Tribhuvan University

Institute of Science and Technology

A Final Year Project Report

On

DeVote: A decentralized voting system using blockchain

Submitted To

Department of Information Technology

Kathford International College of Engineering and Management

In partial fulfillment of the requirement for the Bachelor Degree in Computer Science and Information Technology

Submitted By

Satish Bajagain (28491/078)

Pritam Das (28485/078)

Shreeya Palikhel (28494/078)

Under the Supervision of

Ms. Sarita Neupane

July, 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who supported and guided us throughout the development of our project "**DeVote – A Decentralized Voting System using Blockchain.**"

Firstly, we extend our heartfelt thanks to our respected supervisor and project coordinator, **Ms Sarita Neupane,** for her invaluable guidance, insightful feedback and continuous support. Her encouragement and mentorship have played a crucial role in shaping the direction and quality of our project.

We are also very thankful to our Head of Department (HOD), **Mr. Mukesh Prasad Chaudhary,** for providing us with the necessary academic environment, resources and encouragement to carry out this project successfully.

We would also like to thank our friends and classmates for their encouragement and constructive discussions during the development process.

Finally, we express our heartfelt thanks to our families for their moral support and motivation throughout the project journey.

ABSTRACT

The rapid advancement of technology calls for secure, transparent, and efficient voting systems to address the limitations of traditional and existing digital voting methods. This project, **DeVote-A Decentralized Voting System Using Custom Blockchain,** is a decentralized electronic voting system leveraging blockchain technology to address the inherent challenges of traditional voting methods, namely security, transparency, and immutability. The system facilitates a secure and anonymous voting process, allowing eligible voters to register and cast their ballots using their email and date of birth, thereby ensuring only authorized participation while safeguarding voter privacy. The technical architecture comprises a React with Vite frontend, designed for a responsive and intuitive user experience, and a Go-based blockchain backend utilizing the Practical Byzantine Fault Tolerance (PBFT) consensus mechanism. This robust combination ensures the transparency and tamper-resistance of vote recording and counting processes. Currently, the core blockchain functionalities, including block creation, transaction validation, and consensus, have been successfully implemented and rigorously tested. Front-end development is actively in progress, with key user interfaces and interactions being built, and systematic end-to-end testing is planned to validate the integrated system's performance and security. This project's primary objective is to deliver a practical, highly secure, and verifiable alternative to conventional voting systems, particularly focusing on its potential application in contexts like Nepal, where enhancing electoral integrity is crucial.

Keywords: Decentralized Voting, Blockchain, Electronic Voting, Immutable Ledger, Transparency, PBFT Consensus, Voter Anonymity, Digital Security.

Table of Contents

ACKNOWLEDGEMENT i
ABSTRACT ii
List of Abbreviations v
List of Figures vi
List of Tables vi
1. INTRODUCTION 1
1.1 Introduction 1
1.2 Problem Statement 1
1.3 Objective 2
1.4 Scope and Limitations 2
1.5 Development Methodology 3
1.6 Report Organizations 5
2. BACKGROUND STUDY AND LITERATURE REVIEW 7
2.1 Background Study 7
2.2 Literature Review 9
3. SYSTEM ANALYSIS 14
3.1 Requirement Analysis 14
3.2 FEASIBILITY ANALYSIS 17
3.3 ANALYSIS 20
4. SYSTEM DESIGN 23
4.1 Design 23
4.2 Algorithm Details 27
5. IMPLEMENTATION AND TESTING 32
5.1 Implementation 32
5.2 Testing 34
5.3 Result Analysis 43
6. CONCLUSION 44
6.1 Modules Competed 44
6.2 Modules in Progress 44
6.3 Modules Remaining 44
APPENDICES 45
Screenshots 45

Homepage 45
References 48

List of Abbreviations

- API: Application Programming Interface
- BFT: Byzantine Fault Tolerance
- CLI: Command Line Interface
- CRUD: Create, Read, Update, Delete
- DB: Database
- DOB: Date of Birth
- GUI: Graphical User Interface
- HMR: Hot Module Replacement
- IDE: Integrated Development Environment
- JSON: JavaScript Object Notation
- PBFT: Practical Byzantine Fault Tolerance
- P2P: Peer-to-Peer
- POW: Proof of Work
- POS: Proof of Stake
- REST: Representational State Transfer
- SHA: Secure Hash Algorithm
- UI: User Interface
- UX: User Experience
- zk-SNARKs: Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

List of Figures

Figure 3.1: Use case of DeVote 14
Figure 3.2: Gantt Chart of DeVote 19
Figure 3.3 Class Diagram of DeVote 20
Figure 3.4 Sequence Diagram of DeVote 21
Figure 3.5 Activity Diagram of DeVote 22
Figure 3.6 State Diagram of DeVote 22
Figure 4.1 System Architecture of DeVote 23
Figure 4.2 State Diagram of DeVote 24
Figure 4.3 Activity Diagram of DeVote 25

List of Tables

Table 1: Use Case for DeVote 15
Table 2: Test Case for Voter Registration 36
Table 3: Test Case for Voter Login 38
Table 4: Test Case for Double Voting Restriction 39
Table 5: Test Case for Tamper-Proof-Blockchain 40
Table 6: Test Case for Vote Counting 41
Table 7: Test Case for Admin Controls 42

1. INTRODUCTION

1.1 Introduction

In recent years, the need for secure, tamper-proof, and transparent voting system has grown significantly across the globe. Traditional paper-based or centralized electronic voting methods often face challenges due to their vulnerability to manipulation and lack of public trust. These challenges determine the democratic process and can lead to questions regarding the legitimacy of election results. To address these issues, this project introduces DeVote, an innovative electronic voting system built upon the blockchain technology. DeVote is designed to provide a secure vote-casting environment, prevent vote tampering, and offer unparalleled vote verifiability, which ensures that every single vote is accurately and immutably registered on the blockchain. Unlike general-purpose blockchains, DeVote uses a lightweight, custom-designed architecture optimized for elections. It combines zero-knowledge proofs (zk-SNARKs) to validate votes without revealing identities, ring signatures for anonymous authentication, and Merkle trees to guarantee vote integrity. A consensus mechanism, PBFT, protects against attacks while maintaining fault tolerance [1]. By using the distributed and cryptographic nature of blockchain, DeVote aims to provide transparency in electoral processes and make the election more reliable and accountable.

1.2 Problem Statement

Existing voting systems, particularly traditional paper-based have different vulnerabilities. Their centralized nature often leads to a distinct lack of transparency, making it difficult for the public to verify. Furthermore, they are susceptible to various forms of fraud, including double-voting, and vote manipulation. Also, these systems frequently provide insufficient means for individual voters to verify that their vote was correctly recorded and counted without revealing their identity. Therefore, there is an urgent need for a modern, technologically advanced voting system that can definitively guarantee vote integrity, preserve voter anonymity through strong cryptographic measures, and ensure that only eligible voters can cast a single, verifiable vote.

1.3 Objective

- To develop a secure, user-friendly blockchain voting system with vote immutability, voter anonymity (zk-SNARKs), verifiability (Merkle trees), and Voter ID-based authentication with decentralized validation (PBFT).

1.4 Scope and Limitations

This project aims to develop a decentralized electronic voting system using blockchain technology to ensure secure, tamper-proof, and verifiable elections.

1.4.1 Scope

The key areas covered within the project include:

1. Voter registration: The system provides a mechanism for new users to register using their email and a unique Voter ID. It checks a pre-defined internal voter list to verify the eligibility of the registering individual, preventing unauthorized participation.
2. Secure User Login: Registered users can securely log in to the system using their verified credentials (email and password), granting them access to their voting interface.
3. Blockchain-Based Vote Storage: All votes are securely and immutably recorded on the blockchain. This distributed ledger ensures the transparency of the voting process.
4. Anonymity and Privacy through Cryptography: While votes are recorded on the blockchain, the system uses cryptographic measures, which include hashing and the principle of not linking voter identities directly with their votes which ensures voter privacy and anonymity.
5. "One Voter, One Vote" Enforcement: The system is designed to prevent duplicate votes, ensuring that each registered and eligible user can cast only a single vote per election.
6. Admin Dashboard for Candidate Management: A secure admin dashboard is provided, providing authorized administrators to efficiently manage candidates, including adding new candidates, updating existing candidate information, and removing candidates.
7. Result Generation and Verification: Upon the conclusion of an election, the system automatically generates final results by tallying votes directly from the blockchain.

Furthermore, voters are provided with a mechanism (e.g., using a Merkle path) to cryptographically verify that their vote was indeed included and counted correctly.

1.4.2 Limitations:

1. No Integration with Government Voter Databases: For the scope of this project, the system uses an internal voter list for eligibility verification. It explicitly does not connect or integrate with real-time national or governmental databases for Voter ID validation. This is a practical limitation for a project of this scale, given the complexities and access restrictions associated with such external systems.
2. Limited Scalability: The current system is made and implemented for small to medium-scale elections, such as those conducted within academic institutions, private organizations, or local community elections. While the underlying blockchain technology is scalable, the current design and resource allocation are not optimised for a big election, and further optimisation would be required for national-level deployment.

1.5 Development Methodology

The development of the **"DeVote"** system is being carried out by implementing the **Agile Incremental Development** methodology, which breaks down the entire project into multiple phases and allows each module to be developed, tested and improved progressively. This approach was selected due to its suitability for the project involving emerging technologies, evolving requirements and the need for early validation of critical modules such as vote casting and blockchain integration.

Below is a breakdown of how each stage of Agile Incremental process was applied to DeVote:

- Planning Phase: During the initial planning phase, multiple meetings were held to define project goals, target users and key features. Key problems in traditional voting systems, such as security vulnerabilities, lack of transparency, and user privacy concerns, were outlined. It was decided that blockchain would serve as the core technology due to its immutability and distributed nature. React JS and Go (Golang) were chosen as the development languages. Voter authentication was designed to be based on Voter ID, DOB and location, with the system generating

secure credentials for login. The project scope, constraints and development milestones were defined.

- Requirement Analysis Phase: In this phase, functional and non-functional requirements were collected and analysed. The core functional requirements included modules for voter registration, login authentication, vote casting, candidate management, vote verification and result tallying. Non-functional requirements such as system responsiveness, data privacy, security, performance, anonymity and system integrity. Use case diagrams and activity diagrams were prepared to represent the interaction flows and overall system behavior.
- System Design Phase: Following requirement finalization, object-oriented design principles were applied to structure the system architecture. Class diagrams were developed to represent core system entities, such as voters, elections and blockchain components. Sequence diagram outlined the workflows for registration, login and voting processes. Activity diagram captured the dynamic behavior of major operations. In addition, the design of blockchain, vote encryption flow and block validation mechanisms were conceptualized at this stage.
- Increment 1 - Basic User Management and Authentication: The first increment focused on implementing user registration and authentication modules. A user registration module was implemented to allow voters to register using their Vote ID, DOB and location, which were cross-verified against a predefined internal database. A login system was developed that generated system-assigned credentials (username and password) upon successful registration. The basic admin login and dashboard layout were also designed in this stage.
- Increment 2 - Blockchain Structure and Vote Storage: The second increment included the development of a custom blockchain using Go. A block structure was created to store encrypted vote data along with timestamps and cryptographic hashes. Logic for generating hash values using **SHA-256**, linking blocks and ensuring chain integrity was implemented. This increment ensured that each cast vote would be securely recorded in an immutable chain of blocks.
- Increment 4 - Vote Casting: The third increment addressed the core voting mechanism. Functionality for casting encrypted votes, preventing duplicate submissions and enforcing the **“one voter, one vote”** principle was implemented.

1.6 Report Organizations

The project report is divided into 6 chapters, which are further divided into different headings.

Chapter 1
In the Introduction section, an introduction to the project is given. It also includes objectives, problem statements, scope and limitations, and methodology.

Chapter 2
In the background study and literature review section, general concepts and terms necessary for understanding the system are reviewed, and previous research in this field is discussed, which serves as the starting point of this project.

Chapter 3
In the System Analysis section, the architecture and requirements of the system are discussed by breaking down the system into components, analyzing their interactions, and identifying important modules. This uses a Use case diagram, Class and Object Diagram, Sequence Diagram, and Activity Diagram. A feasibility study is presented for understanding the future requirements and external factors that affect the project.

Chapter 4
In the System Design section, the system is described in detail. The system architecture demonstrates the entire system structure, and the flow charts illustrate important processes in the system.

Chapter 5
In the Implementation and Testing section, a description of the procedure, methods, and algorithm is included alongside the explanation of functionalities of the system as different modules. including the tools used to build the system and the tools that support the development. It comprises unit and system testing used to validate the system's functioning.

Chapter 6

In Conclusion, the progress of the project is described, the current ongoing development process and the remaining work on the project.

2. BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

This section provides a foundational understanding of the core technologies and concepts concerning the DeVote project, a blockchain-based electronic voting system designed to offer secure, transparent, and verifiable elections. It aims to familiarize the reader with the basic principles of:

- Blockchain Technology: Blockchain is a type of Distributed Ledger Technology (DLT) where data is stored in a decentralized and tamper-evident manner. It consists of blocks that hold transaction data and are linked together chronologically using cryptographic hashes.

  Key Concepts:

  1. Blocks & Chains: Each block contains a list of transactions, a timestamp, and a cryptographic hash of the previous block, forming a secure chain.
  2. Cryptography:
     - Hashing: Ensures data integrity by converting input into a fixed-size hash (e.g., SHA-256).
     - Digital Signatures: Authenticate sender identity and ensure non-repudiation.
  3. Immutability: Once data is recorded in a block and added to the chain, it becomes nearly impossible to alter without network consensus.
  4. Decentralization: Eliminates single points of failure and ensures that no single authority controls the system.
  5. Relevance to Voting: These characteristics make blockchain ideal for secure voting, as they prevent vote tampering, promote transparency, and foster public trust in the system.
- Consensus Mechanisms (Practical Byzantine Fault Tolerance (PBFT)): Consensus algorithms are protocols that ensure all nodes in a distributed network agree on the current state of the blockchain.

  Practical Byzantine Fault Tolerance (PBFT) is particularly suited for permissioned blockchains, where nodes are known and controlled (e.g., government-run election nodes).

PBFT Overview:

1. Handles up to (n−1)/3 malicious or faulty nodes among n total.
2. Works through three phases: Pre-prepare → Prepare → Commit.
3. Nodes must reach a majority agreement before accepting a transaction.

Advantages:

1. Finality: Once a block is added, it is final—no forks.
2. Fault Tolerance: Resilient to malicious behavior from a minority of nodes.
3. PBFT ensures that even if some election authorities are compromised, the system remains secure and consistent.

- Electronic Voting Systems (E-voting): E-voting systems enable voters to cast ballots electronically, either online or through electronic voting machines.

  Types:

  1. Direct-Recording Electronic (DRE) Machines
  2. Internet Voting (i-Voting)
  3. Hybrid Systems with both paper and digital backups

  Advantages:

  1. Speed in vote tallying
  2. Convenience for remote voters
  3. Reduction in human error

  Challenges:

  1. Vulnerabilities to cyberattacks

- Cryptographic Primitives: Cryptographic primitives are the building blocks of secure systems.

  Hash Functions (e.g., SHA-256):

  - Input of any length → fixed-length output.
  - One-way and collision-resistant.
  - Used to fingerprint blocks and verify data integrity.

  Public-Key Cryptography:

  - Each user has a public and private key pair.
  - Ensures only eligible voters sign their vote.
  - Enables Digital Signatures: A voter's private key signs the vote, which can be verified with their public key.

Use in DeVote:

- Ensures vote authenticity
- Prevents double voting
- Validates voter identity securely

- Merkle Trees: Merkle Trees, or hash trees, are binary trees where:
  i. Each leaf node is a hash of a data block (e.g., individual vote).
  ii. Each non-leaf node is a hash of its two child nodes.

  Advantages:

  1. Efficient verification of individual data elements without exposing the whole dataset.
  2. Proof of inclusion: A user can verify their vote is part of the block with a small "Merkle proof" path.

  In Voting:

  1. Enables End-to-End Verifiability: Voters can check their vote was included without compromising privacy.
  2. Reduces storage and verification complexity.
- ReactJS and Go Programming Language: ReactJS is a JavaScript library for building interactive UIs. It uses a component-based architecture and virtual DOM for high performance. Go is a statically typed, compiled language known for simplicity, concurrency support, and performance. It is excellent for building scalable, concurrent server-side applications and blockchains.

  Use of React JS in Devote:

  i. Frontend for voters and administrators.
  ii. Real-time updates, state management, and responsive UI.

  Use of Go in Devote:

  i. A statically typed, compiled language known for simplicity, concurrency support, and performance.
  ii. Excellent for building scalable, concurrent server-side applications and blockchains.

2.2 Literature Review

This comprehensive review analyzes existing blockchain voting systems, from academic prototypes to deployed solutions, providing critical insights for decentralized voting system development. The analysis covers architectural approaches, consensus mechanisms, privacy solutions, and implementation experiences across public, private, and hybrid blockchain designs.

Follow My Vote (c.2016)

Follow My Vote represents an early blockchain voting initiative built on the BitShares blockchain-a delegated proof-of-stake platform forked from Bitcoin-with a "voting DAC" (Decentralized Autonomous Corporation) design.

The system uses blind signatures for ballot casting and pseudonymous keypairs for identity verification. Voters generate public/private key pairs, submit blinded tokens to a registrar for signing, then later unblind and cast vote tokens on-chain. A distinctive feature is the "forgiveness" mechanism, allowing vote changes before a deadline while maintaining one-person-one-vote integrity through the blockchain ledger.

Despite offering full decentralization with end-to-end verifiability and open-source transparency, Follow My Vote remained largely theoretical. Its main limitations include requiring a trusted registrar to verify eligibility (introducing potential centralization) and complexity in blind-signature shuffling and key management. The organization advocates for a fully online blockchain-based voting system where voters can use personal devices including computers and mobile phone [2].

BitCongress (c.2014)

BitCongress utilized multiple public blockchains-Bitcoin and Counterparty-for voting operations. The system issued "Vote Coins" as mined vote tokens, verified by miners in cryptocurrency 2.0 layers. Each vote was essentially a Bitcoin transaction with a fee incentive for miners.

The architecture combined several cryptographic technologies including "Bitcoin, BitMessage, BitTorrent, Proof Of Existence, Reddit & Ethereum". Within this framework, the Ethereum network tallied and verified smart contracts labeled as VoteCoins (YES, NO, & NEUTRAL votes) in a blockchain register, maintaining a public record of all votes.

The primary challenge with BitCongress was its minimal privacy protection-all votes were public ledger transactions-and the system effectively functioned as an economic democracy

where vote-buying was technically possible, rather than ensuring one-person-one-vote principles [3].

Chirotonia E-Voting Framework

Chirotonia represents a more recent academic implementation, currently serving as the official e-voting platform at the University of Napoli Federico II in Italy. The system combines blockchain with linkable ring signatures based on Elliptic Curve Cryptography (ECC) to produce compact cryptographic keys and digital signatures.

The voting protocol divides the process into six distinct phases: organization, registration, opening, voting, closing, and counting. Each phase involves different actors: voters, organizers, identity managers, and confidentiality managers.

A notable security improvement in Chirotonia is its recent integration of distributed key generation to address the previous single point of failure. Before this enhancement, the committee generated an RSA key pair, publishing the public key for vote encryption and securely storing the private key for later decryption. The improved design incorporates ETHDKG (Ethereum Distributed Key Generation) protocol, distributing responsibility among multiple entities to enhance security and fault tolerance.

Open Vote Network with zk-SNARKs

The Open Vote Network demonstrates advanced privacy protection through zero-knowledge proofs. Implemented on Ethereum, this system uses zk-SNARKs to verify off-chain computations without revealing sensitive information.

The architecture includes three primary zk-SNARK arithmetic circuits: publicKeyGen, encryptedVoteGen, and Tallying-corresponding to key generation, vote encryption, and result tallying. The design performs computations over a cyclic group with a finite field on the Baby Jubjub elliptic curve, optimized for performance.

Registration requires voters to submit public keys with zero-knowledge proofs, allowing verification without exposing private information. The protocol then enables encrypted vote casting with mathematical guarantees of correctness.

Voatz Mobile Voting Platform

Voatz represents one of the few blockchain voting systems deployed in official U.S. elections. The application connects to a permissioned Hyperledger Fabric network running

approximately 32 validator nodes distributed across AWS and Azure datacenters.The system architecture emphasizes identity verification through a multi-layered approach:

1. Voters download the mobile app and register with government-issued ID
2. Face/fingerprint biometric authentication binds the voter to their device
3. AES256-encrypted votes are sent to the Fabric network
4. Each block contains anonymized vote records, ensuring immutability
5. Voters receive visual receipts for personal verification

While Voatz demonstrates strong identity controls and employs geographically distributed nodes for resilience, critics note its highly centralized governance-states ultimately control the blockchain, and the ledger lacks public auditability. Security researchers have also questioned potential vulnerabilities in the mobile app and device trust chain.

Luxoft e-Vote (Zug, Switzerland 2018)

The Luxoft e-Vote implementation for the City of Zug represents a notable government-sanctioned blockchain voting experiment. Built on Hyperledger Fabric, the system is uniquely integrated with Zug's existing Ethereum-based digital identity system (uPort) for voter authentication.

Architecturally, the system distributed Fabric nodes across three data centers (two in Switzerland, one in Ireland) for redundancy. Votes were anonymized through encryption techniques, likely homomorphic or mix-net style, allowing tallying without linking ballots to voters.

Though limited in scale (approximately 240 registered voters with 72 participants) and non-binding, the pilot demonstrated feasibility for local democratic processes and showed high user satisfaction in post-vote surveys.

Agora (Swiss, c.2018)

Agora employs a sophisticated multi-layer blockchain architecture:

1. A permissioned "Cothority" network collects votes
2. Data is anchored to Bitcoin ("Cotena" ledger) for immutability
3. A public "Bulletin Board" ledger enables end-to-end verification
4. A "Valeda" network of citizen auditors running IPFS validates results

The system uses threshold ElGamal encryption for ballots and Neff shuffling for anonymity. This design enables voters to encrypt ballots under a threshold key, which are

then anonymously shuffled, posted to the blockchain bulletin board, and finally decrypted for tallying.

Agora was used in March 2018 as a third-party observer for Sierra Leone's presidential election, recording 280 polling locations' paper ballots on-chain as an independent audit mechanism.

After reviewing literature, several blockchain voting systems like Follow My Vote, Voatz, and Agora, **DeVote** is a practical and secure solution made according to Nepal's context. Unlike some systems that rely too much on central authority or lack strong privacy, DeVote uses a combination of technologies like PBFT for faster and more reliable consensus, Merkle Trees for data integrity, and plans to use zk-SNARKs for privacy in the future. Voter verification is done through preloaded VoterIDs stored securely in a database, removing the need for constant third-party checks. DeVote aims to make blockchain voting more accessible, trustworthy, and suitable for real-world elections in developing countries like Nepal.

3. SYSTEM ANALYSIS

3.1 Requirement Analysis

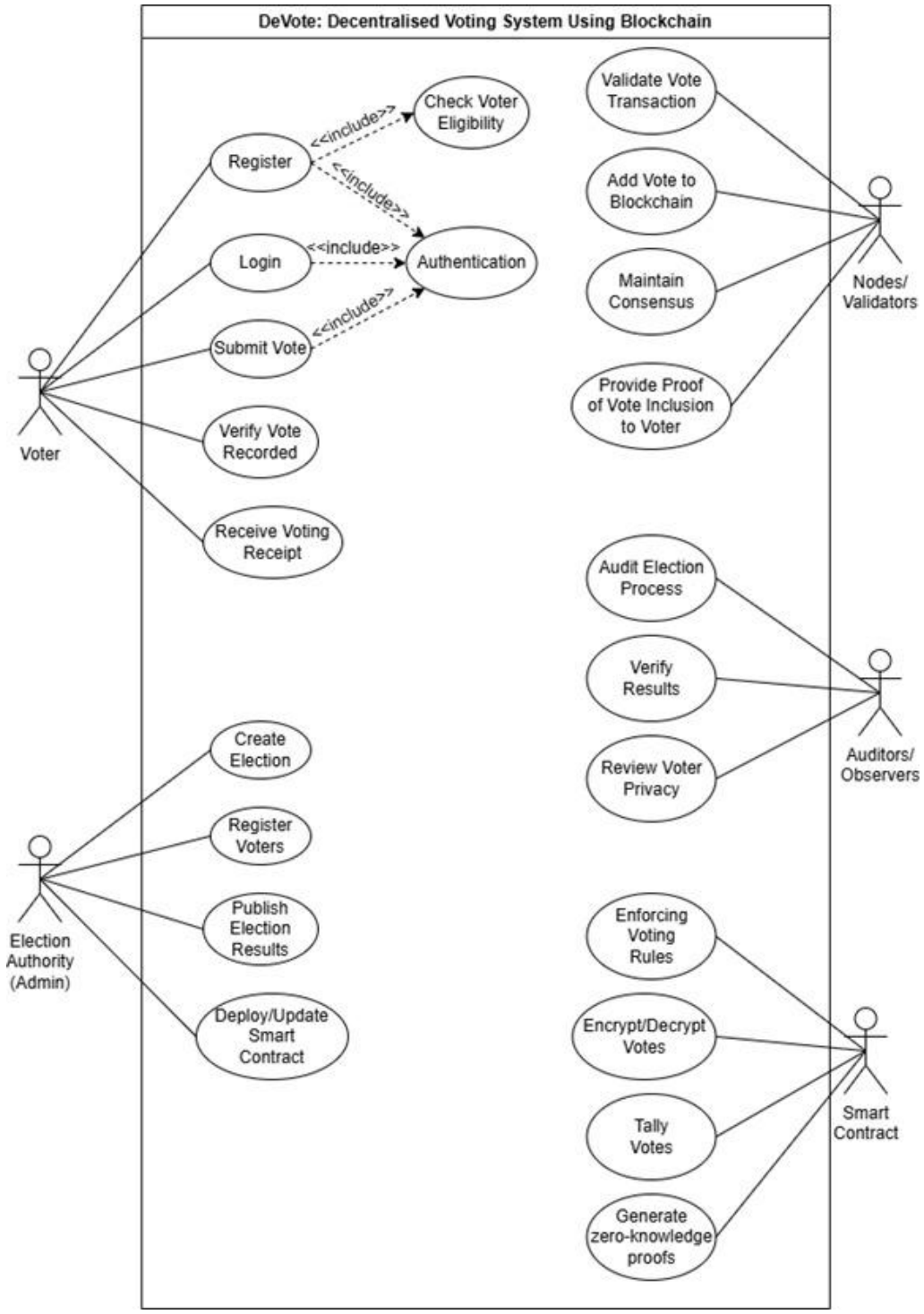


Figure 3.1: Use case of DeVote

3.1.1 Functional Requirements

Table 1: Use Case for DeVote

Use Case	Description
Register	Allows a voter to create an account by entering their email and voter id.
Login	Enables the voter to log in to the system using a verified username and password.
Submit Vote	Allows a verified voter to cast their vote for a selected candidate.
Verify Vote recorded	Allows the voter to check the voter whether their vote was successfully recorded in the system.
Create Election	Allows the admin to initiate a new election process.
Registered voters	The admin can upload or register eligible voters to the system's internal voter list.
Publish Election result	Enables the admin to release the final vote tally to the public.
Check voter eligibility	Checks if a registered user ID exists in the system voter list.
Authentication	Verifies voter identities during registration and login.
Validate vote transaction	Validators ensure the vote transaction is valid and follows the system rules
Add vote to blockchain	Records the verified vote into the blockchain securely
Maintain consensus	Ensures all the nodes/validators agree on the blockchain state.
Proof of vote inclusion	Sends a cryptographic proof (Merkle path) to ensure the voter that their vote is included.
Audit election process	Auditors/ viewers review the fairness and integrity of the voting process.
Verify results	Auditors validate the correctness of the final election result
Review voter privacy	Ensure that voter identity is protected and not traceable.
Encrypt/Decrypt votes	Votes are encrypted before being stored and decrypted for counting security.
Tally votes	After voting ends, votes are counted to determine the final results.
Generate z-k proofs	Used to prove vote validity and inclusion without revealing the voter's identity.

3.1.2 Non-functional Requirements

- Security:
    - Data Confidentiality: All sensitive voter data (e.g., email, voter ID during registration, before hashing) must be protected against unauthorized access.
    - Data Integrity: Votes and blockchain data must be protected from unauthorized modification or corruption. Hashing and digital signatures will be employed.
    - Authentication & Authorization: Users must be authenticated securely using system-generated credentials. Access privileges must be strictly enforced based on roles (e.g. voter or admin).
    - Tamper-Resistance: The blockchain must be inherently tamper-resistant,Once a vote is stored, it must not be modifiable or deletable under any condition.
    - Voter Anonymity: The system must ensure that a voter's identity cannot be linked to their specific vote after it has been cast and recorded on the blockchain. This will be achieved through cryptographic techniques.
    - Resistance to DDoS Attacks: The system should be designed to withstand denial-of-service attempts to maintain service availability during peak election periods.
- Performance:
    - Response Time: The system should provide timely responses to user interactions (e.g., login, vote submission) within acceptable limits (e.g., vote submission < 5 seconds, result generation < 1 minute for small-scale elections).
    - Throughput: The system should be able to process a reasonable number of vote transactions per second, suitable for small to mid-scale elections.
    - Scalability: While initially limited, the architecture should allow for future expansion to accommodate a larger number of voters and concurrent transactions if required.
- Reliability:
    - Availability: The system must remain highly available throughout the election period. The distributed nature of blockchain contributes to continuous system operation.

    - Fault Tolerance: The system must tolerate partial node failures. In future iterations, implementation of the PBFT consensus algorithm will further enhance this capability.
    - Data Persistence: All voter registration data, vote records, and candidate information must be stored persistently and must be recoverable in case of interruptions or failures.
- Usability:
    - User-Friendliness: The frontend interface for voters and administrators must be intuitive and easy to navigate.
- Portability:
    - The system must be deployable across multiple platforms and operating environments, such as Linux and Windows-based servers. Backend services and blockchain nodes should remain functionally consistent regardless of the deployment environment.

3.2 FEASIBILITY ANALYSIS

3.2.1 Technical Feasibility

The DeVote system is technically feasible based on the current tools, technologies, and development approach being used. The key reasons are listed below:

- A custom blockchain is being implemented using **Go (Golang)**, a highly efficient and scalable backend language suitable for concurrent systems.
- Cryptographic security tools such as **SHA-256** and **Merkle Trees** have been successfully integrated, ensuring data integrity and verifiability.
- Planned features like **zk-SNARKs** and **PBFT consensus** are supported by available open-source frameworks and are compatible with the existing architecture.
- The modular design supports easy debugging, updating, and integration of new features.
- The system is platform-independent and deployable on both **Linux** and **Windows** environments.

3.2.2 Operational Feasibility

The operational feasibility of DeVote is high, and can be used effectively within the intended environment such as academic institutions or small-scale community elections. The main reasons include:

- Voter registration uses **Voter ID, DOB, and Location**, verified against a preloaded internal database, eliminating dependency on national voter systems.
- A simple and responsive **web-based interface** has been designed for easy access across desktops and smartphones.
- Features are role-specific, providing separate access controls for voters and administrators, enhancing operational clarity.
- Vote verification using **Merkle proofs** supports transparency without compromising anonymity.
- All processes are conducted internally, avoiding the need for third-party services or external validation.

3.2.3 Economic Feasibility

The system is economically feasible, especially for organizations with limited budgets or institutions seeking to avoid the high cost of physical elections due to following reasons:

- The use of a **custom blockchain** avoids expensive transaction fees (e.g., Ethereum gas fees), reducing operational costs.
- Open-source tools and libraries are being used, minimizing licensing and software acquisition costs.
- Physical election expenses—such as ballot printing, polling stations, and staffing—are eliminated.
- Once deployed, the system requires minimal recurring maintenance and hosting costs.
- The scalable architecture allows future expansion with limited financial input.

3.2.4 Schedule Feasibility

The project is progressing according to the planned development phases and hence, is feasible as:

- An **Agile Incremental Model** has been followed to allow iterative development, regular testing, and continuous improvement
- As of the mid-defense stage, major increments such as voter registration, login, vote casting, blockchain storage, and Merkle Tree-based verification have been completed.
- Remaining increments—zk-SNARKs and PBFT—are well-defined and scheduled for final development.
- A detailed **Gantt chart** has been prepared to track the timeline, and progress is being regularly monitored through sprint reviews.

Project Schedule for DeVote (May-August 2025)

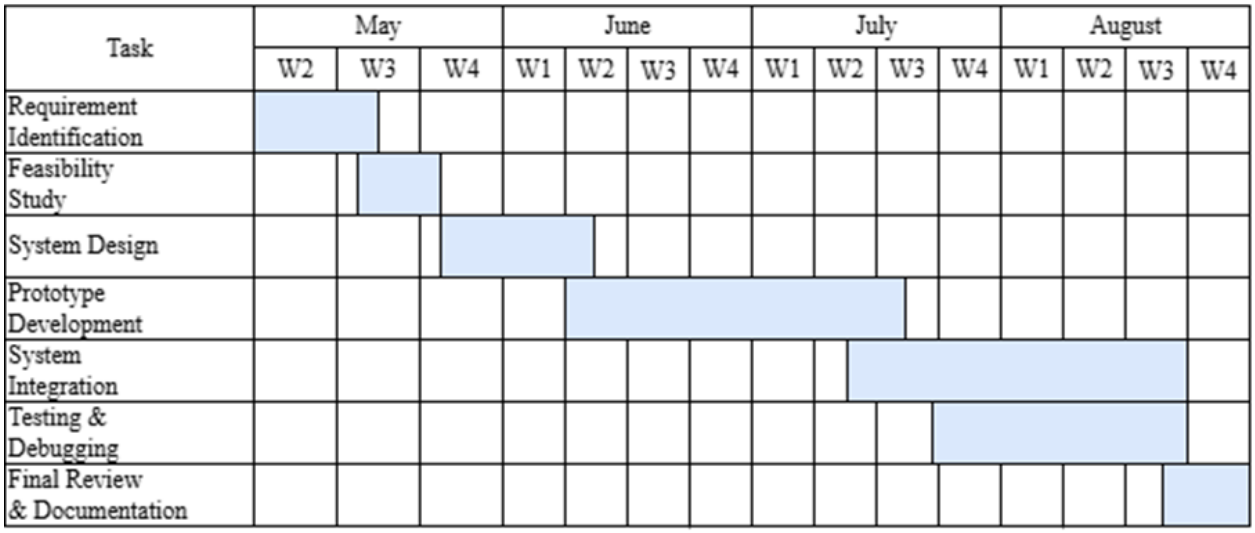


Figure 3.2: Gantt Chart of DeVote

3.3 ANALYSIS

3.3.1 Class Diagram

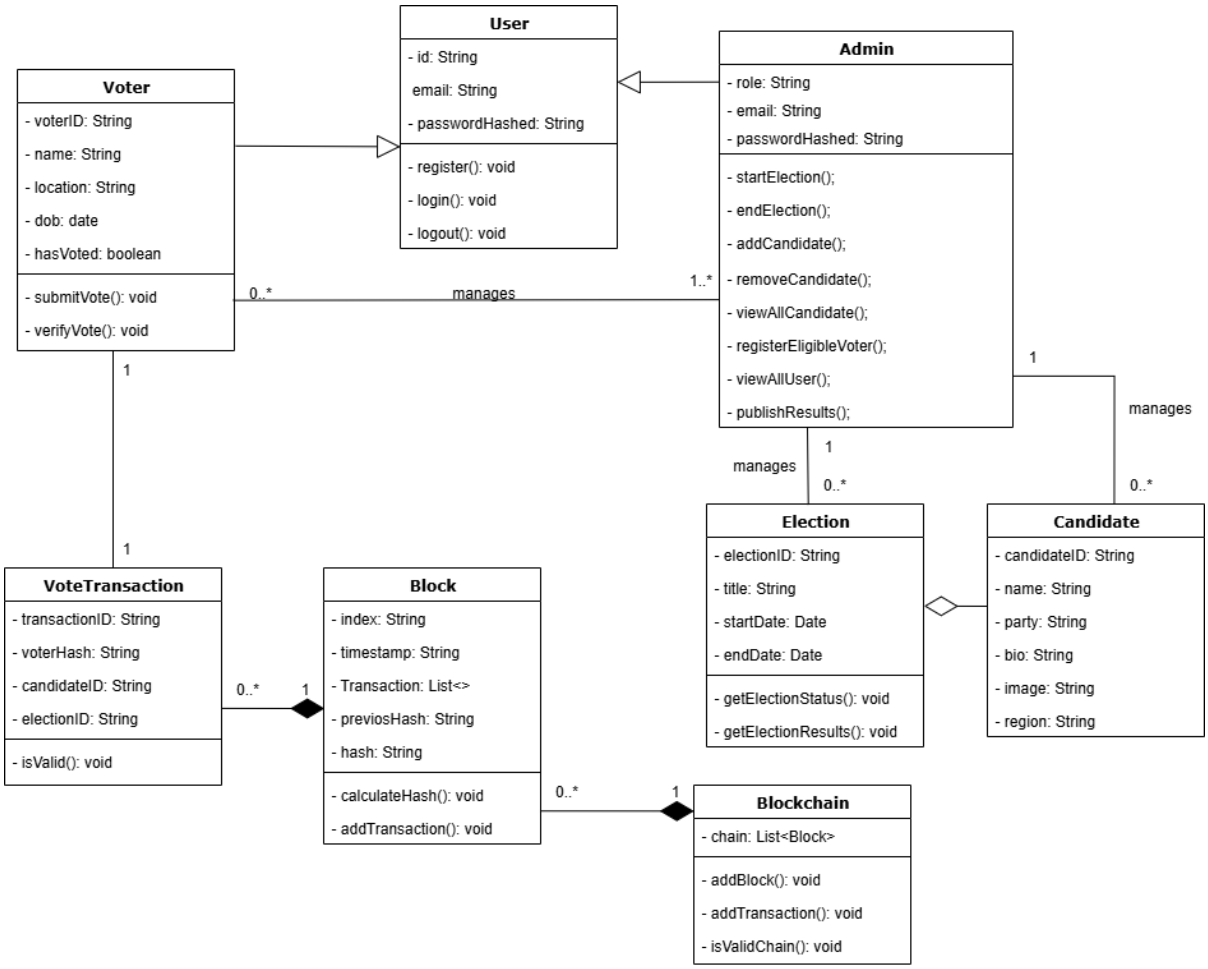


Figure 3.3 Class Diagram of DeVote

3.3.2 Sequence Diagram

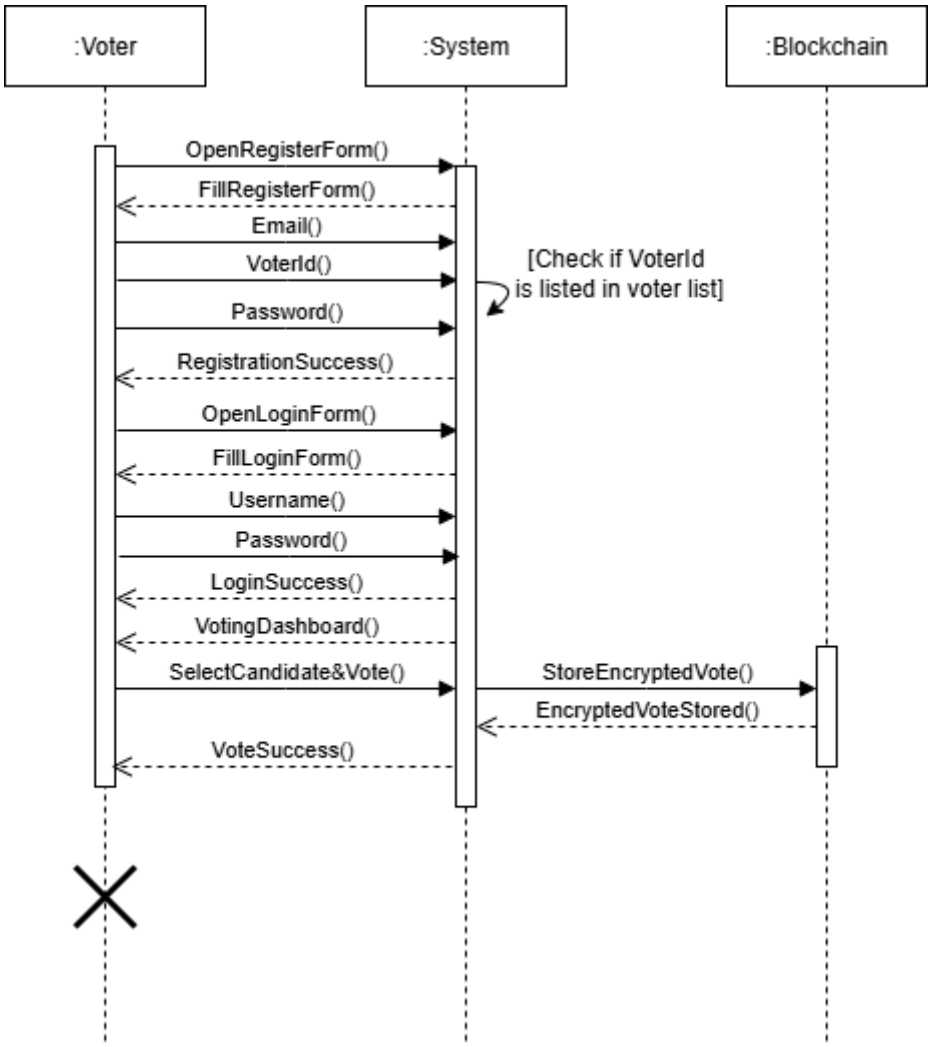


Figure 3.4 Sequence Diagram of DeVote

3.3.3 Activity Diagram

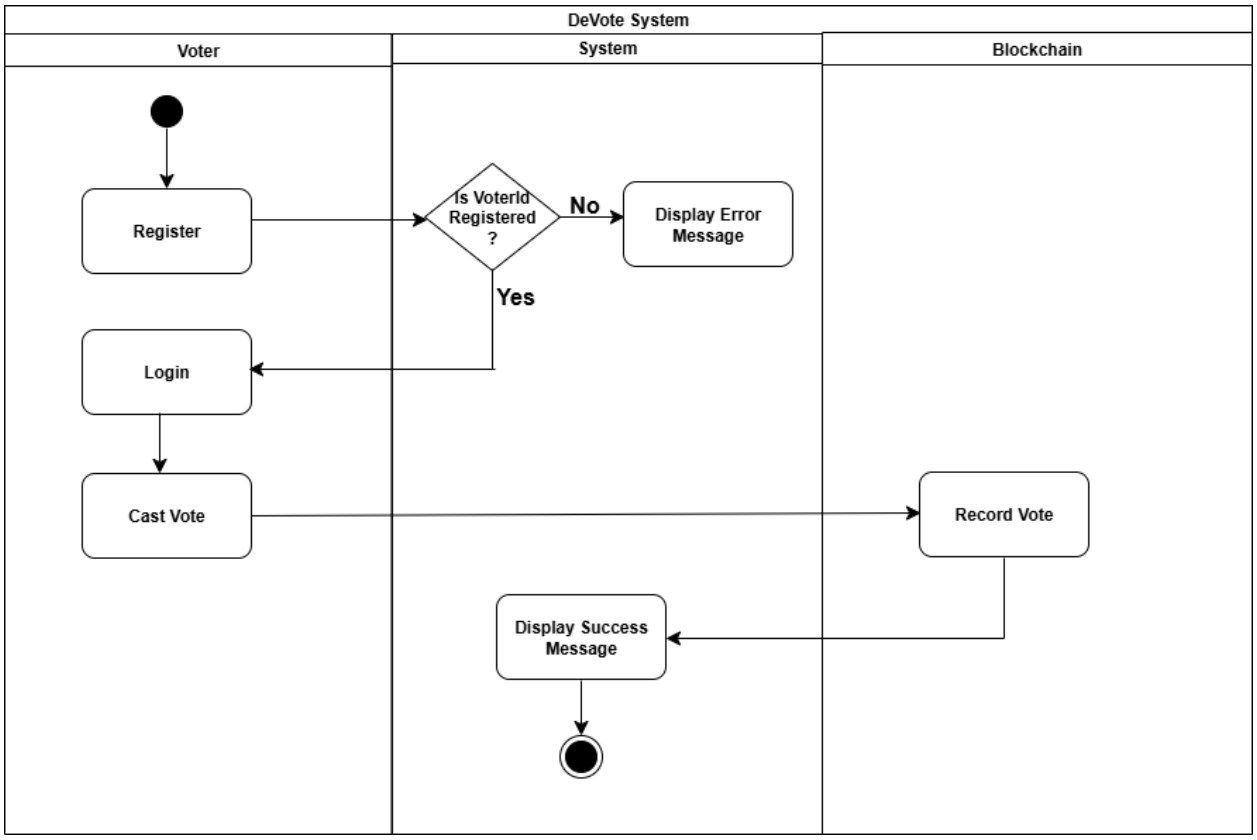


Figure 3.5 Activity Diagram of DeVote

3.3.4 State Diagram

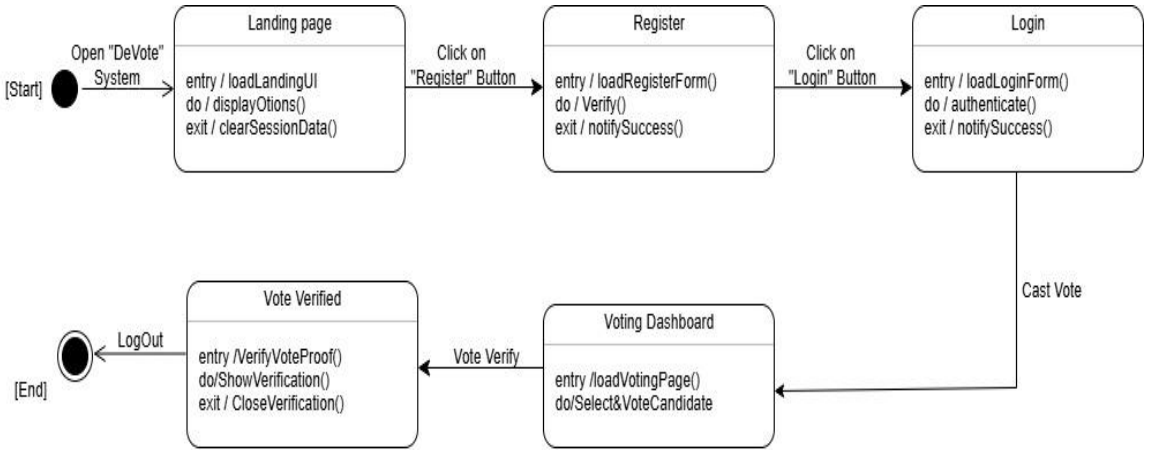


Figure 3.6 State Diagram of DeVote

4. SYSTEM DESIGN

4.1 Design

4.1.1 System Architecture

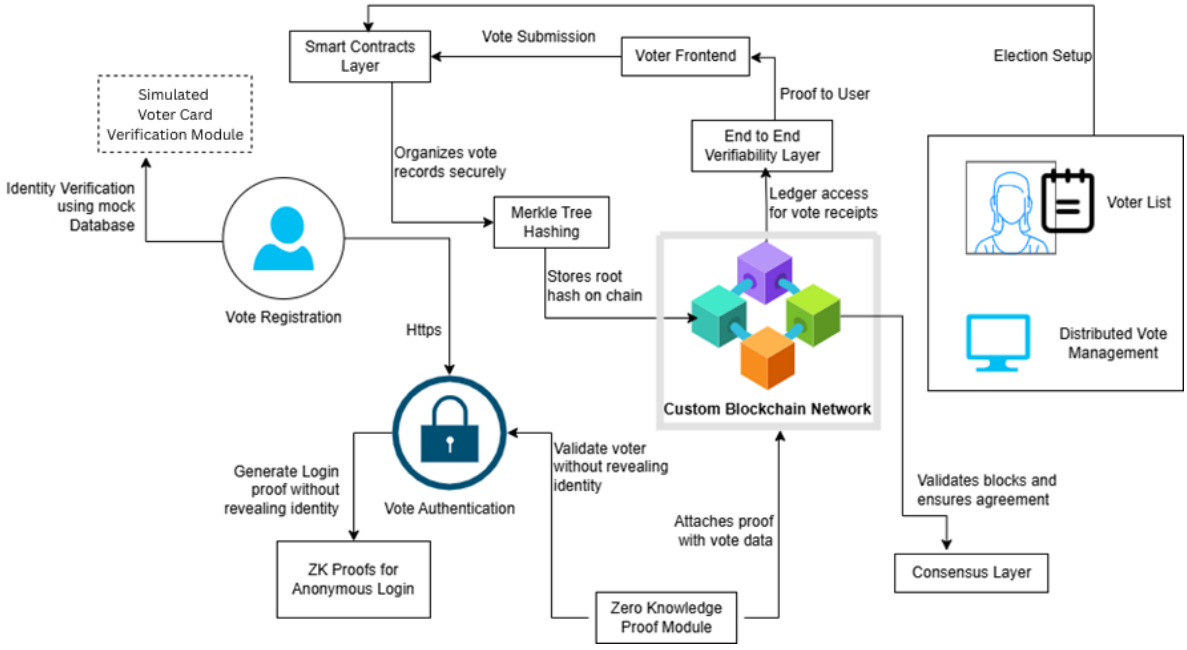


Figure 4.1 System Architecture of DeVote

The DeVote system architecture is based on a decentralized, client-server model that uses blockchain technology.The frontend of the DeVote is built using React, a popular JavaScript library, and developed using Vite, which helps the app run faster during development and deployment. This provides a clean and user-friendly interface to both voters and admin. Voters can use it to sign up, log in, browse the list of candidates, cast their vote, and even verify that their vote was recorded correctly. On the other hand, administrators can log in to manage elections, add or update candidate information, and publish the final results. The frontend communicates with the backend server by making REST API calls.

The backend is built using the Go programming language (Golang), known for its speed and efficiency. The backend handles a lot of important tasks. It manages users by processing voter registration, handling secure logins, and making sure only authorized people can access specific features. It also controls the election process itself like setting up elections, managing the list of candidates, and tracking the current status of each

election. When someone casts a vote, the backend checks if the voter is eligible and hasn't already voted. If everything checks out, it turns that vote into a transaction and sends it to the blockchain. The backend also acts as a middleman between the user interface, the blockchain network, and the database. It uses bolt database to store sensitive but non-blockchain-critical information, such as hashed passwords, voter eligibility records, and admin settings.

In the system, blockchain is used for decentralized and secure purposes which consists of multiple validator nodes built in **Go** that work together to maintain the integrity of the blockchain. Each node checks incoming vote transactions to make sure they are valid for example, confirming that the voter hasn't already voted and that the transaction is properly formatted. The system uses a protocol called Practical Byzantine Fault Tolerance (PBFT) to reach consensus which means all the nodes agree on which votes are valid and in what order they should be recorded. Once agreement is reached, the votes are grouped into blocks and added permanently to the blockchain. The result is a tamper-proof ledger, where every vote is recorded securely and cannot be changed. These blocks also include Merkle trees, a kind of data structure that makes it easy to verify if a particular vote is in the blockchain without needing to see the whole block.

4.1.2 State Diagram

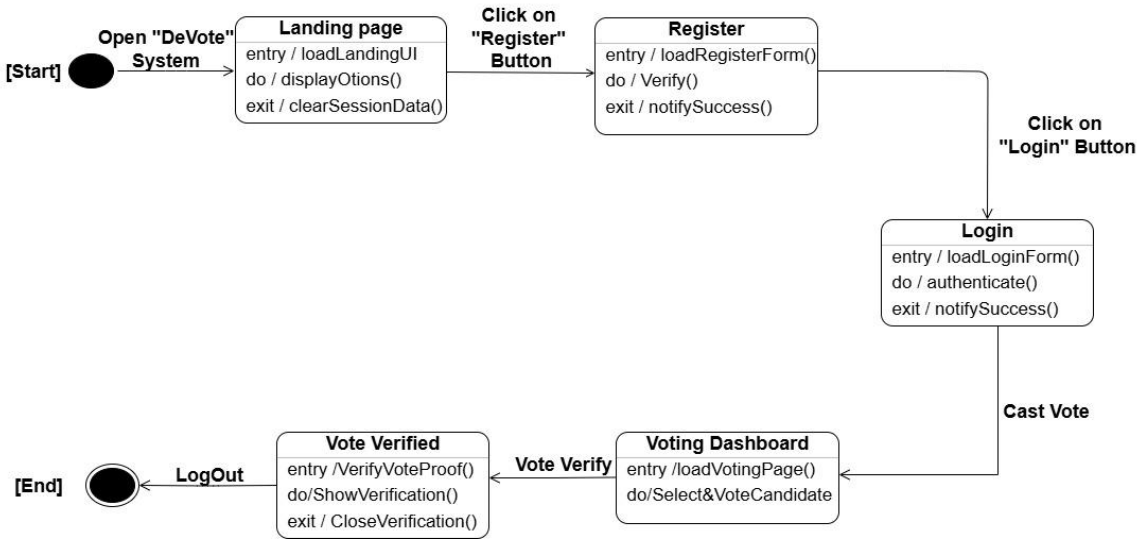


Figure 4.2 State Diagram of DeVote

4.1.3 Activity Diagram

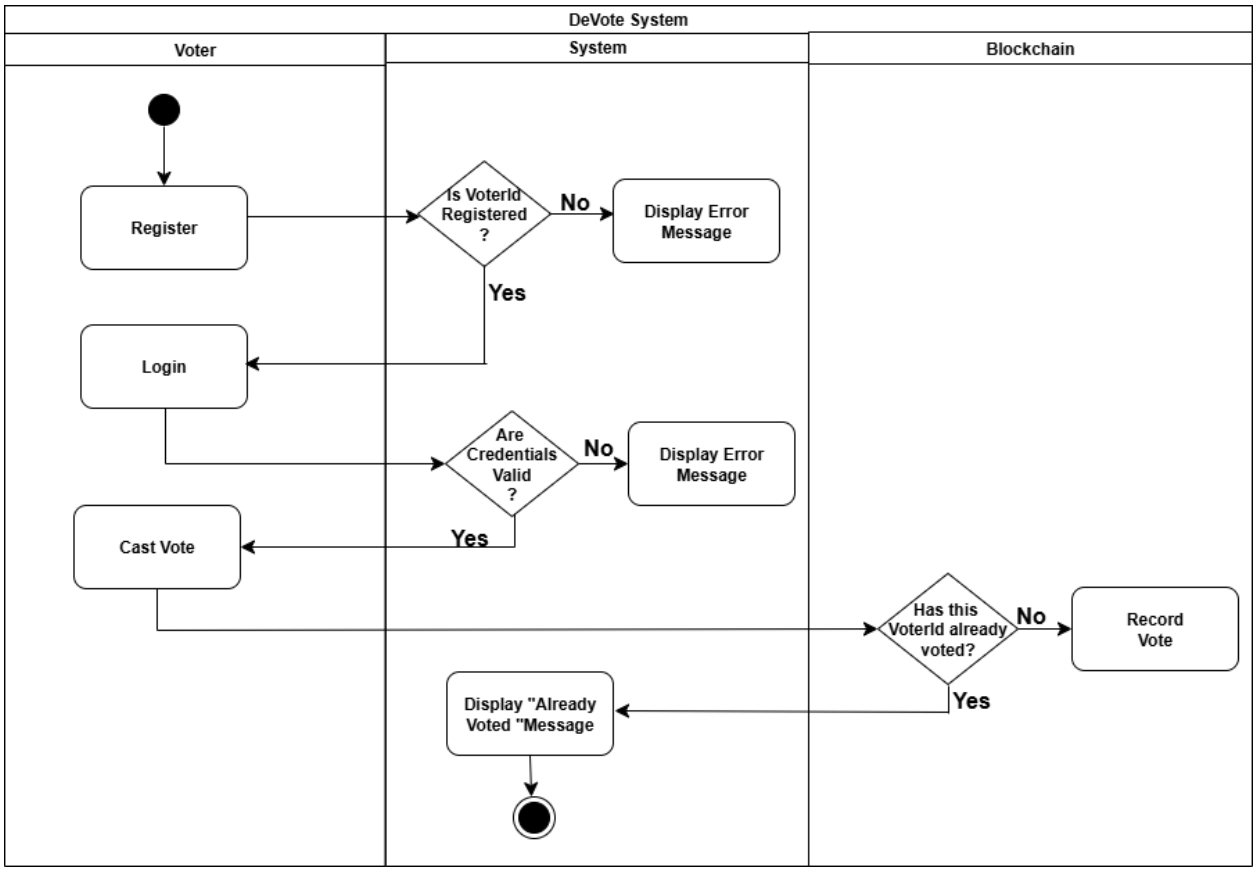


Figure 4.3 Activity Diagram of DeVote

4.1.4 Flowchart of the Working Mechanisms

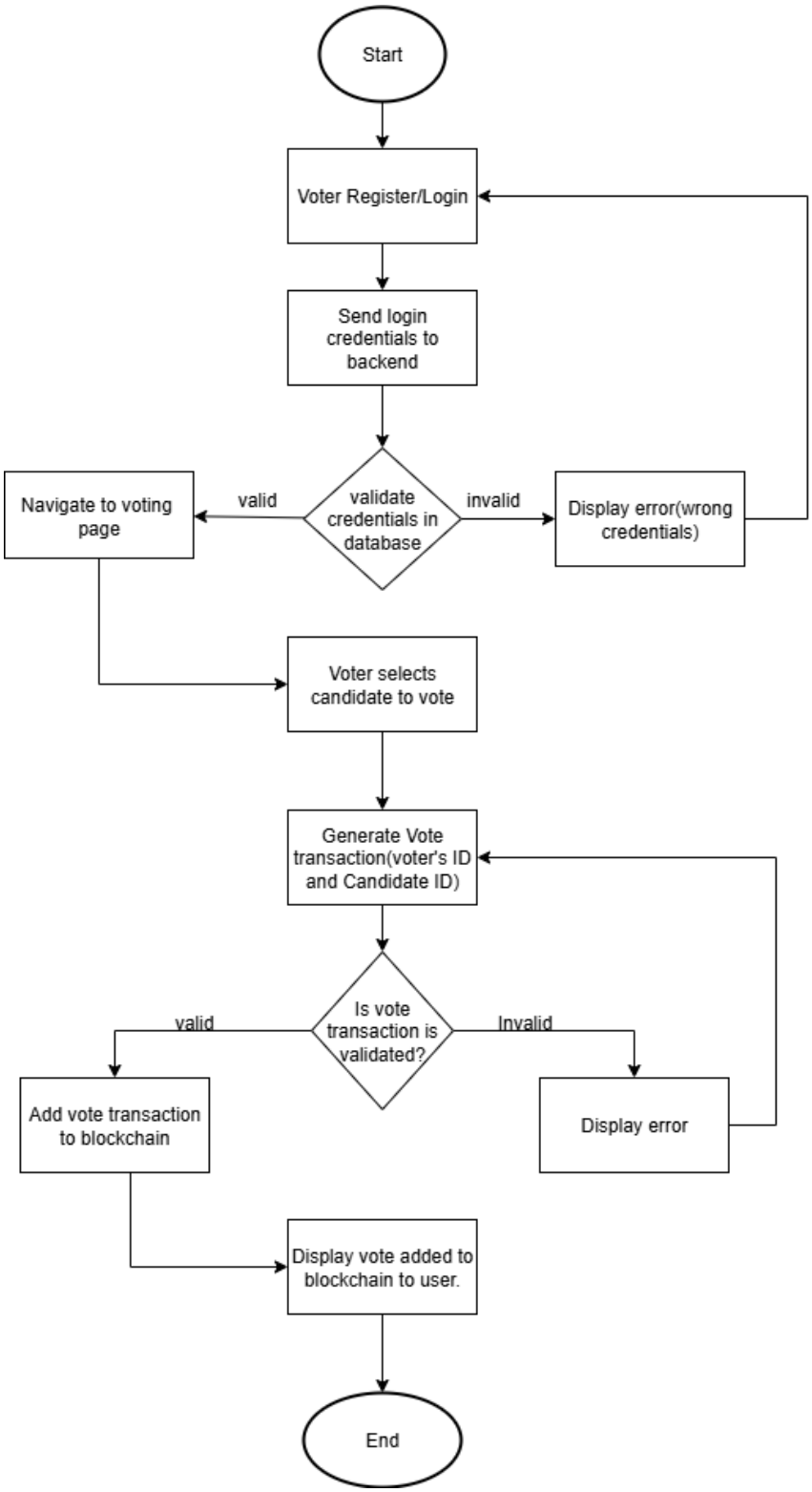


Figure 4.3:Flowchart of Voting

4.2 Algorithm Details

4.2.1 Secure Hash Algorithm-256 (SHA-256)

SHA 256 is one of the main components of the blockchain technology which is used to create a unique output (relatively known as digest, or message digest) for an input having l bits, where $0 \leq l \leq 2^{64}$. Cryptographic Hash functions have some important properties, which includes

1. Pre-image resistant (One-way function): Hash functions are one-way functions, which means calculating the input based on the output is computationally infeasible. For instance, given a digest, find x such that hash(x) = digest is not computationally possible.
2. Second pre-image resistant: It means it is not possible to find a different input for a specific output. Given x, finding y such that hash(x)=hash(y) is not computationally feasible.
3. Collision Resistant: It means we cannot find two different inputs that have same output digest. Finding an x and y such that hash(x) = hash(y) is not feasible.

For this purpose, we will be using SHA (Secure Hash Algorithm) 256 algorithm that has all the above properties. SHA 256 takes an input of any size and converts it into a fixed output of 32 bytes (256 bits). SHA 256 will be used to create unique identifiers for each block in the blockchain [4].

SHA-256 algorithm uses six logical functions which are,

- $\mathrm{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$ ……(i)
- $\mathrm{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$……(ii)
- $\sum_0(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$……(iii)
- $\sum_1(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$……(iv)
- $\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$……(v)
- $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$……..(vi)

Algorithm

1. Preprocessing: It consists of 3 steps, which are padding the message, parsing the message into blocks, and setting the initial hash value.

a. Padding the message: Add a single 1 bit followed by 0 bits to make the length of the message congruent to 448 mod 512. Finally, append the original message length l as a 64-bit binary number. Hence, the padded message becomes a multiple of 512 bits.

Padded Length ≡ 448 (mod 512)

b. Parsing the message: The padded message is split into 512-bit blocks.

Padded message = 512-bit * N blocks

c. Setting the Initial Hash Value: The initial hash value of the SHA-256 consists of following eight 32-bit words.

$$H_0^{(1)} = 6a09e667$$
$$H_1^{(1)} = bb67ae85$$
$$H_2^{(1)} = 3c6ef372$$
$$H_3^{(1)} = a54ff53a$$
$$H_4^{(1)} = 510e527f$$
$$H_5^{(1)} = 9b05688c$$
$$H_6^{(1)} = 1f83d9ab$$
$$H_7^{(1)} = 5be0cd19$$

2. Hash Computation: It uses the six previously defined logical functions. Each message block, $M^{(1)}$, $M^{(2)}$, ……, $M^{(n)}$ processed through the following steps.

a. Prepare message Schedule: Create 64 words W_0 to W_{063}, each consisting of 32 bits from each 512-bit block.

- For t=0 to 15:

$$W_t = From\ the\ current\ message\ block$$

- For t=16 to 63:

$$\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$
$$W_t = \sigma_1(W_{t-2)} + W_{t-7} + \sigma_0(W_{t-15)} + W_{t-16}$$

where, $ROTR^7(x)$ is rotate the bits of the value by 7 positions.

$SHR^3(x)$ shifts right the bits of the value by 3 positions, adding zeros from the left.

b. Compression Function

- Initialize the eight working variables with the $(i-1)^{st}$ hash value.

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

- For t=0 to 63:

$$T_1 = h + \sum_1(e) + Ch(e,f,g) + K_t + W_t \text{.....(vii)}$$
$$T_2 = \sum_0(a) + Maj(a,b,c) \text{.....(viii)}$$

h=g

g=f

e=d+T_1

d=c

c=b

b=a

a=$T_1 + T_2$

c. Compute the i[th] intermediate hash value H[(i)]:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

After repeating steps one to four a total of N times, the resulting 256-bit message digest of the message, M, is

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)}$$

4.2.2 Practical Byzantine Fault Tolerance (PBFT)

PBFT is a consensus algorithm designed for distributed systems that need to tolerate Byzantine faults (malicious or arbitrary failures, not just crashes). It provides strong consistency and finality, meaning once a transaction is committed, it's irreversible. PBFT is particularly well-suited for permissioned blockchain networks like DeVote, where the participants (validator nodes) are known.

Key Concepts:

- Replicas (Nodes): The participants in the consensus process. One replica is designated as the Primary, and the others are Backups.
- Client: The entity that sends a request (e.g., a vote transaction) to the replicas.
- Fault Tolerance: PBFT can tolerate up to f faulty replicas, where the total number of replicas N must satisfy $N \geq 3f+1$.
- View: An ordered sequence of configurations of replicas, with the primary replica for that view. Views change if the primary fails (view change protocol).
- Phases: The consensus process for a request goes through several phases to ensure all honest replicas agree on the order of operations.

Algorithm:

1. Request Phase:
    - The Voter sends a REQUEST message containing the voteTransaction to the Primary replica.
    - Message: REQUEST (voteTransaction, timestamp, clientID)
2. Pre-Prepare Phase:
    - The Primary replica receives the REQUEST. If valid, it assigns a sequence number (n) to the voteTransaction and multicasts a PRE-PREPARE message to all Backup replicas.

- This establishes an order for the transaction.
- Message: PRE-PREPARE (view, n, digest(voteTransaction))
- Note: digest is the SHA-256 hash of the transaction.

3. Prepare Phase:
   - Each Backup replica receives the PRE-PREPARE. It verifies the message (e.g., correct view, valid digest). If valid, it multicasts a PREPARE message to all other replicas (including the primary).
   - Replicas log these messages.
   - Message: PREPARE (view, n, digest(voteTransaction), replicaID)
   - A replica enters the "prepared" state for a request when it has seen 2f+1 valid PRE-PREPARE or PREPARE messages (including its PRE-PREPARE from the primary and PREPARE from other replicas) for the same view, sequence number (n), and digest.
4. Commit Phase:
   - Once a replica is in the "prepared" state, it multicasts a COMMIT message to all other replicas.
   - Message: COMMIT (view, n, digest(voteTransaction), replicaID)
   - A replica enters the "committed" state for a request when it has seen 2f+1 valid PREPARE or COMMIT messages (including its COMMIT) for the same view, sequence number (n), and digest. This signifies that a sufficient number of replicas have agreed.
5. Reply Phase:
   - Once a replica enters the "committed" state, it executes the voteTransaction (i.e., adds it to its local block and appends the block to its chain).
   - It then sends a REPLY message back to the Client.
   - Message: REPLY (view, timestamp, clientID, replicaID, result)
   - The Client waits for f+1 identical REPLY messages from different replicas to confirm that the operation has been successfully committed and that the result is correct, even if some replicas are Byzantine.

It makes sure that all the nodes agree on the same state. It also ensures that new blocks are added to the blockchain compared to other POW consensus algorithms which takes a lot of time. It can withstand a certain malicious attacks which is crucial for a voting system.

5. IMPLEMENTATION AND TESTING

5.1 Implementation

5.1.1 Tools Used

The following tools and technologies were used during the implementation of the "DeVote" project for both frontend and backend development, version control, and API testing:

1. Visual Studio Code (VS Code): An open-source, lightweight yet powerful Integrated Development Environment (IDE) used for writing, debugging, and managing code for both the React frontend and the Go backend. It offered extensive extensions for Go, ReactJS, and seamless integration with version control systems.
2. Web Browser Dev Tools: Integrated development tools available in modern web browsers like Chrome DevTools. These were used to inspect API calls, debug JavaScript code, analyze network requests, and verify correct data rendering between the frontend and backend.
3. Vite: A next-generation frontend tooling that served as the development build tool for the React application. Vite provides fast development server startup time and efficient Hot Module Replacement (HMR), enabling rapid iteration cycles on the front-end UI during development, significantly improving developer productivity.
4. Git: A distributed version control system was used for tracking changes in the codebase, managing different branches, and providing efficient collaboration among team members.
5. React: A declarative, component-based JavaScript library for building user interfaces. React was used for the frontend of DeVote due to its efficiency in creating interactive UIs and its ability to manage complex states, providing a smooth and responsive user experience.
6. Go (Golang): A statically typed, compiled programming language designed by Google, was used for developing the backend, which also included the core blockchain. Go's strong concurrency primitives, efficiency, and suitability for network programming made it an ideal choice for building the decentralized voting system's backend.
7. Postman: A popular API platform used for building, testing, and modifying APIs. Postman was used for testing the backend RESTful APIs, sending various types of

requests (GET, POST), checking responses, and ensured the correct functioning of the API endpoints that interacted with the blockchain and database.

8. BoltDB: BoltDB was used to store non-blockchain critical data such as user authentication details (hashed passwords) and internal voter eligibility lists.

5.1.2 Implementation Detail

1. User Authentication & Authorization Middleware

The system employs a multi-layered authentication framework to ensure only verified voters can participate. JWT tokens serve as the primary authentication mechanism, with additional security measures to prevent unauthorized access.

a. Token Generation & Validation:

Each voter receives a digitally signed JWT token after successful registration. The token contains encrypted voter metadata (ID, eligibility status, and district) and expires after a fixed voting window. The backend validates tokens using a secret key to prevent tampering.

```
# JWT Signature Generation
function generateToken(payload, secret):
    header = {"alg": "HS256", "typ": "JWT"}
    encodedHeader = base64UrlEncode(header)
    encodedPayload = base64UrlEncode(payload)
    signature = HMAC-SHA256(encodedHeader + "." + encodedPayload, secret)  ---(i)
    return encodedHeader + "." + encodedPayload + "." + base64UrlEncode(signature)
# Token Validation Equation
isValid = verifyHMAC256(receivedSig, computedSig) AND currentTime < payload.exp  ---(ii)
```

b. Role-Based Access Control (RBAC):

Voters can only cast votes and view their own voting history.

Election Officials can verify registrations, start/stop elections, and generate results.

Admins manage smart contracts, blockchain nodes, and system configurations.

```
function checkAccess(user, resource):
    if user.role == "voter" AND resource in ["cast_vote", "view_history"]:
        return True  ---(iii)
    elif user.role == "official" AND resource in ["verify_voter", "start_election"]:
        return True
    # ... other role checks
```

2. Voter Registration Process

Registration is a multi-step verification process to ensure only eligible voters participate.

Step 1: Identity Verification

Voters submit government-issued IDs (e.g., national ID, passport) which are cross-checked against a trusted national database.

Step 2: Blockchain-Based Voter Credential Issuance

Once verified, the system generates:

- A unique voter ID (hashed and stored on-chain).
- A one-time-use voting PIN delivered via secure channels (SMS/email).
- A digital voting certificate (stored in the voter's wallet).

```
function generateVoterID(nationalID):
    salt = cryptoSecureRandom(16)
    voterID = SHA3-256(nationalID + salt)  ---(iv)
    storeOnChain(voterID, "pending_verification")
    return voterID
```

Step 3: Immutable Audit Logging

All registration events are recorded on the blockchain, creating a permanent, fraud-resistant trail.

```
logHash = SHA256(timestamp + voterID + operationType)  ---(v)
```

3. Secure Login & Session Management

The login system balances security with usability to prevent disenfranchisement.

Two-Factor Authentication (2FA):

Voters must provide both their assigned credentials (username/PIN) and a time-based OTP (via SMS or authenticator app).

Voting Session Timeout:

To prevent vote selling or coercion, sessions automatically expire after 10 minutes of inactivity.

```
isActive = (currentTime - lastActivity) < 600000  # 10 minutes in ms  ---(vi)
```

4. Blockchain Architecture & Voting Integrity

A permissioned blockchain ensures transparency while maintaining voter privacy.

Smart Contracts for Election Rules:

- Enforce voting windows.
- Validate voter eligibility before accepting votes.
- Tally results only after the election concludes.

Hybrid Encryption for Vote Secrecy:

- Votes are encrypted using AES-256 before submission.
- Only authorized election officials can decrypt and tally votes after polls close.

function encryptVote(vote, publicKey):

sessionKey = generateAES256Key() ---(vii)

encryptedVote = AES256-CBC(vote, sessionKey) ---(viii)

encryptedKey = RSA-OAEP(sessionKey, publicKey)

return (encryptedVote, encryptedKey)

Immutable Vote Logging:

- Each vote is timestamped, hashed, and stored across multiple nodes to prevent tampering.

blockHash = SHA3-256(prevHash + timestamp + merkleRoot) ---(ix)

Post-Election Auditing:

- Any citizen can verify vote counts using cryptographic proofs without exposing individual choices.

5.2 Testing

5.2.1 Unit Testing

Table 2: Test Case for Voter Registration

Test Case ID: TC01	**Test Designed By**: Shreeya Palikhel
Test Priority (Low/Medium/High): High	**Test Design Date**: 7/4/2025
Module Name: Voter Registration	**Test Executed By**: Shreeya Palikhel
Test Title:	**Test Execution Date**: 10/4/2025
Description: Test the registration process works correctly when a new user tries to register with valid	

email and voterID. The system checks if the voterID exists in the internal voter list and ensures the user is listed in the voter list.

Pre-Conditions: System had voter list file. Registration page is accessible. **Dependencies**: Backend verification logic for voter list is implemented. Email validation logic must be correctly implemented. Frontend and Backend should be connected.

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**State(P/F)**
1	Open the registration page.		Registration form should load properly.	Registration form loads correctly.	Pass
2	Enter valid email and VoterID .	Email: asthashrestha@gmail.com VoterId: 1045	Form should accept inputs and the registration should be successful.	Form accepts input and registration is successful.	Pass
3	Enter valid email, but unlisted voterID	Email: asthashrestha@gmail.com VoterId: 100	Registration should be denied with “VoteID not listed in voter list” message	Registration denied with “VoteID not listed in voter list” message	Pass

Post-Conditions:

Only eligible and listed users can register. System securely stores valid voter records only. Unlisted users are restricted from progressing.

Table 3: Test Case for Voter Login

Test Case ID: TC02	**Test Designed By**: Shreeya Palikhel
Test Priority (Low/Medium/High): High	**Test Design Date**: 7/4/2025
Module Name: Voter Login	**Test Executed By**: Shreeya Palikhel
Test Title:	**Test Execution Date**: 10/4/2025
Description: Verify that only users who have been successfully registered (with VoterId and email verified) can log in.	

Pre-Conditions: User is already registered successfully. User knows accurate email and password. **Dependencies**: User data stored securely after eligibility check. Login verification logic is working.

Step	Test Steps	Test Data	Expected Result	Actual Result	State(P/F)
1	Open login page		Login page should load properly	Login form displayed properly	Pass
2	Enter valid username, and password of verified user.	Email: asthashrestha@gmail.com Password: secret123	Login should be successful and user should be	Login is successful and user is redirected	Pass

			redirected to voting dashboard	to voting dashboard	
3	Enter correct email/password of unregistered user.	Email: fake@gmail.com Password: secret123	Login should deny with "Account not registered"	Login denied with "Account not registered" message	Pass
4	Enter wrong password for verified user.	Email: astha@gmail.com Password: pass123	Login should deny with "Incorrect Password".	Login denied with "Incorrect Password" message	Pass

Post-Conditions: Only eligible and registered voters can logged in and granted access.

Table 4: Test Case for Double Voting Restriction

Test Case ID: TC03	**Test Designed By**: Shreeya Palikhel
Test Priority (Low/Medium/High): High	**Test Design Date**: 7/4/2025
Module Name: Voting	**Test Executed By**: Shreeya Palikhel
Test Title: Double Voting Restriction	**Test Execution Date**: 10/4/2025
Description: Ensure the system blocks the same user from casting more than one vote after	

Pre-Conditions: User already cast their vote successfully **Dependencies**: Voter status tracking is active per voter.

Step	Test Steps	Test Data	Expected Result	Actual Result	State(P/F)
1	Login with correct email and password	Email: asthashrestha@gmail.com Password: secret123	Access voting page.	Voting page displayed	Pass
2	Try to cast another vote	Candidate: Toshima Karki	Voting should be denied with "You have already voted" message	Voting denied with "You have already voted" message	Pass

Post-Conditions: One vote per verified user enforced.

Table 5: Test Case for Tamper-Proof-Blockchain

Test Case ID: TC04	**Test Designed By**: Shreeya Palikhel
Test Priority (Low/Medium/High): High	**Test Design Date**: 7/4/2025
Module Name: Blockchain	**Test Executed By**: Shreeya Palikhel
Test Title: Tamper-Proof Blockchain Verification	**Test Execution Date**: 10/4/2025
Description: Ensure that once vote is added to blockchain, It cannot be modified or removed.	

Pre-Conditions: Vote already stored in blockchain
Dependencies: Hashing and chain-linking logic implemented

Step	Test Steps	Test Data	Expected Result	Actual Result	State(P/F)
1	Access blockchain ledger	VoteId: 102	Original hash should be visible	Hash is visible	Pass
2	Try modifying stored vote	Vote changed in backend file	Blockchain should invalidate the changes.	Blockchain invalidates the change and tamper detected	Pass

Post-Conditions: Blockchain detects and prevents tampering.

Table 6: Test Case for Vote Counting

Test Case ID: TC05 **Test Designed By**: Shreeya Palikhel
Test Priority (Low/Medium/High): High **Test Design Date**: 7/4/2025
Module Name: Tally and Result **Test Executed By**: Shreeya Palikhel
Test Title: Accurate Vote Counting **Test Execution Date**: 10/4/2025
Description: Ensure that final tally only includes votes cast by verified and eligible voters

Pre-Conditions: Voting session ended All voters are verified and eligible
Dependencies: Tally logic excludes duplicate or invalid votes.

Step	Test Steps	Test Data	Expected Result	Actual Result	State(P/F)
1	Access result tally page		Count should	Count appears	Pass

			appear based on verified votes	based on verified votes	
2	Compare tally with blockchain entries	Total votes: 100	Count should match blockchain	Count matches blockchain	Pass

Post-Conditions: Results reflect only valid votes.

Table 7: Test Case for Admin Controls

Test Case ID: TC06	**Test Designed By**: Shreeya Palikhel
Test Priority (Low/Medium/High): Medium	**Test Design Date**: 7/4/2025
Module Name: Admin Controls	**Test Executed By**: Shreeya Palikhel
Test Title: Admin Dashboard	**Test Execution Date**: 10/4/2025
Description: Ensure the admin can manage candidate in the system	

Pre-Conditions: Admin is logged in and has access to admin dashboard Dependencies: Admin panel interface and backend API's working

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**State(P/F)**
1	Add a new candidate	Name: Sunta Dangol ID: Sunita12 Party: UML Age: 32	Candidate should be added successfully	Candidate added successfully	Pass

		Bio: I am a Nepalese politician, Newa heritage conservationist, language activist, and media professional	and show in the list	and shown in the list.	
2	Update candidate name	Old: Sunta Dangol New: Sunita Dangol	Name should be updated	Name is updated	Pass
3	Delete a candidate	Name: Ram Pokhrel	Candidate should remove	Candidate removes and is no longer in the list	Pass

Post-Conditions: Candidate list reflects all updates by admin.

5.3 Result Analysis

6. CONCLUSION

6.1 Modules Competed

The basic framework of the DeVote system has been completed. The entire frontend has been built using React and Tailwind CSS, offering a clean and responsive interface for both voters and administrators. The backend, developed using Go, is fully functional and handles all core operations such as user registration, login, vote submission, and result access. The frontend and backend have been properly integrated, allowing real-time communication through RESTful APIs. We have also implemented BoltDB as a lightweight database solution to store voter credentials, eligibility lists, and election-related data. Additionally, the admin dashboard is up and running, giving admin users the ability to manage candidates and oversee the election process.

6.2 Modules in Progress

At the moment, we're working on adding features to strengthen the system's reliability and transparency. One of the main functionalities in progress is the implementation of the Merkle Tree, which will help voters confirm that their vote was successfully included in the blockchain without compromising their privacy. We're also in the process of integrating the Practical Byzantine Fault Tolerance (PBFT) consensus algorithm, which will enable secure, distributed agreement across validator nodes that is needed for a trustworthy decentralized system.

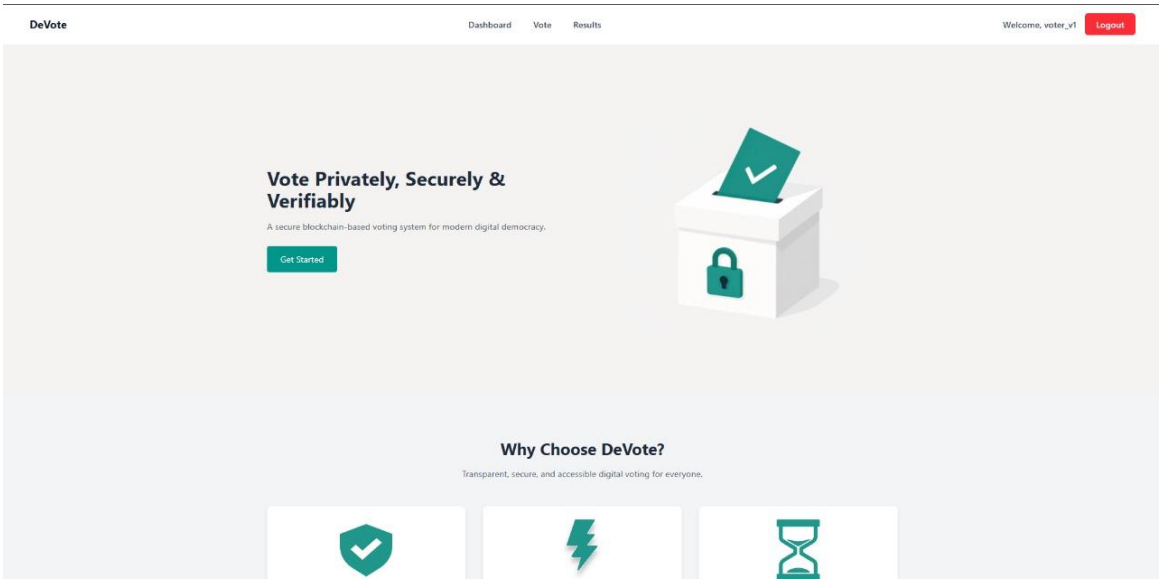
6.3 Modules Remaining

The integration of zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) is left which will allow users to prove that their vote is valid and included in the blockchain without revealing any personal information, further improving privacy and anonymity. Comprehensive testing which includes integration, and system-level testing also needs to be completed. A change in UI will be applied to improve the user interface, and the final report will be completed with documentation, screenshots, test results, and references.

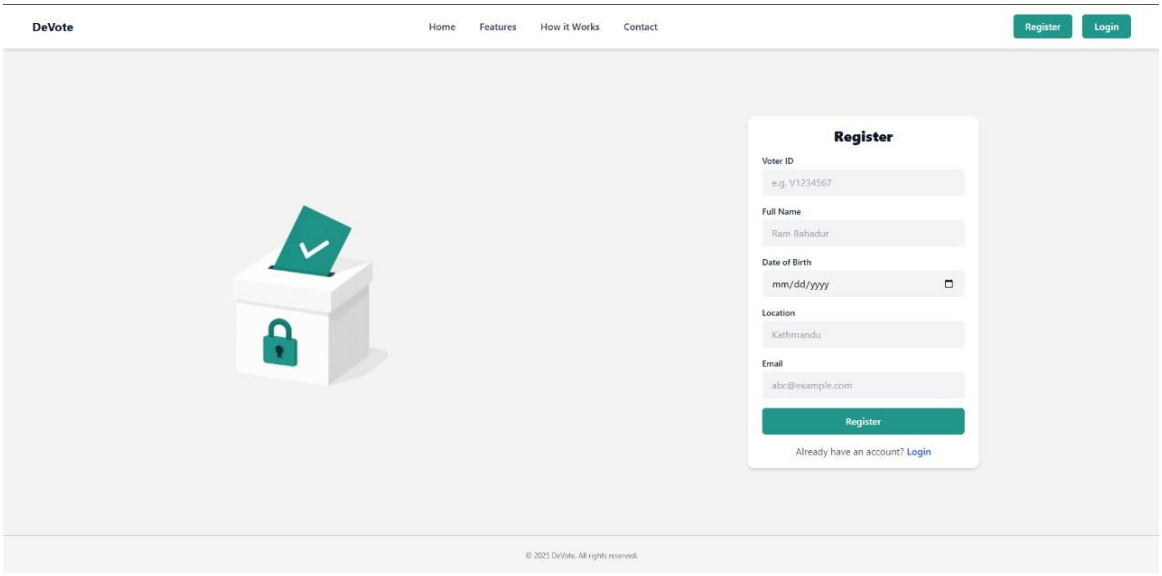
APPENDICES

Screenshots

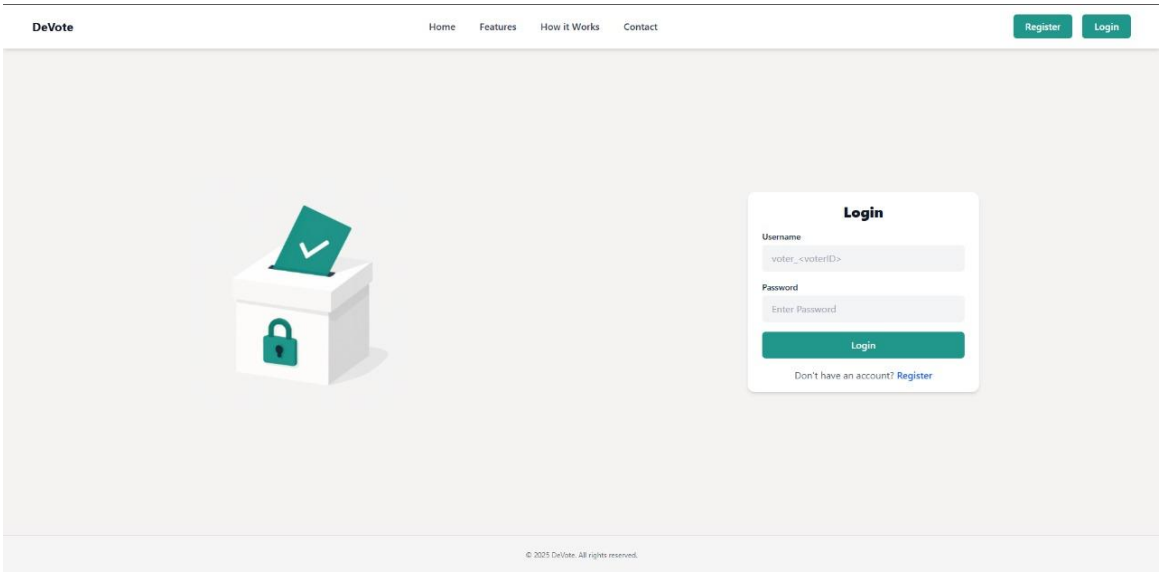
Homepage



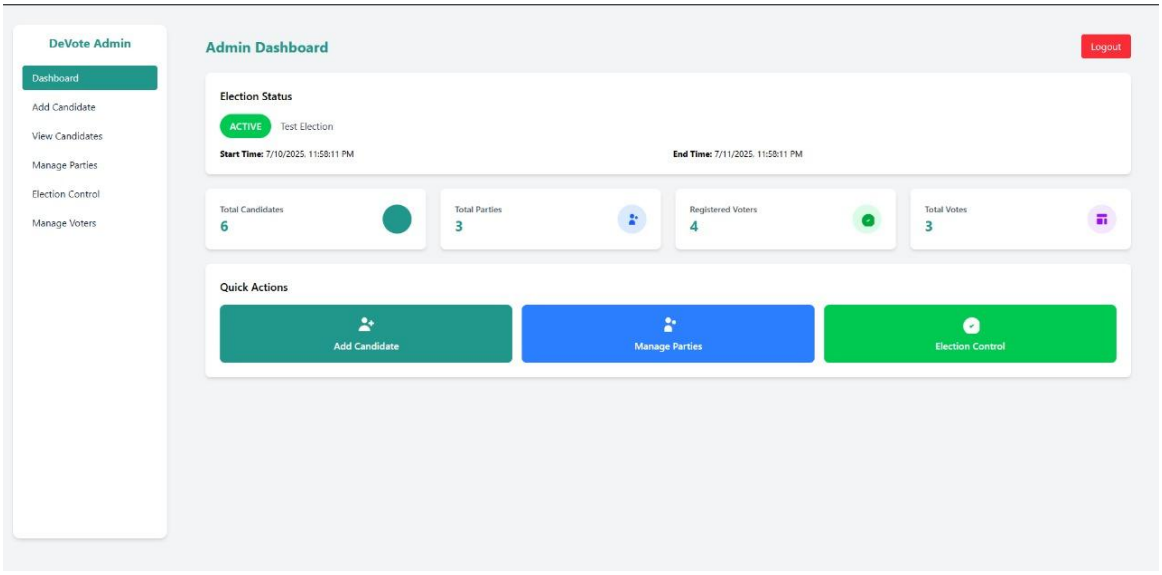
Register



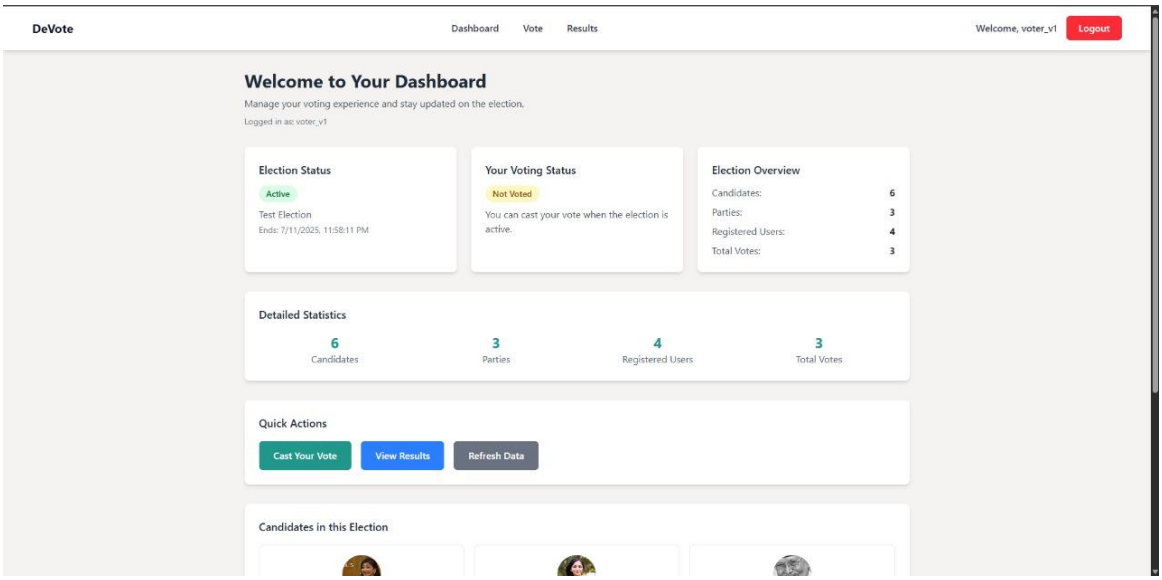
Login



Admin Dashboard



User Dashboard



References

[1] D. Yaga, P. Mell, N. Roby, K. Scarfone, "Blockchain Technology Overview," *National Institute of Standards and Technology,* vol. 1, 10 2018.

[2] "Follow My Vote, “Online Blockchain Voting,”".

[3] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," 2014.

[4] Willie E. May, "Secure Hash Standard (SHS)," *FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION,* 2015.

[5] D. Mahto and D. Kumar, "RSA and ECC: A Comparative Analysis," *International Journal of Applied Engineering Research,* vol. 12, pp. 9053-9061, 2017.

[6] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[7] B. Adida, "Helios: Web-based Open-Audit Voting," 2008.

[8] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," 2014.