidate indices, calling the `candidate` method in the main contract. To get the gens of the candidates, necessary for the calculations, it is also necessary to sequentially iterate and call the `candidateGens` function from the `VotingBooth` for each candidate, however, gens consist of two keys, so for each candidate, it is necessary to call `candidateGens` twice, with the candidate index and with indexes 0 or 1 for different parts of gen.

### Obtaining and Generating Parameters for Voting

After selecting the desired candidate and pressing the button, the process of forming parameters within the `submitVote` function begins. Initially, it is necessary to obtain the ephemeral keys of all users, using the `VotingBooth`'s `getCntOfVoters` method to determine the number of voters and subsequently calling `getAndSetVotersPKs` with the index of each voter. From the obtained array of keys, an array of `AffinePoint` must be formed and the `Voter` class's `computeMpcPK` method is called to calculate the MPC keys used in obtaining the blinded vote.

Based on the index of the selected candidate and the array of `AffinePoint` from the gens of candidates, the `getBlindedVote` method of the `Voter` class is called to obtain the encrypted vote and zero-knowledge proofs.

More details about the calculation of this data were described in the section voting of SBBB-voting protocol [15](#).

Next, we need to obtain decomposed scalars for the obtained proofs  $r$  and  $d$ . This is done by addressing the deployed smart contract `FastEcMul`, passing the value of the proof for the candidate, and the values  $NN$  and  $lambda$  of the elliptic curve as parameters.

The obtained parameters of invalidated scalars of the proof  $r$  and  $d$ , as well as the values of the encrypted vote and proof  $B$ , need to be sent as parameters by calling the `modInvCache4SubmitVote` method of the `VotingBooth` contract. This will return an array of invalidated modulus for vote confirmation.

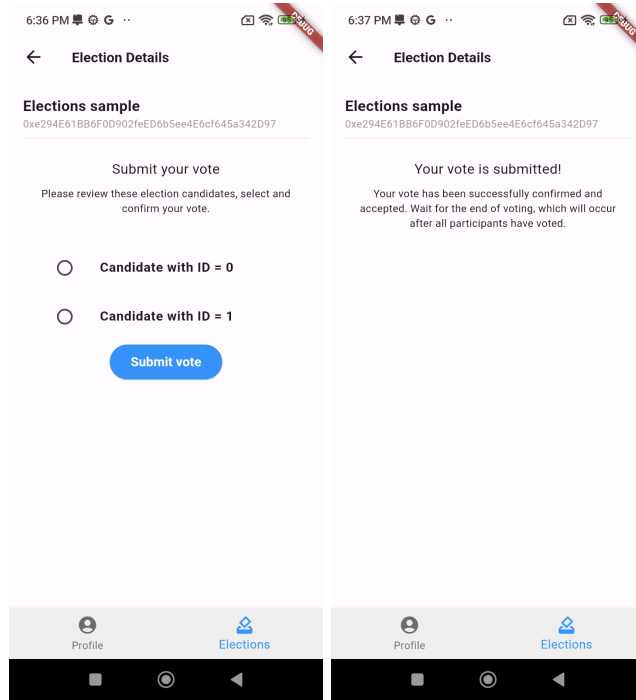


Figure 5.4: Screenshots of VotingDetailed page in VOTING stage

## Submit Vote

Then we call the `submitVote` method of the `VotingBooth` contract, sending the values of all proofs, the encrypted vote, and the array of invalidated modulus. This redirects the user to the Metamask app where they must confirm their transaction.

If everything is successful, the anonymous vote will be confirmed.

If the voter has already voted but the voting is not yet finished, a corresponding message with the status of the successful acceptance of the vote and a request for the end of voting will be displayed.

## Steps for Voting Stage

1. Make a request to `VotingBooth.getBoothStage` to obtain information about the stage of the elections. If the election stage is Voting, then proceed to the next step.
2. Retrieve the voter's ID using `VotingBooth.votersPKidx`.
3. Retrieve information about the candidates. Obtain the number of candidates by querying `MainVoting.getCntOfCandidates`. Using the obtained data, send requests to `MainVoting.candidate` and `VotingBooth.candidateGens` to get the list of candidates and the list of gens.
4. Make a request to `VotingBooth.getBlindedVote` to check if the user has already voted. If the user has not confirmed their vote yet, move to the next step. If the user has voted, proceed to step 12.
5. Display the form for selecting a candidate and confirming the vote.

6. Use `votersCnt` (the number of voters in the booth) and `VotingBooth.votersPKs` calls to retrieve the keys of all users.
7. Calculate MPC keys using the `Voter.computeMpcPK` method.
8. Obtain the encrypted vote and zero-knowledge proofs using the `Voter.getBlindedVote` method.
9. Request `FastEcMul.decomposeScalar` for the proofs  $r$  and  $d$  to obtain the decomposed scalars.
10. Request `VotingBooth.modInvCache4SubmitVote` to obtain the invalidated modulus.
11. Confirm the vote using `VotingBooth.submitVote`.
12. Display a message about the confirmed vote and request to wait for the end of voting.

## 5.10 Fault Repair

### Initialization of Fault Repair Stage

The protocol assumes that the voting booth can transition into a `FAULT_REPAIR` state. This transition occurs when the Authority calls the `changeStageToFaultRepair` method from the `VotingBooth` contract. When the client learns that the current stage is `FAULT_REPAIR`, several states may be displayed depending on the user's previous actions.

#### Unvoted User

First, it's necessary to check if the user has voted using the `getAndSetDidVote` method described earlier. If the user has not voted, they can no longer perform actions; they are shown a message that voting is currently in the fault repair stage, that they have not voted, and must wait for the end of voting.

### Obtaining and Generating Parameters for Vote Repairing

If the user had confirmed their vote, they need to restore it. More details about this process can be read in the SBBB-Voting protocol Fault repair section [15](#).

The page will display a button and information requesting to restore the vote. To restore the vote after pressing the button, the `repairVote` function is called.

First, using the `getAndSetNonVotingIndexes` function, it iterates through all user IDs based on `votersCnt` (the number of users in the booth, obtained earlier) and checks them using the `getBlindedVote` method of the `VotingBooth` contract. If there is no vote for a user's ID, it means they have not voted.

For all non-voting users, their public ephemeral keys are obtained using the `getAndSetVotersPKs` method, which retrieves X and Y keys for all users based on user indexes, iterating through a loop using `votersCnt`.

For the obtained public keys, the `computeZKproofs4FT` method of the `Voter` class is called to obtain the proofs  $m$ ,  $r$ , and hash.

Then, the `decomposeScalar` method of the `FastEcMul` contract is called for the decomposition of the proof  $r$  and hashes. Next, we get the repair key for all public keys of non-voting users, calling the `computeBlindKeyForVoter` method of the `Voter` class.

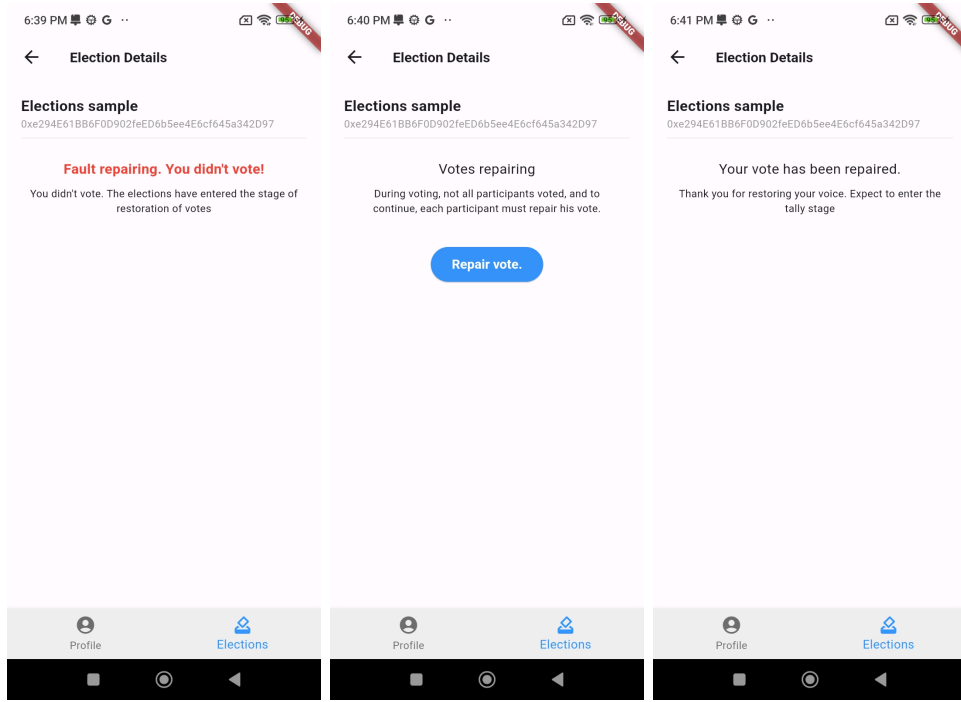


Figure 5.5: Screenshots of VotingDetailed page in FAULT\_REPAIR stage

Then, it is necessary to obtain the invalidated modulus for the repair vote. For this, the `modInvCache4repairVote` method of the `VotingBooth` contract is called, passing the indexes of non-voting participants, repair keys, user ID, proof  $r$ , and decomposed hash.

### Submit Repair Keys

Then we call the `repairBlindedVote` method of the `VotingBooth` contract, where the invalidated modulus, proofs  $r$ ,  $m1$ ,  $m2$ , blind keys of non-voting participants, indexes of non-voting participants, decomposed hash, and user index should be passed. This action redirects the user to the Metamask app to confirm the transaction.

If the transaction is successful, confirmation of this is obtained using the `submittedRepairKeys` method of the `VotingBooth` contract. If the user has already restored their vote, a message is displayed on the page thanking them for restoring their vote and asking them to wait for the tally.

### Steps for Fault Repair Stage

1. Request `VotingBooth.getBoothStage` to get information on the election stage. If the stage is Fault repair, proceed to the next step.
2. Request `VotingBooth.getBlindedVote` to check if the user has already voted. If the vote is not confirmed, proceed to the next step. If the user has voted, skip to step 4.
3. Display a message that this user did not vote during the Voting stage.

4. Check if the user has already repaired their vote with the request to `VotingBooth.submittedRepairKeys`. If the vote has not been repaired, proceed to the next step. If the user previously repaired the vote, skip to step 10.
5. Use `votersCnt` (the number of voters in the booth) and successive requests to `VotingBooth.getBlindedVote` to get the IDs of users who have not voted.
6. Use requests to `VotingBooth.votersPKs` to obtain the keys of all non-voting users.
7. Obtain the hash and proofs  $m$  and  $r$  for non-voting users using `Voter.computeZKproofs4FT`.
8. Retrieve invalidated modulus by calling `VotingBooth.modInvCache4repairVote`.
9. Using the obtained parameters, call the `VotingBooth.repairBlindedVote` method to repair the vote.
10. Display a message about the successful vote repair and ask to wait for the election results to be announced.

## 5.11 Tally

The final stage is **TALLY**. The transition to this stage can occur either from the **VOTING** stage or from the **FAULT\_REPAIR** stage. Initially, it is necessary to determine if the final tally has been completed. To do this, we refer to the main election contract and call the `getFinalTally` method. If it does not return a result, we display information indicating that the vote counting is not yet completed. However, if `getFinalTally` returns a result, then the elections can be considered concluded. In this case, a list of all election candidates and the number of votes received by each are displayed.

### Steps for Tally Stage

1. Request `VotingBooth.getBoothStage` to obtain information about the election stage. If the stage is Tally, then proceed to the next step.
2. Call `MainVoting.getFinalTally` to retrieve the voting results. If the vote counting is not yet finished, move to the next step. If the counting is finished, proceed to the final step.
3. Display a message asking to wait for the completion of the vote counts.
4. Show a message that the voting is completed and present the election results with information on the number of votes received by each candidate.

### Calculation of Votes by Authority

The vote counting is conducted by the Authority, so its implementation is not provided in the application.

The Authority must call the `computeBlindedVotesSum` function for the `VotingBooth` contract. Then, iterating through all possible voting outcomes, they must perform the following steps until the contract confirms the correctness of the voting. Initially, an array is created equal in length to the number of candidates, with values representing the



presumed number of votes received by each candidate. For each element of this array, the `decomposeScalar` method from the `FastEcMul` contract is called, passing along the values of  $NN$  and  $lambda$  of the elliptical curve.

For the obtained values, we get the invalidated modulus by calling the `modInvCache4Tally` method of the `VotingBooth` contract. Next, we call the `computeTally` function, passing the obtained invalidated modulus and decomposed scalars. In the event of a successful vote count in all voting booths, the election results become available to all participants.

## 5.12 Important Aspects of Working with Smart Contracts from Dart application

It is worth mentioning that Dart and Solidity (the language in which smart contracts are written) are structured differently. They have significant differences in data types. Also, some requirements of working with contracts in the SBBB-voting protocol cannot be directly fulfilled following the official documentation of Web3modal by WalletConnect.

### Different Data Types

Both Dart and Solidity are statically typed programming languages. Numerical data types play a significant role in working with smart contracts.

Since gas expenditure and network congestion heavily depend on the complexity of the deployed smart contract's calls, the Solidity language has been developed with a multitude of built-in tools for performance optimization. Consequently, many functions in the protocol that would logically be placed in the smart contract code, without violating the composition of components, must ultimately be implemented and computed on the client side. This is necessary to minimize the load on the network. To optimize the operation of the EVM, Solidity has multiple data types for integers. Firstly, Solidity differentiates between signed and unsigned integers. For these, there are `int` and `uint` types respectively. In Dart, there is only one type for signed integers - `int`. Secondly, in reality, `int` and `uint` in Solidity are abbreviations for `int256` and `uint256`. In Solidity, there are `uint8/int8`, `uint16/int16`, `uint32/int32`, `uint64/int64`, `uint128/int128`, `uint256/int256`, `uint512/int512` - types for numbers of different sizes. In Dart, the `int` type has a fixed number of bits - 64. Thus, `int` in Dart is not equivalent to `int` in Solidity because they have different sizes (64 bits and 256 bits respectively). Due to this, when working with numbers in the communication between Dart and Solidity, it is always necessary to remember that for correct operation, `int` in Dart should be converted to the `BigInt` type. `BigInt` is a class in Dart that represents a number but does not have a fixed size.

### Limited Functionality of Web3modal for Dart

For reading data from a smart contract (call method, read-only without writing), the official WalletConnect documentation recommends using the `requestReadContract` method of the `W3MService` class. However, the Web3modal library does not allow adding the sender's address as a parameter for this method (relevant as of version 3.1.3 of the `web3modal_flutter` library). Some protocol methods, such as `modInvCache4SubmitVote` of the `VotingBooth` contract, require the `from` parameter with the address of the calling wallet to function correctly. Because of this, it was necessary to use a lower-level method from the library - the

`call` method for the `Web3Client` class. Information about this limitation will be sent to the official WalletConnect support channels.

# Chapter 6

## Testing

To gather information, analyze the application’s performance, and identify potential errors, manual testing of all application stages was conducted by five different individuals aged 19 to 29. One participant had a full high school education, two were students, and two held bachelor’s degrees. Tests were conducted on three different Android devices.

### 6.1 Testing Process

Testers were provided with mobile phones equipped with the Metamask app already installed and configured with an address for election participation. The application was in its initial phase—the authorization page was open without any connected cryptocurrency wallet. The testers’ task was to participate in the elections and learn their outcomes. Participants were informed about how to operate the crypto wallet. However, the workings of the application itself were not detailed. They were not expected to perform the role of Authority.

Connecting to the crypto wallet and authorization posed no difficulties for the participants. Moving on to the elections, a gradual transition from one stage of the **SBBB-voting protocol** to another was simulated. Initially, participants were not enrolled in the elections, and a message asking them to wait for election registration was displayed. When they were added to the elections, this was communicated verbally, and participants moved to the next stage. Thus, the participants went through all the main stages of the elections—setup, signup, voting, tally. Four out of five participants successfully reached the stage of announcing election results without additional help. One participant encountered issues related to an outdated mobile phone, which is detailed further on.

The main findings and points from the testing are presented below.

### 6.2 Poor Compatibility with Android 11 and Older Devices

During a session, testing was conducted on a *Samsung A50* device running on **Android 11** (*One UI 3.1*). Despite claims that this version of Android should meet the minimum necessary specifications, this was not the case in reality. The main issue was working with Metamask.

The participant was unable to confirm the ephemeral keys because the transaction sending window did not appear after being redirected to the Metamask app. Subsequent tests were conducted on this device.

It turned out that the *Samsung A50* was extremely unstable during testing. After redirecting the user to the cryptocurrency wallet app, a pop-up message for confirming actions often did not appear. Problems already arose during the attempt to authenticate—messages for connecting the wallet and signing the nonce using the `personal_sign` method might not appear due to circumstances beyond control. However, sometimes it worked, and entering the app was possible.

But when the process reached the confirmation of ephemeral keys, i.e., a transaction needed to be sent for recording, the pop-up window about it simply did not appear. It was decided to test the application on another outdated device as well. Testing was conducted on a *Xiaomi Mi 9*, running **Android 11** and *MIUI 12.5*. The test results were even worse than for the *Samsung A50*. The same issues with Metamask persisted, specifically the absence of pop-up windows to confirm actions.

However, tests were also conducted for the *Xiaomi Redmi 13C*. The entire process there went without major problems, only occasionally requiring a repeat request for wallet connection/personal signature/sending a transaction, but such problems could arise from other issues such as internet speed, network congestion, etc. From this, it can be concluded that using the application on old OS and devices can be unstable and unacceptable for consumers.

In terms of responsive design, everything looked and behaved expected and acceptable.

### 6.3 Unstable Operation of Metamask

As mentioned earlier, testing revealed that requests in Metamask can sometimes be unstable. This is less pronounced in requests for wallet connection and more so in requests for transaction submission or signing. The exact reasons for this are unknown. It could be related to poor internet speed, instability of the Ganache test blockchain, unstable API performance from WalletConnect, and more. However, this was the main issue that users complained about. Should work continue on this application, investigating and improving performance in working with Metamask and the blockchain should become key areas of focus.

### 6.4 Lack of Functionality on the Home Page

Several users noted the sparse functionality and the redundancy of the profile page. They observed that a page with just a log out button looks too empty. They suggested expanding the functionality of this page or removing it entirely if unnecessary, relocating the log out logic elsewhere, such as a dropdown menu in the corner of the screen. However, this page was initially planned for future expansion. As mentioned earlier, this page could host profile settings, such as fields for changing the user's name and surname, with functionality to send data to the server.

### 6.5 Unsorted List of Available Elections

Users suggested adding sorting to the list of elections based on the current stage of the elections. For example, adding divisions by sections that would correspond to ongoing elections, completed elections, elections in which voting is underway, etc. This is an interesting idea worth considering. However, the main problem with this feature is that these data

can only be obtained from the blockchain, and the connection speed can be unstable and slow. Multiple sequential requests would have to be sent for each election, which could significantly slow down the page loading speed. One elegant solution to this problem is creating a Voting Booth entity in the database that contains information about the election status, which would be periodically updated by the Authority client depending on their actions. However, since this work did not contemplate a solution for the Authority side, performing sequential manual updates in the DB would have greatly complicated testing and development. However, this option could be considered in further work on the application.

## **6.6 Visualization of Voting Results**

Participants suggested considering an option for illustrating the voting results. For example, adding a simple pie chart that would display the number of votes each candidate received. Such an extension should not be difficult to implement, though it requires using additional packages for Flutter. Overall, the idea seems reasonable and worth implementing.

## **6.7 General Feedback**

Overall, test participants left neutral to positive reviews about the application, identifying several issues and receiving ideas for improving the application. Often participants were skeptical about the reliability of voting methods; they had questions about the security and anonymity of the protocol. Since the average awareness of blockchain principles is not very high, many people need additional information about how the application works. For this, a new page could be added, explaining the basic principles of blockchain and decentralized applications, the workings of the SBBB-voting protocol, etc. This could also be a website or part of another informational campaign. In any case, this goes beyond the scope of this work, but the thought of working in this direction should be considered. Participants also appreciated the simplicity of using the application, the intuitive and straightforward user experience, and the smooth operation.

## Chapter 7

# Changes Based on Testing

After evaluating the application, Marek Tamaškovič suggested and also expressed the need for improving the visualization of the election results and other enhancements for the same page. Similar suggestions were made by those testing the application.

As a result, based on these opinions, a decision was made to expand the visual component of the `ElectionDetailed` page for the state of completed elections.

### 7.1 Changed Header for ElectionDetailed Page

The first change concerned the header of the page. Previously, the page featured an icon for navigation to the previous page and the title 'Election Details'. This text was replaced with dynamic text corresponding to the name of the current elections, which is retrieved from the centralized backend.

### 7.2 Added Non-Voted Voters Count and Percentage of Participation in Elections

The second change was the addition of a subtitle with information about the number of participants who did not take part in the elections and the percentage of participation. To obtain the number of non-voting participants, it was necessary to calculate the difference between the total number of participants and the total number of votes cast.

To obtain the total number of participants, a request to `getCntOfEligibleVoters` must be sent to the main voting contract while obtaining data of the final votes count.

### 7.3 Added Doughnut Chart

The third innovation was the display of a doughnut chart showing the percentage of votes received for each candidate.

A doughnut chart is a type of pie chart with a hole in the centre. The circumference is divided into sections according to the percentage occupied by each section.

For working with the chart in Flutter, the `fl_chart` library [8] was selected. Other libraries, such as `syncfusion_flutter_charts`, `graphic`, `flutter_echarts`, etc., had several drawbacks. They seemed too cumbersome due to the possibility of deep detailing, or they had a paid license. The `fl_chart` library was chosen for its simplicity, high popularity, and free usage terms.

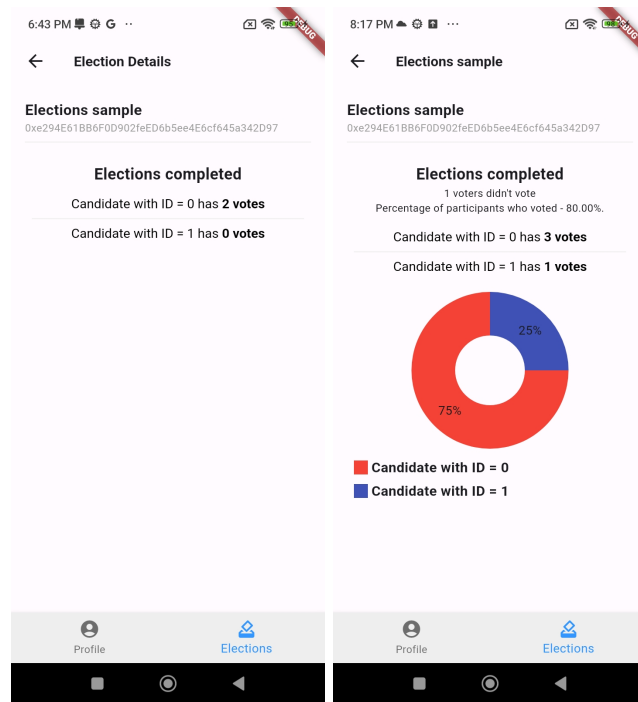


Figure 7.1: Screenshots of `VotingDetailed` page in TALLY stage before and after changes

The collected voting results are used for the chart, and percentages of votes received are calculated for displaying labels. Also created and added was the legend component. It displays colours corresponding to each candidate of an election.

## Chapter 8

# Conclusion

The aim of this work was to implement a decentralized Android application based on the SBBB-voting protocol. The project required careful attention to the theoretical foundation, proper planning, and requirements definition, selection of suitable technologies, and ultimately the implementation itself.

The resulting application was fully satisfactory. It provides stable operation for voters during elections. The application enables interaction with deployed SBBB-Voting protocol contracts at each stage of the elections. A utility-centric centralized backend was also successfully developed and tested, offering functionality that improved the user experience with the application. Meanwhile, all the advantages of the SBBB-Voting protocol were preserved, including the security and anonymity of voting.

The application features a simple and comprehensible design. It is not overloaded with details and provides the necessary widgets for interacting with the application's functionality.

Overall, the feedback from testing was positive. Important insights were gained about some performance-related issues, and ideas were identified for expanding the application's functionality and potential future work.

The chosen technologies generally met the project's requirements and lived up to expectations. The only issue encountered with Web3Modal proved to be avoidable. Information about this problem could be used to update the library. This information will be forwarded to WalletConnect. Overall, the Flutter working environment and the existing toolkit for this framework met all the necessary requirements for developing a decentralized application.

All selected technologies are compatible with iOS, so extending to a second platform should be easily achievable, although this requires additional verification.

Despite the achieved goals, there is still significant scope for improving the application's performance. Efforts can be made to enhance performance, improve design, and provide new features for users, such as push notifications, app design customization (like a dark theme), user information settings, and efforts to clarify the workings of blockchain and the SBBB-voting protocol both inside and outside the application, among other things.

Anyway, the set objectives of the work can be considered fulfilled.

In further development, the application could become a convenient tool for organizing elections facilitated by blockchain technology. This solution has many obvious advantages compared to the traditional organization of the electoral process, although it does have its nuances.



# Bibliography

- [1] *Bitcoin Difficulty Adjustments*. Available at: [https://docs.google.com/spreadsheets/d/1DQYQOLsB-pJWGu5e8CXF4vkxdYHEJD0yxQptBmC\\_030/edit#gid=0](https://docs.google.com/spreadsheets/d/1DQYQOLsB-pJWGu5e8CXF4vkxdYHEJD0yxQptBmC_030/edit#gid=0).
- [2] *Block 0. Main chain. 2009-01-03. Hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f. Block explorer*. Available at: <https://www.blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>.
- [3] *Dart-elliptic library*. Available at: <https://pub.dev/packages/elliptic>.
- [4] *Dart language*. Available at: <https://dart.dev/>.
- [5] *Difficulty*. Available at: <https://en.bitcoin.it/wiki/Difficulty>.
- [6] *ETHEREUM VIRTUAL MACHINE (EVM) - ethereum.org*. Available at: <https://ethereum.org/en/developers/docs/evm/?ref=blog.thirdweb.com>.
- [7] *Flask documentation*. Available at: <https://flask.palletsprojects.com/en/3.0.x/>.
- [8] *Fl\_chart library*. Available at: [https://pub.dev/packages/fl\\_chart](https://pub.dev/packages/fl_chart).
- [9] *Flutter framework*. Available at: <https://flutter.dev/>.
- [10] *Ganache*. Available at: <https://trufflesuite.com/ganache/>.
- [11] *MetaMask developer documentation*. Available at: <https://docs.metamask.io/>.
- [12] *MySQL official page*. Available at: <https://www.mysql.com/>.
- [13] *Truffle suite*. Available at: <https://trufflesuite.com/>.
- [14] *WalletConnect*. Available at: <https://walletconnect.com/>.
- [15] *WalletConnect Terms of Service*. Available at: <https://walletconnect.com/terms>.
- [16] *WalletConnect Web3Modal documentation*. Available at: <https://docs.walletconnect.com/web3modal/about>.
- [17] C. KOMALAVALLI, D. S. and LAROIYA, C. Handbook of research on blockchain technology. In: Elsevier, 2020, chap. 14. OVERVIEW OF BLOCKCHAIN TECHNOLOGY CONCEPTS.
- [18] IVANA, S. and HOMOLIAK, I. *SBvote: Scalable Self-Tallying Blockchain-Based Voting* online. 2020. Available at: <https://arxiv.org/pdf/2206.06019.pdf>.

- [19] SZABO, N. *Formalizing and Securing Relationships on Public Networks*. 1997. Available at: <https://firstmonday.org/ojs/index.php/fm/article/view/548>.
- [20] T., T. *Ethereum EVM illustrated*. Available at: [https://takenobu-hs.github.io/downloads/ethereum\\_evm\\_illustrated.pdf](https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf).
- [21] TERRA, J. *What is Web 1.0, Web 2.0, and Web 3.0? Definitions, Differences Similarities*. Available at: <https://www.simplilearn.com/what-is-web-1-0-web-2-0-and-web-3-0-with-their-difference-article>.
- [22] VENUGOPALAN, e. a. *BBB-Voting: 1-out-of-k Blockchain-Based Boardroom Voting* online. 2022. Available at: <https://arxiv.org/pdf/2206.06019.pdf>.
- [23] WOOD, G. *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER*. Available at: <https://gavwood.com/paper.pdf>. EIP-150 REVISION.
- [24] ZHENG, G.; GAO, L.; HUANG, L. and GUAN, J. Ethereum Smart Contract Development in Solidity. In:. 2021, chap. Chapter 5. Application Binary Interface (ABI).

# Appendix A

## Contents of the attached DVD

The attached DVD contains the following items:

- `src\` - Source code:
  - `flutter-application\` - Source code for Android application on Flutter. Client-side of the application.
  - `be\` - Source code for Flask server and databases dump. Server-side of application.
  - `elections-truffle-proj\` - Source code of truffle project implementing SBBB-Voting protocol made by Stančíková Ivana and Ivan Homoliak.
- `latex\` - Source code of bachelor's thesis text.
- `xpastu04.pdf` - Bachelor's thesis text.