



A Blockchain-based Secure Voting System

Graduation Project (2)

by

Ali Saeed Alghamdi 4102809

Abdalrhman Hamad Alqroni 4104088

Alrasheed Khalid Alrasheed 4002910

Salman Moteb Alharbi 4000826

**A project submitted in partial fulfilment of the requirements for
the degree of Bachelor of Science (Computer Science)**

Supervised by

Dr. Faisal Fahd M Albalwy

2st Semester - Academic Year 1445 (2023/2024)

Abstract

The traditional voting process suffers from several limitations and challenges, such as difficulties in verifying the accuracy of the results, susceptibility to manipulation, fraud, and lack of credibility. Additionally, it is an inefficient process that requires personal attendance at polling stations.

The SmartVoter platform provides an innovative and effective solution to address these challenges. It utilizes cutting-edge blockchain technology to secure the process and ensure transparency and security. Users can easily vote through a web application or a mobile app, recording their votes securely and confidentially.

Keywords Blockchain ; Voting; Security; Transparency ; Smart Contract;

Acknowledgement

We express our sincere and heartfelt gratitude to our esteemed supervisor, Dr. Faisal Albalwy, for his unwavering guidance, patience, and understanding. His encouragement and support have been the bedrock of our progress and the driving force behind the successful completion of our project.

We are blessed to be surrounded by a circle of family and friends whose support has been unfailing. We are deeply thankful for their patience, enduring support, and love, which have been a source of strength and motivation throughout our academic endeavor.

Finally, our thanks go to Taibah University for their generous assistance and provision of resources necessary for this project. Their support has been instrumental in allowing us to achieve our research objectives.

Table of Content

Abstract	ii
Acknowledgement	iii
List of Figures.....	vii
List of Tables	viii
List of Abbreviations	ix
1 Chapter 1: Introduction.....	1
1.1 Introduction.....	1
1.2 Problem Definition and Motivation	2
1.3 Project Objectives	2
1.4 Project Scope	3
1.5 Project Timeline	4
1.6 Document Organization.....	5
2 Chapter 2: Literature Review	7
2.1 Introduction.....	7
2.2 Background.....	7
2.2.1 Introduction to blockchain.....	8
2.2.2 Terms and Terminologies	8
2.2.3 Blockchain network types	9
2.2.4 Consensus Algorithm	10
2.2.5 Different Platforms.....	11
2.2.6 Solidity	12
2.3 Related Work	12
2.3.1 Review of Relevant Work	12
2.3.2 Relationship Between the Relevant Work and Our Own Work	14

2.4	Summary.....	14
3	Chapter 3: System Analysis	15
3.1	Introduction.....	15
3.2	Requirements Elicitation	15
3.2.1	Functional Requirements.....	16
3.2.2	Non-Functional Requirements	17
3.2.3	User Requirements or Domain Requirements	17
3.3	Requirements Specification	18
3.4	Developmental (or Research) Methodology.....	25
3.5	Summary.....	28
4	Chapter 4: System Design.....	29
4.1	Introduction.....	29
4.2	Architectural Design	29
4.3	Object Oriented Design	31
4.3.1	Structural Static Models	31
4.1	Smart Contract Interface.....	33
4.1.1	Dynamic Models	36
4.2	Business Model.....	48
4.3	User Interface Design.....	49
4.4	Summary.....	51
5	Chapter 5: System Implementation.....	52
5.1	Introduction.....	52
5.2	Tools and Languages.....	52
5.3	Main/Most Important Codes.....	53
5.4	System Testing.....	55

5.5	Summary.....	58
6	Chapter 6: Conclusion and Future Work	60
6.1	Conclusion	60
6.2	Goals Achieved	60
6.3	Limitations and Future Work.....	61
7	References.....	64

List of Figures

Figure 1:TimeLine.....	4
Figure 2:TimeLine2.....	4
Figure 3: Relationship Between the Relevant Work and Our Own Work	14
Figure 4:UseCaseDigram	19
Figure 5: Agile software development	26
Figure 6: Architectural Design.....	30
Figure 7: Class diagram.....	32
Figure 8: Sequence daigram for Voter Registration.....	36
Figure 9: Sequence diagram for Joining Voting	38
Figure 10: Sequence diagram for Reveling Voting.....	40
Figure 11: Sequence diagram for Terminating Voting	41
Figure 12: Sequence diagram for Create Voting	43
Figure 13:State diagram.....	44
Figure 14: ER MODEL	46
Figure 15: Interface for Create Account.....	49
Figure 16:AdminControl	50
Figure 17: Interface creating a poll	50
Figure 18:Terminate	51
Figure 19:CreateVoting	53
Figure 20:JoiningVote	54
Figure 21:Revealing	54
Figure 22:GenerateProof	54
Figure 23:TestForUserRegister.....	55
Figure 24:TestForApprove	56
Figure 25:CreateTest	57
Figure 26:Result	58
Figure 27:Result2	58

List of Tables

Table 1: Use Case for User Registration	20
Table 2: Use Case for Joining Voting	20
Table 3: Use Case for Revealing Voting	21
Table 4: Use Case for Terminating Voting.....	23
Table 5: Use Case for Create Voting.....	23
Table 6: Use Case for Identity Verification.....	24
Table 7: Use Case for userApprove	25
Table 8: Tool.....	27

List of Abbreviations

DApp Decentralized application.

ER Entity Relationship

SC Smart Contract

DB Data Base

Chapter 1: Introduction

1.1 Introduction

The history of voting dates back to ancient times, where it was traditionally conducted through paper ballots or by raising hands [6]. As time passed and technology advanced, more sophisticated methods of voting were developed to facilitate the process and increase transparency and credibility.

In recent years, blockchain technology has emerged as one of the most significant innovations in the field of technology. Blockchain is a decentralized and encrypted technology that allows secure and transparent storage and recording of data and transactions. It relies on a network of connected computers working together to verify and validate the integrity of the data and transactions.

Integrating voting with blockchain technology offers new and exciting possibilities for the field of elections. Blockchain can be used to record and secure votes in a decentralized and encrypted manner, reducing the risks of manipulation and fraud in voting processes. Each vote is recorded as a transaction on the blockchain, and its validity and authenticity are verified and documented by the connected network.

Blockchain technology also provides transparency and ensures the accuracy of results. Everyone can access the blockchain and verify the integrity of the transactions and results. Due to its encrypted nature, it becomes difficult to alter or manipulate records without being detected [7].

Furthermore, the integration of voting and blockchain technology can enhance the security and transparency of the voting process. Voters can use electronic devices to cast their votes, which are securely recorded on the blockchain. Data and information are secured through encryption and encrypted verification, enhancing the process's security and protecting voters' privacy.

At present, some countries and institutions are adopting blockchain technology in electronic voting processes and achieving promising results. With ongoing technological advancements and improved security systems and encryption, the integration of voting and

blockchain technology is expected to continue in the future, becoming a primary system for voting processes and opinion polls.

However, it is important to note that blockchain technology and electronic voting face challenges and issues such as validity issues: difficulties in validating the accuracy of election results [8], data storage challenges: storing participant statuses and results [8], third-party risks: vulnerabilities resulting from reliance on trusted third parties [9], high costs: expensive bureaucratic administration of elections [10], system threats: vulnerability to DoS attacks and low availability [11], bandwagon effect: voters being influenced by the perceived popularity of choices [12], and in-person voting: inefficiency due to the mandatory physical presence at the polls.

With continuous monitoring and technological advancements, electronic voting with the use of blockchain technology can become a major method in electoral processes and opinion polls, contributing to enhancing democracy and providing a convenient and reliable voting experience for citizens in the future.

1.2 Problem Definition and Motivation

If these challenges persist, severe consequences may arise, including a decline in public trust and participation in democratic systems. The risk of manipulation and fraud threatens the fundamental principles of democracy.

This is particularly evident in witnessing numerous cases of manipulation in electronic voting in various fields, including football. The lack of credibility, cheating, and repetition of votes have led to unfairness and injustice for some players. As a solution, we have decided to create an electronic voting system supported by blockchain technology with the aim of achieving fair competition among all participants.

1.3 Project Objectives

This The project aims to complete the analysis and modeling stages of the system in order to provide a secure electronic voting process without manipulation by the parties involved and to achieve maximum efficiency in the voting process. This will be achieved in the current phase of the project (the first phase), with future implementation on platforms. To accomplish this goal, we have worked on achieving the following objectives:

- Reviewing previous similar projects to identify the requirements and specifications for a secure electronic voting platform, including the use of blockchain technology and encryption techniques to ensure security and transparency.
- Studying the traditional voting process and analyzing its challenges and weaknesses, particularly regarding security and transparency.
- Developing a secure electronic voting platform based on the defined requirements and specifications, including designing a user-friendly interface.
- Explore the feasibility of using Ethereum-based smart contracts for vote verification.
- Evaluate the efficiency of different encryption techniques in ensuring vote secrecy.
- Utilizing Blockchain Technology: Explore and integrate blockchain technology, such as Ethereum, to provide transparency, immutability, and tamper-proof recording of votes.
- Implementing Identity Verification: Develop a reliable and secure identity verification system to authenticate voters and prevent fraudulent voting.

1.4 Project Scope

This project aims to design a prototype for a blockchain-based voting system with the objective of enhancing the security and transparency of the voting process. The prototype will incorporate blockchain technologies to achieve accurate and secure vote auditing and to ensure the participation of a larger number of users in the voting process. It is expected that this project will play a crucial role in improving the electoral processes and increasing trust in the voting system.

By utilizing blockchain technology, the security and transparency of the voting process will be improved, contributing to enhancing credibility and engaging voters in the decision-making process.

1.5 Project Timeline

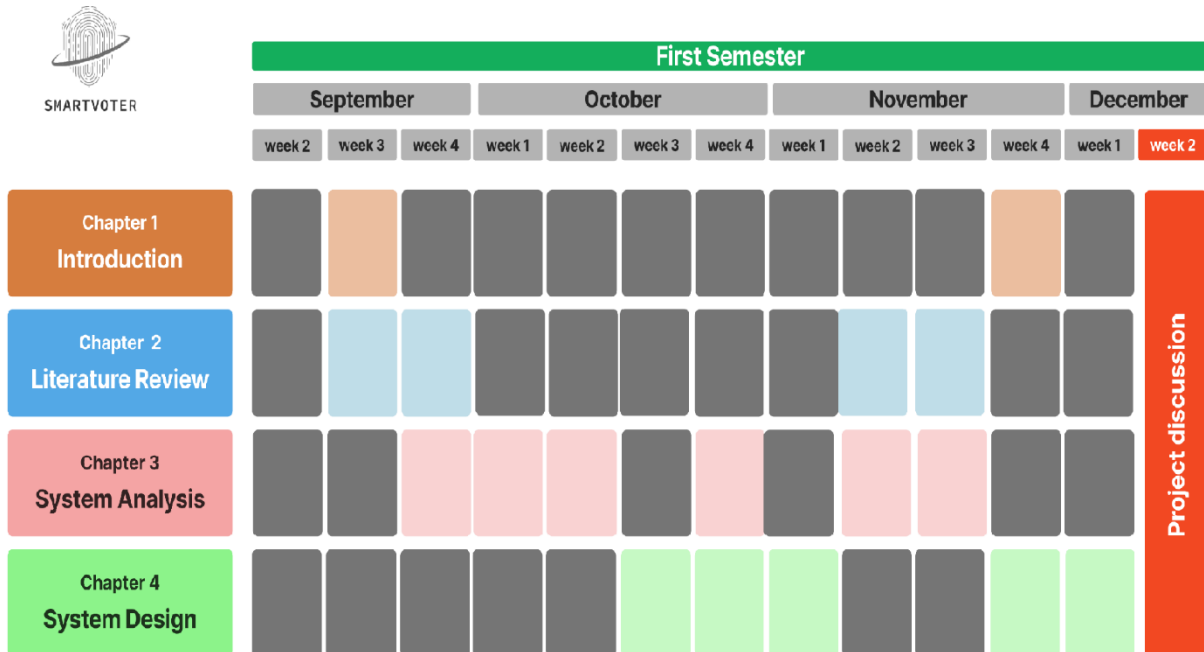


Figure 1:TimeLine

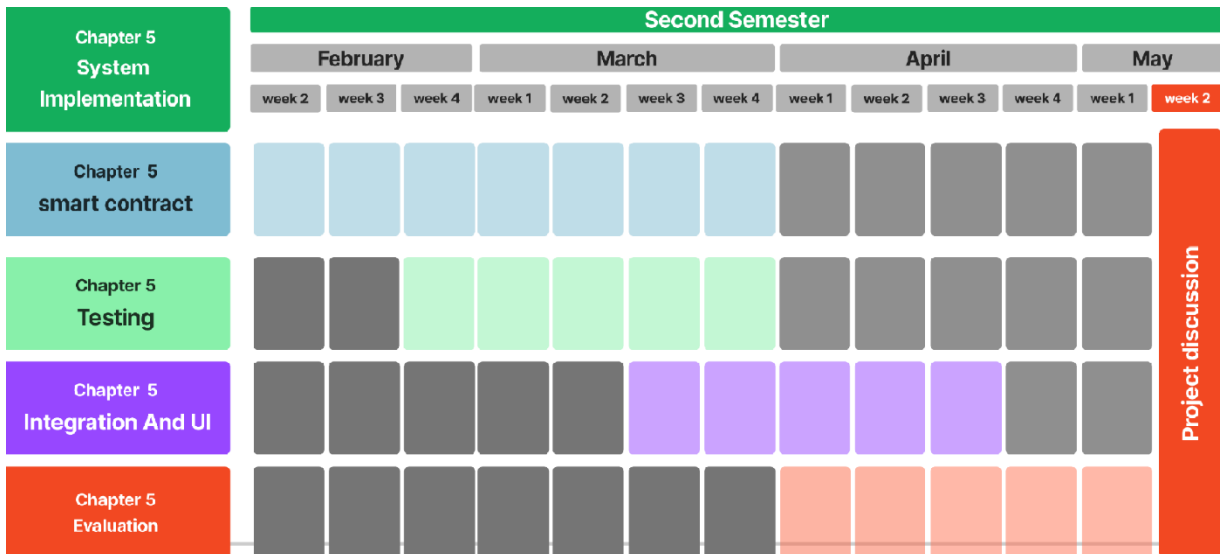


Figure 2:TimeLine2

1.6 Document Organization

This project consists of four chapters in addition to one (or several) appendixes. These chapters are organized to reflect the scientific steps toward our main objective. A brief description about the contents of each chapter is given in the following paragraphs:

1. Chapter 1: Introduction

- Problem Definition: Explains the core problem or issue addressed by the project.
- Objectives: Identifies the goals and aims the project seeks to achieve.
- Approach Used: Clarifies the methods and approaches used in executing the project.
- Scope of Work: Defines the boundaries and areas covered by the project.
- Project Layout: Presents an overview of the organization and arrangement of the project parts.

2. Chapter 2: Literature Review

- Overview: Provides foundational information about previous studies and research related to the project's topic.
- Related Work: Discusses research and projects that are relevant to the study's subject.
- Relational Table: Offers a classification or arrangement of the information and studies related to the project.

3. Chapter 3: System Analysis

- Requirement Elicitation: Includes identifying functional, non-functional, and domain requirements.
- Requirement Specification: Details the project's requirements thoroughly.
- Developmental Methodology: Explains the method used in developing the system.

4. Chapter 4: Design of the Proposed System

- Architectural Design: Displays the basic structure of the system.
- Object-Oriented Design: Includes structural static models and dynamic models.
- User Interface Design: Focuses on developing an interactive and user-friendly interface for the system.

Chapter 2: Literature Review

2.1 Introduction

This literature review aims to explore the use of blockchain technology in voting systems. Research and academic works in this area are analyzed to understand how to improve the fairness, security, and efficiency of voting processes via blockchain. Emphasis is placed on the benefits of blockchain such as transparency, tamper-proof, and decentralization, in addition to examining the challenges and issues associated with implementing this technology in voting systems.

2.2 Background

A blockchain-based voting system is designed to enhance traditional voting processes by utilizing blockchain technology, offering transparency, security, and efficiency. Every vote is securely recorded in the blockchain network, and its integrity cannot be compromised. Smart contracts can be used to streamline the voting process further. Nevertheless, challenges related to privacy and scalability need innovative solutions for successful implementation.

By harnessing the decentralized, immutable, and secure aspects of blockchain technology, such a system can significantly improve transparency, security, and efficiency. However, addressing privacy concerns and scalability issues is vital for effective integration into voting systems.

Decentralization is a fundamental feature of blockchain, with multiple nodes maintaining copies of the entire ledger. This structure ensures that voting data is distributed across the network, enhancing transparency and verification.

Immutability is another key element. Once a vote is recorded on the blockchain, it becomes permanent and cannot be altered, providing a high level of security and trust in the voting process. Transparency and Auditability are inherent to blockchain. The transparent nature of the technology allows for public verification and auditing of the voting process, ensuring accountability. Additionally, it generates an auditable trail of votes, facilitating post-election audits and result verification.

2.2.1 Introduction to blockchain

Blockchain has become an omnipresent term that encompasses a social promise and a new technology. Originally proposed as a solution for Bitcoin's cryptocurrency record keeping system, blockchains are now used to store the records of all types of applications. Blockchain means something more in many people's minds. The promise many associate with blockchain applications is that they will collapse all centralized systems. Centralized systems are everywhere people need to trust a counterparty and don't have the resources themselves to do so independently. An easy way to identify a place where blockchain technology may be applied is to look for areas where a middleman is needed to facilitate trust. Trust is essential for things such as the transfer of money, voting, land records, IP rights, and identity. Blockchain software can be programmed to take the place of the middleman by becoming the trusted record keeping system [20].

2.2.2 Terms and Terminologies

Some of the terms and terminologies you may encounter while going through this Project is described here.

- **Block** is used to store the transaction along with their hash value and data.
- **Transaction** Any state change occurred in a blockchain.
- **Smart contract** self-executing contract with terms and conditions written in lines of codes.
- **Ledger** Blockchain ledger is used to record the transactions in a blockchain.
- **Hash** The encrypted value of the data in the block.
- **Node** Each computer connected to the blockchain network.
- **Solidity** Programming language for writing smart contracts in Ethereum.
- **Mining** The validation process in a blockchain (in Bitcoin and Ethereum).
- **Wallets** Digital wallet to store, send and receive cryptocurrencies and other digital assets.
- **Peer2Peer(P2P)** Decentralized network architecture. There is no dedicated server in this case.

2.2.3 Blockchain network types

In blockchain technology, there are three distinct types of networks: Public, Private, and Hybrid blockchains. Here are simplified descriptions for each[20].

- **Public Blockchain**

Public chains are considered open-source systems and are known as permissionless ledgers, where any participant can join the network without requiring prior permission. Participants in these networks have full rights to read and write into the network, and they possess identical copies of the ledger. Public blockchains operate efficiently in trustless networks due to the immutable nature of the records. Examples of public blockchains include Bitcoin and Ethereum.

- **Private Blockchain**

Private blockchains function as permissioned ledgers, allowing only select participants access to the network. Every participant has access to the complete record of transactions and associated blocks. The information in all blocks is encrypted with a private key, rendering it unreadable to outsiders. These blockchains are exclusively controlled by authorized users from specific organizations. Businesses may permit only trusted nodes to join their network. User validation is often facilitated by an identity management system, making this type of blockchain ideal for internal ledger sharing.

- **Hybrid Blockchain**

A hybrid blockchain, also known as a consortium blockchain, represents a fusion of public and private blockchain characteristics. This type of blockchain features a semi-decentralized and semi-private structure. It supports attributes of both public and private blockchains within a hybrid network. While it operates with a controlled user group, it spans across various organizations.

Table 1: Comparison of Blockchain Types

Property	Public	Private	Hybrid
Consensus algorithm	Permissionless	Permissioned. Only authorized users are allowed	Permissioned. Only
Efficiency	Low	High	High
Determination of consensus	All miners of the block will be participating	Users of one organization will be participating	Selected set of nodes only will be participating
Read	Open to public, anyone can read	Can be public or restricted	Can be public or restricted
Immutability	Tampering is not possible	Possibility of tampering	Possibility of tampering

2.2.4 Consensus Algorithm

A consensus algorithm in blockchain is a set of rules used to ensure that all parties in a blockchain network agree on the current state of the distributed ledger and can trust the validity of the records it contains. These mechanisms are essential for maintaining a decentralized and trustworthy system where there is no central authority controlling the data. Some of the methods are described below[21].

- **Proof of Work (POW):**

First algorithm POW is developed and used in Bitcoin cryptocurrency. This algorithm is widely used in mining process. To add a block, miners are attempting to solve the puzzle. Miners are trying to find out the hash value for the next block. In this algorithm, the nodes/computers in the network agree on the hash of the minor and block will be added in the network. First miner will get the reward for solving puzzle. But, it requires high computational power for solving the puzzle.

- **Proof of Stake (POS):**

It is a common alternative of POW. Here, the validators are chosen based on the fraction of coins they own in the system. The nodes with more number of coins have more chance to be selected than the node with lesser number of coins. In POS the reward is in the form of transaction fee, new coins are not created for paying the validators. Presently, Ethereum, BlackCoin , NXT and Peercoin blockchains uses the POS algorithm.

- **Proof of Activity (POA):**

POA is a hybrid approach, and it is introduced to overcome some of the problems in POS and POW. In this method, the mining begins with POW and at some point the process is switched POS. Presently, 'Decred' is the only coin that is using a variation of proof of activity.

2.2.5 Different Platforms

In the blockchain, various platforms have emerged, each contributing to the development and diversification of decentralized systems. Here is a simplified description of each[21].

- **Bitcoin Blockchain:**

Bitcoin is the first well-known blockchain network based on cryptocurrency. It follows POW consensus algorithm. This is a single-chain architecture and C++ is used for programming.

- **Ethereum:**

Ethereum is most popular platform used by various communities. It is an open-source platform and used widely for testing purpose. It is designed for smart contracts and large variety of decentralized applications. It is a public network and open-source network. Ethereum supports Solidity language for programming. Ethereum offers faster processing and private transactions within a permissioned group of participants within a network.

- **Hyperledger:**

Hyperledger is a collaborative effort from different industry leaders to frame an open source, Cross-Industry Blockchain aided technologies. The movement basically aims to develop the distributed ledgers that can support enterprise level business transactions. The entire project is developed on the open-source platform. Even though the project is hosted and driven by the free folk of the internet 'Linux Foundation', technology giants like IBM, Intel, Samsung and many more others already became part of the project.

2.2.6 Solidity

Solidity is a high-level programming language, which is designed to work with the technology of the time -Blockchain. More specifically speaking, Solidity is designed to develop Smart Contracts in Ethereum Blockchain platform [21] .

- **Smart Contracts:**

A contract in the sense of Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum Blockchain. In each Contract, we can define State Variables, Methods, and Events etc. This contract can manage transactions between blocks in Blockchain network. Each block has a particular address in the form of a cryptographic key that generated by the result of some functions including hashing of adjacent blocks. This creates a strong relationship between adjacent blocks. So that manipulation or any other form of hacking in nodes or blocks are not easy or not even possible. Solidity is one of the many languages that can be used to develop EVM (Ethereum Virtual Machine) understandable bytecode. There are many built-in classes and Libraries in Solidity which support hassle free smart contract development. You can use the IDEs like Remix, Visual Studio (With Solidity extension), Ether atom, IntelliJ IDEA plugin (both with Solidity extension) to develop. Following are some of the features of solidity which are very similar to common high-level languages like Java and C++.

2.3 Related Work

In this section, we present a review of several apps that provide some of the related work to our app. We determine the advantages and disadvantages of each of them. Additionally, we clarify the similarities and differences between these apps and Smart voter.

2.3.1 Review of Relevant Work

- Follow my vote:



This company offers a secure online voting platform based on blockchain technology, featuring the capability to audit polling boxes and observe real-time democratic processes [13]. The platform allows voters to submit their votes remotely and securely, and to choose their preferred candidate. It then enables the use of their identification to literally open the ballot box, locate their ballot, and verify that it is correct and that the election results are mathematically proven to be accurate.

- Voatz

This company has developed a smartphone-based voting system utilizing blockchain technology, enabling remote and anonymous voting, along with verification to ensure votes are correctly counted [14]. Voters use the application to validate their identities and those of the candidates, providing evidence including a photograph and identification details, along with a unique biometric signature such as fingerprints or retina scans for added security.



- Agora

This collective has rolled out a digital voting platform underpinned by blockchain technology. Founded in 2015, it saw partial deployment during the March 2018 presidential elections in Sierra Leone. The framework of Agora hinges on a suite of tech innovations: a proprietary blockchain, a participatory security model, and a robust consensus algorithm [15]. Within Agora's ecosystem, the native token is known as 'Vote.' It incentivizes citizens and electoral authorities globally to partake in and uphold a voting process that is both secure and transparent. 'Vote' stands as the universal currency within the Agora network.



- Polyas

Established in Finland in 1996, the firm utilizes blockchain technology to offer electronic voting systems to both public and private entities [16]. In 2016, Polyas gained recognition from the German Federal Office for Information Security for its robust electronic voting solutions. Numerous



prominent German corporations rely on Polyas for conducting their electronic voting needs. Currently, Polyas serves a diverse clientele across the United States and Europe.

2.3.2 Relationship Between the Relevant Work and Our Own Work

In this Figure 3 shows a comparison chart of various blockchain-based voting systems, detailing their framework, programming language, platform compatibility, user privacy, security, data integrity, and open-source availability.

properties of /Relevant Work	Agora	Polyas	Voatz	Follow My Vote	Quick Voting	Smart Voter
Framework	Bitcoin	Private/Local Blockchains	Hyperledger Fabric	Bitcoin	Various Technologies	Ethereum
Language	Python	NP	Go/JavaScript	C++/Python	-	Solidity
Platform Compatibility	web-based	web-based	Applications	web-based	web-based	web-based
User Privacy	✓	✓	✓	✓	✗	✓
Security	✓	✓	✓	✓	✗	✓
Data Integrity	✓	✗	✗	✗	✗	✓
Open source	✗	✗	✗	✗	✗	✓

Figure 3: Relationship Between the Relevant Work and Our Own Work

2.4 Summary

In this chapter, we discussed the studies that illustrate the importance of SmartVoter app and the background of our work. Also, we reviewed some previous work relevant to SmartVoter. As a result, we used this knowledge to overcome the weaknesses and enhance the strengths of our project. Thus, the project will end successfully and produce a work result that meets individuals' needs.

Chapter 3: System Analysis

3.1 Introduction

This chapter contains three major introductory sections for the development of an innovative blockchain-based solution. It begins with the requirements elicitation section, which covers the functional, non-functional, and user/domain requirements of our project.

Following this, the second section presents the use case diagram with a detailed description that illustrates the actors and their interactions with the system, along with a detailed explanation for each use case, its event flow, and the system's response to the actors. The final section addresses the software development process, explaining the selection of the blockchain platform for the project, and all the necessary tools for completion. The preceding part to these sections introduces the vital role of system analysis in exposing vulnerabilities and inefficiencies in current voting mechanisms, thereby paving the way for the development of this pioneering solution that ensures secure and transparent voting processes.

3.2 Requirements Elicitation

The requirements elicitation process focuses on clarifying the system's purpose; the client, developers, and users collaborate to identify a specific problem and define a system that addresses this issue. This definition, known as a requirements specification, acts as a contract between the client and developers and is essential for understanding the true needs of the users [17].

The following requirements were selected based on the topics discussed in chapter two and the examination of systems related to this project, which aided in distinguishing the most vital key requirements. Here, we will describe the functional and non-functional requirements in natural language that is comprehensible to both the developer and the client, ensuring clarity and effective communication.

3.2.1 Functional Requirements

The Functional Requirements ensure that the blockchain-based voting system functions as intended. These requirements include:

- **Voter**

- As a Voter, I need to register on the platform by providing personal details (name, wallet address) to be eligible to vote, ensuring my information is securely hashed and stored off-chain.
- As a Voter, I want to join the voting process by submitting a hashed proof of my vote during the voting phase, ensuring my choice is private and secure.
- As a Voter, I need to reveal my vote in the post-voting phase by submitting my original vote and the salt for verification, ensuring my vote is accurately counted.
- As a Voter, I can manage my wallet to access the website.

- **Voting Coordinator**

- As a Voting Coordinator, I need to register on the platform by providing personal details (name, wallet address) to be eligible to vote, ensuring my information is securely hashed and stored off-chain.
- As a Voting Coordinator, I need to create a voting event with specific parameters (title, description, timeframe, options) and ensure these details are recorded on the blockchain to facilitate an organized and transparent voting process.
- As a Voting Coordinator, I can manage my wallet to access the website, and I also have the ability to make payments using it.

- **System Administrator**

- As a System Administrator, I can manage my wallet to access the website.
- As a System Administrator, I want Smart Contract deployment to be streamlined and secure, ensuring reliable integration and maintenance on the blockchain.

- As a System Administrator, I want Smart Contract Management to ensure seamless operation, efficient troubleshooting, and robust security of the contracts on the blockchain.

3.2.2 Non-Functional Requirements

Non-Functional Requirements are the standards that dictate how a system operates, rather than the specific behaviors it performs (which are defined by the Functional Requirements). For a blockchain-based voting system, the include:

- **System Security:** The system utilizes advanced encryption techniques to ensure that votes are recorded accurately and protected from modification or deletion.
- **Transparency:** The system provides a transparent voting process that can be publicly verified while protecting voter privacy.
- **Integrity:** The system ensures accuracy and stability of data, with votes being immutable once recorded.
- **Usability:** The front-end interface should be user-friendly, intuitive, and accessible, accommodating users with varying levels of technical proficiency and ensuring ease of use for all functionalities.
- **User Privacy:** User privacy must be maintained, especially in storing and processing personal information. Sensitive data must be securely stored off-chain, and only essential data should be recorded on the blockchain.
- **Availability:** Due to decentralized network, user can make of use DApp and its functions even if some nodes in the blockchain network are inactive.

3.2.3 User Requirements or Domain Requirements

User Requirements in the context of a blockchain-based voting system refer to the specific needs and expectations of the end-users who interact with the system. These requirements include:

- User must have internet connection to use application.
- User must have a blockchain wallet.

- User must have a national identity.
- User must maintain the salt.

3.3 Requirements Specification

This type of diagram is used in software engineering to depict the interactions between users (called "actors") and the system itself to achieve specific goals. Here's a detailed description and This includes a specific image below **Error! Reference source not found.:**

- **Actors:**
 - Voter: An individual who participates in the voting process.
 - Voting Coordinator: A person who manages the voting process.
 - System Administrator: The individual responsible for the technical maintenance and operation of the voting system.
 - Identity Verification Partner: An entity or individual responsible for verifying the identities of users within the system.
- **Use Cases:**
 - Joining Voting: The Voter can join the voting process.
 - Revealing Voting: The Voter can reveal their vote at the appropriate stage in the voting process.
 - Deposit: The Voting Coordinator Deposit money to initiate the voting process.
 - User Registration: A Voter or Voting Coordinator can register on the SmartVoter platform.
 - Create Voting: The Voting Coordinator can create a new voting event.
 - Terminating Voting: Automatically, the smart contract concludes the voting at the set time, processes, and then publishes the results for public transparency.
 - Wallet Management: Both the Voter and Voting Coordinator can manage their digital wallets, which are used for transactions on the platform.
 - Smart Contract Deployment: The System Administrator is responsible for deploying smart contracts, which are self-executing contracts with the terms of the agreement directly written into lines of code.
 - Smart Contract Management: The System Administrator manages these smart contracts throughout their lifecycle.

- Withdrawal: The System Administrator also handles the withdrawal process, which could be related to financial transactions within the system.
- User Approve: administrator approve a user's registration, changing their status from 'unverified' to 'verified', enabling them to participate in voting activities on the website.

- **Relationships:**

- The arrows indicate the interactions each actor has with the different use cases. For example, the Voter is linked to joining voting, revealing voting, deposit, user registration, and wallet management.
- The createVoting function in the SmartVoter contract allows a voting coordinator to set up a new voting session with required parameters and a fee, implicitly including a process to terminate the vote after its completion.

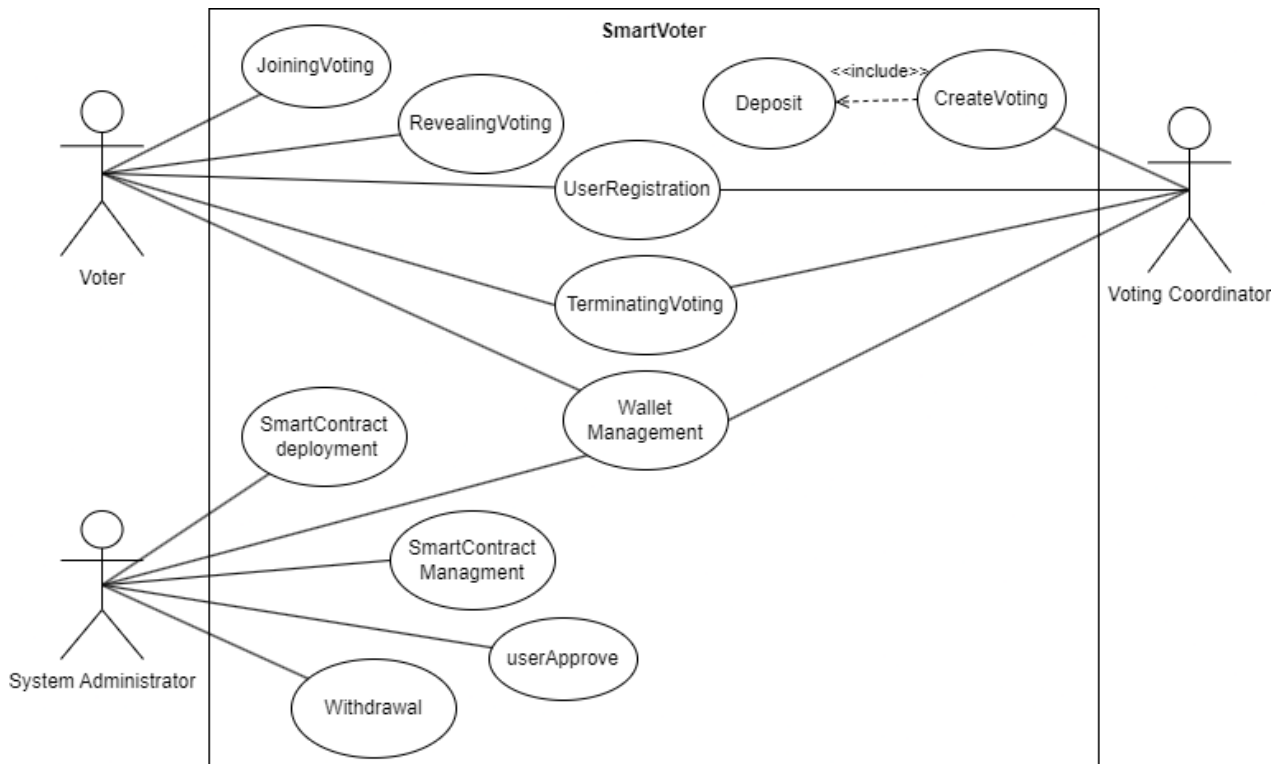


Figure 4: Use Case Diagram

- **These tables contain detailed explanations below:**

Table 2: Use Case for User Registration

Name	User Registration
Brief description	New users register to participate in the voting process.
Participating actors	Voter, Voting Coordinator
Entry condition	The voter has access to the registration portal and he has a wallet.
Exit condition	The voter is registered and can log into the system.
Flow of events	<ol style="list-style-type: none"> 1. The process begins with the user accessing the frontend interface and filling out the registration form. 2. Once the registration form is completed, the frontend hashes the data and stores it in an off-chain database. It also generates a unique reference for the user. 3. The off-chain database invokes a function in the smart contracts and sets the user's status to 'Pending'.
Special requirement	<ul style="list-style-type: none"> • User Privacy: User privacy must be maintained, especially in the storage and processing of personal information, ensuring that sensitive data is protected.

Table 3: Use Case for Joining Voting

Name	Joining Voting
Brief description	Voters can cast their votes.
Participating actors	Voter.
Entry condition	The voter is registered, authenticated, and the poll is open for voting.
Exit condition	The voter has successfully submitted their vote.
Flow of events	<ol style="list-style-type: none"> 1. The process begins with the voter accessing the voting page.

	<ol style="list-style-type: none"> 2. On the voting page, the voter selects their preferred voting option and adds a 'salt', which is a random piece of data used to hide the vote in the hashing process to ensure privacy and security. 3. The voter then secures the salt, meaning they save it in a secure manner for future reference or verification. 4. With the voting option and salt chosen, the voter generates voting proof. By hashing the voting options and salts a choice, this proof, combined with the salt, will be used to verify the vote without revealing the voter's choice. 5. Finally, the voter submits the voting proof to the smart contract on the blockchain.
Special requirement	None

Table 4: Use Case for Revealing Voting

Name	Revealing Voting
Brief description	Voters reveal their votes in the post-voting phase by submitting the original vote and the salt for verification.
Participating actors	Voter
Entry condition	After the Joining Voting.
Exit condition	Votes are revealed and counted.
Flow of events	<ol style="list-style-type: none"> 1. The voter starts the process by accessing the voting page specifically for the purpose of revealing their vote.

	<ol style="list-style-type: none"> 2. The voter submits their vote along with the salt (the random data that was used during the voting proof generation phase). 3. The voting page sends the voter's vote and the associated salt to the smart contract. 4. The smart contract uses the vote and salt to regenerate the voting proof, which should match the proof submitted during the voting phase. 5. Inside the smart contract, a comparison is made between the regenerated voting proof and the original voting proof provided during the voting phase. <ul style="list-style-type: none"> • If Voting Proof Matches: The smart contract counts the vote as valid. The voting page reveals the vote status as 'Valid' to the voter. • If Voting Proof Mismatch: The vote is considered invalid. The voting page reveals the vote status as 'Invalid' to the voter.
Special requirement	None

Table 5: Use Case for Terminating Voting

Name	Terminating Voting
Brief description	This use case handles the closure of the voting process.
Participating actors	Voting Coordinator
Entry condition	After the Revealing Voting.
Exit condition	Voting process is closed and results are ready to be revealed.
Flow of events	<ol style="list-style-type: none"> 1. The process is initiated by the Voting Coordinator who starts the result tallying process. 2. The command from the Voting Coordinator is received by the voting page, which then initiates the result tallying procedure. 3. The smart contract on the blockchain processes the final tally of the votes. 4. Once the final tally is processed, the voting page publishes and displays the results. 5. Finally, users (voters or other interested parties) access the voting page and view the results of the voting process.
Special requirement	None

Table 6: Use Case for Create Voting

Name	Create Voting
Brief description	A voting coordinator sets up a new poll.
Participating actors	Voting Coordinator
Entry condition	Voting Coordinator is authenticated in the system.
Exit condition	A new poll is available for voters.
Flow of events	<ol style="list-style-type: none"> 1. The Voting Coordinator begins by accessing the page dedicated to creating a new vote. 2. The Voting Coordinator enters the necessary information for the vote, which includes the description, title, and the options that will be available for voting. 3. After the voting information is entered, the Create Voting Page

	<p>displays the price for creating the vote.</p> <p>4. Finally, the Voting Coordinator submits the vote creation information along with a deposit to the Smart Contract.</p>
Special requirement	None

Table 7: Use Case for Identity Verification

Name	Identity Verification
Brief description	ensures that a voter's identity is verified by an external Identity Verification Partner through the Oracle Service to maintain the integrity of the voting process.
Participating actors	Identity Verification Partner, Oracle Service
Entry condition	The user must register, and their status will show as Pending.
Exit condition	The result of the identity verification process
Flow of events	<ol style="list-style-type: none"> 1. The smart contracts trigger the Oracle Service to begin identity verification. 2. The Oracle Service requests verification from an external identity verification partner. 3. The identity verification partner responds with the verification status. 4. The Oracle Service updates the smart contracts with the user's verification status. 5. The smart contracts update the off-chain database with the status, which can be 'Verified' or 'Unverified'.
Special requirement	None

Table 8: Use Case for user Approve.

Name	user Approve
Brief description	Approves or rejects a user's registration based on their role.
Participating actors	Identity Verification Partner, Oracle Service
Entry condition	The result has reached the oracle service, which will invoke the user Approve.
Exit condition	Change of status.
Flow of events	<ol style="list-style-type: none">1. Upon receiving the verification confirmation, the Oracle Service calls the user Approve function.2. Once the user Approve function is executed, the smart contract updates the blockchain with the user's new status.3. The front-end system notifies the user of their verification status.
Special requirement	None

3.4 Developmental (or Research) Methodology

The section is integral to any comprehensive study or project report, especially when delving into fields such as software development, scientific research, or academic inquiry. This methodology outlines the systematic approach that researchers or developers will take to address the problem at hand.

It details the techniques, tools, and processes that will be employed to gather data, analyze information, and develop solutions or contributions to the body of knowledge.

By articulating a clear and structured methodology, the groundwork is laid for a transparent, replicable, and rigorous investigation or development process, ensuring that the results are credible, and the work can be verified by peers.

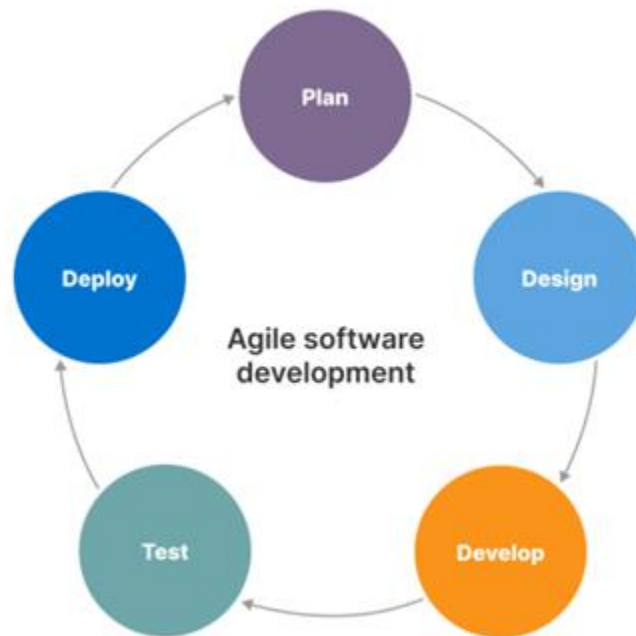


Figure 5: Agile software development

The image shown represents a model of the Agile Software Development process. This model illustrates the software development lifecycle following the Agile methodology, consisting of five main phases that are cyclically and continuously repeated to enhance the process and the final product.

1. **Plan:** Identifying goals, requirements, planning tasks, and the necessary resources [18].
2. **Design:** Designing the system, architecture, and user interface based on requirements [18].
3. **Develop:** Writing code and developing functions and components [18].
4. **Test:** Examining the code and features to ensure they are error-free and meet the requirements [18].
5. **Deploy:** Launching the ready version for users and monitoring its operation [18].

The core idea of the Agile methodology is iteration and flexibility in the development process; this approach allows development teams to quickly adapt to changes and continuously improve the product based on feedback from end-users or clients [19].

Table 9: Tool

Tool	Description
Remix IDE	An interface for programming and debugging Ethereum smart contracts in Solidity language.
Truffle Framework	A development and testing environment for Ethereum smart contracts.
React Framework	A JavaScript framework for developing user interfaces for applications.
Drizzle Framework	A framework that facilitates interaction between the user interface and smart contracts.
Ganache	Used to create a local Ethereum blockchain for development and testing purposes.
MetaMask	A wallet that enables interaction with the blockchain and management of Ethereum accounts.
Node.js	A runtime environment for building backend services in JavaScript.
MongoDB	A database for off-chain storage.
Discord	A social communication and collaborative platform, used for coordination among development teams.
Canva	An online graphic design tool, used for creating design materials like logos and presentations.
Figma	An online design and collaboration tool, used for designing user interfaces and collaborating on design projects.
StarUML	A tool for designing and documenting models using UML (Unified Modeling Language), used for planning, and documenting the architecture of software and systems.
Web3	is a term that refers to the third generation of the internet, where networks are decentralized and built on blockchain technologies. This allows users to interact directly and securely without the need for central intermediaries, and is characterized by increased privacy, transparency, and trust.

Choosing the right tools is a critical factor in the success of software development projects, especially in a dynamic and rapidly changing environment like blockchain. Developers must select tools that fit the requirements of the project and allow for flexibility and adaptation.

3.5 Summary

This chapter comprehensively addresses three key sections crucial for the foundation of our blockchain-based voting system. The initial segment, requirement elicitation, delves into the functional and non-functional prerequisites, along with user and domain-specific needs, to capture the system's full scope. Following this, the requirement specification section elucidates the use case diagram, offering a high-level overview and detailed narratives for each potential system interaction. This includes the voting use case, which exemplifies the user's journey from authentication to casting a vote, with stringent safeguards to ensure vote integrity and confidentiality.

As we draw the chapter to a close, we transition into outlining our developmental methodologies. Here, we articulate the software development processes to be employed, the chosen blockchain platform, and the suite of tools that will be instrumental in realizing our project's objectives. This lays down a strategic roadmap for the subsequent phases of design and development.

The chapter culminates with a synthesis of the current system's constraints and a clearly delineated set of requirements and methodologies. This convergence forms the blueprint that will guide the forthcoming design and implementation phases of our sophisticated, blockchain-based voting system, which will be expounded upon in the next chapter.

Chapter 4: System Design

4.1 Introduction

In this chapter, we explore the design and implementation of a complex system, focusing on architectural and object-oriented design. We discussed static and dynamic models, including registration and voting processes. Additionally, we delve into database design and user interface, emphasizing clarity and efficiency. This chapter provides a comprehensive foundation for understanding the system's overall architecture.

4.2 Architectural Design

Users and Roles: The system identifies three types of users: a regular user (voter), a Voting Coordinator, and an Identity Verification Partner.

- **User Interactions:**

- Voter: Can initiate transactions by invoking smart contract functions.
- Voting Coordinator: Manages the overall voting process.
- Identity Verification Partner: Responsible for verifying the identity of users to ensure that only authorized individuals participate in the voting process.

- **Resources:**

- On-chain resources: These include smart contracts that generate on-chain data, logs, and events. Users, particularly voters, can access this data for transaction logs and events, which are crucial for ensuring transparency and auditability in the voting system.
- Off-chain resources: Resources not stored on the blockchain but accessible to authorized users. The diagram indicates an off-chain secure database and an oracle service, potentially used for retrieving real-world data pertinent to the smart contracts.
- Smart Contracts: Act as the cornerstone of the voting system, orchestrating the logic for creating on-chain data and interfacing with off-chain resources as necessary.

- **Data Management:**

- Logs and Event Management: This functionality records all system actions and events, available for user review. This might include votes cast, amendments to the voting protocol, and other significant occurrences.
- Access Data Management: Refers to the protocols for controlling and overseeing access to different system components, likely involving the Identity Verification Partner.
- This system leverages the intrinsic properties of blockchain, such as immutability, transparency, and security, making it ideally suited for critical applications like voting where integrity is of utmost importance.

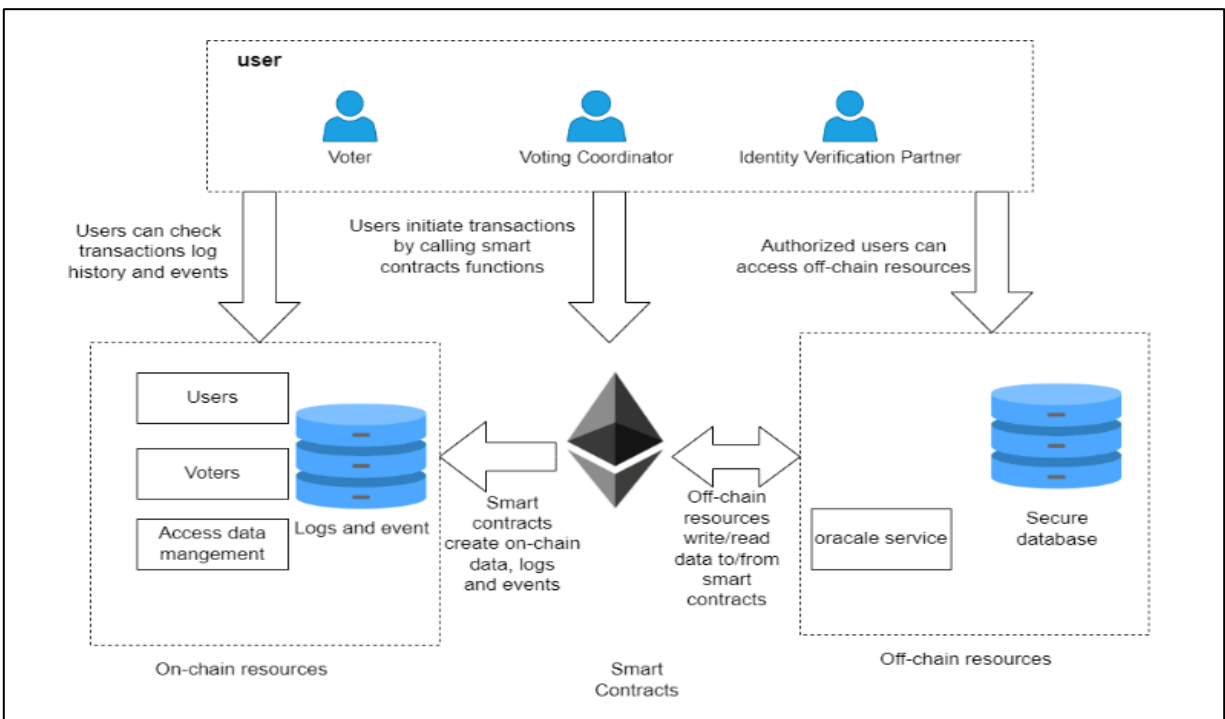


Figure 6: Architectural Design

4.3 Object Oriented Design

This section is divided into two sections: Dynamic Models and Structural Static Models.

The Dynamic Models represent the system's behavior over time, as depicted through the Sequence Diagram and the Activity Diagram.

The Structural Static Models represent the system's structure, including the ER and class diagram.

4.3.1 Structural Static Models

Here's a detailed description of the Smart_Voter class structure, as illustrated in Figure 7, for a blockchain-based voting application. This class is pivotal for orchestrating the various components of the voting process, which includes handling user registrations, voting transactions, and maintaining the integrity of the election lifecycle. It ensures a secure and streamlined experience for all participants within the blockchain network:

- **Statuses and Roles:** To manage the state of the voting process and define user permissions.
- **Variables:** Such as fees, administrator address, and voting-related identifiers.
- **Mappings:** To associate votes and users with their data, and track voting proofs.
- **Functions:** For actions like user registration, vote creation, and voting process management.
- **Events:** To log user actions and votes for transparency.

The User and Vote are entities managed by Smart_Voter, with each user having a wallet, role, and status, and each vote having attributes like title, options, and dates.

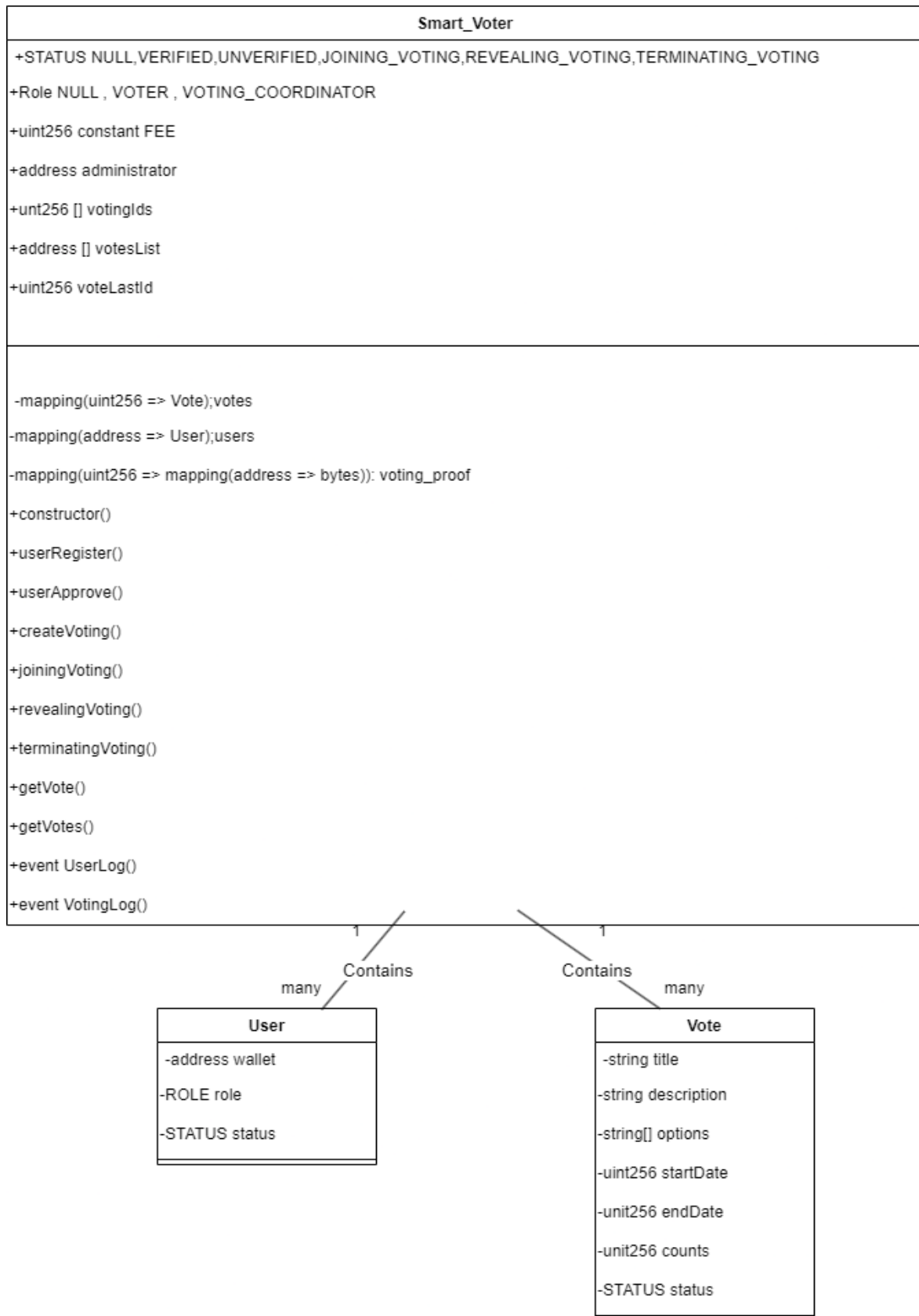


Figure 7: Class diagram

4.1 Smart Contract Interface

This code is an interface definition for an Ethereum smart contract designed to manage a secure voting system, specifying the necessary functions and structures for user registration and managing the voting process.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

/// @title Secure Voting System Interface
/// @notice Interface for the Secure_Voting smart contract
interface ISecureVoting {
    // Enumerations for user and voting status
    enum STATUS {
        NULL,
        VERIFIED,
        UNVERIFIED,
        JOINING_VOTING,
        REVEALING_VOTING,
        TERMINATING_VOTING
    }

    enum ROLE {
        NULL,
        VOTER,
        VOTING_COORDINATOR
    }

    // Struct for voting details
    struct Vote {
        string title;
        string description;
        string[] options;
    }
}
```

```

uint256 revealingVotingDeadline;

uint256 joiningVotingDeadline;

uint256 counts;

// STATUS status;
}

// Struct for user details
struct User {
    address wallet;

    ROLE role;

    STATUS status;
}

// Events for logging user and voting activities
event UserLog(
    address indexed userWallet,

    ROLE role,

    STATUS status,

    uint256 timestamp
);

// Events for logging user and voting activities
event createVotingLog(
    uint256 indexed voteId,

    address indexed coordinator,

    uint256 timestamp
);

event joiningVotingLog(
    uint256 indexed voteId,

    address voter,

    bytes32 proof,

    uint256 timestamp

```

```

);

event revealingVotingLog(
    uint256 indexed voteId,
    address voter,
    string option,
    uint256 timestamp
);

event VotingTerminated(uint256 indexed voteId, string[] options, uint256[] results);


event withdrawalLog(address withdrawer, uint256 amount);


// Interface functions
function userRegister(ROLE _role) external;
function userApprove(address _userWallet) external;
function createVoting(
    string memory _title,
    string memory _description,
    string[] memory _options,
    uint256 _joiningVotingDuration,
    uint256 _revealingVotingDuration
) external payable;
function joiningVoting(uint256 voteId, bytes32 _proof) external;
function revealingVoting(
    uint256 voteId,
    string memory option,
    uint256 _salt
) external;
function terminatingVoting(uint256 voteId) external returns (string[] memory options, uint256[] memory results)
;

function getVote(uint256 voteId) external view returns (Vote memory);
function getVotes(

```

```

// uint256 voteId
) external view returns (uint256[] memory, Vote[] memory);
}

```

4.1.1 Dynamic Models

The Dynamic Models represent the system's behavior over time, as depicted through the Sequence Diagram and the Activity Diagram.

4.1.1.1 Registration Process

The image shows Figure 8 a diagram of a voter registration process using smart contract technology, where data is processed between user interfaces, databases, and identity verification systems to ensure a secure and reliable voter registration.

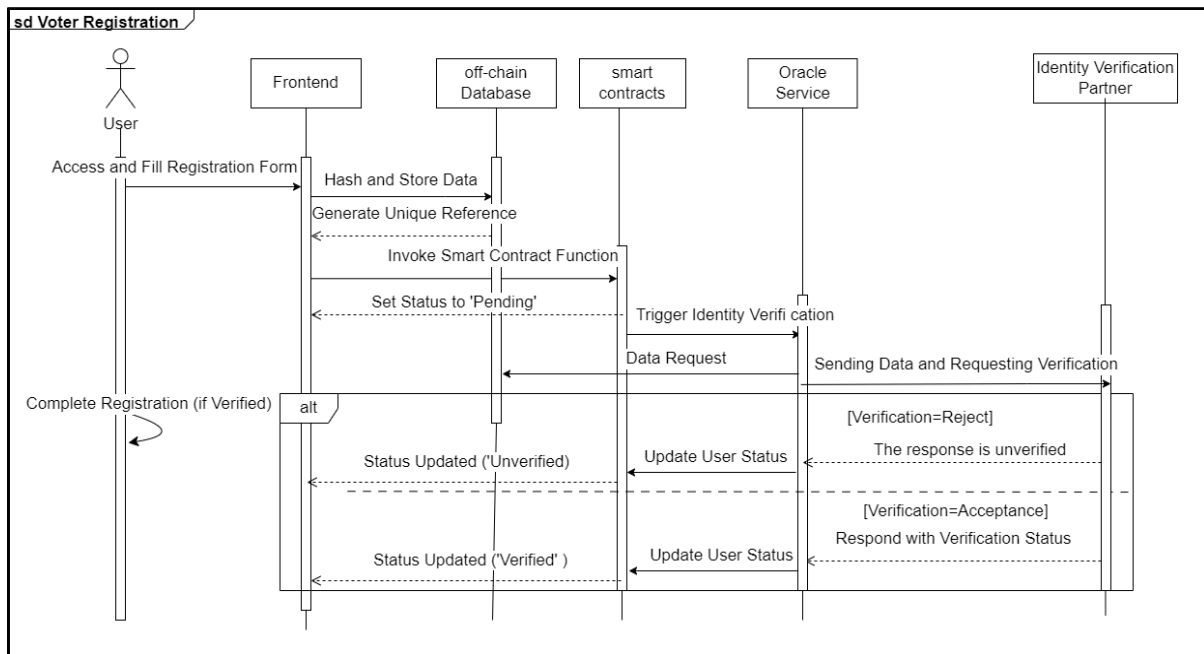


Figure 8: Sequence daigram for Voter Registration

1. Accessing the Registration Page:

- Users (Voters and Voting Coordinators) access the registration page via the system's front-end interface.

2. Form Submission:

- Users fill in the registration form, providing necessary information such as their name, ID, Date of Birth, wallet address, and other relevant details.

3. Data Hashing and Off-Chain Storage:

- Once the form is submitted, the user's information is hashed in the front-end. This hashed data, which anonymizes personal details, is then sent to the offchain database.
- The off-chain database securely stores the hashed personal information and generates a unique database reference for each user's record.

4. Blockchain Record Creation:

- Users then invoke a smart contract function from the front-end. This function records the registration on the blockchain.
- The information passed to the blockchain includes user metadata (nonsensitive data) along with the hash values and the database reference from the off-chain storage.
- At this point, the user's status is set to 'Pending.'

5. Identity Verification via Oracle Service:

- The Oracle Service is automatically triggered to verify the user's identity. It sends a request to the Identity Verification Partner (e.g., Nafad) with the user's details.
- The Identity Verification Partner processes this request and responds with a verification status – either 'true' (verified) or 'false' (unverified).

6. Updating User Registration Status:

- If the response is 'true,' the user's status on the blockchain is updated to Verified, indicating that their identity has been confirmed.
- If the response is 'false,' the status remains 'Unverified,' indicating a problem with the identity verification.

7. Completion of Registration:

- Once verified, the user is fully registered and eligible to use the system, including participating in voting activities.

4.1.1.2 Phase 1: Joining Voting

The image Figure 9 is a sequence diagram for the process of joining a vote using a blockchain-based system. It outlines the steps a voter takes to access the voting page, select an option, and submit their vote securely through a smart contract.

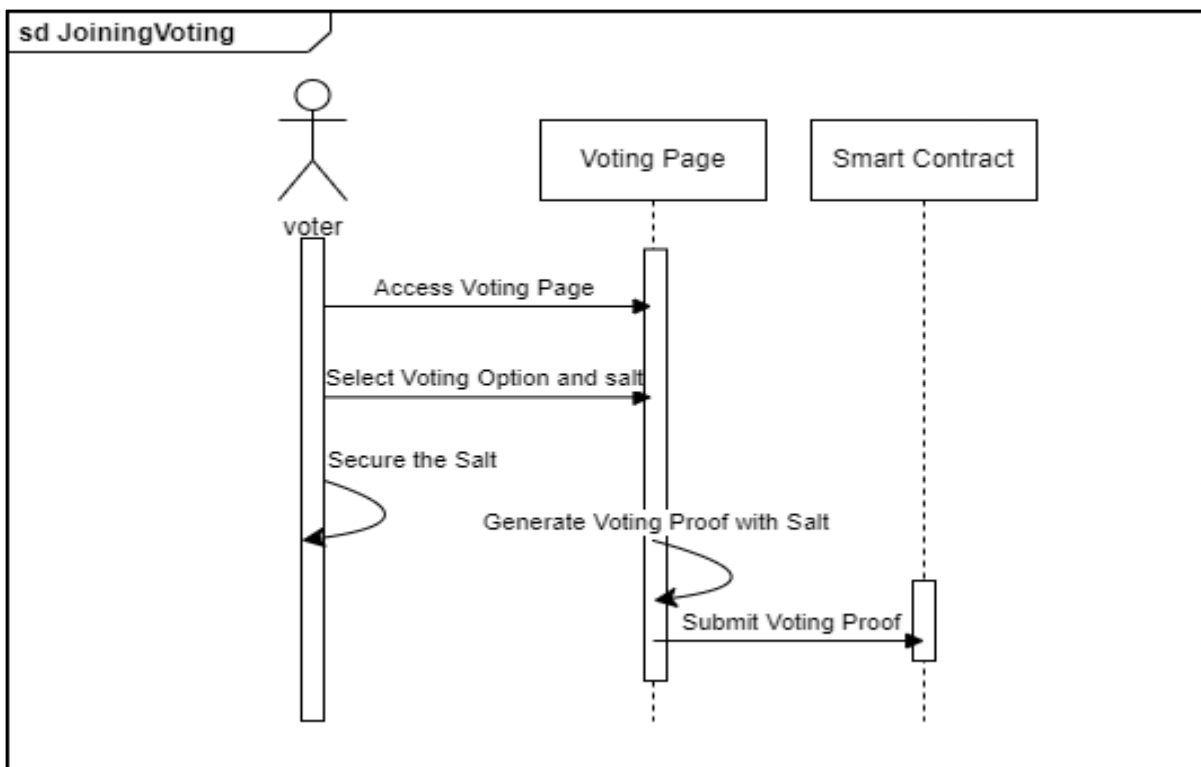


Figure 9: Sequence diagram for Joining Voting

1. Accessing the Voting Page:
 - During the specified timeframe for a poll, voters access the voting page on the system's front-end interface.
2. Voting Option Selection:
 - Voters select their preferred option from the choices provided for the poll.
3. Generating Voting Proof:
 - Voters enter a random 'salt' – a piece of data added to make the hash function output unique.
 - The selected voting option is combined with the salt and hashed to create a 'voting proof.'
4. Securing the Salt:
 - It is crucial for voters to keep their salt secure and confidential, as it will be needed in the next phase for vote verification.
5. Submitting the Voting Proof:
 - A smart contract function for joining the voting is called from the front-end, where voters submit their voting proof.
 - This approach ensures that the actual voting choice remains private, as only the hashed proof is stored on the blockchain during this phase.

4.1.1.3 Phase 2: Revealing Voting

The image Figure 10 is a sequence diagram illustrating the steps involved in revealing a vote in a secure voting system, where the voter submits their vote along with a unique 'salt', and the smart contract is used to verify the authenticity of the vote before counting it.

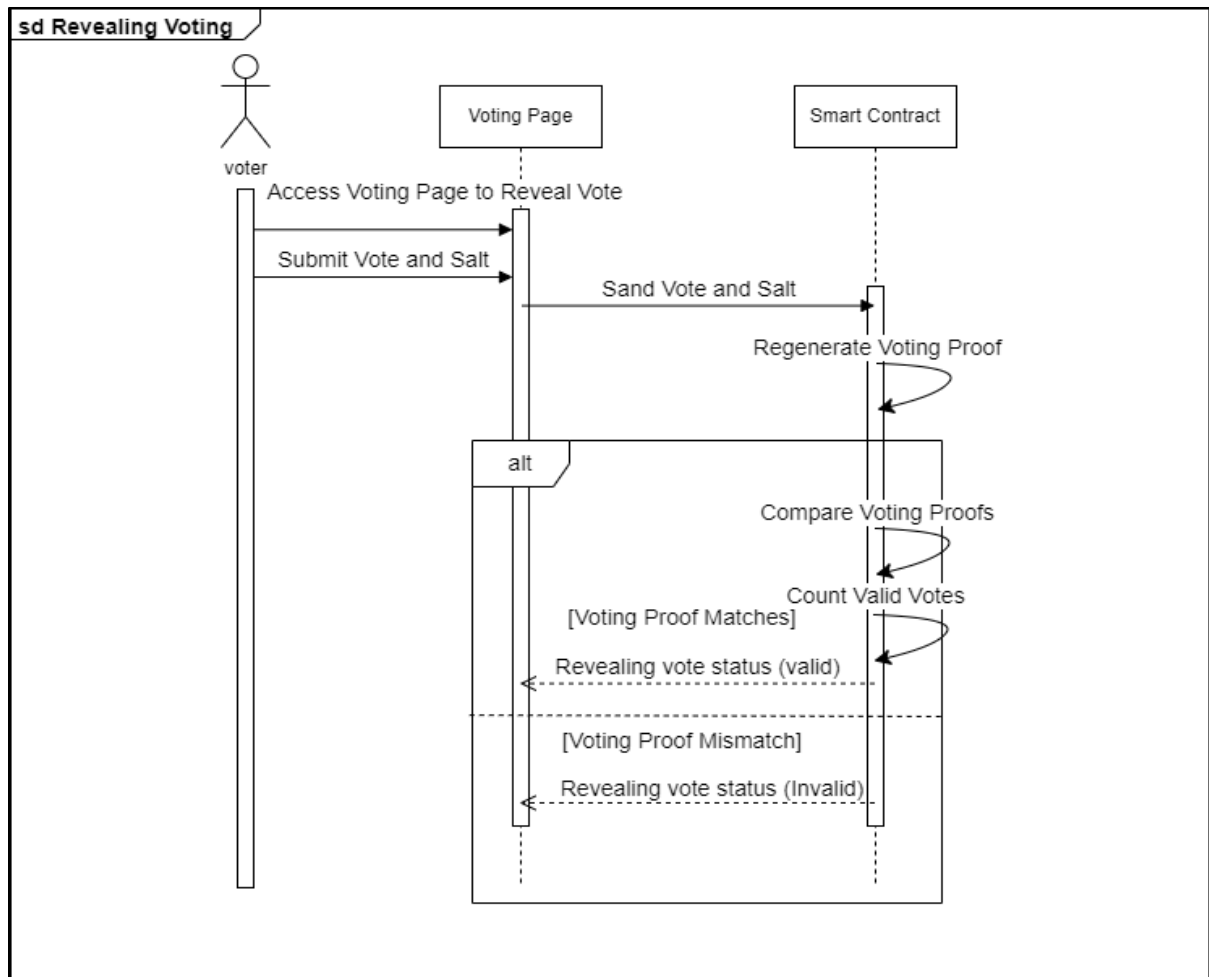


Figure 10: Sequence diagram for Revealing Voting

1. Revealing Vote Post Voting Period:
 - After the initial voting timeframe concludes, voters revisit the voting page to reveal their vote.
2. Vote and Salt Submission:
 - Voters enter their previously selected voting option, and the corresponding salt is used in Phase 1.
3. Smart Contract Verification:
 - The smart contract retrieves the user's voting proof and combines the newly entered vote and salt to generate a hash.

- It then compares this hash with the stored voting proof. If they match, it confirms that the voter is revealing the same option they initially submitted, and the vote is considered valid.

4. Vote Counting:

- Valid votes are counted towards the result, while mismatched or unverified entries are rejected and not included in the tally.

4.1.1.4 Phase 3: Terminating Voting

The image Figure 11 is a sequence diagram depicting the termination phase of a voting process, where a voting coordinator initiates the tallying of results, which are then processed by a smart contract and published for users to view.

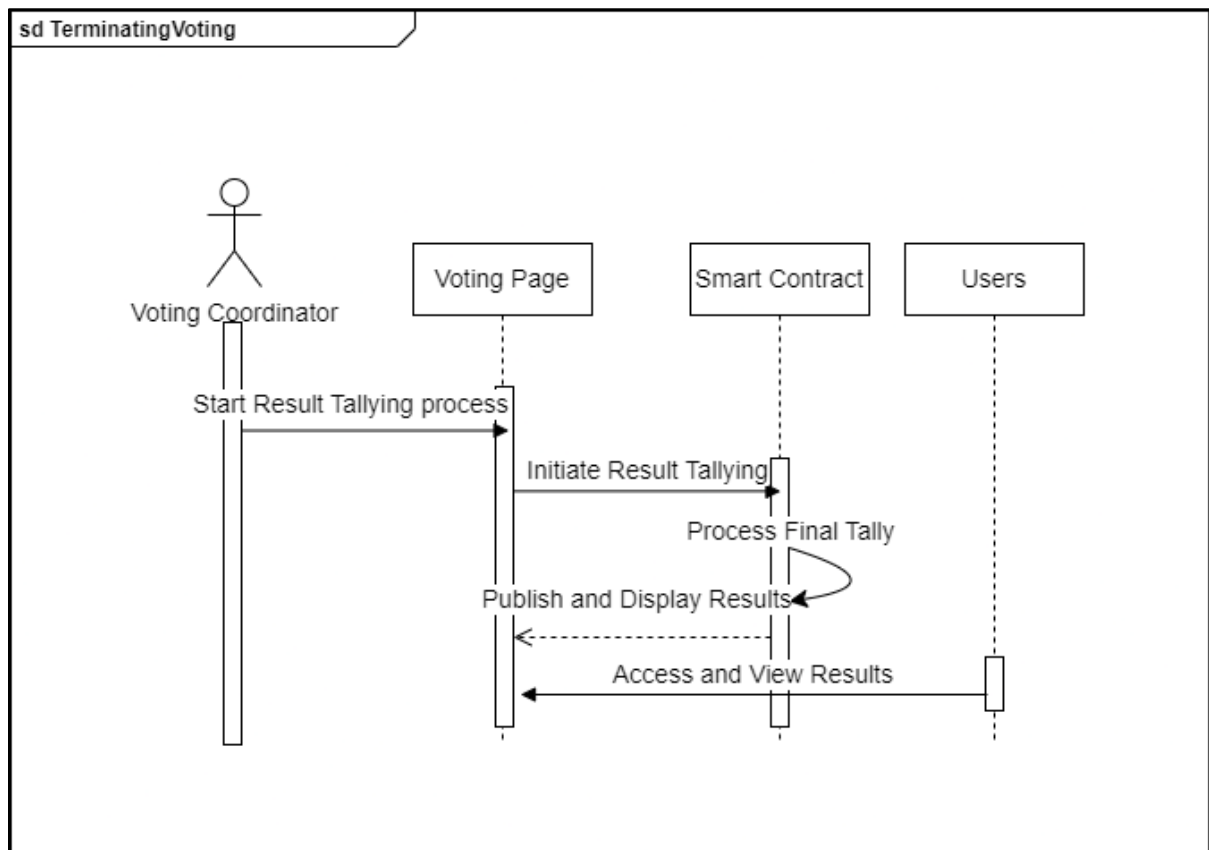


Figure 11: Sequence diagram for Terminating Voting

1. Initiating Result Tallying:

- Once the vote-revealing phase ends, the Voting Coordinator initiates the final counting process.

2. Smart Contract Processing:

- A specific smart contract function is called to start the final tallying process.

3. Publishing Results:

- The results, based on valid entries from Phase 2, are calculated and then published on the blockchain.
- The transparency of blockchain technology ensures that all transactions and actions taken during the voting process are traceable and auditable, fostering trust and accountability.

4. Accessibility of Results:

- All users can view the results and track the transactions involved in each phase, ensuring transparency and verifiability.

4.1.1.5 Create Voting

The image Figure 12 is a sequence diagram showing the steps a voting coordinator follows to create a new voting event on a voting page, which involves entering voting information and submitting it with a deposit to a smart contract.

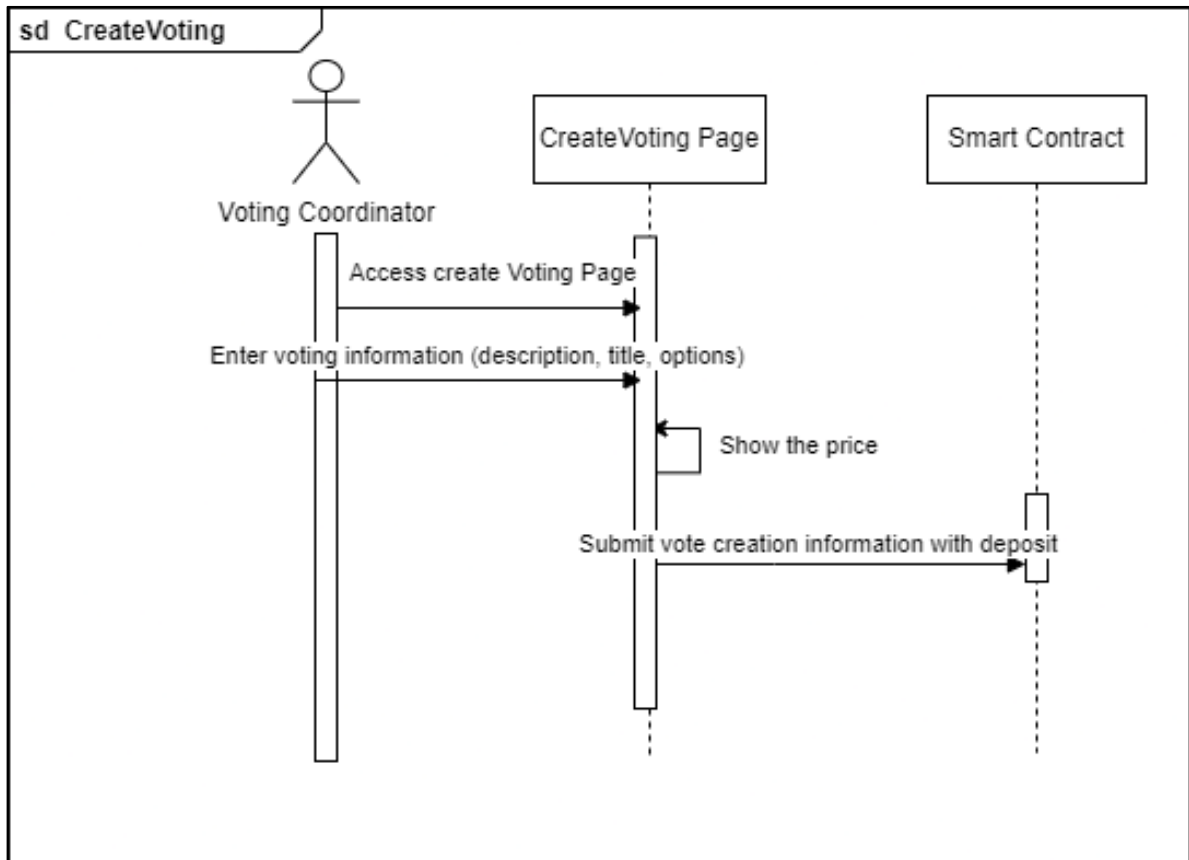


Figure 12: Sequence diagram for Create Voting

1. Accessing the Voting Page:
 - When it's time to vote, voters will log onto the system's front-end interface to access the voting page.
2. Entering Voting Information:
 - On the voting page, voters input necessary information such as the description of what they are voting on, the title of the vote, and the options available.
3. Price Display:
 - After the voting information is entered, the system calculates and displays the cost of creating the vote.
4. Voting Proof Submission:

- Voters generate unique voting proof by hashing their chosen option along with a piece of random data known as 'salt' to ensure the hash is unique.
- This voting proof is then submitted to the system along with a deposit if required.

5. Smart Contract Interaction:

- The voting proof and any other required information are sent to a smart contract, which handles the vote creation process on the blockchain.
- This ensures the integrity and immutability of the vote, as the blockchain records the voting proof without revealing the voter's choice.

4.1.1.6 State Diagram for Voting Status

The image Figure 13 is a flowchart that outlines the decision-making process in a voting system based on user verification. If the user is approved, they proceed to the 'Verified' status, allowing them to join, reveal, and terminate voting. If the user approval fails, they are marked 'Unverified'.

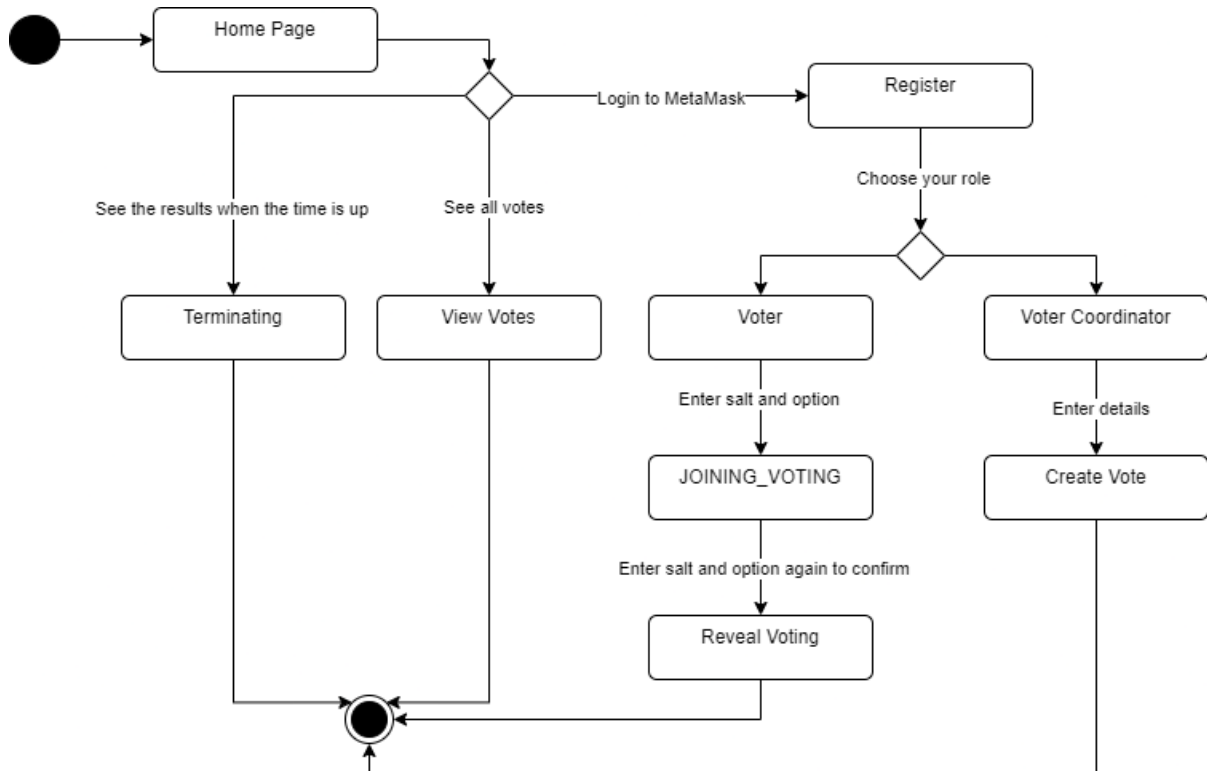


Figure 13:State diagram.

1. Home Page:

- Users start at the home page.

2. Login to MetaMask:

- Users need to log in to MetaMask to interact with the system.

3. Register:

- Users choose to register in the system.

4. Choose Your Role:

- Users select either to be a "Voter" or a "Voter Coordinator".

5. For Voters:

- Users enter information to join the voting (Joining_Voting).
- They confirm their choice by re-entering the salt and option (Enter salt and option again to confirm).
- They reveal their vote in the revealing phase (Reveal Voting).

6. For Voter Coordinators:

- Coordinators enter details to create a new vote (Create Vote).

7. View Votes:

- Users can view all the votes recorded in the system.

8. Terminating:

- Results can be seen when the designated time is up.

9. Return to Home Page:

- After completing the processes, users can return to the home page.

4.1.1.7 Entity-Relationship Model

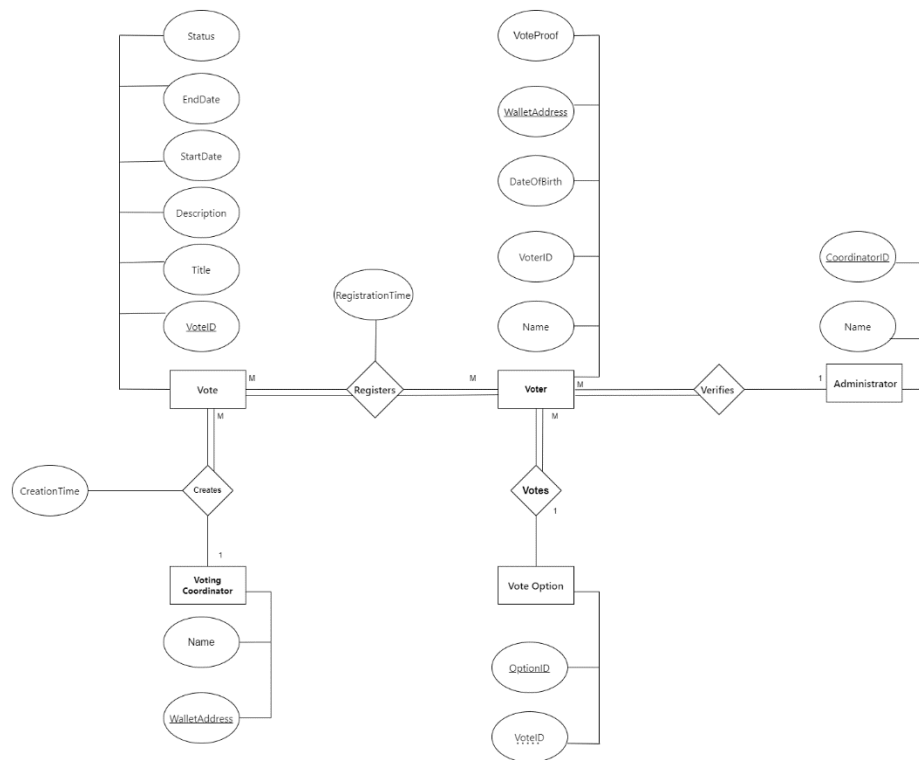


Figure 14: ER MODEL

This diagram Figure 14 is an Entity-Relationship Model (ER Model) of a voting system. This type of model is used to design databases and describe the relationships between various entities within a system. Let's examine each section in detail:

Entities and Attributes:

Vote:

CreationTime: The time when the vote was created.

VoteID: A unique identifier for the vote.

Voter:

VoterID: A unique identifier for the voter.

Name: The voter's name.

DateOfBirth: The voter's date of birth.

WalletAddress: The wallet address of the voter, possibly indicating the use of digital currencies or a digital financial system.

VoteProof:

WalletAddress: The wallet address associated with the voter.

VoterID: The voter's identifier linked to the proof.

VoteOption:

OptionID: A unique identifier for the voting option.

VoteID: The vote identifier associated with the option.

Description: A description of the voting option.

Administrator:

Name: The name of the administrator.

VotingCoordinator:

Name: The name of the voting coordinator.

WalletAddress: The wallet address of the voting coordinator.

Vote (Central):

VoteID: A unique identifier for the vote.

Title: The title of the vote.

Description: A description of the vote.

StartDate: The start date of the vote.

EndDate: The end date of the vote.

Status: The status of the vote (e.g., active, closed).

Relationships:

Registers:

Links Voter to Vote, where voters register to participate in the voting process.

Votes:

Links Voter to VoteOption, where voters select specific options in the vote.

Verifies:

Links Administrator to VoteProof, where administrators verify the voting proof.

Coordinates:

Links Vote to VotingCoordinator, where the voting coordinator manages the voting process.

4.2 Business Model

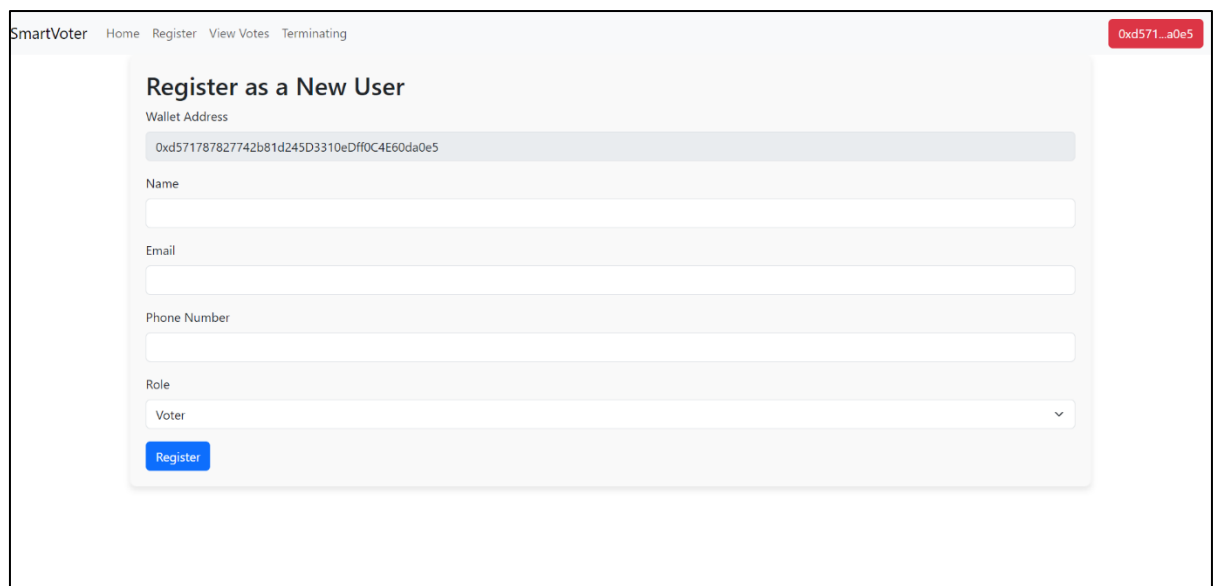
- **Revenue Generation:** The primary revenue source is fees charged to Voting Coordinators for creating polls. These fees, paid in Ethereum (ETH), cover the costs of blockchain resource utilization and system maintenance.

- **Cost Management:** Operational costs are minimized through the automation provided by smart contracts and blockchain technology. Off-chain storage is used for sensitive data, balancing cost with privacy and security.
- **Value Proposition:** The system offers enhanced security and transparency in the voting process, appealing to a wide range of users, including government bodies, corporations, and organizations.
- **Sustainability and Scalability:** The fee-based model ensures the system's sustainability and supports ongoing improvements and scalability for larger user bases and more complex voting scenarios.

4.3 User Interface Design

The image Figure 15 is a registration interface for creating a new account in the "SmartVoter" system, aimed at enrolling new users.

The image is a login screen for "SmartVoter" aimed at user authentication.



The screenshot displays the "SmartVoter" registration interface. At the top, a navigation bar includes links for "Home", "Register", "View Votes", and "Terminating". A red status box in the top right corner shows the wallet address "0xd571...a0e5". The main content area is titled "Register as a New User" and contains a form with the following fields: "Wallet Address" (pre-filled with "0xd571787827742b81d245D3310eDff0C4E60da0e5"), "Name", "Email", "Phone Number", and "Role" (a dropdown menu currently set to "Voter"). A blue "Register" button is located at the bottom of the form.

Figure 15: Interface for Create Account

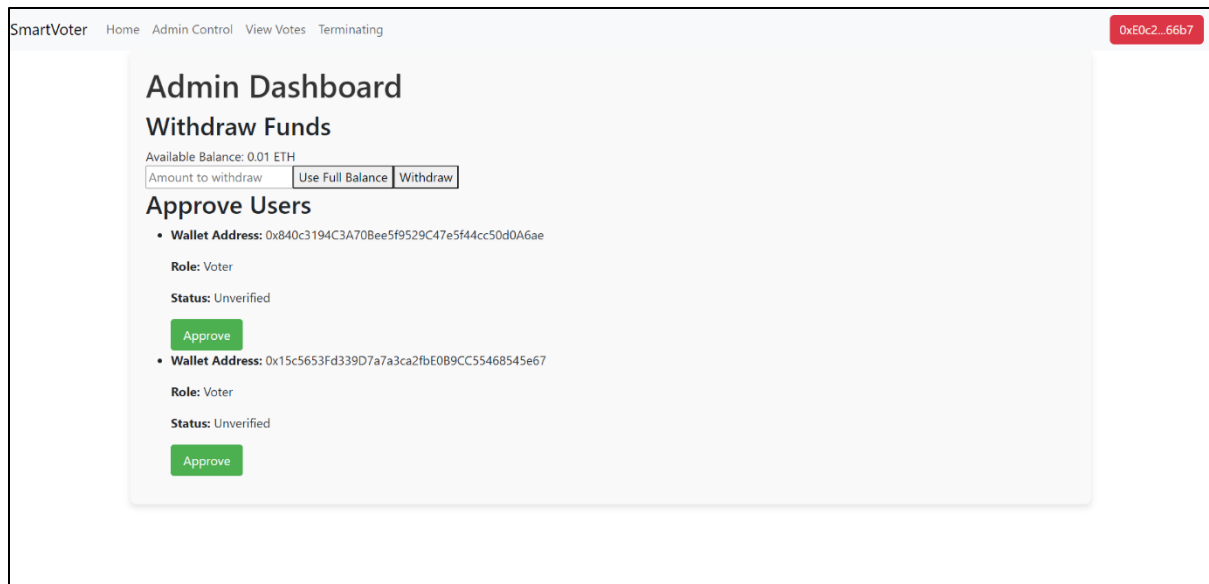


Figure 16:AdminControl

A setup interface Figure 17 for creating a poll with fields for title, options, and scheduling times, within the "SmartVoter" system. The purpose is to facilitate the creation of a voting session.

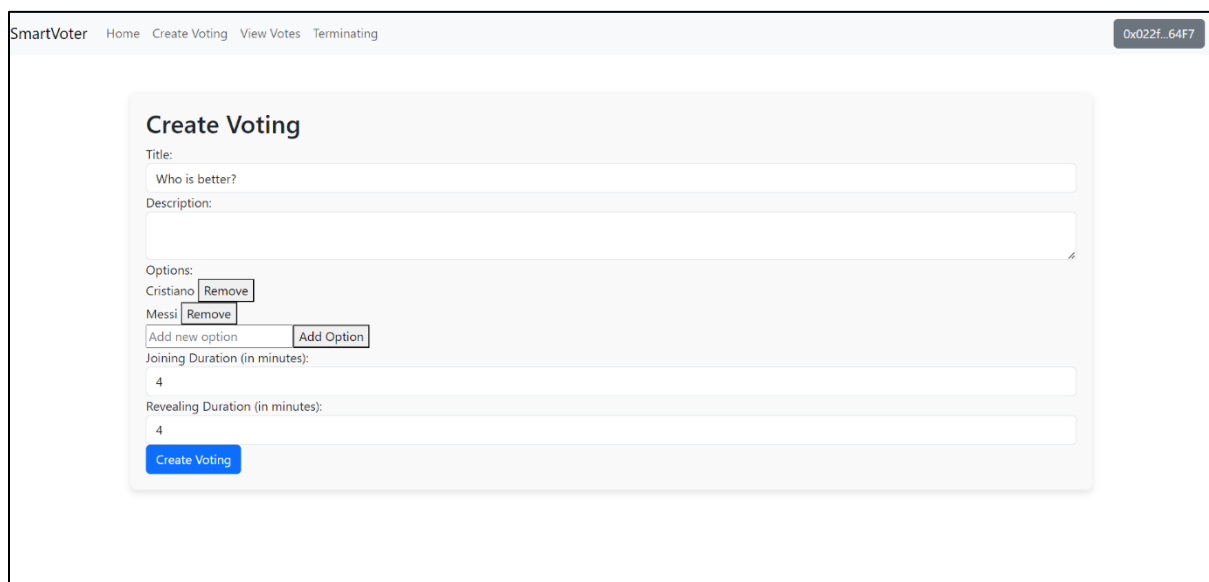


Figure 17: Interface creating a poll

SmartVoter Home Admin Control View Votes Terminating 0xE0c2...66b7

Terminate Voting

Vote ID:

100

Fetch Results

Option	Count
Cristiano	0
Messi	2

Figure 18: Terminate

4.4 Summary

In this chapter we have covered 4 sections of designing our system which include Architectural Design, Object Oriented Design (which includes static and dynamic), Data Modeling and User Interfaces. In the next chapter we will talk about the conclusion from our work and the future work.

Chapter 5: System Implementation

5.1 Introduction

In the 5th chapter of the dissertation, it gives an account of how to implement a secure voting system on a blockchain. The shift is made from design to practice in this part. It gives an overview of tools and programming languages necessary for creating the application while focusing mainly on development environment configuration as well as central functionality coding. Some key code snippets are looked at which help address technical challenges such as voter identification verification and vote privacy. To test the system, different methods were used including unit testing, integration testing and system testing so that they can ensure reliability, security, and usability. then we analyzed their findings against what was expected from them during its creation process. The last part of this chapter assesses the extent to which goals have been achieved by evaluating whether objectives were met. It also provides criticism where applicable but suggests areas for future improvement regarding scalability enhancement, security reinforcement and friendliness towards users when dealing with blockchain based voting systems.

5.2 Tools and Languages

Blockchain Development:

- **Solidity:** Programming language for writing smart contracts on Ethereum.
- **Ethereum:** Blockchain platform used for deploying decentralized applications.
- **Ganache:** Local blockchain for development and testing.
- **Truffle:** Development environment, testing framework, and asset pipeline for Ethereum.

Frontend Development:

- **React:** JavaScript library for building user interfaces, particularly for single-page applications.

- **JavaScript:** Programming language used to implement complex features on web pages, including the interaction logic for the frontend.

Wallet and Transactions:

- **MetaMask:** Crypto wallet and gateway to blockchain apps, used for managing Ethereum wallets and interacting with the Ethereum blockchain.

5.3 Main/Most Important Codes

```
function createVoting(  
    string memory _title,  
    string memory _description,  
    string[] memory _options,  
    uint256 _joiningVotingDuration,  
    uint256 _revealingVotingDuration  
) external payable override onlyCoordinator {  
  
    require(msg.value == FEE, "Must pay the voting fee");  
    require(_options.length >= 2, "At least two options are required");  
  
    Vote storage newVote = votes[nextVoteId];  
    newVote.title = _title;  
    newVote.description = _description;  
    newVote.options = _options;  
    newVote.joiningVotingDeadline = block.timestamp + (_joiningVotingDuration * Minutes);  
    newVote.revealingVotingDeadline = newVote.joiningVotingDeadline + (_revealingVotingDuration * Minutes);  
  
    for (uint256 i = 0; i < _options.length; i++) {  
        options_check[nextVoteId][_options[i]] = true;  
    }  
  
    emit createVotingLog(nextVoteId, msg.sender, block.timestamp);  
    nextVoteId++;  
}
```

Figure 19:CreateVoting

```

function joiningVoting(
    uint256 voteId,
    bytes32 _proof
) external override onlyVoter {
    require(block.timestamp < votes[voteId].joiningVotingDeadline,
        "Operation not permitted: Tender must be in the JOINING PHASE");

    require(voting_proof[voteId][msg.sender] == bytes32(0),
        "Voter has already joined this voting");

    voting_proof[voteId][msg.sender] = _proof;

    emit joiningVotingLog(voteId, msg.sender, _proof, block.timestamp);
}

```

Figure 20:JoiningVote

```

function revealingVoting(
    uint256 voteId,
    string memory _option,
    uint256 _salt
) external override onlyVoter {

    require(block.timestamp > votes[voteId].joiningVotingDeadline &&
        block.timestamp < votes[voteId].revealingVotingDeadline,
        "Revealing phase has ended or not started"
    );

    require(options_check[voteId][_option] == true, "Invalid option");
    require(!hasVoted[voteId][msg.sender], "Already revealed vote for this voting");

    require(voting_proof[voteId][msg.sender] == generateProof(_option, _salt, msg.sender),
        "Proofhash mismatch");

    votesPerOption[voteId][_option]++;
    votes[voteId].counts += 1;
    hasVoted[voteId][msg.sender] = true;

    emit revealingVotingLog(voteId, msg.sender, _option, block.timestamp);
}

```

Figure 21:Revealing

```

function generateProof(string memory _option, uint256 _salt, address _voter) internal pure returns (bytes32) {
    return sha256(abi.encodePacked(_option, _salt, _voter));
}

function testGenerateProof(string memory _option, uint256 _salt, address _voter) public pure returns (bytes32) {
    return generateProof(_option, _salt, _voter);
}

```

Figure 22:GenerateProof

5.4 System Testing

```
describe("Testing userRegister function", async () => {
  it("should register a new user", async () => {
    const tx = await SmartVoterInstance.userRegister(ROLE.VOTER, { from: voter1 });
    const block = await web3.eth.getBlock(tx.receipt.blockNumber);

    const user = await SmartVoterInstance.users(voter1);

    assert(user.wallet === voter1);
    assert(parseInt(user.role) === ROLE.VOTER);
    assert(parseInt(user.status) === STATUS.UNVERIFIED);

    expectEvent(tx, "UserLog", {
      userWallet: voter1,
      role: toBN(ROLE.VOTER),
      status: toBN(STATUS.UNVERIFIED),
      timestamp: toBN(block.timestamp),
    });
  });

  it("should revert if user is already registered", async () => {
    await SmartVoterInstance.userRegister(ROLE.VOTER, { from: voter1 });

    await expectRevert(
      SmartVoterInstance.userRegister(ROLE.VOTER, { from: voter1 }),
      "User already registered"
    );
  });
});
```

Figure 23:TestForUserRegister


```

describe("Testing userApprove function", async () => {
  beforeEach(async () => {
    await SmartVoterInstance.userRegister(ROLE.VOTER, { from: voter1 });
  });

  it("should approve a registered user", async () => {
    const tx = await SmartVoterInstance.userApprove(voter1, { from: administrator });
    const block = await web3.eth.getBlock(tx.receipt.blockNumber);

    const user = await SmartVoterInstance.users(voter1);
    assert(parseInt(user.status) === STATUS.VERIFIED, "User status should be VERIFIED");

    expectEvent(tx, "UserLog", {
      userWallet: voter1,
      role: toBN(ROLE.VOTER),
      status: toBN(STATUS.VERIFIED),
      timestamp: toBN(block.timestamp),
    });
  });

  it("should revert if user is not registred", async () => {
    await expectRevert(
      SmartVoterInstance.userApprove(voter2, {
        from: administrator,
      }),
      "User not awaiting approval"
    );
  });
});

```

Figure 24: TestForApprove

```

describe("Testing createVoting function", async () => {
  const voteId = 100;
  beforeEach(async () => {
    await SmartVoterInstance.userRegister( ROLE.VOTING_COORDINATOR, { from: coordinator1 });
    await SmartVoterInstance.userApprove(coordinator1, { from: administrator });

    await SmartVoterInstance.userRegister(ROLE.VOTER, { from: voter1 });
    await SmartVoterInstance.userApprove(voter1, { from: administrator });
  });

  it("should create a new voting", async () => {
    //const joiningDurationMinutes = 1440;
    //const revealingDurationMinutes = 2880;

    const tx = await SmartVoterInstance.createVoting(
      title,
      description,
      options,
      joiningDurationMinutes,
      revealingDurationMinutes,
      {
        from: coordinator1,
        value: FEE,
      }
    );
    const block = await web3.eth.getBlock(tx.receipt.blockNumber);
    const vote = await SmartVoterInstance.getVote(voteId);

    const joiningDeadline = parseInt(vote.joiningVotingDeadline);
    const revealingDeadline = parseInt(vote.revealingVotingDeadline);

    assert.strictEqual(joiningDeadline, block.timestamp + joiningDurationMinutes * 60);
    assert.strictEqual(revealingDeadline, block.timestamp + (joiningDurationMinutes + revealingDurationMinutes) * 60);

    expectEvent(tx, "createVotingLog", {
      voteId: toBN(voteId),
      coordinator: coordinator1,
      timestamp: toBN(block.timestamp),
    });
  });
});

```

Figure 25: CreateTest

```

Contract: SmartVoter
Smart Contract Deployment
  ✓ should set administrator
Testing userRegister function
  ✓ should register a new user (74ms)
  ✓ should revert if user is already registered (200ms)
Testing userApprove function
  ✓ should approve a registered user (51ms)
  ✓ should revert if user is not registered
  ✓ should revert if caller is not administrator (56ms)
Testing createVoting function
  ✓ should create a new voting (109ms)
  ✓ should fail when a non-coordinator tries to create a vote
  ✓ should fail when the voting fee is incorrect
  ✓ should fail if less than two options are provided
Testing joiningVoting function
  ✓ should allow a verified voter to join voting (61ms)
  ✓ should revert if voting is not in the joining phase
  ✓ should revert if voter tries to join the same voting more than once (49ms)
  ✓ should fail when a non-voter tries to joiningVoting
Testing revealingVoting function
  ✓ should allow a voter to reveal their vote (81ms)
  ✓ should revert if voting is not in the revealing phase (86ms)
  ✓ should revert when trying to reveal a vote with an invalid option
  ✓ should revert if a voter tries to reveal their vote more than once (86ms)
  ✓ should revert if the voter tries to reveal with an incorrect proof

```

Figure 26:Result

```

Testing terminatingVoting function
  ✓ should allow a coordinator to terminate a voting successfully (74ms)
  ✓ should revert if attempting to terminate voting before the revealing phase ends
Testing getVote function
  ✓ should successfully retrieve vote details for a valid voteId
  ✓ should revert when trying to retrieve details for a non-existent voteId
Testing getUser function
  ✓ should retrieve a registered user
  ✓ should fail to retrieve a non-registered user
Testing getVotingResults function
  ✓ should retrieve voting results correctly
  ✓ should fail when trying to retrieve results for a non-existent voteId
Testing getVotes function
  ✓ should retrieve all votes correctly (528ms)

28 passing (12s)

PS C:\Users\hp\Desktop\FrontEnd\smartvoter-app>

```

Figure 27:Result2

5.5 Summary

In Chapter 5, we have successfully implemented a secure voting system on the blockchain, transitioning from theoretical design to practical application. The development process utilized essential tools like Solidity, Ganache, and Truffle, and was complemented by

React for the frontend, ensuring a robust and user friendly interface. Through rigorous testing phases unit, integration, and system testing we verified the reliability, security, and usability of the system. Although the system meets many of the initial objectives, areas such as scalability, enhanced security measures, and user experience require further improvement to fully harness the potential of blockchain technology in electoral processes.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This project successfully designed and analyzed a blockchain-based secure voting system, SmartVoter, which addresses the inherent challenges in traditional voting processes, such as security vulnerabilities, lack of transparency, and potential for fraud. By integrating blockchain technology, this system ensures that every vote is securely recorded and remains tamper-proof, providing an immutable ledger of transactions that enhances transparency and voter trust. Throughout the project, we conducted an extensive literature review to understand current practices and technologies, followed by a detailed system analysis that helped define functional and non-functional requirements. The design phase introduced a robust architecture encompassing user roles, voting sessions, and secure transactions, all facilitated by smart contracts. The proposed system utilizes Ethereum blockchain technology to manage the voting process transparently and securely, ensuring that all participants voters, voting coordinators, and system administrators engage in a straightforward and verified manner. This project's use of smart contracts and decentralized applications (DApps) guarantees that the voting process is autonomous and resistant to common electoral malpractices. In conclusion, the SmartVoter project exemplifies how blockchain technology can revolutionize traditional systems making voting not only more secure and transparent but also accessible to a broader population. This project sets a foundation for future advancements in blockchain applications within the voting sector and paves the way for further academic and practical exploration.

6.2 Goals Achieved

During this project, we successfully met the majority of our initial objectives for the initial phases, focusing on system analysis and design. Additionally, we accomplished personal learning objectives by gaining a deep understanding of decentralized networks and blockchain technology, including its operational mechanisms and the prerequisites for developing blockchain-based projects. Here's an alternative summary of our achievements:

- **Development of an Integrated Voting System:** Successfully designed and analyzed the SmartVoter system based on blockchain technology to provide a secure and transparent solution for electoral processes.
- **Enhancement of Security and Transparency:** The system ensures that votes are securely recorded and immutable, enhancing transparency and building trust in the election results.
- **Utilization of Blockchain Technology:** Leveraged blockchain features such as decentralization and self-auditing to enhance integrity and credibility in the voting process.
- **Implementation of Smart Contracts:** Developed and integrated smart contracts to automate the voting process, ensuring that operations are tamper-proof and transparent.
- **User-Friendly Interface Design:** Created an accessible and intuitive interface that facilitates easy interaction for all users, thus promoting higher participation rates.
- **Scalable and Adaptable Architecture:** Established a system architecture that can easily be scaled and adapted for various types of elections, from local to national.
- **Systematic Analysis and Design:** Conducted a thorough system analysis and crafted detailed design documents that serve as a robust foundation for future development phases.

6.3 Limitations and Future Work

We discuss the challenges faced during the project and propose directions for future research and development. This section serves as a reflection on the constraints we encountered and how they may guide the evolution of the SmartVoter system. Here's a detailed breakdown:

Limitations:

1. **Scalability Concerns:** While the system is designed to be scalable, the actual deployment on a large scale, such as national elections, might present performance

- issues due to the inherent limitations of current blockchain technology such as transaction speed and block size limits.
2. **Complexity of User Interface:** Despite efforts to make the user interface intuitive, the inherent complexities of blockchain technology may still pose usability challenges for less tech-savvy users.
 3. **Regulatory and Legal Challenges:** As with any blockchain application, there are ongoing uncertainties regarding regulation and legal acceptance, particularly in the context of sensitive applications like voting.
 4. **Security Risks:** Although blockchain enhances security, the system may still be susceptible to novel cyber-attacks that could target smart contracts or other aspects of the infrastructure.

Future Work:

1. **Integration with Nafad:** We plan to integrate our voting platform with the Saudi National Single Sign-On (SSO) system, Nafath, to streamline user authentication and enhance security.
2. **Mobile Application:** We intend to create a mobile application that offers a user-friendly interface and enhances accessibility for all users.
3. **Backend Development:** We aim to develop a robust backend system to support efficient data processing and management.
4. **Enhancing Scalability:** Future efforts could focus on integrating advanced blockchain solutions such as sharding or layer-two protocols to handle larger transaction volumes and reduce latency, making the system more suitable for large-scale elections.
5. **User Interface Simplification:** Continued research and testing on the user interface to make it more accessible to all voter demographics, potentially incorporating more interactive and guided user experiences.
6. **Addressing Regulatory Compliance:** Engaging with legal experts and regulators to ensure that the voting system complies with national and international election laws and standards, adapting to changes in blockchain regulation.

7. **Strengthening Security Measures:** Ongoing security assessments and updates to safeguard against emerging cyber threats, including regular audits of smart contracts and the implementation of enhanced cryptographic measures.

By addressing these limitations and focusing on these areas of future work, the project can evolve to meet the rigorous demands of modern electoral systems, ensuring that it remains at the forefront of technological advancements in secure and transparent voting solutions.

References

- [1] I. Sommerville, "Software Engineering", 11th ed., Addison-Wesley, 2015.
- [2] V. J. Blue, and J. L. Adler, "Cellular automata micro-simulation of bi-directional pedestrian flows," J. Transportation Research, pp. 135-141, 2000.
- [3] S. Sarmady, F. Haron, and A. Z. H. Talib, "Modelling groups of pedestrians in least effort crowd movements using cellular automata," in Proc. 2009 2nd Asia International Conference on Modelling & Simulation, Bali, Indonesia, 2009, pp. 520-525.
- [4] F. H. Hassan, "Heuristic search methods and cellular automata modeling for layout design," Ph.D dissertation, Sch. of Info. Sys, Comp. and Math., Brunel Univ., UK, 2013.
- [5] G. K. Still. (2010, July 15). Crowd Disasters [Online]. Available: <http://www.gkstill.com/CrowdDisasters.html>.
- [6] Roos, D. (2020, April 13). How Americans Have Voted Through History: From Voices to Screens. HISTORY. Retrieved November 30, 2023, from <https://www.history.com/news/voting-elections-ballots-electronic>.
- [7] Brookings. (n.d.). How blockchain could improve election transparency. Retrieved November 30, 2023, from <https://www.brookings.edu/articles/how-blockchain-could-improve-election-transparency/>.
- [8] Halderman, J. A., & Teague, V. (2015). The New South Wales iVote system: Security failures and verification flaws in a live online election. In E-Voting and Identity: 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings 5 (pp. 35-53). Springer International Publishing.

- [9] Dilley, J., Poelstra, A., Wilkins, J., Piekarska, M., Gorlick, B., & Friedenbach, M. (2016). Strong federations: An interoperable blockchain solution to centralized third-party risks. arXiv preprint arXiv:1612.05491.
- [10] Brady, H. E., & McNulty, J. E. (2011). Turning out to vote: The costs of finding and getting to the polling place. *American Political Science Review*, 105(1), 115-134.
- [11] Chaganti, R., Bhushan, B., & Ravi, V. (2023). A survey on Blockchain solutions in DDoS attacks mitigation: Techniques, open challenges, and future directions. *Computer Communications*, 197, 96-112.
- [12] Farjam, M. (2021). The bandwagon effect in an online voting experiment with real political organizations. *International Journal of Public Opinion Research*, 33(2), 412-421.
- [13] Vote, F.M. The Secure Mobile Voting Platform of The Future—Follow My Vote. 2020. Available online: <https://followmyvote.com/>.
- [14] Voatz. Voatz—Voting Redefined ®®. 2020. Available online: <https://voatz.com> .
- [15] Agora. Agora. 2020. Available online: <https://www.agora.vote>.
- [16] Polyas. Polyas. 2015. Available online : <https://www.polyas.com>.
- [17] Ali, N., & Lai, R. (2017). A method of software requirements specification and validation for global software development. *Requirements Engineering*, 22, 191-214.
- [18] The Product Manager. (n.d.). The Software Development Life Cycle (SDLC): 7 Phases and 5 Models. Retrieved from <https://theproductmanager.com/topics/software-development-life-cycle/>.

[19] Kuhrmann, M., Tell, P., Hebig, R., Klünder, J., Münch, J., Linssen, O., ... & Richardson, I. (2021). What makes agile software development agile?. *IEEE transactions on software engineering*, 48(9), 3523-3539.

[20] Laurence, T. (2019). Introduction to blockchain technology. Van Haren.

[21] Blockchain Expert, “Blockchain Book - Blockchain Experts,” n.d.,
<https://www.blockchainexpert.uk/blockchain-book/>.