Blockchain Voting System

Robert-Andrei Anghelina

SUMMARY

The developed project represents a decentralized voting system, built on the Ethereum network, using blockchain technology. It employs a “smart contract”, written in the Solidity language, to manage the voting process and to store the voting-related data. The contract holds functions for adding candidates, voters, changing the contract’s administrator, and many others.

The User Interface (UI) of the voting system is created with the help of HTML, CSS, and JavaScript. Additionally, “Web3.js” library makes the interaction with the blockchain possible. The interface is used for connecting the virtual wallet and for various interactions with the contract functions.

Furthermore, the interface includes dynamic updates to reflect the current status of the voting system, as well as notifications for various contract functions, providing comprehensive feedback. For user comfort, a dark mode for page viewing has been implemented, as well as a developed system for managing Ethereum transactions and errors.

Personally, building this voting system was a challenging experience, being one of the most interesting applications I have developed. I encountered a multitude of problems, as this type of programming is not very well-known, given that Solidity released the first stable version of the language in 2018. One of these challenges was understanding the complexity of the Solidity language and the “Ethereum Virtual Machine” (EVM), which represents the execution engine and the logic of all contracts built on the Ethereum network.

I could say that the most interesting challenge offered by this project is understanding the security principles on the blockchain, as I had to ensure that the voting system cannot be manipulated and that its execution is correct.

In conclusion, the presented project is a decentralized application (DApp), built on the Ethereum blockchain, demonstrating various aspects of blockchain programming, such as interactions with it, smart contract programming, and UIs specifically built for decentralized applications.

Keywords: Blockchain, Ethereum, Smart Contract, Solidity, Voting System, Decentralized Application, Web3.js, User Interface (UI), Wallet Connection, Candidate Management, Whitelisting, Ownership Transfer, Voting Statistics, Dark Mode UI, Leaderboard, Ethereum Address Validation, Transaction Handling, Error Handling, Notifications, Pagination, CSS Theming, Contract Method Calls, Event Listeners, Dynamic Content Generation, Ethereum Network Interaction, Voting Round Management, Ethereum Address Management, Contract Ownership, Ethereum Testnet (Goerli)

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 The Scope

1.1.1 Overview

This project's objective is to create a fully operational voting system using the blockchain technology. The system utilizes an Ethereum smart contract to facilitate a secure and transparent voting procedure. The system's administration is handled by the owner, responsible for managing both candidates and voters. Whitelisted addresses are permitted to vote for candidates, assuring transparency and security. Every vote cast is then logged on the Ethereum blockchain.

1.1.2 Objectives

- Secure Voting Process: The project provides a secure environment for voting by utilizing an Ethereum smart contract, where each vote is recorded on the blockchain. This ensures the transparency of each vote.

- Role-based Interaction: The system accommodates three types of users - the owner, whitelisted voters, and the public. Candidates are public users, with the exception that they are eligible to be voted for in elections. Each user type has different access rights and available functions.

- Dynamic Voting Administration: The owner of the contract can perform various administrative tasks such as adding/removing candidates, starting/ending the voting process, adding/removing whitelisted addresses, and transferring the ownership of the contract.

- Real-time Updates and Insights: Users can access real-time information on voting progress and results. The system also features a leaderboard that updates dynamically.

- User-friendly Interface: The system is accessed via a website that is designed to be interactive and intuitive.

1.1.3 Expected Outcomes

After the project completion, the following outcomes are expected:

- A smart contract deployed on the Ethereum network, representing the logic of the voting system.

- A website that interacts with the smart contract, allowing users to participate in the voting process.
- A complete documentation detailing the steps taken in building the project.

1.1.4 Scope Limitations

While the project aims to deliver a fully functional blockchain-based voting system, the following limitations should be noted:

- The system is dependent on the Ethereum network, its performance, and the associated gas fees.
- User identity verification is not part of this project's scope. The project assumes that user authentication and eligibility verification are handled before whitelisting addresses.
- The front-end interface is designed for desktop platforms. It is not optimized for smaller screens, such as smartphones.
- The project's infrastructure is designed to work with a single owner who administers the voting process. In case of multiple owners, like a committee, a new and modified contract must be deployed.
- The system does not support automatic tallying of votes or declaration of results. The owner must manually end the voting process, and users must manually check for winners.
- While the project uses Ethereum smart contracts, it does not explore other blockchain platforms or technologies. Other platforms may offer different features or advantages which are not covered by this project.

1.1.5 Scope Conclusion

Overall, the goal of this project is to create a safe and transparent voting system, highlighting the potential of decentralized applications on the Ethereum blockchain.

1.2 The Motivation

The motivation behind choosing this project arises from my passion for cryptocurrency, which began in 2016. Recently, I started viewing this passion from a programming perspective, seizing the opportunity that comes with it.

Over the past year, I have been involved in several Ethereum-related personal projects, primarily focusing on JavaScript. This experience raised my interest in Solidity and smart contract programming.

The opportunity to select a project within this theme for my diploma was something I greatly appreciated. Furthermore, I intend to build a career in Web3 soon, making this project not only a learning experience but also a stepping stone towards my career goals.

2 STATE OF THE ART

2.1 Usage Context

Voting systems, a cornerstone of democracies, have been existent for hundreds of years. But these conventional systems are flawed due to problems like insufficient transparency, vulnerability to manipulation, and logistical hurdles. Lately, though, the emergence of blockchain technology holds the key to overcoming these challenges.

With a voting system in mind, blockchain technology can craft a clear and secure space for votes to be cast and logged. Think of each vote like a transaction that is recorded on the blockchain. Once it is logged, this information cannot be changed or erased, which guarantees the trustworthiness of the voting procedure.

2.2 An Introduction to Blockchain Technologies

2.2.1 Cryptocurrency

Cryptocurrencies are digital/virtual currencies that use cryptography for security. Compared to conventional currencies issued by central entities such as a bank, cryptocurrencies operate on a database of transactions called “Blockchain”.

Bitcoin was introduced in 2009 by an unidentified person to address various issues that arise from having a central authority like a bank or government in control. These issues encompass concerns regarding trust, privacy, and the extensive control such authorities exert over monetary resources.

2.2.2 Ethereum

Ethereum is an open-source cryptocurrency created by Vitalik Buterin in 2015. The main feature of this cryptocurrency is the new concept of “smart contract”. Smart contracts are self-executable interactive codes that run on the blockchain, such as the presented voting system.

Ethereum is built on its own blockchain, using as native currency “Ether” (ETH). Ether is used as a monetary value and to reward the validators that help facilitate transactions by confirming them, as well as create new blocks on the blockchain.

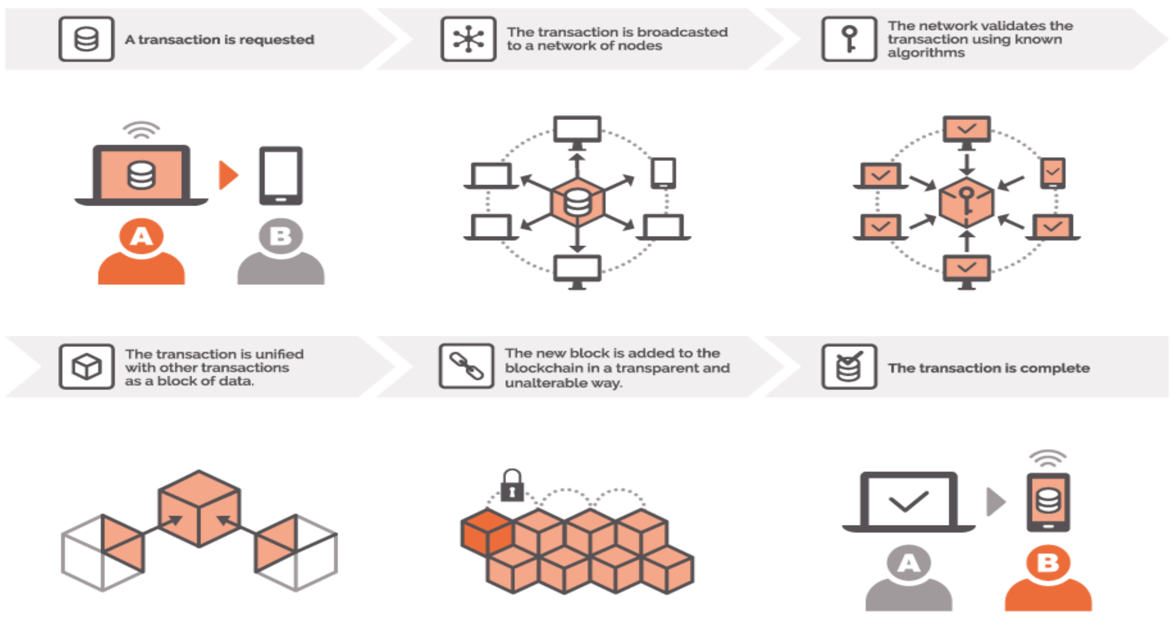
2.3 Blockchain Technology

A blockchain is a decentralized and distributed digital ledger that records transactions across multiple computers in such a way that any involved record cannot be altered, without the alteration of all subsequent blocks. It is similar to a database of transactions. This allows the participants to verify and audit transactions without a centralized authority.

Each time a transaction occurs, it is put into a ‘block’. This block is then added to a chain of similar blocks, being linked together in chronological order. Due to its decentralized nature, it provides a level of transparency and security which is particularly useful for maintaining integrity in systems like voting.

The presented Voting System is an example of an active area of research and development regarding the application of blockchain technology in voting systems. Specifically, within the context of Ethereum technology and the Web3.js library. This project takes advantage of the smart contract functionality present in the Ethereum network via a web page interface. The voting process is managed by a smart contract that leverages the existing proposal-sorting algorithm, which is written in Solidity.

Finally, it interacts with the Ethereum network carried out through the Web3.js library from where it receives information about voters (through node’s ‘getch’ function), establishes events to trigger transaction at each stage, and prioritizes transaction processing similarly to other programming languages.



A transaction is requested
The transaction is broadcasted to a network of nodes
The network validates the transaction using known algorithms
A
B
The transaction is unified with other transactions as a block of data.
The new block is added to the blockchain in a transparent and unalterable way.
The transaction is complete
A
B

Figure 1: Understanding How Blockchain Works (Kilroy, n.d.)

2.4 Smart Contracts

A smart contract is essentially a script that operates on the Ethereum network. It automatically executes transactions when certain pre-set conditions are met. It is “smart” because once it is set up, it can operate and enforce itself automatically.

In essence, it can be explained as a digital protocol that facilitates, verifies, and enforces the negotiation and performance of a contract. It allows trusted transactions to take place without the need for a centralized authority, legal system, or external mechanism.

In a voting system context, the smart contract is programmed to function like this: IF a valid voter submits their vote, THEN that vote is automatically and securely recorded. The smart contract would ensure the vote is from a valid voter, and that each voter only votes once.

A smart contract can be perceived as transparent, verifiable, and immutable. The contract rules are visible to all, and once the contract gets executed, it remains unalterable. These features prove themselves useful in a voting system, where maintaining integrity and public trust is a priority.

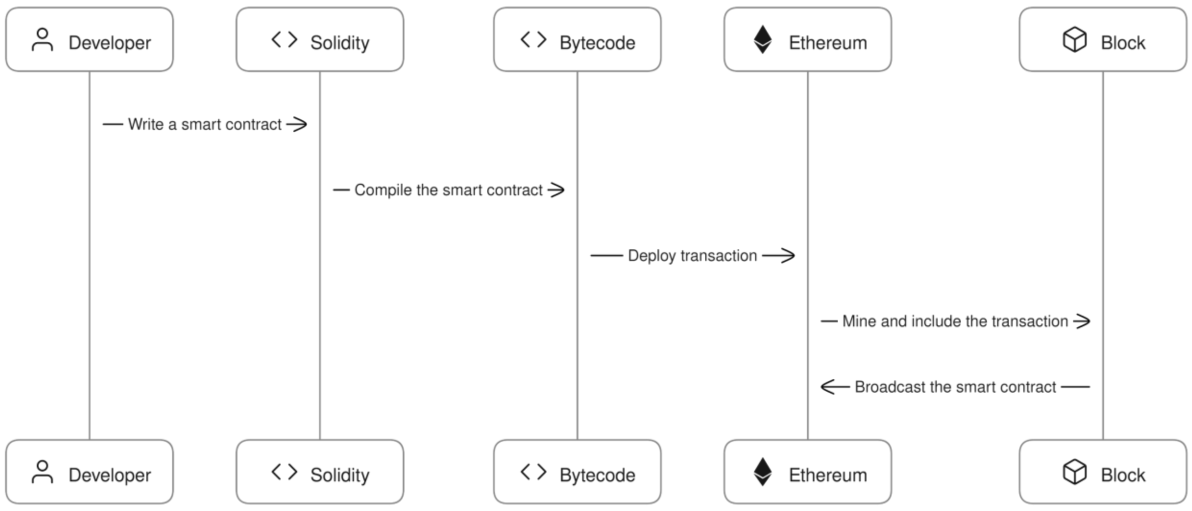


Figure 2: Deployment of a Smart Contract on the Ethereum Blockchain

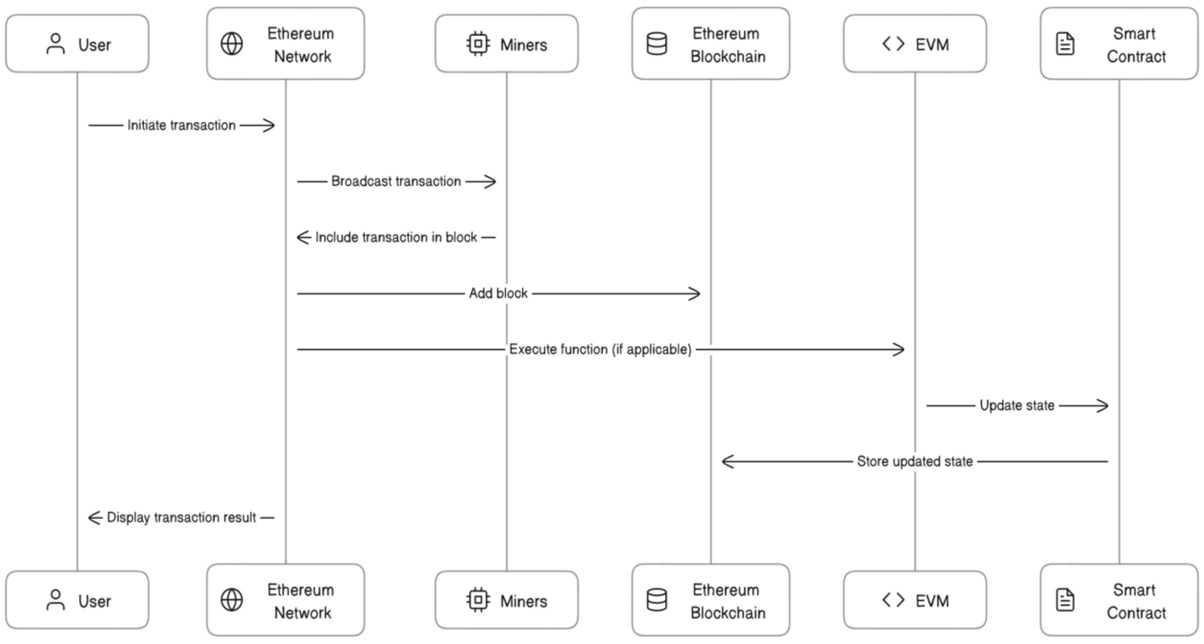


Figure 3: Interaction with a Smart Contract on the Ethereum Blockchain

3 SYSTEM AND REQUIREMENTS ANALYSIS

3.1 Electronic Voting System

The evolution of electronic voting systems marks a significant shift in the arena of electoral votes. These systems seek to make voting simpler, boost the numbers at the ballot box, and increase the safety and authenticity of elections.

A major advantage of electronic voting systems is their feature of automating the voting procedure. They quickly calculate the total number of votes, which significantly trims the wait time for election results.

Nonetheless, traditional electronic voting systems have been subjected to a wave of criticism and skepticism, mainly due to the concerns surrounding their security and openness. Often, these systems lean heavily on external software, which is closed off from public review, fueling concerns of potential manipulation of the vote count.

The voting system, developed within this project, introduces a fresh era of electronic voting systems. It cleverly utilizes blockchain technology to solve issues that are found in traditional electronic voting systems. By logging votes on the blockchain, the system offers a clear, unchangeable ledger of votes, accessible for review by anyone. This removes the concerns of secrecy, focusing on transparency and trust in the voting procedure.

Moreover, the system employs the smart contract to automate the voting process. Open to public visibility, these contracts oversee the voting procedure, making sure it is conducted in a fair and open manner. This innovation represents a remarkable leap in the creation of secure, transparent, and efficient electronic voting systems.

3.2 System Goals

The key goals of the voting system are:

- Security: The application should ensure that votes are secure and tamper-proof.
- Transparency: All actions, including voting and vote tallying, should be transparent and verifiable.
- Flexibility: The system should allow the contract owner to manage candidates and whitelisted voters.

- User Interactivity: The system should provide an interactive and intuitive interface for users.

3.3 Functional Requirements

For a better understanding of the interactions with the DApp, the functional requirements are highlighted using screenshots and detailed explanations in the User Manual (Chapter 7).

3.3.1 Wallet Connection

On connection, the application should identify the type of the user: owner, whitelisted, or public address. The interface presented should correspond to the user's role.

3.3.2 Owner Interactions

The owner should be able to perform various actions including starting and ending voting, adding or removing candidates, adding or removing whitelisted addresses, transferring contract ownership, and removing all candidates or whitelisted addresses.

3.3.3 Whitelisted Interactions

Whitelisted users should be able to vote for their chosen candidates when voting is open.

3.3.4 Public Address / Visitor Interactions

Public addresses should be able to check if an address is a candidate or whitelisted, and view the winner(s) and leaderboard.

3.3.5 Voting Flow

The voting process should follow a clearly defined flow from start to end, with transparency at all stages.

3.3.6 Leaderboard and Winners

Any address should be able to view the leaderboard and winners, but the winners should only be available after voting ends. In the case of the owner, the removal of candidates should be possible by using the leaderboard function when the voting is closed. In the case of whitelisted addresses, they should be able to vote from the leaderboard when the voting is open.

3.3.7 Theme Switching

The system should allow the user to switch between light and dark themes.

3.4 Non-Functional Requirements

3.4.1 Usability

Usability is a critical aspect of the voting system, ensuring that users can easily understand and interact with the application. A few examples are:

- User-friendly Interface: The user interface should be instinctive, featuring unambiguous navigation and pleasing controls. Users should easily perform actions and access necessary information.
- Accessibility: Keeping different user abilities in mind, the system should adhere to accessibility standards.
- Error Management: In case of incorrect inputs or errors, the system should provide users with understandable error messages and guidance.

3.4.2 Performance

High-performance is key to the system’s ability to efficiently handle a considerable number of voters. Non-functional requirements related to performance might include:

- Scalability: Regardless of user volume, the system should sustain a good performance level.
- Responsiveness: To deliver a seamless user experience, the system should rapidly react to user interactions.
- Optimized Gas Usage: The smart contract code should be optimized to be efficient and reduce transaction costs.

3.4.3 Compatibility

Compatibility ensures that the voting system can seamlessly integrate with various Ethereum wallets and accommodate a wide range of users. The non-functional requirements related to compatibility may include:

- Integration with Wallets: The system should align seamlessly with widely-used Ethereum wallets such as MetaMask and Trust Wallet.
- Compatibility Across Platforms: The user interface should harmonize with a variety of web browsers and devices.

3.4.4 Security

Securing the authenticity and privacy of votes is a paramount aspect of the voting system. Hence, a primary focus of the voting system is its security measures, including:

- Unalterable Voting: The system should ensure that once votes are securely recorded on the Ethereum blockchain, they cannot be tampered with.
- Controlled Access: The system should protect user-specific functions from unauthorized access by other users.
- Data Privacy: User data and voting information should be fully private to the user casting the vote.
- Secure Communication: The system should utilize protocols designed to ensure the security of user interactions.

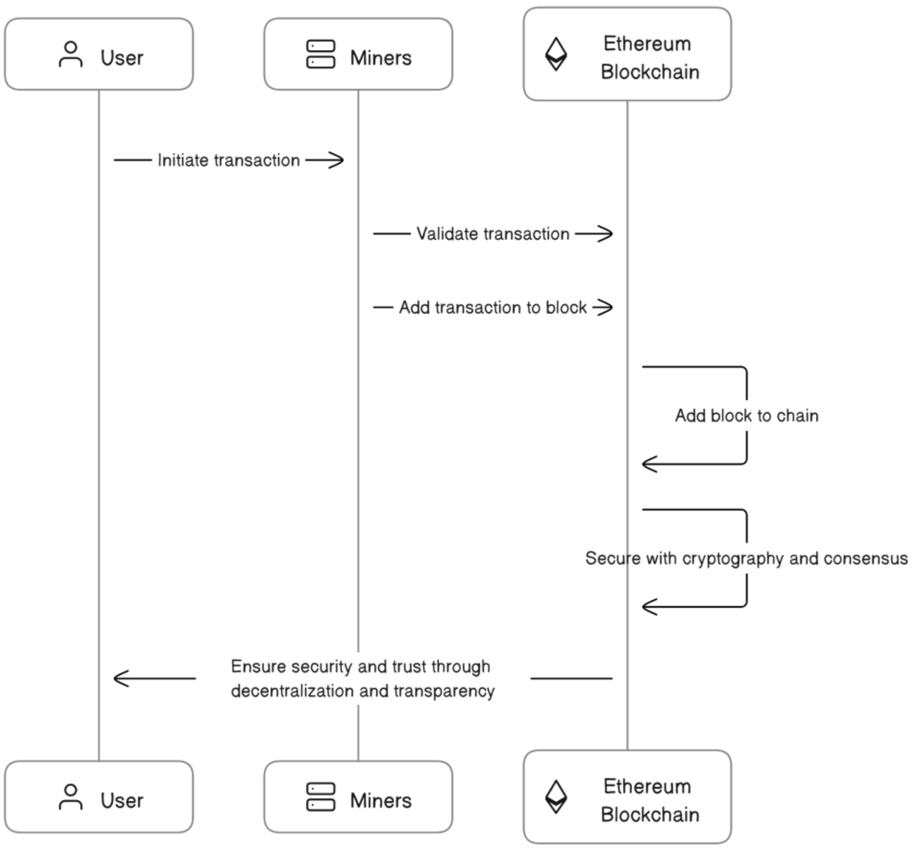


Figure 4: Diagram Showcasing the Security of the Ethereum Blockchain

3.5 Diagrams and Charts of the Voting System

Diagrams are an important tool in visualizing the functions and logic of the app, with the scope of making the viewer understand the way it works. I have created a multitude of diagrams using UML, Mermaid, and other diagram related tools.

3.5.1 User Class Diagrams

These User Class diagrams might be divided into three main classes: Owner, Whitelisted, and Public Addresses.

The Owner class is a special type of user who has access to multiple functions like starting or ending voting, adding/removing a candidate, adding/removing a whitelisted address, transferring ownership, and clearing all candidates or whitelisted addresses.

The Whitelisted class represents the users who are allowed to vote for the candidates while voting is open.

The Public class represents general users, visitors, who can check if an address is a candidate or whitelisted, and view the winner(s) and the leaderboard. They do not have any write permissions.

Relationships between these classes would be defined by their interactions with the 'VotingSystem' contract.

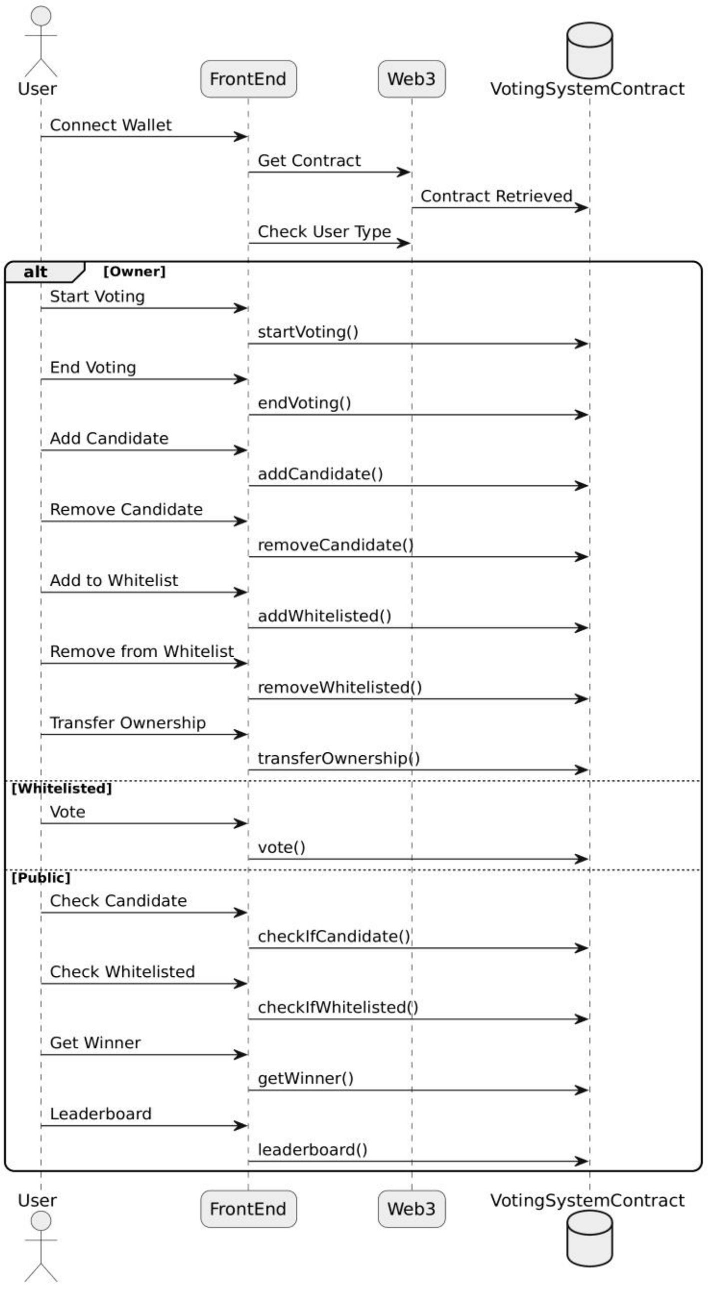


Figure 5: Simple User Class Diagram

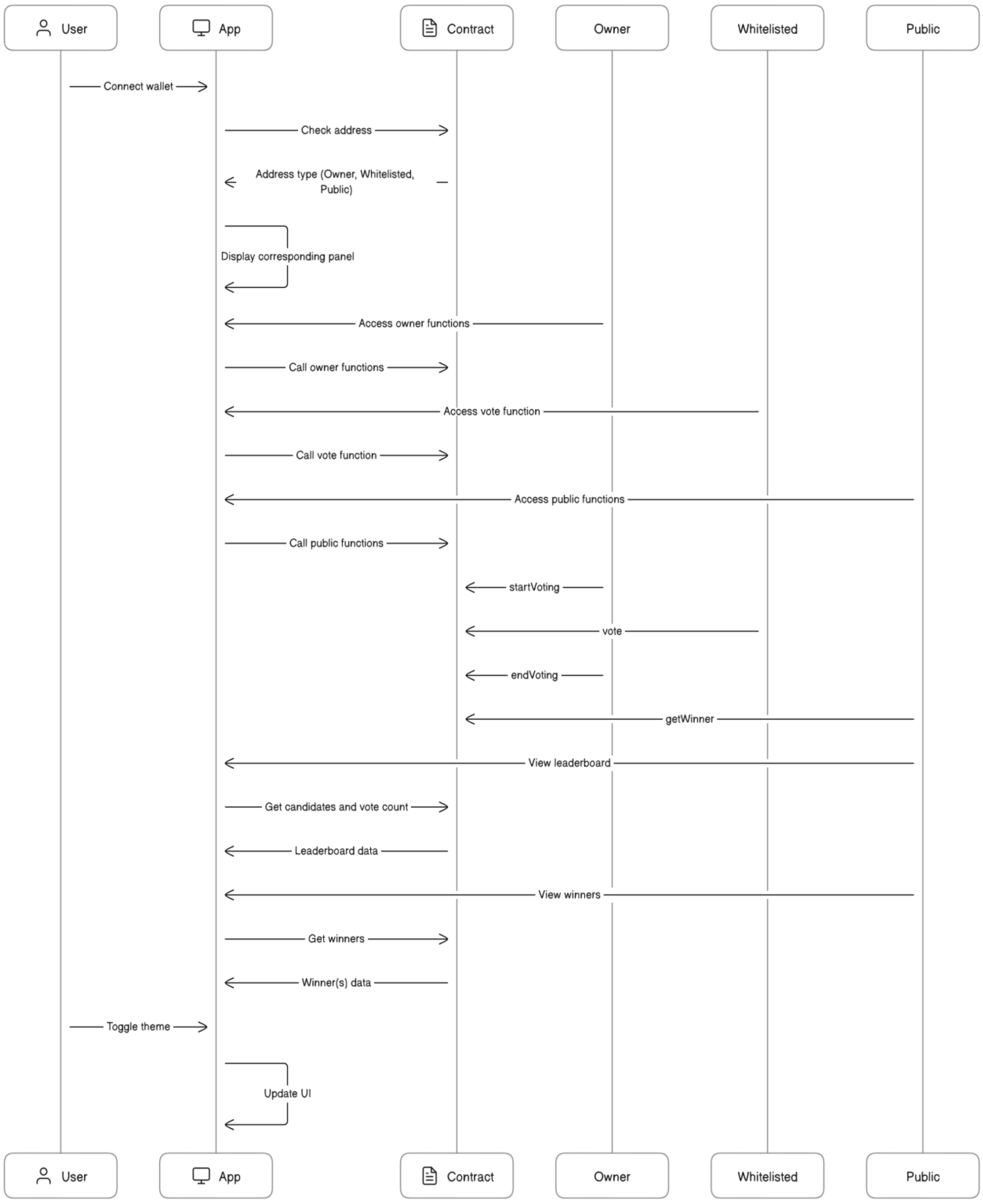


Figure 6: Detailed User Class Diagram

3.5.2 Voting Process Flowchart

The Voting Process Flowchart is a diagram that states the events that happen since the creation of the contract till the end of a voting round.

After the contract has been created, the owner adds the candidates and whitelists the addresses eligible for voting.

Then, the owner calls the 'startVoting' function. This process can only be initiated if voting is not currently open.

After voting starts, whitelisted addresses are allowed to vote for a candidate by calling the 'vote' function. This continues until the owner decides to call the 'endVoting' function to close the voting.

Once the voting has ended, the system would be in a state where no votes can be cast until the owner starts another voting process. After the voting ends, anyone can view the winner(s) by calling 'getWinner'.

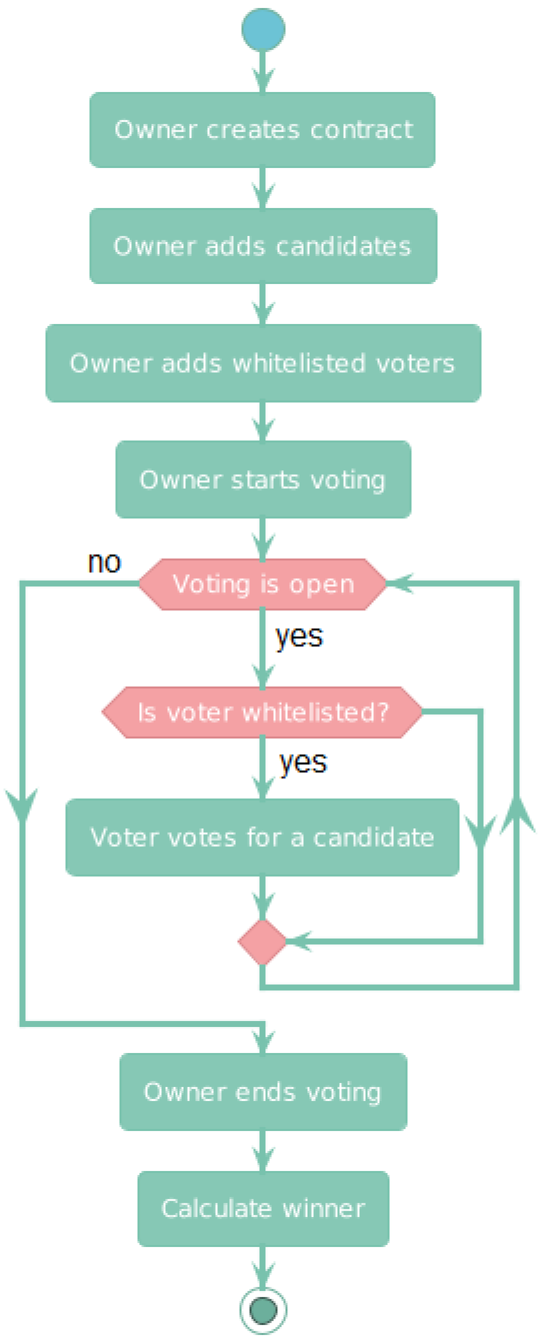


Figure 7: Voting Process Flowchart

3.5.3 Component Diagram

This diagram offers a high-level architectural view of the voting system, displaying its main components and their interactions.

The user connects the wallet to the website and receives the corresponding menu to the assigned user role. Then, the front-end interface interacts with the Solidity contract, through function calls. These function calls correspond to the various actions a user can perform, such as starting or ending voting, adding or removing candidates, and casting votes.

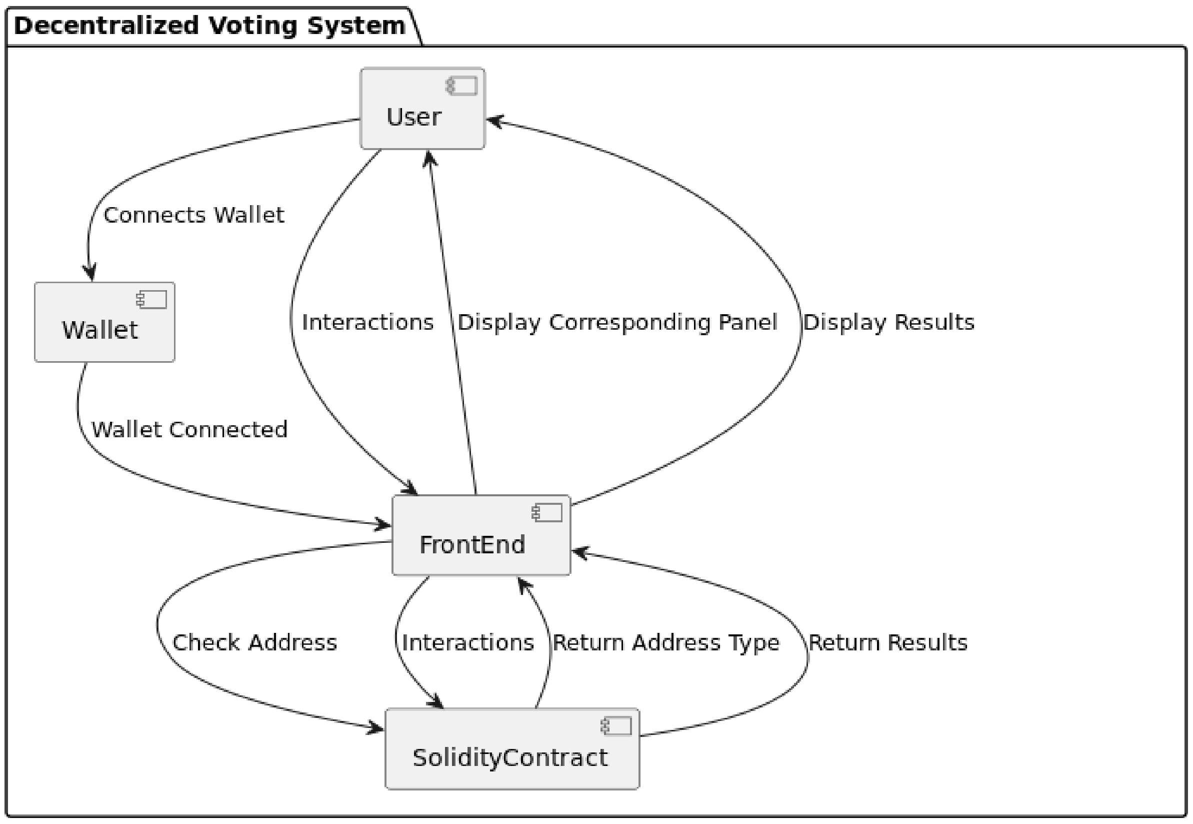


Figure 8: Component Diagram

3.5.4 Data Flow Diagram

This diagram visualizes how data moves through the system and the logic behind it.

In this diagram, the logic of a voting round is being shown in regards with the functions required to run a round. Such function calls alter the state of the contract, including actions like adjusting the voting status or modifying a candidate's vote count. The results of these operations are relayed back to the front-end interface and presented to the user.

The diagram effectively illustrates the flow of data in response to user actions and the system's reactive nature.

Moreover, the diagram accentuates the importance of access control mechanisms, imposed through the use of Solidity's 'modifier' constructs, in maintaining the integrity of the voting process. It also features the temporal aspect of voting, reflected through 'startVoting' and 'endVoting' functions, which act as gatekeepers for the voting period.

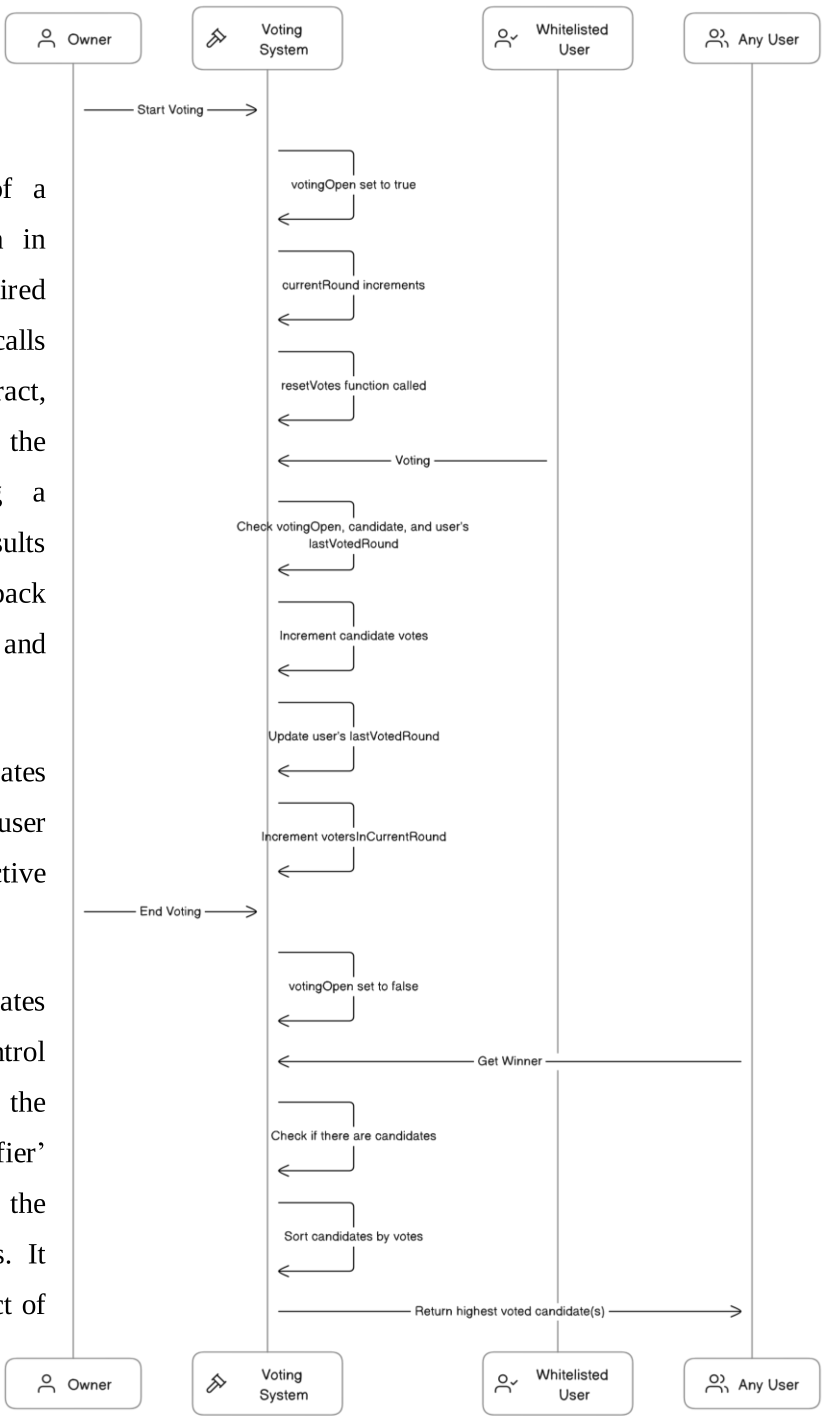

Owner
Voting System
Whitelisted User
Any User
Start Voting
votingOpen set to true
currentRound increments
resetVotes function called
Voting
Check votingOpen, candidate, and user's lastVotedRound
Increment candidate votes
Update user's lastVotedRound
Increment votersInCurrentRound
End Voting
votingOpen set to false
Get Winner
Check if there are candidates
Sort candidates by votes
Return highest voted candidate(s)

Figure 9: Data Flow Diagram

3.5.5 Sequence Diagrams

The first diagram displays the sequence of interactions between a user, the wallet, the UI, and the solidity contract.

The wallet is connected via the Voting System UI where it is verified within the contract in order to find the role of the user. The proper panel is displayed to the user and upon interacting with the functions available, a call is sent to the contract, and in return a result is being sent back. That result is displayed as feedback for the user.

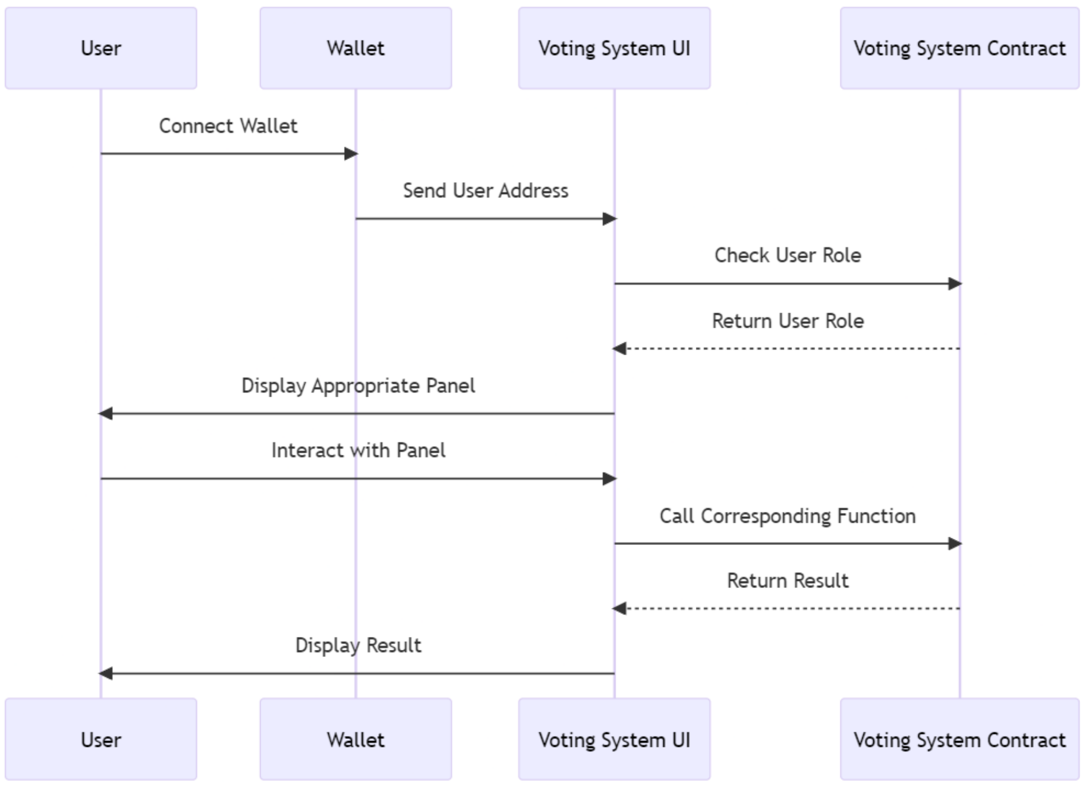
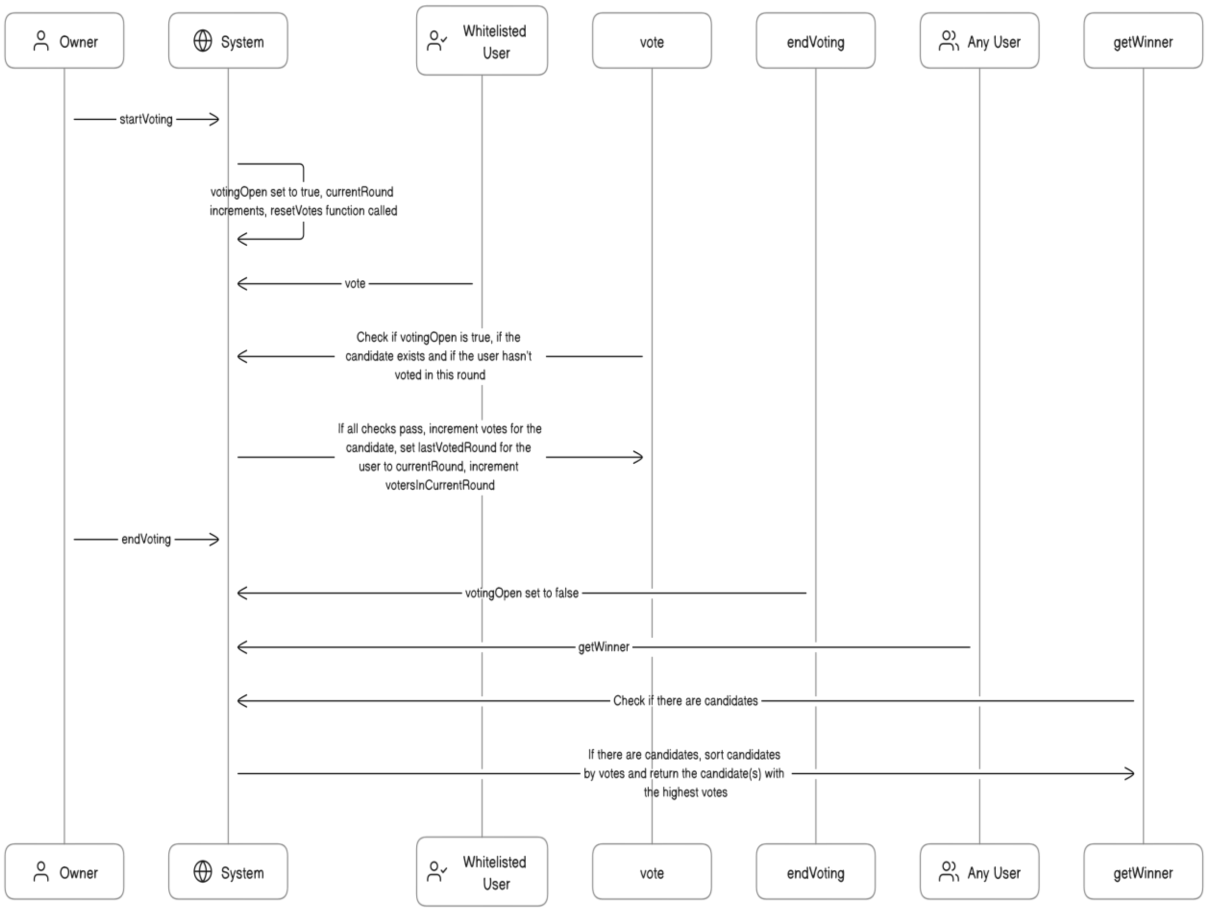


Figure 10: Sequence Diagram 1

The second diagram shows the interactions between the different user types and the Solidity contract during a typical voting process. It shows the chronological order of these interactions, starting with the Owner initiating the voting process. Following this, Whitelisted users cast their votes, each interaction marked by a separate sequence. The diagram also shows conditional checks, such as verifying if voting is open or if a user has already voted. The sequence concludes with the Owner ending the voting process and the display of voting results. This diagram provides a temporal perspective on the system's operations, highlighting the sequence and timing of interactions.



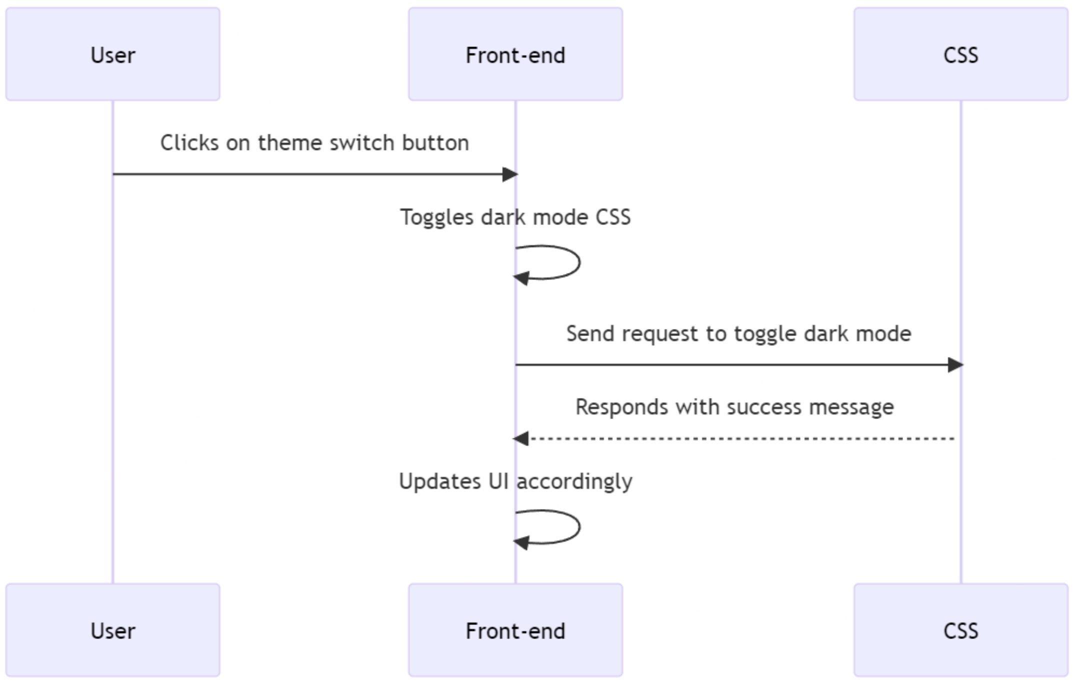
Owner
System
Whitelisted User
vote
endVoting
Any User
getWinner
startVoting
votingOpen set to true, currentRound increments, resetVotes function called
vote
Check if votingOpen is true, if the candidate exists and if the user hasn't voted in this round
If all checks pass, increment votes for the candidate, set lastVotedRound for the user to currentRound, increment votersInCurrentRound
endVoting
votingOpen set to false
getWinner
Check if there are candidates
If there are candidates, sort candidates by votes and return the candidate(s) with the highest votes
Owner
System
Whitelisted User
vote
endVoting
Any User
getWinner

Figure 11: Sequence Diagram 2

3.5.6 View Mode Diagram

The View Mode Diagram illustrates the functionality of switching between light and dark modes in the front-end interface.

The themes can be switched by clicking on the button situated in the top right corner. Upon clicking the button, a function is activating the corresponding CSS for the chosen mode, either enabling or disabling the dark mode CSS. The diagram further illustrates how this function promptly updates the UI to mirror the selected mode, providing a visual representation of the feature.


User
Front-end
CSS
Clicks on theme switch button
Toggles dark mode CSS
Send request to toggle dark mode
Responds with success message
Updates UI accordingly
User
Front-end
CSS

4 THE ARCHITECTURE

The architecture of this voting system is built around three major components: the Ethereum blockchain, the smart contract, and the web-based user interface.

4.1 Ethereum Blockchain: The Foundation

The Ethereum blockchain acts as the spine of the app. It is a decentralized platform where the smart contracts are functioning on, making sure the voting process is transparent and honest. Each vote cast is recorded as a transaction on the Ethereum blockchain via the contract. Once a vote is lodged, it becomes immutable, beyond alteration by any party, including the contract’s owner.

Ethereum is public, open-source and includes a decentralized virtual machine known as the Ethereum Virtual Machine (EVM). The EVM can run scripts using a global network of public nodes. In this project, the whole user-contract interaction is done via the EVM. This makes Ethereum an ideal foundation for a voting system that is both secure and transparent.

4.2 Smart Contract: The Core

The center of the voting system is the smart contract. It is written in Solidity and holds all the rules and logic for how voting happens. This contract has functions for adding or taking out candidates, whitelisting voters, starting, and stopping the vote, and counting votes. It can also check on how voting is going and display the results by using the implemented functions.

The smart contract is running on the Ethereum blockchain, and it is activated by transactions coming from the user interface. It is self-executing, given the user successfully calls the functions of the contract. These functions can be public values that do not require gas transactions, commonly known as “read” transactions, such as ‘votingOpen’, which returns ‘true’ if the voting is open, and ‘false’ if the voting is false, or transactions that do cost gas, commonly known as “write” transactions, such as ‘function vote(address candidate)’, which adds a vote to the candidate in the arguments of the function. This mechanic guarantees that the voting process follows the rules and cannot be tampered with.

4.3 Web-based User Interface: The Interaction Point

The user interface is represented by a website, built using HTML, CSS, and JavaScript. It is designed to be user friendly and intuitive. The interface enables users to connect their Ethereum wallet, view the candidate lineup, cast their votes, and examine the voting results.

The user interface uses the Web3.js library to connect with the Ethereum blockchain and trigger the functions of the smart contract. This library lets the app listen for events, run functions on the contract, and read contract data. The user interface also has dynamic updates and notifications to give users live updates on how voting is going.

5 IMPLEMENTATION

The implementation of the voting system is composed from the development of the smart contract, the creation of the user interface, and the integration of the two components.

5.1 Smart Contract Development

The initial stage of the implementation involves the creation of the smart contract, executed using the Solidity programming language.

The smart contract contains several functions and modifiers that define the rules and logic of the voting process. These include user dependent functions for managing and viewing the state and flow of the voting process, as well as retrieving the results once the voting has ended.

Once the smart contract is written, it is compiled into bytecode and deployed on the Ethereum test network blockchain. This is done using Remix IDE[1], an online tool for developing, testing, and deploying smart contracts on the Ethereum network.

5.2 User Interface Creation

The user interface, a website created with HTML, CSS, and JavaScript, offers an easy-to-navigate experience. It offers a host of features that streamline interaction with the smart contract, allowing users to link their Ethereum wallet, observe the candidate list, submit votes, and monitor the ongoing voting results among other functions. Real-time updates and alerts are also provided, offering immediate insights into the voting procedure’s current state.

Notably, the interface design is responsive, adapting seamlessly to various screen sizes for optimal viewing. A dark mode has been included to enhance user comfort during prolonged use. Additionally, it effectively manages transaction-related issues and errors, further ensuring a smooth user experience.

5.3 Integration of Smart Contract and User Interface

The integration of the smart contract and user interface is done using the Web3.js library. The web3.js library is implemented into the UI via an online library that contains the web3.min.js file.

[1] https://remix.ethereum.org/

5.4 Used Technologies

5.4.1 Solidity

Known for its design tailored towards crafting smart contracts on the Ethereum blockchain, Solidity is a statically-typed programming language. This choice of language for the smart contract's creation is attributed to its strong security features, extensive use within the Ethereum sphere, and its robustness. Solidity enables developers to design intricate contract functions and guarantees contract validity through built-in protective measures.

5.4.2 Ethereum Blockchain

The selection of the Ethereum blockchain as the platform for the voting system is justified by its substantial community support and its compatibility with smart contracts. It offers a transparent and secure platform for the voting process, ensuring votes are protected against tampering and are verifiable.

5.4.3 Web3.js

Web3.js, a suite of libraries, allows communication with Ethereum nodes, either locally or remotely, via HTTP, IPC, or WebSocket. The decision to use Web3.js stems from its capability to enable interaction between the user interface and the Ethereum blockchain. It provides the user interface the ability to trigger contract functions, monitor events, and access contract data.

5.4.4 HTML, CSS, and JavaScript

HTML, CSS, and JavaScript were used in the development of the user interface due to their adaptability and widespread use in web development. HTML lays the foundation of the webpage, CSS takes charge of aesthetics, and JavaScript enables dynamic functionalities. Together, they contribute to a user-friendly, responsive interface.

5.4.5 Remix IDE

The Remix IDE is an online tool for developing, testing, and deploying smart contracts on the Ethereum network. It was chosen for its user-friendly interface, debugging features, and its ability to compile and deploy contracts directly to the Ethereum network.

5.4.6 Truffle Framework

Truffle is used as the development environment of the tests, as well as the testing framework. It is a popular choice among solidity developers as it allows an easy way to compile and run the tests.

5.4.7 Ganache

Ganache is a simulated Ethereum blockchain environment. It acts and runs similar to the real Ethereum blockchain with the exception that the user is controlling all the wallets used in the environment, initially funded with high balances of simulated Ethereum. The environment allows the host computer to mine every block and confirm each transaction made by the testing framework. It has been used in pair with Truffle to test the application.

6 TESTING

6.1 Overview

Considering the nature of Ethereum smart contracts, once deployed, their code cannot be changed. Therefore, the testing of the contract must be extensive and thorough. I built a JavaScript test suite composed of ~550 lines of code with the help of Truffle as the testing framework and Ganache as a simulated Ethereum blockchain environment.

Remix IDE was not used for this task due to its inferiority in the testing field, when compared with Truffle & Ganache capabilities. Conventional testing templates were not chosen either, due to the existence of specialized testing frameworks and the way the interaction with the smart contract works, meaning no data is saved on the local machine, all of it being extracted from the blockchain via the Voting System contract in a dynamic way.

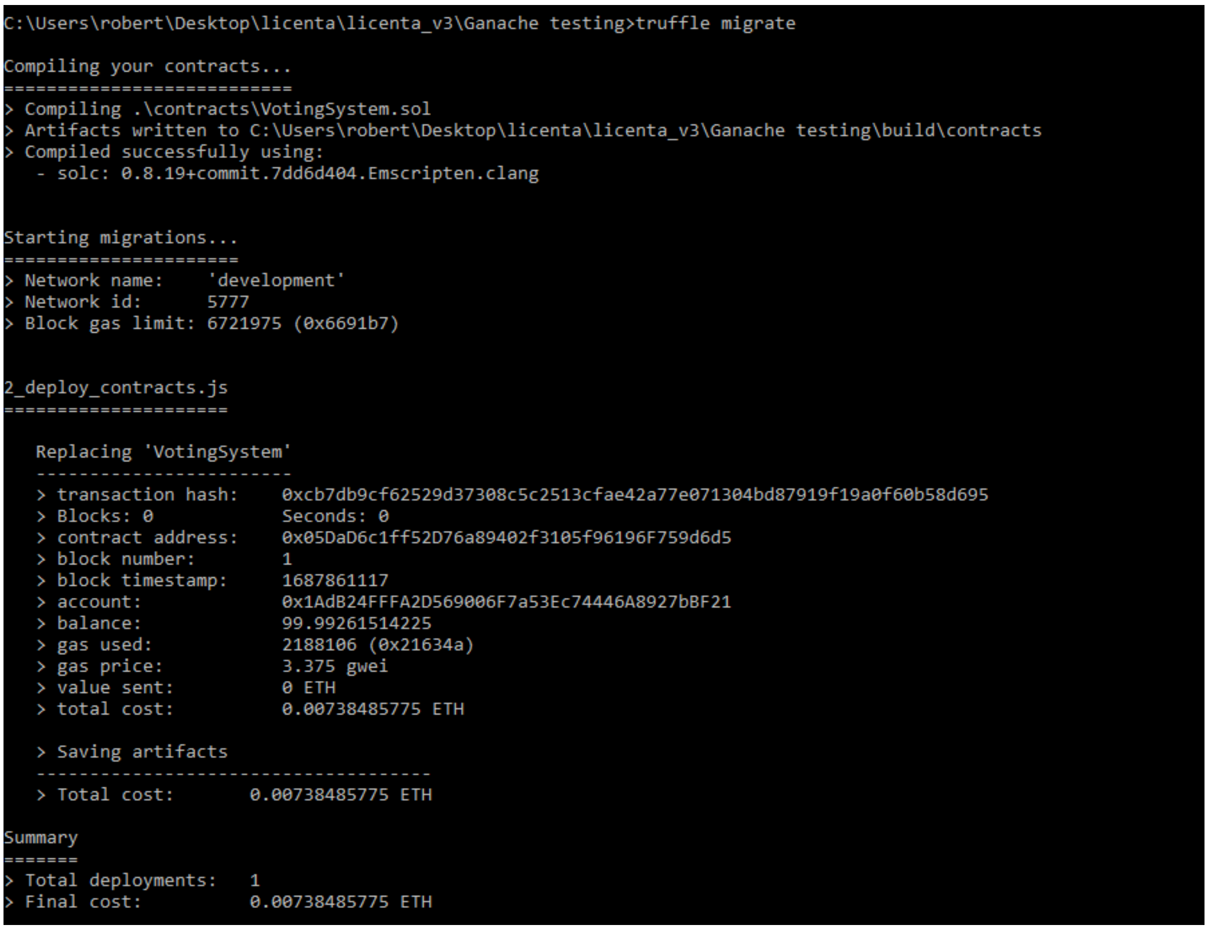
6.2 Testing Environment

6.2.1 Truffle Framework

Truffle is a world-class development environment, testing framework, and asset pipeline for Ethereum, enabling developers to test and build smart contracts effectively and efficiently. Truffle offers built-in smart contract compilation, linking, deployment, and binary management, which simplifies the testing process.

Key features include:

- Automated contract testing for rapid development.
- Scriptable deployment and migrations framework.
- Network management for deploying to many public and private networks.
- Interactive console for direct contract communication.



```
C:\Users\robert\Desktop\licenta\licenta_v3\Ganache testing>truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\VotingSystem.sol
> Artifacts written to C:\Users\robert\Desktop\licenta\licenta_v3\Ganache testing\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd6d404.Emscripten.clang


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


2_deploy_contracts.js
=====================

   Replacing 'VotingSystem'
   ------------------------
   > transaction hash:    0xcb7db9cf62529d37308c5c2513cfae42a77e071304bd87919f19a0f60b58d695
   > Blocks: 0            Seconds: 0
   > contract address:    0x05DaD6c1ff52D76a89402f3105f96196F759d6d5
   > block number:        1
   > block timestamp:     1687861117
   > account:             0x1AdB24FFFA2D569006F7a53Ec74446A8927bBF21
   > balance:             99.99261514225
   > gas used:            2188106 (0x21634a)
   > gas price:           3.375 gwei
   > value sent:          0 ETH
   > total cost:          0.00738485775 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:       0.00738485775 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.00738485775 ETH
```

Figure 13: Contract Deployment on Ganache Test Network Using Truffle ‘migrate’ Command

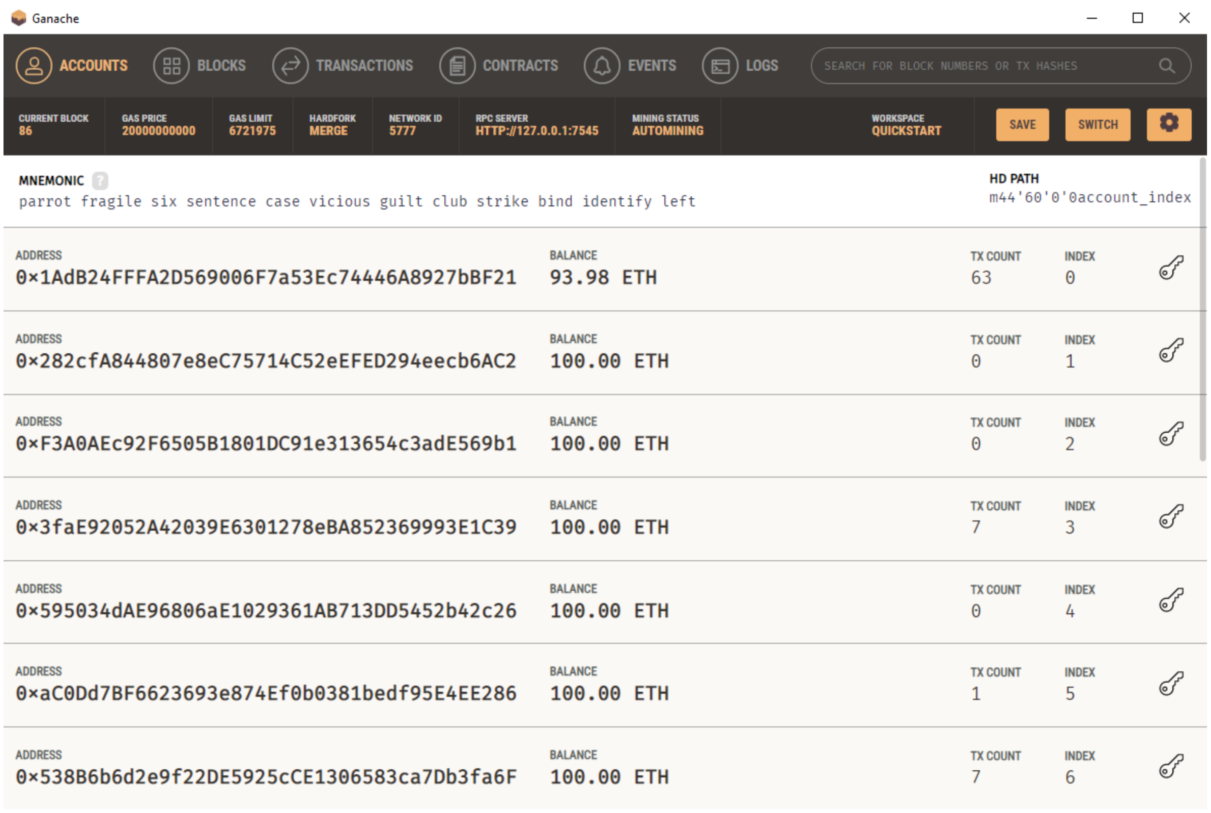
6.2.2 Ganache

Ganache is a personal blockchain for Ethereum development used to deploy contracts, develop applications, and run tests. It is available as both a desktop application as well as a command-line tool (formerly known as the TestRPC). For this project, the desktop application has been used. Ganache creates a virtual Ethereum blockchain, and it generates a couple of fake accounts that are used during testing.

Key features include:

- Ability to quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates.
- Advanced mining controls for mining blocks instantly, on a timer, or manually, for development and testing purposes.

- Built-in block explorer, which allows viewing the details of all transactions that have taken place, including contract creation transactions.



CURRENT BLOCK 86 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MERGE | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE QUICKSTART

MNEMONIC
parrot fragile six sentence case vicious guilt club strike bind identify left

HD PATH
m44'60'0'0account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×1AdB24FFFA2D569006F7a53Ec74446A8927bBF21	93.98 ETH	63	0
0×282cfA844807e8eC75714C52eEFED294eecb6AC2	100.00 ETH	0	1
0×F3A0AEc92F6505B1801DC91e313654c3adE569b1	100.00 ETH	0	2
0×3faE92052A42039E6301278eBA852369993E1C39	100.00 ETH	7	3
0×595034dAE96806aE1029361AB713DD5452b42c26	100.00 ETH	0	4
0×aC0Dd7BF6623693e874Ef0b0381bedf95E4EE286	100.00 ETH	1	5
0×538B6b6d2e9f22DE5925cCE1306583ca7Db3fa6F	100.00 ETH	7	6

Figure 14: Ganache "Accounts" Page, Featuring Some of the 16 Accounts Used for Testing

6.3 Testing Strategy

The security and the correctness of the contract are of major importance. As my testing strategy, I have implemented tests designed to cover all functions of the contract, both for expected (positive) and unexpected (negative) behavior. There have been 16 wallets used for testing and 32 tests conducted.

6.3.1 Positive Testing

Positive tests were executed to make sure the contract functions execute correctly under normal conditions.

These tests include:

1) Initialization: The contract correctly initializes with the owner being the address that deployed it.

2) Candidate Management: The owner can add and remove candidates, and the system prevents the addition of duplicate candidates.

3) Whitelist Management: The owner can add and remove whitelisted voters, and the system prevents the addition of duplicate voters.

4) Voting Control: The owner can start and end the voting process. During the voting process, only whitelisted voters can vote and only for existing candidates. Each whitelisted voter can vote only once per round.

5) Ownership Transfer: The owner can transfer ownership to another address. After the transfer, the old owner loses their owner privileges.

6) Batch Actions: The owner can remove all candidates or all whitelisted voters at once.

7) Winner Calculation: After the voting ends, the system can correctly calculate the winner or winners.

8) Leaderboard: The system correctly ranks candidates based on the number of votes they have received.

6.3.2 Negative Testing

Negative testing ensures that the system can handle errors or unexpected inputs. For the voting system, the following scenarios were tested:

1) Access Control: Non-owners cannot perform owner actions such as adding/removing candidates, adding/removing whitelisted voters, starting/ending voting, or transferring ownership.

2) Voting Limitations: Whitelisted voters cannot vote more than once per round, and non-whitelisted addresses cannot vote at all.

3) Candidate Voting: Candidates cannot vote.

4) Voting Status: The system does not allow voting when it is not open, and it does not allow the addition or removal of candidates or whitelisted voters during the voting process.

5) Winner Reveal: The system does not reveal the winner while the voting is still ongoing.

6.4 Testing Cases and Results

All of the following test cases passed successfully, proving the integrity and correct functioning of the smart contract. The test cases cover the owner actions, non-owner access controls, voting rules, and the leaderboard system.

1) Owner Actions

- contract should initialize with correct owner: Validates that the contract initializes with the correct owner.
- add a candidate: Tests if the contract owner can add a candidate successfully.
- add a whitelisted voter: Tests if the contract owner can add a whitelisted voter successfully.
- cannot add duplicate candidate: Checks the system’s capability to prevent the addition of a duplicate candidate.
- cannot add duplicate voters: Validates the contract’s ability to prevent the addition of duplicate voters.
- remove a candidate: Ensures the contract owner can successfully remove a candidate.
- remove a whitelisted voter: Checks the contract owner’s ability to successfully remove a whitelisted voter.
- start voting: Validates that the contract owner can successfully start voting.
- end voting: Validates that the contract owner can successfully end voting.
- transfer ownership: Tests the contract owner’s ability to transfer ownership to another address.
- owner cannot perform owner actions after transfer: Verifies that, after transferring ownership, the old owner cannot perform actions reserved for the contract.
- only owner can transfer ownership: Checks that only the current contract owner can transfer ownership.
- remove all candidates: Tests the contract owner’s ability to remove all candidates at once.

- remove all whitelisted voters: Checks the owner’s ability to remove all whitelisted voters at once.

2) **Non-Owner Actions and Access Controls**

- non-owner cannot add a candidate: Tests that a non-owner cannot add a candidate.
- non-owner cannot remove a candidate: This test validates that a user who is not the contract’s owner cannot remove a candidate.
- non-owner cannot add a whitelisted voter: This test verifies that users other than the owner cannot add a whitelisted voter.
- non-owner cannot remove a whitelisted voter: This ensures that only the contract’s owner can remove a voter from the whitelist.
- non-owner cannot start voting: Checks that a non-owner cannot start the voting process.
- non-owner cannot end voting: Checks that a non-owner cannot end the voting process.

3) **Voting Actions and Rules**

- vote: Tests that a whitelisted address can vote.
- whitelisted user cannot vote more than once: Validates the contract’s ability to prevent a whitelisted user from voting more than once in a round.
- non-whitelisted user cannot vote: This test confirms that only whitelisted users are permitted to cast a vote.
- cannot vote for a non-candidate: This verifies that the system disallows votes for addresses not listed as candidates.
- candidate cannot vote: Tests that candidates are not able to vote.
- cannot get winner when voting is still open: Validates that the system does not reveal the winner while voting is still ongoing.
- cannot add candidate while voting is in progress: Checks that the system does not allow the addition of candidates during the voting process.

- cannot add whitelisted voter while voting is in progress: Tests that the system does not allow addition of whitelisted voters during the voting process.

- cannot vote when voting is not open: Validates that voting is disallowed when voting is not open.

4) **Leaderboard**

- leaderboard works correctly: Tests that the leaderboard function correctly ranks candidates by vote count.

5) **Winner Determination**

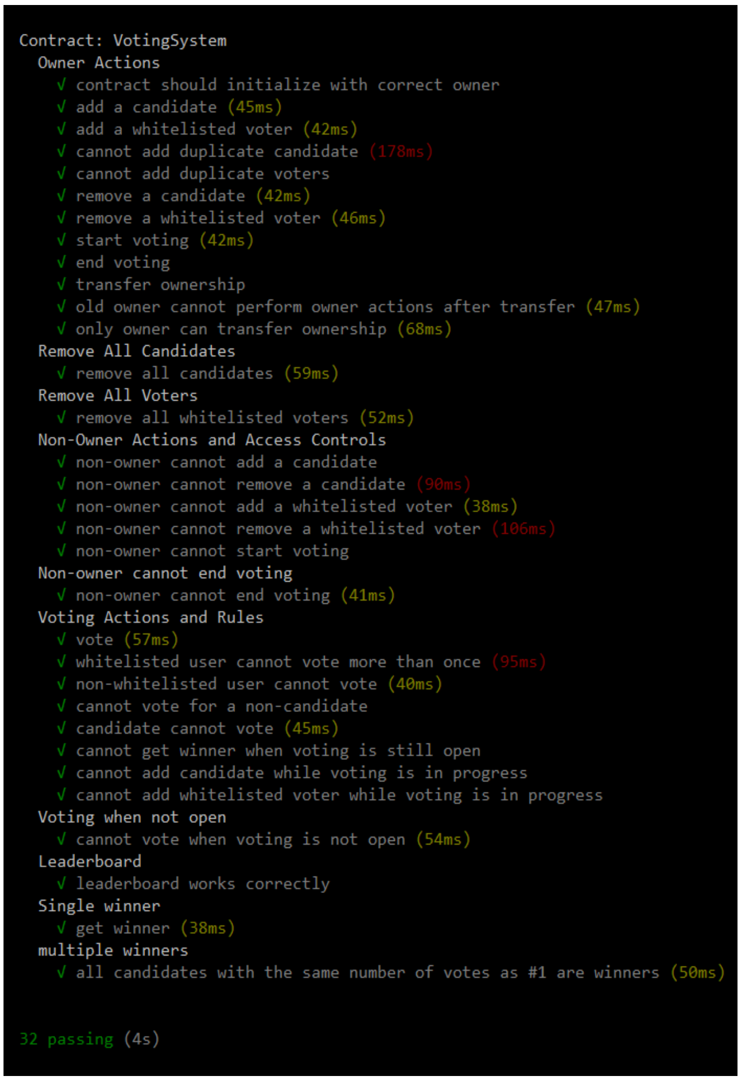
- get winner: Checks that the getWinner function correctly returns the winner after voting has ended.

- all candidates with the same number of votes as #1 are winners: Validates that the system correctly identifies multiple winners in case of a tie.

6.5 Testing Conclusions

All these test cases passed, demonstrating that the voting system can handle both normal and exceptional conditions in a robust and reliable manner.

The successful execution of these tests ensures the integrity and correctness of the voting system.



```
Contract: VotingSystem
  Owner Actions
    √ contract should initialize with correct owner
    √ add a candidate (45ms)
    √ add a whitelisted voter (42ms)
    √ cannot add duplicate candidate (178ms)
    √ cannot add duplicate voters
    √ remove a candidate (42ms)
    √ remove a whitelisted voter (46ms)
    √ start voting (42ms)
    √ end voting
    √ transfer ownership
    √ old owner cannot perform owner actions after transfer (47ms)
    √ only owner can transfer ownership (68ms)
  Remove All Candidates
    √ remove all candidates (59ms)
  Remove All Voters
    √ remove all whitelisted voters (52ms)
  Non-Owner Actions and Access Controls
    √ non-owner cannot add a candidate
    √ non-owner cannot remove a candidate (90ms)
    √ non-owner cannot add a whitelisted voter (38ms)
    √ non-owner cannot remove a whitelisted voter (106ms)
    √ non-owner cannot start voting
  Non-owner cannot end voting
    √ non-owner cannot end voting (41ms)
  Voting Actions and Rules
    √ vote (57ms)
    √ whitelisted user cannot vote more than once (95ms)
    √ non-whitelisted user cannot vote (40ms)
    √ cannot vote for a non-candidate
    √ candidate cannot vote (45ms)
    √ cannot get winner when voting is still open
    √ cannot add candidate while voting is in progress
    √ cannot add whitelisted voter while voting is in progress
  Voting when not open
    √ cannot vote when voting is not open (54ms)
  Leaderboard
    √ leaderboard works correctly
  Single winner
    √ get winner (38ms)
  multiple winners
    √ all candidates with the same number of votes as #1 are winners (50ms)


32 passing (4s)
```

Figure 15: Truffle 'test' Command, the 32 Tests and the Execution Time for Each Test

7 USER MANUAL

7.1 Connect Your Wallet to MetaMask

MetaMask holds its reputation as one of the most popular Ethereum wallets. It is accessible as a browser extension, offering users a seamless connection to the Ethereum blockchain and the various decentralized applications built on it.

MetaMask boasts an intuitive interface that facilitates the creation of wallets, monitoring of balances and past activities, transaction submissions to the blockchain, and interactions with DApps, among other functionalities.

MetaMask was selected as the wallet software for the Voting System because of its user-friendly interface and robust security measures. Nevertheless, using MetaMask is not a prerequisite for the users. They are free to use any wallet software, as all of them should function effectively with the system.

7.1.1 Download the MetaMask Extension

The MetaMask extension can be downloaded from the official website[2].

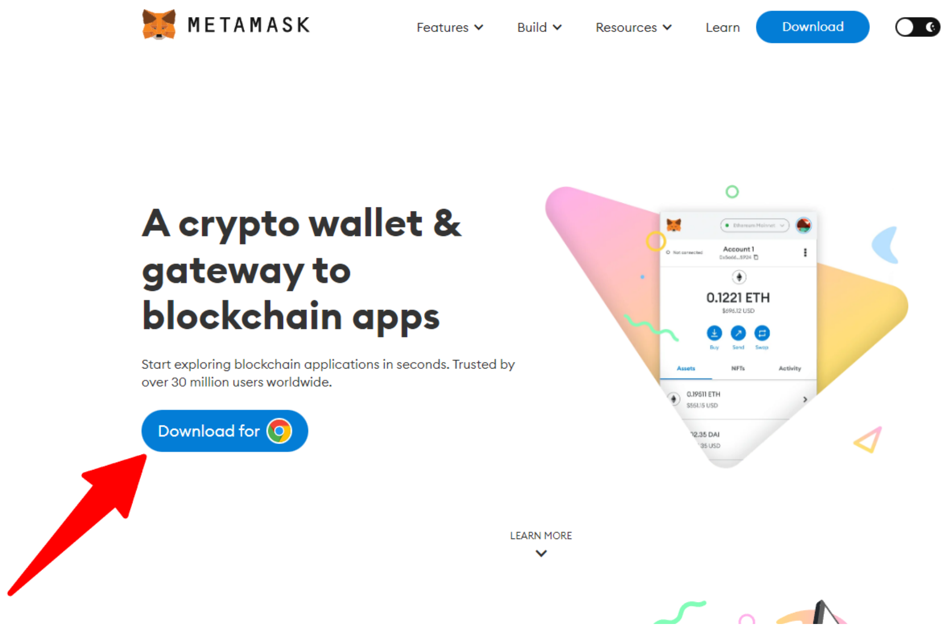

METAMASK
Features
Build
Resources
Learn
Download
A crypto wallet & gateway to blockchain apps
Start exploring blockchain applications in seconds. Trusted by over 30 million users worldwide.
Download for
0.1221 ETH
LEARN MORE

Figure 16: Download the MetaMask Extension from the Official Website

[2] https://metamask.io/

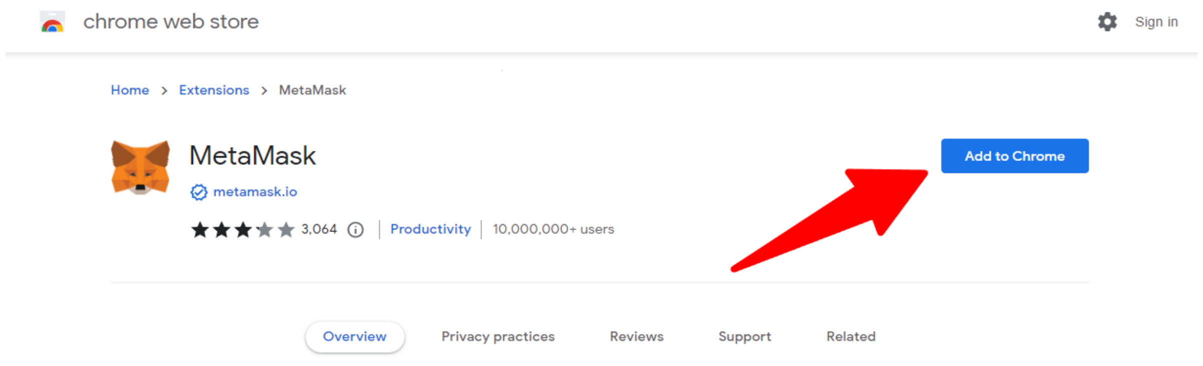


Figure 17: Add the MetaMask Extension to the Web Browser

7.1.2 Create / Import Your Wallet

After the installation of the MetaMask extension, the user will receive a prompt to set up a new wallet or import an existing one. Follow the guided steps, and securely store the seed phrase. This seed phrase will be particularly important if you plan to use the wallet on the actual “mainnet” blockchain as well.

7.1.3 Change MetaMask Network to Goerli Test Network

The default network for MetaMask is mainnet. In order to avoid the real cost of using mainnet Ethereum, the Goerli Test Network has been used.

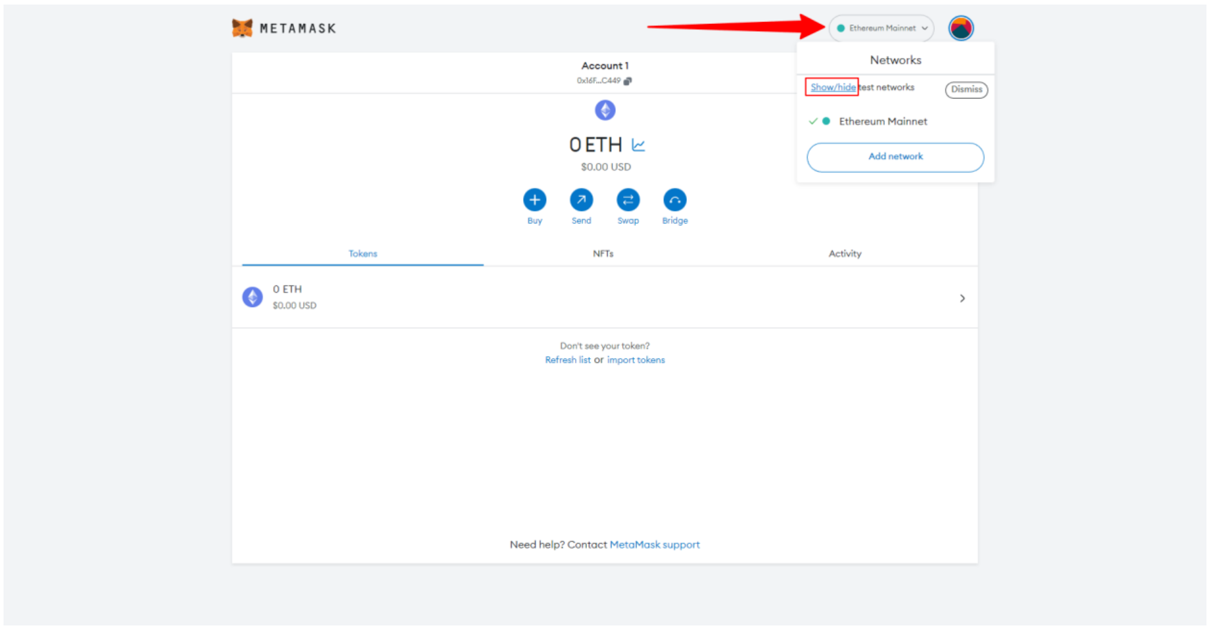
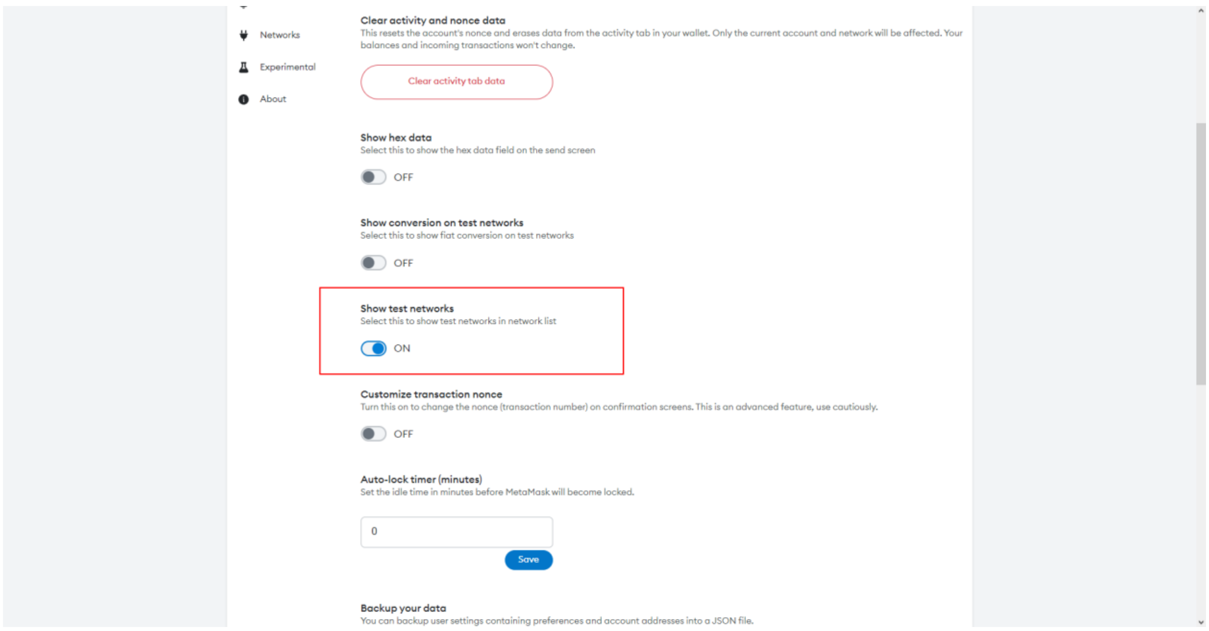


Figure 18: Change the Network to Goerli Test Network Part 1



Clear activity and nonce data
This resets the account's nonce and erases data from the activity tab in your wallet. Only the current account and network will be affected. Your balances and incoming transactions won't change.

Clear activity tab data

Show hex data
Select this to show the hex data field on the send screen

OFF

Show conversion on test networks
Select this to show fiat conversion on test networks

OFF

Show test networks
Select this to show test networks in network list

ON

Customize transaction nonce
Turn this on to change the nonce (transaction number) on confirmation screens. This is an advanced feature, use cautiously.

OFF

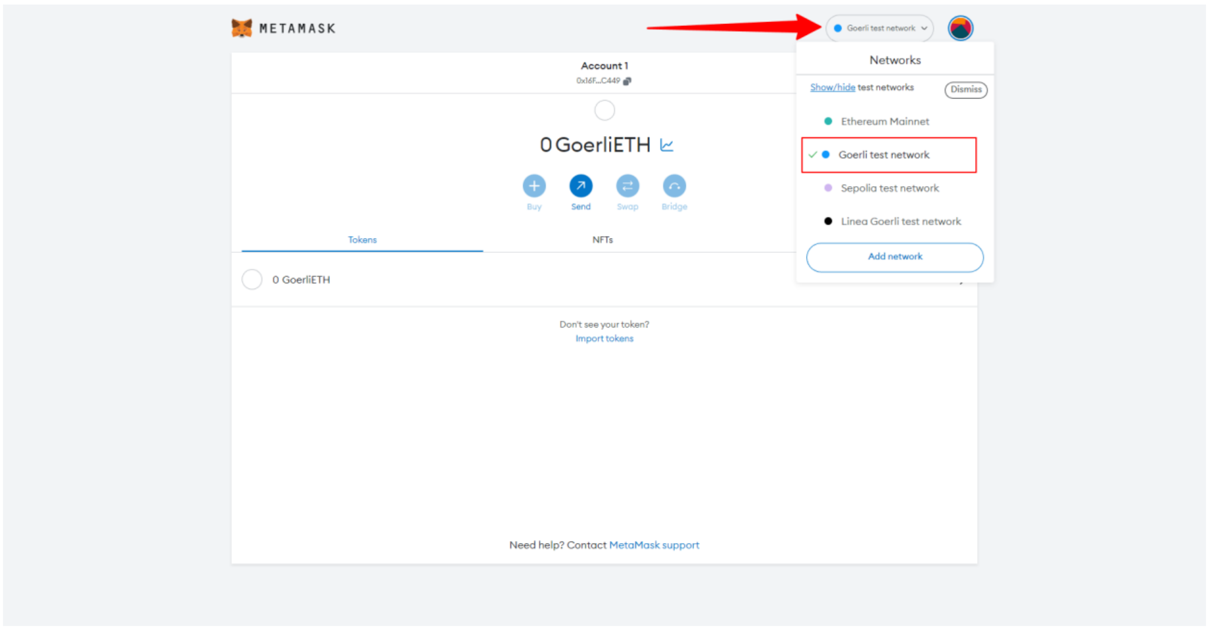
Auto-lock timer (minutes)
Set the idle time in minutes before MetaMask will become locked.

0

Save

Backup your data
You can backup user settings containing preferences and account addresses into a JSON file.

Figure 19: Change the Network to Goerli Test Network Part 2



Account 1
0x16F...C449

0 GoerliETH

0 GoerliETH

Don't see your token?
Import tokens

Need help? Contact MetaMask support

Networks
Show/hide test networks
Ethereum Mainnet
✓ Goerli test network
Sepolia test network
Linea Goerli test network

Figure 20: Change the Network to Goerli Test Network Part 3

7.1.4 Add Goerli Ethereum to Your Wallet

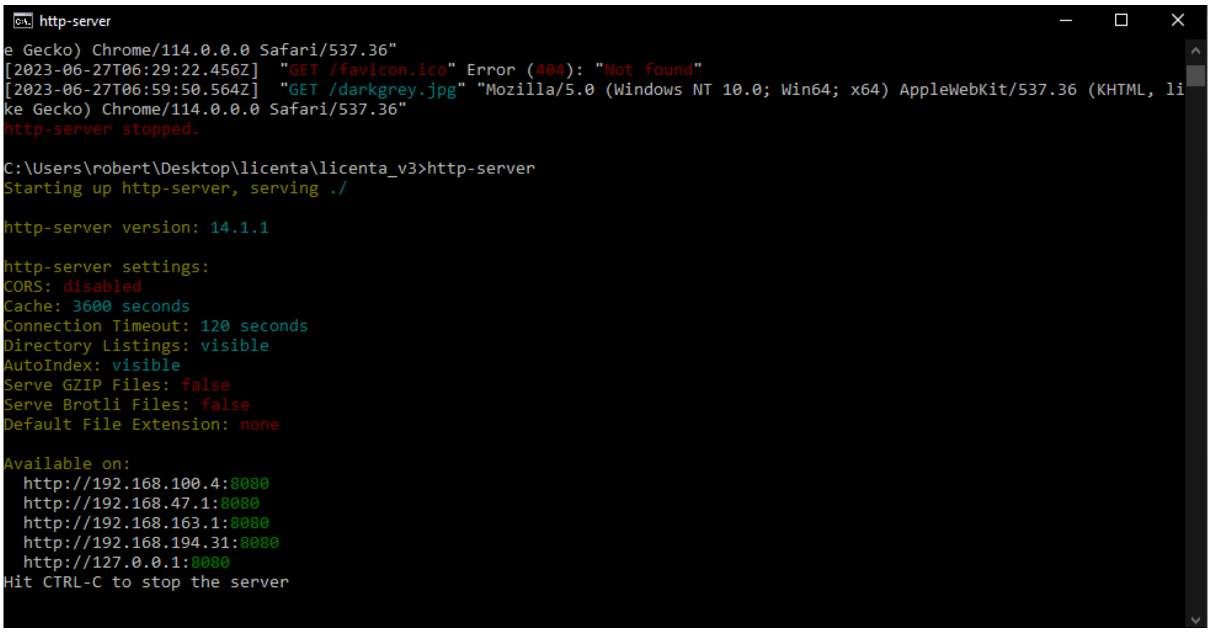
In order to execute “write” transactions such as voting a candidate or starting/ending the voting, a positive balance of Goerli Ethereum is needed, with enough balance to cover the gas costs of the transaction. You can either receive the balance from another account via a transfer or by using a free faucet.

A safe and reliable website to get free Goerli Ethereum is the Alchemy Faucet[3]. Keep in mind that at the moment, an active balance of at least 0.001 Ethereum on the mainnet and an Alchemy account are required to use the faucet.

7.2 Connect to the Voting System

7.2.1 Start a Server

To successfully interact with MetaMask and the DApp interface, a server is needed. I have used 'http-server' package in a command prompt with the help of Node.js's npm.



```
http-server
e Gecko) Chrome/114.0.0.0 Safari/537.36"
[2023-06-27T06:29:22.456Z]  "GET /favicon.ico" Error (404): "Not found"
[2023-06-27T06:59:50.564Z]  "GET /darkgrey.jpg" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
ke Gecko) Chrome/114.0.0.0 Safari/537.36"
http-server stopped.

C:\Users\robert\Desktop\licenta\licenta_v3>http-server
Starting up http-server, serving ./

http-server version: 14.1.1

http-server settings:
CORS: disabled
Cache: 3600 seconds
Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  http://192.168.100.4:8080
  http://192.168.47.1:8080
  http://192.168.163.1:8080
  http://192.168.194.31:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Figure 21: Starting a Server by Using the 'http-server' Package via Node.js's npm

7.2.2 Navigate to the Voting System Website and Connect Wallet

Now that the Ethereum wallet is ready and the server is running, we need to connect to the DApp. The wallet connection to the website is done via the "Connect Wallet" button, as highlighted in the following screenshots:

[3] https://goerlifaucet.com/

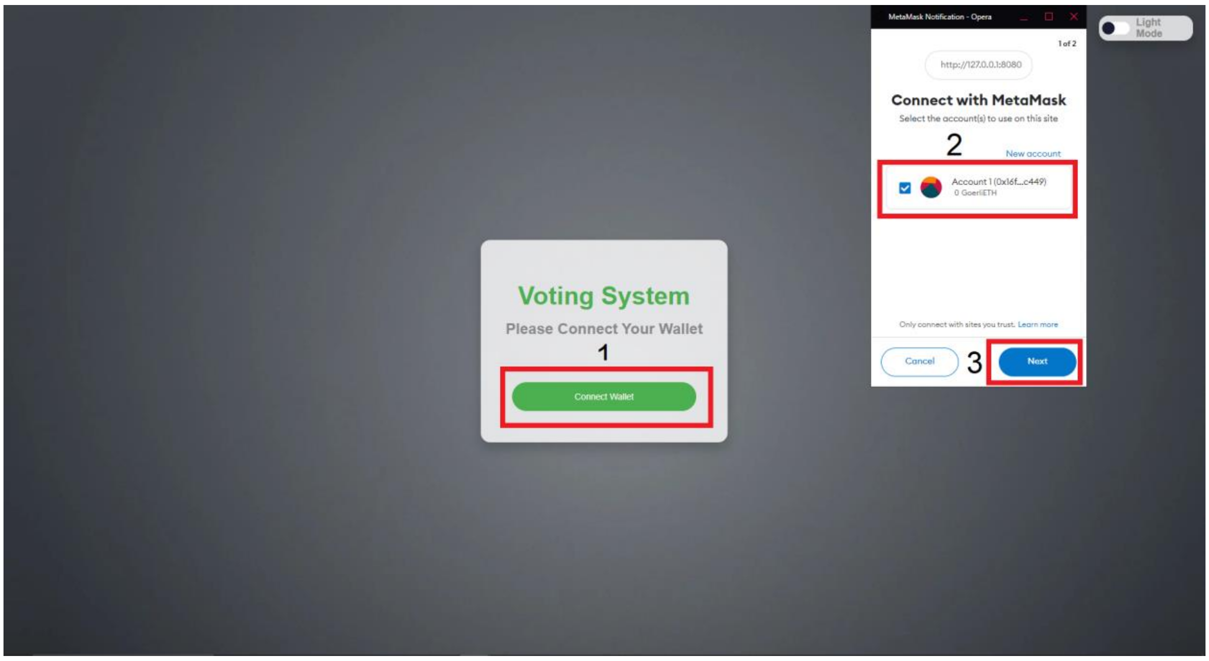


Figure 22: Connecting the Wallet to the Voting System DApp Part 1

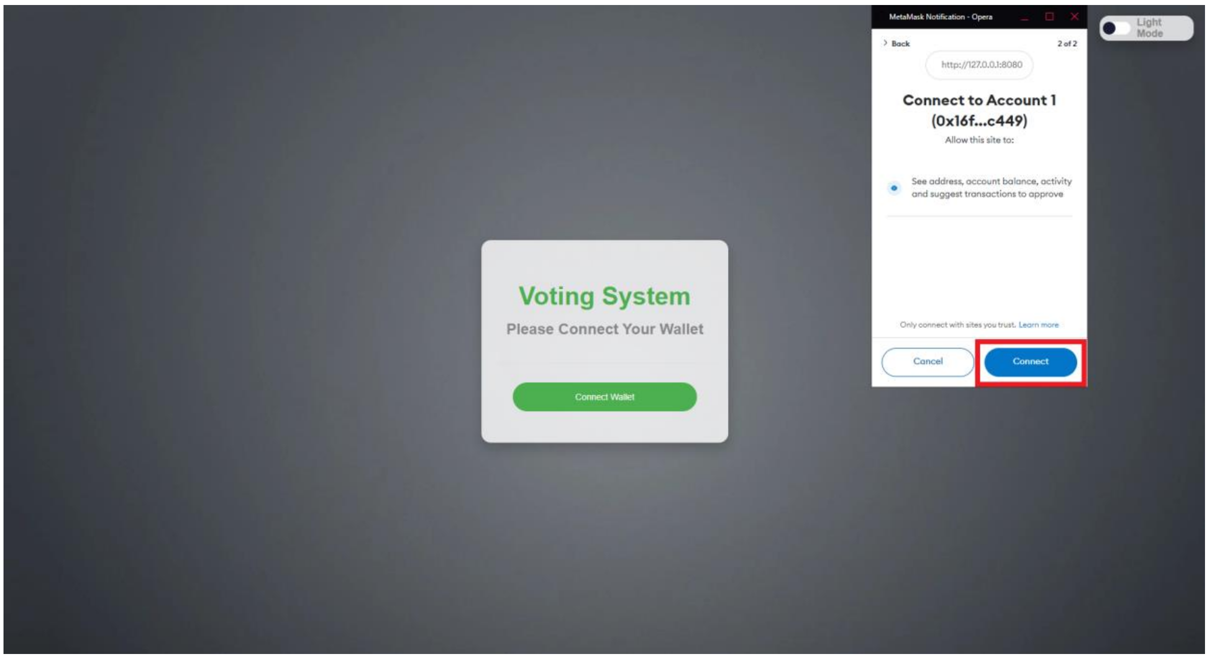
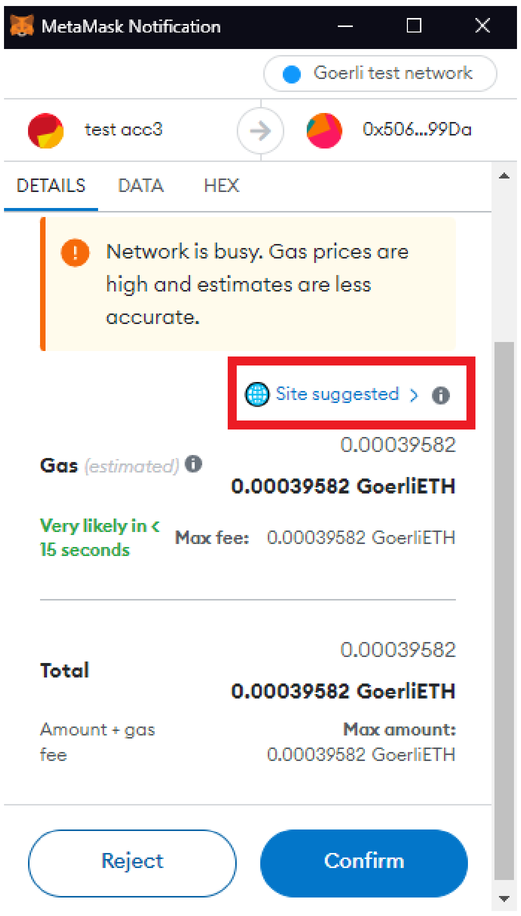

MetaMask Notification - Opera
Back
2 of 2
http://127.0.0.1:8080
Connect to Account 1 (0x16f...c449)
Allow this site to:
See address, account balance, activity and suggest transactions to approve
Only connect with sites you trust. Learn more
Cancel
Connect
Light Mode
Voting System
Please Connect Your Wallet
Connect Wallet

Figure 23: Connecting the Wallet to the Voting System DApp Part 2

7.3 Interact with the Voting System

Depending on the address connected, there will be different interactions available. Keep in mind that “write” transactions consume gas, which will be paid using your Goerli Ethereum balance. For transaction confirmations, it is advisable to go with the “Market” or “Aggressive” options for gas, as shown below:



MetaMask Notification

Goerli test network

test acc3 → 0x506...99Da

DETAILS DATA HEX

Network is busy. Gas prices are high and estimates are less accurate.

Site suggested >

Gas (estimated) 0.00039582
0.00039582 GoerliETH

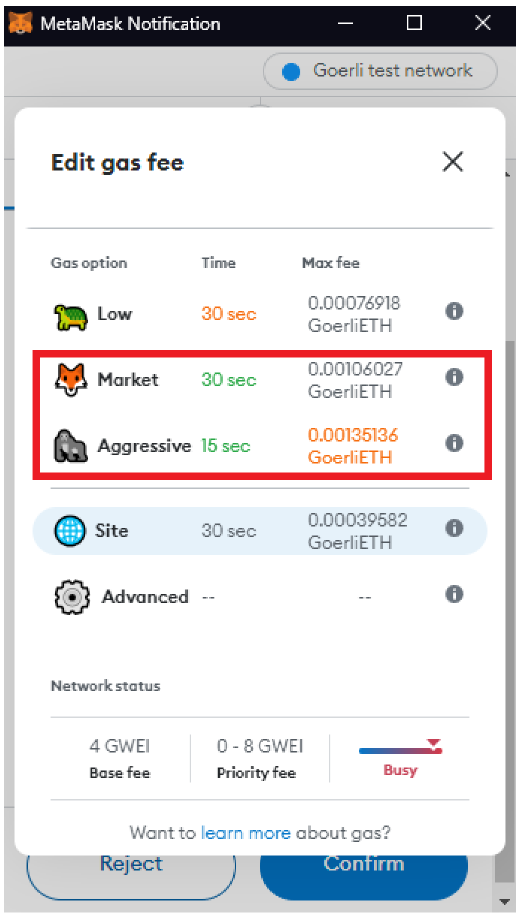
Very likely in < 15 seconds Max fee: 0.00039582 GoerliETH

Total 0.00039582
0.00039582 GoerliETH

Amount + gas fee Max amount: 0.00039582 GoerliETH

Reject Confirm

Figure 24: Adjusting Gas Fees Part 1



MetaMask Notification

Goerli test network

Edit gas fee

Gas option	Time	Max fee
Low	30 sec	0.00076918 GoerliETH
Market	30 sec	0.00106027 GoerliETH
Aggressive	15 sec	0.00135136 GoerliETH
Site	30 sec	0.00039582 GoerliETH
Advanced	--	--

Network status

4 GWEI	0 - 8 GWEI	
Base fee	Priority fee	Busy

Want to learn more about gas?

Reject Confirm

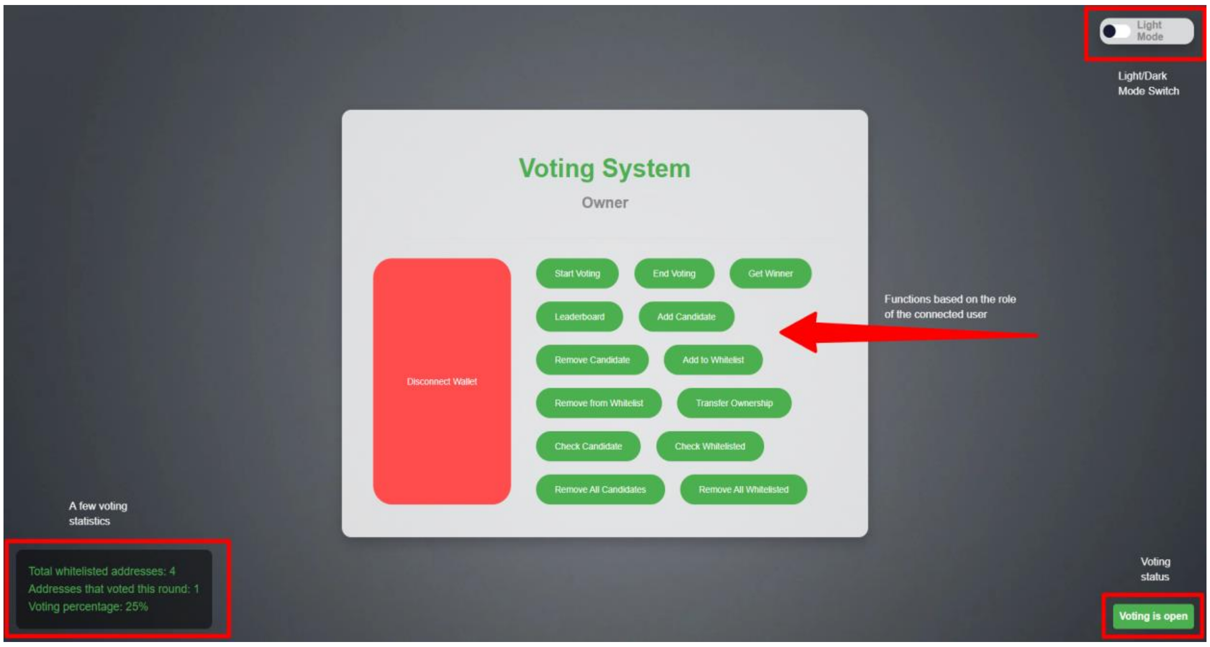
Figure 25: Adjusting Gas Fees Part 2

7.3.1 Owner Interactions

If the connected address corresponds to the contract's owner, a variety of administrative functions become accessible:

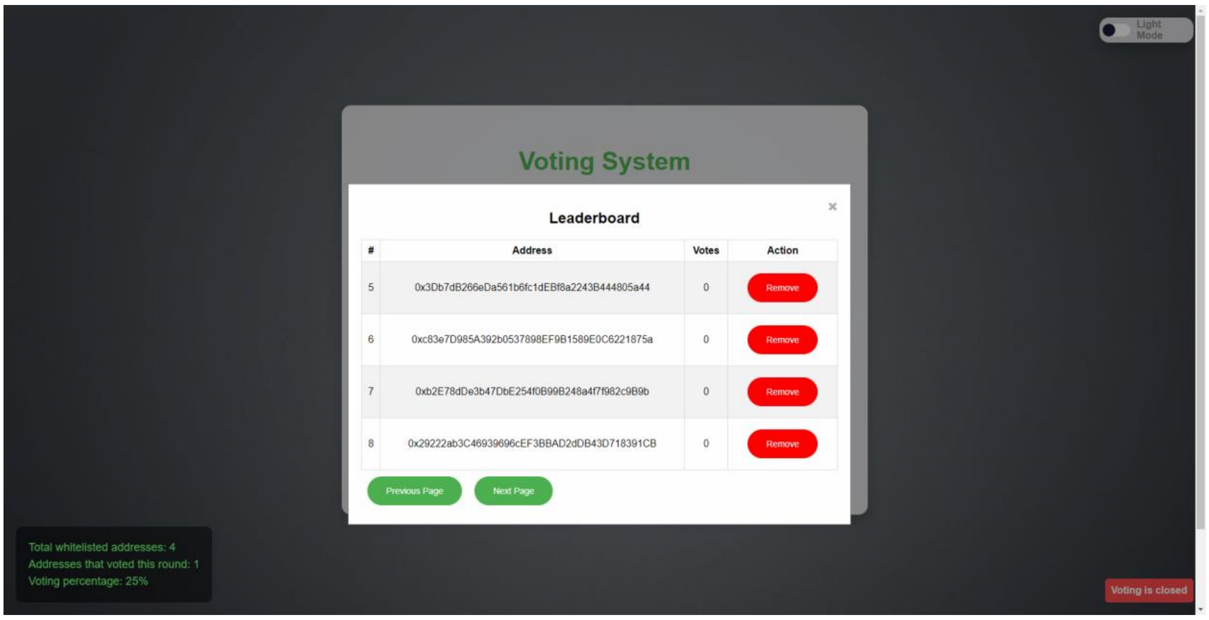
- Start Voting: Clicking this button sets the voting process in motion. It is worth noting that this function is only available when no voting is underway.
- End Voting: This button allows the contract owner to close the ongoing voting process. This action is only available when voting is active.
- Add Candidate: This function enables the contract owner to add a candidate to the pool. The owner has to press this button and input the candidate's address. This action is only permissible when voting is not active.

- Remove Candidate: This button permits the removal of a candidate by entering their address. This operation can only be conducted when voting is not active.
- Remove All Candidates: Using this function, the owner can clear the entire list of candidates. This operation is also limited to periods when voting is not open.
- Add to Whitelist: The owner can add an address to the whitelist by pressing this button and entering the intended address. The function is available only when the voting status is closed.
- Remove from Whitelist: In a similar vein, the owner has the ability to exclude an address from the whitelist while voting is not active.
- Remove All Whitelisted: This function allows the owner to purge all addresses from the whitelist. It is only executable when voting is not ongoing.
- Transfer Ownership: This function permits the transfer of contract ownership. The owner can click this button and enter the address they wish to transfer the ownership to.
- Leaderboard: By clicking this button, the contract owner can view the leaderboard. It presents the candidates sorted according to their vote count. If voting is closed, a "Remove" button will appear under the "Action" column.



Total whitelisted addresses: 4
Addresses that voted this round: 1
Voting percentage: 25%

Figure 26: Owner Panel with Details Regarding the UI



Leaderboard

#	Address	Votes	Action
5	0x3Db7dB266eDa561b6fc1dEBf8a2243B444805a44	0	Remove
6	0xc83e7D985A392b0537898EF9B1589E0C6221875a	0	Remove
7	0xb2E78dDe3b47DbE254f0B99B248a4f7f982c9B9b	0	Remove
8	0x29222ab3C46939696cEF3BBAD2dDB43D718391CB	0	Remove

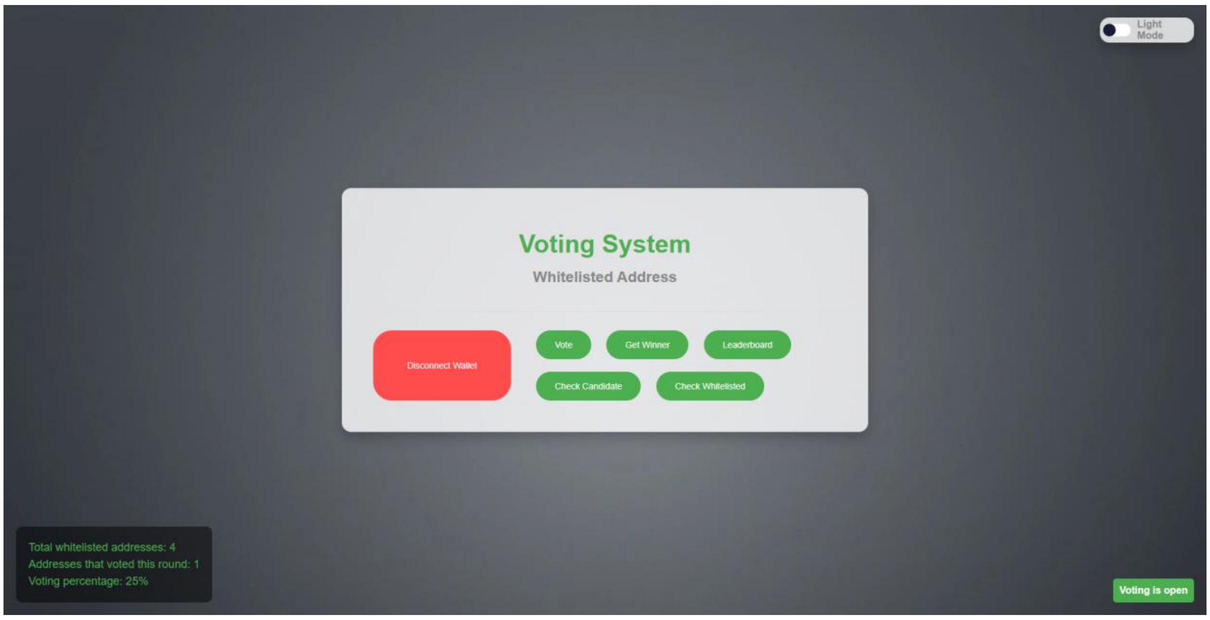
Total whitelisted addresses: 4
Addresses that voted this round: 1
Voting percentage: 25%

Figure 27: Owner's 'Leaderboard' When the Status of the Voting is Closed

7.3.2 Whitelisted / Voter Interactions

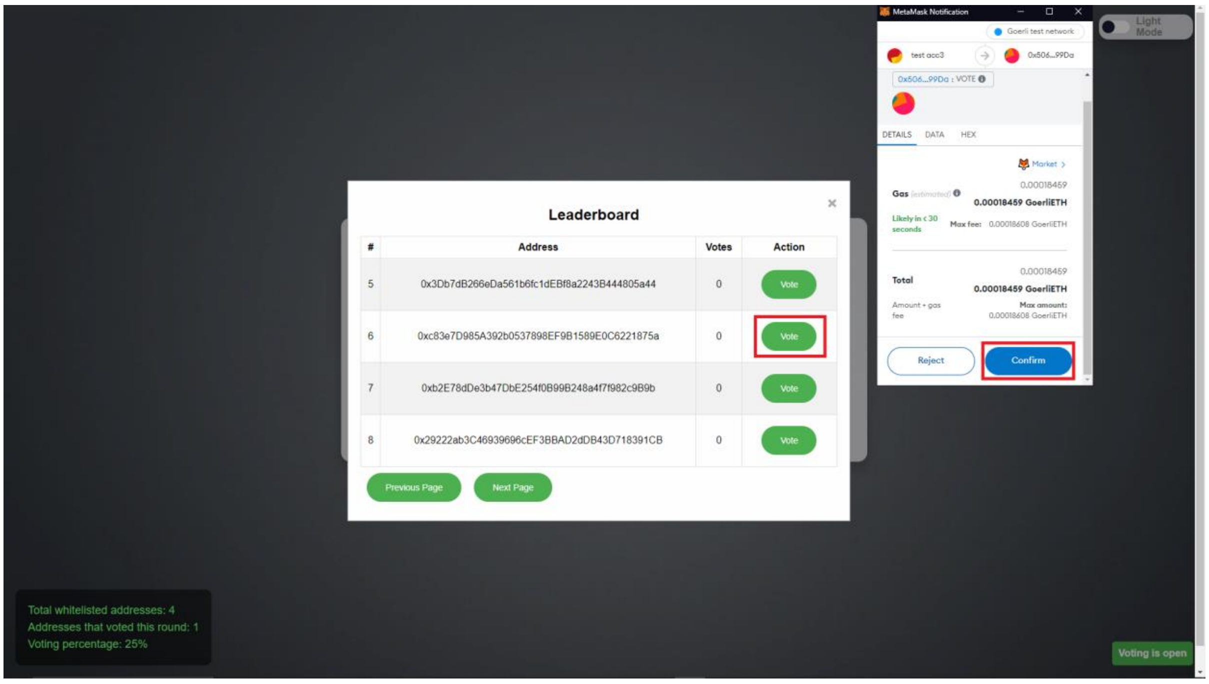
In case the connected address belongs to a whitelisted individual, the system will grant the user access to the voting function as well as the "Public Address" functions:

- Vote: This button activates the voting mechanism. On clicking it, the user has to enter the candidate's address they wish to vote for. Voting is only possible when the voting process is officially open.
- Leaderboard: This button directs the user to the leaderboard, listing all candidates in order of the vote count they have received. During open voting periods, a "Vote" button will become visible under the "Action" column.



Total whitelisted addresses: 4
Addresses that voted this round: 1
Voting percentage: 25%

Figure 28: Whitelisted Address Panel



Gas (estimated) 0.00018459
0.00018459 GoerliETH
Likely in < 30 seconds
Max fee: 0.00018608 GoerliETH

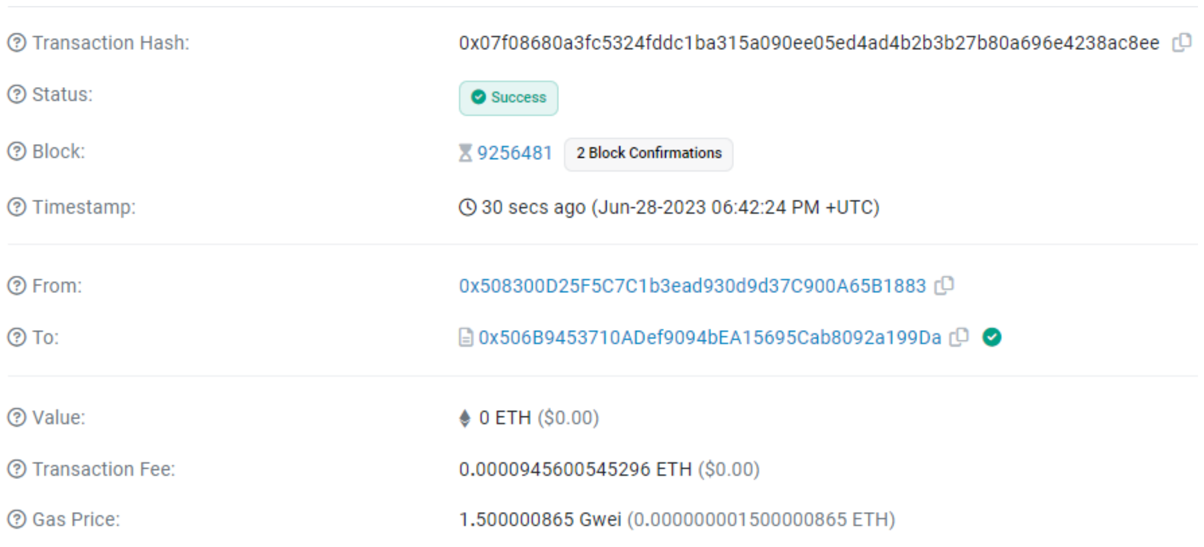
Total 0.00018459
0.00018459 GoerliETH
Amount + gas fee
Max amount: 0.00018608 GoerliETH

Leaderboard

#	Address	Votes	Action
5	0x3Db7dB266eDa561b6fc1dEBf8a2243B444805a44	0	Vote
6	0xc83e7D985A392b0537898EF9B1589E0C6221875a	0	Vote
7	0xb2E78dDe3b47DbE254f0B99B248a4f7f982c9B9b	0	Vote
8	0x29222ab3C46939696cEF3BBAD2dDB43D718391CB	0	Vote

Total whitelisted addresses: 4
Addresses that voted this round: 1
Voting percentage: 25%

Figure 29: Executing a Vote by Using the ‘Leaderboard’



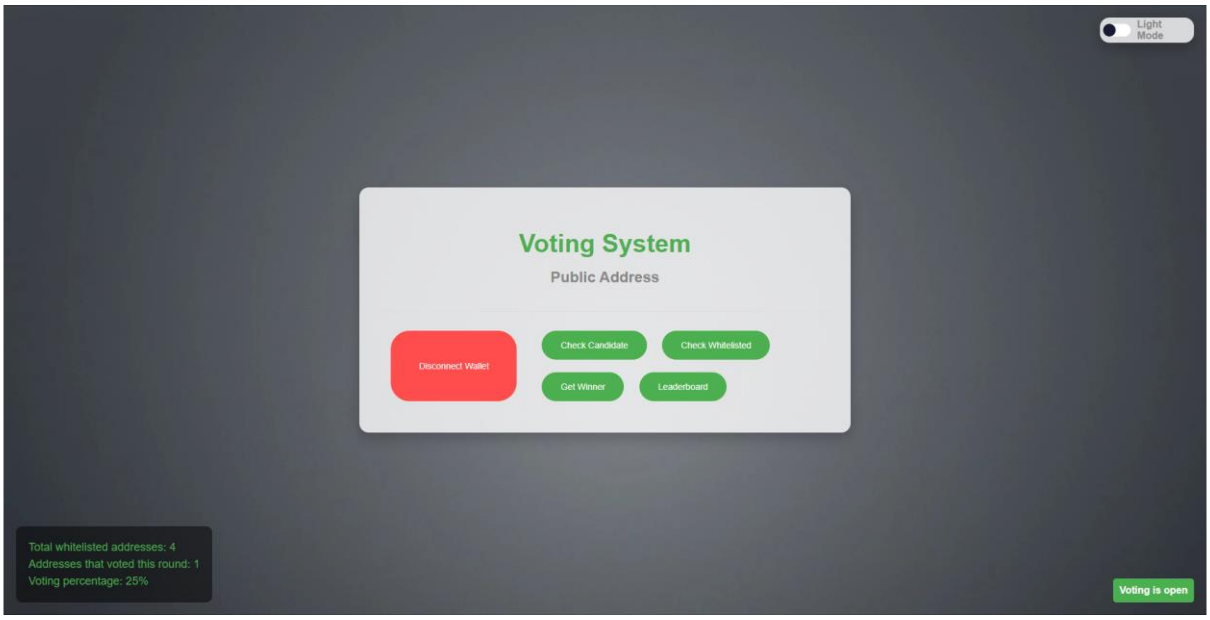
Transaction Hash:	0x07f08680a3fc5324fddc1ba315a090ee05ed4ad4b2b3b27b80a696e4238ac8ee
Status:	Success
Block:	9256481 2 Block Confirmations
Timestamp:	30 secs ago (Jun-28-2023 06:42:24 PM +UTC)
From:	0x508300D25F5C7C1b3ead930d9d37C900A65B1883
To:	0x506B9453710ADef9094bEA15695Cab8092a199Da
Value:	0 ETH ($0.00)
Transaction Fee:	0.0000945600545296 ETH ($0.00)
Gas Price:	1.500000865 Gwei (0.000000001500000865 ETH)

Figure 30: Blockchain Confirmation of the 'Vote' Transaction

7.3.3 Public Address / Visitor Interactions

If the user is connected via a public address, the system enables the following functions:

- Check Candidate: By clicking this button and providing an address, the user can verify if it belongs to a registered candidate.
- Check Whitelisted: Similar to the candidate check, clicking this button and entering an address can confirm if it is on the whitelist.
- Get Winner: This button reveals the winner(s) of the vote, though it is only functional after the voting period has ended.
- Leaderboard: This button provides access to the leaderboard, presenting the candidates sorted by their vote tally.

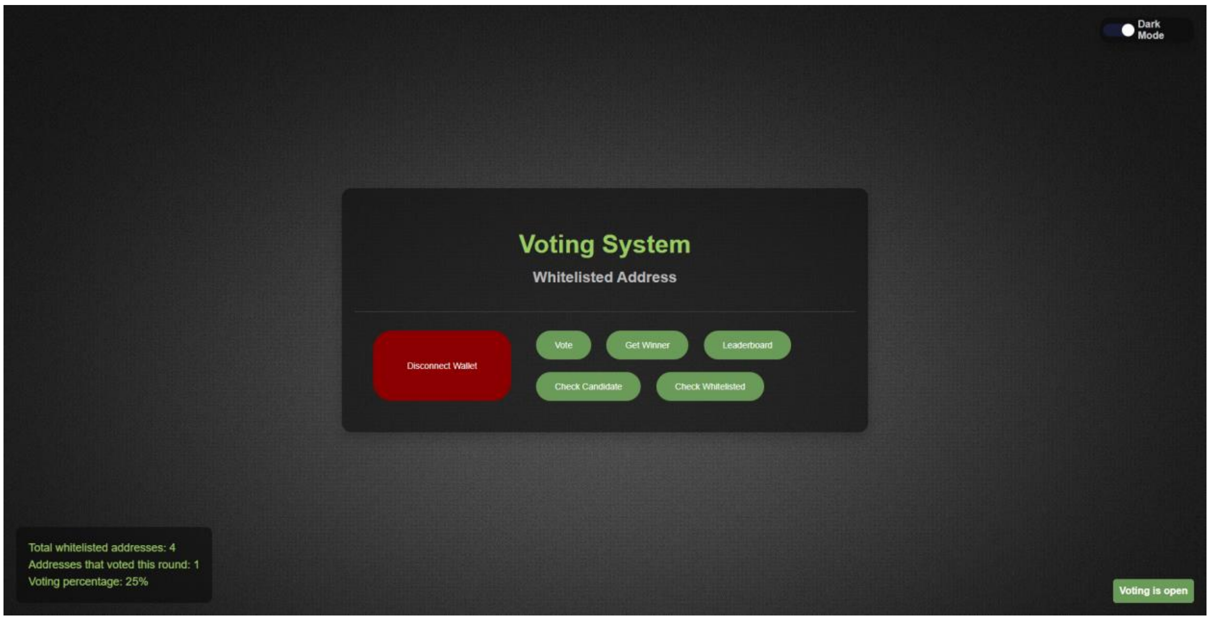


Total whitelisted addresses: 4
Addresses that voted this round: 1
Voting percentage: 25%

Figure 31: Public Address / Visitor Panel

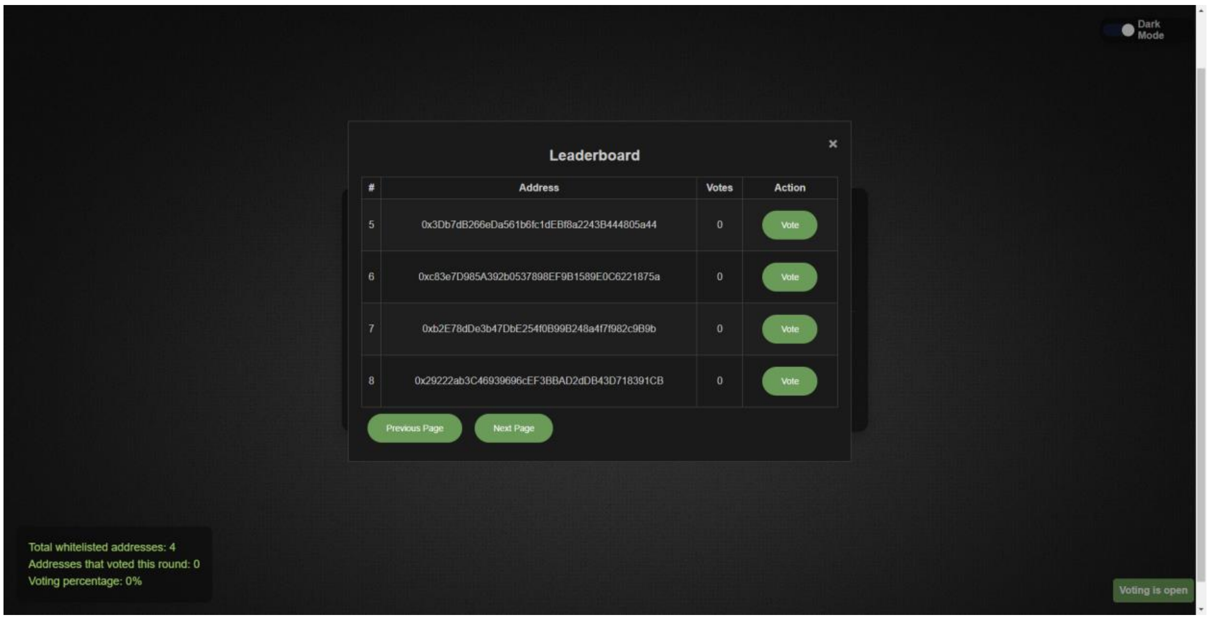
7.4 Switch Themes

You can switch between light and dark themes by toggling the theme switch button from the top right corner on the website.



Total whitelisted addresses: 4
Addresses that voted this round: 1
Voting percentage: 25%

Figure 32: Dark Theme Home Page Example 1



Leaderboard

#	Address	Votes	Action
5	0x3Db7dB266eDa561b6fc1dEBf8a2243B444805a44	0	Vote
6	0xc83e7D985A392b0537898EF9B1589E0C6221875a	0	Vote
7	0xb2E78dDe3b47DbE254f0B99B248a4f7f982c9B9b	0	Vote
8	0x29222ab3C46939696cEF3BBAD2dDB43D718391CB	0	Vote

Total whitelisted addresses: 4
Addresses that voted this round: 0
Voting percentage: 0%

Figure 33: Dark Theme Leaderboard Example 2

8 CONCLUSIONS

In closing, the presented voting system showcases a promising transformation in how future voting procedures could be executed. This project underlines the potential of blockchain technology to increase transparency and efficiency while reducing fraud risks.

Throughout this project I have encountered various challenges, such as ensuring the solidity contract's robustness and creating a user-friendly and intuitive front-end interface. I had to consider the multitude of different roles and permissions within the system, in order to ensure a flexible yet controlled environment.

The presented system has shown not only the capability of conducting a voting election, but its security, transparency, and ease of access too. The decentralized nature of the Ethereum blockchain brings significant advantages to the voting system. It ensures that any individual, at any time, using any virtual wallet, can verify the status of the candidates, inspect the leaderboard, and access other voting-related information.

The journey to develop this blockchain-powered voting system has been more than just a technical hurdle. It has also been a chance to drive innovation and reevaluate traditional procedures. The experience has emphasized the transformative capabilities of blockchain technology, signaling a new phase marked by transparent, secure, and efficient voting procedures.

9 BIBLIOGRAPHY

Kilroy, K. (n.d.). What Is Blockchain? How Does Blockchain Work? Retrieved from Kilroy Blockchain: https://kilroyblockchain.com/what-is-blockchain

W3Schools. (n.d.). CSS Tutorial. Retrieved from https://www.w3schools.com/css/

ConsenSys. (n.d.). Ethereum for developers. Retrieved from https://consensys.net/developers/

Ethereum. (n.d.). Solidity Documentation. Retrieved from https://docs.soliditylang.org/

Truffle Suite. (n.d.). Truffle Documentation. Retrieved from https://www.trufflesuite.com/docs

Truffle Suite. (n.d.). Ganache Documentation. Retrieved from https://www.trufflesuite.com/docs/ganache/

A. SOURCE CODE

Due to the size of the application: Solidity contract + Testing code + HTML + JavaScript + CSS, the source code is too large for the documentation. However, I will provide the source code for the Solidity contract. The rest is provided in the Github link.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.6.0 <0.9.0;

// Define the contract called VotingSystem
contract VotingSystem {
    // Define public state variables
    address public owner;  // Stores the owner's address
    mapping (address => bool) public candidates;  // Stores candidate addresses
    mapping (address => bool) public whitelisted;  // Stores whitelisted voter addresses
    mapping (address => uint) public votes;  // Stores the vote count for each candidate
    address[] public candidateList;  // List of candidate addresses
    address[] public whitelistedList;  // List of whitelisted voter addresses
    bool public votingOpen = false;  // Tracks whether voting is currently open
    uint public votersInCurrentRound = 0;  // Counts the number of voters in the current
round
    uint public currentRound = 0;  // Tracks the current round of voting
    mapping (address => uint) public lastVotedRound;  // Stores the last round in which
each address voted

    // Define a modifier called onlyOwner, which ensures only the owner can call a function
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    // Define a modifier called votingEnded, which ensures a function can only be called
after voting has ended
    modifier votingEnded {
        require(currentRound > 0 && !votingOpen, "Voting has not ended or has not started
yet");
        _;
    }

    // Define a modifier called onlyWhitelisted, which ensures only whitelisted addresses
can call a function
    modifier onlyWhitelisted {
        require(whitelisted[msg.sender] == true);
        _;
    }

    // Constructor function that is called when the contract is deployed
    constructor() {
        owner = msg.sender;  // Set the owner as the address that deployed the contract
    }

    // Function to add a new candidate
    function addCandidate(address candidate) public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        require(candidates[candidate] == false, "Candidate is already added");  // Check
that the candidate is not already added
        candidates[candidate] = true;  // Add the candidate
        candidateList.push(candidate);  // Add the candidate to the candidate list
    }

    // Function to remove a candidate
    function removeCandidate(address candidate) public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        candidates[candidate] = false;  // Remove the candidate
        // Loop through the candidate list to find and remove the candidate
        for (uint i = 0; i < candidateList.length; i++) {
            if (candidateList[i] == candidate) {
                candidateList[i] = candidateList[candidateList.length - 1];
                candidateList.pop();
                break;
            }
        }
    }

    // Function to add a new whitelisted voter
    function addWhitelisted(address voter) public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        require(whitelisted[voter] == false, "Voter is already added");  // Check that the
voter is not already added
        whitelisted[voter] = true;  // Add the voter to the whitelist
        whitelistedList.push(voter);  // Add the voter to the whitelisted list
    }

    // Function to remove a whitelisted voter
    function removeWhitelisted(address voter) public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        whitelisted[voter] = false;  // Remove the voter from the whitelist
        // Loop through the whitelisted list to find and remove the voter
        for (uint i = 0; i < whitelistedList.length; i++) {
            if (whitelistedList[i] == voter) {
                whitelistedList[i] = whitelistedList[whitelistedList.length - 1];
                whitelistedList.pop();
                break;
            }
        }
    }

    // Private function to reset the votes for each candidate
    function resetVotes() private {
        for (uint i = 0; i < candidateList.length; i++) {
            votes[candidateList[i]] = 0;
        }
    }

    // Function to start a new round of voting
    function startVoting() public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        votingOpen = true;  // Set voting to open
        currentRound++;  // Increment the current round
        votersInCurrentRound = 0;  // Reset the voters count for the new round
        resetVotes();  // Reset votes for each candidate
    }

    // Function to end the current round of voting
    function endVoting() public onlyOwner {
        require(votingOpen, "Voting is not currently open");  // Check that voting is
currently open
        votingOpen = false;  // Set voting to closed
    }

    // Function to cast a vote for a candidate
    function vote(address candidate) public onlyWhitelisted {
        require(votingOpen, "Voting is not currently open");  // Check that voting is
currently open
        require(candidates[candidate] == true, "Candidate does not exist");  // Check that
the candidate exists
        require(lastVotedRound[msg.sender] < currentRound, "You have already voted in this
round");  // Check that the voter has not already voted in this round

        votes[candidate]++;  // Increment the vote count for the candidate
        lastVotedRound[msg.sender] = currentRound;  // Update the last round in which the
voter voted
        votersInCurrentRound++;  // Increment the count of voters in this round
    }

    // Function to get the winners of the election
    function getWinner() public view votingEnded returns (address[] memory) {
        require(candidateList.length > 0, "No candidates");  // Check that there are
candidates

        address[] memory sortedCandidates = leaderboard();  // Get the sorted list of
candidates
        uint topVotes = votes[sortedCandidates[0]];  // Get the highest vote count

        // Calculate the count of winners
        uint winnersCount = 0;
        for (uint i = 0; i < sortedCandidates.length; i++) {
            if (votes[sortedCandidates[i]] == topVotes) {
                winnersCount++;
            } else {
                break;
            }
        }

        // Fill the winners array
        address[] memory winners = new address[](winnersCount);
        for (uint i = 0; i < winnersCount; i++) {
            winners[i] = sortedCandidates[i];
        }
        return winners;  // Return the array of winners
    }

    // Function to check if an address is a candidate
    function checkIfCandidate(address _address) public view returns (bool) {
        return candidates[_address];  // Return true if the address is a candidate, false
otherwise
    }

    // Function to check if an address is whitelisted
    function checkIfWhitelisted(address _address) public view returns (bool) {
        return whitelisted[_address];  // Return true if the address is whitelisted, false
otherwise
    }

    // Function to transfer ownership of the contract
    function transferOwnership(address newOwner) public onlyOwner {
        owner = newOwner;  // Set the owner to the new owner
    }

    // Function to get the total number of whitelisted voters
    function totalWhitelisted() public view returns (uint) {
        return whitelistedList.length;  // Return the length of the whitelisted list
    }

    // Function to get the percentage of whitelisted voters who have voted
    function votingPercentage() public view returns (uint) {
        uint total = totalWhitelisted();  // Get the total number of whitelisted voters

        return total == 0 ? 0 : (votersInCurrentRound * 100) / total;  // Calculate and
return the voting percentage
    }

    // Function to remove all candidates
    function removeAllCandidates() public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        for (uint i = 0; i < candidateList.length; i++) {
            candidates[candidateList[i]] = false;  // Remove each candidate from the
candidates mapping
        }
        delete candidateList;  // Delete the candidate list
    }

    // Function to remove all whitelisted voters
    function removeAllWhitelisted() public onlyOwner {
        require(!votingOpen, "Voting is currently open");  // Check that voting is not
currently open
        for (uint i = 0; i < whitelistedList.length; i++) {
            whitelisted[whitelistedList[i]] = false;  // Remove each voter from the
whitelisted mapping
        }
        delete whitelistedList;  // Delete the whitelisted list
    }

    // Function to get a sorted list of candidates based on vote count
    function leaderboard() public view returns (address[] memory) {
        require(candidateList.length > 0, "No candidates");  // Check that there are
candidates
        address[] memory sortedCandidates = candidateList;  // Copy the candidate list

        // Sort the candidates by vote count
        for (uint i = 0; i < sortedCandidates.length; i++) {
            for (uint j = i + 1; j < sortedCandidates.length; j++) {
                if (votes[sortedCandidates[j]] > votes[sortedCandidates[i]]) {
                    // Swap the candidates if the one at position j has more votes than the
one at position i
                    address temp = sortedCandidates[i];
                    sortedCandidates[i] = sortedCandidates[j];
                    sortedCandidates[j] = temp;
                }
            }
        }

        return sortedCandidates;  // Return the sorted list of candidates
    }
}
```

B. INDEX