

**BLOCKCHAIN ENABLED DECENTRALIZED
TRUST MANAGEMENT AND SECURE
VOTING SYSTEM**

A PROJECT REPORT

Submitted by

BADHRI KESAVA RAJA S M	[211419104034]
GODSON RAJ R	[211419104079]
GUNA M	[211419104087]

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2023

BONAFIDE CERTIFICATE

Certified that this project report “**BLOCKCHAIN ENABLED DECENTRALIZED TRUST MANAGEMENT AND SECURE VOTING SYSTEM**” is the bonafide work of “**BADHRI KESAVA RAJA S M (211419104034), GODSON RAJ R (211419104079), GUNA M (211419104087)**” who carried out the project work under my supervision.

SIGNATURE

**Dr.L.JABASHEELA,M.E.,Ph.D.,
HEAD OF THE DEPARTMENT**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

**Mrs. P. DEEPA M.E.,(Ph.D),
SUPERVISOR
ASSOCIATE PROFESSOR**

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the End Semester Project

Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION BY THE STUDENT

We **BADHRI KESAVA RAJA S M (211419104034), GODSON RAJ R (211419104079), GUNA M (211419104087)** hereby declare that this project report titled “**BLOCKCHAIN ENABLED DECENTRALIZED TRUST MANAGEMENT AND SECURE VOTING SYSTEM**”, under the guidance of **Mrs. P. DEEPA M.E., (Ph.D.)**, is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

BADHRI KESAVA RAJA S M

GODSON RAJ R

GUNA M

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHI KUMAR, M.E.,Ph.D** and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.**, for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. L.JABASHEELA, M.E.,Ph.D.**, for the support extended throughout the project.

We would like to thank our parents, friends, project Guide **Mrs. P. DEEPA M.E.,(Ph.D.)**,and coordinator **Dr.N.PUGHAZENDI M.E., Ph.D.**, and all the faculty members of theDepartment of CSE for their advice and encouragement for the successful completion ofthe project.

BADHRI KESAVA RAJA S M

GODSON RAJ R

GUNA M

ABSTRACT

An electoral system or voting system is a set of rules that determine how elections and referendums are conducted and how their results are determined. Political electoral systems are organized by governments. Current voting systems like ballot box voting or electronic voting suffer from various security threats such as DDoS attacks, polling booth capturing, vote alteration and manipulation, malware attacks, etc., and require huge amounts of paperwork, human resources, and time. This creates a sense of distrust among existing systems. The microchip or server can be hacked. Blockchain voting, if implemented properly, can boost voter turnout, and offer more accessible and transparent elections. Citizens can easily cast their votes via their personal computers or mobile phones after completing identity verification. Voting records are easily verifiable and vote tallying is conveniently confirmed in real-time on the network. Blockchain voting saves time, reduces costs, and paves a path for direct democracy. However, blockchain voting is not yet ready. Where the User can cast vote using their personal devices. while using blockchain technology it prevents from DDoS attack and any other malware. It uses peer to peer network so no centralized authority. This system is designed to focus on a secure voting system, lower costs, faster wait times, no disparities due to various erroneous proxies, high scalability, and geographic independence. Overall, an effective election mechanism to strengthen the democratic process. Our project allows voters to vote from the comfort of their own homes, saving time and reducing the number of false votes registered.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	iii
	LIST OF TABLES	vi
	LIST OF FIGURES	vii
	LIST OF SYMBOLS, ABBREVIATIONS	viii
1.	INTRODUCTION	1
	1.1 Overview	1
	1.2 Problem Definition	2
2.	LITERATURE SURVEY	3
3.	SYSTEM ANALYSIS	6
	3.1 Existing System	7
	3.2 Proposed system	8
	3.3 Feasibility Study	9
	3.4 Hardware Environment	10
	3.5 Software Environment	11
4.	SYSTEM DESIGN	12
	4.1 Entity-Relationship Diagram	13
	4.2 Data Dictionary	14
	4.3 Table Normalization	15
	4.4 Data Flow Diagram	18
	4.5 UML Diagrams	21
	4.5.1 Use Case Diagram	22
	4.5.2 Sequence Diagram	23
	4.5.3 Class Diagram	24
	4.5.4 Activity Diagram	25

CHAPTER NO.	TITLE	PAGE NO.
5.	SYSTEM ARCHITECTURE	26
	5.1 Architecture Diagram	26
6.	SYSTEM IMPLEMENTATION	27
	6.1 Module Design Specification	27
	6.1.1 Login Module	27
	6.1.2 Initializing a new ballot	29
	6.1.3 Voting process	29
	6.1.4 Resultant process	30
7.	SYSTEM TESTING	32
	7.1 Unit Testing	32
	7.2 Integration Testing	34
	7.3 Test Cases and Reports	35
8.	CONCLUSION & FUTURE ENHANCEMENT	36
	8.1 Results & Discussion	36
	8.2 Conclusion	36
	8.3 Future Enhancements	37
	APPENDICES	39
	A.1 Coding	39
	A.2 Sample Screens	57
	REFERENCES	61

LIST OF TABLES

TABLE NO.	TABLE DESCRIPTION	PAGE NO.
7.1	Test case and Test results	35

LIST OF FIGURES

FIGURE NO.	FIGURE DESCRIPTION	PAGE NO.
4.1	System Design Process	12
4.2	E-R Diagram	13
4.3	Level-0-DFD	20
4.4	Level-1-DFD	21
4.5	Use-Case Diagram	22
4.6	Sequence Diagram	23
4.7	Class Diagram	24
4.8	Activity Diagram	25
5.1	System Architecture Design	26
6.1	Login Module	28
6.2	Initializing a new ballot Module	29
6.3	Voting process Module	30
6.4	Resultant Module	31
A2.1	Executing the input	57
A2.2	Admin Login	57
A2.3	Candidate registration	58
A2.4	Initializing new ballot	58
A2.5	Voter Registration	59
A2.6	Approving the voters	59
A2.7	Voting process	60
A2.8	Resultant process	60

LIST OF SYMBOLS, ABBREVIATIONS

SERIAL NO.	ABBREVIATION	EXPANSION
1	DLT	Distributed Ledger Technology
2	DAO	Decentralized Autonomous Organization
3	EVM	Ethereum Virtual Machine
4	ETH	Ethereum cryptocurrency
5	BTC	Bitcoin cryptocurrency
6	KYC	Know Your Customer
7	AML	Anti-Money Laundering
8	ERC	Ethereum Request for comments
9	ZKP	Zero Knowledge Proof

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

1.1 OVERVIEW

The goal of this project is to create a decentralized transparent voting and analysis system that can be implemented with blockchain to provide an efficient and highly secure justifiable method of implementing election systems in countries where traditional physical voting with gameable securities is used, increasing the chances of rigged elections. This system is designed to focus on a secure voting system, lower costs, faster wait times, no disparities due to various erroneous proxies, high scalability, and geographic independence. Overall, an effective election mechanism to strengthen the democratic process. Our project allows voters to vote from the comfort of their own homes, saving time and reducing the number of false votes registered. An electoral system or voting system is a set of rules that determine how elections and referendums are conducted and how their results are determined. Political electoral systems are organized by governments. Political electoral systems are defined by constitutions and electoral laws, are typically conducted by election commissions, and can use multiple types of elections for different offices. Electronic voting (e-voting) is becoming an increasingly popular way to cast ballots in elections. However, the security of e-voting systems has long been a concern due to the potential for fraud and manipulation. To address these issues, blockchain technology has emerged as a promising solution for secure and transparent e-voting. A blockchain is a decentralized, tamper-proof database that stores information in a secure and transparent.

1.2 PROBLEM STATEMENT

Current voting systems like ballot box voting or electronic voting suffer from various security threats such as DDoS attacks, polling booth capturing, vote alteration and manipulation, malware attacks, etc., and also require huge amounts of paperwork, human resources, and time. This creates a sense of distrust among existing systems. Electronic voting systems are rapidly overlapping the traditional paper-based voting. In traditional voting there are number of factors that make rigging in whole electoral process such as counting of votes, fake voters and involvement of outside sources and other problems like time consumption, cost budget problems etc. So, the purpose of this proposal is to investigate how to model an authentic reliable and upright E-voting system so that a voter is submitted a vote in secure manner while maintaining the time, verification, budget, and the security of the entire system.

Online Voting System provides the online registration form for the users before voting and makes the users to cast their vote online. The system is to be developed with high security and user friendly. Since there is no tracing back of your vote. But if you use blockchain- it stores everything as a transaction that will be explained soon below; and hence gives you a receipt of your vote (in a form of a transaction ID) and you can use it to ensure that your vote has been counted securely.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 "Blockchain-Based E-Voting System for Enhancing Transparency and Security"

Author Name: Ali Alzahrani, Muhammad Khurram Khan, and Bilal Alatas

Year of Publish: 2019

The paper proposes a blockchain-based electronic voting system to enhance transparency and security. The proposed system uses a permissioned blockchain, where the voting nodes are authorized to participate in the election process. The paper also proposes a consensus mechanism for the validation of transactions on the blockchain.

2.2 "A Decentralized E-Voting System using Ethereum Blockchain"

Author Name: P. R. Balamurugan, S. Sathiyabama, and S. Balamurugan

Year of Publish: 2019

The paper proposes a decentralized electronic voting system using the Ethereum blockchain. The proposed system aims to provide a secure and transparent voting process, where the voting records are immutable and tamper-proof. The paper also proposes a smart contract-based approach to automate the voting process.

2.3 "A Survey on Blockchain-Based Voting Systems"

Author Name: Alessandro Bruni and Andrea Pinna

Year of Publish: 2018

The paper provides a comprehensive survey of blockchain-based voting systems. The paper discusses the advantages and disadvantages of using

blockchain technology for electronic voting. The paper also provides an overview of the various types of blockchain-based voting systems, such as permissioned and permissionless blockchains.

2.4 "Secure Voting System Based on Blockchain Technology"

Author Name: Shoaib Akhtar, Mohsin Nazir, and Tahir Abbas

Year of Publish: 2018

The paper proposes a secure voting system based on blockchain technology. The proposed system uses a permissioned blockchain and a hybrid consensus mechanism to validate transactions. The paper also proposes a system architecture for the proposed voting system.

2.5 "Blockchain-Based E-Voting: A Survey of the State-of-the-Art"

Author Name: Vincenzo Scotti and Angelo Trotta

Year of Publish: 2020

The paper provides a survey of the state-of-the-art in blockchain-based electronic voting. The paper discusses the advantages and disadvantages of using blockchain technology for electronic voting. The paper also provides an overview of the various types of blockchain-based electronic voting systems, such as permissioned and permissionless blockchains.

2.6 "Decentralized Blockchain-Based Electronic Voting Systems: A Comprehensive Survey"

Author Name: K. Srinivasan and V. P. Singh

Year of Publish: 2020

The paper provides a comprehensive survey of decentralized blockchain-based electronic voting systems. The paper discusses the advantages about the

of using blockchain technology for electronic voting. The paper also provides an overview of the various types of decentralized blockchain-based electronic voting systems, such as decentralized autonomous organizations (DAOs) and smart contract-based voting systems.

2.7 "A Decentralized E-Voting System using Smart Contracts and Blockchain Technology"

Author Name: S. A. Fatima, S. Bano, and F. Khan

Year of Publish: 2021

The paper proposes a decentralized electronic voting system using smart contracts and blockchain technology. The proposed system uses a permissioned blockchain and a consensus mechanism based on proof-of-authority (PoA). The paper also proposes a system architecture for the proposed voting system.

2.8 "Blockchain-Based Voting System: A Systematic Review"

Author Name: N. N. Abdul Razak, N. M. Abdul Rahman, and N. N. Abdul Aziz

Year of Publish: 2018

The paper provides a systematic review of blockchain-based voting systems. The paper discusses the advantages and disadvantages of using blockchain technology for electronic voting. The paper also provides an overview of the various types of blockchain-based voting systems, such as permissioned and permissionless blockchains.

CHAPTER 3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

System analysis is an important aspect of designing and implementing any technology-based system, including decentralized voting systems using blockchain. In the context of blockchain-based voting, system analysis can help identify the requirements, features, and constraints of the system, and ensure that it meets the needs of stakeholders, such as voters, election officials, and regulators.

Some of the key aspects to consider in system analysis for a decentralized voting system using blockchain include:

Security: Blockchain technology provides inherent security features, such as immutability and cryptographic verification. However, it is important to ensure that the system is designed to prevent attacks, such as double-spending, 51% attacks, and Sybil attacks. This may involve implementing measures such as multi-factor authentication, consensus mechanisms, and digital signatures.

Privacy: One of the key benefits of blockchain-based voting is the ability to maintain anonymity and protect voter privacy. However, it is important to ensure that the system is designed to prevent voter identification and protect the integrity of the vote. This may involve implementing measures such as zero-knowledge proofs and homomorphic encryption.

Accessibility: Decentralized voting systems should be designed to be accessible to all voters, regardless of their technical or physical abilities. This may involve implementing measures such as user-friendly interfaces, multiple language support, and accessibility options for people with disabilities.

Scalability: As the number of voters and transactions increases, decentralized voting systems may face scalability challenges. It is important to design the system to handle a large volume of transactions without compromising security or performance. This may involve implementing measures such as sharing, sidechains, and off-chain solutions.

Regulatory compliance: Decentralized voting systems using blockchain must comply with relevant regulations and laws, such as those related to election processes and data protection. It is important to ensure that the system is designed to meet these requirements.

3.1. EXISTING SYSTEM

The existing system does not have a secure blockchain architecture and also the existing E-voting applications mostly use Aadhar card for verification and this is not the best efficient method for authentication. Since there are lot of important data stored in the application, the application must have strong security layers so it protects from SQL, injection, and other attacks, but the existing system does not have a reliable security system.

It is worth noting that while blockchain technology can provide a secure and transparent platform for online voting, it is not a silver bullet solution. There are still challenges to be addressed, such as ensuring voter privacy, preventing coercion, and ensuring that the technology is accessible to all voters.

3.2. PROPOSED SYSTEM

The proposed system has a secured blockchain architecture and the security layers are well protected so that the data is protected from all attacks. Also In the proposed system, web application is done using React framework which makes the system scalable. Since we are using React JS, our project is efficient in terms of loading speed and performance. Electronic voting is one area in which blockchain has a drastic impact. Due to the high level of impediments, e-voting as an individual system is not a feasible option. If an E-voting system is compromised, the repercussions will be severe. Since a blockchain network is complete, centralized its design ensures that fraud is theoretically impossible until it is adequately implemented. There is nothing coherent with blockchain technology that precludes its application to other cryptocurrencies. The concept of using blockchain technology to create a foolproof e-voting network is gaining traction. A blockchain-based voting system would be indistinguishable to end users from a conventional electronic voting system to cast votes. Encryption plays a critical role in the security of blockchain systems. In a blockchain, each block contains a unique cryptographic hash of the previous block, creating a chain of blocks that are virtually impossible to alter without detection. This makes blockchains a secure and tamper-proof way to store data. Encryption in a blockchain is typically achieved using public-key cryptography, which involves the use of two keys: a public key and a private key. The public key can be shared with others, while the private key is kept secret. When data is encrypted using the public key, only the private key can decrypt and access the information. In a blockchain system, each user has a unique public-private key pair, which is used to sign and verify transactions. When a user initiates a transaction, they use their private key to sign the transaction, which is then broadcast to the network. Other nodes in the network can verify the transaction

using the user's public key, ensuring that the transaction is valid and secure. Encryption can also be used to protect the contents of smart contracts; Smart contracts are computer programs that run on a blockchain network and automate the execution of contract terms between two or more parties. Smart contracts work by encoding the terms of an agreement in code and storing it on the blockchain. When certain conditions are met, such as the receipt of payment or the completion of a task, the contract is automatically executed, without the need for intermediaries such as lawyers or banks. This makes the process of executing contracts faster, cheaper, and more secure.

3.3. Feasibility Study

Technical feasibility: Blockchain technology offers the possibility of creating a decentralized online voting system that is transparent, secure, and tamper-proof. However, the feasibility study should consider the technical challenges of implementing such a system, such as the scalability of the blockchain network, the latency of the transactions, and the potential risks of attacks on the network.

Legal feasibility: The feasibility study should consider the legal aspects of online voting, such as the legal framework that governs voting in the relevant jurisdiction. The study should determine whether the use of blockchain technology is compatible with the legal requirements and whether there are any regulatory barriers to implementing a decentralized online voting system.

Security feasibility: The feasibility study should assess the security risks associated with the decentralized online voting system. The study should consider the potential vulnerabilities in the system and the risks of cyberattacks, hacking, and other threats. The study should also evaluate the potential security

measures that can be implemented to mitigate these risks.

Cost feasibility: The feasibility study should evaluate the costs associated with implementing a decentralized online voting system. This includes the costs of developing and maintaining the blockchain network, as well as the costs of training personnel and educating the public on how to use the system.

User adoption feasibility: The feasibility study should assess the willingness of voters to adopt a decentralized online voting system. The study should consider factors such as the familiarity of the users with blockchain technology, the accessibility of the system, and the user experience of the platform.

3.4. Hardware Environment



Disk space
350 GB



Download
500 MB/day (15 GB/month)*



Upload
5 GB/day (150 GB/month)



Memory (RAM)
1 GB



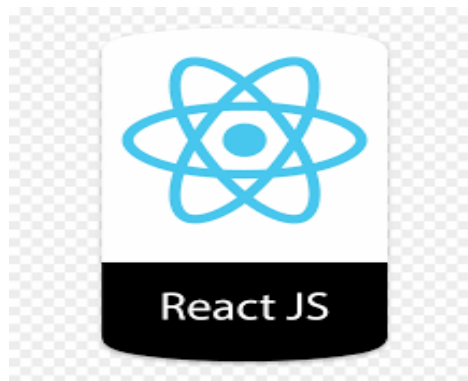
System
Desktop
Laptop
Some ARM chipsets >1 GHz



Operating system
Windows 7/8.x/10
Mac OS X
Linux

3.5. Software Environment

Front-end: Java Script, CSS, HTML, React JS



Back-end: Solidity, Python



CHAPTER 4

SYSTEM DESIGN

4. SYSTEM DESIGN

The design and the working of the whole system is organized into two modules which includes Blockchain Wallet Integration and Online web management. This graphic provides a concise and understandable description of all the entities currently integrated into the system. The diagram shows how the many actions and choices are linked together. You might say that the whole process and how it was carried out is a picture. The figure below shows the functional connections between various entities.

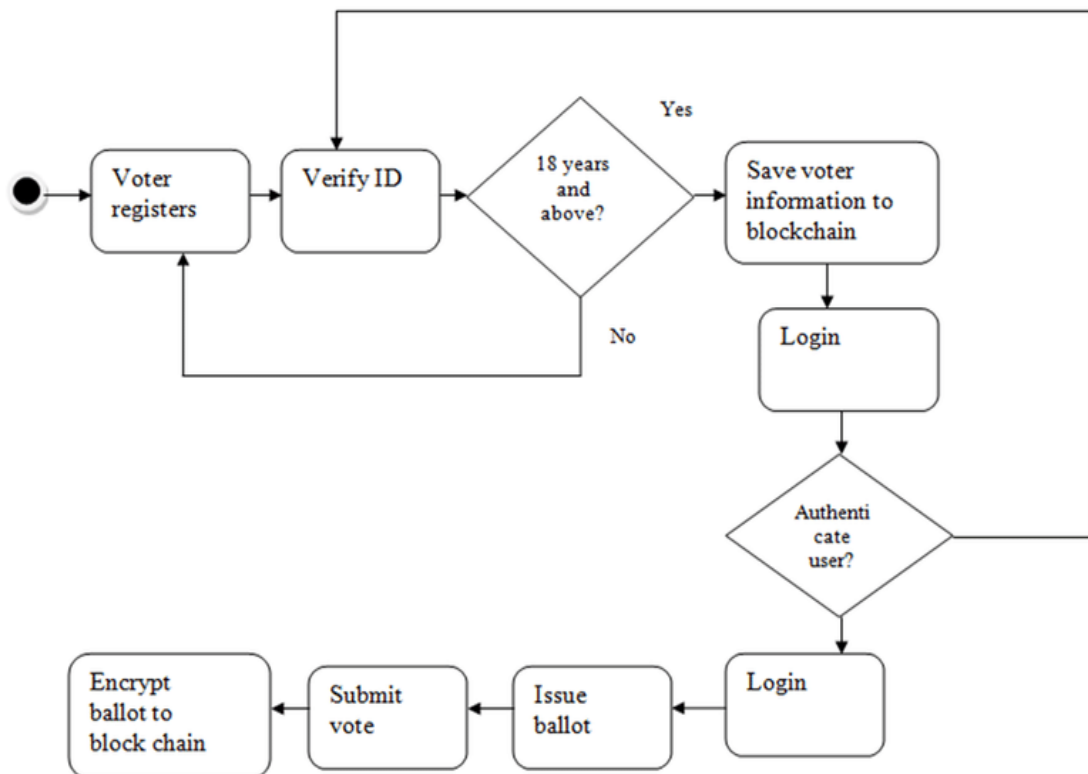


Fig-4.1 System Design Process

4.1. ENTITY-RELATIONSHIP DIAGRAM

Entity Relationship Diagram, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design. An ERD contains different symbols and connectors that visualize two important information The major entities within the system scope, and the interrelationships among these entities. A basic ER model is composed of entity types and specifies relationships that can exist between entities.

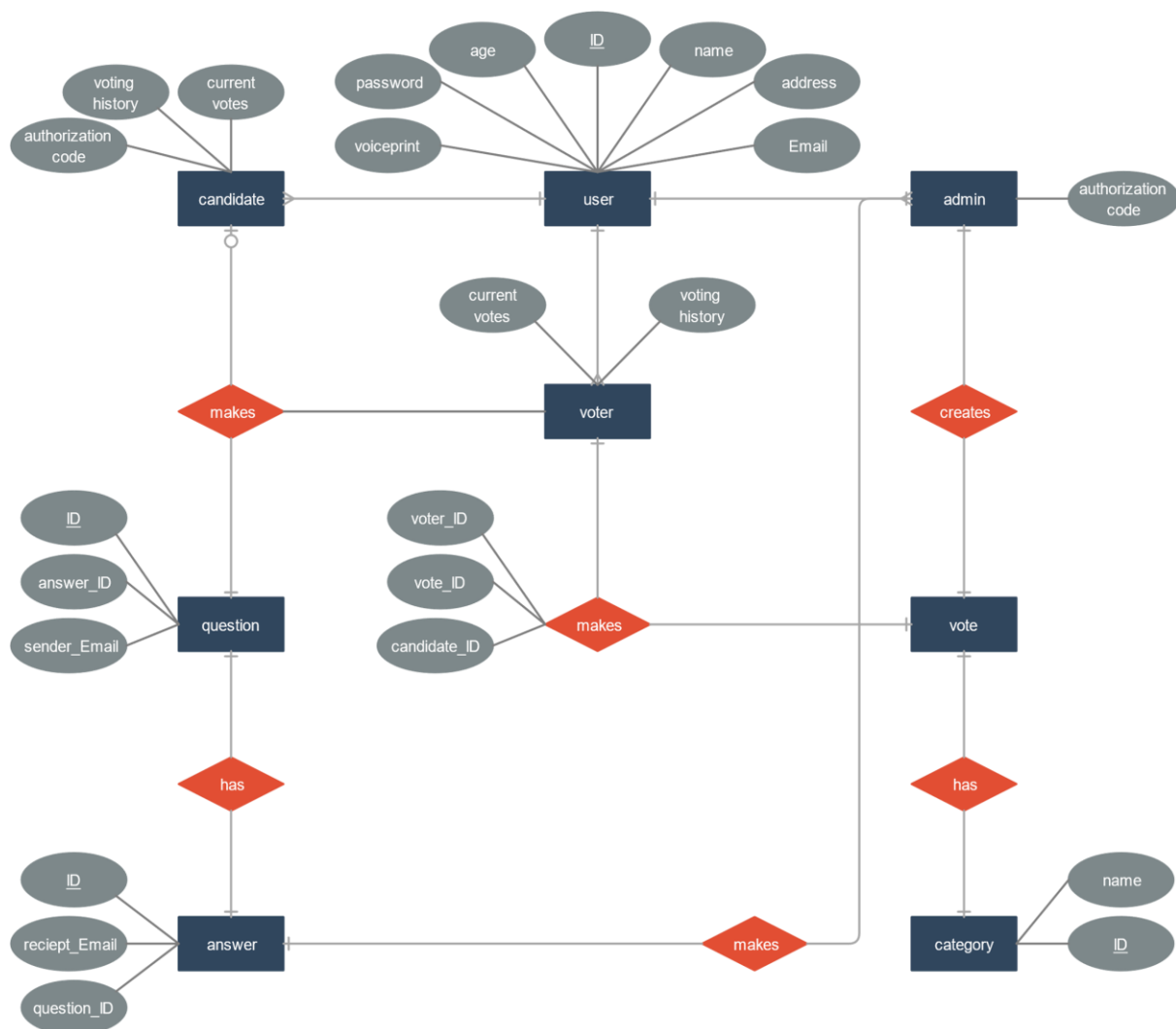


Fig-4.2 E-R Diagram

4.2. Data Dictionary

Data dictionary includes the main entities involved in a decentralized online voting system using blockchain, such as voters, elections, ballots, and the blockchain itself. It also includes relevant attributes for each entity, such as unique identifiers, digital signatures, and timestamps.

Voter

voter_id: unique identifier for the voter

name: name of the voter

email: email address of the voter

public_key: public key for the voter's digital signature

encrypted_private_key: encrypted private key for the voter's digital signature

vote_cast: boolean value indicating whether the voter has cast their vote or not

vote: the voter's encrypted vote

Election

election_id: unique identifier for the election

title: title of the election

description: description of the election

start_time: start time of the election

end_time: end time of the election

candidates: list of candidates running in the election

Ballot

ballot_id: unique identifier for the ballot

election_id: identifier for the election associated with the ballot

voter_id: identifier for the voter associated with the ballot

signature: digital signature of the ballot

encrypted_vote: encrypted vote cast by the voter

timestamp: timestamp of when the ballot was cast

Blockchain

block_id: unique identifier for the block

previous_block_hash: hash of the previous block in the chain

nonce: arbitrary value used in proof-of-work consensus mechanism

transactions: list of transactions (ballots) included in the block

timestamp: timestamp of when the block was added to the chain

4.3. Table Normalization

Table normalization is the process of organizing a relational database into tables with well-defined relationships between them, while reducing data redundancy and dependencies. The goal of normalization is to eliminate data duplication and inconsistencies, which can cause problems such as update anomalies and data inconsistencies.

There are several levels of normalization, each with increasing levels of normalization and reduction of data redundancy. The most common levels of normalization are:

First Normal Form (1NF): Each table has a primary key, and each column contains only atomic (indivisible) values.

Second Normal Form (2NF): All non-key attributes depend on the primary key, and no partial dependencies exist.

Third Normal Form (3NF): All non-key attributes depend only on the primary key, and no transitive dependencies exist.

There are also higher levels of normalization, such as Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF), but these are less commonly used.

By normalizing tables in a database, we can reduce data redundancy and improve data integrity, making it easier to maintain, query, and analyze the data.

Voter Table

The Voter table contains information about the voters, including their unique identifier, name, email, public key, encrypted private key, and whether they have cast their vote or not.

voter_id (primary key)

name

email

public_key

encrypted_private_key

vote_cast

Election Table

The Election table contains information about the elections, including their unique identifier, title, description, start time, and end time.

election_id (primary key)

title

description

start_time

end_time

Candidate Table

The Candidate table contains information about the candidates running in an election, including their unique identifier, name, and the election they are associated with.

candidate_id (primary key)

name

election_id (foreign key)

Ballot Table

The Ballot table contains information about the ballots cast by the voters, including their unique identifier, the voter who cast the ballot, the election the ballot is associated with, the signature of the ballot, the encrypted vote, and the timestamp of when the ballot was cast.

ballot_id (primary key)

voter_id (foreign key)

election_id (foreign key)

signature

encrypted_vote

timestamp

Blockchain Table

The Blockchain table contains information about the blockchain, including the unique identifier of each block, the hash of the previous block in the chain, the nonce, and the timestamp.

block_id (primary key)

previous_block_hash

nonce

timestamp

Blockchain Transactions Table

The Blockchain Transactions table links the ballots to the blocks in the blockchain, including the unique identifier of each transaction, the block the transaction is associated with, and the ballot being included in the transaction.

transaction_id (primary key)

block_id (foreign key)

ballot_id (foreign key)

By normalizing the tables in this way, we can ensure that the data is organized efficiently and without redundancy, making it easier to query and analyze the data.

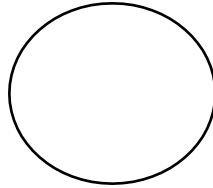
4.4. Data Flow Diagram

To illustrate the movement of information throughout a procedure or system, one might use a Data-Flow Diagram (DFD). A data-flow diagram does not include any decision rules or loops, as the flow of information is entirely one-way. A flowchart can be used to illustrate the steps used to accomplish a certain data-driven task. Several different notations exist for representing data-flow graphs. Each data flow must have a process that acts as either the source or the target of the information exchange. Rather than utilizing a data-flow diagram, users of UML often substitute an activity diagram. In the realm of data-flow plans, site-oriented data-flow plans are a subset. Identical nodes in a data-flow diagram and a Petri net can be thought of as inverted counterparts since the semantics of data memory are represented by the locations in the network. Structured data modeling (DFM) includes processes, flows, storage, and terminators.

Data Flow Diagram Symbols

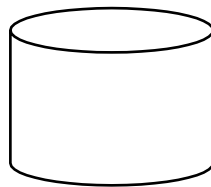
Process

A process is one that takes in data as input and returns results as output.



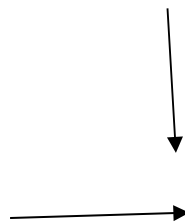
Data Store

In the context of a computer system, the term "data stores" is used to describe the various memory regions where data can be found. In other cases, "files" might stand in for data.



Data Flow

Data flows are the pathways that information takes to get from one place to another.



External Entity

In this context, "external entity" refers to anything outside the system with which the system has some kind of interaction. These are the starting and finishing positions for inputs and outputs, respectively.



Level 0 DFD

Level 0 DFDs, also known as context diagrams, are the most basic data flow diagrams. They provide a broad view that is easily digestible but offers little detail. Level 0 data flow diagrams show a single process node and its connections to external entities. For instance, the example shown below illustrates the hotel reservation process with the flow of information between admin and guests.

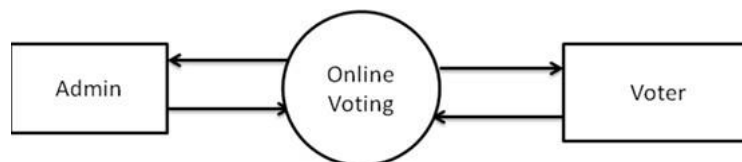


Fig-4.3 Level-0-DFD

Level 1 DFD

Level 1 DFDs are still a general overview, but they go into more detail than a context diagram. In level 1 DFD, the single process node from the context diagram is broken down into sub-processes. As these processes are added, the diagram will need additional data flows and data stores to link them together. In the hotel reservation example, this can include adding the room selection and inquiry processes to the reservation system, as well as data stores.

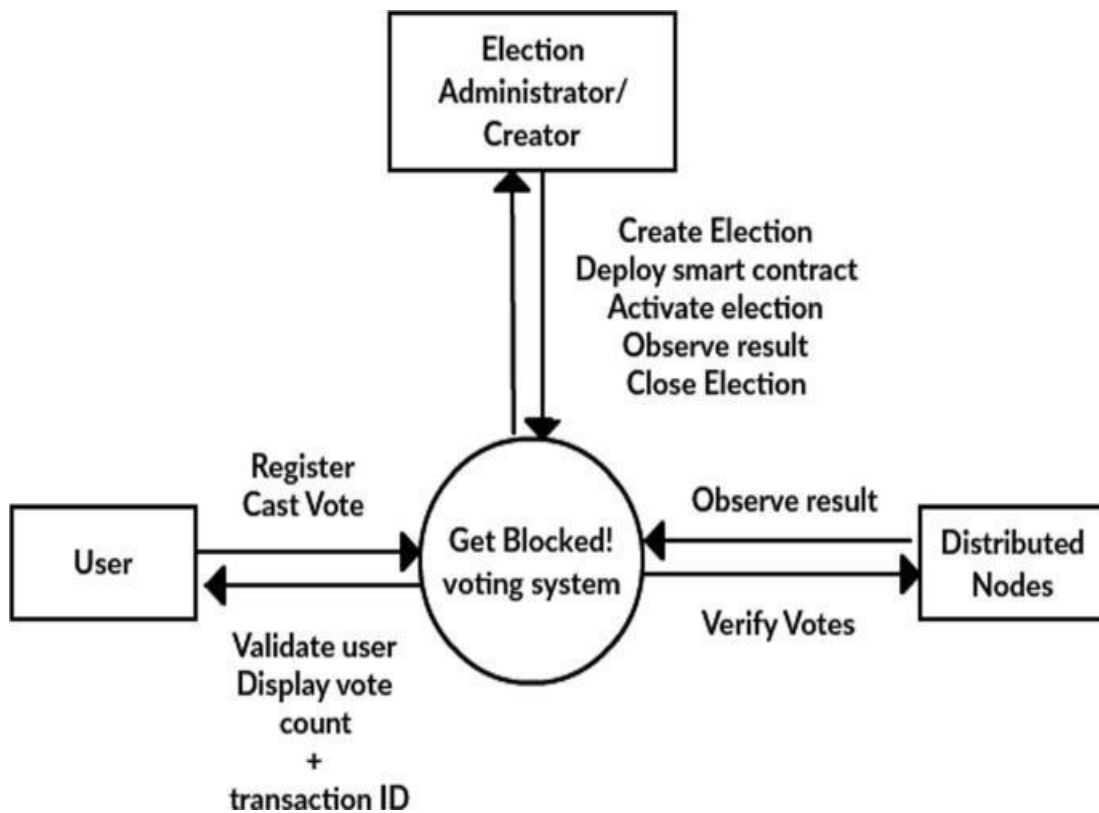


Fig-4.4 Level-1-DFD

4.5. UML Diagrams

UML (Unified Modeling Language) diagrams are graphical representations used in software development to visualize, communicate, and document various aspects of a system. UML diagrams can be used throughout the software development lifecycle, from requirements gathering and analysis to design, implementation, and testing. They can help ensure that all stakeholders have a common understanding of the system, improve communication and collaboration, and support more effective decision-making.

4.5.1. Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

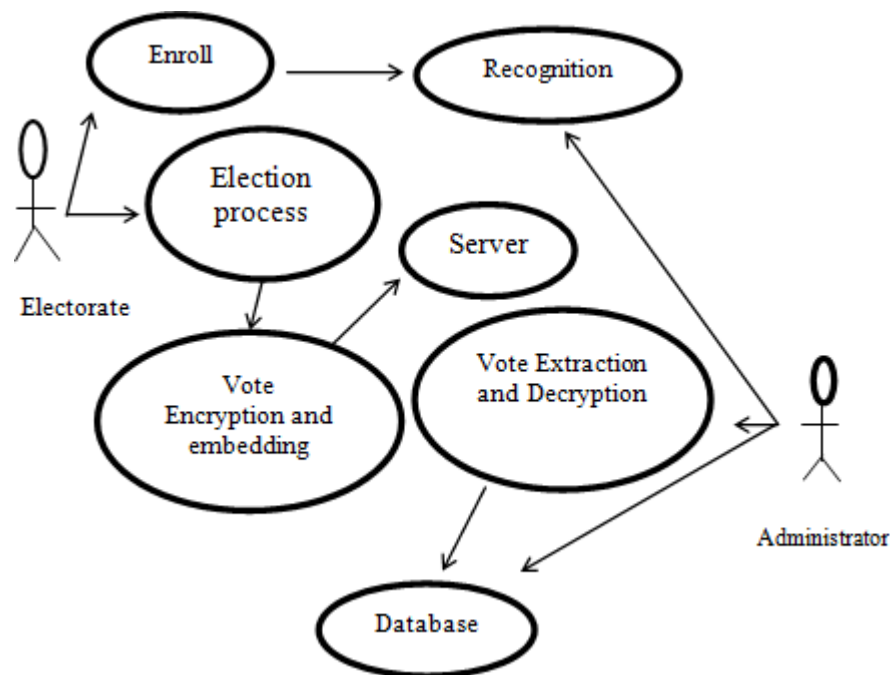


Fig-4.5 Use-Case Diagram

The use cases are represented by either circles or ellipses. While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a higher-level view of the system. Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders. The drawings attempt to mimic the real world and provide a view for the stakeholder to understand how the system is going to be designed.

4.5.2. Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function.

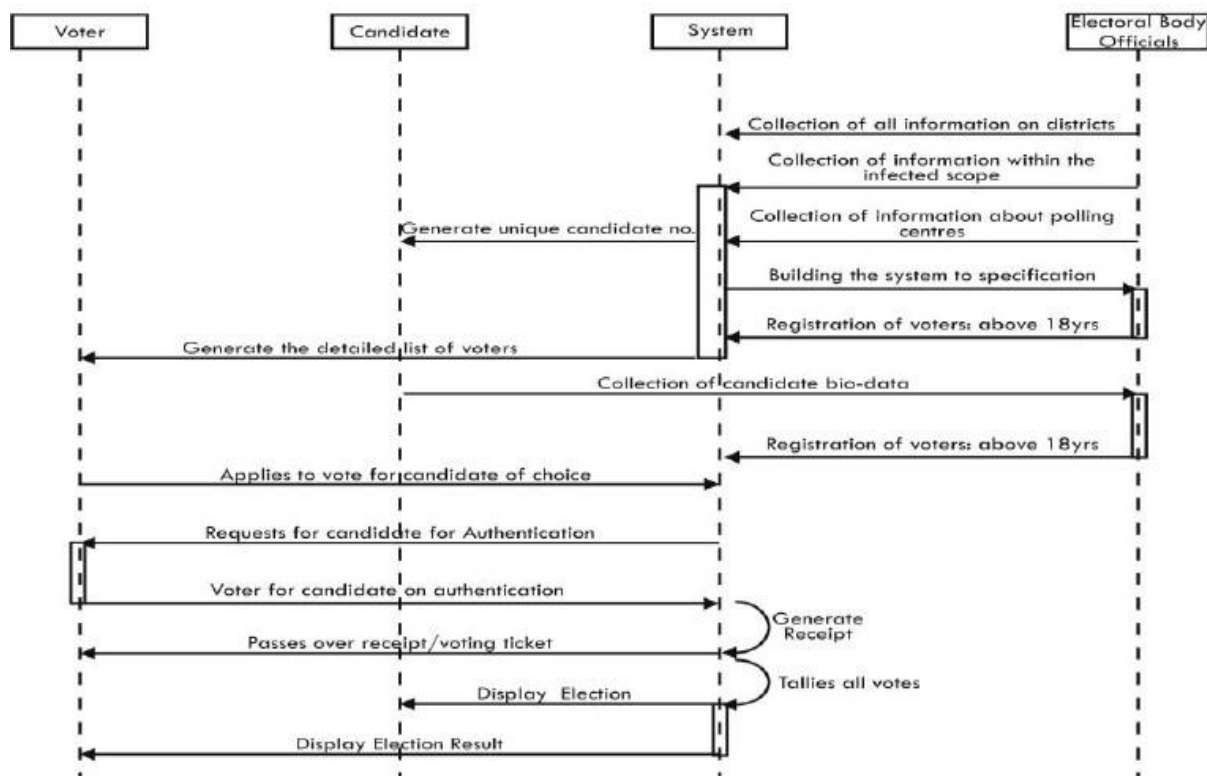


Fig-4.6 Sequence Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

4.5.3. Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the structure of the application, and for detailed modeling translating the models into programming code.

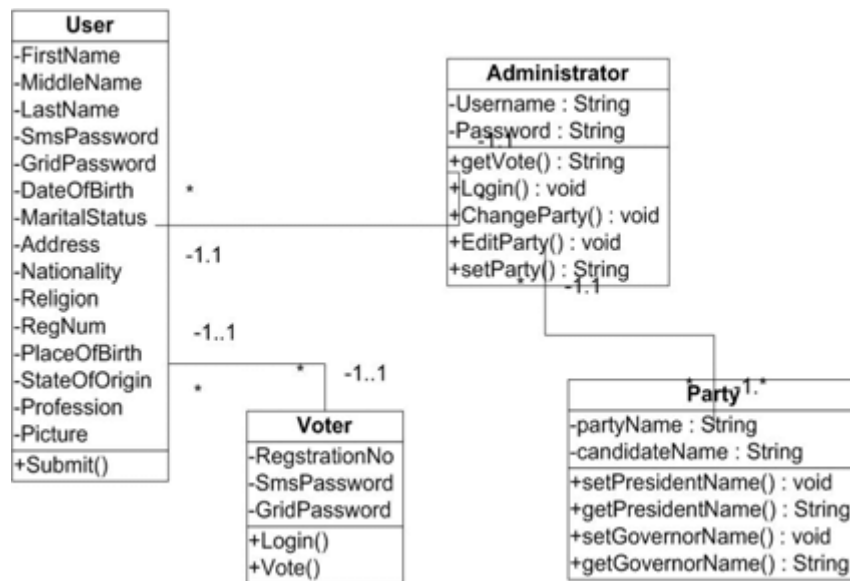


Fig-4.7 Class Diagram

Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

4.5.4. Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram that represents the workflow or the sequence of activities in a system, process or use case. It is a visual representation of the steps, decisions and actions that occur in a particular process.

Activity diagrams consist of activities, which are the basic units of work, and transitions, which represent the paths between activities. Activities are represented by rounded rectangles, while transitions are represented by arrows. The starting point of the diagram is represented by a filled circle, and the ending point is represented by a hollow circle.

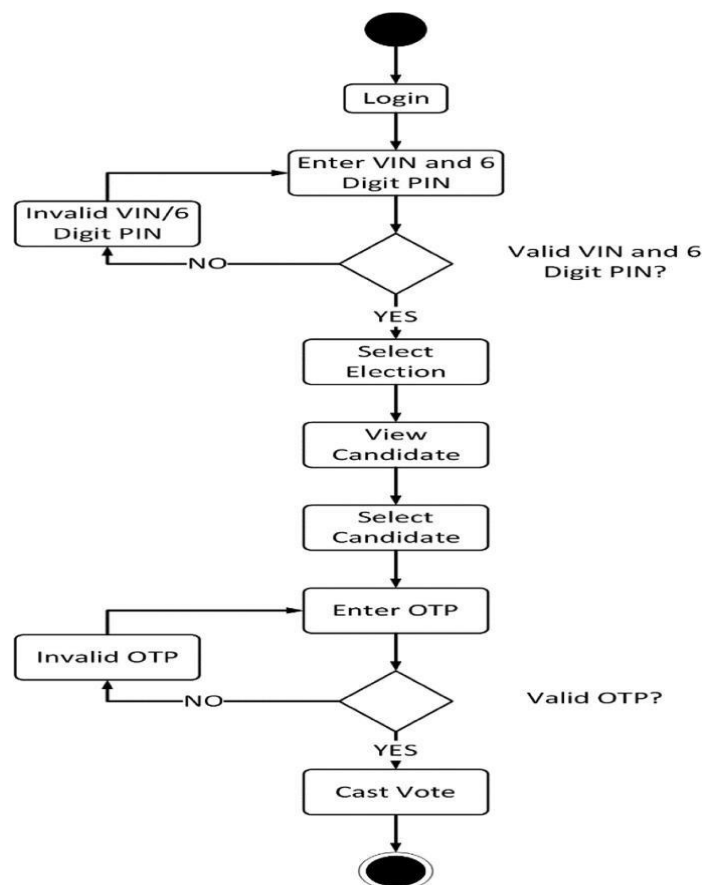


Fig-4.8 Activity Diagram

CHAPTER 5

SYSTEM ARCHITECTURE

5. SYSTEM ARCHITECTURE

5.1 ARCHITECTURE DIAGRAM

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

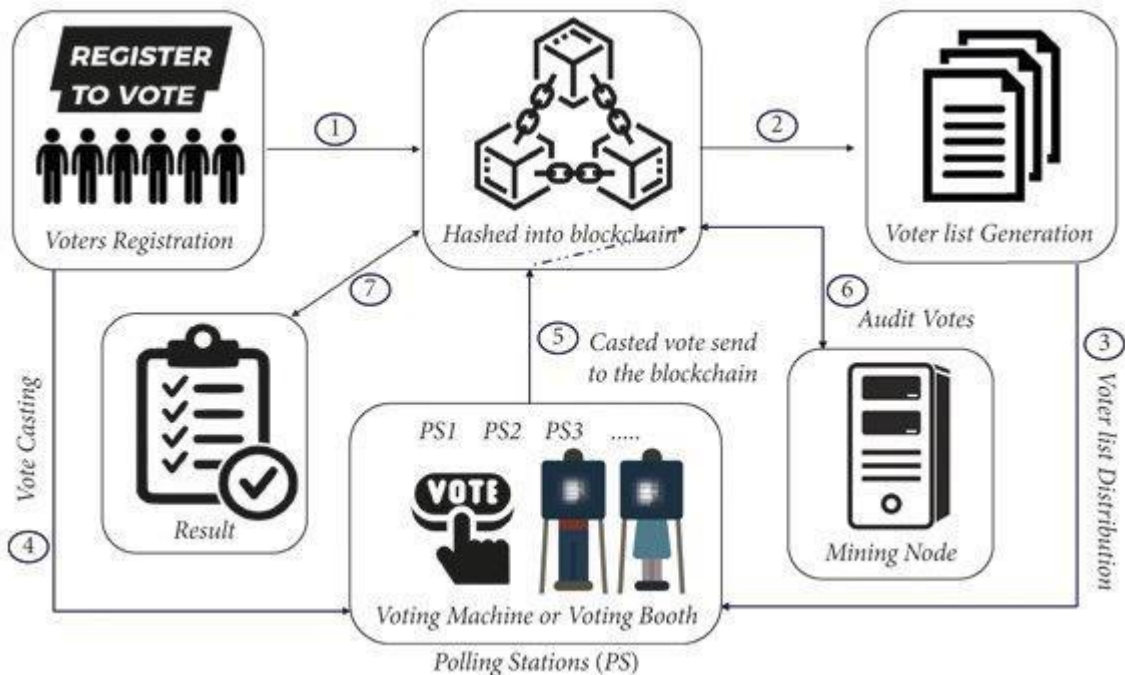


Fig-5.1 System Architecture design

The key components of a system architecture may include the operating system, the hardware components, the network infrastructure, the application software, and the data storage and retrieval mechanisms. The architecture may also involve considerations such as scalability, security, and reliability. The goal of system architecture is to create a system that meets the requirements of its users while minimizing costs and maximizing performance.

CHAPTER 6

SYSTEM

IMPLEMENTATION

6. SYSTEM IMPLEMENTATION

6.1 MODULE DESIGN SPECIFICATION

A requirement specification is a collection of all requirements that are to be imposed on the design and verification of the product. The specification also contains other related information necessary for the design, verification, and maintenance of the product

- Login Module
- Initializing a new ballot
- Voting Process
- Resultant Module

6.1.1 LOGIN MODULE

Unlike traditional web2 applications where the user logs in to their account using a username and a password, web3 applications use wallets to connect to a user account. The public key of an account uniquely identifies the user whereas the private key is used for authorization. In our voting process, we use meta mask wallet which is available as a browser extension. To log in, the user must simply connect the app to an account from the wallet.

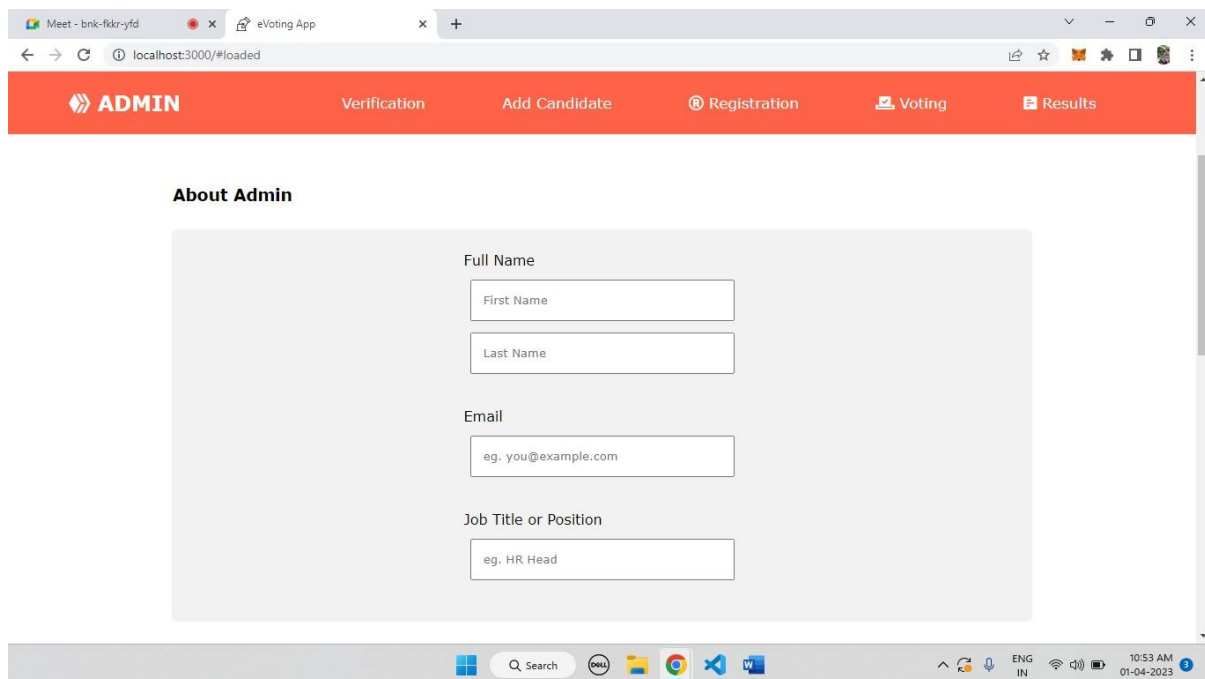
Choose a blockchain platform: The first step is to choose a blockchain platform that will serve as the backbone of the voting system. Some popular options include Ethereum, Hyperledger Fabric, and EOS. Once you have chosen a platform, you will need to create a smart contract that will manage the login process.

Define the login process: The login process should be designed in a way that ensures security and transparency. One approach is to use a two-factor authentication process that requires voters to provide a password and a unique

identifier such as a fingerprint or facial recognition. You can also use cryptographic protocols like zero-knowledge proofs to provide additional security.

Create a voter registry: Before voters can log in to the system, they must be registered. This involves collecting their personal information and verifying their identity. The voter registry should be stored on the blockchain to ensure that it is tamper-proof.

Develop a user interface: The user interface should be designed in a way that is intuitive and easy to use. It should provide clear instructions on how to log in and cast a vote. You can also include features such as password reset and account recovery to ensure that voters can access their accounts in case of any issues.



The screenshot displays a web browser window with the URL `localhost:3000/#loaded`. The browser tabs include 'Meet - bnk-fkkr-yfd' and 'eVoting App'. The application's header is orange and contains the 'ADMIN' logo and navigation links: 'Verification', 'Add Candidate', 'Registration', 'Voting', and 'Results'. The main content area is titled 'About Admin' and features a light gray form with the following fields:

- Full Name:** Two input fields for 'First Name' and 'Last Name'.
- Email:** One input field with the placeholder text 'eg. you@example.com'.
- Job Title or Position:** One input field with the placeholder text 'eg. HR Head'.

The Windows taskbar at the bottom shows the system clock as 10:53 AM on 01-04-2023, along with various system icons and a search bar.

Fig-6.1 Login Module

6.1.2 INITIALIZING A NEW BALLOT

- Any user can start a new ballot by specifying the information regarding the ballot and by authorizing the transaction using the wallet.
- The transaction is sent to the blockchain network and upon success is appended to the ledger permanently.
- The ballot creator must add a list of addresses (public address of user accounts) that are eligible to vote.

The screenshot displays the 'ADMIN' interface of the 'eVoting App' in a web browser. The top navigation bar is orange and contains links for 'Verification', 'Add Candidate', 'Registration', 'Voting', and 'Results'. The main content area is titled 'About Election' and features a light gray form with two input fields: 'Election Title' (with placeholder text 'eg. School Election') and 'Organization Name' (with placeholder text 'eg. Lifeline Academy'). Below the form, a yellow warning box states: 'Do not forget to add candidates. Go to [add_candidates](#) page.' The browser's address bar shows 'localhost:3000/#loaded'. The Windows taskbar at the bottom indicates the time is 10:54 AM on 01-04-2023.

Fig-6.2 Initializing a new ballot Module

6.1.3 VOTING PROCESS

- A user who is eligible to vote can connect their accounts in a similar manner using meta mask.
- If the match is found, the voter is then presented with a list of available candidates with the option to cast vote against them. On the contrary, if the match is unsuccessful, any further access would be denied.

- The user's vote is recorded on the ledger as a transaction is sent to the network by signing a transaction in the wallet.

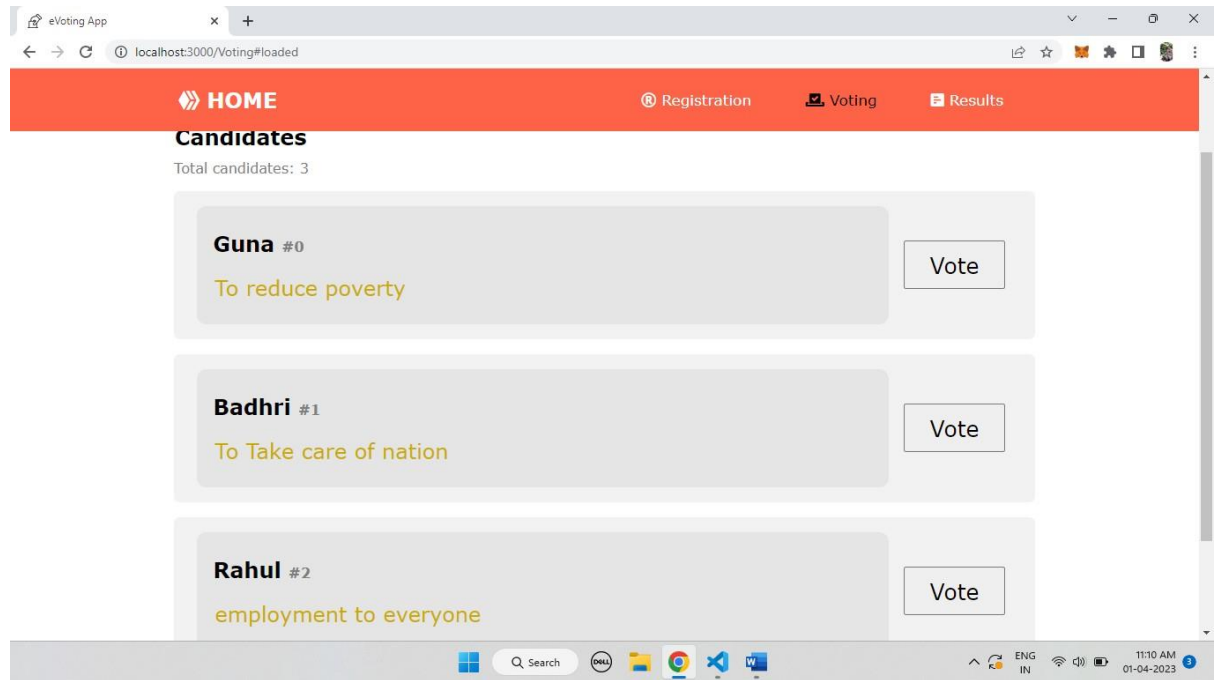


Fig-6.3 Voting Process Module

- To this end, a successful vote cast is considered as a transaction within the blockchain of the voting application.

6.1.4 RESULTANT PROCESS

At the end of the specified ballot time, all users can view the results of the ballot. A vote cast is added as a new block (after successful mining) in the blockchain as well as being recorded in data tables at the backend of the database. The system ensures only one-person, one-vote (democracy) property of voting systems. In the evaluation process, it was found that while sending the asset to the address, the transaction hash was generated carrying the transfer of vote. At the end of the voting cycle, each candidate's wallet balance can be checked, and the candidate with the highest tally wins. Results can be verified easily as all transaction details are stored in the public

blockchain. Application flow is as follows: Each voter using his wallet will transact a token to a candidate's wallet (the interface will be handled by app automatically).

The transaction of a token represents casting of a vote. A voter will get a transaction ID which can be used to verify that his vote has been correctly casted. After the voting phase, the candidate's wallet balance will show the final tally. Tally of all candidates can now be compared to decide and set the winner.

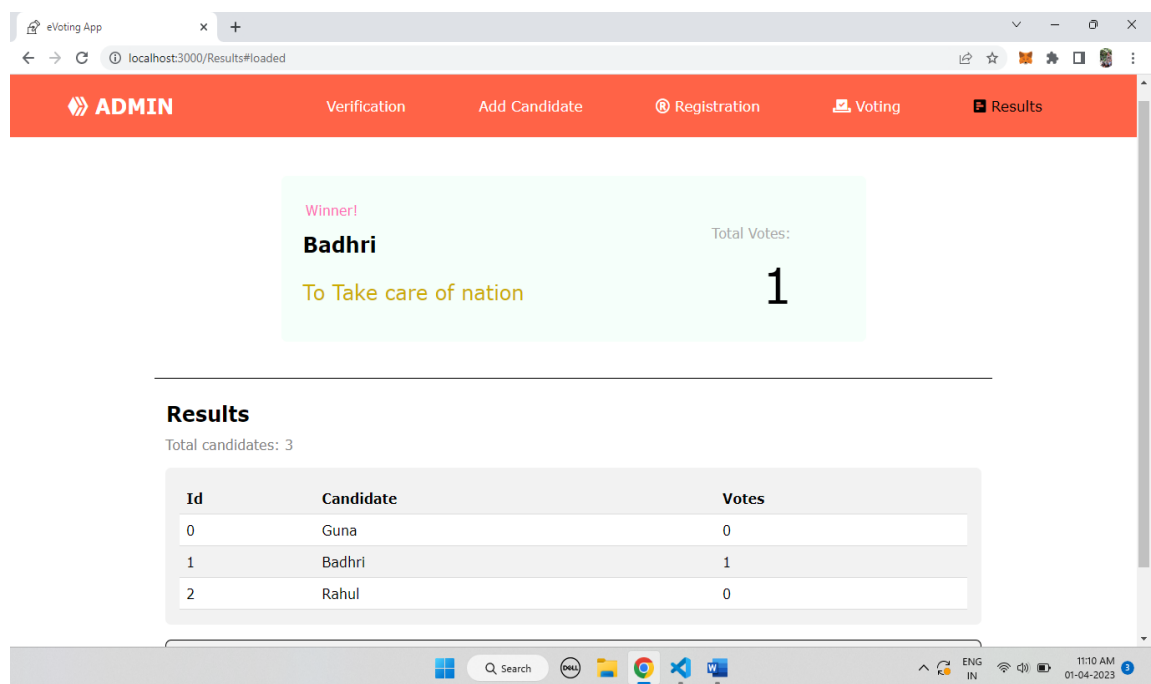


Fig-6.4 Resultant Module

CHAPTER 7

SYSTEM TESTING

7. SYSTEM TESTING

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.

In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.

We have tested a list of disease with our model in order to determine the best one some of the crucial and best results we have mentioned below.

7.1 Unit Testing

Unit testing for a decentralized online voting system using blockchain involves testing the various components of the system in isolation to ensure that they function correctly and meet the requirements of the system. Here are some steps that can be followed to perform unit testing for such a system:

Identify the components: The first step is to identify the various components of the system that need to be tested. In a decentralized online voting system using blockchain, these components might include the smart contracts, the user interface, the cryptographic algorithms, and the network communication protocols.

Write test cases: Once the components have been identified, write test cases for each of them. These test cases should cover all possible scenarios that the component might encounter, including both expected and unexpected inputs.

Test the smart contracts: The smart contracts are the heart of a blockchain-based system. They define the rules for the voting process and ensure the integrity of the system. Test the smart contracts thoroughly to ensure that they work as expected and that they can handle all possible scenarios.

Test the user interface: The user interface is the part of the system that users interact with. Test the user interface to ensure that it is user-friendly, easy to use, and that it can handle all possible user inputs.

Test the cryptographic algorithms: Cryptographic algorithms are used to ensure the security and privacy of the voting process. Test the cryptographic algorithms to ensure that they work as expected and that they can handle all possible inputs.

Test the network communication protocols: Network communication protocols are used to ensure that the system can communicate with other nodes on the blockchain network. Test the network communication protocols to ensure that they work as expected and that they can handle all possible network conditions.

Test the system as a whole: Once all the components have been tested in isolation, test the system to ensure that they work together seamlessly and that the system meets all the requirements.

In summary, unit testing for a decentralized online voting system using blockchain involves testing each component of the system in isolation and then testing the system to ensure that it meets all the requirements and works as expected.

7.2 Integration Testing

Integration testing for a decentralized online voting system using blockchain is an important step in ensuring the reliability and security of the system. Here are some key steps and considerations for conducting integration testing for such a system:

Define the scope of testing: The first step is to define the scope of testing. This includes identifying the different components of the system that need to be tested and the interactions between them.

Identify test scenarios: Based on the scope of testing, identify different test scenarios that need to be executed. This includes scenarios related to authentication, vote casting, vote counting, and result verification.

Develop test cases: Once the test scenarios are identified, develop test cases for each scenario. Test cases should cover all possible scenarios, including positive and negative test cases.

Set up the test environment: Set up the test environment by configuring the blockchain network, deploying the smart contracts, and configuring the nodes.

Report defects: If any defects are found during testing, report them in a defect tracking system. The defects should be prioritized based on their severity and fixed before the system is released for production.

Document test results: Document the test results, including any defects found and their resolution. This documentation will be helpful in future releases of the system.

In summary, integration testing for a decentralized online voting system using blockchain involves defining the scope of testing, identifying test scenarios, developing test cases, setting up the test environment, executing test cases, reporting defects, and documenting test results.

7.3 Test Cases and Test Results

Test Case ID	Test Case Name	Test Objective	Test Steps	Expected Result	Actual Result	Pass/Fail
UT001	Smart Contract Test	To verify the smart contract functions as expected	1. Deploy the smart contract	The smart contract is deployed successfully	Pass	Pass
UT002	User Creation Test	To verify that users can be created successfully	1. Create a new user	The new user is created successfully	Pass	Pass
UT003	User Authentication Test	To verify that users can be authenticated successfully	1. Enter the correct username and password	The user is authenticated successfully	Pass	Pass
UT004	Vote Casting Test	To verify that a user can cast their vote successfully	1. Select a candidate to vote for 2. Submit the vote	The vote is recorded successfully and added to the blockchain	Pass	Pass
UT005	Vote Counting Test	To verify that the votes are counted accurately	1. Count the votes for each candidate 2. Verify the vote count with the blockchain data	The vote count matches the blockchain data	Pass	Pass

Table 7.1 Test case and Test Results

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8. CONCLUSION AND FUTURE ENHANCEMENT

8.1 RESULTS & DISCUSSION

Decentralized online voting systems that use blockchain technology have the potential to increase transparency, security, and trust in voting processes. In this system, a distributed network of nodes validates and records votes on a blockchain, creating an immutable and tamper-proof ledger of all votes cast.

One major advantage of this system is that it eliminates the need for a centralized authority, which reduces the risk of tampering and manipulation. The use of cryptography and smart contracts can also ensure that votes are cast anonymously, securely, and accurately.

However, there are still some challenges and limitations to consider. One of the biggest concerns is the potential for voter fraud, particularly through vote buying and coercion. Additionally, there are technical issues such as scalability and the cost of running a blockchain network that may need to be addressed.

Overall, decentralized online voting systems using blockchain technology have the potential to increase trust and confidence in the democratic process, and provide a more secure and transparent way for people to cast their votes.

8.2 CONCLUSION

Decentralized online voting systems using blockchain technology have the potential to revolutionize the way we conduct elections and increase trust in the democratic process. By leveraging the power of distributed ledger technology, these systems can provide an immutable and tamper-proof record of all votes cast, which can greatly reduce the risk of fraud and manipulation.

While there are still challenges to overcome, such as scalability and the cost of running a blockchain network, continued research and development will likely lead to more sophisticated and efficient solutions.

The benefits of decentralized online voting systems are clear: increased transparency, security, and accessibility for voters, as well as reduced costs and administrative burden for election officials. With proper safeguards and regulations in place, decentralized online voting systems using blockchain technology could become a valuable tool for enhancing the integrity of democratic processes around the world.

8.3 FUTURE ENHANCEMENT

There are several potential enhancements for decentralized online voting systems using blockchain technology that could be implemented in the future. Some of these include:

- **Integration with other technologies:** Decentralized online voting systems could be integrated with other technologies such as biometric authentication, artificial intelligence, and machine learning to enhance security and accuracy.
- **Improved scalability:** Scalability is a major challenge for blockchain-based systems, but there are several potential solutions that could be explored, such as sharding and off-chain transactions.
- **More user-friendly interfaces:** The user interface of decentralized online voting systems could be improved to make it more intuitive and accessible for a wider range of voters.
- **Enhanced privacy and anonymity:** While blockchain technology provides a high level of security and transparency, there are still concerns around voter privacy and anonymity. Future enhancements could focus on improving these aspects of the system.

- **More widespread adoption:** For decentralized online voting systems to become a viable alternative to traditional voting systems, they will need to be adopted on a larger scale. Future enhancements could focus on improving the accessibility and usability of these systems to encourage wider adoption.

Overall, there is a great deal of potential for decentralized online voting systems using blockchain technology, and continued research and development will likely lead to further enhancements and improvements in the years to come.

APPENDICES

APPENDICES

A.1 CODING

Home.js

```
// Node modules
import React, { Component } from "react";
import { useForm } from "react-hook-form";
import { Link } from "react-router-dom";

// Components
import Navbar from "../Navbar/Navigation";
import NavbarAdmin from "../Navbar/NavigationAdmin";
import UserHome from "../UserHome";
import StartEnd from "../StartEnd";
import ElectionStatus from "../ElectionStatus";

// Contract
import getWeb3 from "../getWeb3";
import Election from "../contracts/Election.json";

// CSS
import "../Home.css";

// const buttonRef = React.createRef();
export default class Home extends Component {
  constructor(props) {
    super(props);
    this.state = {
      ElectionInstance: undefined,
      account: null,
      web3: null,
      isAdmin: false,
      elStarted: false,
      elEnded: false,
      elDetails: {},
    };
  }

  // refreshing once
  componentDidMount = async () => {
    if (!window.location.hash) {
      window.location = window.location + "#loaded";
    }
  }
}
```

```

// Get network provider and web3 instance.const
web3 = await getWeb3();

// Use web3 to get the user's accounts.
const accounts = await web3.eth.getAccounts();

// Get the contract instance.
const networkId = await web3.eth.net.getId();
const deployedNetwork = Election.networks[networkId];
const instance = new web3.eth.Contract(
  Election.abi,
  deployedNetwork && deployedNetwork.address
);

// Set web3, accounts, and contract to the state, and then proceed with an
// example of interacting with the contract's methods.
this.setState({
  web3: web3,
  ElectionInstance: instance,
  account: accounts[0],
});

const admin = await this.state.ElectionInstance.methods.getAdmin().call();
if (this.state.account === admin) {
  this.setState({ isAdmin: true });
}

// Get election start and end values
const start = await this.state.ElectionInstance.methods.getStart().call();
this.setState({ elStarted: start });
const end = await this.state.ElectionInstance.methods.getEnd().call();
this.setState({ elEnded: end });

// Getting election details from the contract
const adminName = await this.state.ElectionInstance.methods
  .getAdminName()
  .call();
const adminEmail = await this.state.ElectionInstance.methods
  .getAdminEmail()
  .call();
const adminTitle = await this.state.ElectionInstance.methods
  .getAdminTitle()
  .call();
const electionTitle = await this.state.ElectionInstance.methods
  .getElectionTitle()
  .call();

```

```

const organizationTitle = await this.state.ElectionInstance.methods
  .getOrganizationTitle()
  .call();

this.setState({
  elDetails: {
    adminName: adminName,
    adminEmail: adminEmail,
    adminTitle: adminTitle,
    electionTitle: electionTitle,
    organizationTitle: organizationTitle,
  },
});
} catch (error) {
  // Catch any errors for any of the above operations.
  alert(
    `Failed to load web3, accounts, or contract. Check console for details.`
  );
  console.error(error);
}
};
// end election
endElection = async () => {
  await this.state.ElectionInstance.methods
    .endElection()
    .send({ from: this.state.account, gas: 1000000 });
  window.location.reload();
};
// register and start election
registerElection = async (data) => {
  await this.state.ElectionInstance.methods
    .setElectionDetails(
      data.adminFName.toLowerCase() + " " + data.adminLName.toLowerCase(),
      data.adminEmail.toLowerCase(),
      data.adminTitle.toLowerCase(),
      data.electionTitle.toLowerCase(),
      data.organizationTitle.toLowerCase()
    )
    .send({ from: this.state.account, gas: 1000000 });
  window.location.reload();
};

render() {
  if (!this.state.web3) {
    return (
      <

```

```

    <Navbar />
    <center>Loading Web3, accounts, and contract...</center>
  </>
);
}
return (
  <
    {this.state.isAdmin ? <NavbarAdmin /> : <Navbar />}
    <div className="container-main">
      <div className="container-item center-items info">
        Your Account: {this.state.account}
      </div>
      {!this.state.elStarted & !this.state.elEnded ? (
        <div className="container-item info">
          <center>
            <h3>The election has not been initialize.</h3>
            {this.state.isAdmin ? (
              <p>Set up the election.</p>
            ) : (
              <p>Please wait..</p>
            )}
          </center>
        </div>
      ) : null}
    </div>
    {this.state.isAdmin ? (
      <
        <this.renderAdminHome />
      </>
    ) : this.state.elStarted ? (
      <
        <UserHome el={this.state.elDetails} />
      </>
    ) : !this.state.isElStarted && this.state.isElEnded ? (
      <
        <div className="container-item attention">
          <center>
            <h3>The Election ended.</h3>
            <br />
            <Link
              to="/Results"
              style={{ color: "black", textDecoration: "underline" }}
            >

```

```

        See results
      </Link>
    </center>
  </div>
</>
) : null}
</>
);
}

renderAdminHome = () => {
  const EMsg = (props) => {
    return <span style={{ color: "tomato" }}>{props.msg}</span>;
  };

  const AdminHome = () => {
    // Contains of Home page for the Admin
    const {
      handleSubmit,
      register,
      formState: { errors },
    } = useForm();

    const onSubmit = (data) => {
      this.registerElection(data);
    };

    return (
      <div>
        <form onSubmit={handleSubmit(onSubmit)}>
          { !this.state.elStarted & !this.state.elEnded ? (
            <div className="container-main">
              { /* about-admin */}
              <div className="about-admin">
                <h3>About Admin</h3>
                <div className="container-item center-items">
                  <div>
                    <label className="label-home">
                      Full Name{ " "}
                      {errors.adminFName && <EMsg msg="*required" />}
                    <input
                      className="input-home"
                      type="text"
                      placeholder="First Name"
                      {...register("adminFName", {
                        required: true,

```

```

    }}}
  />
  <input
    className="input-home"
    type="text"
    placeholder="Last Name"
    {...register("adminLName")}
  />
</label>

<label className="label-home">
  Email{ " " }
  {errors.adminEmail && (
    <EMsg msg={errors.adminEmail.message} />
  )}
  <input
    className="input-home"
    placeholder="eg. you@example.com"
    name="adminEmail"
    {...register("adminEmail", {
      required: "*Required",
      pattern: {
        value: /^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,4}$/ ,
// email validation using RegExpmessage: "*Invalid",
      },
    })}
  />
</label>

<label className="label-home">
  Job Title or Position{ " " }
  {errors.adminTitle && <EMsg msg="*required" />}
  <input
    className="input-home"
    type="text"
    placeholder="eg. HR Head "
    {...register("adminTitle", {
      required: true,
    })}
  />
</label>
</div>
</div>
</div>
{ /* about-election */ }
<div className="about-election">
  <h3>About Election</h3>

```

```

    <div>
      <label className="label-home">
        Election Title{ " "}
        {errors.electionTitle && <EMsg msg="*required" />}
        <input
          className="input-home"
          type="text"
          placeholder="eg. School Election"
          {...register("electionTitle", {
            required: true,
          })}
        />
      </label>
      <label className="label-home">
        Organization Name{ " "}
        {errors.organizationName && <EMsg msg="*required" />}
        <input
          className="input-home"
          type="text"
          placeholder="eg. Lifeline Academy"
          {...register("organizationTitle", {
            required: true,
          })}
        />
      </label>
    </div>
  </div>
</div>
) : this.state.elStarted ? (
  <UserHome el={this.state.elDetails} />
) : null}
<StartEnd
  elStarted={this.state.elStarted}
  elEnded={this.state.elEnded}
  endElFn={this.endElection}
/>
<ElectionStatus
  elStarted={this.state.elStarted}
  elEnded={this.state.elEnded}
/>
</form>
</div>
);
};
return <AdminHome />;
};
}

```


Admin.js

```
import React from "react";

const AdminOnly = (props) => {
  return (
    <div className="container-item attention" style={{ borderColor: "tomato" }}>
      <center>
        <div style={{ margin: "17px" }}>
          <h1>{props.page}</h1>
        </div>
        <p>Admin access only.</p>
      </center>
    </div>
  );
};

export default AdminOnly;
```

Voting.js

```
// Node modules
import React, { Component } from "react";
import { Link } from "react-router-dom";

// Components
import Navbar from "../Navbar/Navigation";
import NavbarAdmin from "../Navbar/NavigationAdmin";
import NotInit from "../NotInit";

// Contract
import getWeb3 from "../getWeb3";
import Election from "../contracts/Election.json";

// CSS
import "../Voting.css";

export default class Voting extends Component {
  constructor(props) {
    super(props);
    this.state = {
      ElectionInstance: undefined,
      account: null,
      web3: null,
      isAdmin: false,
      candidateCount: undefined,
      candidates: [],
      isElStarted: false,
```

```

    isElEnded: false,
    currentVoter: {
      address: undefined,
      name: null,
      phone: null,
      hasVoted: false,
      isVerified: false,
      isRegistered: false,
    },
  };
}
componentDidMount = async () => {
  // refreshing once
  if (!window.location.hash) {
    window.location = window.location + "#loaded";
    window.location.reload();
  }
  try {
    // Get network provider and web3 instance.
    const web3 = await getWeb3();

    // Use web3 to get the user's accounts.
    const accounts = await web3.eth.getAccounts();

    // Get the contract instance.
    const networkId = await web3.eth.net.getId();
    const deployedNetwork = Election.networks[networkId];
    const instance = new web3.eth.Contract(
      Election.abi,
      deployedNetwork && deployedNetwork.address
    );

    // Set web3, accounts, and contract to the state, and then proceed with an
    // example of interacting with the contract's methods.
    this.setState({
      web3: web3,
      ElectionInstance: instance,
      account: accounts[0],
    });

    // Get total number of candidates
    const candidateCount = await this.state.ElectionInstance.methods
      .getTotalCandidate()
      .call();
    this.setState({ candidateCount: candidateCount });
  }

```

```

// Get start and end values
const start = await this.state.ElectionInstance.methods.getStart().call();
this.setState({ isEStarted: start });
const end = await this.state.ElectionInstance.methods.getEnd().call();
this.setState({ isEEnded: end });

// Loading Candidates details
for (let i = 1; i <= this.state.candidateCount; i++) {
  const candidate = await this.state.ElectionInstance.methods
    .candidateDetails(i - 1)
    .call();
  this.state.candidates.push({
    id: candidate.candidateId,
    header: candidate.header,
    slogan: candidate.slogan,
  });
}
this.setState({ candidates: this.state.candidates });

// Loading current voter
const voter = await this.state.ElectionInstance.methods
  .voterDetails(this.state.account)
  .call();
this.setState({
  currentVoter: {
    address: voter.voterAddress,
    name: voter.name,
    phone: voter.phone,
    hasVoted: voter.hasVoted,
    isVerified: voter.isVerified,
    isRegistered: voter.isRegistered,
  },
});

// Admin account and verification
const admin = await this.state.ElectionInstance.methods.getAdmin().call();
if (this.state.account === admin) {
  this.setState({ isAdmin: true });
}
} catch (error) {
  // Catch any errors for any of the above operations.
  alert(
    `Failed to load web3, accounts, or contract. Check console for details.`
  );
  console.error(error);
}
};

```

```

renderCandidates = (candidate) => {
  const castVote = async (id) => {
    await this.state.ElectionInstance.methods
      .vote(id)
      .send({ from: this.state.account, gas: 1000000 });
    window.location.reload();
  };
  const confirmVote = (id, header) => {
    var r = window.confirm(
      "Vote for " + header + " with Id " + id + ".\nAre you sure?"
    );
    if (r === true) {
      castVote(id);
    }
  };
  return (
    <div className="container-item">
      <div className="candidate-info">
        <h2>
          {candidate.header} <small>#{candidate.id}</small>
        </h2>
        <p className="slogan">{candidate.slogan}</p>
      </div>
      <div className="vote-btn-container">
        <button
          onClick={() => confirmVote(candidate.id, candidate.header)}
          className="vote-bth"
          disabled={
            !this.state.currentVoter.isRegistered ||
            !this.state.currentVoter.isVerified ||
            this.state.currentVoter.hasVoted
          }
        >
          Vote
        </button>
      </div>
    </div>
  );
};

render() {
  if (!this.state.web3) {
    return (
      <div>
        {this.state.isAdmin ? <NavbarAdmin /> : <Navbar />}
        <center>Loading Web3, accounts, and contract...</center>
      </div>
    );
  }
}

```

```

}

return (
  <
    {this.state.isAdmin ? <NavbarAdmin /> : <Navbar />}
    <div>
      {!this.state.isElStarted && !this.state.isElEnded ? (
        <NotInit />
      ) : this.state.isElStarted && !this.state.isElEnded ? (
        <
          {this.state.currentVoter.isRegistered ? (
            this.state.currentVoter.isVerified ? (
              this.state.currentVoter.hasVoted ? (
                <div className="container-item success">
                  <div>
                    <strong>You've casted your vote.</strong>
                    <p />
                    <center>
                      <Link
                        to="/Results"
                        style={{
                          color: "black",
                          textDecoration: "underline",
                        }}
                      >
                        See Results
                      </Link>
                    </center>
                  </div>
                </div>
              ) : (
                <div className="container-item info">
                  <center>Go ahead and cast your vote.</center>
                </div>
              )
            ) : (
              <div className="container-item attention">
                <center>Please wait for admin to verify.</center>
              </div>
            )
          ) : (
            <
              <div className="container-item attention">
                <center>
                  <p>You're not registered. Please register first.</p>
                  <br />
                  <Link
                    to="/Registration"

```

```

        style={{ color: "black", textDecoration: "underline" }}
      >
        Registration Page
      </Link>
    </center>
  </div>
</>
)}
<div className="container-main">
  <h2>Candidates</h2>
  <small>Total candidates: {this.state.candidates.length}</small>
  {this.state.candidates.length < 1 ? (
    <div className="container-item attention">
      <center>Not one to vote for.</center>
    </div>
  ) : (
    <div
      {this.state.candidates.map(this.renderCandidates)}
      className="container-item"
      style={{ border: "1px solid black" }}
    >
      <center>That is all.</center>
    </div>
  )}
</div>
</>
): !this.state.isElStarted && this.state.isElEnded ? (
  <div className="container-item attention">
    <center>
      <h3>The Election ended.</h3>
      <br />
      <Link
        to="/Results"
        style={{ color: "black", textDecoration: "underline" }}
      >
        See results
      </Link>
    </center>
  </div>
</>
): null}
</div>
</>
);
}

```

SERVER-SIDE-CODING

Election.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.21 <0.9.0;

contract Election {
    address public admin;
    uint256 candidateCount;
    uint256 voterCount;
    bool start;
    bool end;

    constructor() public {
        // Initalizing default values
        admin = msg.sender;
        candidateCount = 0;
        voterCount = 0;
        start = false;
        end = false;
    }

    function getAdmin() public view returns (address) {
        // Returns account address used to deploy contract (i.e. admin)
        return admin;
    }

    modifier onlyAdmin() {
        // Modifier for only admin access
        require(msg.sender == admin);
        _;
    }

    // Modeling a candidate
    struct Candidate {
        uint256 candidateId;
        string header;
        string slogan;
        uint256 voteCount;
    }
    mapping(uint256 => Candidate) public candidateDetails;

    // Adding new candidates
    function addCandidate(string memory _header, string memory _slogan)
```

public

```
// Only admin can add onlyAdmin
{
    Candidate memory newCandidate =
        Candidate({
            candidateId: candidateCount,
            header: _header,
            slogan: _slogan,
            voteCount: 0
        });
    candidateDetails[candidateCount] = newCandidate;
    candidateCount += 1;
}
```

// Modeling a Election Details

```
struct ElectionDetails {
    string adminName;
    string adminEmail;
    string adminTitle;
    string electionTitle;
    string organizationTitle;
}
ElectionDetails electionDetails;
```

```
function setElectionDetails(
    string memory _adminName,
    string memory _adminEmail,
    string memory _adminTitle,
    string memory _electionTitle,
    string memory _organizationTitle
)
```

public

// Only admin can add
onlyAdmin

```
{
    electionDetails = ElectionDetails(
        _adminName,
        _adminEmail,
        _adminTitle,
        _electionTitle,
        _organizationTitle
    );
    start = true;
    end = false;
}
```



```

// Get Elections details
function getAdminName() public view returns (string memory) {
    return electionDetails.adminName;
}

function getAdminEmail() public view returns (string memory) {
    return electionDetails.adminEmail;
}

function getAdminTitle() public view returns (string memory) {
    return electionDetails.adminTitle;
}

function getElectionTitle() public view returns (string memory) {
    return electionDetails.electionTitle;
}

function getOrganizationTitle() public view returns (string memory) {
    return electionDetails.organizationTitle;
}

// Get candidates count
function getTotalCandidate() public view returns (uint256) {
    // Returns total number of candidates
    return candidateCount;
}

// Get voters count
function getTotalVoter() public view returns (uint256) {
    // Returns total number of voters
    return voterCount;
}

// Modeling a voter
struct Voter {
    address voterAddress;
    string name;
    string phone;
    bool isVerified;
    bool hasVoted;
    bool isRegistered;
}
address[] public voters; // Array of address to store address of voters
mapping(address => Voter) public voterDetails;

```

```

// Request to be added as voter
function registerAsVoter(string memory _name, string memory _phone) public {
    Voter memory newVoter =
        Voter({
            voterAddress: msg.sender,
            name: _name,
            phone: _phone,
            hasVoted: false,
            isVerified: false,
            isRegistered: true
        });
    voterDetails[msg.sender] = newVoter;
    voters.push(msg.sender);
    voterCount += 1;
}

// Verify voter
function verifyVoter(bool _verifiedStatus, address voterAddress)
    public
    // Only admin can verify
    onlyAdmin
{
    voterDetails[voterAddress].isVerified = _verifiedStatus;
}

// Vote
function vote(uint256 candidateId) public {
    require(voterDetails[msg.sender].hasVoted == false);
    require(voterDetails[msg.sender].isVerified == true);
    require(start == true);
    require(end == false);
    candidateDetails[candidateId].voteCount += 1;
    voterDetails[msg.sender].hasVoted = true;
}

// End election
function endElection() public onlyAdmin {
    end = true;
    start = false;
}

// Get election start and end values
function getStart() public view returns (bool) {
    return start;
}

```

```
function getEnd() public view returns (bool) {  
    return end;  
}  
}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT  
pragma solidity >=0.4.21 <0.8.0;  
  
contract Migrations {  
    address public owner;  
    uint256 public last_completed_migration;  
  
    modifier restricted() {  
        if (msg.sender == owner) _;  
    }  
  
    constructor() public {  
        owner = msg.sender;  
    }  
  
    function setCompleted(uint256 completed) public restricted {  
        last_completed_migration = completed;  
    }  
}
```

A.2 SAMPLE SCREENS

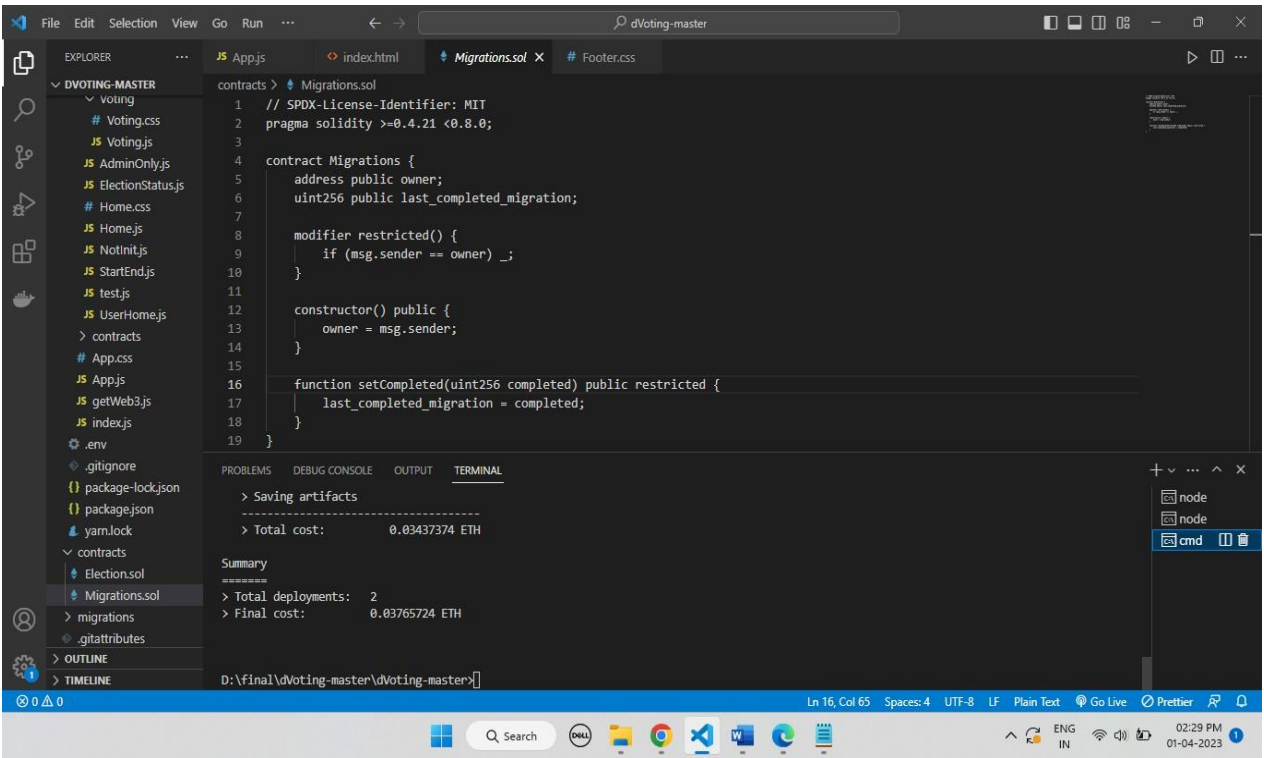


Fig A2.1 Executing the input

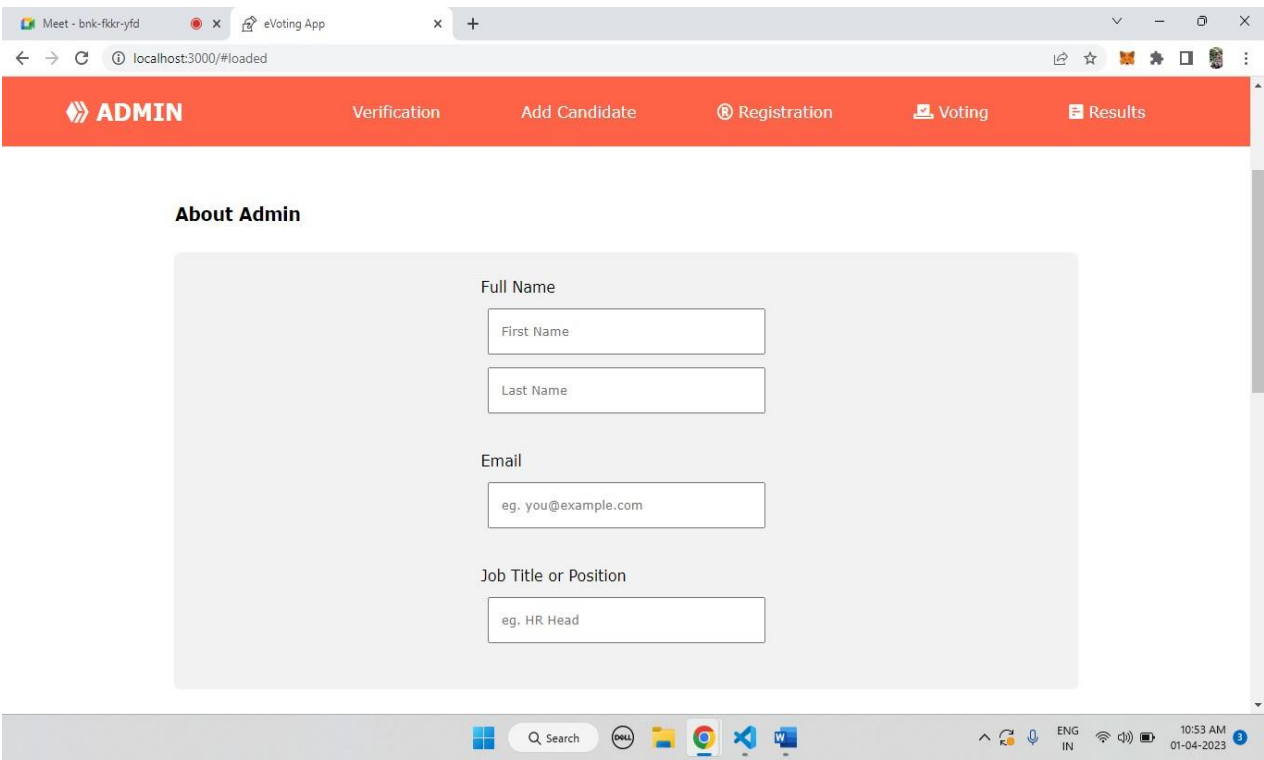


Fig A2.2 Admin Login

ADMIN Verification Add Candidate Registration Voting Results

Add a new candidate

Total candidates: 0

Header
eg. Marcus

Slogan
eg. It is what it is

Add

Candidates List

Fig A2.3 Candidate registration

ADMIN Verification Add Candidate Registration Voting Results

About Election

Election Title
eg. School Election

Organization Name
eg. Lifeline Academy

Do not forget to add candidates.
Go to [add candidates](#) page.

Fig A2.4 Initializing new ballot

HOME **Registration** **Voting** **Results**

Total registered voters: 5

Registration

Register to vote.

Account Address:

Name:

Phone number *:

Note:
Make sure your account address and Phone number are correct.
Admin might not approve your account if the provided Phone number nub does not matches the account address registered in admins catalogue.

Fig A2.5 Voter Registration

ADMIN **Verification** **Add Candidate** **Registration** **Voting** **Results**

Verification

Total Voters: 5

List of registered voters

Account address	0x0C290dcBe15297fac16d6651d90d96cd8cdB2446
Name	Ajith
Phone	8907654312
Voted	False
Verified	False
Registered	True

Fig A2.6 Approving the voters

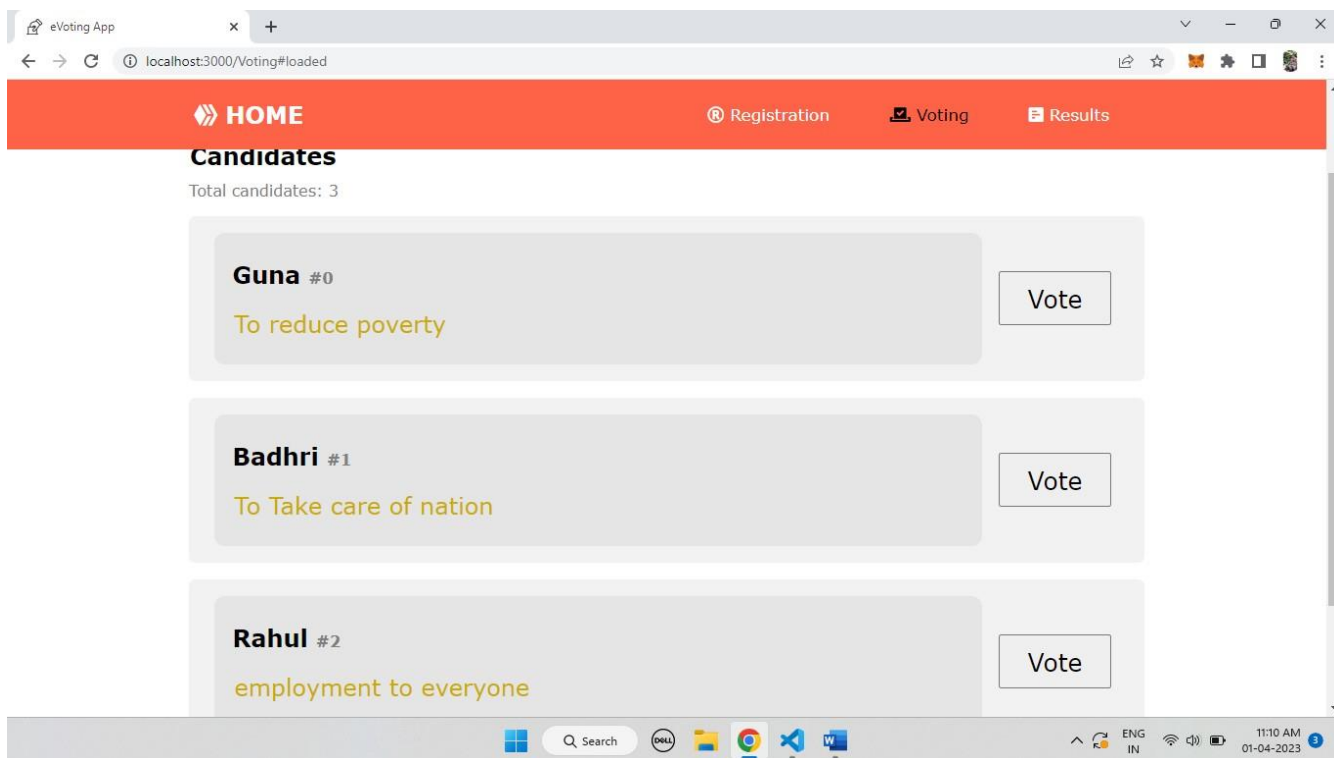


Fig A2.7 Voting process

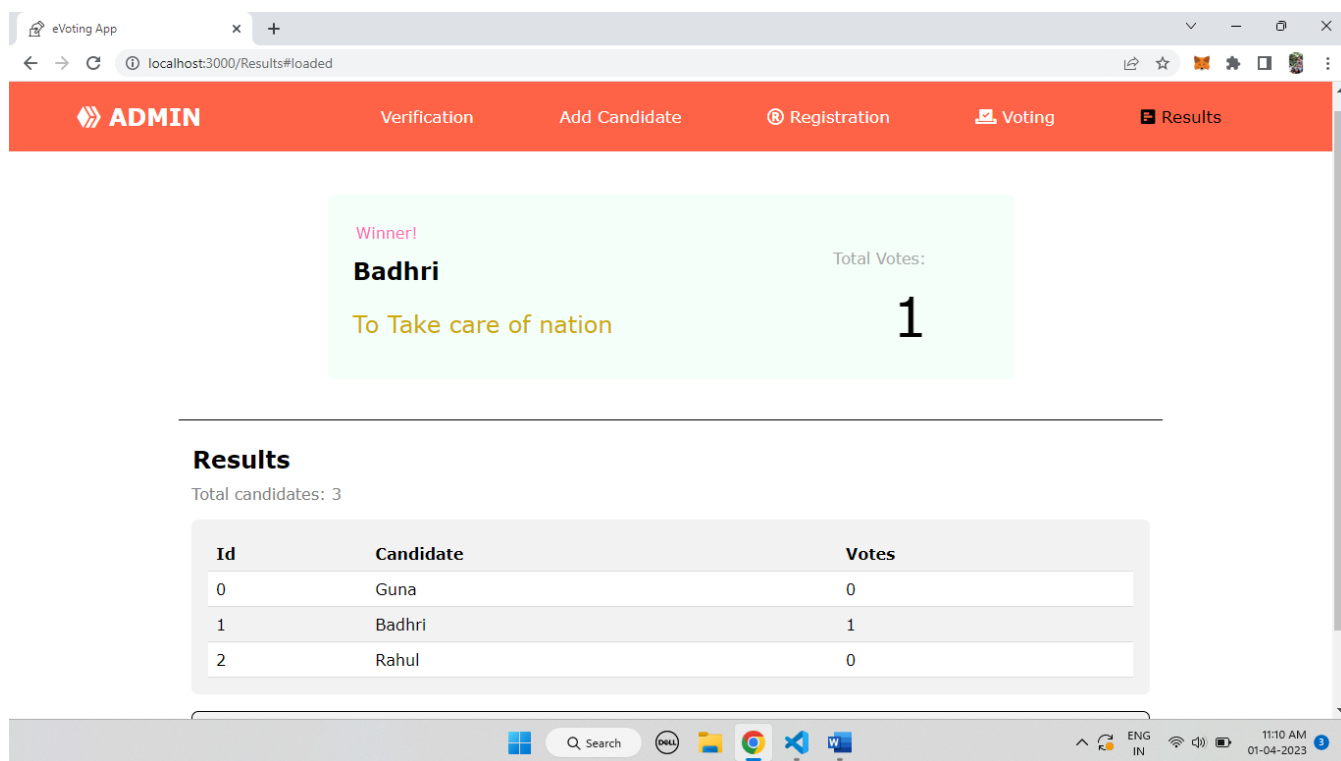


Fig A2.8 Resultant process

REFERENCES

REFERENCES

- [1] Nakamoto Satoshi, inventing bitcoin, implementing the first blockchain, deploying the first decentralized digital currency “A Peer-to-Peer Electronic Cash System” original 20 March 2014.
- [2] Azaria, Asaph, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. "MedRec: Using Blockchain for Medical Data Access and Permission Management." In Open and Big Data (OBD), International Conference on, pp. 25-30. IEEE, 2016.
- [3] Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, Computer Networks: The International Journal of Computer and Telecommunications Networking, December 2017.
- [4] Nicole J. Goodman; Jon H. Pammett , 2014. “Internet Voting in a Local Election in Canada”, in Internet and Democracy in Global Perspective, Studies in Public Choice 31, Eds. Bernard Grofman, Alex Trechsel, and Mark Franklin, Springer Verlag.
- [5] “Enigma Protocol Overview.” <https://enigma.co/protocol>. Accessed: 2020
- [6] V. Buterin et al., “A next-generation smart contract and decentralized application platform,” white paper, vol. 3, no. 37, 2014.
- [7] G. Zyskind, O. Nathan, et al., “Decentralizing privacy: Using blockchain to protect personal data,” in 2015 IEEE Security and Privacy Workshops, pp. 180–184, IEEE, 2015.

- [8] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: Decentralized computation platform with guaranteed privacy,” arXiv preprint arXiv:1506.03471, 2015.
- [9] Y. Zhou, Y. Liu, C. Jiang, and S. Wang, “An improved foo voting scheme using blockchain,” *International Journal of Information Security*, vol. 19, no. 3, pp. 303–310, 2020.
- [10] A. Fujioka, T. Okamoto, and K. Ohta, “A practical secret voting scheme for large scale elections,” in *International Workshop on the Theory and Application of Cryptographic Techniques*, pp. 244–251, Springer,
- [11] F. . Hj’almarsson, G. K. Hreiarsson, M. Hamdaqa, and G. Hj’almt’ysson, “Blockchain-based e-voting system,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 983–986, IEEE, 2018.
- [12] H.-D. Park, “A decentralized e-voting system based on blockchain network,” *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8, 2019.
- [13] M. Kumar, S. Chand, and C. P. Katti, “A secure end-to-end verifiable internet-voting system using identity-based blind signature,” *IEEE* 2041, 2020.