AVE Trends in Intelligent Computing Systems

NextGenVote: A Trusted Blockchain-Based System for Secure Digital Voting

V. Sahaya Sakila[1,*], R. Sujeetha[2], S. Revathy[3]

[1,2,3]Department of Computer Science and Engineering, SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India.
sahayasakilav@gmail.com[1], sujeethasrm@gmail.com[2], revathyrajesh04@gmail.com[3]

*Corresponding author

Abstract: Even in this day and age, when digital technologies are becoming more and more prevalent, it is still extremely important for democratic systems to maintain the honesty and openness of their voting procedures. This article introduces NextGenVote, a decentralised online voting platform developed to address the security, transparency, and confidence issues traditional electronic voting systems face. Automation of election operations, including voter registration, candidate administration, ballot casting, and result computation, is achieved through smart contracts written in the Solidity programming language. The system is built on the Ethereum blockchain. MetaMask is a React-based frontend that uses Web3.js to connect to the blockchain. MetaMask is responsible for ensuring that user authentication and transaction signatures are secure. Therefore, to prevent unauthorised manipulation, the platform utilises a role-based access control approach that clearly distinguishes between administrative capabilities and voter credentials. NextGenVote assures that election results are tamper-proof, traceable, and auditable. It was deployed and tested in a local blockchain environment powered by Ganache. The system provides a solid foundation for scalable, secure, and transparent digital elections by eliminating centralised intermediaries and relying solely on processes executed on the blockchain.

Keywords: Blockchain and Ethereum; Smart Contracts; MetaMask and Ganache; Decentralised Application; Electoral Integrity; Zero-knowledge Proofs; Electronic Voting System.

Cite as: V. S. Sakila, R. Sujeetha, and S. Revathy, "NextGenVote: A Trusted Blockchain-Based System for Secure Digital Voting," *AVE Trends in Intelligent Computing Systems*, vol. 2, no. 4, pp. 196–206, 2025.

Journal Homepage: https://www.avepubs.com/user/journals/details/ATICS

Received on: 23/12/2024, **Revised on:** 13/04/2025, **Accepted on:** 04/06/2025, **Published on:** 12/12/2025

DOI: https://doi.org/10.64091/ATICS.2025.000212

1. Introduction

The quick move to digital democracy has changed how people in modern societies think about participation, government, and civic duty. As technology becomes increasingly part of everyday life, people expect voting systems to be quick, easy to use, and safe [2]. This move towards digital voting systems, on the other hand, has shown that traditional electronic voting methods have significant problems. Most traditional systems are built on centralised architectures that have single points of failure. This makes them vulnerable to data manipulation, unauthorised access, operational disruptions, and a lack of clear verification measures [3]. These structural weaknesses can make people less confident in the system and undermine the legitimacy of election results, especially in places where faith in institutions is already weak [4]. To address these problems, new technologies, namely blockchain, are being explored as possible solutions to long-standing issues of trust and security in digital voting

systems [5]. Blockchain technology has a decentralised structure, an unchangeable ledger, and the ability to retain records that are easily verifiable [6]. This makes it a good starting point for building secure and verifiable voting systems. Blockchain doesn't rely on a single central authority to store, process, and verify votes [7]. Instead, it spreads data across a network of nodes, ensuring that no single person controls the entire system. This decentralisation makes it much less likely that someone will tamper with, commit fraud, or make changes without permission [8]. Also, because blockchain is unchangeable, once a vote is recorded, it can't be changed without the agreement of the whole network. This makes election data more reliable and trustworthy. Transparency enables checking every transaction and maintaining a permanent record of each step of the voting process [9]. This shifts trust from institutional promises to cryptographic proof.

Smart contracts are a key reason blockchain is a good choice for digital voting [10]. These are pieces of code that run on the blockchain and perform specific actions when certain criteria are met. Smart contracts eliminate intermediaries and reduce errors and manipulation by automating key steps, including voter registration, ballot casting, vote counting, and outcome determination [11]. Because they always act the same way, the rules encoded into the system control all actions, making the election process fair, consistent, and verifiable. The blockchain records every interaction that a smart contract starts. This provides the system with an auditable trail of activity, making it more accountable while still protecting users' privacy through pseudonymous identifiers [13]. This article provides a comprehensive examination of NextGenVote, a decentralised voting system built on the Ethereum blockchain that aims to improve security, transparency, confidence, and equity in digital elections. NextGenVote shows how blockchain and smart contract technology can address many of the problems associated with traditional voting systems [14]. Using Solidity, the programming language for making smart contracts on Ethereum, the system runs the whole election process, from registering voters to adding candidates to casting votes to counting the votes and figuring out the final results. React.js is used to build the frontend interface, which makes it modern, responsive, and easy to use for both administrators and voters. Web3.js integration makes it easy to connect to the Ethereum network, and MetaMask, a popular cryptocurrency wallet, lets users log in and use the platform.

One of NextGenVote's best features is its role-based access control (RBAC), which governs the permissions granted to different users in the system. RBAC ensures that participants can only perform actions in line with their assigned duties. This reduces misuse and ensures the process is fair. Registered voters can vote only once, which keeps things fair and prevents multiple voting or voting fraud. At the same time, only authorised personnel can have administrative privileges, which allow them to start and end elections, register candidates, and handle other important election details. This organised division of duties makes the platform safer and more accountable by stopping unauthorised changes and keeping a record of all administrative operations that can be traced. Any voting system needs to protect people's privacy, and NextGenVote does so by using cryptographic Ethereum addresses to hide voters' identities rather than retaining their personal information. Votes are tied to unique blockchain accounts, which keep each user's ballot secret while still allowing the system to verify that the voter is who they claim to be and eligible to vote. This pseudonymous concept finds a middle ground between openness and privacy. It lets the public check the election results while protecting people's voting choices. Because the blockchain keeps a permanent record of all interactions, the system has a strong audit trail that anybody can check. This lets observers and stakeholders verify the correctness of the results without revealing the identities of the voters.

Ganache is a local Ethereum blockchain emulator that the system uses for development and testing. It makes it easy to quickly create prototypes, debug code, and simulate on-chain interactions. Developers can test the functionality, performance, and security of smart contracts using Ganache without paying or dealing with the hassle of deploying to the actual Ethereum network. This environment behaves like a real blockchain, allowing us to fully test the election procedure, transaction processes, and system dependability across a variety of scenarios. When testing, you can look at many situations, such as people trying to vote at the same time, people doing things they shouldn't be able to do, and potential attack scenarios, to ensure the platform works safely and effectively in real life. As democracies around the world change, the demand for reliable digital voting systems grows stronger. Paper ballots, electronic voting machines, and centralised online platforms are traditional ways to vote, but each has major problems. For example, counting can take a long time, they can be manipulated, they cost a lot to run, and they aren't very accessible to voters who live far away or are disabled. Digital mistrust persists because of ongoing cyberattacks, data breaches, and claims of manipulated or tainted elections. Blockchain-based voting systems offer a forward-thinking approach to addressing these problems by integrating security and openness into the technology itself. NextGenVote shows how blockchain can decentralise trust. This means voters don't have to rely on central authorities and can monitor the election process themselves.

Blockchain-based voting systems may also be able to handle more voters. They can be used for community referendums, organisational elections, academic institutions, professional associations, and choices about business governance. The technology can lower expenses and increase participation rates by operating safely without requiring physical polling places or a large number of administrative staff. Blockchain's transparency and verifiability may also help rebuild public faith in election institutions. This is important in a time of widespread misinformation, political division, and doubt about institutional processes. NextGenVote and other blockchain voting systems have significant potential. Still, it's also vital to be aware of the problems

and issues that need to be addressed before everyone can use them. Problems with network congestion, transaction costs, digital literacy requirements, and the need to make it available to people without access to technology could all make implementation harder. A major challenge with remote voting systems is ensuring that identity verification is secure while protecting people's privacy. Also, for blockchain-based voting to be fair and reliable, substantial infrastructure, legal frameworks, and public education would be needed for large-scale national elections. Even with these problems, NextGenVote's progress shows how such systems can be improved, expanded, and tailored to different situations. Overall, NextGenVote offers a safe, practical, and open way to rethink digital voting using blockchain and smart contract technology. The solution addresses many of the problems that afflict traditional electronic voting platforms by leveraging decentralised infrastructure, automated processes, cryptographic privacy, and unambiguous access controls. The successful creation and testing of this prototype show that blockchain voting systems can be a practical, scalable, and reliable option for small-scale or organisational elections. As research and innovation continue to progress, these technologies may facilitate more secure and transparent democratic processes, ultimately fostering improved governance and increased public trust in electoral results.

2. Literature Review

Because of its inherent properties of transparency, immutability, and decentralisation, blockchain technology has been widely researched for digital voting. Several academic and industry-led efforts have proposed blockchain-based solutions to address the limitations of centralised electronic voting systems. Zero-knowledge proofs (ZKPs) were investigated by Fazekas [1] to facilitate anonymous voting on Ethereum, highlighting the potential of privacy-preserving solutions in public blockchains. This method introduces increased cryptographic complexity and greater computational costs, while also guaranteeing voter anonymity. Alternative methods, as described in Wood [15] and Antonopoulos and Wood [12], highlight the role of smart contracts in automating voting procedures. This work is predominantly concerned with the feasibility of decentralised voting, with minimal consideration of user-centred frontend design or real-world feasibility.

Blockchain-based voting systems have been provided by projects such as Voatz and FollowMyVote. These solutions, however, often employ hybrid designs that combine centralised identity management with off-chain storage via mechanisms such as IPFS. Such an architecture could simplify on-chain data loads; however, it could compromise the trustless nature of decentralised systems. NextGenVote's solution is entirely on-chain, which eliminates external dependencies on current models. It uses commonly used Ethereum tools, such as a web-based frontend, cryptographic user authentication via MetaMask, and Solidity, Truffle, Ganache, and Web3.js, to provide an end-to-end solution. It preserves decentralisation throughout the voting process by avoiding offloading computations or data to centralised servers, unlike many previous systems. NextGenVote helps fill the gap between conceptual blockchain voting frameworks and practically deployable systems by combining an implementation that works with a highly user-centric user interface and close on-chain enforcement of state.

3. System Design and Architecture

Figure 1 presents a layered architectural overview of NextGenVote. It delineates the flow of data and user interaction across the admin and voter interfaces, MetaMask wallet authentication, Web3.js middleware, and the React.js frontend. These components interface with the smart contract logic deployed on a local Ethereum blockchain via Ganache.

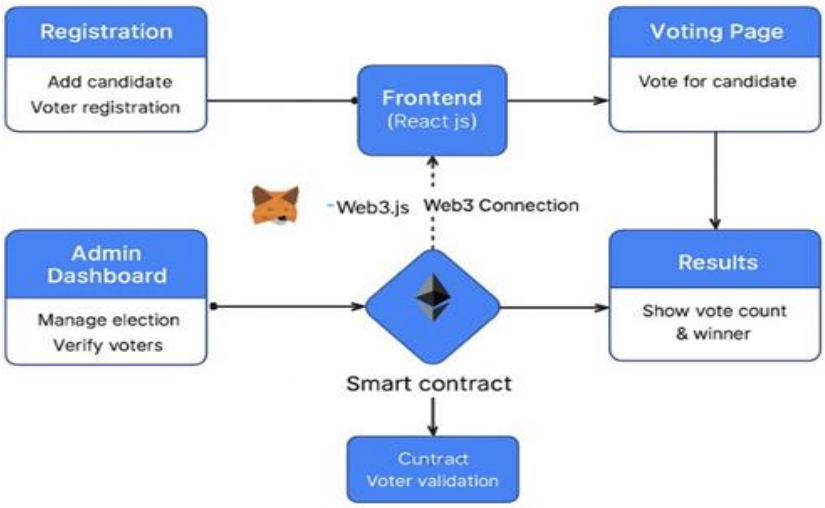

Figure 1: Workflow diagram of NextGenVote

The diagram reflects the modular nature of the system and how real-time updates and secure transactions are handled end- to-end (Figure 2). The architecture of NextGenVote is designed to be modular, decentralised, and secure. It comprises the following key components:

- **Registration Module:** Allows administrators to add candidates and voters to the system.
- **Frontend Interface:** Built using React.js, this connects to the blockchain using Web3.js and MetaMask.
- **Admin Dashboard:** Offers administrative controls for managing the election and verifying voters.
- **Voting Module:** Enables verified voters to cast their vote securely.
- **Smart Contracts:** Written in Solidity, these enforce business logic like one vote per address, vote counting, and election state.
- **Results Module:** Displays results in real time by reading blockchain data.

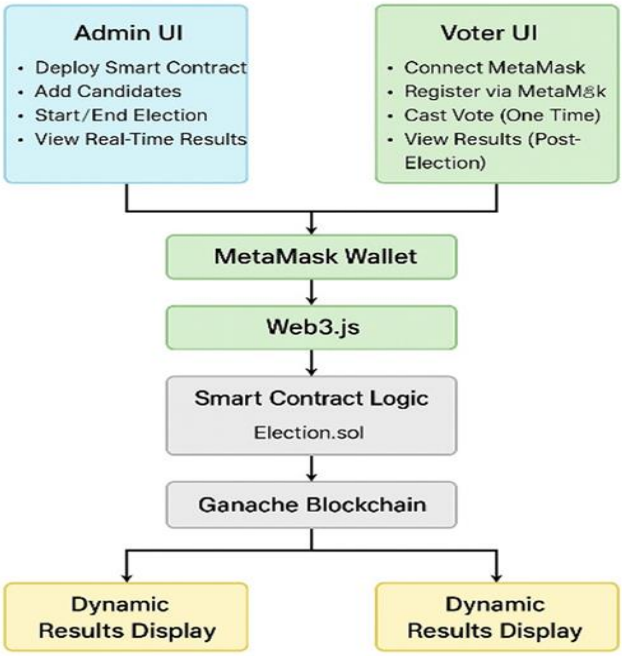

Figure 2: System architecture of NextGenVote illustrating the interaction between user layers, frontend components, smart contracts, and blockchain

The entire system is secured by blockchain immutability and cryptographically signed MetaMask transactions. Web3.js facilitates communication between the frontend and smart contracts (Table 1).

Table 1: Comparison with existing e-voting platforms

System	Transparency	Auditability	Decentralization
Traditional EVM	✓		
Voatz FollowMyVote	✓	✓	
NextGenVote	✓	✓	✓

Table 2 outlines the layered architecture of the proposed voting system, from the user interface to the blockchain backend. It highlights how React, MetaMask, Web3.js, Solidity smart contracts, and Ganache work together to enable secure and decentralised election management.

Table 2: System architecture components

Layer	Description
User Interface	React.js with role-based routing
Wallet Integration	MetaMask for key handling and transactions

Web3 Middleware	JavaScript bridge to Ethereum nodes
Smart Contracts	Solidity-based election logic
Blockchain Backend	Ganache simulates the Ethereum network.

4. Methodology

The development of NextGenVote follows a layered methodology to achieve a secure, decentralised, and fully transparent online voting system. The process involves the coordinated design and integration of smart contracts, blockchain interaction layers, and a role-based frontend interface.

4.1. Smart Contract Development

At the foundation, the smart contract Election.sol is written in Solidity. It encapsulates all election-related logic, including candidate registration, voter registration, voting, and result computation. The contract defines two core roles—admin and voter—and uses mappings to track registration status, voting activity, and vote counts. Constraints such as one vote per address and role-based permissions are enforced using Solidity modifiers and require () conditions.

Local Blockchain Setup: For development and testing, Ganache simulates a personal Ethereum blockchain. This environment provides deterministic behaviour and enables rapid contract deployment, testing, and iteration without the cost and latency of public networks. Contract migration is managed using the Truffle framework, which also facilitates unit testing and contract interaction via JavaScript.

Frontend Integration: The frontend is implemented in React.js and is structured using components that distinguish between voter and admin roles. On application load, the system detects an injected Web3 provider (typically MetaMask) using the getWeb3.js utility function. Once connected, the system retrieves the user's Ethereum address and establishes a contract instance for interaction.

Web3 and MetaMask Communication: Web3.js acts as the bridge between the frontend and the Ethereum blockchain. It enables the frontend to invoke smart contract functions asynchronously, while MetaMask handles key management and transaction signing. This ensures that all actions—such as casting a vote or verifying a voter—are cryptographically secured and recorded on-chain.

4.2. Data Consistency and Feedback

To ensure real-time feedback, the application continuously monitors blockchain state variables, including voting status, candidate lists, and voter eligibility. React's state management ensures that changes to blockchain data are reflected in the user interface immediately, creating a seamless, transparent experience. By adhering to this methodology, NextGenVote achieves a tightly integrated system where all components—from smart contracts to the user interface—operate in unison to support secure, auditable, and trustless digital elections.

5. Implementation

The implementation of NextGenVote is divided into three core modules: the smart contract backend, the React.js frontend, and the middleware facilitating blockchain interaction. Together, these components form a decentralised application (DApp) that operates without centralised control.

Smart Contract Logic: The election logic is implemented in the Election.sol smart contract. It defines structs for Candidate and Voter and maintains mappings to store candidate details, voter statuses, and vote counts. Functions such as addCandidate(), registerAsVoter(), verifyVoter (), and vote() are protected with access controls using onlyAdmin and conditional require() checks. The contract also manages the election state using boolean flags start and end.

Truffle Suite and Ganache: The contract is deployed using the Truffle Suite, which automates compilation, migration, and testing. Ganache provides a local Ethereum environment for running and debugging transactions. This setup ensures consistent test results, quick redeployment, and cost-free simulations of real blockchain behaviour.

Frontend Interface: The user interface is developed using React.js. It includes separate views for administrators and voters, rendered based on user roles. For example, only the admin can access routes such as/AddCandidate and /StartEnd, while voters can register and vote via dedicated components. React hooks and state management ensure responsive user interaction and real-time updates.

Web3 Integration: Web3.js facilitates interaction between the frontend and the blockchain. Upon startup, the system uses getWeb3.js to detect MetaMask and initialise Web3. Contract artefacts generated by Truffle are used to instantiate the deployed contract. Each transaction—such as casting a vote or verifying a voter—is routed through Web3, signed in MetaMask, and broadcast to the blockchain.

Real-Time Result Visualisation: The frontend retrieves vote counts and election status using read-only smart contract functions. Data is displayed in tables and updated dynamically as votes are cast. Candidate lists, registration logs, and election state transitions are fetched in real time from the blockchain, ensuring transparency and auditability. Security Considerations: Key security features include:

- Role-based access control using msg. sender
- One-vote-per-address enforcement
- MetaMask-based transaction signing
- Immutability of on-chain data to prevent tampering

While gas fees are not incurred on Ganache, the contract is designed with gas efficiency in mind. Compact storage structures and minimal state changes are used to optimise performance for future deployment on public testnets or mainnet.

6. Results and Discussion

To evaluate the execution efficiency of the deployed smart contract, researchers estimated the gas consumption for each critical function within Election.sol. As shown in Figure 3, functions such as vote() and registerAsVoter () incur higher gas costs due to state-changing operations and updates to the mapping. The values were derived from simulated transactions on a local Ganache blockchain using standard Solidity compiler optimisations. Although the platform currently operates in a test environment, these metrics provide a baseline for assessing gas-cost overhead and can inform future optimisation strategies when deployed on live Ethereum testnets or the mainnet (Appendix A).

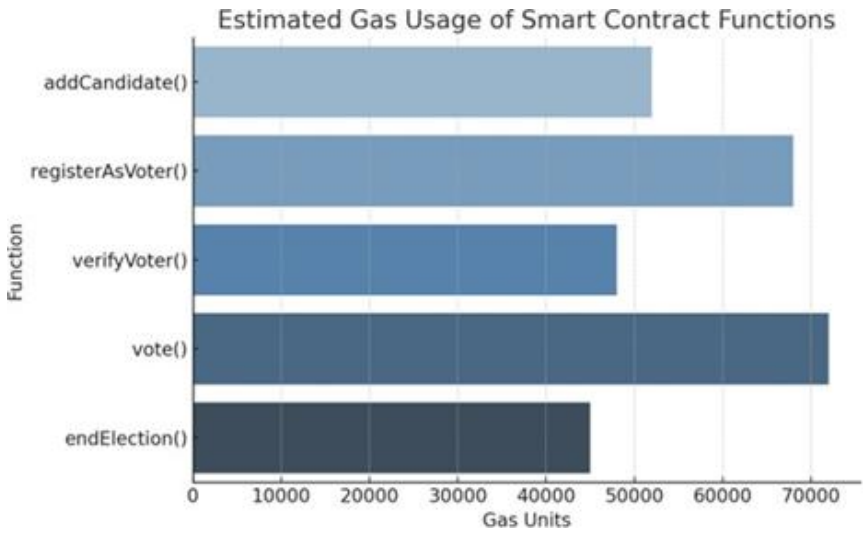

Figure 3: Estimated gas consumption for core functions in the Election sol smart contract, with values representing typical execution costs during local testing on Ganache

The NextGenVote system was successfully deployed and tested on a local Ethereum blockchain using Ganache. The testing environment simulated real-world interactions between administrators and voters through MetaMask-enabled browser sessions. The system's functionality, responsiveness, and integrity were validated through multiple voting cycles, demonstrating its feasibility for small-scale, decentralised elections.

System Verification: All core functionalities—including voter registration, candidate addition, vote casting, and result computation—were verified to operate as expected. The smart contract reliably enforced rules, such as one vote per verified voter, and prevented unauthorised access to admin-specific functions. Each transaction triggered corresponding changes in the contract's state variables and emitted blockchain events, confirming correct execution.

Frontend Integration: The React.js frontend dynamically reflected the blockchain state. For example, vote counts are updated in real time as transactions are mined, and the interface is adapted based on user roles. Election status (started or ended) was displayed accurately through read-only function calls. Error handling was also implemented to handle issues such as MetaMask disconnection or duplicate voting attempts.

Data Immutability and Transparency: All actions—including voter registrations and ballot submissions—were immutably stored on-chain. The system guaranteed transparency by exposing public read functions, enabling real-time auditing without backend servers. The public could view results without login credentials, maintaining accessibility while not compromising security.

Security and Trust Model: The platform enforced a trustless environment by removing reliance on centralised servers or manual oversight. Role-based access control ensured that only authorised accounts could manage elections, while MetaMask's cryptographic signing ensured that every transaction was non-repudiable. The immutable ledger provided an auditable trail of every action, enhancing trust and legitimacy.

6.1. Limitations

Although successful in a local testing environment, the current implementation does not address real-world scalability challenges such as network latency, gas fees, and blockchain congestion. Additionally, while anonymity is preserved through Ethereum addresses, advanced privacy features such as zero-knowledge proofs were not implemented to avoid added complexity. Overall, the results validate the practicality of blockchain-based voting systems for secure, transparent elections within controlled environments. The architecture supports modular upgrades and enables real-world pilot deployments across academic, organisational, and civic use cases. Figure 4 illustrates a simulated election result, visualising the number of votes each candidate received. These vote counts are derived directly from on-chain data stored in the smart contract and retrieved via read-only Web3.js functions. Although this dataset represents a test election on Ganache, it demonstrates the platform's ability to reflect real-time, immutable vote counts. The frontend dynamically queries the blockchain, ensuring that results are updated in real time and remain tamper-proof throughout the voting process.

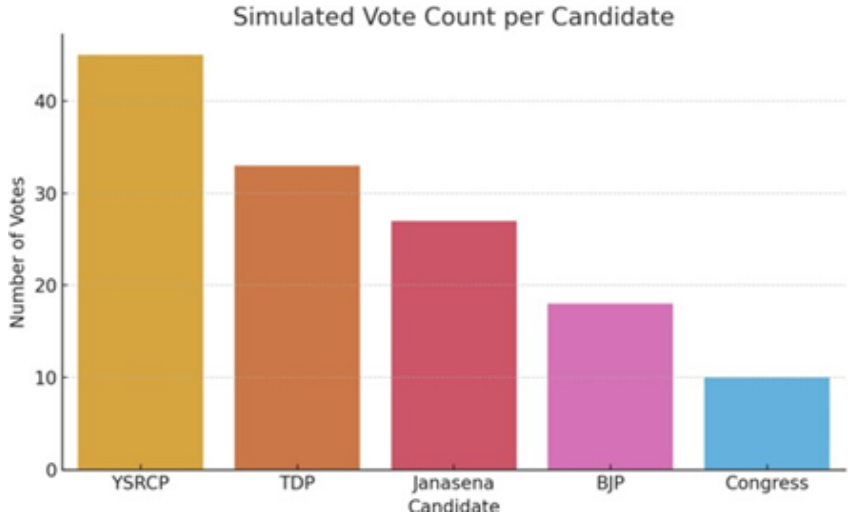

Figure 4: Simulated vote distribution among candidates as captured on the blockchain

7. System Walkthrough and Use Case Demonstration

To illustrate the practicality and user experience of NextGen-Vote, researchers conducted a series of controlled demonstrations that simulated the entire election lifecycle. This section presents a step-by-step walkthrough based on the implemented user interface and smart contract logic.

Voter Registration and Authentication: Users initiate the process by connecting their MetaMask wallet to the dApp. On the /Registration page, voters provide their Ethereum address, full name, and registered phone number. Upon clicking Register, a blockchain trans- action is initiated, requiring MetaMask confirmation. Once confirmed, the user's registration details are stored on-chain in the Voter struct.

Admin Verification: The administrator accesses the /Verification route to review the list of registered voters. Each entry displays the account address, voter name, phone number, and current status flags: Voted, Verified, and Registered. Upon confirming the voter's legitimacy, the admin clicks Approve, triggering a smart contract transaction that updates the Verified flag.

Gas Cost to Verify the User: MetaMask transaction request triggered during the user verification process in the dVoting app. It indicates a gas fee of 0.02 ETH for interacting with the smart contract deployed on the local blockchain (Ganache). This step ensures that only authenticated users can be added or verified on the platform via blockchain validation.

Adding the Candidates: When the admin adds a new candidate to the election, a MetaMask transaction is triggered to store the candidate's details on the blockchain. This operation incurs a gas cost of approximately 0.02 ETH, ensuring the addition is securely logged and immutable. It reinforces the integrity of the election by preventing unauthorised or off-chain candidate entries. To officially start the election, the admin needs to approve a blockchain transaction via MetaMask. This action costs approximately 0.02 ETH in gas fees and updates the smart contract to mark the beginning of the voting phase. This ensures a transparent and tamper-proof transition to the live election state. Verified users can access the /Voting page. The interface displays a list of registered candidates, each accompanied by a Vote button. A vote cast is confirmed via MetaMask and stored on-chain. The administrator manages the election lifecycle via the /StartEnd route. Initially, the admin clicks Start to begin the election, which updates the election status on-chain. 2 Once voting concludes, the admin clicks End (Figure 5).

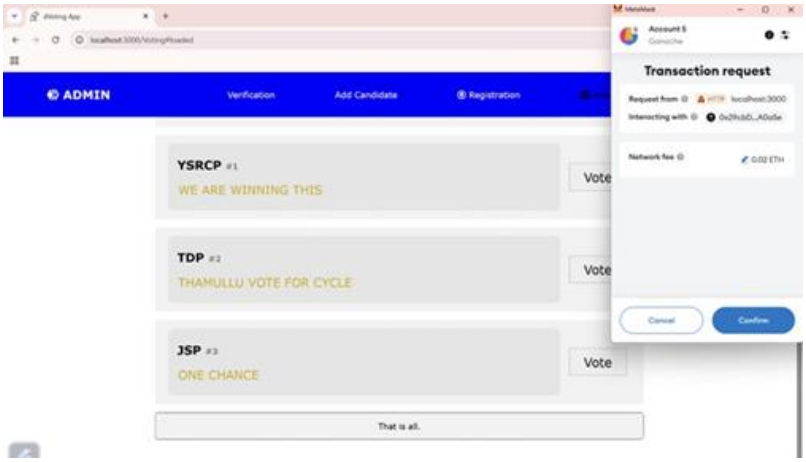

Figure 5: Admin panel showing election status: Started = True, Ended = False

After the election ends, all users can navigate to the /Results page. The system queries the blockchain for candidate vote counts and displays them dynamically (Figure 6).

Figure 6: Results page displaying the winner and vote count per candidate

Auditability and Transparency: Every action, from voter registration to result declaration, is backed by signed blockchain transactions. This ensures traceability and non-repudiation, hallmarks of trustworthy electoral systems. Through this end-to-end use case demonstration, NextGen-Vote validates its readiness for deployment in secure, mid-scale digital elections where auditability, user accessibility, and trustlessness are critical requirements.

8.1. Conclusion and Future Work

This paper presented NextGenVote, a decentralised, blockchain-based voting platform designed to enhance the security, transparency, and auditability of electronic elections. By leveraging Ethereum smart contracts, React.js, Web3.js, and MetaMask, the system successfully eliminates centralised points of failure and automates critical electoral functions, including voter verification, candidate management, voting, and result computation. The platform was tested in a simulated blockchain environment using Ganache, confirming its reliability in enforcing role-based permissions, preventing double voting, and enabling real-time visualisation of results. All election-related data was recorded immutably on-chain, establishing a trustless and verifiable audit trail. The frontend provided a user-friendly interface with dynamic updates based on blockchain state, ensuring accessibility for both administrators and voters.

8.2. Future Work

While the current prototype is effective for small-scale elections, several areas of improvement remain. Future enhancements will include:

- Migration to public Ethereum testnets (e.g., Goerli, Sepolia) for real-world simulation
- Integration of Layer-2 scaling solutions such as Opti- mistic Rollups or zk-Rollups to reduce gas costs and increase throughput
- Support for off-chain storage using IPFS for metadata and large data assets
- Implementation of advanced privacy features like Zero- Knowledge Proofs (ZKPs) or ring signatures to ensure voter anonymity
- Expansion to support multiple concurrent elections with modular ballot configurations
- Mobile DApp compatibility and biometric-based identity verification

NextGenVote demonstrates the viability of decentralised electoral systems and lays the groundwork for future deployments in institutional, organisational, and public-sector voting scenarios.

Appendix A

A.1. Smart Contract Code Snippets

A.1.1. Voter Registration Function

```
function registerAsVoter (string memory _name, string memory
        _phone) public {
      require (! voters [msg. sender]. isRegistered, " Already
            registered");
      voters [msg. sender] = Voter (_name, _phone, false, false, true);
      voterList.push(msg. sender);
}
```

Figure A1: Function to register a new voter

Figure A1 shows the smart contract function that registers a new voter by saving their personal information and ensuring they haven't already registered. This stops duplicate entries (Source: Proposed blockchain-based voting smart contract).

A.1.2. Vote Casting Function

```
function registerAsVoter (string memory _name, string memory _phone)
       public {
      require (! voters [msg. sender]. isRegistered, " Already
            registered");
      voters [msg. sender] = Voter (_name, _phone, false, false, true);
      voterList.push(msg. sender);
}
```

Figure A2: Function for a verified voter to cast their vote

Figure A2 shows the function that allows a confirmed voter to cast their vote after verifying their eligibility. This ensures that the vote is recorded on the blockchain in a secure, tamper-proof manner (Source: Proposed blockchain-based voting smart contract).

A.1.3. Election Lifecycle Management

```
function registerAsVoter (string memory _name, string memory _phone)
        public {
      require (! voters [msg. sender]. isRegistered, " Already registered");
      voters [msg. sender] = Voter (_name, _phone, false, false, true);
      voterList.push(msg. sender);
}
```

Figure A3: Functions to start and end the election lifecycle

Figure A3 depicts the smart contract methods that safely start and end the voting process, which is what controls the election lifecycle. These routines ensure that voting occurs only during the official election period (Source: Proposed blockchain-based voting smart contract).

Acknowledgement: The authors would like to express their sincere gratitude to SRM Institute of Science and Technology and other collaborating institutions for providing the academic support and resources necessary for this research.

Data Availability Statement: The datasets supporting the findings of this study are available from the authors upon reasonable request, ensuring transparency and facilitating reproducibility.

Funding Statement: The authors collectively confirm that this research and the preparation of the manuscript were carried out without external funding support.

Conflicts of Interest Statement: The authors declare that there are no known financial or personal relationships among them that could have influenced the work reported in this paper.

Ethics and Consent Statement: All authors have reviewed and approved the final manuscript and agree to its publication and accessibility for academic and educational purposes.

References

1. L. Fazekas, “How I built an anonymous voting system on the Ethereum blockchain using zero-knowledge proof,” *Medium*, 2023. Available: https://thebojda.medium.com/how-i-built-an-anonymous-voting-system-on-the-ethereum-blockchain-using-zero-knowledge-proof-d5ab286228fd [Accessed by 02/10/2024].
2. MetaMask, “MetaMask: A crypto wallet and gateway to blockchain apps,” *MetaMask website*, 2023. Available: https://metamask.io/ [Accessed by 06/10/2024].
3. Truffle Suite, “Truffle: Smart Contracts Made Sweeter,” *Truffle Suite*, 2023. Available: https://trufflesuite.com/ [Accessed by 08/10/2024].
4. Truffle Suite, “Ganache: Personal blockchain for Ethereum development,” *Truffle Suite*, 2023. Available: https://trufflesuite.com/ganache/ [Accessed by 12/10/2024].
5. Solidity Documentation, “Solidity: The smart contract programming language,” *Solidity*, 2023. Available: https://docs.soliditylang.org/ [Accessed by 16/10/2024].
6. Web3.js, “Ethereum JavaScript API,” *Web3.js*, 2023. Available: https://web3js.readthedocs.io/ [Accessed by 17/10/2024].
7. React.js, “A JavaScript library for building user interfaces,” *React*, 2023. Available: https://legacy.reactjs.org/ [Accessed by 19/10/2024].
8. Node.js, “Node.js JavaScript runtime,” *Node.js*, 2023. Available: https://nodejs.org/ [Accessed by 21/10/2024].
9. Etherscan, “Ethereum blockchain explorer,” *Etherscan*, 2023. Available: https://etherscan.io/ [Accessed by 23/10/2024].
10. OpenZeppelin, “Secure smart contract library for Ethereum,” *OpenZeppelin*, 2023. Available: https://openzeppelin.com/ [Accessed by 25/10/2024].

11. Express.js, “Fast, unopinionated, minimalist web framework for Node.js,” *Express*, 2023. Available: https://expressjs.com/ [Accessed by 26/10/2024].
12. A. M. Antonopoulos and G. Wood, “Mastering Ethereum: Building Smart Contracts and Dapps,” *O’Reilly*, Sebastopol, California, United States of America, 2018.
13. N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” *in Proc. 6th Int. Conf. Principles of Security and Trust (POST)*, Uppsala, Sweden, 2017.
14. M. Swan, “Blockchain: Blueprint for a New Economy,” *O’Reilly*, Sebastopol, California, United States of America, 2015.
15. G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, no. 4, pp. 1-32, 2014.