



K S R INSTITUTE FOR ENGINEERING AND TECHNOLOGY
TIRUCHENGODE - 637 215

Computer Science and Engineering

NAAN MUDHALVAN
SB8024- Blockchain Development
by Naan Mudhalvan Scheme – 2023

TEAM ID: NM2023TMID11749

PROJECT DOMAIN: BLOCKCHAIN TECHNOLOGY

PROJECT TITLE: BIOMETRIC SECURITY SYSTEM FOR VOTING
PLATFORM USING BLOCKCHAIN

TEAM MEMBERS

REGISTER NUMBER	NAME
731620104006	DEEPA R
731620104014	DURGADEVI K S
731620104042	PRIYADHARSHINI S

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
1.	INTRODUCTION 1.1 Project Overview 1.2 Purpose	4
2.	LITERATURE SURVEY 2.1 Existing problem 2.2 References 2.3 Problem Statement Definition	5
3.	IDEATION & PROPOSED SOLUTION 3.1 Empathy Map Canvas 3.2 Ideation & Brainstorming	7
4.	REQUIREMENT ANALYSIS 4.1 Functional requirement 4.2 Non-Functional requirements	12
5.	PROJECT DESIGN 5.1 Data Flow Diagrams & User Stories 5.2 Solution Architecture	16
6.	PROJECT PLANNING & SCHEDULING 6.1 Technical Architecture 6.2 Sprint Planning & Estimation 6.3 Sprint Delivery Schedule	21
7.	CODING & SOLUTIONING 7.1 Feature 1 7.2 Feature 2 7.3 Database schema	23

8.	PERFORMANCE TESTING	32
	8.1 Performance Metrics	
9.	RESULTS	34
	9.1 Output Screenshots	
10.	ADVANTAGES & DISADVANTAGES	36
11.	CONCLUSION	38
12.	FUTURE SCOPE	38
13.	APPENDIX	40
	Source Code	
	GitHub & Project Demo Link	

1. INTRODUCTION

1.1 Project Overview

The Biometric Security System for voting platforms is a groundbreaking initiative aimed at revolutionizing the security and credibility of electoral processes. In an age where the integrity of elections is of utmost concern, this project seeks to develop and implement a cutting-edge system that leverages biometric technology to authenticate and secure the identities of voters. Our primary objectives are to eliminate the risk of voter impersonation, enhance the overall security of voting platforms, promote transparency in the electoral process, and foster public trust in the democratic system.

1.2 Purpose

The primary purpose of the Biometric Security System for voting platforms is to fortify the integrity and security of the electoral process in an increasingly digital age. This project seeks to address the pressing concerns related to election fraud, voter impersonation, and the overall vulnerability of traditional voting methods. By incorporating biometric technology, the system's fundamental purpose is to ensure that each vote is cast by the legitimate voter, eliminating the possibility of impersonation and enhancing the overall security of the voting platform. Furthermore, the purpose extends to promoting transparency and public trust in the electoral process. By implementing a decentralized ledger to securely store voting data, the system provides an unprecedented level of transparency while safeguarding the anonymity of voters.

2. LITERATURE SURVEY

2.1 Existing Problem

The current state of electoral systems worldwide faces significant challenges related to security, authenticity, and public trust. Traditional voting methods, relying on paper ballots or electronic voting machines, are vulnerable to a range of issues, including voter impersonation, ballot tampering, and hacking. These vulnerabilities have, at times, cast doubt on the legitimacy of election outcomes and have eroded public confidence in the democratic process. Moreover, the absence of a foolproof mechanism for verifying the identity of voters has allowed for instances of multiple voting and other fraudulent activities, which threaten the fairness and accuracy of elections. The lack of transparent, tamper-resistant mechanisms for recording and storing voting data exacerbates these problems, making it difficult to verify the integrity of election results.

2.2 References

Biometric security systems have been considered for enhancing the security and integrity of voting platforms. While I can provide some information and references up to my last knowledge update in January 2022, it's essential to note that the use of biometrics in voting systems can be a complex and controversial topic, as it raises concerns about privacy, security, and accessibility. Always consult with experts and adhere to the latest legal and ethical standards when considering such solutions.

NIST (National Institute of Standards and Technology): NIST is a reputable source for standards and guidelines related to biometric systems. They provide valuable information on biometric technologies, testing methodologies, and performance standards.

IEEE (Institute of Electrical and Electronics Engineers): IEEE publishes numerous papers, journals, and conferences related to biometrics, which can be a good source of information for implementing biometric security in voting systems.

IEC (International Electrotechnical Commission): IEC provides international standards for electrical, electronic, and related technologies, which can be relevant to the development of biometric systems for voting.

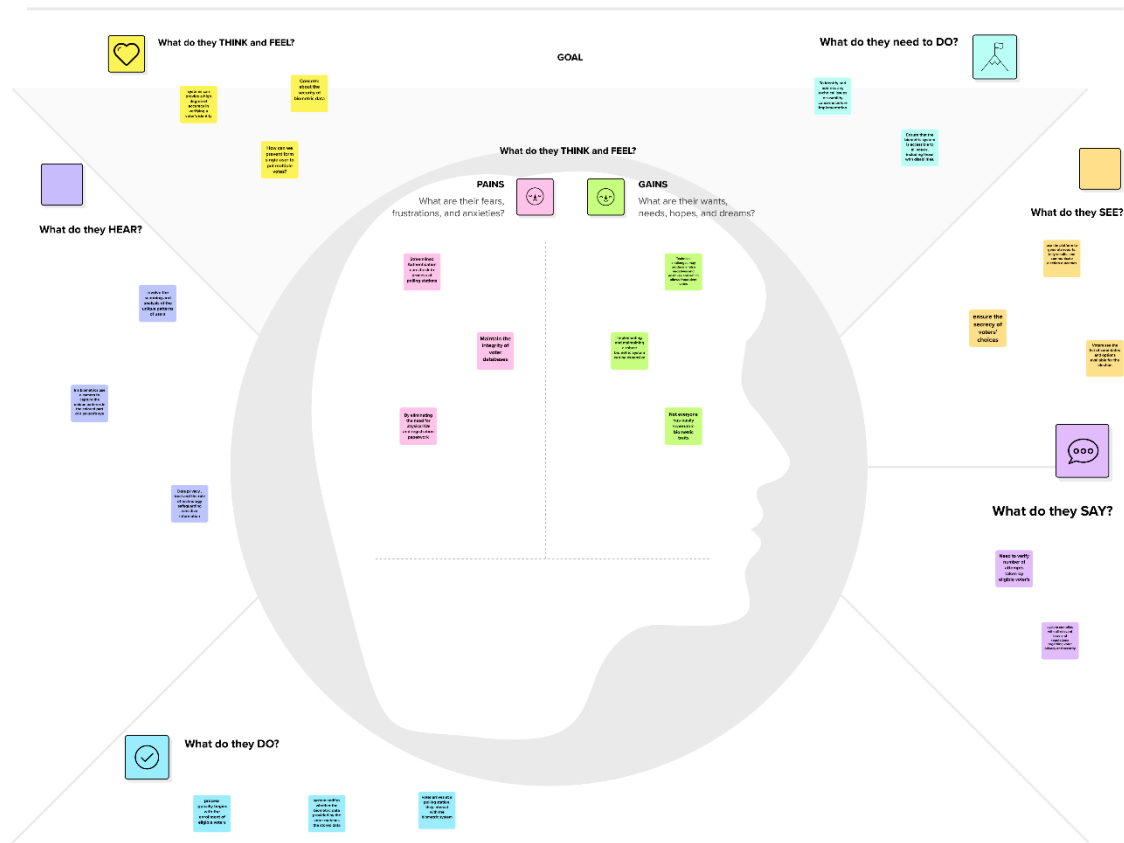
ISO (International Organization for Standardization): ISO develops and publishes international standards applicable to biometric technologies, ensuring consistency and interoperability in biometric systems.

2.3 Problem Statement Definition

The problem at hand is the systemic vulnerability of contemporary voting platforms to fraudulent activities and the erosion of public confidence in electoral integrity. Traditional voting methods, be they paper-based or electronic, are susceptible to issues such as voter impersonation, ballot tampering, and cyberattacks, which undermine the sanctity of the electoral process. Furthermore, the absence of robust identity verification mechanisms enables multiple voting and other illicit actions that jeopardize the fairness and transparency of elections. These challenges collectively constitute a problem that calls for the development and implementation of a Biometric Security System for voting platforms, which aims to mitigate vulnerabilities and enhance trust by leveraging biometric technology and secure ledger systems transparency in library operations.

3. IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation and Brainstorming



define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm,

 **10 minutes**

Design and implement a blockchain-based library management system to address the challenges faced by traditional library management systems, including data security, accessibility, and transparency, while also ensuring efficient and cost-effective operations for libraries of varying sizes and resources. Libraries play a vital role in knowledge dissemination, research, and education. However, traditional library management systems often face challenges such as data security, transparency, and efficient resource allocation. In the digital age, there is a growing need to leverage emerging technologies like blockchain to address these issues and create a more secure, transparent, and efficient library management system.

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Deepa R

Biometric
Voter
Registration

Real-Time
Verification

Data
Anonymization

DurgaDevi K S

Secure
Transmission

Redundancy
and Failover

Biometric
Liveness
Detection

Priyadharshini S

Voter
Rights

Secure
Biometric
Storage


Reduction
in
Paperwork

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

TIP  Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Deepa R

- Biometric authentication which help in reduce the fraudulent
- Prevent multiple voting attempts by the same individual

DurgaDevi K S

- Protecting voters' privacy while verifying their identity.
- Implement backup systems and failover mechanisms

Priyadharshini S

- Provide a transparent and tamper resistant record of who voted
- Implement multiple biometric authentication methods

4

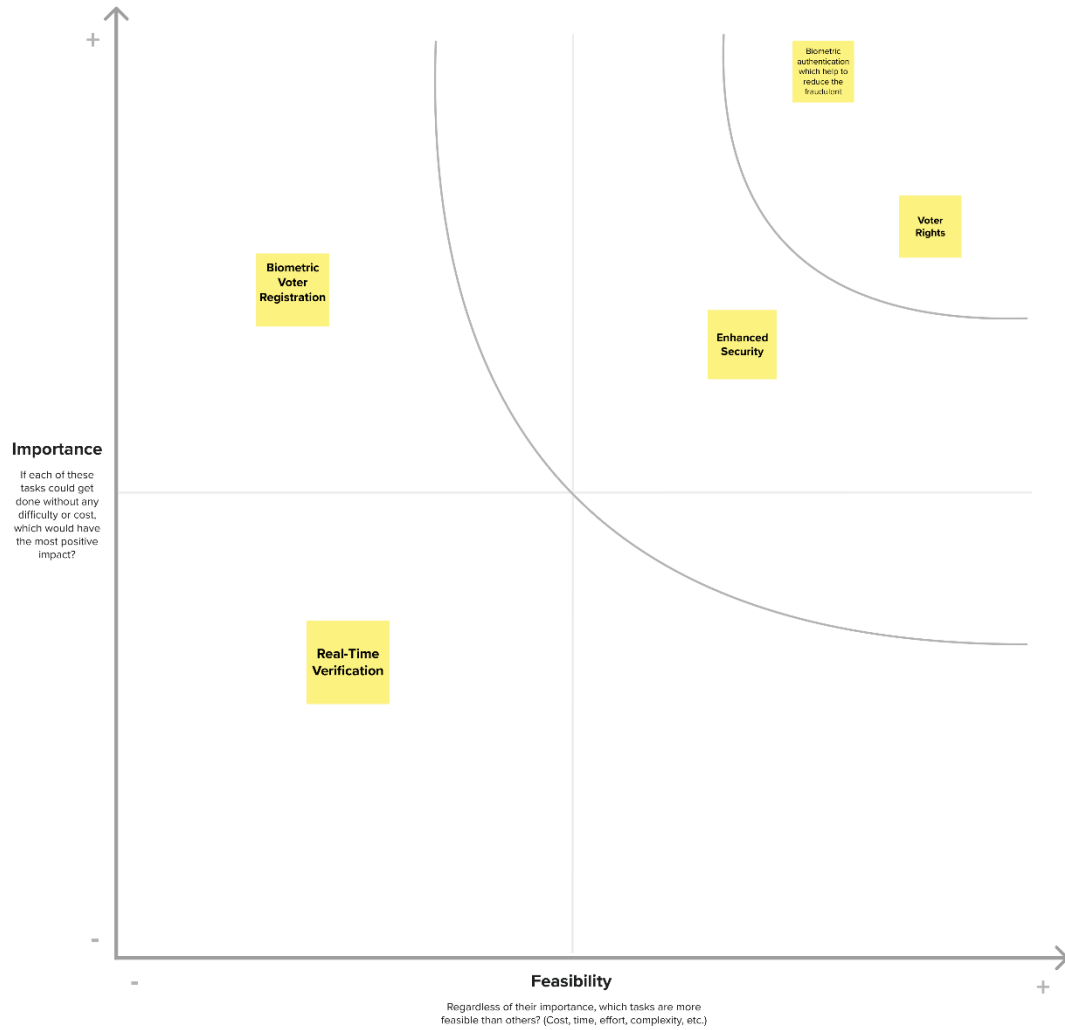
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

🕒 20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H** key on the keyboard.



4.REQUIREMENT ANALYSIS

4.1 Functional Requirements

User Management:

- i. **User Registration:** The system should provide a user registration process for administrators, election officials, and other authorized personnel. User registration should collect essential information, including name, contact details, and assigned roles.
- ii. **User Roles and Permissions:** The system must support multiple user roles, such as super administrators, election officials, and system operators, each with specific permissions and access rights. Administrators should have the ability to define and customize user roles.
- iii. **Access Control:** The system should enforce access control to ensure that users can only access functionalities and data relevant to their roles. Access control rules should be based on the principle of least privilege, where users are given the minimum access necessary to perform their duties.
- iv. **User Profile Management:** Users should be able to manage their profiles, including updating contact information and, where applicable, biometric data. Changes to user profiles should be recorded in an audit trail for accountability.
- v. **Integration and Compatibility:** The system should integrate seamlessly with existing election infrastructure, including voter databases, if applicable. It must be compatible with various

hardware and software components commonly used in the election.

- vi. **Scalability:** The system should be scalable to accommodate varying numbers of voters and increasing demands during elections.

Catalog Management:

- i. **Catalog Creation:** The system should allow administrators to create and manage catalogs of eligible voters. Catalogs should be created based on criteria such as geographic location, age, or other relevant factors.
- ii. **Voter Information:** The catalog should store voter information, including name, biometric data, voter ID, and relevant demographic details. It should support the updating of voter information as needed.
- iii. **Biometric Data Integration:** The catalog should integrate with the biometric data repository to link each voter's identity with their biometric information. Biometric data should be securely stored and associated with the correct voter.
- iv. **Voter Verification:** The system should allow election officials to verify voter eligibility by accessing the catalog. Verification should be based on biometric data to prevent impersonation.
- v. **Catalog Search and Retrieval:** Users should be able to search for and retrieve voter information efficiently from the catalog using various search criteria (e.g., name, voter ID, biometric data). The system should provide fast and accurate results.
- vi. **Catalog Security:** Catalog data should be encrypted to prevent unauthorized access or data breaches. Access to the catalog should be

limited to authorized personnel.

4.2 Non-Functional Requirements:

Security:

- i. **Data Security:** The system should provide robust data encryption and protection mechanisms to safeguard voter and biometric data from unauthorized access or breaches.
- ii. **Authentication Security:** The biometric authentication process should be highly secure and resistant to spoofing or tampering.

Performance:

- i. **Response Time:** The system should respond to user interactions within acceptable time frames to ensure a smooth and efficient voting process.
- ii. **Throughput:** The system should handle a high volume of concurrent users during election periods without performance degradation.

Reliability:

- i. The system should be highly reliable, with minimal failure rates, to ensure the integrity of the electoral process.
- ii. The system should maintain comprehensive audit logs to track user activities and system events for accountability and forensic analysis.

Usability:

- i. The system should have an intuitive and user-friendly interface to ensure that voters and election officials can easily use it without extensive training.

Interoperability:

- i. The system should be compatible with various hardware and software components commonly used in election processes and should support standardized data exchange formats.

Performance Testing:

- i. The system should undergo rigorous testing, including security testing, performance testing, and usability testing, to ensure its reliability and effectiveness.

Data Backup and Recovery:

- i. The system should have robust data backup and disaster recovery mechanisms to ensure data integrity and continuity of operations in case of system failures or disasters. Implement regular automated data backups to protect against data loss.

Testing and Quality Assurance:

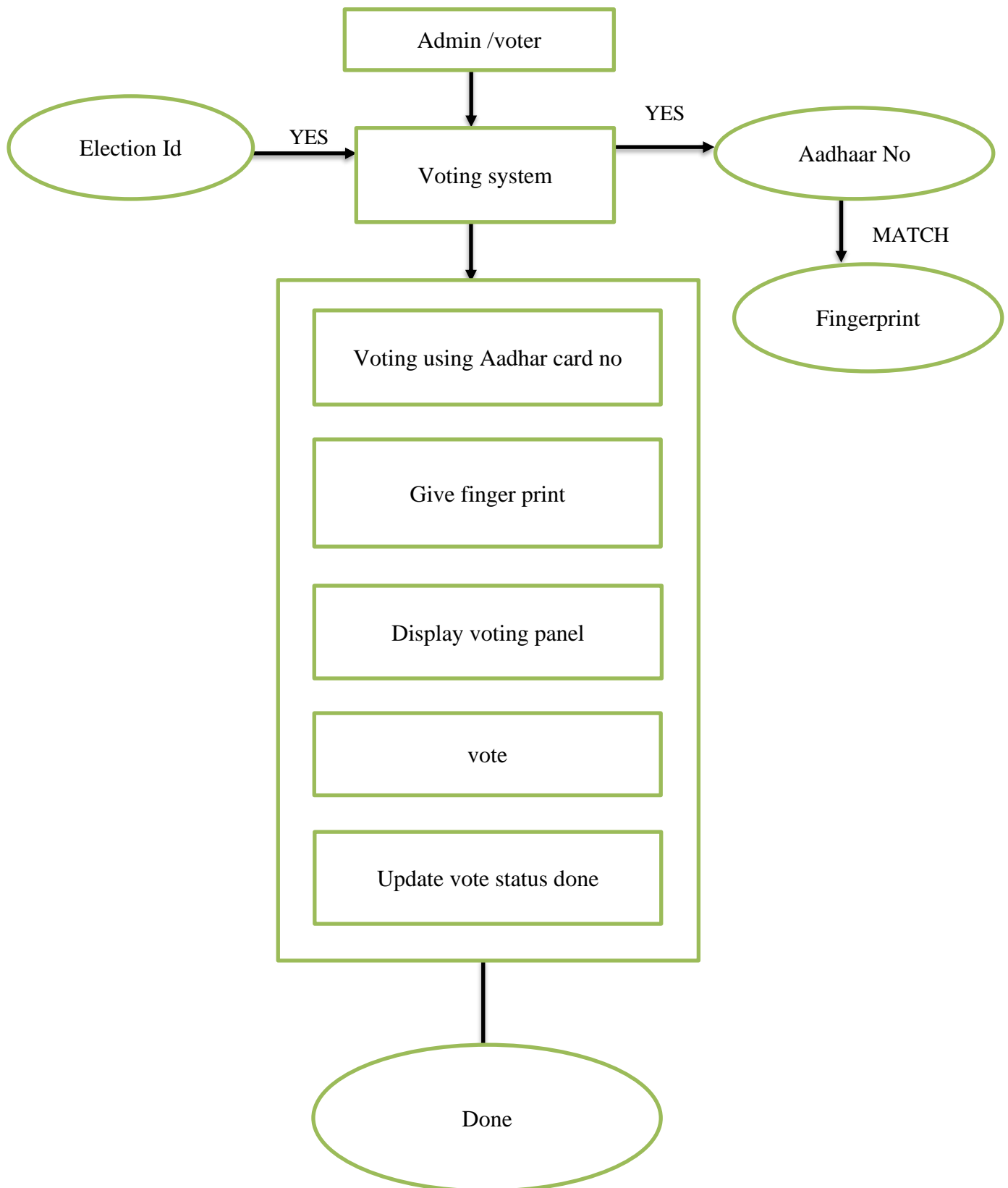
- i. The system should undergo rigorous testing, including security testing, performance testing, and usability testing, to ensure its reliability and effectiveness.

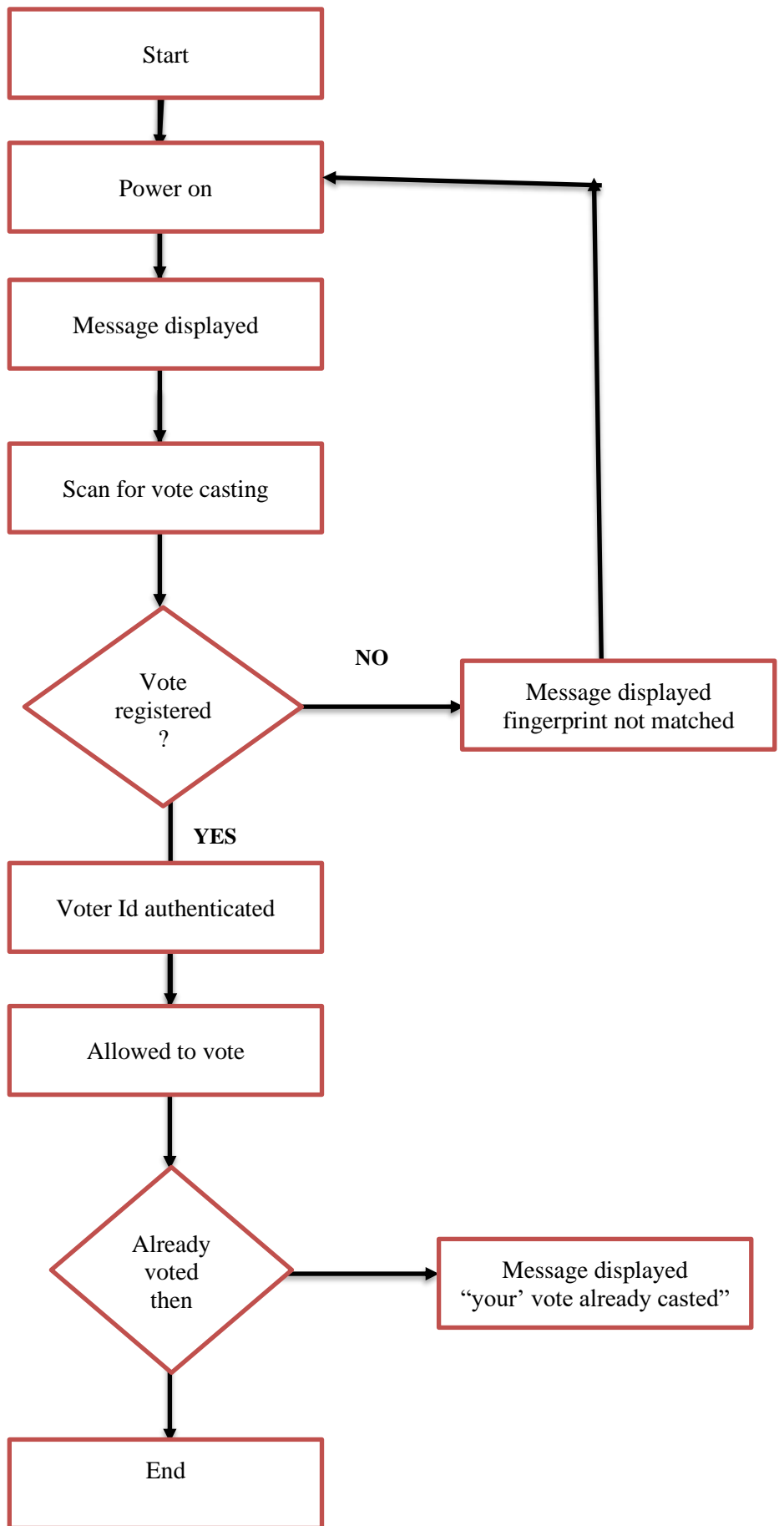
Support and Maintenance:

- i. The system should have provisions for ongoing support, maintenance, and updates to address issues and improve system performance.

5. PROJECT DESIGN

5.1 Data Flow Diagram & User Stories





5.2 Solution Architecture

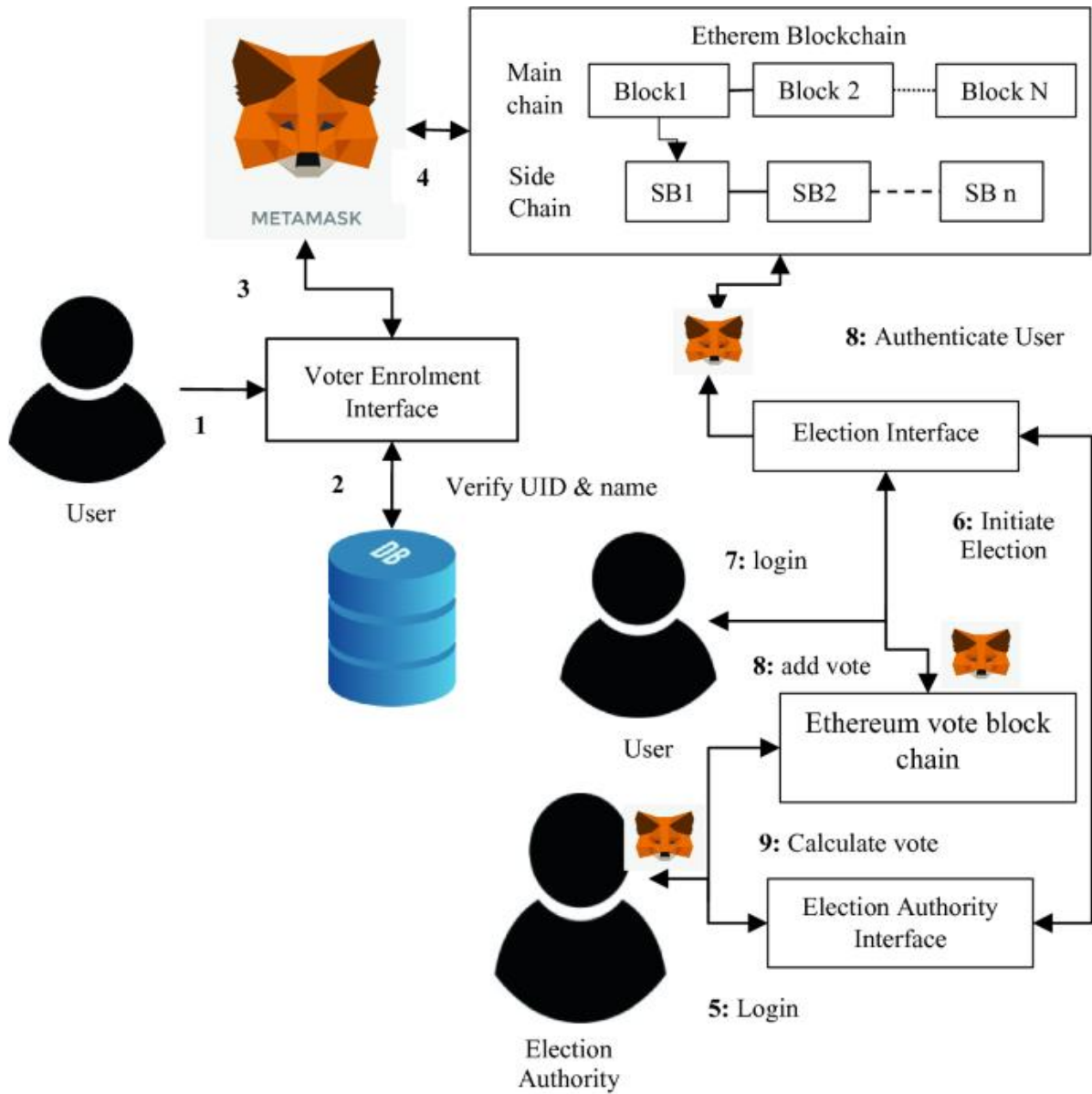
Story1 :

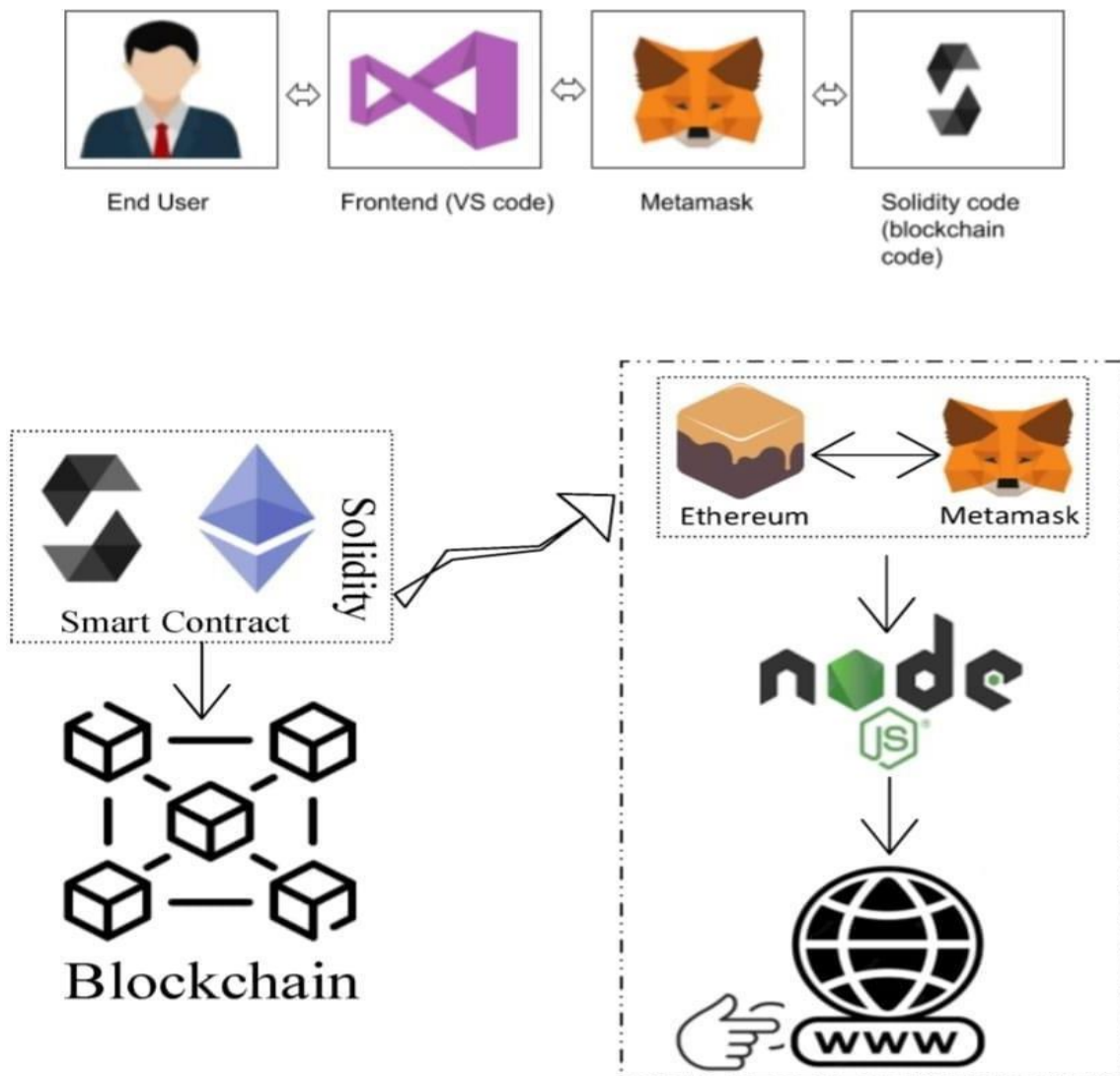
John, the Indian citizen person, logs in using secure blockchain credentials and searches for a voter rights. He had eligibility rights, and a smart contract on the blockchain confirms it. The system updates the catalog and notifies John of the due date. He added a unique information using the biometric security system like facial recognition and finger print and his voter details, another smart contract verifies it and get confirmed unique identification. John's experience the Blockchain- Biometric Security System's efficiency and security, benefiting both users and administrators.

Story2 :

Once upon a time in a thriving democracy named Tacolandia, the citizens were preparing for a crucial election. The previous elections had been marred by suspicions of fraud and impersonation, causing many to question the integrity of the democratic process. Mostly a person put a vote for favorite person multiple times where it becomes break of public trust and most votes are fake due to impersonation identity. The Tacolandia government recognized the urgent need for a solution that would enhance trust and security.

Solution Architecture

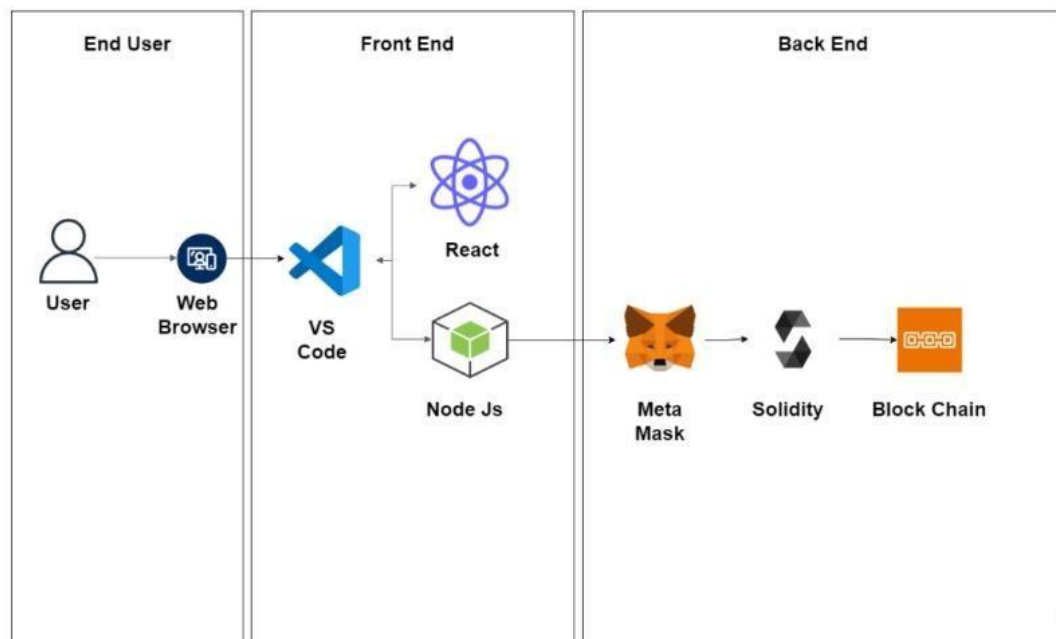




Interaction between web and the Contract

6. PROJECTPLANNING & SCHEDULING

6.1 Technical Architecture



GENERAL ARCHITECTURE

6.2 Sprint Planning and Estimation

Sprint planning and estimation for a biometric security system, like any software development project, involves breaking down the work into manageable tasks, estimating the effort required for each task, and planning for a sprint or development iteration. Here's how you can approach sprint planning and estimation for a biometric security system:

- **Gather Requirements:**

Start by gathering and understanding the detailed requirements for the biometric security system. This should include the specific biometric methods to be used (e.g., fingerprint, facial recognition), integration with existing systems, security and compliance requirements, and user experience expectations.

- **Product Backlog:**

Create a product backlog that lists all the features, user stories, and tasks required to build the biometric security system.

- **Sprint Planning:**

Determine the sprint duration. Sprints in Agile development typically last 2-4 weeks. In collaboration with the development team and stakeholders, select a subset of items from the product backlog to work on during the sprint. These should be the highest-priority items that can be completed within the sprint.

- **User Story Breakdown:**

Break down user stories or tasks into smaller, more manageable sub-tasks. For example, if one user story is "Implement fingerprint recognition," sub-tasks may include "research fingerprint recognition algorithms," "acquire necessary hardware," and "develop and test fingerprint recognition module."

- **Sprint Goal:**

Define a clear sprint goal that articulates what the team intends to achieve during the sprint. It should align with the product backlog items selected for the sprint.

- **Sprint Backlog:**

Create a sprint backlog that includes the user stories and tasks selected for the sprint, along with their estimates.

6.3 Sprint Delivery Schedule:

- **Week 1: Establish the Core**

- Setup the basic blockchain infrastructure.
- Implement the user registration and authentication system.
- Develop security transaction recording feature.

➤ **Week 2: Expand and Enhance**

- Extend transaction recording to support more transaction types.
- Add basic real-time updates for transaction status.
- Enhance user authentication with multi-factor authentication.
- Begin developing a user dashboard.

➤ **Week 3: Finalize and Prepare**

- Complete the user dashboard with additional features.
- Perform compliance checks.
- Conduct basic testing and issues solution.
- Create essential user support resources.

7. CODING AND SOLUTIONS:

7.1 Feature 1:

Smartcontract(Solidity)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract BallotBox {

 // Define the owner of the contract (election authority).

 address public owner;

 // Define the structure of a voter.

 struct Voter {

 bytes32 biometricData; // Encrypted biometric data

 bool hasVoted; // Indicates if the voter has cast a vote

 }

 // Define the structure of a candidate.

 struct Candidate {

 string name;

 uint256 voteCount;

 }

 // Define the election parameters.

 string public electionName;

 uint256 public registrationDeadline;

```

uint256 public votingDeadline;

// Store the list of candidates.
Candidate[] public candidates;

// Store the mapping of voters.
mapping(address => Voter) public voters;

// Event to announce when a vote is cast.
event VoteCast(address indexed voter, uint256 candidateIndex);

// Modifiers for access control.
modifier onlyOwner() {
    require(msg.sender == owner, "Only the owner can call this
function.");
    _;
}
modifier canVote() {
    require(block.timestamp < votingDeadline, "Voting has ended.");
    require(block.timestamp < registrationDeadline, "Registration has
ended.");
    require(!voters[msg.sender].hasVoted, "You have already voted.");
    _;
}

// Constructor to initialize the contract.
constructor(
    string memory _electionName,
    uint256 _registrationDeadline,
    uint256 _votingDeadline,
    string[] memory _candidateNames
) {
    owner = msg.sender;
    electionName = _electionName;
    registrationDeadline = _registrationDeadline;
    votingDeadline = _votingDeadline;

    // Initialize the list of candidates.
    for (uint256 i = 0; i < _candidateNames.length; i++) {
        candidates.push(Candidate({
            name: _candidateNames[i],
            voteCount: 0

```



```

        }));
    }
}

// Function to register a voter and store their encrypted biometric data.
function registerVoter(bytes32 _encryptedBiometricData) public
canVote {
    voters[msg.sender] = Voter({
        biometricData: _encryptedBiometricData,
        hasVoted: false
    });
}

// Function to cast a vote for a candidate.
function castVote(uint256 _candidateIndex) public canVote {
    require(_candidateIndex < candidates.length, "Invalid candidate
index.");
    require(voters[msg.sender].biometricData != 0, "You must register
first.");

    // Mark the voter as having voted.
    voters[msg.sender].hasVoted = true;

    // Increment the candidate's vote count.
    candidates[_candidateIndex].voteCount++;

    // Emit a VoteCast event.
    emit VoteCast(msg.sender, _candidateIndex);
}
}

```

Solidity is a programming language used to develop smart contracts on blockchain platforms like Ethereum. "In the emerging landscape of biometric security and blockchain technology, Solidity plays a pivotal role in creating a secure and decentralized ecosystem for biometric data management and identity verification. By harnessing the power of the Ethereum blockchain and writing smart contracts in Solidity, we can establish a tamper-resistant and transparent infrastructure for biometric

security. Smart contracts can securely store biometric data hashes, ensuring that sensitive information remains private while allowing for efficient verification processes. The decentralized nature of blockchain technology, along with the capabilities of Solidity, provides a foundation for a trustless and auditable biometric security system, reducing the risk of data breaches and identity fraud while maintaining the privacy and consent of users."

7.2 Feature 2:

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "../connector";

function Home() {
  const [Wallet, setWallet] = useState("");

  const [CandidateIndex, setCandidateIndex] = useState("");

  const [VoterData, setVoterData] = useState("");

  const [CandidateIndexed, setCandidateIndexed] = useState("");

  const [CandidatesData, setCandidatesData] = useState("");

  const [RegDeadline, setRegDeadline] = useState("");

  const [VoteDeadline, setVoteDeadline] = useState("");

  const [Election, setElection] = useState("");

  const handleCandidateIndex = (e) => {
    setCandidateIndex(e.target.value)
  }
  const handleCastVote = async () => {
    try {
      let tx = await contract.castVote(CandidateIndex.toString())
```

```

    let wait = await tx.wait()
    console.log(wait);
    alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}
const handleVoterBiometricData = (e) => {
  setVoterData(e.target.value)
}

const handleRegisterVoter = async () => {
  try {
    let tx = await contract.registerVoter(VoterData)
    let wait = await tx.wait()
    console.log(wait)
    alert(wait.transactionHash)

  } catch (error) {
    alert(error)
  }
}

const handleCandidateIndexs = (e) => {
  setCandidateIndexed(e.target.value)
}

const handleCandidate = async () => {
  try {
    let tx = await contract.candidates(CandidateIndexed.toString())
    setCandidatesData(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleRegdeadline = async () => {
  try {

```

```

    let tx = await contract.registrationDeadline()
    setRegDeadline(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}
const handleVoteDeadline = async () => {
  try {
    let tx = await contract.votingDeadline()
    setVoteDeadline(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error)

{
  alert(error)
}
}

const handleElecName = async () => {
  try {
    let tx = await contract.electionName()
    setElection(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleWallet = async () => {
  if (!window.ethereum) {
    return alert('please install metamask');
  }

  const addr = await window.ethereum.request({
    method: 'eth_requestAccounts',
  });

```

```

        setWallet(addr[0])

    }

    return (
    <div>
        <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Ballot
        Box on Blockchain</h1>
        {!Wallet ?

            <Button onClick={handleWallet} style={{ marginTop: "30px",
            marginBottom: "50px" }}>Connect Wallet </Button>
            :
            <p style={{ width: "250px", height: "50px", margin: "auto",
            marginBottom: "50px", border: '2px solid #2096f3' }}>
            {
                Wallet.slice(0, 6)}....{Wallet.slice(-6)}</p>
            }
        <Container>
            <Row>
                <Col style={{ marginRight:"100px" }}>
                    <div>

                        <input style={{ marginTop: "10px", borderRadius: "5px" }}
                        onChange={handleCandidateIndex} type="number"
                        placeholder="Candidate Index" value={CandidateIndex} /> <br />
                        <Button onClick={handleCastVote} style={{ marginTop: "10px" }}
                        variant="primary">Cast Vote</Button>

                    </div>
                </Col>

                <Col style={{ marginRight: "100px" }}>
                    <div>
                        <input style={{ marginTop: "10px", borderRadius: "5px" }}
                        onChange={handleVoterBiometricData} type="string"
                        placeholder="Vote Encrypted data" value={VoterData} /> <br />
                        <Button onClick={handleRegisterVoter} style={{ marginTop:

```

```

"10px" }} variant="primary">Register Voter</Button>

</div>
</Col>

</Row>
<Row style={{ marginTop: "100px" }}>
  <Col style={{ marginRight: "100px" }}>
    <div>
      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleCandidateIndexs} type="number"
placeholder="Candidate Index" value={CandidateIndexed} /> <br />
      <Button onClick={handleCandidate} style={{ marginTop:
"10px" }} variant="primary"> Get transaction Count</Button>
      {CandidatesData ? CandidatesData?.map(e =>
<p>{e.toString()}</p>) : <p></p>
      }
    </div>
  </Col>

  <Col style={{ marginRight: "100px" }}>
    <div>
      <Button onClick={handleRegdeadline} style={{ marginTop:
"10px" }} variant="primary">Registration deadline</Button>
      {RegDeadline ? <p>{RegDeadline.toString()}</p> :
<p></p>}
    </div>
  </Col>
</Row>
<Row style={{ marginTop: "50px" }}>
  <Col style={{ marginRight: "100px" }}>
    <div>
      <Button onClick={handleVoteDeadline} style={{
marginTop: "10px" }} variant="primary">Voting deadline</Button>
      {VoteDeadlne ? <p>{VoteDeadlne.toString()}</p> :
<p></p>}
    </div>
  </Col>
  <Col style={{ marginRight: "100px" }}>
    <div>

```

```

        <Button onClick={handleElecName} style={{ marginTop:
"10px" }} variant="primary">Election Name</Button>
        {Election ? <p>{Election.toString()}</p> : <p></p>}

    </div>
  </Col>
</Row>
</Container>
</div>
)
}
export default Home;

```

7.3 Database schema:

Contract ABI (Application Binary Interface):

The variable holds the ABI of an Ethereum smart contract. ABIs' are essential for encoding and decoding function calls and data when interacting with the Ethereum blockchain.

MetaMask Check:

The code first checks whether the MetaMask wallet extension is installed in the user's browser. If MetaMask is not detected, it displays an alert notifying the user that MetaMask is not found and provides a link to download it.

Ethers.js Configuration:

It imports the ethers ballot, which is a popular library for Ethereum development. It creates a provider using Web3 Provider, which connects to the user's MetaMask wallet and provides access to Ethereum. It creates a signer to interact with The Ethereum blockchain on behalf of the user. It defines an Ethereum contract address and sets up the contract object using ethers. Contract, allowing the JavaScript code to interact with the contract's functions. In summary, this code is used for interacting with an Ethereum smart

contract through MetaMask and ethers.js. It configures the necessary Ethereum provider and signer for communication with the blockchain and sets up a contract object for executing functions and fetching data from the specified contract address using the provided ABI.

8. PERFORMANCE TESTING

8.1 Performance Metrics:

Performance testing for a biometric security system in a voting platform is crucial to ensure that the system can handle the expected load, provide timely responses, and maintain security and accuracy during elections. Here are steps and considerations for conducting performance testing in such a system:

➤ Define Performance Objectives:

Clearly define your performance objectives. Determine key performance indicators (KPIs) such as response time, throughput, concurrent users, and resource utilization.

➤ Simulate Realistic Scenarios:

Create realistic scenarios that mimic the expected usage of the biometric security system during an election. Consider various types of biometric data (e.g., fingerprints, facial recognition) and voter load.

➤ Load Testing:

Test the system's ability to handle expected loads. This includes determining the system's capacity by gradually increasing the number of concurrent users and measuring response times and resource usage.

➤ Stress Testing:

Conduct stress tests to determine the system's breaking point. Exceed the expected load to identify the system's limitations and

weaknesses.

➤ **Scalability Testing:**

Verify the system's scalability by adding more resources, such as servers or processing power, and assessing how it responds to increased demand.

➤ **Resource Utilization:**

Monitor the utilization of system resources, including CPU, memory, storage, and network bandwidth. Ensure that the system operates efficiently.

➤ **Data Volume Testing:**

Test the system with different volumes of biometric data and assess its performance in managing, storing, and retrieving data effectively.

➤ **Response Time Testing:**

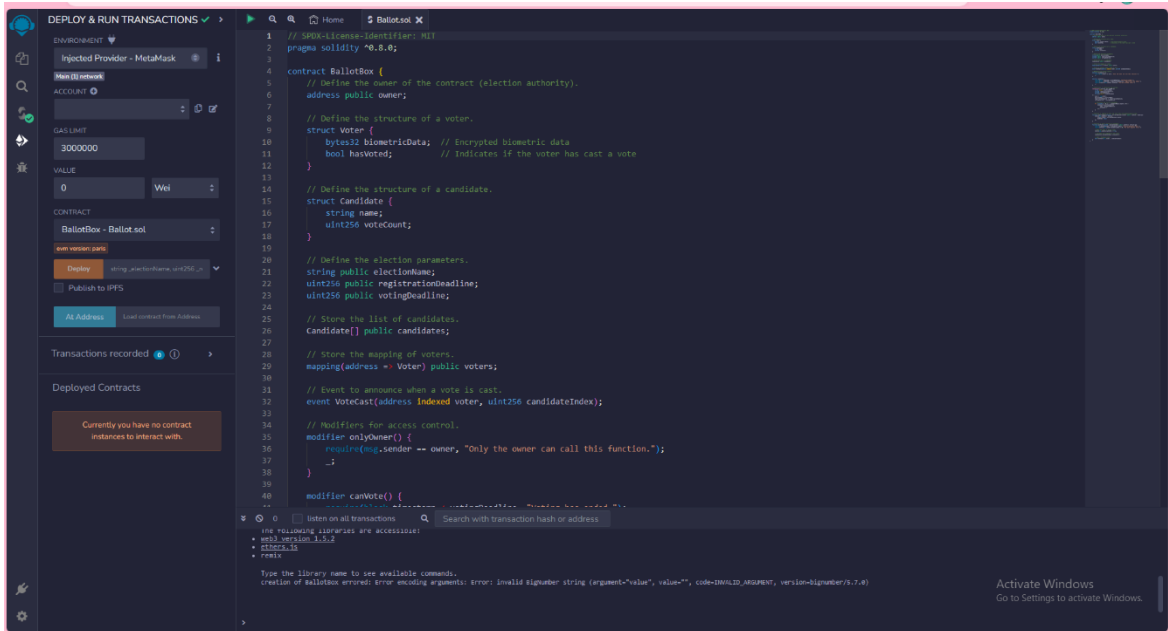
Measure the response times for various system interactions, such as user authentication, voter registration, and vote submission, to ensure they meet acceptable thresholds.

➤ **Security Testing:**

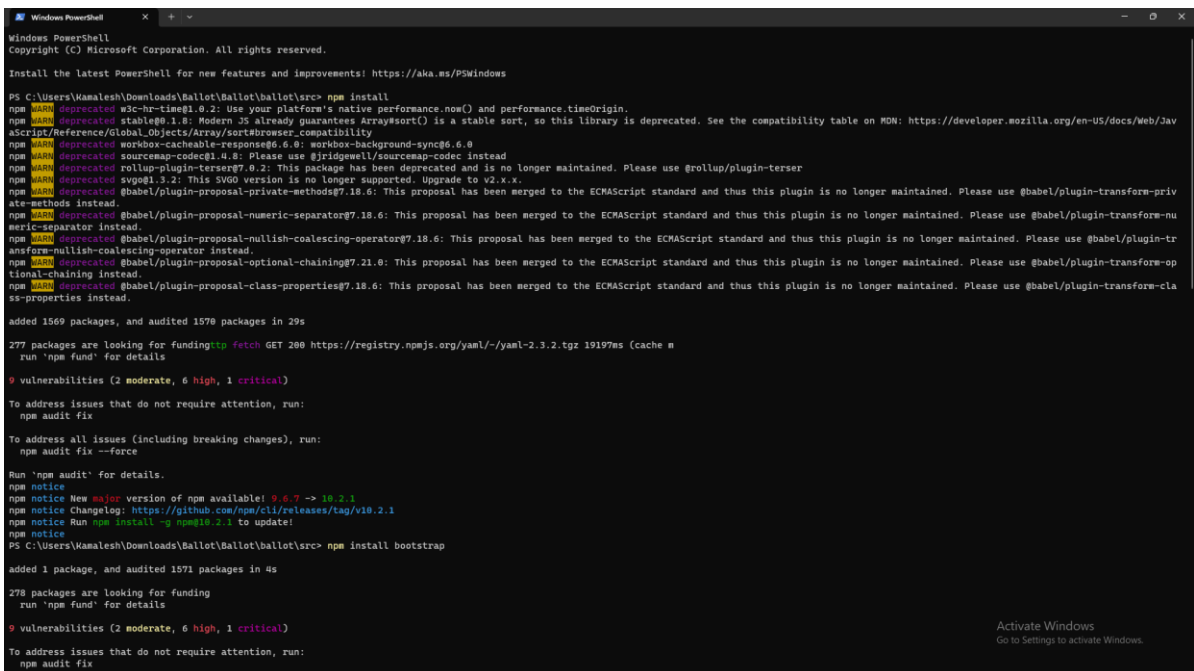
Assess the system's security under high loads, including the ability to resist DDoS attacks, protect sensitive data, and maintain the integrity of biometric information.

9. RESULTS

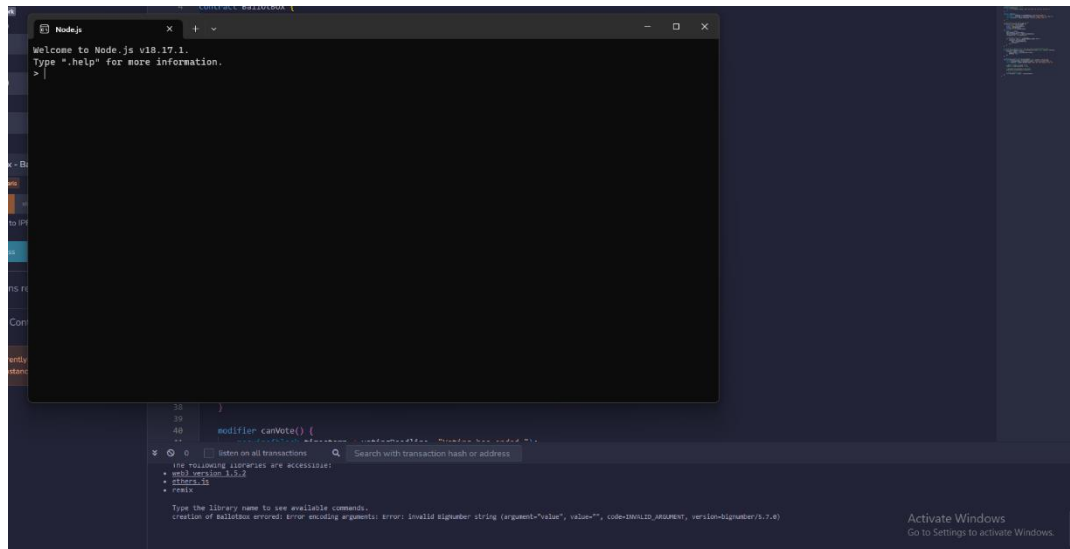
9.1 Output Screenshots



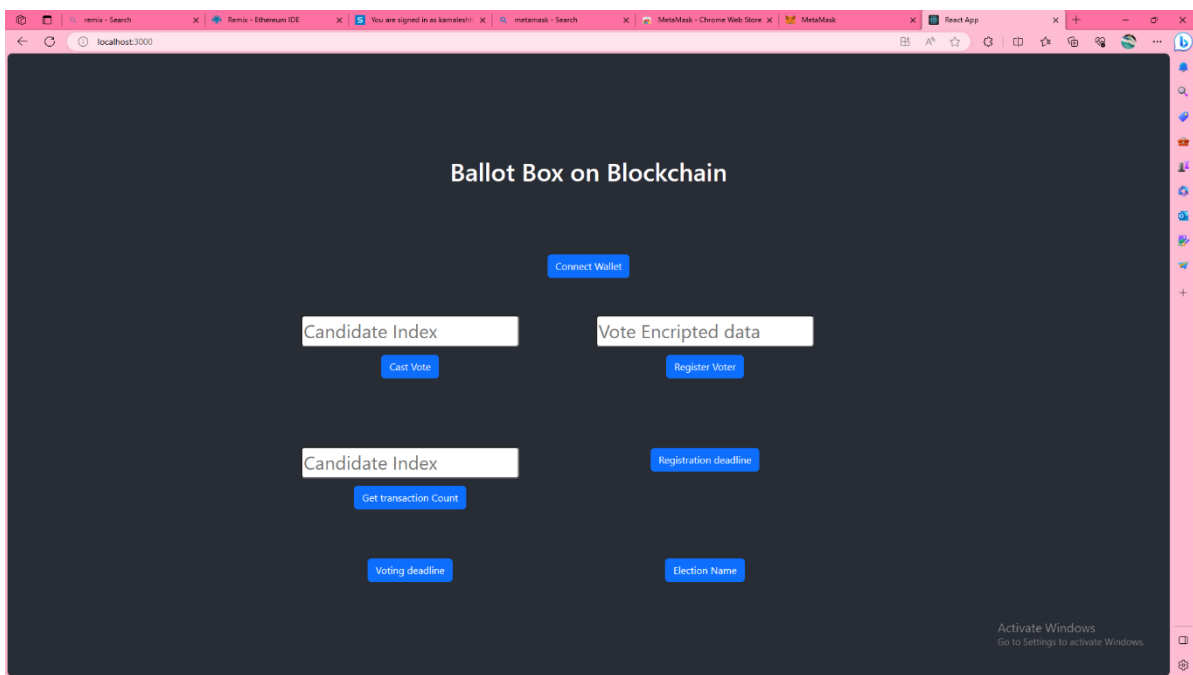
CREATING A SMART CONTRACT



INSTALLING DEPENDENCIES



HOSTING THE SITE LOCALLY



OUTPUT SCREEN

10. ADVANTAGES AND DISADVANTAGES

Advantages:

Enhanced Security: Biometric authentication systems use unique physical or behavioral characteristics, such as fingerprints, iris scans, or facial recognition, making it extremely difficult for unauthorized individuals to impersonate voters.

Reduced Voter Fraud: With biometrics, it becomes much more challenging for people to vote multiple times under different identities or for ineligible individuals to cast their votes. This can significantly reduce voter fraud and ensure a fairer electoral process.

Accurate Voter Identification: Biometric systems can accurately verify a voter's identity, ensuring that the person casting the vote is indeed the registered voter. This helps maintain the integrity of the electoral process.

Reduced Administrative Costs: Biometric systems can lead to cost savings by automating the voter identification process and reducing the need for paper-based records and manual data entry. This can make elections more efficient and cost-effective.

Prevention of Double Voting: Biometric data can be cross-referenced with a central voter database, making it difficult for a person to vote in multiple locations, thereby preventing double voting.

Tamper Resistance: Biometric data is difficult to forge or tamper with, which makes the system more resistant to hacking and manipulation.

Increased Confidence in Elections: A secure biometric voting system can boost public confidence in the electoral process by ensuring that votes are accurately and fairly counted, reducing concerns about election fraud and irregularities.

Scalability: Biometric systems can be scaled up or down to accommodate the needs of different-sized elections, from local to national levels.

Disadvantages:

Privacy Concerns: Collecting and storing biometric data raises significant privacy concerns. Citizens may be uncomfortable with their biometric information being stored in government databases, as it can potentially be misused or breached.

Data Security: Biometric databases are attractive targets for cyberattacks. If the system is not adequately secured, there is a risk of data breaches and unauthorized access, potentially compromising the integrity of the electoral process.

Accuracy and Reliability: Biometric systems are not infallible. False positives and false negatives can occur, leading to the incorrect identification of voters. This can disenfranchise eligible voters or allow ineligible individuals to cast their votes.

Technical Challenges: Biometric systems may face technical challenges, such as malfunctioning hardware, connectivity issues, or software glitches, which can disrupt the voting process.

Costs: Implementing biometric systems can be expensive, requiring the purchase of specialized hardware and software, as well as ongoing maintenance and support costs. Smaller or financially constrained jurisdictions may find it challenging to adopt such systems.

Inclusivity: Not all citizens may have suitable biometric characteristics for authentication (e.g., certain disabilities or medical conditions). This could potentially exclude certain segments of the population from voting.

11. CONCLUSION

A biometric security system for a voting platform holds great promise for enhancing the integrity and trustworthiness of the electoral process. Biometric authentication can simplify the voting process, making it more accessible and efficient for eligible voters, reducing wait times and administrative burdens. By reducing the risk of voter fraud, a biometric system can enhance the overall credibility of the electoral system and protect the sanctity of the democratic process. Biometric data can prevent individuals from voting multiple times, further safeguarding the fairness of elections. Biometric systems must be designed with the utmost security to prevent unauthorized access, hacking, or tampering. Legal frameworks and ethical guidelines must be established to regulate the collection, storage, and use of biometric data for voting. Ensuring that all eligible voters can participate, including those who may not have access to the required biometric devices, is crucial.

12. FUTURESCOPE

The future scope for biometric security systems in voting platforms is promising, with advancements in technology and increasing concerns about election security and integrity. Here are some key areas where biometric security systems in voting platforms are likely to see growth and development:

- **Wider Adoption:**

The adoption of biometric security systems in voting platforms is expected to increase as more countries and regions seek secure and efficient ways to conduct elections. Biometrics can help combat voter

fraud and enhance the overall integrity of the electoral process.

- **Remote Voting and Accessibility:**

Biometric systems will play a crucial role in enabling remote and online voting, making it more accessible to citizens who may have difficulty physically attending polling stations. This could lead to increased voter participation.

- **Blockchain Integration:**

Integration with blockchain technology can enhance the security and transparency of biometric voting systems. Blockchain can provide a tamper-proof ledger of votes, ensuring the accuracy and auditability of election results.

- **Continuous Improvement:**

Advancements in biometric recognition technologies, such as facial recognition, fingerprint scanning, and iris recognition, will continue to enhance accuracy and speed, making the voting process more efficient and user-friendly.

13. APPENDIX

Source code:

Ballot.Sol(SmartContract)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract BallotBox {
    // Define the owner of the contract (election authority).
    address public owner;
    // Define the structure of a voter.
    struct Voter {
        bytes32 biometricData; // Encrypted biometric data
        bool hasVoted;        // Indicates if the voter has cast a vote
    }
    // Define the structure of a candidate.
    struct Candidate {
        string name;
        uint256 voteCount;
    }
    // Define the election parameters.
    string public electionName;
    uint256 public registrationDeadline;
    uint256 public votingDeadline;
    // Store the list of candidates.
    Candidate[] public candidates;
    // Store the mapping of voters.
    mapping(address => Voter) public voters;
    // Event to announce when a vote is cast.
    event VoteCast(address indexed voter, uint256 candidateIndex);
    // Modifiers for access control.
```

```

modifier onlyOwner() {
    require(msg.sender == owner, "Only the owner can call this
function.");
    _;
}
modifier canVote() {
    require(block.timestamp < votingDeadline, "Voting has ended.");
    require(block.timestamp < registrationDeadline, "Registration has
ended.");
    require(!voters[msg.sender].hasVoted, "You have already voted.");
    _;
}
// Constructor to initialize the contract.
constructor(
    string memory _electionName,
    uint256 _registrationDeadline,
    uint256 _votingDeadline,
    string[] memory _candidateNames
) {
    owner = msg.sender;
    electionName = _electionName;
    registrationDeadline = _registrationDeadline;
    votingDeadline = _votingDeadline;
    // Initialize the list of candidates.
    for (uint256 i = 0; i < _candidateNames.length; i++) {
        candidates.push(Candidate({
            name: _candidateNames[i],
            voteCount: 0
        }));
    }
}

```

```

    }
    // Function to register a voter and store their encrypted biometric data.
    function registerVoter(bytes32 _encryptedBiometricData) public
    canVote {
        voters[msg.sender] = Voter({
            biometricData: _encryptedBiometricData,
            hasVoted: false
        });
    }
    // Function to cast a vote for a candidate.
    function castVote(uint256 _candidateIndex) public canVote {
        require(_candidateIndex < candidates.length, "Invalid candidate
index.");
        require(voters[msg.sender].biometricData != 0, "You must register
first.");
        // Mark the voter as having voted.
        voters[msg.sender].hasVoted = true;
        // Increment the candidate's vote count.
        candidates[_candidateIndex].voteCount++;
        // Emit a VoteCast event.
        emit VoteCast(msg.sender, _candidateIndex);
    }
}

```

Connector.js :

```

const { ethers } = require("ethers");
const abi = [
    {
        "inputs": [
            {
                "internalType": "string",

```

```

    "name": "_electionName",
    "type": "string"
  },
  {
    "internalType": "uint256",
    "name": "_registrationDeadline",
    "type": "uint256"
  },
  {
    "internalType": "uint256",
    "name": "_votingDeadline",
    "type": "uint256"
  },
  {
    "internalType": "string[]",
    "name": "_candidateNames",
    "type": "string[]"
  }
],
"stateMutability": "nonpayable",
"type": "constructor"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "internalType": "address",
      "name": "voter",

```

```

    "type": "address"
  },
  {
    "indexed": false,
    "internalType": "uint256",
    "name": "candidateIndex",
    "type": "uint256"
  }
],
"name": "VoteCast",
"type": "event"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "name": "candidates",
  "outputs": [
    {
      "internalType": "string",
      "name": "name",
      "type": "string"
    }
  ],
  {
    "internalType": "uint256",

```

```

    "name": "voteCount",
    "type": "uint256"
  },
  {
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_candidateIndex",
        "type": "uint256"
      }
    ],
    "name": "castVote",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {
    "inputs": [],
    "name": "electionName",
    "outputs": [
      {
        "internalType": "string",
        "name": "",
        "type": "string"
      }
    ]
  }

```

```

],
"stateMutability": "view",
"type": "function"
},
{
  "inputs": [],
  "name": "owner",
  "outputs": [
    {
      "internalType": "address",
      "name": "",
      "type": "address"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "bytes32",
      "name": "_encryptedBiometricData",
      "type": "bytes32"
    }
  ],
  "name": "registerVoter",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
}

```

```

},
{
  "inputs": [],
  "name": "registrationDeadline",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "",
      "type": "address"
    }
  ],
  "name": "voters",
  "outputs": [
    {
      "internalType": "bytes32",
      "name": "biometricData",
      "type": "bytes32"
    }
  ],

```



```

    {
      "internalType": "bool",
      "name": "hasVoted",
      "type": "bool"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [],
  "name": "votingDeadline",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
}
]
if (!window.ethereum) {
  alert('Meta Mask Not Found')
  window.open("https://metamask.io/download/")
}

export const provider = new
ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();

```

```
export const address =
"0xD52c1b477B072d5A9cA2f73690Ba14334d0D8Ce0"
export const contract = new ethers.Contract(address, abi, signer)
```

Home.js:

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "../connector";

function Home() {
  const [Wallet, setWallet] = useState("")
  const [CandidateIndex, setCandidateIndex] = useState("");
  const [VoterData, setVoterData] = useState("");
  const [CandidateIndexed, setCandidateIndexed] = useState("");
  const [CandidatesData, setCandidatesData] = useState("");
  const [RegDeadline, setRegDeadline] = useState("");
  const [VoteDeadline, setVoteDeadline] = useState("");
  const [Election, setElection] = useState("");
  const handleCandidateIndex = (e) => {
    setCandidateIndex(e.target.value)
  }
  const handleCastVote = async () => {
    try {
      let tx = await contract.castVote(CandidateIndex.toString())
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }
}
```

```

    }
    const handleVoterBiometricData = (e) => {
        setVoterData(e.target.value)
    }
    const handleRegisterVoter = async () => {
        try {
            let tx = await contract.registerVoter(VoterData)
            let wait = await tx.wait()
            console.log(wait)
            alert(wait.transactionHash)
        } catch (error) {
            alert(error)
        }
    }
    const handleCandidateIndexs = (e) => {
        setCandidateIndexed(e.target.value)
    }
    const handleCandidate = async () => {
        try {
            let tx = await contract.candidates(CandidateIndexed.toString())
            setCandidatesData(tx)
            console.log(tx);
            // alert(wait.transactionHash)
        } catch (error) {
            alert(error)
        }
    }

    const handleRegdeadline = async () => {
        try {

```

```

    let tx = await contract.registrationDeadline()
    setRegDeadline(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleVoteDeadline = async () => {
  try {
    let tx = await contract.votingDeadline()
    setVoteDeadline(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleElecName = async () => {
  try {
    let tx = await contract.electionName()
    setElection(tx)
    console.log(tx);
    // alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleWallet = async () => {

```

```

    if (!window.ethereum) {
      return alert('please install metamask');
    }
    const addr = await window.ethereum.request({
      method: 'eth_requestAccounts',
    });
    setWallet(addr[0])
  }
  return (
    <div>
      <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Ballot
      Box on Blockchain</h1>
      {!Wallet ?
        <Button onClick={handleWallet} style={{ marginTop: "30px",
        marginBottom: "50px" }}>Connect Wallet </Button>
        <p style={{ width: "250px", height: "50px", margin: "auto",
        marginBottom: "50px", border: '2px solid #2096f3' }}>{Wallet.slice(0,
        6)}....{Wallet.slice(-6)}</p>
        }
      <Container>
        <Row>
          <Col style={{ marginRight: "100px" }}>
            <div>
              <input style={{ marginTop: "10px", borderRadius: "5px" }}
              onChange={handleCandidateIndex} type="number"
              placeholder="Candidate Index" value={CandidateIndex} /> <br />
              <Button onClick={handleCastVote} style={{ marginTop: "10px" }}
              variant="primary">Cast Vote</Button>
            </div>
          </Col>
          <Col style={{ marginRight: "100px" }}>

```

```

<div>
    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleVoterBiometricData} type="string"
placeholder="Vote Encrypted data" value={VoterData} /> <br />
    <Button onClick={handleRegisterVoter} style={{ marginTop:
"10px" }} variant="primary">Register Voter</Button>
</div>
</Col>
</Row>
<Row style={{ marginTop: "100px" }}>
    <Col style={{ marginRight: "100px" }}>
        <div>
            <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleCandidateIndexs} type="number"
placeholder="Candidate Index" value={CandidateIndexed} /> <br />
            <Button onClick={handleCandidate} style={{ marginTop: "10px" }}
variant="primary"> Get transaction Count</Button>
            {CandidatesData ? CandidatesData?.map(e =>
<p>{e.toString()}</p>) : <p></p>
            }
        </div>
    </Col>
    <Col style={{ marginRight: "100px" }}>
        <div>
            <Button onClick={handleRegdeadline} style={{ marginTop:
"10px" }} variant="primary">Registration deadline</Button>
            {RegDeadline ? <p>{RegDeadline.toString()}</p> :
<p></p>}
        </div>
    </Col>
</Row>
<Row style={{ marginTop: "50px" }}>

```

```

    <Col style={{ marginRight: "100px" }}>
      <div>
        <Button onClick={handleVoteDeadline} style={{
marginTop: "10px" }} variant="primary">Voting deadline</Button>
        {VoteDeadline ? <p>{VoteDeadline.toString()}</p> :
<p></p>}
      </div>
    </Col>
    <Col style={{ marginRight: "100px" }}>
      <div>
        <Button onClick={handleElecName} style={{ marginTop:
"10px" }} variant="primary">Election Name</Button>
        {Election ? <p>{Election.toString()}</p> : <p></p>}

      </div>
    </Col>
  </Row>
</Container>
</div>
)
}

```

export default Home;

App.js:

```

import './App.css';
import Home from './Page/Home'
function App() {
  return (
    <div className="App">
      <header className="App-header">

```

```

    <Home />
  </header>
</div>
);
}
export default App;
App.css:
.App {
  text-align: center;
}
.App-logo {
  height: 40vmin;
  pointer-events: none;
}
@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}
.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

```



```

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

Index.js:

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

```

Index.css:

```
body {  
    margin: 0;  
    font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',  
    'Oxygen',  
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',  
    sans-serif;  
    -webkit-font-smoothing: antialiased;  
    -moz-osx-font-smoothing: grayscale;  
}  
code {  
    font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',  
    monospace;
```

GITHUB LINK AND PROJECT DEMO LINK:**GitHub Link**

<https://github.com/deepa6369729134/NM-BLOCKCHAIN-BIOMETRIC-SECURITY-SYSTEM-FOR-VOTING-PLATFORM.git>

Project demo link

<https://drive.google.com/file/d/1vj6np4cTXgPCpsgPs0DuB8Gga1sKwIBC/view?usp=sharing>