DECENTRALIZED VOTING SYSTEM USING BLOCKCHAIN

A PROJECT REPORT

Submitted by

DHUSNY RAJ P	**[963520104022]**
ROSHAN LAL J	**[963520104041]**
AANI MONCY M	**[963520104001]**
JINCY A	**[963520104029]**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

STELLA MARY'S COLLEGE OF ENGINEERING

ANNA UNIVERSITY: CHENNAI 600025

MAY 2023

BONAFIDE CERTIFICATE

Certified that this project titled "**DECENTRALIZED VOTING SYSTEM USING BLOCKCHAIN"** is the bonafide work of "**DHUSNY RAJ P (963520104022), ROSHAN LAL J (963520104041), AANI MONCY M (963520104001), JINCY A (963520104029)"** who carried out the project under my supervision.

SIGNATURE	**SIGNATURE**
Dr. F. R SHINY MALAR M. Tech, Ph.D.,	Mrs. V. SUBITHA M.E.,
HEAD OF THE DEPARTMENT	SUPERVISOR
Professor and Head	Assistant Professor,
Department of Computer Science	Department of Computer Science
and Engineering,	and Engineering,
Stella Mary's College of Engineering,	Stella Mary's College of Engineering,
Aruthenganvilai,	Aruthenganvilai,
Azhikal post,	Azhikal post,
Kanyakumari District	Kanyakumari District

Submitted for the Project Report viva-voce Examination held on ....

at Stella Mary's College of Engineering, Aruthenganvilai.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project report presents the design and implementation of a Decentralized voting system using blockchain technology. The purpose of this system is to provide a secure, transparent, and immutable platform for conducting elections. The system employs the use of smart contracts on the Ethereum blockchain to automate the voting process and ensure that all votes are counted accurately. The system was developed using Solidity, a programming language for smart contracts, and Truffle, a development framework for Ethereum. The system's frontend was developed using HTML, CSS and JavaScript. The system's security was ensured through the use of encryption techniques and private keys to secure users' identities and their voting records. The system's architecture allows for the registration of voters, the creation of ballots, the casting of votes, and the tallying of results, all on the blockchain. The system's decentralized nature ensures that votes cannot be tampered with, and the results are transparent and available for all participants to verify. This project report concludes that a decentralized voting system using blockchain technology has the potential to revolutionize the way elections are conducted, by providing a secure, transparent, and immutable platform that can be trusted by all participants. The system can be used in various settings, including governmental and non-governmental elections, corporate boardroom elections, and more.

Keywords: Ethereum, Blockchain voting, Decentralization, election

ACKNOWLEDGEMENT

We wish to express our sincere thanks to our parents and almighty, the supreme guide whose continuous grace and mercy see us through each day and have given us courage in the undertaking of this project.

Our sincere thanks goes to the chairman **Dr. Nazareth Charles Ex. Indian Navy**, for his encouragement and for rendering a platform for completing the project in a successful manner.

We wish to record our sincere gratitude to **Dr. J. Jenix Rino M.E., M.B.A., Ph.D.,** Director for his encouragement and kind support in all our endeavors.

We are grateful to thank **Dr. R. Suresh Premil Kumar M.E., Ph.D.,** Principal for giving a source of inspiration throughout our study in the college.

It’s our solace to thank **Dr. F. R Shiny Malar M.Tech., Ph.D.,** HOD for her continuous encouragement and sustained interest in completion of our project.

We extend our thanks to **Mrs. V. Subitha M.E.,** our internal guide for his inspiring guidance and valuable advice in doing this project.

We also thank our institution and our faculty members without whom this project would have been a distant reality.

Last but not least we thank all our friends and well – wishers who helped us during the project duration.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In today's digital age, the need for transparent and secure voting systems has become increasingly critical. Traditional centralized voting systems often face challenges related to trust, security, and transparency. However, with the advent of blockchain technology, we now have the opportunity to revolutionize the way elections are conducted. This document outlines the concept and development of a decentralized voting system that leverages the power of blockchain to ensure integrity, transparency, and tamper-resistant elections.

The current voting systems suffer from issues such as voter fraud, manipulation, and lack of transparency. Centralized authorities control the entire process, leaving room for potential corruption and mistrust. To address these challenges, we propose the development of a decentralized voting system built on blockchain technology. By utilizing the inherent features of blockchain, such as immutability, decentralization, and cryptographic security, we aim to create a robust and trustworthy platform that enables fair and verifiable elections.

The decentralized voting system will be built on a blockchain platform, specifically Ethereum, which provides a solid foundation for smart contract development and execution. Smart contracts will serve as the backbone of the voting system, enabling the automation of various voting processes while ensuring transparency and security.

The decentralized voting system offers significant benefits compared to traditional centralized systems. Transparency is a fundamental aspect, as all transactions and votes recorded on the blockchain are transparent and auditable by anyone, promoting trust and eliminating the need for reliance on central authorities. The cryptographic security of blockchain ensures the integrity and immutability of voting data, significantly reducing the risk of tampering or fraudulent activities. The system's tamper resistance ensures that once votes are recorded on the blockchain, they become immutable, making it nearly impossible for external parties to manipulate or alter the results.

Key Components:

1. *Smart Contracts*: Smart contracts will be used to define the rules and logic of the voting system. They will handle tasks such as user registration, ballot creation, vote casting, and result tabulation. By executing on the blockchain, these contracts will guarantee transparency, immutability, and tamper resistance.
2. *User Authentication*: To maintain the integrity of the voting process, user authentication is crucial. Users will be authenticated using their Ethereum addresses, and MetaMask will be utilized as the wallet provider for secure account management and transaction signing.
3. *Front-end Interface*: A user-friendly front-end interface will be developed to interact with the voting system. This interface will allow users to register, cast their votes, and view election results in a transparent manner. The interface will integrate with MetaMask to enable secure transactions and seamless user experience.

One of the key advantages of a blockchain-based application is its inherent security. In a decentralized voting system, security is paramount to ensure the integrity and trustworthiness of the electoral process. This page delves into the various security measures employed by a blockchain application to safeguard the voting system.

Immutability and Data Integrity: Blockchain achieves immutability by using cryptographic hashes to link each block to the previous block in a chain. Once a block is added to the blockchain, it cannot be altered without invalidating the entire chain. In a decentralized voting system, this ensures that votes recorded on the blockchain remain tamper-proof and unchangeable, preserving the integrity of the electoral data.

Consensus Mechanisms: Consensus mechanisms are employed in blockchain to validate and agree on the state of the blockchain. In Ethereum, the most widely used consensus mechanism is Proof of Work (PoW) or Proof of Stake (PoS). These mechanisms ensure that malicious actors cannot alter the blockchain without the majority of network participants validating the changes. By relying on consensus, a decentralized voting system can maintain the security and validity of the recorded votes.

Decentralization and Fault Tolerance: A blockchain application is decentralized, meaning that it is distributed across multiple nodes or computers. This decentralized nature makes the application resilient to single points of failure and reduces the risk of a single entity controlling the voting system. In the context of a decentralized voting system, this enhances the security by eliminating the reliance on a central authority and making it difficult for malicious actors to manipulate the votes.

Cryptography and Private Key Security: Blockchain applications rely on cryptographic algorithms to secure transactions and user identities. Each user is assigned a unique private key that is used to sign transactions and authenticate their actions on the blockchain. It is crucial to educate users about the importance of securely storing their private keys. For instance, using hardware wallets or encrypted key stores can significantly enhance the security of user accounts and prevent unauthorized access.

Secure Smart Contract Development: In a decentralized voting system, the smart contracts that govern the voting processes must be developed with security in mind. It is essential to follow best practices for secure smart contract development, such as code reviews, extensive testing, and utilizing audited libraries.

Network Security: Blockchain networks, like Ethereum, have their own security measures to protect against network attacks. These measures include mechanisms like gas fees to prevent spam and Denial-of-Service (DoS) attacks.

The security of a blockchain application, such as a decentralized voting system, relies on a combination of immutability, consensus mechanisms, decentralization, cryptography, and secure development practices. By leveraging these security measures, a blockchain application can provide a high level of integrity, transparency, and tamper resistance, ensuring that the electoral process remains secure and trustworthy.

The decentralized voting system presents a transformative solution to the challenges faced by traditional voting systems. By leveraging blockchain technology, we can create a transparent, secure, and tamper-resistant platform that restores trust in the electoral process. With the use of Ethereum smart contracts and MetaMask for user authentication, we can achieve a robust system that empowers individuals and ensures fair and verifiable elections for a more democratic future.

CHAPTER 2
LITERATURE SURVEY

Title : Blockchain-Based E-Voting System, 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)

Authors: Friðrik Þ. Hjálmarsson, Gunnlaugur K. Hreiðarsson

Year : 2018

[1]Building an electronic voting system that satisfies the legal requirements of legislators has been a challenge for a long time. Distributed ledger technologies is an exciting technological advancement in the information technology world. Blockchain technologies offer an infinite range of applications benefiting from sharing economies. This paper aims to evaluate the application of blockchain as service to implement distributed electronic voting systems. The paper elicitates the requirements of building electronic voting systems and identifies the legal and technological limitations of using blockchain as a service for realizing such systems. The paper starts by evaluating some of the popular blockchain frameworks that offer blockchain as a service. We then propose a novel electronic voting system based on blockchain that addresses all limitations we discovered. More generally this paper evaluates the potential of distributed ledger technologies through the description of a case study, namely the process of an election and implementing a blockchain-based application which improves the security and decreases the cost of hosting a nationwide election.

In every democracy, the security of an election is a matter of national security. The computer security field has for a decade studied the possibilities of electronic voting systems , with the goal of minimizing the cost of having a national election, while fulfilling and increasing the security conditions of an election. From the dawn of democratically electing candidates, the voting system has been based on pen and paper. Replacing the traditional pen and paper scheme with a new election system is critical to limit fraud and having the voting process traceable and verifiable. Electronic voting machines have been viewed as flawed, by the security community, primarily based on physical security concerns. Anyone with physical access to such machine can sabotage the machine, thereby affecting all votes cast on the aforementioned machine. Enter blockchain technology. A blockchain is a distributed, immutable, incontrovertible, public ledger. This new technology works through four main features:

(i) The ledger exists in many different locations: No single point of failure in the maintenance of the distributed ledger.

(ii) There is distributed control over who can append new transactions to the ledger.

(iii) Any proposed "new block" to the ledger must reference the previous version of the ledger, creating an immutable chain from where the blockchain gets its name, and thus preventing tampering with the integrity of previous entries.

(iv) A majority of the network nodes must reach a consensus before a proposed new block of entries becomes a permanent part of the ledger.

DISADVANTAGES:

- Very difficult to use
- Cost of Implementation
- Need to hire professionals to handle the application
- Very slow
- Do no cover lot of vulnerabilities

Title : A Smart Contract for Boardroom Voting with Maximum Voter Privacy, 2017 International Conference on Financial Cryptography and Data Security

Authors: Patrick McCorry, Siamak F. Shahandashti and Feng Hao

Year : 2017

[2]The solutions achieve voter privacy with the involvement of a trusted authority. In Follow My Vote, the authority obfuscates the correspondence between the voter's real world identity and their voting key. Then, the voter casts their vote in plaintext. In TIVI, the authority is required to shuffle the encrypted votes before decrypting and counting the votes. In our work, we show that the voter's privacy does not need to rely on a central authority to decouple the voter's real world identity from their voting key, and the votes can be counted without the cooperation of a central authority. Furthermore, these solutions only use the Blockchain as an append-only and immutable global database to store the voting data. We propose that the network's consensus that secures the Blockchain can also enforce the execution of the voting protocol itself.

To date, both Bitcoin and Ethereum have inherent scalability issues. Bitcoin only supports a maximum of 7 transactions per second and each transaction dedicates 80 bytes for storing arbitrary data. On the other hand, Ethereum explicitly measures computation and storage using a gas metric, and the network limits the gas that can be consumed by its users. As deployed today, these Blockchains cannot readily support storing the data or enforcing the voting protocol's execution for national scale elections. For this reason, we chose to perform a feasibility study of a boardroom election over the Blockchain which involves a small group of voters (i.e. 40 participants) whose identities are publicly known before the voting begins. For example, a boardroom election may involve stakeholders voting to appoint a new director.

We chose to implement the boardroom voting protocol as a smart contract on Ethereum. These smart contracts have an expressive programming language and the code is stored directly on the Blockchain. Most importantly, all peers in the underlying peer-to-peer network independently run the contract code to reach consensus on its output. This means that voters can potentially not perform all the computation to verify the correct execution of the protocol. Instead, the voter can trust the consensus computing provided by the Ethereum network to enforce the correct execution of the protocol. This enforcement turns detection measures seen in publicly verifiable voting protocols into prevention measures.

We provide the first implementation of a decentralized and self-tallying internet voting protocol. The Open Vote Network is a boardroom scale voting protocol that is implemented as a smart contract in Ethereum. The Open Vote Network provides maximum voter privacy as an individual vote can only be revealed by a full-collusion attack that involves compromising all 2 other voters; all voting data is publicly available; and the protocol allows the tally to be computed without requiring a tallying authority. Most importantly, our implementation demonstrates the feasibility of using the Blockchain for decentralized and secure e-voting.

DISADVANTAGES:

- Not Suitable for High User Availability
- Does Not cover the Information Gathering phase
- Require high configuration system

Title : An efficient and effective Decentralized Anonymous Voting System, ArXiv
Authors: Wei-Jr Lai, Ja-Ling Wu

Year: 2018

[3]Although some countries have begun to use electronic voting for national scale election, there is still no suitable, trusted, efficient electronic voting system for people because it requires many contradictory properties. The election needs one or more authority for both authentication and protect the privacy of participant, however, it is difficult for voter to believe in the government or authority that will always follow the rules or never get hacked. There have been many studies and discussion on elections open all ballot to ensure transparent, then permute for anonymity, but it is still centralized and tally phase is time-consuming proposed Open Vote network protocol, it is decentralized, anonymous and transparent. But it cannot tally the result even only one voter doesn't cast his ballot, which make their scheme only suitable for small-scale voting.

With the development of blockchain, some work has discuss, electronic voting can use it as immutable, public bulletin board. Run Open Vote protocol on Ethereum smart contract, make whole process more convenient, build their voting system based on blockchain, require all voter open their mask after voting, which property cause the same problem as and it is obviously unreasonable to force each voter to pay a deposit before voting proposed a simple and efficient scheme, and introduced multiple authority to protect privacy of voter. But it is centralized and has some information that is not completely transparent.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In the existing system of Electronic Voting Machines (EVMs), limitations such as the availability of source and data for monitoring, heavyweight software requirements, cost of implementation, and time-consuming operations need to be addressed. Efforts should be made to improve data availability, optimize EVM software for lightweight systems, explore cost-effective solutions, and streamline processes to enhance accessibility, reliability, and transparency in the electoral process. By addressing these limitations, the existing EVM system can be strengthened to build trust among voters and stakeholders, ensuring a more efficient and secure voting experience.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM

- Lack of Voter Verifiability
- Security and Tampering Concerns
- Lack of Transparency
- Limited Accessibility
- Cost and Infrastructure Requirements
- Lack of Paper Trail
- Technology Obsolescence
- Limited Public Scrutiny

3.2 PROPOSED SYSTEM

The proposed blockchain-based voting system revolutionizes the traditional voting process by leveraging the advantages of blockchain technology. With enhanced transparency, increased security, and improved accessibility, the system ensures a trustworthy and tamper-resistant voting experience. By implementing robust voter authentication, automating result tabulation, and providing an immutable record of votes, the proposed system offers efficient and real-time results. Additionally, the system can potentially reduce costs associated with traditional methods while fostering trust, accountability, and public participation in the electoral process.

3.1.2 ADVANTAGES OF PROPOSED SYSTEM

- Enhanced Transparency
- Increased Security
- Improved Accessibility
- Voter Authentication
- Efficient and Real-Time Results
- Immutable Record of Votes
- Cost Savings

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

• Processor	:	Multi-core processor with a clock speed of at least 2 GHz.
• RAM	:	4GB
• Storage	:	10 GB
• Keyboard	:	Standard Keyboard
• Mouse	:	Standard Mouse
• Monitor	:	Standard Monitor

4.2 SOFTWARE REQUIREMENTS

Operating system	:	Windows, macOS, or Linux.
Language	:	Solidity, JavaScript(NodeJS)
Development Framework	:	Truffle Suite
Local Development EVM based Blockchain	:	Ganache-CLI
Interfacing	:	Web3.js
Block Explorer	:	Ethernal
Wallet	:	Metamask

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

A system architecture diagram is a visual representation of a system's components and how they interact with each other. System architecture diagrams can be used to Communicate the system's architecture to stakeholders and developers, Identify potential problems with the system's architecture, Document the system's architecture for future reference, Plan for changes to the system's architecture.

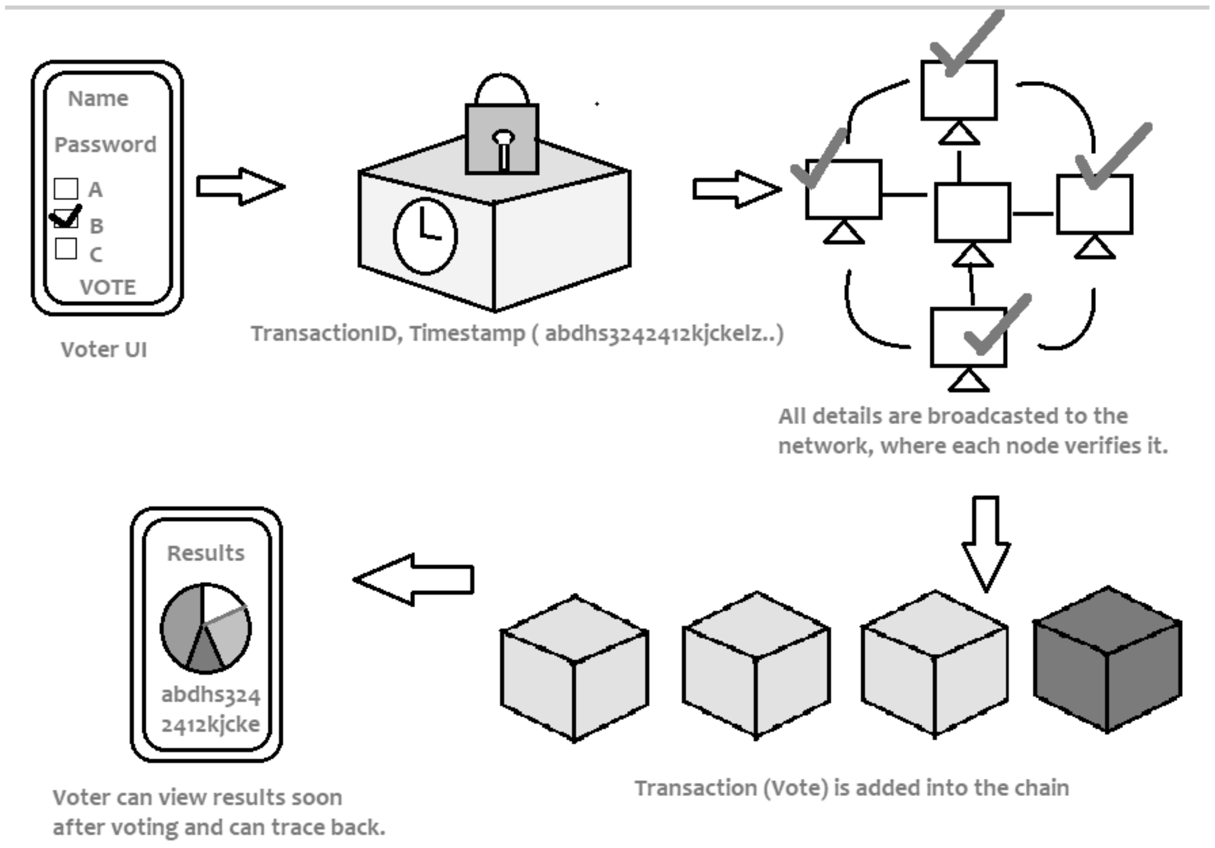

Name
Password
A
B
C
VOTE
Voter UI
TransactionID, Timestamp (abdhs3242412kjckelz..)
All details are broadcasted to the network, where each node verifies it.
Results
abdhs324
2412kjcke
Voter can view results soon after voting and can trace back.
Transaction (Vote) is added into the chain

Fig 5.1 System Architecture

5.2 USECASE DIAGRAM:

A use case diagram is a graphic depiction of the interactions among the element of a system. It is usually referred to as behavior diagrams used to describe a set of action that some system or system should or can perform in collaboration with one or more external users of the system.

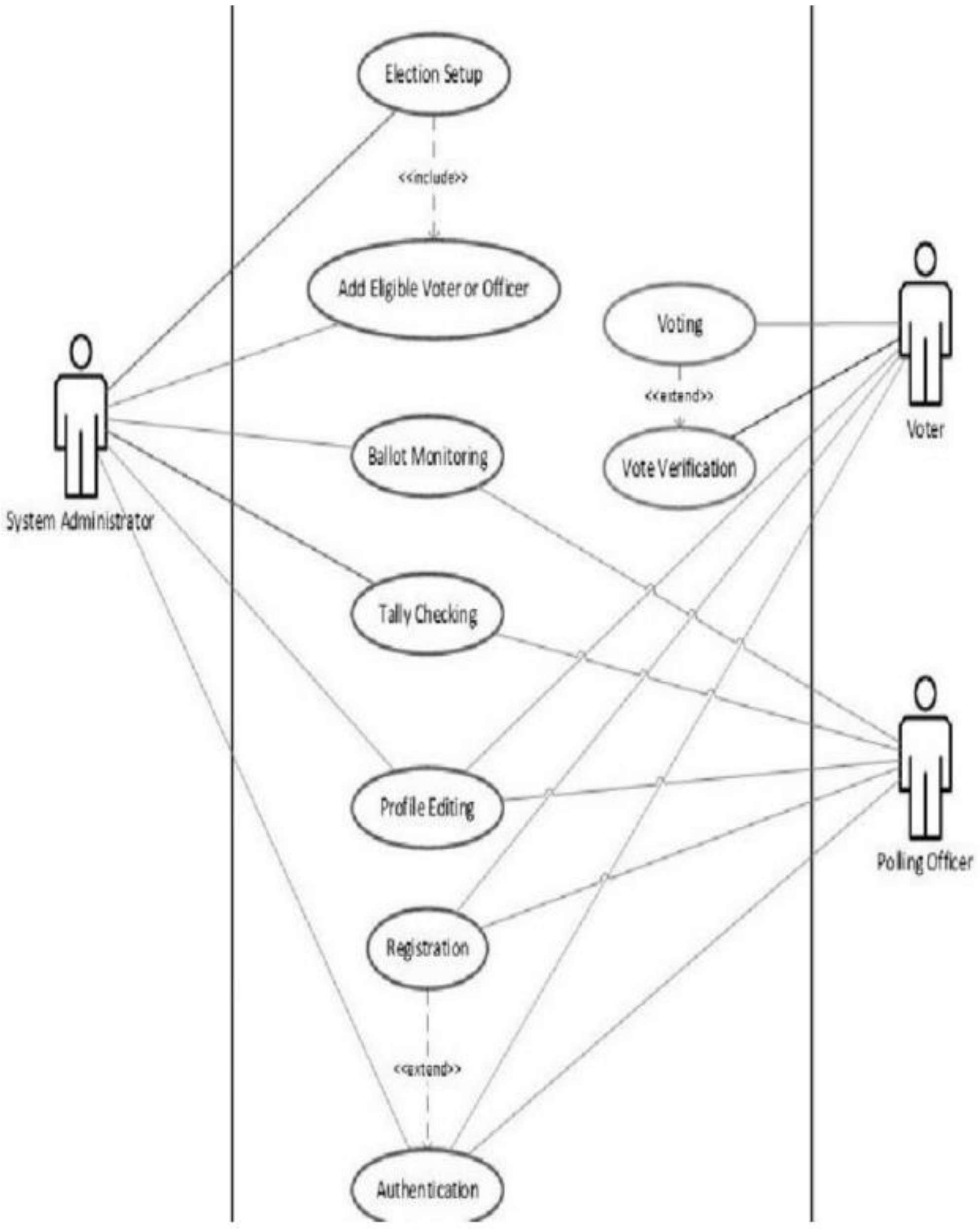


Fig 5.2 Usecase Diagram

5.3 DATA FLOW DIAGRAM:

A data flow diagram (DFD) is a graphical representation of the flow of data through an information system. It is a tool that can be used to identify and understand the components of a system, as well as the relationships between those components.

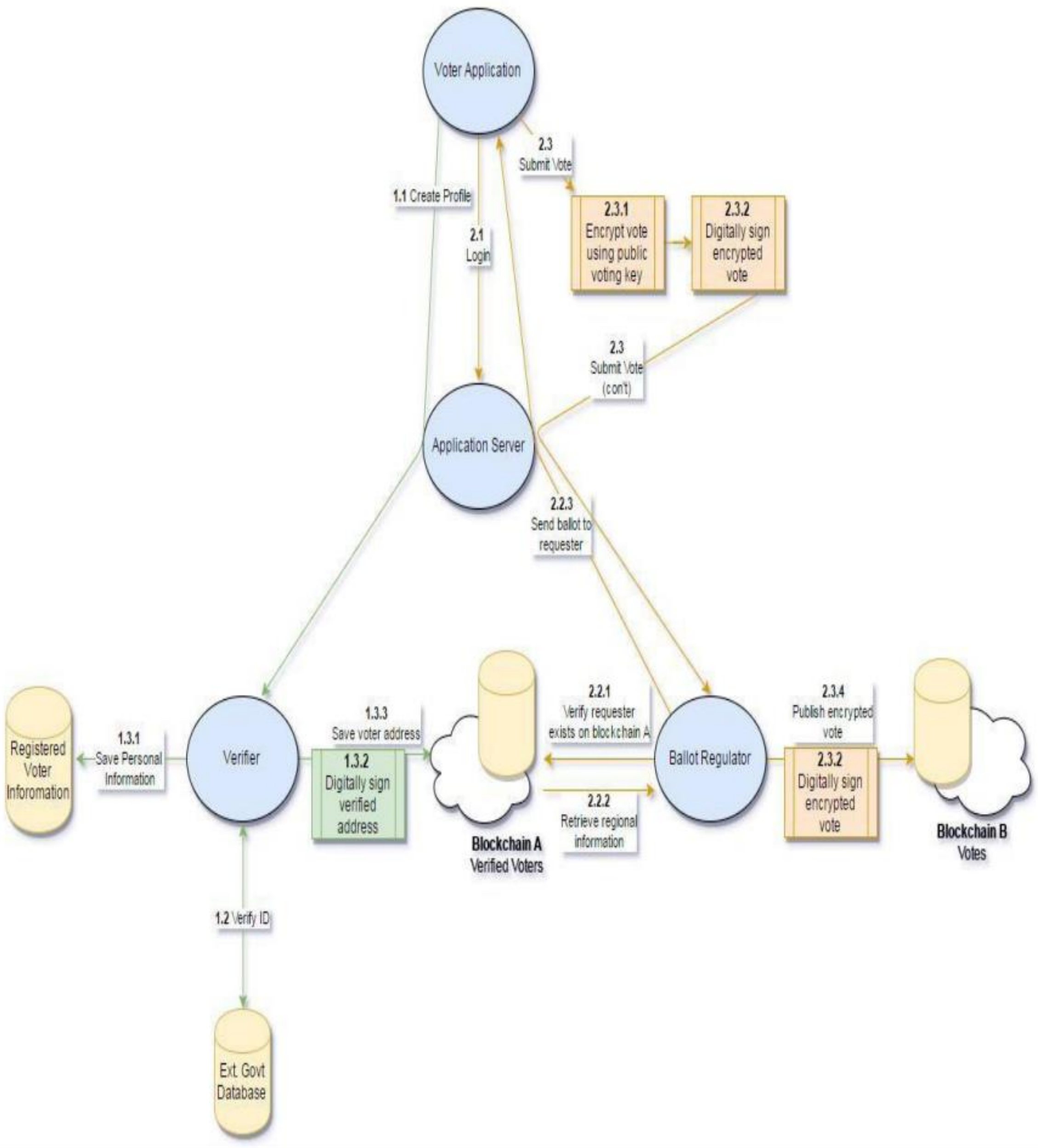

Voter Application
2.3 Submit Vote
1.1 Create Profile
2.1 Login
2.3.1 Encrypt vote using public voting key
2.3.2 Digitally sign encrypted vote
2.3 Submit Vote (con't)
Application Server
2.2.3 Send ballot to requester
Registered Voter Inforomation
1.3.1 Save Personal Information
Verifier
1.3.3 Save voter address
1.3.2 Digitally sign verified address
Blockchain A Verified Voters
2.2.1 Verify requester exists on blockchain A
2.2.2 Retrieve regional information
Ballot Regulator
2.3.4 Publish encrypted vote
2.3.2 Digitally sign encrypted vote
Blockchain B Votes
1.2 Verify ID
Ext. Govt Database

Fig 5.3 Data Flow Diagram

5.4 SEQUENCE DIAGRAM

A sequence diagram is a type of UML diagram that shows how objects interact with each other over time. It is a useful tool for understanding how a system works and for identifying potential problems..

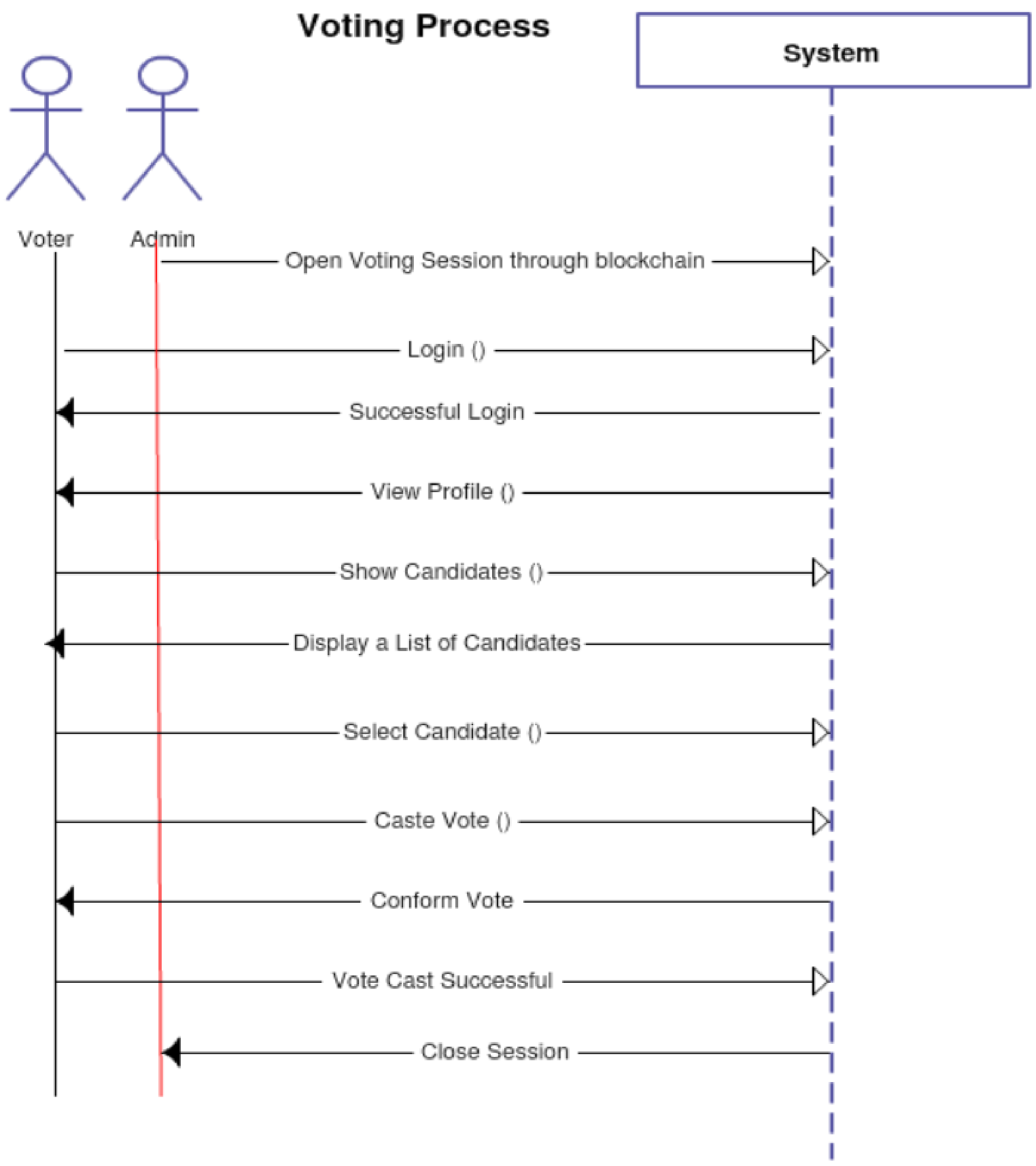


Fig 5.4 Sequence Diagram

CHAPTER 6
TOOLS USED IN PROJECT

In the implementation of the decentralized voting system using blockchain technology, various tools and technologies were utilized to create a functional and user-friendly solution. The primary tools employed in this project include Solidity, Truffle Suite, Ganache-CLI, MetaMask, Web3.js, HTML, CSS, Bootstrap, jQuery, and a tool called Ethernal as a local block explorer..

6.1 Solidity:

Solidity is a programming language specifically designed for writing smart contracts on blockchain platforms like Ethereum. It is one of the most widely used languages for developing decentralized applications and plays a crucial role in the implementation of the decentralized voting system. Solidity provides a secure and expressive syntax that allows developers to define the behavior and logic of smart contracts governing the voting process.

Smart contracts, written in Solidity, are self-executing agreements with the terms of the agreement directly written into code. In the context of the decentralized voting system, smart contracts are responsible for enforcing the voting rules, managing the storage and manipulation of voting data, and ensuring the transparency and integrity of the voting process. Smart contracts define the structure of the ballots, handle the casting and counting of votes, and facilitate the retrieval and presentation of voting results.

Solidity offers a range of features and functionalities that support the development of robust and secure smart contracts. It provides data types, control structures, and libraries for handling cryptographic operations, ensuring the confidentiality and authenticity of voting data.

Furthermore, Solidity comes with a comprehensive development environment, including tools like the Solidity compiler and various integrated development environments (IDEs) with Solidity plugins. These tools enable developers to write, test, and debug smart contracts efficiently, ensuring their correctness and adherence to the desired specifications.

By leveraging Solidity and smart contracts, the decentralized voting system can benefit from the transparency, immutability, and security provided by blockchain technology. Solidity facilitates the development of trustworthy and auditable smart contracts, ensuring that the voting process remains tamper-proof and resilient against manipulation.

6.2 Truffle Suite:

Truffle Suite is a powerful set of development tools specifically designed for building, testing, and deploying decentralized applications (DApps) on the Ethereum blockchain. It offers a comprehensive suite of tools that streamline the development process and provide a robust framework for creating and managing smart contracts.

One of the key components of Truffle Suite is Truffle, which is a development environment and testing framework for Ethereum. Truffle provides a command-line interface (CLI) that allows developers to compile, deploy, and test their smart contracts. It simplifies the process of contract deployment, migration, and interaction, making it easier to manage the lifecycle of smart contracts.

Truffle Suite also includes Ganache, a personal blockchain for development and testing purposes. Ganache allows developers to create a local Ethereum network that mimics the behavior of the main Ethereum network, but in a controlled and isolated environment. It provides a set of pre-funded accounts, which can be used for testing smart contracts without incurring any real transaction costs.

Additionally, Truffle Suite offers tools for automated testing of smart contracts. It integrates with testing frameworks like Mocha and Chai, allowing developers to write and execute unit tests, integration tests, and end-to-end tests for their smart contracts. This ensures the reliability and correctness of the contracts, enabling developers to identify and fix any issues or vulnerabilities before deploying them to the production network.

Truffle Suite's development tools also include Drizzle, which is a collection of front-end libraries that simplify the integration of Ethereum functionality into web applications. It provides an easy-to-use API for interacting with smart contracts and retrieving data from the blockchain.

Overall, Truffle Suite provides a comprehensive and user-friendly development environment for building decentralized applications on the Ethereum blockchain. Its suite of tools simplifies the development, testing, and deployment processes, enabling developers to create secure and robust smart contracts and DApps more efficiently..

6.3 Ganache-CLI:

Ganache-CLI, part of the Ganache suite of tools, is a command-line interface that provides a local Ethereum blockchain environment for development and testing purposes. It allows developers to create a private blockchain network on their local machine, providing a sandboxed environment to deploy and test smart contracts without incurring any real transaction costs

Ganache-CLI offers several key features that make it a valuable tool for the project. Firstly, it provides a set of pre-configured accounts with pre-funded Ether, making it easy to simulate real-world scenarios and test various functionalities of the decentralized voting system. Developers can interact with these accounts, simulate transactions, and observe the state changes on the local blockchain.

Moreover, Ganache-CLI offers flexibility and control over the blockchain environment. Developers can customize the blockchain network by adjusting parameters such as gas limits, block times, and network ID. This allows for the simulation of specific network conditions, ensuring that the decentralized voting system can handle various scenarios and edge cases effectively.

Ganache-CLI also provides powerful logging and debugging capabilities. It records detailed information about each transaction, including gas usage, contract events, and stack traces. These logs facilitate the identification and resolution of any issues or errors during development and testing.

6.4 MetaMask:

MetaMask is a popular browser extension that serves as a digital wallet and Ethereum interface, enabling users to interact with decentralized applications (DApps) on the Ethereum blockchain. It is a vital tool used in the project for providing a seamless and secure way for users to participate in the decentralized voting system.

MetaMask acts as a bridge between the web browser and the Ethereum network. It allows users to create Ethereum accounts, securely store private keys, and manage their cryptocurrency holdings. Users can import existing accounts or create new ones directly within MetaMask, providing a convenient and user-friendly experience.

One of the key features of MetaMask is its ability to seamlessly integrate with web applications. Once installed, it injects a JavaScript object into the web page, enabling DApps to interact with the user's Ethereum account.

MetaMask provides a user interface that allows users to review and approve transactions before they are sent to the Ethereum network. Additionally, MetaMask provides transaction history and account balance information, enabling users to keep track of their voting activities.

Furthermore, MetaMask supports multiple Ethereum networks, allowing users to switch between test networks (such as the local Ganache network) and the Ethereum mainnet. This flexibility is essential for developers and users to test and deploy the decentralized voting system in different environments and configurations.

Overall, MetaMask plays a critical role in the project by providing a user-friendly and secure interface for users to interact with the decentralized voting system. Its integration with web browsers and support for multiple Ethereum networks simplify the user experience and ensure compatibility with various testing and deployment scenarios. By leveraging MetaMask, the decentralized voting system can provide a seamless and intuitive voting experience while maintaining the privacy and security of user interactions on the Ethereum blockchain..

6.5 Web3.js:

Web3.js is a powerful JavaScript library that serves as a bridge between decentralized applications (DApps) and the Ethereum blockchain. It provides a comprehensive set of functionalities for interacting with smart contracts, querying blockchain data, and managing transactions.

One of the main features of Web3.js is its ability to connect DApps to the Ethereum network. It allows developers to establish a connection to a specific Ethereum node. Web3.js provides APIs to send transactions, read data from the blockchain, and listen to events emitted by smart contracts.

Web3.js simplifies the process of interacting with smart contracts by providing an abstraction layer that handles the low-level details of contract interaction. It allows developers to easily instantiate contract instances, call contract functions, and retrieve contract data using JavaScript. Web3.js also handles the creation and signing of transactions, making it convenient for developers to send transactions to the Ethereum network.

In addition to smart contract interaction, Web3.js also enables developers to work with blockchain data. It provides methods to query account balances, retrieve transaction details, and access other relevant blockchain information. This allows DApps to display real-time data from the Ethereum network and provide users with up-to-date information about their interactions with the decentralized voting system.

Furthermore, Web3.js supports the integration of decentralized identity and authentication mechanisms, such as Ethereum-based wallets, into DApps. It allows users to securely sign transactions and interact with smart contracts using their Ethereum accounts, ensuring the authenticity and security of their actions within the decentralized voting system.

Overall, Web3.js plays a crucial role in the project by providing the necessary tools and APIs to connect the decentralized voting system with the Ethereum blockchain. Its comprehensive set of functionalities simplifies smart contract interaction, blockchain data retrieval, and transaction management, making it an essential tool for developing robust and user-friendly DApps on the Ethereum platform.

6.6 HTML, CSS. JS, JQuery and Bootstrap:

The combination of HTML, CSS, JavaScript (JS), jQuery, and Bootstrap forms the core set of tools used in the project for developing the user interface and enhancing the overall user experience of the decentralized voting system.

HTML (Hypertext Markup Language) provides the structural foundation of web pages. It is used to create the layout and structure of the user interface components. HTML tags define the various elements, such as headings, buttons, forms, and tables, that make up the interface for users to interact with the decentralized voting system.

CSS (Cascading Style Sheets) is used to enhance the visual presentation and styling of the user interface. CSS allows developers to define colors, fonts, spacing, and other visual aspects, ensuring a cohesive and visually appealing design. It provides flexibility in customizing the appearance of the decentralized voting system to align with the project's branding and user interface requirements.

JavaScript (JS) adds interactivity and dynamic behavior to the web pages of the decentralized voting system. JS enables developers to implement functionalities such as form validation, real-time updates, and event handling. With JS, user interactions can trigger actions like submitting votes, updating vote counts, and displaying notifications, enhancing the overall user experience.

jQuery is a fast and concise JavaScript library that simplifies many common tasks and interactions in web development. It provides a range of pre-built functions and methods, making it easier to manipulate HTML elements, handle events, and perform AJAX requests. jQuery's intuitive syntax and powerful features enhance the development process and contribute to the smooth functioning of the decentralized voting system.

Bootstrap is a popular front-end framework that offers a collection of CSS and JS components, as well as responsive design utilities. It provides ready-to-use templates, grids, navigation bars, and other UI elements that facilitate the creation of a responsive and visually consistent user interface. Bootstrap streamlines the development process by offering a responsive and mobile-friendly layout out of the box.

By combining HTML, CSS, JS, jQuery, and Bootstrap, developers can create a visually appealing, interactive, and user-friendly interface for the decentralized voting system. These tools work together to ensure a seamless user experience, efficient development process, and consistent design across different devices and browsers.

6.7 Ethernal:

Ethernal is a newly introduced tool in the project that serves as a local block explorer specifically designed for the locally running Ganache blockchain. It provides a convenient and user-friendly interface to explore and analyze the blockchain data generated by the decentralized voting system.

Ethernal offers a range of features that aid in understanding and visualizing the blockchain data. It provides detailed information about blocks, transactions, and smart contracts deployed on the Ganache network. Developers and users can easily track the progress of the decentralized voting system by examining the blocks and transactions in chronological order.

One of the key functionalities of Ethernal is the ability to inspect smart contracts. It allows users to view the source code, bytecode, and ABI (Application Binary Interface) of deployed smart contracts, providing transparency and visibility into the underlying logic of the decentralized voting system. This feature enables developers to verify the integrity and correctness of the smart contracts used in the voting process.

Ethernal also provides visualization tools that enhance the understanding of the blockchain data. It offers graphical representations, such as charts and graphs, to depict the distribution of votes, historical trends, and other relevant statistics. These visualizations facilitate the analysis of the voting system's performance, patterns, and any potential anomalies

Additionally, Ethernal includes advanced search and filtering capabilities, allowing users to search for specific transactions, addresses, or smart contracts. This feature streamlines the process of locating specific information within the blockchain data, making it easier to track and audit the voting activities and results.

CHAPTER 7
SYSTEM IMPLEMENTATION

7.1 MODULES:

The modules that are used in this project given below:

- User Management
- Voting Management
- Blockchain Integration
- Ballot Generation
- Vote Casting
- Vote Counting and Results
- Audit and Transparency
- Security and Privacy

7.2 MODULE DESCRIPTION

User Management:

The User Management module handles user registration, authentication, and authorization for the decentralized voting system. It allows users to create accounts, log in securely, and manage their profiles. User authentication ensures that only authorized individuals can access the voting system and participate in the elections. The module also includes features to validate user eligibility based on criteria such as age, citizenship, or other requirements set for voters.

Voting Management:

The Voting Management module focuses on administering the voting process within the decentralized system. It enables the creation and configuration of elections, including specifying the duration, voting rules, and candidate lists. This module allows administrators to set up multiple elections concurrently and provides functionalities to monitor the progress of each election, ensuring transparency and accountability throughout the voting period.

Blockchain Integration:

The Blockchain Integration module is responsible for connecting the decentralized voting system to a blockchain network. It involves integrating with a specific blockchain platform, such as Ethereum, and deploying smart contracts that govern the voting process. This module ensures the secure storage of voting data on the blockchain, immutability of votes, and transparency of the voting process through public verification of transactions and results.

Ballot Generation:

The Ballot Generation module generates unique ballots or voting tokens for eligible voters. It ensures that each voter receives a unique identifier that allows them to cast their vote. The module manages the secure delivery of ballots to voters, ensuring that only authorized individuals receive access to their specific voting token, thereby preventing duplicate voting and maintaining the integrity of the voting process.

Additionally, the Ballot Generation module implements a reliable distribution mechanism to securely deliver the generated ballots to the respective voters. It employs encryption techniques and secure channels to prevent unauthorized access and interception of the ballots, safeguarding the privacy and confidentiality of the voting tokens during transit.

Vote Casting:

The Vote Casting module enables voters to cast their votes securely and anonymously. It provides user interfaces through which voters can view the candidates, select their preferred choices, and submit their votes. This module validates the eligibility of voters, ensuring they meet the necessary criteria to participate in the election. It then records the cast votes on the blockchain, maintaining the privacy and anonymity of the voters while preserving the transparency and integrity of the voting process.

Vote Counting and Results:

The Vote Counting and Results module is responsible for tallying and calculating the voting results. It retrieves the recorded votes from the blockchain, applies any specific voting algorithms or rules, and generates accurate results. This module ensures that the results are transparent and verifiable, providing stakeholders with an auditable trail of the voting process. It may also include real-time result display or provide aggregated results after the voting period ends.

Audit and Transparency:

The Audit and Transparency module focuses on ensuring the integrity and transparency of the decentralized voting system. It includes functionalities to audit the system's behavior, track any modifications or tampering attempts, and provide transparency to stakeholders and election observers. This module enables independent verification of the voting process and helps build trust in the system's reliability and fairness. The module incorporates cryptographic techniques, such as digital signatures and hash functions, to ensure the immutability and integrity of the audit logs. This prevents unauthorized modifications or tampering of the recorded information, providing a verifiable and tamper-proof trail of events.

Security and Privacy:

The Security and Privacy module ensures the security and privacy of the decentralized voting system. It includes measures to protect the system against cyber threats, secure the transmission and storage of sensitive data, and enforce privacy protocols to maintain voter confidentiality. This module incorporates encryption, secure communication channels, access controls, and other security measures to safeguard the integrity and privacy of the voting process, building trust and confidence among the users and stakeholders.

By incorporating these security and privacy measures, the module safeguards the decentralized voting system against potential threats, protects sensitive data from unauthorized access, and ensures the confidentiality and privacy of voters. These measures not only instill confidence among the system's users but also adhere to legal and regulatory requirements for secure and private voting processes.

CHAPTER 8
SYSTEM TESTING

8.1 SYSTEM TESTING

System testing is a phase in the software development lifecycle that aims to assess the overall functionality, performance, and reliability of the decentralized voting system as a whole. It involves testing the integrated system to ensure that it meets the specified requirements and functions correctly in various real-world scenarios.

During system testing, the decentralized voting system is evaluated against the defined functional and non-functional requirements. This testing phase focuses on validating the system's behavior and performance in different usage scenarios, including normal, peak, and stress conditions. It also involves testing the system's response to invalid inputs, error handling, and security vulnerabilities.

System testing encompasses a wide range of tests, including functional testing, usability testing, performance testing, security testing, and compatibility testing. Functional testing verifies that all the system's functionalities work as expected and meet the specified requirements. Usability testing assesses the system's user-friendliness and ensures that it is intuitive and easy to navigate for voters. Performance testing evaluates the system's speed, scalability, and responsiveness under varying workloads. Security testing focuses on identifying and mitigating potential vulnerabilities and ensuring the system's protection against unauthorized access and data breaches. Compatibility testing verifies the system's compatibility with different platforms, browsers, and devices to ensure a consistent user experience across various environments.

By conducting thorough system testing, any defects, errors, or inconsistencies can be identified and resolved, ensuring that the decentralized voting system is reliable, secure, and performs optimally.

8.2 White Box Testing

White box testing, also known as clear box testing or structural testing, is a testing technique that examines the internal structure and logic of the decentralized voting system. It involves analyzing the system's code, architecture, and design to ensure that it functions correctly and adheres to coding standards and best practices.

During white box testing, testers have access to the system's internal components, such as source code, APIs, and databases. They use this knowledge to design test cases that target specific paths, conditions, and variables within the system. This approach allows for a detailed examination of the system's internal workings and helps identify potential issues, such as logic errors, improper error handling, or code vulnerabilities.

White box testing techniques include statement coverage, branch coverage, path coverage, and condition coverage. These techniques aim to exercise different parts of the code to ensure that all possible paths and conditions are tested. By examining the system from within, white box testing provides insights into the system's structure, control flow, and data flow, enabling the identification of defects and potential optimizations.

White box testing plays a vital role in ensuring the quality and reliability of the decentralized voting system. By examining the system's internal components, testers can verify the accuracy of calculations, validate data transformations, and identify any coding errors or vulnerabilities. It helps improve the overall code quality, maintainability, and performance of the system, leading to a more robust and secure voting solution.

However, it's important to note that white box testing should not be the sole testing approach used. It is most effective when combined with other testing techniques, such as black box testing (which focuses on the system's external behavior), to provide comprehensive test coverage and ensure the integrity and functionality of the decentralized voting system.

8.3 Black Box Testing

Black box testing is a testing approach that focuses on evaluating the decentralized voting system's external behavior without considering its internal structure or implementation details. Testers perform black box testing without any knowledge of the system's internal workings, treating it as a "black box" where only inputs and outputs are considered.

During black box testing, the system is tested based on its functional requirements and specifications. Testers design test cases that cover different scenarios, input combinations, and user interactions to validate the system's behavior and ensure that it meets the intended functionality. This approach helps assess the system's usability, functionality, and compliance with user expectations.

Black box testing techniques include equivalence partitioning, boundary value analysis, error guessing, and exploratory testing. These techniques enable testers to identify common input values, edge cases, and potential errors or anomalies that the system might encounter during real-world usage.

Black box testing is particularly valuable in ensuring that the decentralized voting system works as expected from an end-user perspective. It helps identify any discrepancies between the system's actual behavior and the intended requirements or specifications. By simulating various user interactions and inputs, testers can detect errors, usability issues, and functional defects, ensuring a reliable and user-friendly voting experience.

While black box testing does not provide insights into the system's internal structure or code quality, it plays a critical role in validating the system's external behavior, identifying user-centric issues, and ensuring the system's overall reliability and conformance to requirements. When combined with other testing approaches, such as white box testing, it provides a comprehensive evaluation of the decentralized voting system, enabling the identification and resolution of potential defects and enhancing its overall quality.

8.4 Unit Testing

Integration testing is a critical phase in the software development lifecycle that focuses on verifying the interactions between different components, modules, or subsystems of a system. It ensures that these components work seamlessly together and effectively communicate with each other as intended.

During integration testing, the decentralized voting system's individual components, such as smart contracts, user interfaces, and backend services, are combined and tested as a whole. The objective is to identify and resolve any issues or discrepancies that may arise from the integration process, ensuring that the system functions correctly and meets the desired requirements.

Integration testing involves various types of tests, including functional testing, where the system's functionalities are validated in different usage scenarios, and data flow testing, which ensures the smooth flow of data between different components. It also includes compatibility testing to verify the system's compatibility with various browsers, operating systems, and devices.

The testing process involves creating test cases, test data, and test environments that simulate real-world scenarios. The system is subjected to both positive and negative test cases to validate its behavior in different conditions. Integration testing helps uncover defects that may arise due to the interactions between components, such as data inconsistencies, communication failures, or functional conflicts.

By conducting comprehensive integration testing, potential issues can be identified early in the development cycle, allowing for timely resolution and ensuring the reliability and stability of the decentralized voting system. This testing phase plays a crucial role in validating the system's overall functionality, performance, security, and interoperability, ultimately contributing to the successful deployment of a robust and efficient voting solution.

8.5 Integration Testing

Integration testing is a testing approach that focuses on evaluating the interactions and interoperability between different components or modules of the decentralized voting system. The purpose of integration testing is to ensure that the integrated components function as intended and work seamlessly together.

During integration testing, individual components are combined and tested as a group to validate their interactions, data flow, and communication. The goal is to identify any integration issues or defects that may arise from the combination of these components. This testing phase helps ensure that the decentralized voting system behaves as a cohesive unit and that data is accurately exchanged between different modules.

Integration testing can be performed using various strategies, such as top-down, bottom-up, or a combination of both. In top-down integration testing, higher-level components are tested first, with stubs or simulated lower-level components used to simulate their behavior. In bottom-up integration testing, lower-level components are tested first, with drivers or simulated higher-level components used to facilitate the testing process.

The focus of integration testing is on verifying the proper integration and functioning of interfaces, data consistency, error handling, and the overall system behavior when multiple components are working together. It helps uncover defects related to communication protocols, compatibility issues, interface mismatches, or incorrect data transformation.

By conducting thorough integration testing, the decentralized voting system can be validated for its overall functionality, stability, and reliability. This testing phase plays a vital role in ensuring that all components of the system work together seamlessly, promoting a smooth and error-free voting process. Integration testing enhances the confidence in the system's interoperability.

8.6 Validation Testing

Validation testing, also known as acceptance testing or user acceptance testing (UAT), focuses on evaluating the decentralized voting system's compliance with user requirements and ensuring that it meets the intended purpose and expectations of its stakeholders. The primary goal of validation testing is to verify that the system satisfies the specified functional and non-functional requirements and that it is ready for deployment and actual usage.

During validation testing, the decentralized voting system is evaluated from a user's perspective. Testers, often representing end-users or stakeholders, perform various test scenarios and real-world use cases to validate the system's behavior and confirm its suitability for the intended purpose. The testing process involves executing typical voting scenarios, validating system responses, and assessing user-friendliness, accuracy, and overall satisfaction.

Validation testing ensures that the system meets the defined acceptance criteria and aligns with user expectations. It validates that the decentralized voting system provides the desired functionalities, accurately processes votes, protects voter privacy and security, and offers an intuitive user interface. Additionally, it may involve assessing system performance, scalability, and compatibility with different platforms or devices.

The successful completion of validation testing indicates that the decentralized voting system is ready for deployment and meets the requirements and needs of its intended users. It provides confidence to stakeholders that the system is reliable, effective, and suitable for facilitating fair and transparent elections. The feedback and insights gained during validation testing can also inform further enhancements or refinements to ensure the system's continued success and user satisfaction.

8.7 Strategic approach to Software Testing

A strategic approach to software testing involves a systematic and planned methodology to ensure the quality, reliability, and effectiveness of the decentralized voting system. It encompasses a comprehensive set of testing activities that are aligned with the project's objectives, stakeholders' expectations, and industry best practices.

In a strategic approach, testing is not viewed as a standalone activity but as an integral part of the software development lifecycle. It begins early in the development process and continues throughout, ensuring that quality is built into the system from the start. The approach includes defining clear testing objectives, establishing a well-defined test strategy, and creating a test plan that outlines the testing scope, activities, and timelines.

A strategic approach to software testing also involves selecting appropriate testing techniques, tools, and methodologies based on the specific requirements of the decentralized voting system. It includes a combination of different testing types, such as functional testing, integration testing, performance testing, security testing, and usability testing, to ensure comprehensive coverage and validation.

Furthermore, the strategic approach emphasizes the importance of test automation to improve efficiency and scalability. It involves identifying suitable areas for test automation, designing reusable test cases, and implementing automation frameworks that enable efficient test execution, reporting, and maintenance.

Another critical aspect of a strategic approach to software testing is the establishment of clear communication and collaboration channels among the development team, testers, and stakeholders. This ensures that testing activities are aligned with the project goals, and any issues or defects are identified, reported, and resolved in a timely manner.

8.8 Test Case Matrix:

A test case matrix, also known as a test coverage matrix or test traceability matrix, is a tool used in software testing to establish a relationship between test cases and requirements or other relevant artifacts. It provides a structured and organized overview of the testing efforts by mapping test cases to specific functional requirements, use cases, or system components. The test case matrix helps ensure comprehensive test coverage by identifying which requirements have been tested, which test cases are associated with each requirement, and the status or results of each test case. It serves as a valuable reference for tracking the progress of testing activities, identifying any gaps in test coverage, and ensuring that all specified requirements are adequately addressed during the testing process.

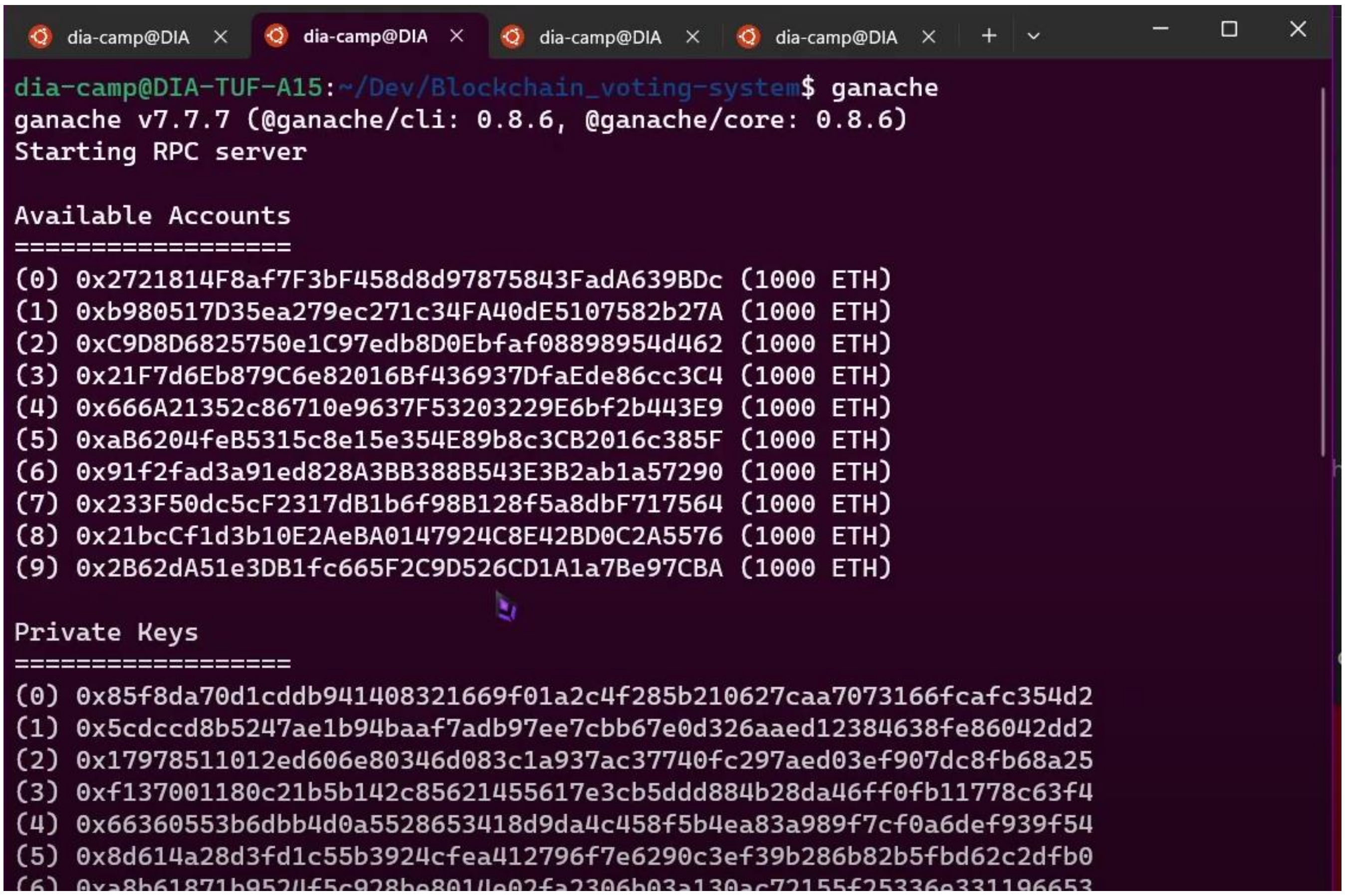
A test case matrix is typically structured in a tabular format, with test cases listed in rows and requirements or other relevant artifacts listed in columns. Each intersection of a test case and a requirement represents the association between the two. The matrix can also include additional columns to track the status or results of each test case, indicating whether it has been executed, passed, failed, or is yet to be executed.

By using a test case matrix, testers and stakeholders can easily identify the coverage of test cases across different requirements or system components. It helps ensure that all critical functionalities and scenarios are adequately tested, reducing the risk of overlooking any crucial aspects of the software. Moreover, it facilitates traceability and impact analysis, enabling the team to assess the potential impact of changes to requirements or system components on the existing test cases and ensuring proper regression testing.

Test Case	**Expected Results**	**Actual Result**	**Pass/Fail**
Login In (Positive Match)	Should positively identify a user and redirect to a dashboard with specific rights that are appended to the user.	Positively identifies a user and redirects to voting page with information about the user. The user also is granted specific rights depending on level of access.	Pass
Login In (Negative Match)	Should decline login request and display a reason why the request was declined.	Declines login request and displays the reason why the request was declined.	Pass
Add candidate	Should be able to add candidate into the voting process	Candidate details are displayed in the voting page where voters can choose.	Pass
Vote	Should able to cast a vote for the preferred candidate.	Voters select their preferred candidate and click on vote. That candidates vote count increases by 1	Pass
Check results	Should accurately show the tally of all the votes casted	Once someone casts their vote they are taken to the results page which shows a real time vote tally	Pass
Can’t vote twice	Should prevent users from casting a second vote once they have already voted.	The select candidate option and vote button disappears once a candidate has casted their vote.	Pass

CHAPTER 9

SCREENSHOTS



```
dia-camp@DIA-TUF-A15:~/Dev/Blockchain_voting-system$ ganache
ganache v7.7.7 (@ganache/cli: 0.8.6, @ganache/core: 0.8.6)
Starting RPC server

Available Accounts
==================
(0) 0x2721814F8af7F3bF458d8d97875843FadA639BDc (1000 ETH)
(1) 0xb980517D35ea279ec271c34FA40dE5107582b27A (1000 ETH)
(2) 0xC9D8D6825750e1C97edb8D0Ebfaf08898954d462 (1000 ETH)
(3) 0x21F7d6Eb879C6e82016Bf436937DfaEde86cc3C4 (1000 ETH)
(4) 0x666A21352c86710e9637F53203229E6bf2b443E9 (1000 ETH)
(5) 0xaB6204feB5315c8e15e354E89b8c3CB2016c385F (1000 ETH)
(6) 0x91f2fad3a91ed828A3BB388B543E3B2ab1a57290 (1000 ETH)
(7) 0x233F50dc5cF2317dB1b6f98B128f5a8dbF717564 (1000 ETH)
(8) 0x21bcCf1d3b10E2AeBA0147924C8E42BD0C2A5576 (1000 ETH)
(9) 0x2B62dA51e3DB1fc665F2C9D526CD1A1a7Be97CBA (1000 ETH)

Private Keys
==================
(0) 0x85f8da70d1cddb941408321669f01a2c4f285b210627caa7073166fcafc354d2
(1) 0x5cdccd8b5247ae1b94baaf7adb97ee7cbb67e0d326aaed12384638fe86042dd2
(2) 0x17978511012ed606e80346d083c1a937ac37740fc297aed03ef907dc8fb68a25
(3) 0xf137001180c21b5b142c85621455617e3cb5ddd884b28da46ff0fb11778c63f4
(4) 0x66360553b6dbb4d0a5528653418d9da4c458f5b4ea83a989f7cf0a6def939f54
(5) 0x8d614a28d3fd1c55b3924cfea412796f7e6290c3ef39b286b82b5fbd62c2dfb0
(6) 0xa8b61871b9524f5c928be8014e02fa2306b03a130ac72155f25336e331196653
```

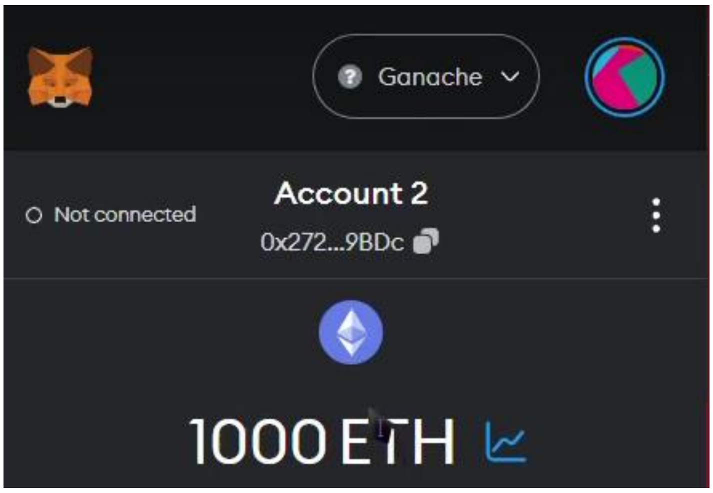
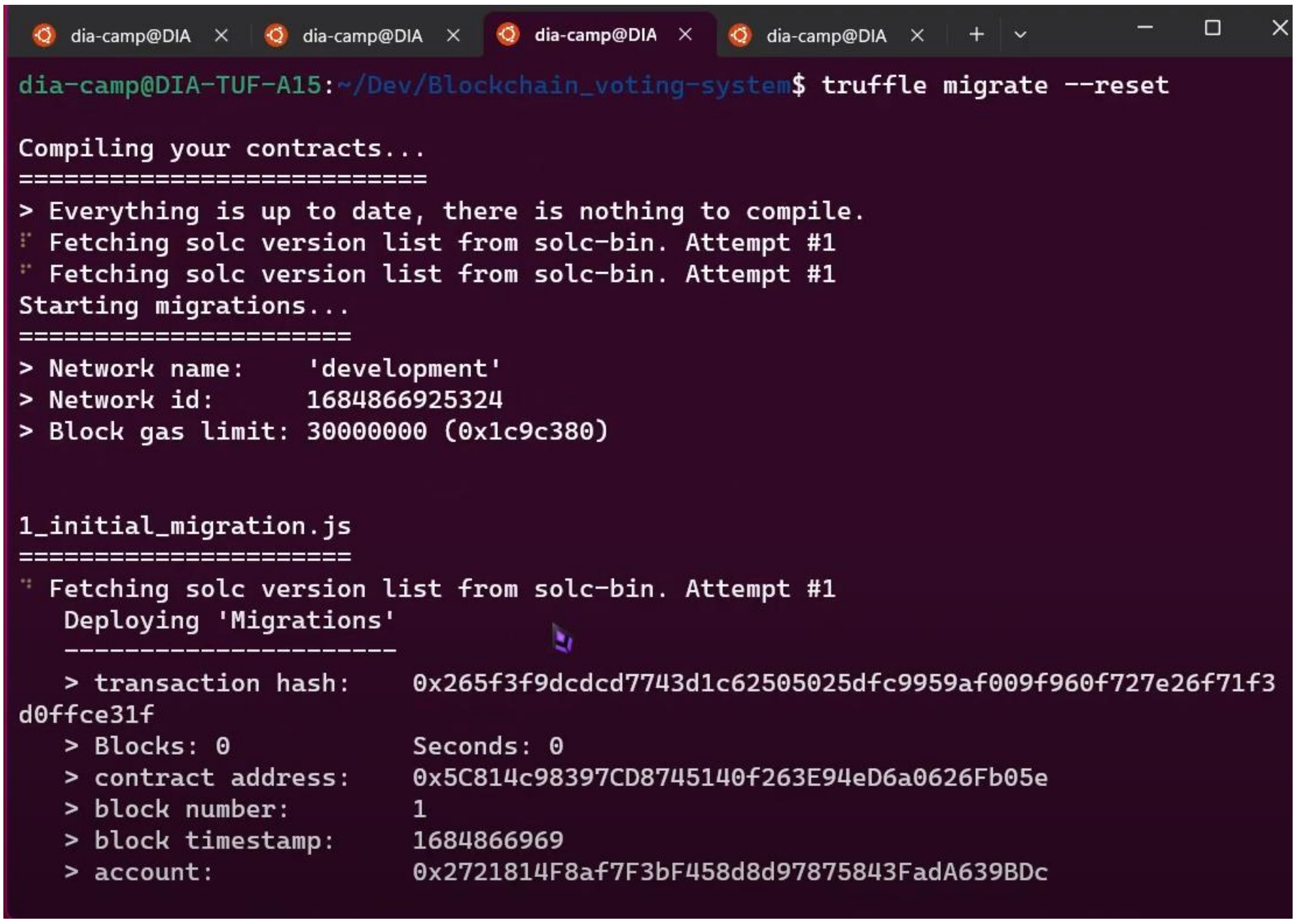
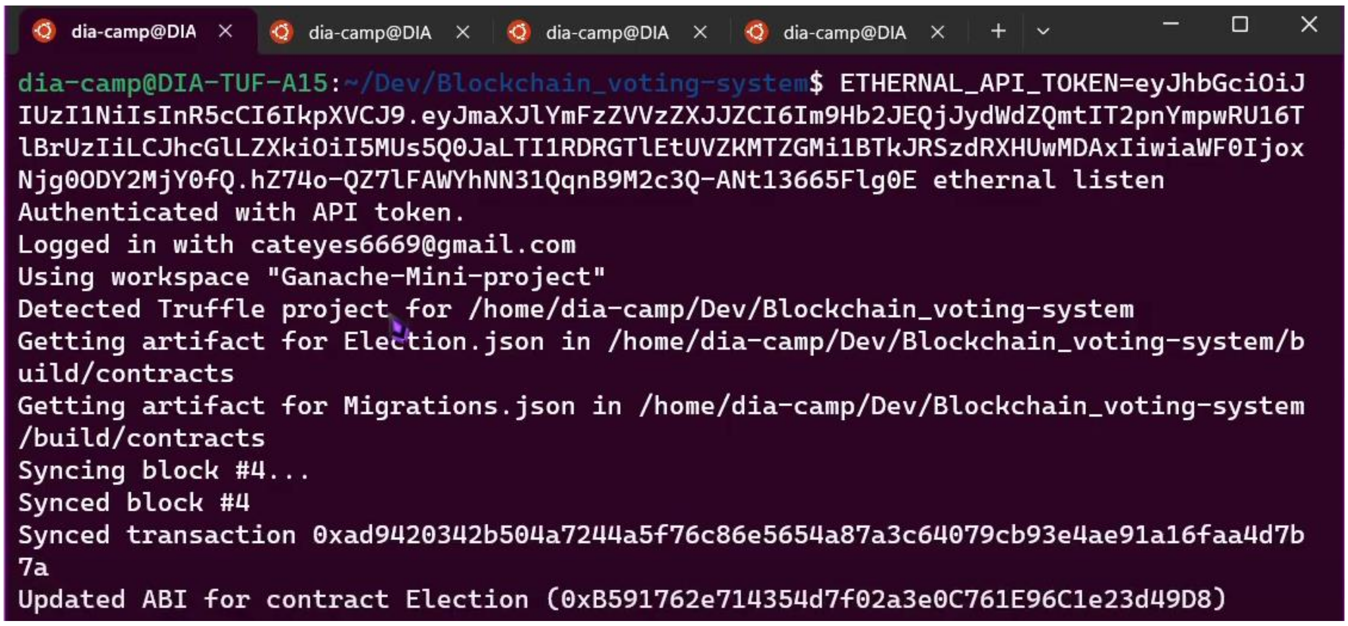


Fig 9.1 Starting Ganache and Importing Account



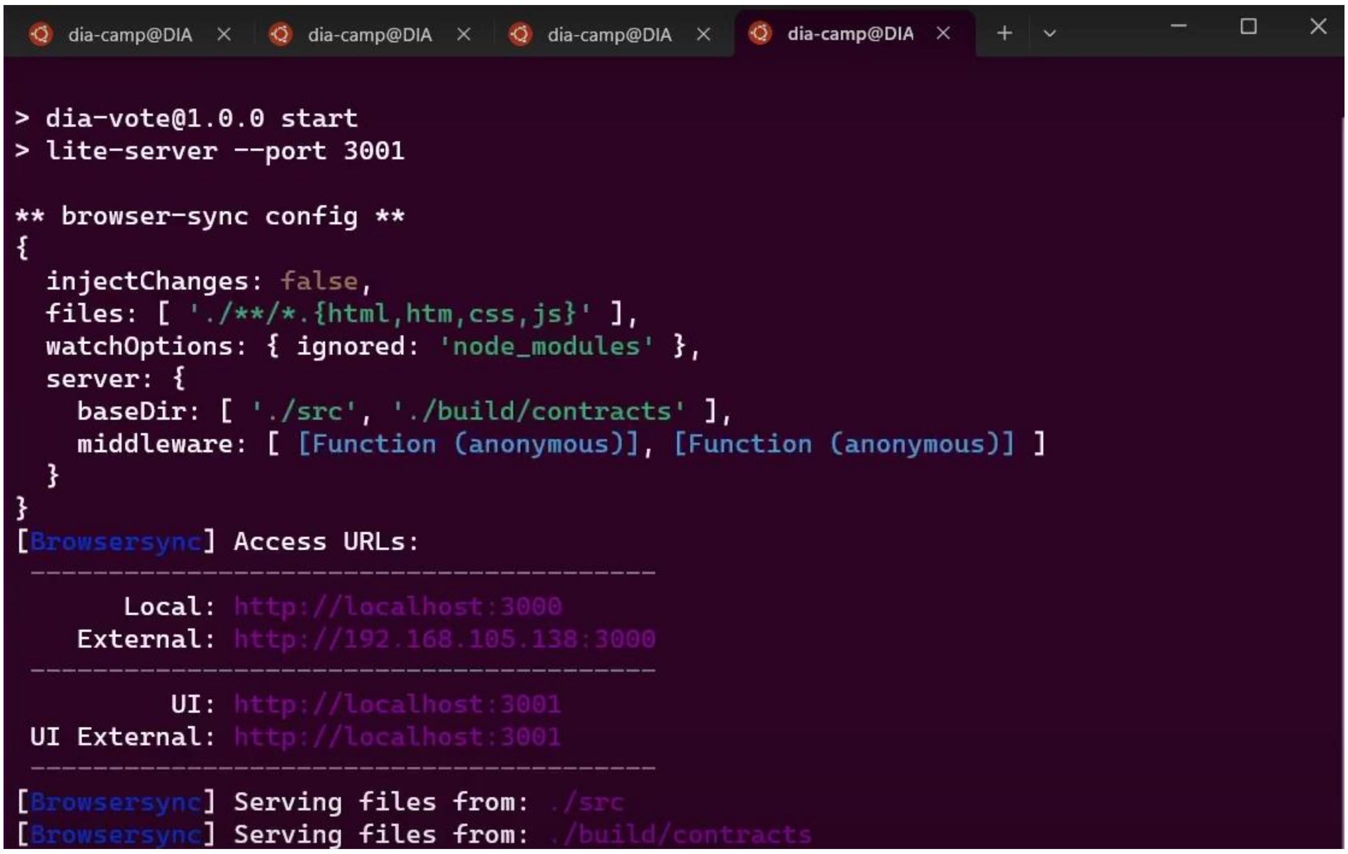
```
dia-camp@DIA-TUF-A15:~/Dev/Blockchain_voting-system$ truffle migrate --reset

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.
⠏ Fetching solc version list from solc-bin. Attempt #1
⠋ Fetching solc version list from solc-bin. Attempt #1
Starting migrations...
======================
> Network name:    'development'
> Network id:      1684866925324
> Block gas limit: 30000000 (0x1c9c380)


1_initial_migration.js
======================
⠙ Fetching solc version list from solc-bin. Attempt #1
   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0x265f3f9dcdcd7743d1c62505025dfc9959af009f960f727e26f71f3
d0ffce31f
   > Blocks: 0            Seconds: 0
   > contract address:    0x5C814c98397CD8745140f263E94eD6a0626Fb05e
   > block number:        1
   > block timestamp:     1684866969
   > account:             0x2721814F8af7F3bF458d8d97875843FadA639BDc
```

Fig 9.2 Migrating Truffle Contracts to Ganache



```
dia-camp@DIA-TUF-A15:~/Dev/Blockchain_voting-system$ ETHERNAL_API_TOKEN=eyJhbGciOiJ
IUzI1NiIsInR5cCI6IkpXVCJ9.eyJmaXJlYmFzZVVzZXJJZCI6Im9Hb2JEQjJydWdZQmtIT2pnYmpwRU16T
lBrUzIiLCJhcGlLZXkiOiI5MUs5Q0JaLTI1RDRGTlEtUVZKMTZGMi1BTkJRSzdRXHUwMDAxIiwiaWF0Ijox
Njg0ODY2MjY0fQ.hZ74o-QZ7lFAWYhNN31QqnB9M2c3Q-ANt13665Flg0E ethernal listen
Authenticated with API token.
Logged in with cateyes6669@gmail.com
Using workspace "Ganache-Mini-project"
Detected Truffle project for /home/dia-camp/Dev/Blockchain_voting-system
Getting artifact for Election.json in /home/dia-camp/Dev/Blockchain_voting-system/b
uild/contracts
Getting artifact for Migrations.json in /home/dia-camp/Dev/Blockchain_voting-system
/build/contracts
Syncing block #4...
Synced block #4
Synced transaction 0xad9420342b504a7244a5f76c86e5654a87a3c64079cb93e4ae91a16faa4d7b
7a
Updated ABI for contract Election (0xB591762e714354d7f02a3e0C761E96C1e23d49D8)
```

Fig 9.3 Connecting to Ethernal



```
> dia-vote@1.0.0 start
> lite-server --port 3001

** browser-sync config **
{
  injectChanges: false,
  files: [ './**/*.{html,htm,css,js}' ],
  watchOptions: { ignored: 'node_modules' },
  server: {
    baseDir: [ './src', './build/contracts' ],
    middleware: [ [Function (anonymous)], [Function (anonymous)] ]
  }
}
[Browsersync] Access URLs:
 --------------------------------------
       Local: http://localhost:3000
    External: http://192.168.105.138:3000
 --------------------------------------
          UI: http://localhost:3001
 UI External: http://localhost:3001
 --------------------------------------
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
```

Fig 9.4 Serving the Frontend using npm start

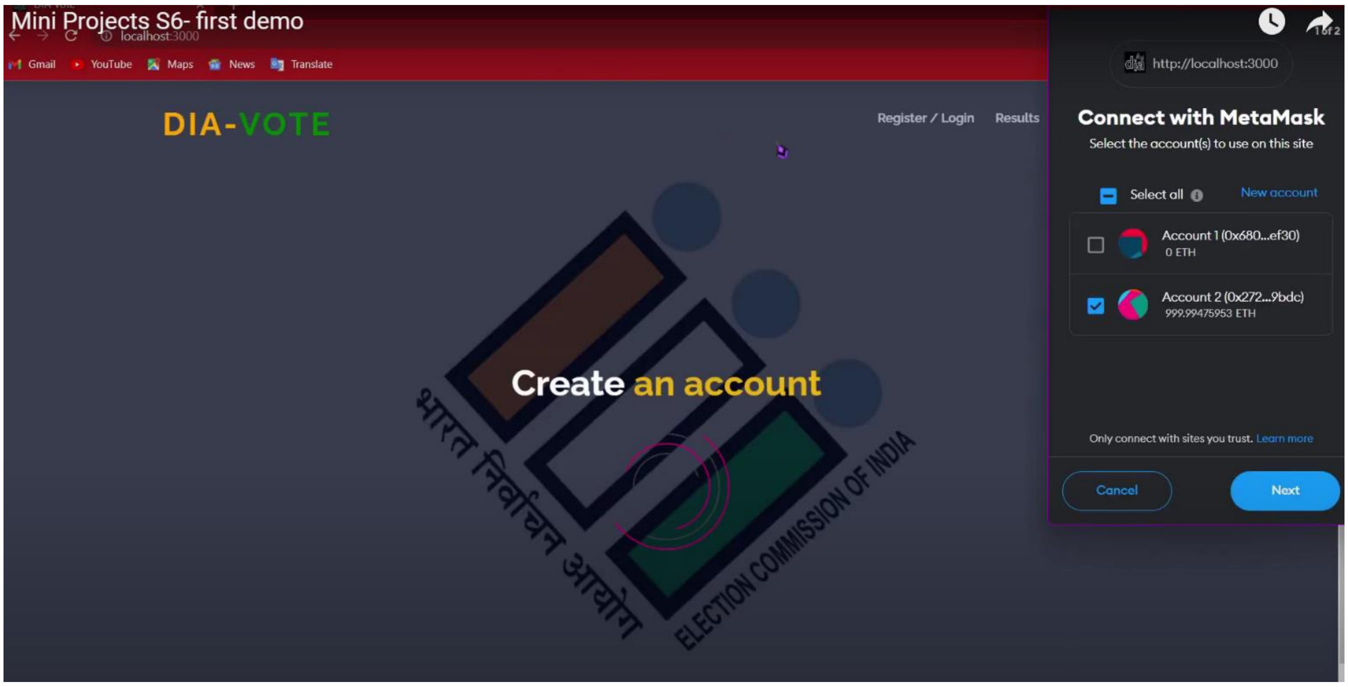
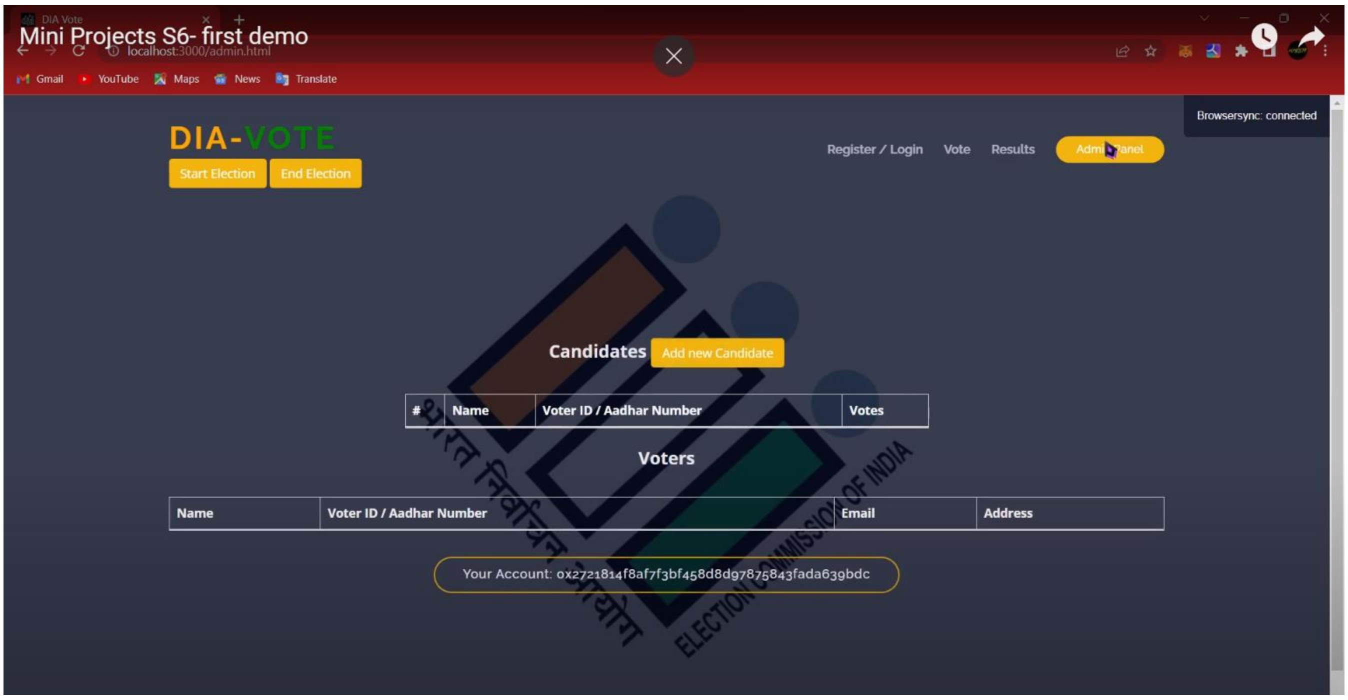


Fig 9.5 Wallet connection Request



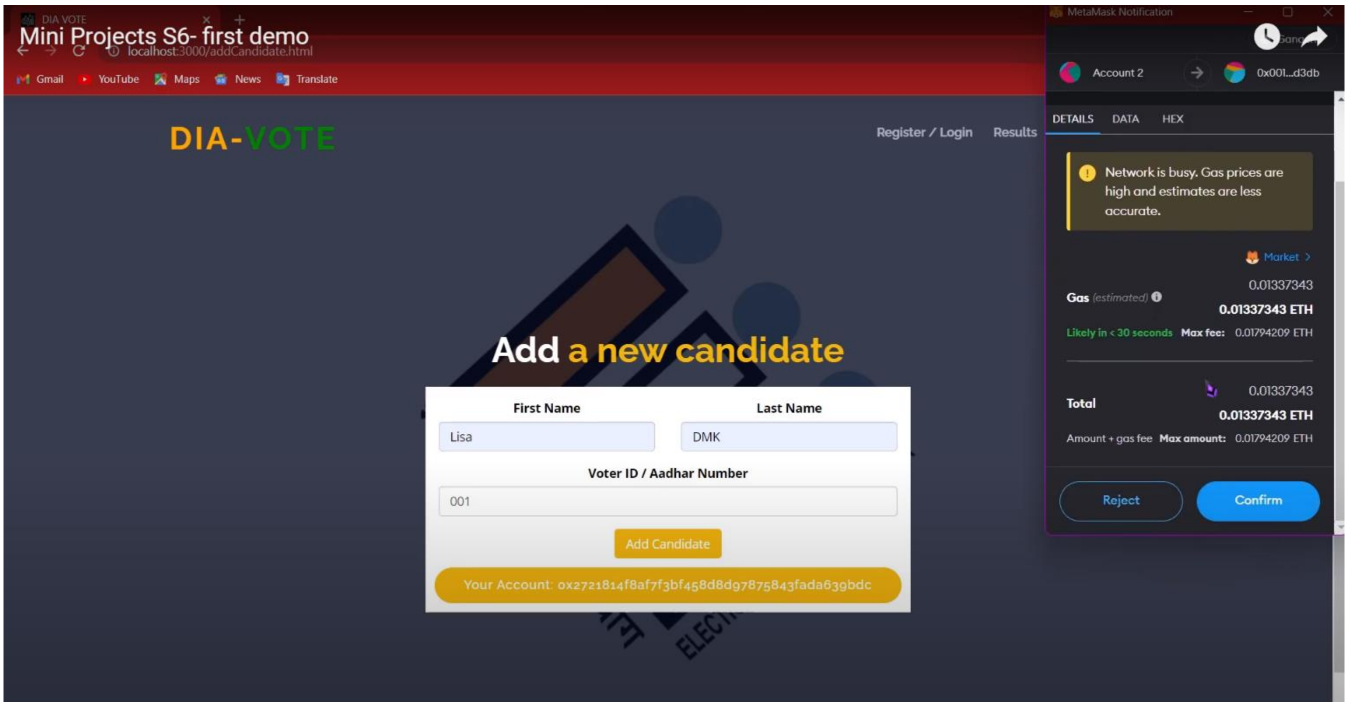
#	Name	Voter ID / Aadhar Number	Votes

Voters

Name	Voter ID / Aadhar Number	Email	Address

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

Fig 9.6 Home Page



Add a new candidate

First Name	Last Name
Lisa	DMK

Voter ID / Aadhar Number: 001

Add Candidate

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

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

Market >

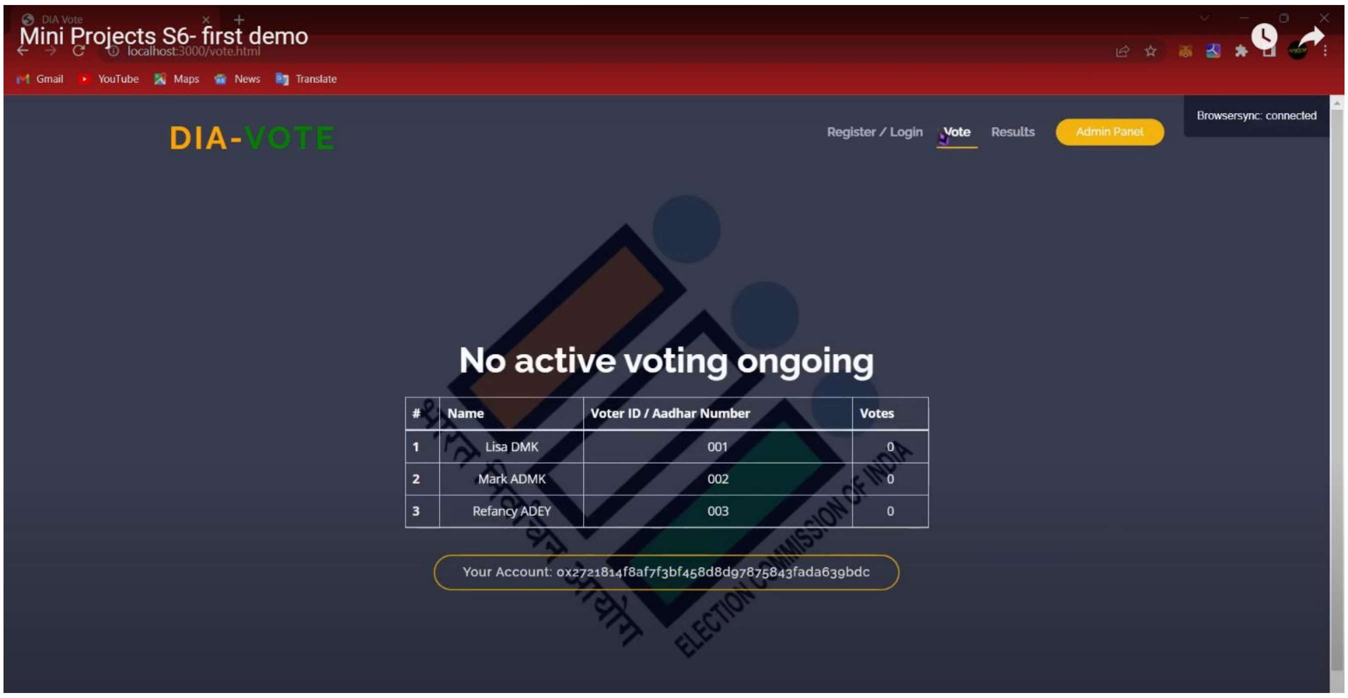
Gas (estimated) 0.01337343 0.01337343 ETH

Likely in < 30 seconds Max fee: 0.01794209 ETH

Total 0.01337343 0.01337343 ETH

Amount + gas fee Max amount: 0.01794209 ETH

Fig 9.7 Add New Candidate

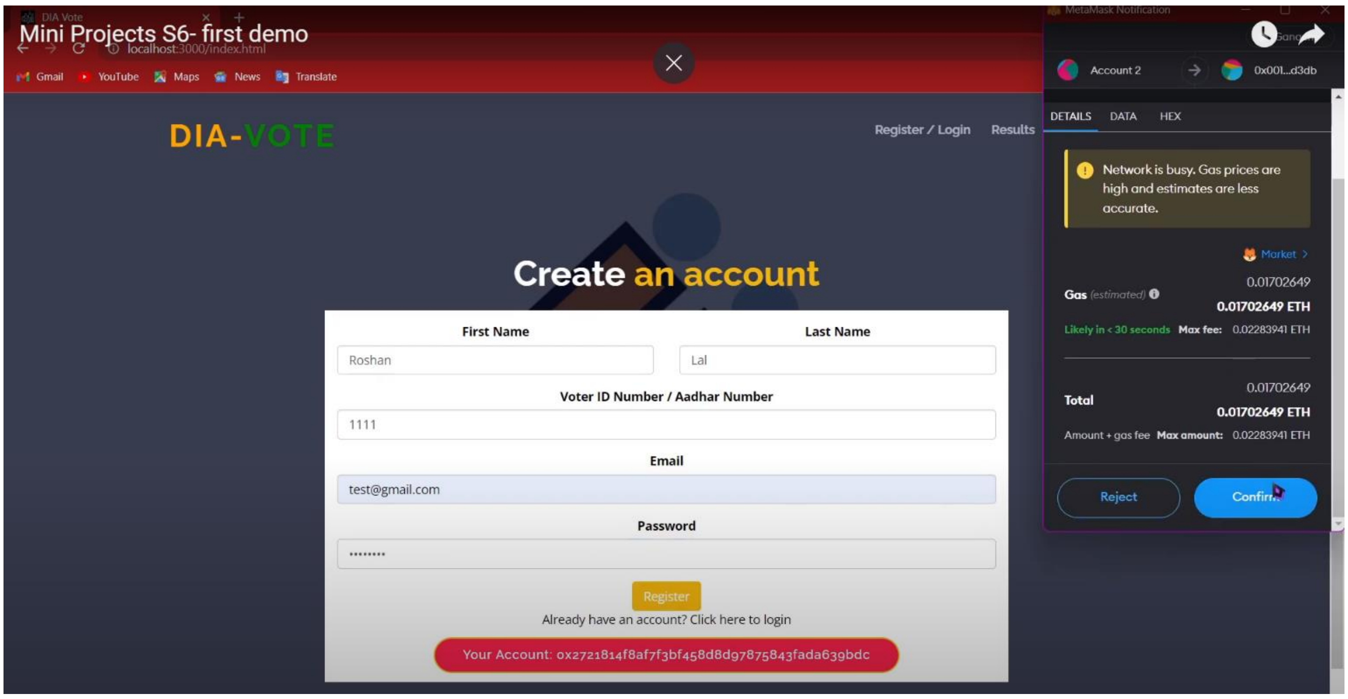


No active voting ongoing

#	Name	Voter ID / Aadhar Number	Votes
1	Lisa DMK	001	0
2	Mark ADMK	002	0
3	Refancy ADEY	003	0

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

Fig 9.8 Election Status



Create an account

First Name: Roshan

Last Name: Lal

Voter ID Number / Aadhar Number: 1111

Email: test@gmail.com

Password: ••••••••

Already have an account? Click here to login

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

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

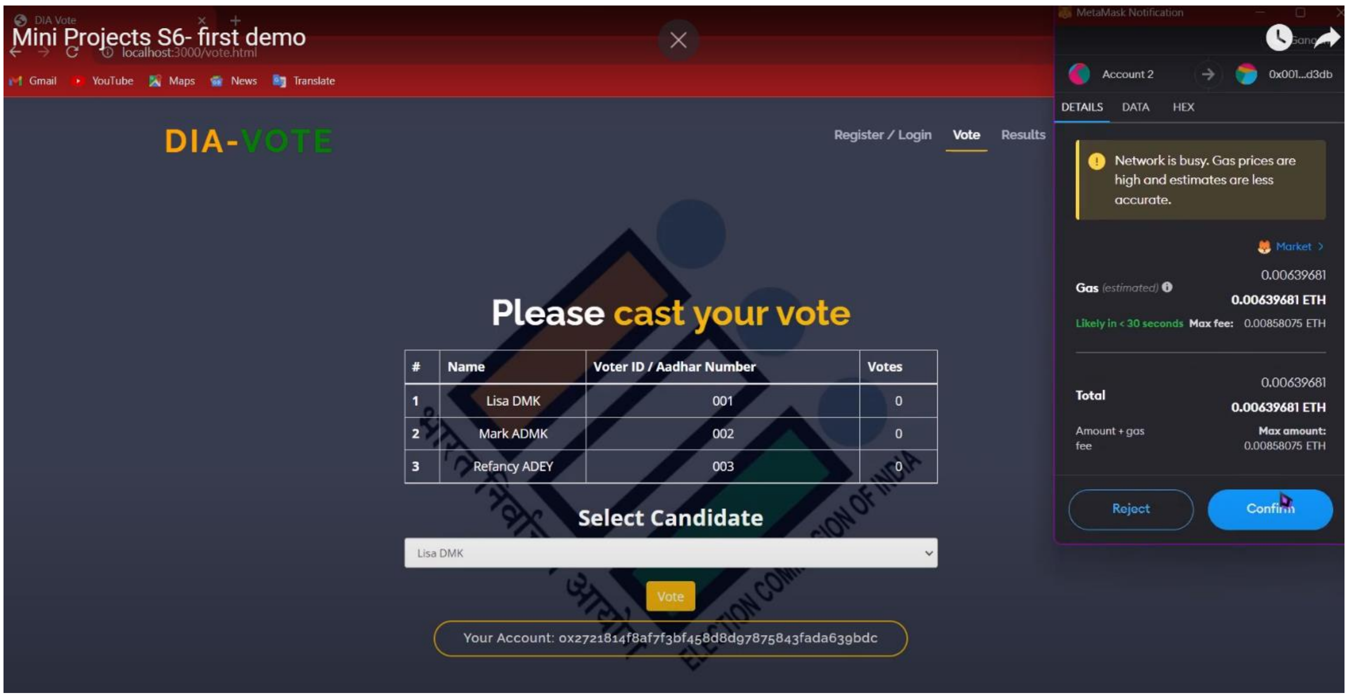
Gas (estimated) 0.01702649 **0.01702649 ETH**

Likely in < 30 seconds **Max fee:** 0.02283941 ETH

Total 0.01702649 **0.01702649 ETH**

Amount + gas fee **Max amount:** 0.02283941 ETH

Fig 9.9 Creating a voter account



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

Market

Gas (estimated) 0.00639681 0.00639681 ETH

Likely in < 30 seconds Max fee: 0.00858075 ETH

Total 0.00639681 0.00639681 ETH

Amount + gas fee Max amount: 0.00858075 ETH

Please cast your vote

#	Name	Voter ID / Aadhar Number	Votes
1	Lisa DMK	001	0
2	Mark ADMK	002	0
3	Refancy ADEY	003	0

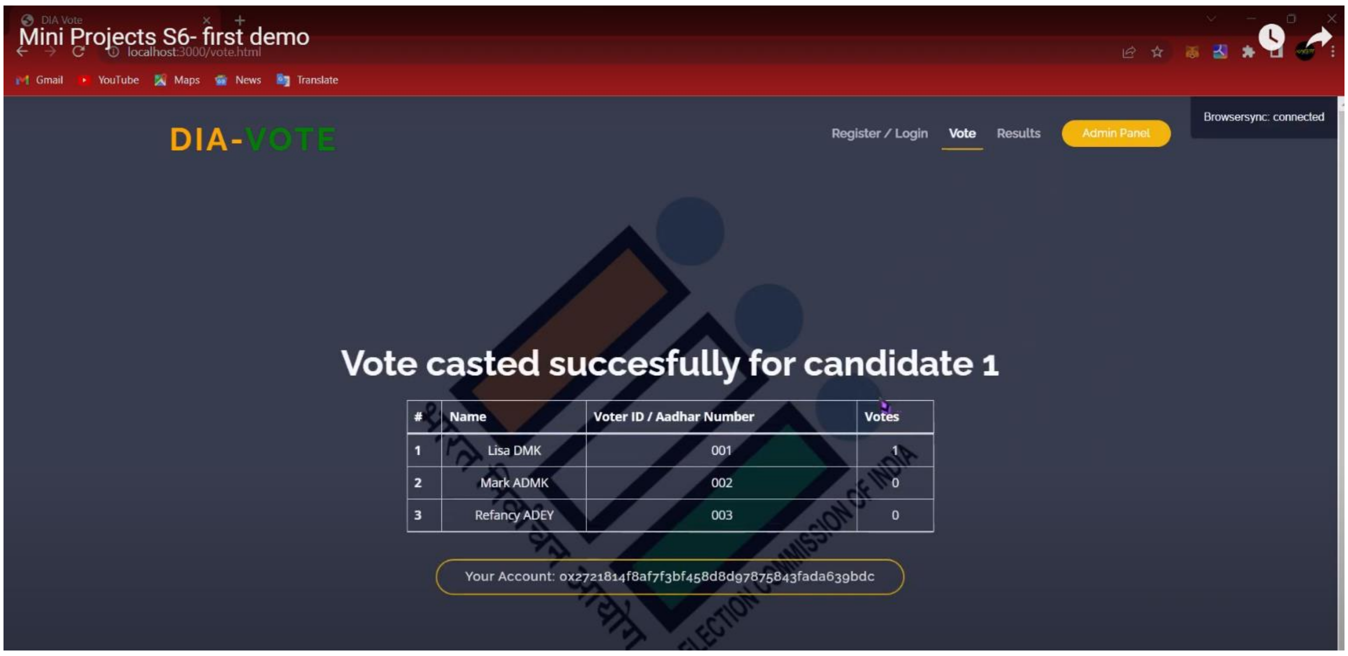
Select Candidate

Lisa DMK

Vote

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

Fig 9.10 Vote Casting

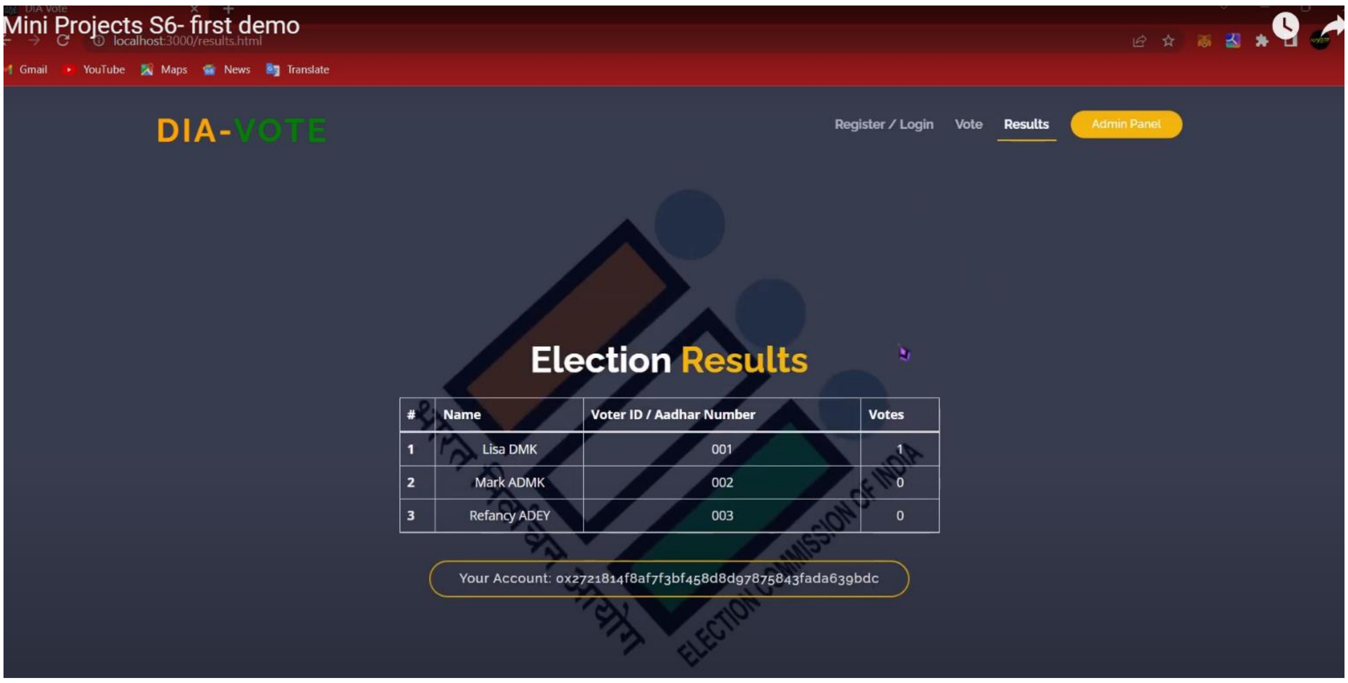


Vote casted succesfully for candidate 1

#	Name	Voter ID / Aadhar Number	Votes
1	Lisa DMK	001	1
2	Mark ADMK	002	0
3	Refancy ADEY	003	0

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

Fig 9.11 Vote Casted Confirmation

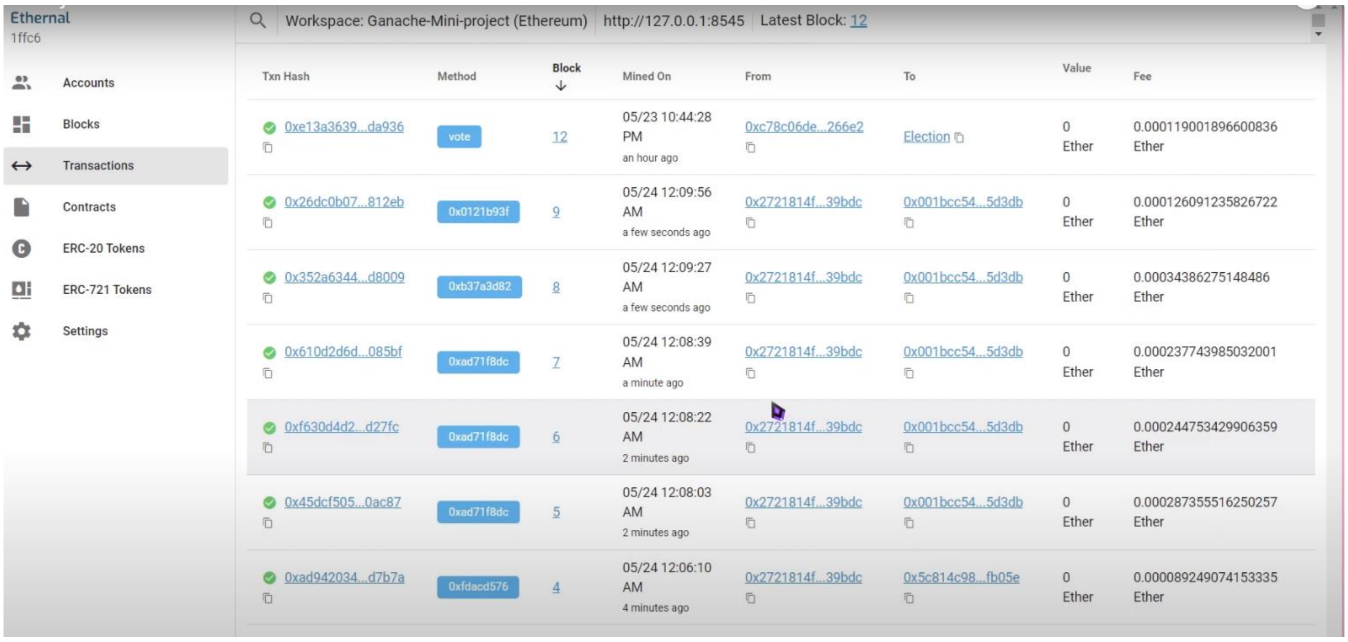


Election Results

#	Name	Voter ID / Aadhar Number	Votes
1	Lisa DMK	001	1
2	Mark ADMK	002	0
3	Refancy ADEY	003	0

Your Account: 0x2721814f8af7f3bf458d8d97875843fada639bdc

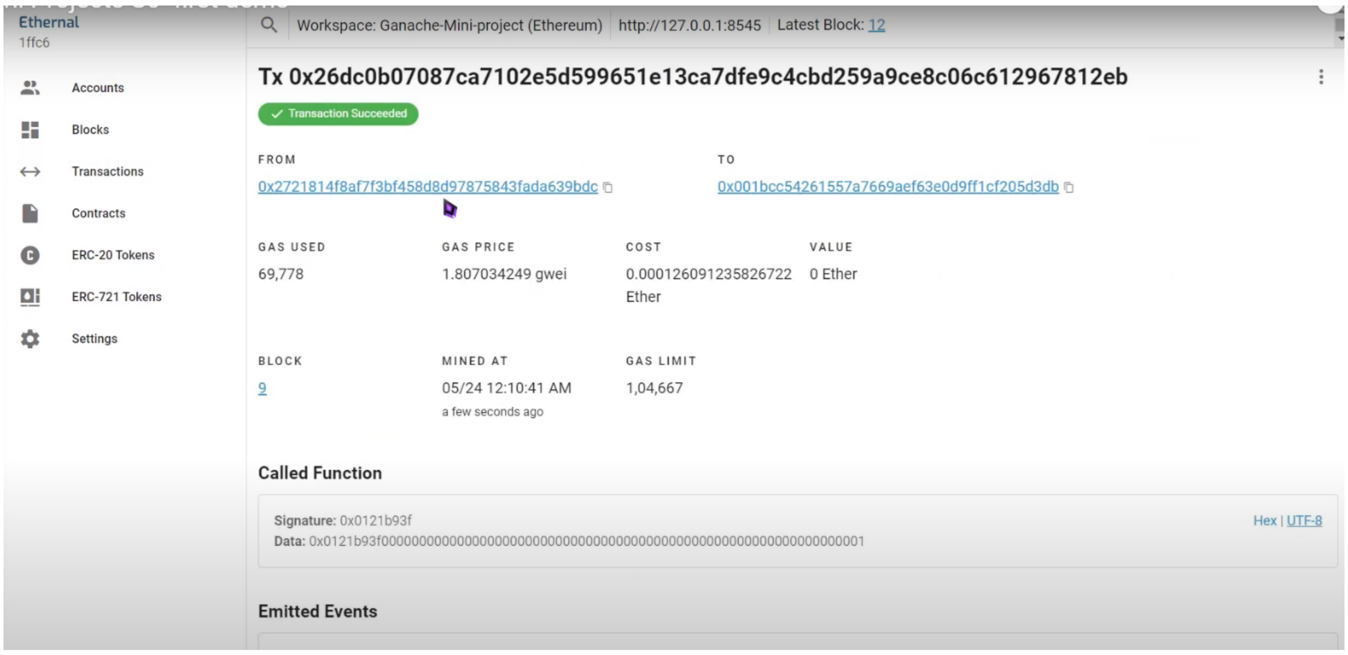
Fig 9.12 Election Results



Workspace: Ganache-Mini-project (Ethereum) | http://127.0.0.1:8545 | Latest Block: 12

Txn Hash	Method	Block ↓	Mined On	From	To	Value	Fee
0xe13a3639...da936	vote	12	05/23 10:44:28 PM an hour ago	0xc78c06de...266e2	Election	0 Ether	0.000119001896600836 Ether
0x26dc0b07...812eb	0x0121b93f	9	05/24 12:09:56 AM a few seconds ago	0x2721814f...39bdc	0x001bcc54...5d3db	0 Ether	0.000126091235826722 Ether
0x352a6344...d8009	0xb37a3d82	8	05/24 12:09:27 AM a few seconds ago	0x2721814f...39bdc	0x001bcc54...5d3db	0 Ether	0.00034386275148486 Ether
0x610d2d6d...085bf	0xad71f8dc	7	05/24 12:08:39 AM a minute ago	0x2721814f...39bdc	0x001bcc54...5d3db	0 Ether	0.000237743985032001 Ether
0xf630d4d2...d27fc	0xad71f8dc	6	05/24 12:08:22 AM 2 minutes ago	0x2721814f...39bdc	0x001bcc54...5d3db	0 Ether	0.000244753429906359 Ether
0x45dcf505...0ac87	0xad71f8dc	5	05/24 12:08:03 AM 2 minutes ago	0x2721814f...39bdc	0x001bcc54...5d3db	0 Ether	0.000287355516250257 Ether
0xad942034...d7b7a	0xfdacd576	4	05/24 12:06:10 AM 4 minutes ago	0x2721814f...39bdc	0x5c814c98...fb05e	0 Ether	0.000089249074153335 Ether

Fig 9.13 Ethernal Block Explorer



Tx 0x26dc0b07087ca7102e5d599651e13ca7dfe9c4cbd259a9ce8c06c612967812eb

Transaction Succeeded

FROM 0x2721814f8af7f3bf458d8d97875843fada639bdc

TO 0x001bcc54261557a7669aef63e0d9ff1cf205d3db

GAS USED	GAS PRICE	COST	VALUE
69,778	1.807034249 gwei	0.000126091235826722 Ether	0 Ether

BLOCK	MINED AT	GAS LIMIT
9	05/24 12:10:41 AM a few seconds ago	1,04,667

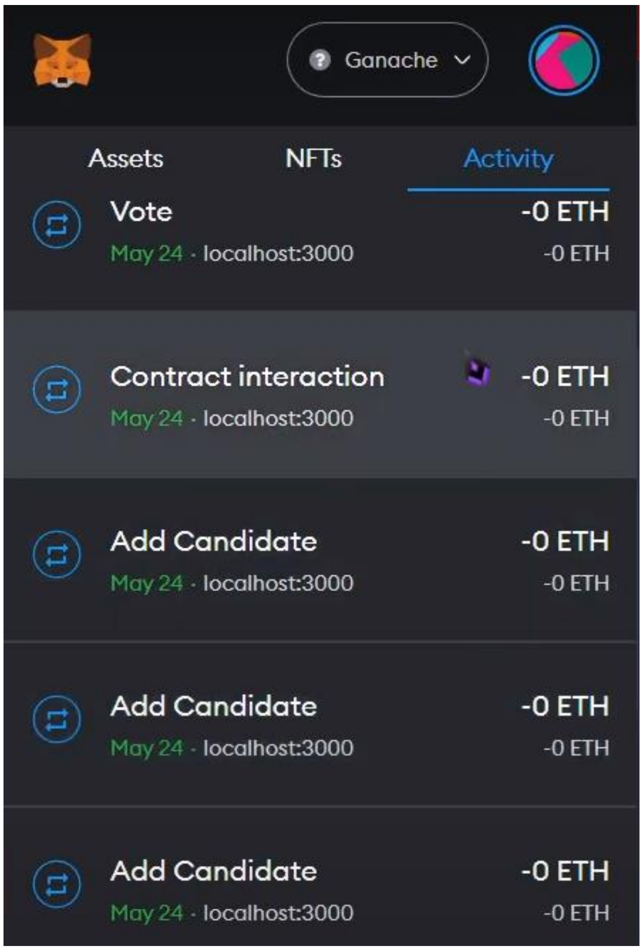
Called Function

Signature: 0x0121b93f
Data: 0x0121b93f0000000000000000000000000000000000000000000000000000000000000001

Hex | UTF-8

Emitted Events

Fig 9.14 Transaction Info



Vote -0 ETH
May 24 · localhost:3000 -0 ETH

Contract interaction -0 ETH
May 24 · localhost:3000 -0 ETH

Add Candidate -0 ETH
May 24 · localhost:3000 -0 ETH

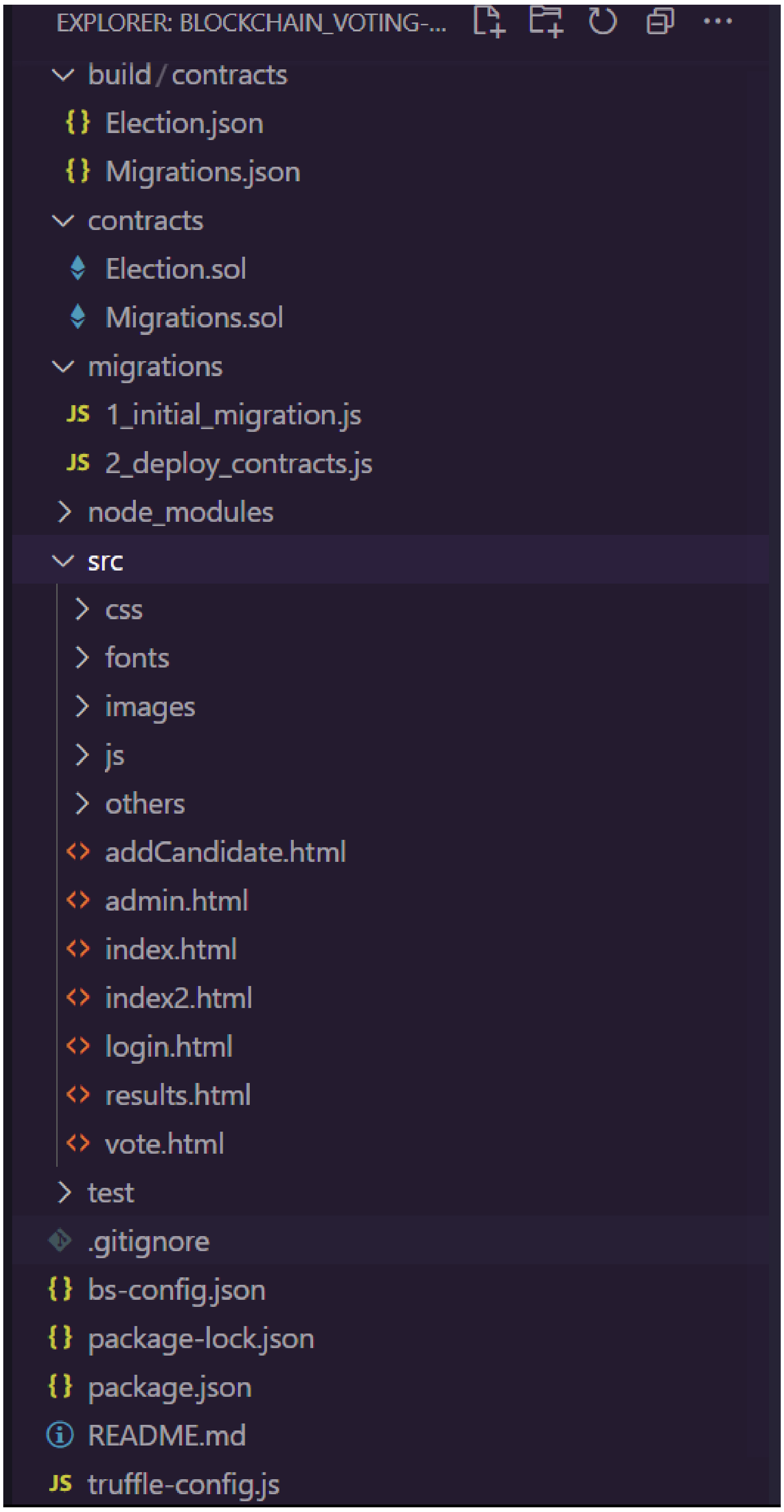
Add Candidate -0 ETH
May 24 · localhost:3000 -0 ETH

Add Candidate -0 ETH
May 24 · localhost:3000 -0 ETH

Fig 9.15 Transaction info on Wallet

CHAPTER 10
SOURCE CODE

Project Structure:



```
EXPLORER: BLOCKCHAIN_VOTING-...
∨ build / contracts
    Election.json
    Migrations.json
∨ contracts
    Election.sol
    Migrations.sol
∨ migrations
    1_initial_migration.js
    2_deploy_contracts.js
> node_modules
∨ src
  > css
  > fonts
  > images
  > js
  > others
    addCandidate.html
    admin.html
    index.html
    index2.html
    login.html
    results.html
    vote.html
> test
  .gitignore
  bs-config.json
  package-lock.json
  package.json
  README.md
  truffle-config.js
```

Election.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Election {

    address public manager;

    struct Candidate {
        uint id;
        string CfirstName;
        string ClastName;
        string CidNumber;
        uint voteCount;
    }

    mapping (address => bool) public voters;

    mapping (uint => Candidate) public candidates;

    uint public candidatesCount;

    event votedEvent (
        uint indexed candidateId
    );

    constructor () {
        manager = msg.sender;
    }

    function addCandidate (string memory _CfirstName, string memory _ClastName, string
memory _CidNumber) public restricted {
        candidatesCount++;
        candidates[candidatesCount] = Candidate(candidatesCount, _CfirstName, _ClastName,
_CidNumber, 0);
    }

    modifier restricted () {
        require(msg.sender == manager, "Access denied. Only manager can perform this
action.");
        _;
    function vote (uint _candidateId) public {
        require(!voters[msg.sender], "You have already voted.");

        require(_candidateId > 0 && _candidateId <= candidatesCount, "Invalid candidate
ID.");

        voters[msg.sender] = true;

        candidates[_candidateId].voteCount++;

        uint candidateId = _candidateId;

        emit votedEvent(_candidateId);
    }

    // Users
    // Register
    struct User {
        string firstName;
        string lastName;
        string idNumber;
        string email;
        string password;
        address add;
    }

    mapping (uint => User) public users;

    uint public usersCount;

    function addUser (string memory _firstName, string memory _lastName, string memory
_idNumber, string memory _email, string memory _password) public {
        usersCount++;
        users[usersCount] = User(_firstName, _lastName, _idNumber, _email, _password,
msg.sender);
    }

}
```

Migrations.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Migrations {
    address public owner;
    uint public last_completed_migration;

    modifier restricted() {
        if (msg.sender == owner) _;
    }

    constructor() {
        owner = msg.sender;
    }

    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }

    function upgrade(address new_address) public restricted {
        Migrations upgraded = Migrations(new_address);
        upgraded.setCompleted(last_completed_migration);
    }
}
```

Truffle-config.js

```javascript
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*" // Match any network id
    }
  },
  compilers: {
    solc: {
      version: "^0.8.0" // Use a specific version of Solidity
    }
  }
};
```

App.js

```
App = {
    web3Provider: null,
    contracts: {},
    account: '0x0',
    hasVoted: false,
    votedForID: 0,
    finishElection: 0,
    mins: 0,

    // web3 connects our client side application to the blockchain.
    // metamask gives us an instance of web3 that we will use to connect to the blockchain
    // if this doesn't happen we will set a default web3 provider from our local
blockchain instance 'localhost 7545'
    init: function () {
        return App.initWeb3();
    },

    initWeb3: async function () {
        // Modern dapp browsers...
        if (window.ethereum) {
            App.web3Provider = window.ethereum;
            try {
                // Request account access
                await window.ethereum.enable();
            } catch (error) {
                // User denied account access...
                console.error("User denied account access")
            }
        }
// Legacy dapp browsers...
        else if (window.web3) {
            App.web3Provider = window.web3.currentProvider;
        }
// If no injected web3 instance is detected, fall back to Ganache
        else {
            App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
        }
        web3 = new Web3(App.web3Provider);
        web3.eth.defaultAccount=web3.eth.accounts[0]
        return App.initContract();
    },


    // we then initailze our contract
    // this function loads up our contract to our front end application
    initContract: function () {
        $.getJSON("Election.json", function (election) {
            // Instantiate a new truffle contract from the artifact
            App.contracts.Election = TruffleContract(election);
            // Connect provider to interact with contract
            App.contracts.Election.setProvider(App.web3Provider);

            App.listenForEvents();

            return App.render();
        });
    },


    // Listen for events emitted from the contract
    listenForEvents: function () {
        App.contracts.Election.deployed().then(function (instance) {
            // Restart Chrome if you are unable to receive this event
            // This is a known issue with Metamask
            // https://github.com/MetaMask/metamask-extension/issues/2393
            instance.votedEvent({}, {
                fromBlock: 0,
                toBlock: 'latest'
            }).watch(function (error, event) {
                console.log("event triggered", event)
                // Reload when a new vote is recorded
                // App.render();
            });
        });
    },



    // render function which is what will layout all the content on the page
```

```
render: function () {
    var electionInstance;
    var loader = $("#loader");
    var content = $("#content");
    var register = $("#register");

    loader.show();
    content.hide();

    // Load account data
    web3.eth.getCoinbase(function (err, account) {
        if (err === null) {
            App.account = account;
            $("#accountAddress").html("Your Account: " + account);
        }
    });


    App.contracts.Election.deployed().then(function(instance) {
        electionInstance = instance;
        // document.querySelector('.buy-tickets').style.display = 'none';

        return electionInstance.manager();
    }).then(function (manager) {
        if (manager !== App.account){
            document.querySelector('.buy-tickets').style.display = 'none';
        }

      return electionInstance.candidatesCount();
    }).then(function(candidatesCount) {
      var candidatesResults = $("#candidatesResults");
      candidatesResults.empty();

      var candidatesSelect = $('#candidatesSelect');
      candidatesSelect.empty();

      for (var i = 1; i <= candidatesCount; i++) {
        electionInstance.candidates(i).then(function(candidate) {
          var id = candidate[0];
          var fname = candidate[1];
          var lname = candidate[2];
          var idNumber = candidate[3];
```

```
              var voteCount = candidate[4];

              // Render candidate Result
              var candidateTemplate = "<tr><th>" + id + "</th><td>" + fname+ " " + lname +
"</td><td>" + idNumber  + "</td><td>" + voteCount + "</td></tr>"
              candidatesResults.append(candidateTemplate);

              // Render candidate ballot option
              var candidateOption = "<option value='" + id + "' >" + fname + " " + lname +
"</ option>"
              candidatesSelect.append(candidateOption);

                return electionInstance.candidatesCount();
            });
          }
          return electionInstance.voters(App.account);
        }).then( function(hasVoted) {
          // Do not allow a user to vote twice
          if(hasVoted) {
            $('form').hide();
              $("#index-text").html("You are successfully logged in!");
              $("#new-candidate").html("New candidates can't be added. The election
process has already started.");
            $("#vote-text").html("Vote casted succesfully for candidate " +
localStorage.getItem("votedForID"));
          }
          loader.hide();
          content.show();
          return electionInstance.usersCount();
        }).then(function (usersCount) {
            var voterz = $("#voterz");
            voterz.empty();

            for (var i = 1; i <= usersCount; i++) {
                electionInstance.users(i).then(function (user) {
                    var firstName = user[0];
                    var lastName = user[1];
                    var idNumber = user[2];
                    var email = user[3];
                    var address = user[5];

                    let voterTemplate = "<tr><td>" + firstName + " " + lastName +
"</td><td>" + idNumber + "</td><td>" + email + "</td><td>" + address + "</td></tr>"
                    voterz.append(voterTemplate);
                });
            }

            if (localStorage.getItem("finishElection") === "1") {
                $('form').hide();
                $("#index-text").html("There is no active election ongoing at the
moment");
                $("#vote-text").html("No active voting ongoing");
                // $("#result-text").html("The voting process has ended. These are the
final results");
                document.querySelector('.addCandidateForm').style.display = 'block';
                // document.querySelector('.reg').style.display = 'none';
                document.querySelector('.vot').style.display = 'none';
            } else if (localStorage.getItem("finishElection") === "0") {

            }

        }).catch(function(error) {
          console.warn(error);
        });
    },


    castVote:  function () {
        var candidateId = $('#candidatesSelect').val();
        App.votedForID = candidateId;
        localStorage.setItem("votedForID", candidateId);
        App.contracts.Election.deployed().then(function (instance) {
            return instance.vote(candidateId, {from: App.account});
        }).then(function (result) {
            // Wait for votes to update
            $("#content").hide();
            $("#loader").show();

            location.href='results.html';
        }).catch(function (err) {
            console.error(err);
        });
    },
    addUser: async function () {
        var firstName = $('#firstName').val();
        var lastName = $('#lastName').val();
        var idNumber = $('#idNumber').val();
        var email = $('#email').val();
        var password = $('#password').val();
        var app = await App.contracts.Election.deployed();
        await app.addUser(firstName, lastName, idNumber, email, password);
        $("#content").hide();
        $("#loader").show();
        document.querySelector('.vot').style.display = 'block';
        location.href='vote.html';

    },

    addCandidate: async function (){
        var CfirstName = $('#CfirstName').val();
        var ClastName = $('#ClastName').val();
        var CidNumber = $('#CidNumber').val();

        var app = await App.contracts.Election.deployed();
        await app.addCandidate(CfirstName, ClastName, CidNumber);
        $("#content").hide();
        $("#loader").show();

        location.href='admin.html';
    },

    login: async function() {
        var lidNumber = $('#lidNumber').val();
        var lpassword = $('#lpassword').val();

        var app = await App.contracts.Election.deployed();
        var users = await app.users();
        var usersCount = await app.usersCount;

        for (var i = 1; i <= usersCount; i++) {
            electionInstance.users(i).then(function (user) {
                var idNumber = user[2];
                var password = user[4];
            });
```

```
            if (lidNumber === idNumber) {
                if(lpassword === password)
                {
                    location.href='results.html';
                }
                else {
                    prompt("Incorrect login details, Please try again");
                }

                break;
            }

        }

    },

    startElection: function () {
        localStorage.setItem("finishElection", "0");
        location.href='index.html';
    },

    endElection: function () {
        localStorage.setItem("finishElection", "1");
        location.href='results.html';
    }

};

$(function () {
    $(window).load(function () {
        App.init();
    });
});
```

CHAPTER 11
CONCLUSION AND FUTURE ENHANCEMENT

11.1 CONCLUSION:

In conclusion, the development and implementation of a decentralized voting system using blockchain technology hold great promise in addressing the challenges faced by traditional voting systems. By leveraging the transparency, immutability, and security provided by blockchain, this system aims to restore trust, enhance transparency, and streamline the electoral process.

Through the utilization of tools such as Solidity, Truffle, Ganache, HTML, CSS, Bootstrap, jQuery, and Ethernal, a robust and user-friendly decentralized voting system has been created. These tools facilitated the development of secure smart contracts, streamlined the deployment process, ensured a visually appealing user interface, and enabled the exploration of the local blockchain.

11.2 FUTURE ENHANCEMENT:

The future enhancements for the decentralized voting system using blockchain technology aim to continuously improve its functionality, security, and usability. These efforts will contribute to the ongoing evolution of the system, ensuring its adaptability, scalability, and effectiveness in revolutionizing the electoral process and fostering trust in democratic governance.

- Integration of Privacy Features: In the future, enhancing the decentralized voting system with privacy features, such as zero-knowledge proofs or cryptographic techniques, can further protect the identities and voting records of participants while preserving the system's transparency.

- Integration of Additional Blockchain Platforms: Currently, the decentralized voting system utilizes the Ethereum blockchain. However, exploring the integration of other blockchain platforms, such as Polka Dot or Hyperledger, could expand the system's compatibility and scalability.
- Mobile Application: Developing a mobile application for the decentralized voting system would allow for greater accessibility and convenience, enabling voters to cast their ballots securely from their smartphones.
- Advanced Data Analytics: Implementing advanced data analytics capabilities within the system could provide insights into voting patterns, demographics, and voter behaviour, which can aid in making informed decisions and improving future elections.
- Integration with Identity Verification Systems: Integrating with established identity verification systems, such as government-issued IDs or biometric authentication, would further enhance the system's security and reduce the risk of fraudulent voting.
- Collaboration with Regulatory Bodies: Engaging in collaboration with electoral regulatory bodies and governments to ensure compliance and legal frameworks could pave the way for the adoption of the decentralized voting system on a larger scale.

REFERENCES

- https://www.geeksforgeeks.org/decentralized-voting-system-using-blockchain/
- https://learn.metamask.io/
- https://academy.binance.com/en/articles?page=1&tags=blockchain
- https://ethereum.org/en/developers/docs/
- https://www.investopedia.com/financial-term-dictionary-4769738
- https://trufflesuite.com/docs/
- https://docs.metamask.io/
- https://docs.soliditylang.org/en/latest/
- https://web3js.readthedocs.io/en/v1.3.4/web3.html
- https://doc.tryethernal.com/
- https://chat.openai.com/

[1] Blockchain-Based E-Voting System, Friðrik Þ. Hjálmarsson, Gunnlaugur K. Hreiðarsson, IEEE 11th International Conference on Cloud Computing (CLOUD), 2018

[2] A Smart Contract for Boardroom Voting with Maximum Voter Privacy, Patrick McCorry, Siamak F. Shahandashti and Feng Hao, International Conference on Financial Cryptography and Data Security, 2017

[3] An efficient and effective Decentralized Anonymous Voting System. Wei-Jr Lai, Ja-Ling Wu, ArXiv, 2018