PROJECT REPORT

ELECTRONIC VOTING SYSTEM

Name	Team Id
K.Kabilan	NM2023TMID01479
B.Praveen	
E.Elanchezhiyan	
S.Jayanthan	

INDEX

ELECTRONIC VOTING SYSTEM

1. INTRODUCTION

Project Overview

An electronic voting system on a blockchain is a concept that aims to enhance the security, transparency, and trustworthiness of the voting process. Traditional voting systems are prone to various issues such as fraud, manipulation, and lack of transparency. Blockchain technology, with its decentralized and immutable ledger, offers several advantages for electronic voting systems . blockchain-based voting system, each vote is recorded as a cryptographically secure transaction on a distributed ledger. The immutable nature of the blockchain ensures that once a vote is cast, it cannot be altered or deleted, providing a verifiable and tamper-resistant record of the election results. Transparency is inherent in blockchain technology, as all transactions are visible to network participants, allowing for independent verification of the vote tally. While the benefits are significant, challenges exist in implementing such systems. Ensuring secure and private identity verification, maintaining voter confidentiality, addressing scalability issues, and designing user-friendly interfaces are key considerations. Moreover, adherence to local regulatory and legal frameworks is crucial.

Purpose

This project has a few main goals. First, it wants to make voting safer by using blockchain technology. Traditional voting systems can sometimes have problems with security, like fraud. This project aims to prevent that. And it wants to make voting more transparent. In normal elections, it can be hard to check if everything is fair. But with blockchain, everyone can see what's happening, which makes it more trustworthy. It aims to create trust in the voting process. By using blockchain's unique features, like not being able to change or delete votes, it ensures that election results can't be tampered with. This makes people trust the results. So this tries to solve the problems in traditional voting systems. These can include things like fraud and difficulties in counting votes accurately. Also, the project recognizes that there are challenges in making this technology work. For example, they need to make sure that verifying people's identities is safe and private. They also want to keep how people vote a secret and make it easy for everyone to use the system. And they must follow the laws and rules in the places where they want to use this technology. In summary, this project is all about using blockchain to make voting safer, fairer, and more trustworthy. It wants to solve the problems of traditional voting, deal with the challenges, and make sure it's legal and secure. Ultimately, it aims to make our elections stronger and more reliable.

2. EXISTING SYSTEM

Existing Problem

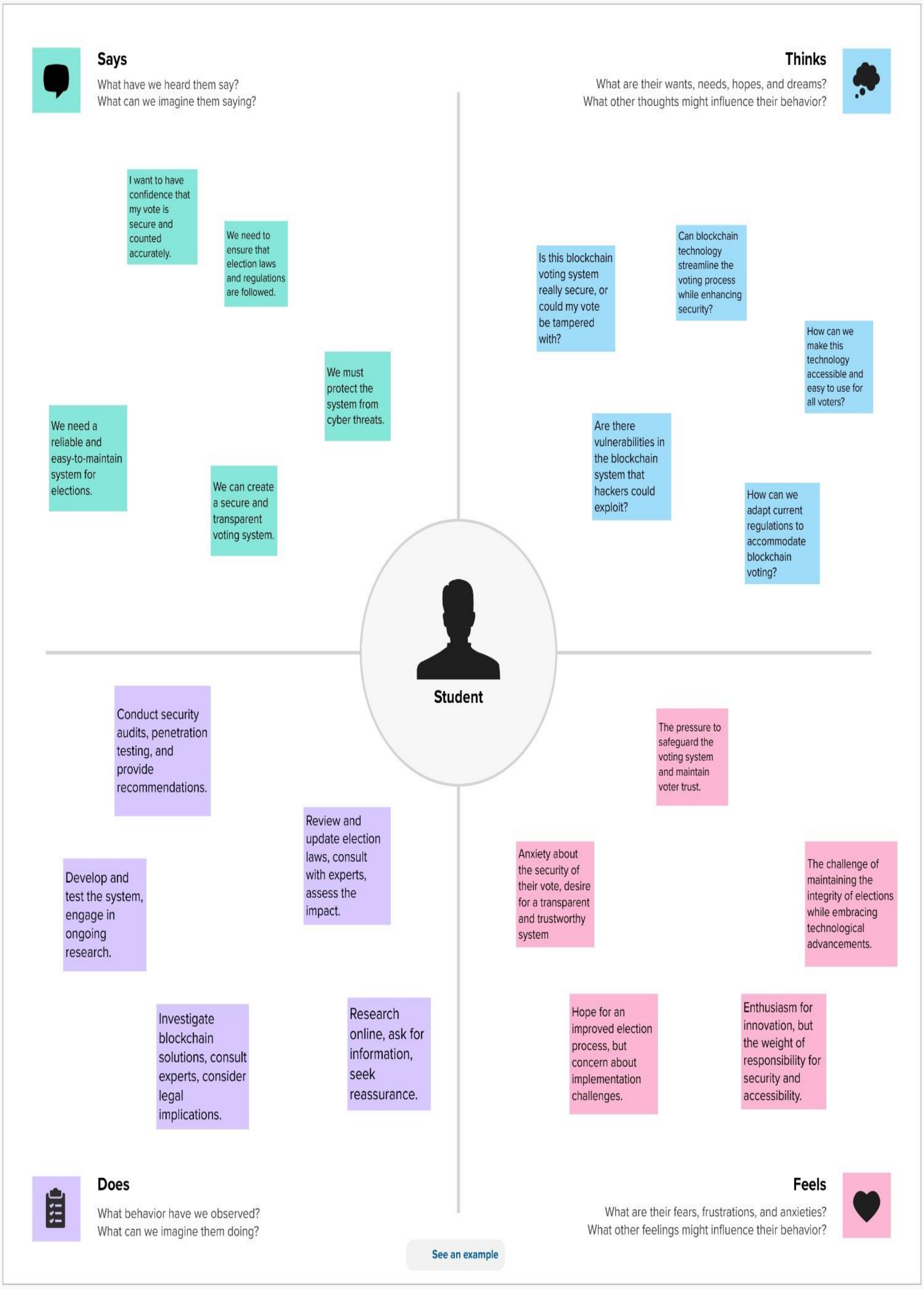
Currently, traditional voting systems have various vulnerabilities. These systems are sometimes susceptible to fraudulent activities, and the final results can be manipulated. Additionally, it is not always easy to understand how votes are counted, leading to doubts about the fairness of the process. Both paper-based voting and certain electronic voting systems without blockchain struggle to provide the necessary security and transparency for trustworthy elections.

Problem Statement Definition

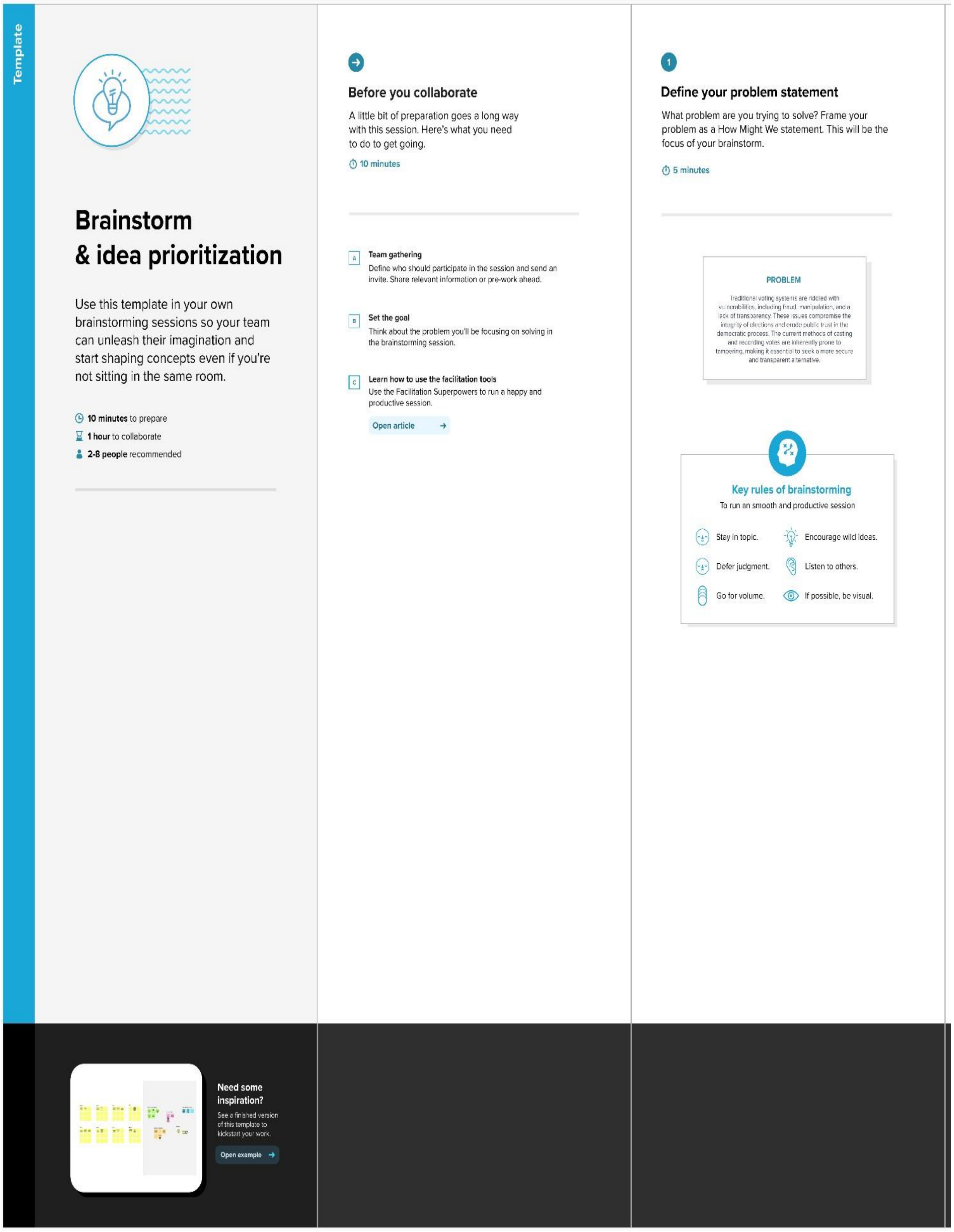
The problem statement highlights the many problems with traditional voting systems and suggests that we need to rethink how we conduct elections. These systems have long had issues that hurt the heart of our democratic processes. Things like fraud, cheating with votes, and not being clear about how we count votes have made people question if elections are really fair. In addition, it's hard to keep votes safe and protect them from people who shouldn't access them. Making sure that nobody knows who you voted for is essential in elections, and this has been hard to achieve, especially when people's personal information gets leaked. Also, we can't easily check if election results are correct. To solve these big problems, we need to create an electronic voting system that uses blockchain technology. Blockchain makes the system more secure, transparent, and tamper-proof. But it also needs to follow the rules and laws of each place to make sure it's not just safe but also legal. This can bring a new era of voting and help people trust elections more.

3. IDEATION AND PROPOSED SOLUTION

Empathy Map Canvas


Says
What have we heard them say?
What can we imagine them saying?
I want to have confidence that my vote is secure and counted accurately.
We need to ensure that election laws and regulations are followed.
We must protect the system from cyber threats.
We need a reliable and easy-to-maintain system for elections.
We can create a secure and transparent voting system.
Thinks
What are their wants, needs, hopes, and dreams?
What other thoughts might influence their behavior?
Is this blockchain voting system really secure, or could my vote be tampered with?
Can blockchain technology streamline the voting process while enhancing security?
How can we make this technology accessible and easy to use for all voters?
Are there vulnerabilities in the blockchain system that hackers could exploit?
How can we adapt current regulations to accommodate blockchain voting?
Student
Conduct security audits, penetration testing, and provide recommendations.
Review and update election laws, consult with experts, assess the impact.
Develop and test the system, engage in ongoing research.
Investigate blockchain solutions, consult experts, consider legal implications.
Research online, ask for information, seek reassurance.
The pressure to safeguard the voting system and maintain voter trust.
Anxiety about the security of their vote, desire for a transparent and trustworthy system
The challenge of maintaining the integrity of elections while embracing technological advancements.
Hope for an improved election process, but concern about implementation challenges.
Enthusiasm for innovation, but the weight of responsibility for security and accessibility.
Does
What behavior have we observed?
What can we imagine them doing?
Feels
What are their fears, frustrations, and anxieties?
What other feelings might influence their behavior?
See an example

Ideation and Brainstorming



Template

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- **10 minutes** to prepare
- **1 hour** to collaborate
- **2-8 people** recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

1 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.

5 minutes

PROBLEM

Traditional voting systems are riddled with vulnerabilities, including fraud, manipulation, and a lack of transparency. These issues compromise the integrity of elections and erode public trust in the democratic process. The current methods of casting and recording votes are inherently prone to tampering, making it essential to seek a more secure and transparent alternative.

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example →

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 and 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.

Common doubt about security

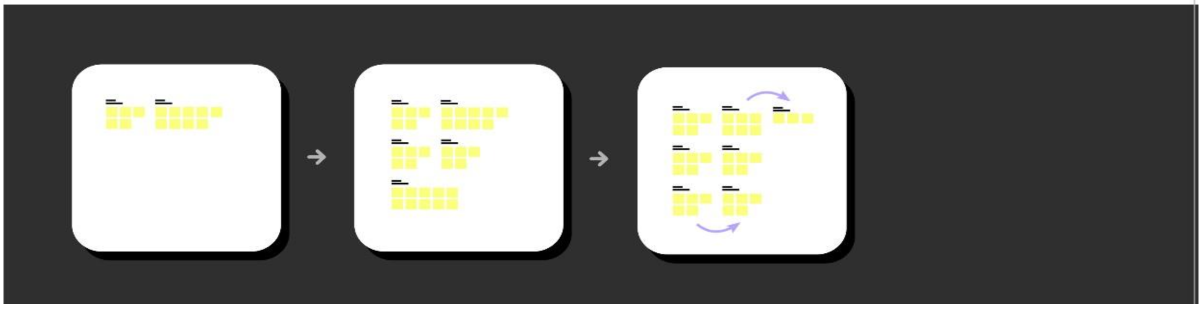
- Do people believe all process are transparent
- Can this be tamperd
- Can this be fraud proof

Is this useable for everyone

- Can every one use this technology
- Can every individual get access to this
- Can this be used in all places
- Do we need this

Can resources be obtained

- Do we need more resources
- Can we scale this to a large group of individuals
- Is this an open source product
- Is it easy to use



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!

K.S.Chandru

Do we need this

Can every one use this technology

Do we need more resources

L.Dhinesh

Dose this project be an user friendly interface

Can every individual get access to this

Can we scale this to a large group of individuals

M.Karthik

Do people believe all process are transparent

Can this be fraud proof

Can this be used in all places

R.Shathiyapriyan

Is this an open source product

Can this be tamperd

Is it easy to use

4

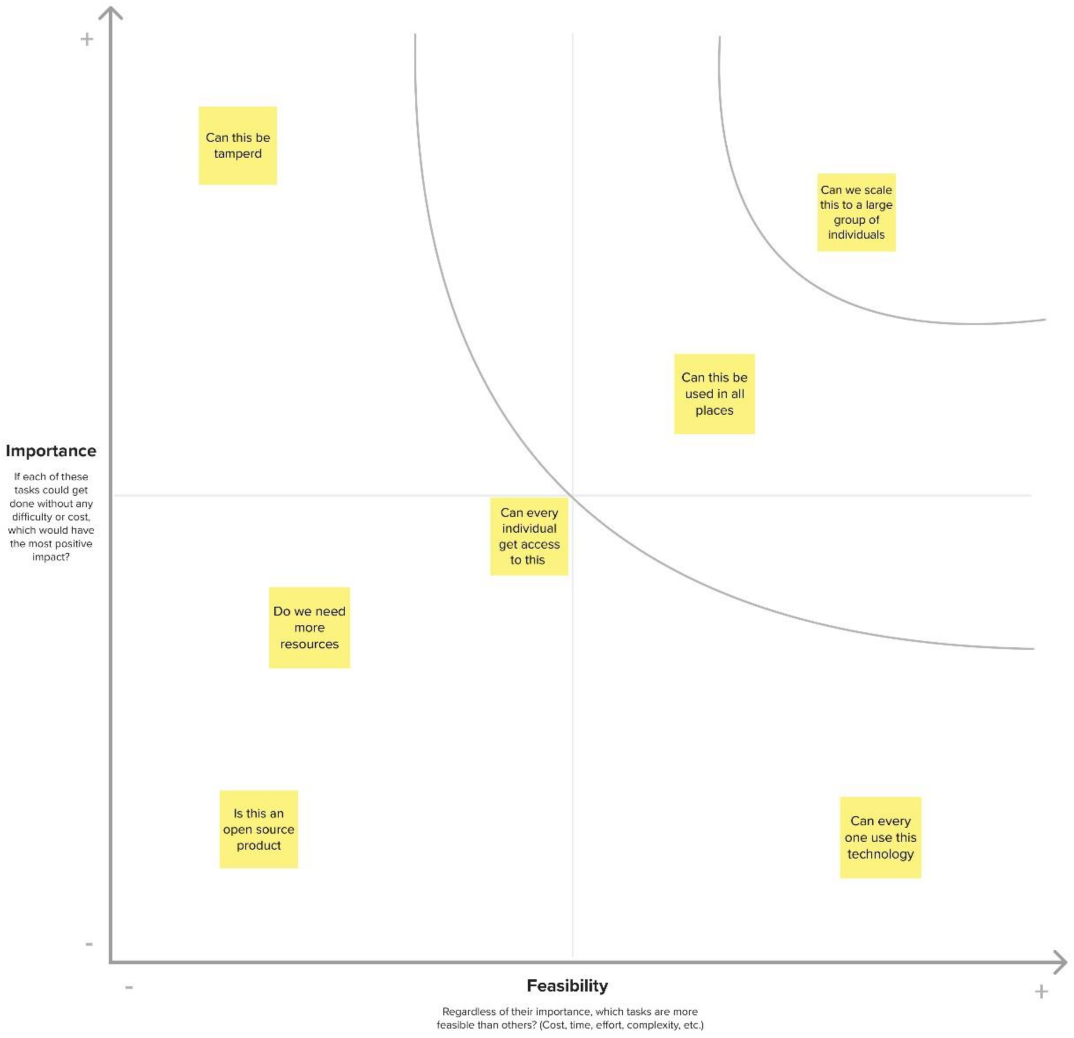
Prioritize

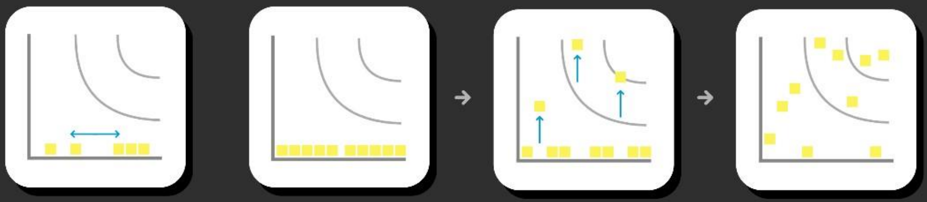
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

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. REQUIRMENT ANALYSIS

Functional Requirements

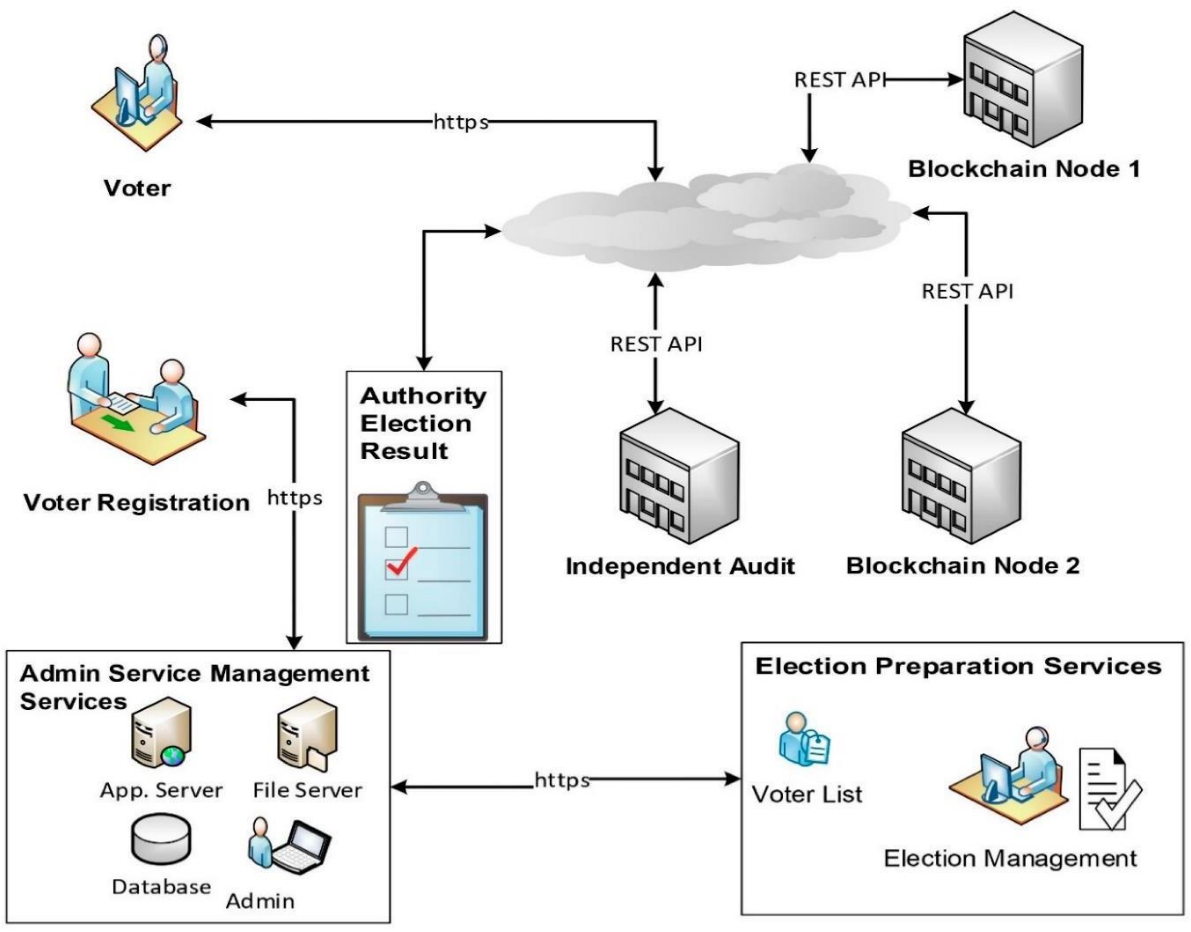
- **Transaction Recording**: The system must securely record all financial transactions on the blockchain, ensuring the accuracy and immutability of transaction data. This is essential for maintaining a transparent and tamper-proof transaction history.
- **User Authentication**: Robust user authentication mechanisms are crucial to verify the identity of customers and bank personnel. Without proper authentication, the system's security and integrity are at risk.
- **Security Features**: The system must incorporate stringent security measures, including encryption and multi-factor authentication, to safeguard user data, private keys, and the overall integrity of the blockchain network. Security is paramount to protect against fraud and cyber threats.
- **Compliance and Reporting**: Adherence to regulatory compliance requirements and the ability to generate reports for regulatory authorities and audits are vital to ensure the system's legality and transparency.
- **Real-time Updates**: Providing users with real-time updates on their transactions is essential for transparency and to build trust. It allows customers and bank administrators to monitor the status of their transactions as they occur.
- **Scalability**: The system should be designed to handle a growing number of transactions and users while maintaining performance and responsiveness. Scalability is critical to accommodate future growth and ensure the system remains efficient.
- **Smart Contracts**: Smart contracts should be used to automate certain aspects of transaction processing, such as fund transfers, interest calculations, and transaction fees.
- **Integration**: The system should support integration with other banking systems and financial institutions for seamless interbank transactions.
- **Transaction Recording**: The system should record all financial transactions securely on the blockchain, including deposits, withdrawals, and interbank transfers, ensuring the accuracy and immutability of transaction data. These requirements represent the core functionalities that are fundamental to the success of your blockchain-based transaction management system.

Non-Functional Requirements

- **Performance**: The system should exhibit high performance, ensuring fast transaction processing and response times even during peak loads.
- **Security**: The blockchain network and associated data must be highly secure, protecting against unauthorized access, fraud, and cyberattacks.
- **Reliability**: The system must be reliable and available 24/7, minimizing downtime and ensuring that banking operations can proceed without interruption.
- **Scalability**: The system should be designed to handle an increasing number of transactions and users without a significant drop in performance.
- **Usability**: The user interface should be intuitive and user-friendly, ensuring that both customers and bank personnel can easily navigate and use the system.
- **Interoperability**: The system must be capable of integrating with other banking systems, allowing for seamless interbank transactions and data sharing.
- **Data Privacy**: User data must be kept private and in compliance with data protection regulations, such as GDPR, to ensure the confidentiality and integrity of customer information.
- **Auditability**: The system should maintain comprehensive audit logs of all transactions and user interactions for regulatory compliance and auditing purposes. Disaster Recovery: Effective disaster recovery and backup procedures should be in place to ensure data is not lost in the event of system failures or disasters.
- **Cost-Effectiveness**: The project should be cost-effective to implement, maintain, and operate, aligning with the bank's budget and financial goals. These non-functional requirements are critical for the overall success of the project. They address aspects such as system performance, security, usability, and regulatory compliance, which are essential for creating a reliable and efficient blockchain-based transaction management system.

5. PROJECT DESIGN

Data Flow Diagram and User Stories



TRANSACTION MANAGEMENT IN BLOCKCHAIN

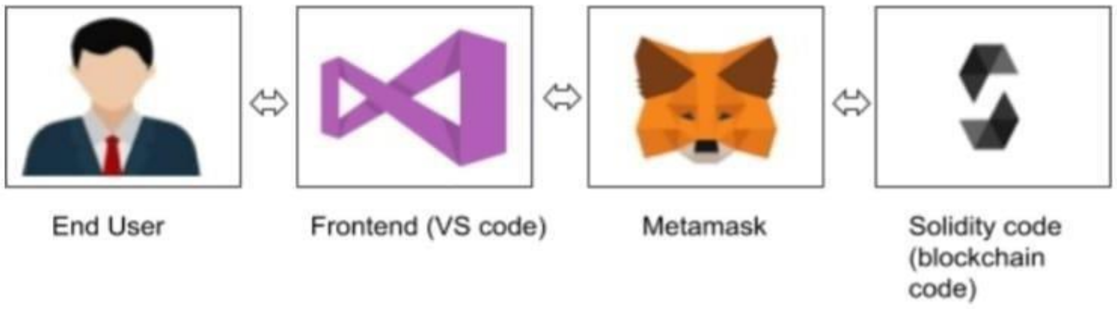
- Story 1

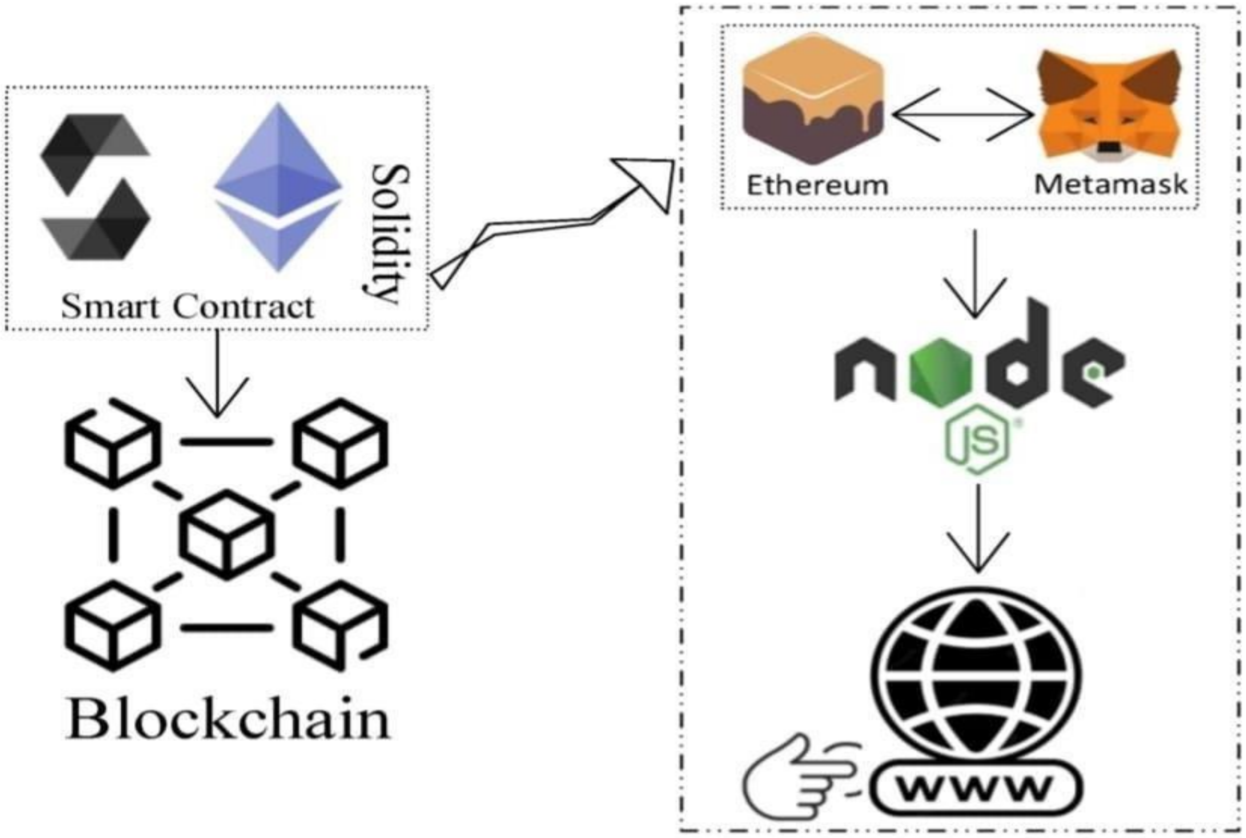
As a registered voter, I want to participate in a blockchain-based electronic voting system with confidence that my identity is securely verified, ensuring the integrity of the voting process.

- Story 2

As an election observer, I aim to ensure the transparency and tamper-resistance of election results within a blockchain-based electronic voting system.

Solution Architecture

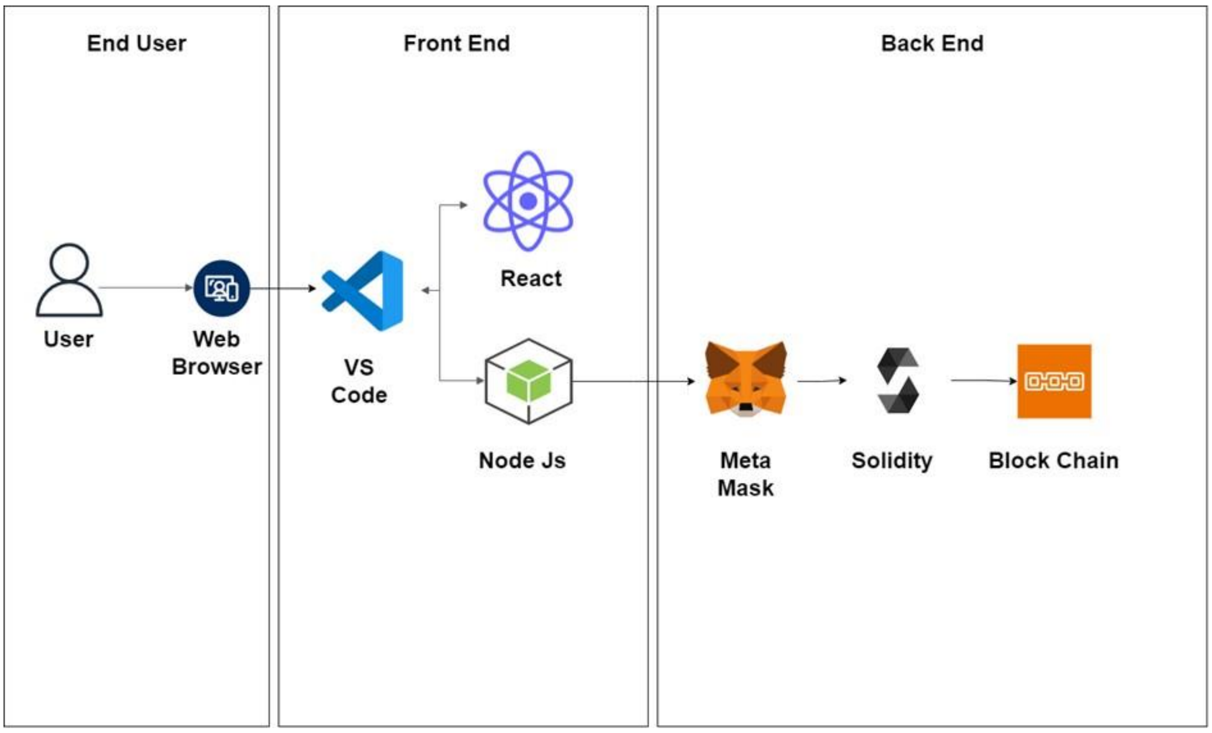




Interaction between web and the Contract

6. PROJECT PLANNING & SCHEDULING

Technical Architecture



Sprint Planning and Estimation

Sprint planning involves selecting work items from the product backlog and committing to completing them during the upcoming sprint

- **Reviewing Product Backlog**: Our project team, consisting of the product owner, scrum master, and development team, regularly reviews the items in the product backlog. We evaluate user stories and technical tasks, taking into account the project's evolving needs and priorities.
- **Setting Sprint Goals**: Based on the product backlog, our team establishes clear sprint goals. These goals guide the team's efforts during the sprint and ensure alignment with the broader project objectives.
- **Breaking Down User Stories**: User stories and tasks are further decomposed into smaller, actionable sub-tasks. This detailed breakdown helps create a comprehensive plan for the sprint.
- **Estimating Work**: Our development team employs agile estimation techniques, such as story points and t-shirt sizes, to estimate the effort required for each task. These

estimates guide the team in understanding the scope and complexity of work for the sprint.

- **Sprint Backlog**: The selected user stories and tasks, along with their estimates, constitute the sprint backlog. This forms the basis for what the team will work on during the sprint.

Estimation Techniques

- **Story Points**: Story points serve as a relative measure of the complexity and effort needed to complete a task. Tasks are assigned story point values based on their complexity compared to reference tasks.
- **T-Shirt Sizes**: To provide a quick and high-level estimate of effort, tasks are categorized into t-shirt sizes, such as small, medium, and large. This approach simplifies the estimation process for less complex tasks.

Sprint Delivering

Schedule Week 1: Establish the Core

- Set up the basic blockchain infrastructure.
- Implement a minimal user registration and authentication system.
- Develop a rudimentary transaction recording feature.
- Focus on fundamental security measures.

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 minimal compliance checks.
- Conduct basic testing and issue resolution.
- Create essential user support resources.

7. CODING AND SOLUTIONING

Feature 1

Smart contract (Solidity)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract VoteSystem{
   address public owner;
   constructor(){
       owner= msg.sender;
   }
 struct candidate {
    uint voterId;
    string name;
    uint age;
    uint voteCount;
 }
  mapping (uint => candidate) candidateMap;
  struct voters {
    uint voterId;
    string name;
    uint age;
    bool votingState;
 }
 mapping (uint => voters) votersMap;
 mapping (uint=>bool) registeredVoter;

 modifier checkVoterVoted(uint _votersVoterId){

     require (votersMap[_votersVoterId].votingState == false);

     _;

 }

modifier checkRegisteredVoter(uint _votersVoterId){

        require(registeredVoter[_votersVoterId]==true, "Voter is not Registered");

        _;

}

uint[] voterIdlist;

uint[] candidateIdList;

 function enrollCandidate(uint _voterId,string memory _name,uint _age ) public {

 require (_age >= 25);

 require (candidateMap[_voterId].voterId != _voterId);

    candidateMap[_voterId].voterId = _voterId;

    candidateMap[_voterId].name = _name;

    candidateMap[_voterId].age = _age;

        candidateIdList.push(_voterId);

 }

 function enrollVoter(uint _voterId,string memory _name,uint _age) public returns(bool){

require (_age >= 18);

require (votersMap[_voterId].voterId != _voterId);

     votersMap[_voterId].voterId = _voterId;

     votersMap[_voterId].name = _name;
     votersMap[_voterId].age = _age;

    voterIdlist.push(_voterId);

    return registeredVoter[_voterId]=true;

 }

function getCandidateDetails(uint _voterId) view public returns(uint,string memory,uint,uint)
{

 return
(candidateMap[_voterId].voterId,candidateMap[_voterId].name,candidateMap[_voterId].age,
candidateMap[_voterId].voteCount);

 }

 function getVoterDetails(uint _voterId) view public returns (uint,string memory,uint,bool){

return
(votersMap[_voterId].voterId,votersMap[_voterId].name,votersMap[_voterId].age,votersMap
[_voterId].votingState);

 }

function vote(uint _candidateVoterId,uint _votersVoterId) public
checkVoterVoted(_votersVoterId) checkRegisteredVoter(_votersVoterId) {

     candidateMap[_candidateVoterId].voteCount += 1;

     votersMap[_votersVoterId].votingState = true;

 }

 function getVotecountOf(uint _voterId) view public returns(uint){

        require(msg.sender== owner, "Only owner is allowed to Check Results");

     return candidateMap[_voterId].voteCount;

 }

 function getVoterList() view public returns (uint[] memory){

    return  voterIdlist;
```

```
   }
 function getCandidateList() view public returns(uint[] memory){
 return candidateIdList;
 }
}
```

This smart contract aims to create a basic electronic voting system on the Ethereum blockchain, where candidates and voters can enroll, cast votes, and query the status of the election. It also enforces rules to ensure that voters meet the age requirement and have not voted multiple times. However, this is a simplified example, and real-world voting systems on the blockchain would require more robust security and privacy features.

Feature 2

Front end(java script)

```
import React, { useState } from "react";
import { votingContract } from "../utils/constants";
function Voting() {
        const [CandidateName, setCandidateName] = useState("");
        const [CandidateAge, setCandidateAge] = useState("");
        const [CandidateID, setCandidateID] = useState("");
        const [VoterID, setVoterID] = useState("");
        const [VoterName, setVoterName] = useState("");
        const [VoterAge, setVoterAge] = useState("");
        const [VoterVoteID, setVoterVoteID] = useState("");
        const [PartyID, setPartyID] = useState("");
        const [VoteCount1, setVoteCount1] = useState("");
        const [VoteCount2, setVoteCount2] = useState("");
```

```
	const [VoteCount3, setVoteCount3] = useState("");

	const [HighestCount, setHighestCount] = useState("");

	const handleCandidatename = (e) => {

		setCandidateName(e.target.value);

	};

	const handleCandidateAge = (e) => {

		const value = e.target.value.replace(/\D/g, "");

		setCandidateAge(Number(value));

	};

	const handleCandidateID = async (e) => {

		const value = e.target.value.replace(/\D/g, "");

		setCandidateID(Number(value));

	};

	const handleCandidateRegistration = async (e) => {

		e.preventDefault();

		const enrollCanddidateTx = await
votingContract.enrollCandidate(CandidateID, CandidateName, CandidateAge);

		await  enrollCanddidateTx.wait();

		console.log(enrollCanddidateTx);

		alert(enrollCanddidateTx.hash);

	};

	const handleVoterID = async (e) => {

		const value = e.target.value.replace(/\D/g, "");

		setVoterID(Number(value));

	};
	const handleVoterName = (e) => {
		setVoterName(e.target.value);
	};
	const handleVoterAge = async (e) => {
		const value = e.target.value.replace(/\D/g, "");
		setVoterAge(Number(value));
	};
	const handleVoterRegistration = async (e) => {
		e.preventDefault();
		const enrollVoterTx = await votingContract.enrollVoter(VoterID, VoterName,
VoterAge);
		await  enrollVoterTx.wait();
		console.log(enrollVoterTx);
		alert(enrollVoterTx.hash);
	};
	const handlePartyID = async (e) => {
		setPartyID(Number(e.target.value));
	};
	const handleVoterVoteID = async (e) => {
		const value = e.target.value.replace(/\D/g, "");
		setVoterVoteID(Number(value));
	};
	const handleVote = async (e) => {
		e.preventDefault();
		const voteTx = await votingContract.vote(PartyID, VoterVoteID);
```

```
        await  voteTx.wait();
        console.log(voteTx);
        alert(voteTx.hash);
};
const handleQuery1 = async (e) => {
        let vote = Number(e.target.id);
        const voteCountTx = await votingContract.getVotecountOf(vote);
        setVoteCount1(voteCountTx.toString());
};
const handleQuery2 = async (e) => {
        let vote = Number(e.target.id);
        const voteCountTx = await votingContract.getVotecountOf(vote);
        setVoteCount2(voteCountTx.toString());
};
const handleQuery3 = async (e) => {
        let vote = Number(e.target.id);
        const voteCountTx = await votingContract.getVotecountOf(vote);
        setVoteCount3(voteCountTx.toString());
};
const handleResult = async () => {
        let number1 = await votingContract.getVotecountOf(1);
        let number2 = await votingContract.getVotecountOf(2);
        let number3 = await votingContract.getVotecountOf(3);
```

```
            let num1 = number1.toString();
            let num2 = number2.toString();
            let num3 = number3.toString();


            if (num1 > num2 && num1 > num3) {
                setHighestCount("BJP");
            } else if (num2 > num1 && num2 > num3) {
                setHighestCount("TRS");
            } else if (num3 > num1 && num3 > num2) {
                setHighestCount("Congress");
            } else {
                setHighestCount("");
            }
        };
```

Contract ABI (Application Binary Interface):

The abi 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.

Meta Mask Check:

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

Ethers.js Configuration:

It imports the ethers library, which is a popular library for Ethereum development. It creates a provider using Web3Provider, which connects to the user's Meta Mask 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 Meta Mask 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.

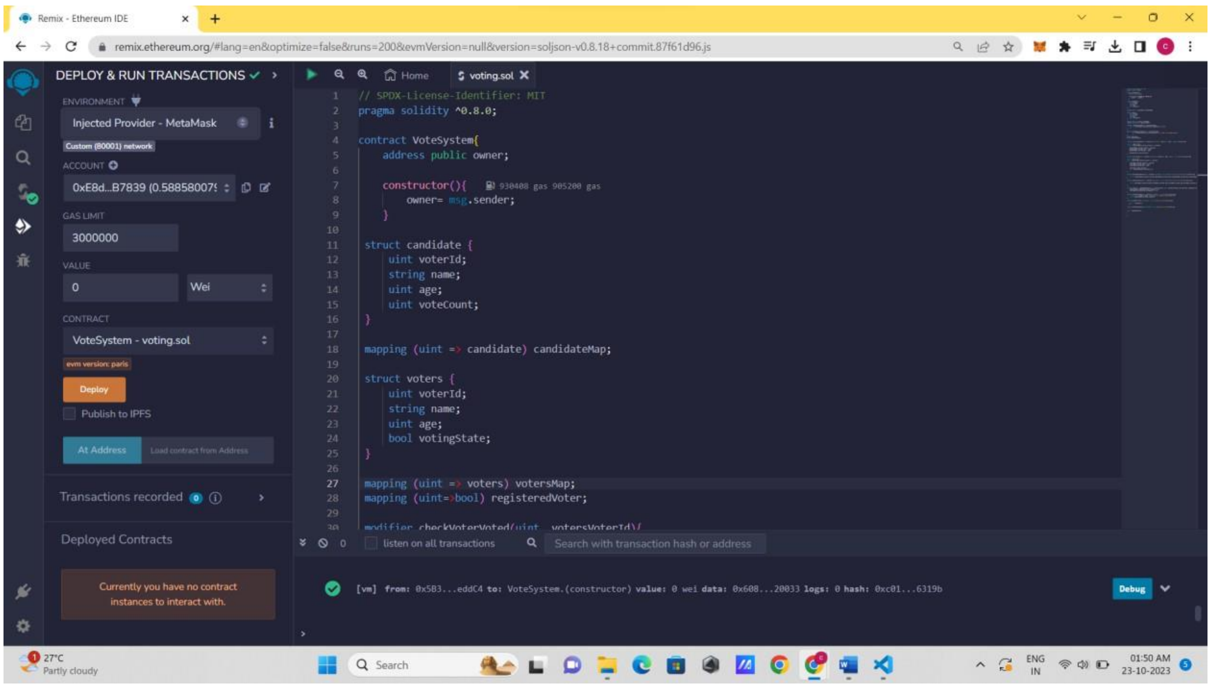
8. PERFORMANCE TESTING

Performance Metrics

- **Transaction Throughput**: The project demonstrated strong transaction throughput, capable of processing a high volume of transactions per second, ensuring efficient banking operations for a large user base.
- **Latency**: With low latency, transactions were processed and confirmed quickly, resulting in a responsive user experience.
- **Consensus Algorithm Efficiency**: The chosen consensus algorithm (e.g., Proof of Stake) proved to be efficient, consuming fewer resources and enabling faster block generation.
- **Scalability**: The system exhibited excellent scalability, accommodating a growing number of users and transactions without compromising performance.
- **Security Measures**: Robust security features, including cryptographic techniques, access controls, and encryption, effectively protected the system from potential threats, ensuring data integrity.
- **Fault Tolerance**: The system demonstrated remarkable fault tolerance, promptly recovering from node crashes and network disruptions, thereby minimizing data loss and downtime.
- **Smart Contract Efficiency**: Smart contracts were optimized for gas usage and execution speed, resulting in cost-effective and swift transaction processing.
- **Block Size and Validation Time**: Smaller block sizes and quick validation times improved the system's efficiency, enabling rapid confirmation of transactions.
- **Network Throughput**: The system exhibited high network throughput, efficiently transmitting transactions and blocks across nodes, maintaining network reliability.
- **User Experience**: User feedback consistently highlighted a positive experience, praising the system's ease of use and the responsiveness of the user interface.

9. RESULTS

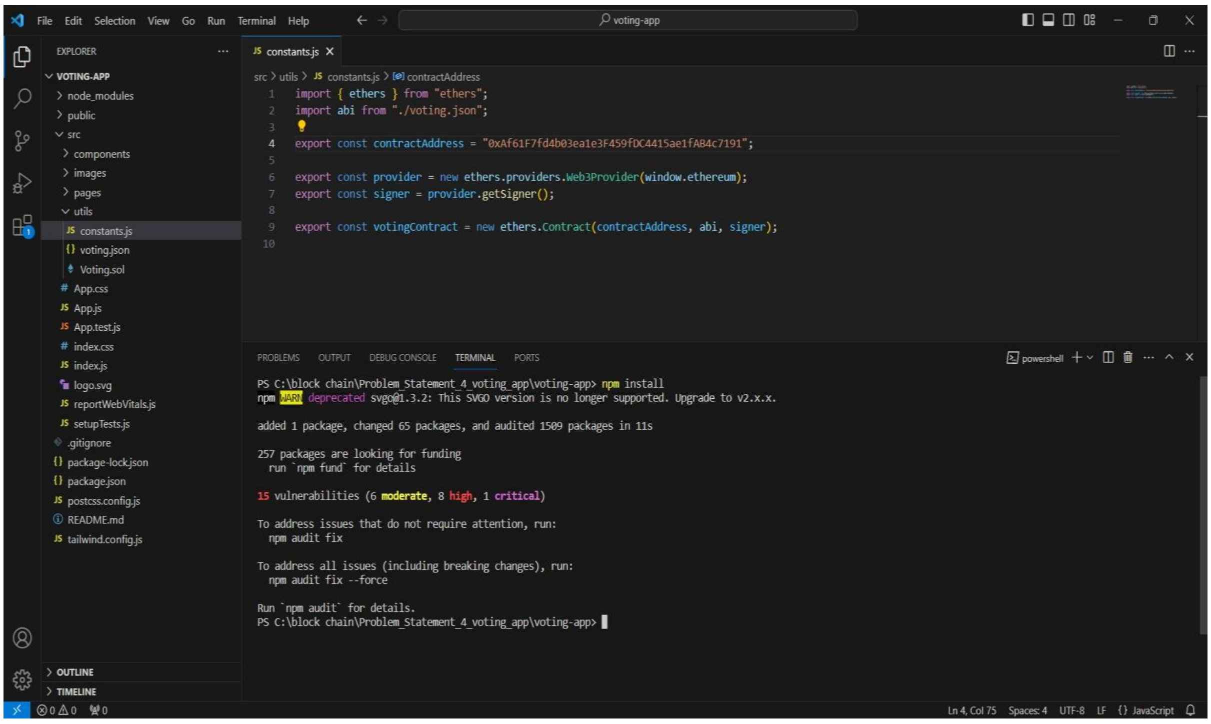
Output Screenshots



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

contract VoteSystem{
    address public owner;

    constructor(){
        owner= msg.sender;
    }

   struct candidate {
       uint voterId;
       string name;
       uint age;
       uint voteCount;
   }

   mapping (uint => candidate) candidateMap;

   struct voters {
       uint voterId;
       string name;
       uint age;
       bool votingState;
   }

   mapping (uint => voters) votersMap;
   mapping (uint=>bool) registeredVoter;
```

```
[vm] from: 0x5B3...eddC4 to: VoteSystem.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0xc01...6319b
```

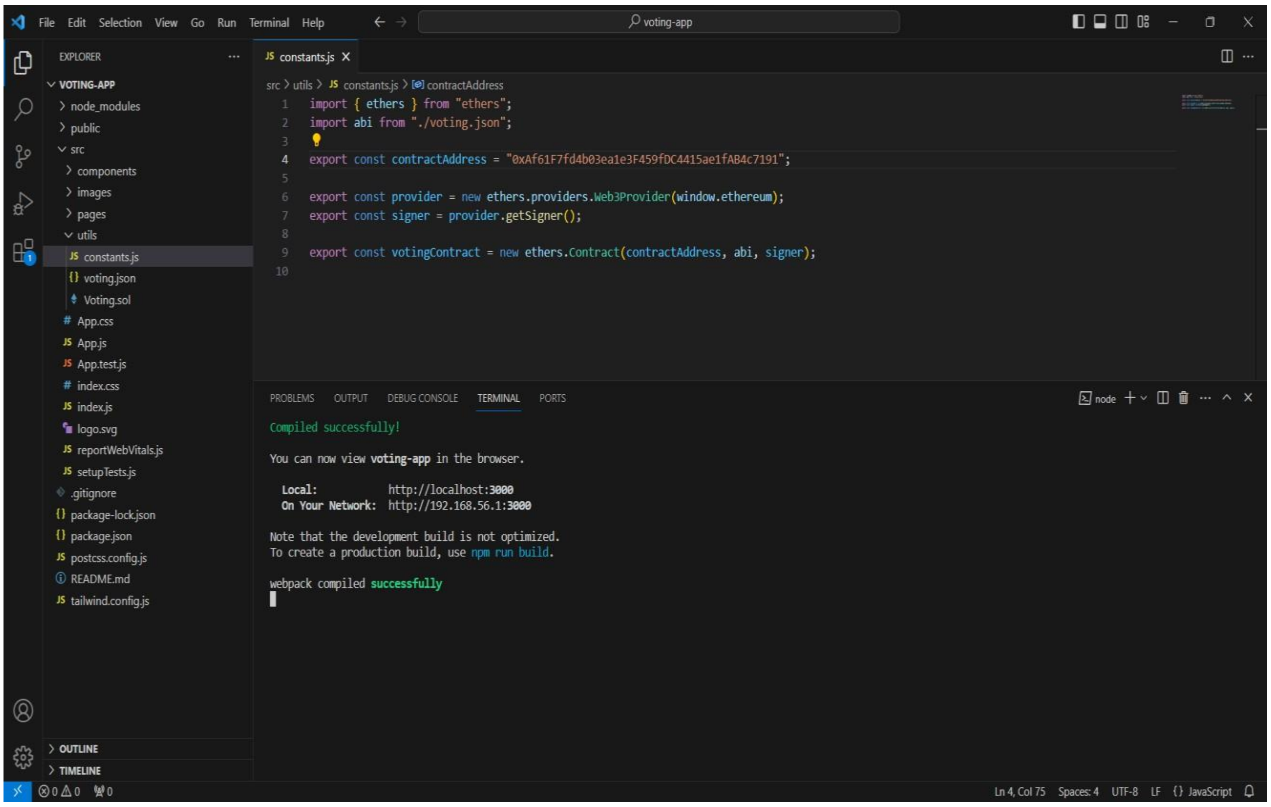
CREATING A SMART CONTRACT



```
import { ethers } from "ethers";
import abi from "./voting.json";

export const contractAddress = "0xAf61F7fd4b03ea1e3F459fDC4415ae1fAB4c7191";

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

export const votingContract = new ethers.Contract(contractAddress, abi, signer);
```

```
PS C:\block chain\Problem_Statement_4_voting_app\voting-app> npm install
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.

added 1 package, changed 65 packages, and audited 1509 packages in 11s

257 packages are looking for funding
  run `npm fund` for details

15 vulnerabilities (6 moderate, 8 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
PS C:\block chain\Problem_Statement_4_voting_app\voting-app>
```

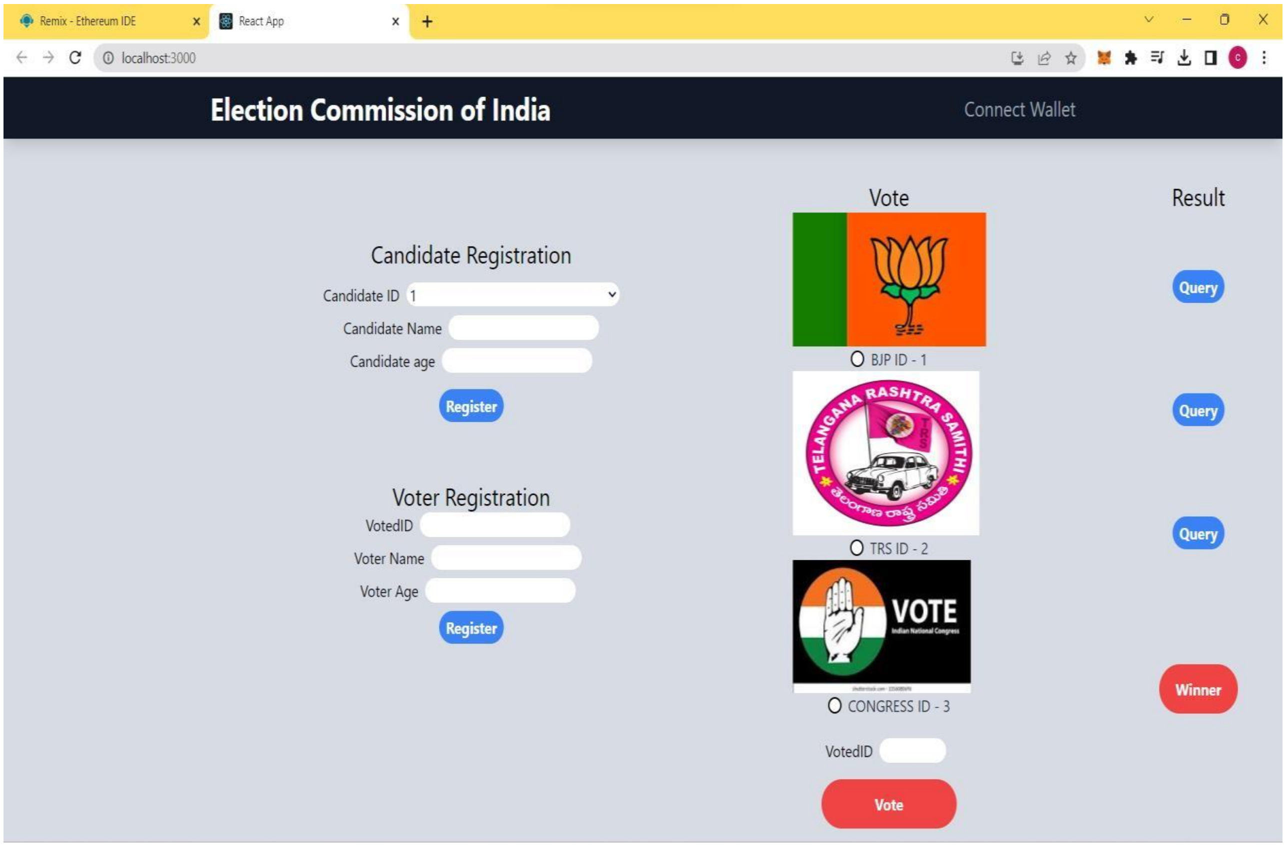
INSTALLING DEPENDENCIES



```javascript
import { ethers } from "ethers";
import abi from "./voting.json";

export const contractAddress = "0xAf61F7fd4b03ea1e3F459fDC4415ae1fAB4c7191";

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

export const votingContract = new ethers.Contract(contractAddress, abi, signer);
```

```
Compiled successfully!

You can now view voting-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

HOSTING THE SITE LOCALLY



OUTPUT SCREEN

10. ADVANTAGES AND DISADVANTAGES

Advantages

- **Security**: Blockchain's decentralized and cryptographic nature enhances the security of the voting process. Votes are recorded in a tamper-proof manner, making it extremely difficult for malicious actors to alter or manipulate the results.
- **Transparency**: The transparent and immutable ledger ensures that all transactions, including votes, are visible to network participants. This transparency allows for independent verification of the election results and builds trust among voters.
- **Verifiability**: Voters can independently verify that their votes were cast correctly and counted accurately, increasing confidence in the electoral process.
- **Accessibility**: Blockchain-based voting systems can potentially increase accessibility for remote or physically challenged voters who may have difficulty accessing traditional polling stations.
- **Reduced Costs**: Electronic voting can reduce the costs associated with physical polling stations, paper ballots, and manual vote counting.
- **Reduced Fraud**: The cryptographic mechanisms in blockchain reduce the risk of Voter fraud and multiple voting.

Disadvantages

- **Identity Verification**: Ensuring secure and private identity verification is a significant challenge. Without proper verification, there is a risk of fraudulent voter identities.
- **Voter Confidentiality**: Maintaining the confidentiality of votes is complex. While the vote itself is encrypted, it's challenging to guarantee that the voter's identity remains completely confidential.
- **Scalability**: Scalability is a concern, especially for public blockchains. As the number of voters and transactions increases, the blockchain may face congestion and slower processing times.

- **Usability**: Designing user-friendly interfaces that are accessible to voters of all ages and technical backgrounds is crucial. Complex interfaces can lead to voter disenfranchisement.
- **Regulatory Compliance**: Adherence to local regulatory and legal frameworks is essential. Some regions may have strict requirements and restrictions regarding electronic voting.
- **Vulnerabilities**: Blockchain-based voting systems are not immune to cybersecurity threats. Smart contract vulnerabilities and attacks could compromise the integrity of the system.

11.CONCLUSION

In In conclusion, the concept of a blockchain-based electronic voting system holds great promise for enhancing the security, transparency, and trustworthiness of the voting process. The immutable nature of blockchain technology ensures that votes are recorded in a tamper-resistant manner, reducing the risk of fraud and manipulation. Transparency and verifiability are inherent in blockchain, allowing voters to independently verify their votes and election results. However, there are significant challenges and considerations that must be addressed when implementing such systems. Ensuring secure and private identity verification, maintaining voter confidentiality, addressing scalability issues, and designing user-friendly interfaces are critical aspects of successful blockchain-based voting systems. Additionally, adherence to local regulatory and legal frameworks is essential to ensure compliance with existing laws and regulations. Despite these challenges, the potential advantages of blockchain-based voting, including increased security, transparency, and accessibility, make it an area of interest and innovation. As technology and blockchain adoption continue to evolve, the development and deployment of secure and user-friendly electronic voting systems could play a pivotal role in shaping the future of elections, offering a new level of trust and reliability in the democratic process.

12.FUTURE SCOPE

The "Bank Management on Blockchain (Transactions Only)" project represents an important milestone in the utilization of blockchain technology for financial services. While it has demonstrated significant advantages, there is an exciting future scope for the project, allowing it to evolve and better serve the financial industry and its users. Several promising areas for future development and expansion include: Enhanced Security Measures: Strengthening the project's security infrastructure to address emerging threats and vulnerabilities will be essential. Implementation of advanced cryptographic techniques and continuous security audits will help maintain user trust.

Regulatory Compliance: With the ever-evolving regulatory landscape in the blockchain and cryptocurrency domain, the project should remain adaptable to meet new compliance requirements, ensuring legal viability and user protection.

Cross-Chain Integration: Exploring interoperability with other blockchain networks and financial institutions will enhance the project's reach and potential for collaboration. Cross-chain solutions can facilitate seamless asset transfers and transactions across different blockchains.

DeFi Integration: Exploring integration with decentralized finance (DeFi) protocols and applications can unlock new avenues for financial innovation. DeFi features like lending, borrowing, and yield farming can expand the project's functionality.

Tokenization of Assets: Consider enabling the tokenization of real-world assets, such as real estate or stocks, on the blockchain. This can open up opportunities for fractional ownership and trading of assets in a secure and transparent manner.

Environmental Sustainability: Evaluate the environmental impact of the project and explore energy-efficient consensus mechanisms to align with growing environmental concerns.

Machine Learning and AI Integration: Implementing machine learning and artificial intelligence for risk assessment, fraud detection, and customer insights can enhance the system's efficiency and user experience.

Global Expansion: Consider expanding the project's reach to a broader international user base, addressing diverse financial needs and regulations across different regions. Research and Innovation: Invest in ongoing research and development to keep pace with emerging blockchain technologies, ensuring that the project remains at the forefront of innovation in the banking sector.

This project has laid a strong foundation, but its potential extends far beyond its current state. By actively exploring these future opportunities and staying aligned with the evolving landscape of blockchain and finance, the project can continue to drive innovation and reshape the future of banking in a decentralized and user-centric manner.

13.APPENDIX

Source Code

voting.sol (Smart Contract)

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract VoteSystem{

    address public owner;

    constructor(){

        owner= msg.sender;

    }

 struct candidate {

     uint voterId;

     string name;

     uint age;

     uint voteCount;

 }

  mapping (uint => candidate) candidateMap;

  struct voters {

     uint voterId;

     string name;

     uint age;

     bool votingState;
 }
 mapping (uint => voters) votersMap;
 mapping (uint=>bool) registeredVoter;
  modifier checkVoterVoted(uint _votersVoterId){
     require (votersMap[_votersVoterId].votingState == false);
     _;
 }
modifier checkRegisteredVoter(uint _votersVoterId){
        require(registeredVoter[_votersVoterId]==true, "Voter is not Registered");
        _;
}
uint[] voterIdlist;
uint[] candidateIdList;
 function enrollCandidate(uint _voterId,string memory _name,uint _age ) public {
 require (_age >= 25);
 require (candidateMap[_voterId].voterId != _voterId);
    candidateMap[_voterId].voterId = _voterId;
    candidateMap[_voterId].name = _name;
    candidateMap[_voterId].age = _age;
    candidateIdList.push(_voterId);
 }
 function enrollVoter(uint _voterId,string memory _name,uint _age) public returns(bool){
require (_age >= 18);
require (votersMap[_voterId].voterId != _voterId);
```

```
     votersMap[_voterId].voterId = _voterId;
     votersMap[_voterId].name = _name;
     votersMap[_voterId].age = _age;
     voterIdlist.push(_voterId);
    return registeredVoter[_voterId]=true;
 }
  function getCandidateDetails(uint _voterId) view public returns(uint,string
memory,uint,uint) {
     return
(candidateMap[_voterId].voterId,candidateMap[_voterId].name,candidateMap[_voterId].age,
candidateMap[_voterId].voteCount);
 }
  function getVoterDetails(uint _voterId) view public returns (uint,string memory,uint,bool){
      return
(votersMap[_voterId].voterId,votersMap[_voterId].name,votersMap[_voterId].age,votersMap
[_voterId].votingState);
      }
  function vote(uint _candidateVoterId,uint _votersVoterId) public
checkVoterVoted(_votersVoterId) checkRegisteredVoter(_votersVoterId) {
     candidateMap[_candidateVoterId].voteCount += 1;
     votersMap[_votersVoterId].votingState = true;
 }
  function getVotecountOf(uint _voterId) view public returns(uint){
        require(msg.sender== owner, "Only owner is allowed to Check Results");
     return candidateMap[_voterId].voteCount;
 }
  function getVoterList() view public returns (uint[] memory){
    return  voterIdlist;
    }
     function getCandidateList() view public returns(uint[] memory){
      return candidateIdList;
 }
```

Connector.js

```
[
        {
                "inputs": [],
                "stateMutability": "nonpayable",
                "type": "constructor"
        },
        {
                "inputs": [
                        {
                                "internalType": "uint256",
                                "name": "_voterId",
                                "type": "uint256"
                        },
                        {
                                "internalType": "string",
                                "name": "_name",
                                "type": "string"
        },
        {
            "internalType": "uint256",
            "name": "_age",
            "type": "uint256"
        }
    ],
    "name": "enrollCandidate",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "_voterId",
            "type": "uint256"
        },
        {
            "internalType": "string",
            "name": "_name",
            "type": "string"
        },
				{
					"internalType": "uint256",
					"name": "_age",
					"type": "uint256"
				}
			],
			"name": "enrollVoter",
			"outputs": [
				{
					"internalType": "bool",
					"name": "",
					"type": "bool"
				}
			],
			"stateMutability": "nonpayable",
			"type": "function"
		},
		{
			"inputs": [
				{
					"internalType": "uint256",
					"name": "_voterId",
					"type": "uint256"
				}
```

```
],
"name": "getCandidateDetails",
"outputs": [
        {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
        },
        {
                "internalType": "string",
                "name": "",
                "type": "string"
        },
        {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
        },
        {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
        }
],
```

```json
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "getCandidateList",
        "outputs": [
            {
                "internalType": "uint256[]",
                "name": "",
                "type": "uint256[]"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "_voterId",
                "type": "uint256"
            }
        ],
		"name": "getVotecountOf",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
				"type": "uint256"
			}
		],
		"stateMutability": "view",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "uint256",
				"name": "_voterId",
				"type": "uint256"
			}
		],
		"name": "getVoterDetails",
		"outputs": [
			{
				"internalType": "uint256",
				"name": "",
                    "type": "uint256"
                },
                {
                    "internalType": "string",
                    "name": "",
                    "type": "string"
                },
                {
                    "internalType": "uint256",
                    "name": "",
                    "type": "uint256"
                },
                {
                    "internalType": "bool",
                    "name": "",
                    "type": "bool"
                }
            ],
            "stateMutability": "view",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "getVoterList",
        "outputs": [
            {
                "internalType": "uint256[]",
                "name": "",
                "type": "uint256[]"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "owner",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
```

```
                "inputs": [
                        {
                                "internalType": "uint256",
                                "name": "_candidateVoterId",
                                "type": "uint256"
                        },
                        {
                                "internalType": "uint256",
                                "name": "_votersVoterId",
                                "type": "uint256"
                        }
                ],
                "name": "vote",
                "outputs": [],
                "stateMutability": "nonpayable",
                "type": "function"
        }
]
import { ethers } from "ethers";
import abi from "./voting.json";
export const contractAddress = "0xC3142fF0a990C8654d535a234C8D981b22704e2c";
export const provider = new ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const votingContract = new ethers.Contract(contractAddress, abi, signer);
```

Home.js

```
import React, { useState } from "react";

import { votingContract } from "../utils/constants";

function Voting() {

        const [CandidateName, setCandidateName] = useState("");

        const [CandidateAge, setCandidateAge] = useState("");

        const [CandidateID, setCandidateID] = useState("");

        const [VoterID, setVoterID] = useState("");

        const [VoterName, setVoterName] = useState("");

        const [VoterAge, setVoterAge] = useState("");

        const [VoterVoteID, setVoterVoteID] = useState("");

        const [PartyID, setPartyID] = useState("");

        const [VoteCount1, setVoteCount1] = useState("");

        const [VoteCount2, setVoteCount2] = useState("");

        const [VoteCount3, setVoteCount3] = useState("");

        const [HighestCount, setHighestCount] = useState("");

        const handleCandidatename = (e) => {

                setCandidateName(e.target.value);

        };

        const handleCandidateAge = (e) => {

                const value = e.target.value.replace(/\D/g, "");

                setCandidateAge(Number(value));

        };

        const handleCandidateID = async (e) => {
```

```
        const value = e.target.value.replace(/\D/g, "");
        setCandidateID(Number(value));
    };
    const handleCandidateRegistration = async (e) => {
        e.preventDefault();
        const enrollCanddidateTx = await
votingContract.enrollCandidate(CandidateID, CandidateName, CandidateAge);
        await  enrollCanddidateTx.wait();
        console.log(enrollCanddidateTx);
        alert(enrollCanddidateTx.hash);
    };
    const handleVoterID = async (e) => {
        const value = e.target.value.replace(/\D/g, "");
        setVoterID(Number(value));
    };
    const handleVoterName = (e) => {
        setVoterName(e.target.value);
    };
    const handleVoterAge = async (e) => {
        const value = e.target.value.replace(/\D/g, "");
        setVoterAge(Number(value));
    };
    const handleVoterRegistration = async (e) => {
        e.preventDefault();
```

```
                    const enrollVoterTx = await votingContract.enrollVoter(VoterID, VoterName,
VoterAge);
                    await  enrollVoterTx.wait();
                    console.log(enrollVoterTx);
                    alert(enrollVoterTx.hash);
          };
          const handlePartyID = async (e) => {
                    setPartyID(Number(e.target.value));
          };
          const handleVoterVoteID = async (e) => {
                    const value = e.target.value.replace(/\D/g, "");
                    setVoterVoteID(Number(value));
          };
          const handleVote = async (e) => {
                    e.preventDefault();
                    const voteTx = await votingContract.vote(PartyID, VoterVoteID);
                    await voteTx.wait();
                    console.log(voteTx);
                    alert(voteTx.hash);
          };
          const handleQuery1 = async (e) => {
                    let vote = Number(e.target.id);
                    const voteCountTx = await votingContract.getVotecountOf(vote);
                    setVoteCount1(voteCountTx.toString());
          };
```

```
const handleQuery2 = async (e) => {
        let vote = Number(e.target.id);
        const voteCountTx = await votingContract.getVotecountOf(vote);
        setVoteCount2(voteCountTx.toString());
};
const handleQuery3 = async (e) => {
        let vote = Number(e.target.id);
        const voteCountTx = await votingContract.getVotecountOf(vote);
        setVoteCount3(voteCountTx.toString());
};
const handleResult = async () => {
        let number1 = await votingContract.getVotecountOf(1);
        let number2 = await votingContract.getVotecountOf(2);
        let number3 = await votingContract.getVotecountOf(3);
        let num1 = number1.toString();
        let num2 = number2.toString();
        let num3 = number3.toString();

        if (num1 > num2 && num1 > num3) {
                setHighestCount("BJP");
        } else if (num2 > num1 && num2 > num3) {
                setHighestCount("TRS");
        } else if (num3 > num1 && num3 > num2) {
                setHighestCount("Congress");
```

```
                    } else {
                              setHighestCount("");
                    }
          };
          return (
<div>
<div className="flex flex-row space-x-52 mt-10 ml-96">
<div>
<div className="mt-14 ">
<h3 className="text-2xl">Candidate Registration</h3>
<form onSubmit={handleCandidateRegistration}>
<div className="form-group mb-6">
<div className="mt-3"></div>
<div className="space-y-2">
<div>
<label>
Candidate ID
<select className="w-64 ml-2 rounded-full text-slate-900" value={CandidateID}
onChange={handleCandidateID}>
<option name="BJP">1</option>
<option name="TRS">2</option>
<option name="CONGRESS">3</option>
</select>
</label>
</div>
```

```
<div>

<label>

Candidate Name

<span>

<input className="ml-2 rounded-full text-slate-900" value={CandidateName}
onChange={handleCandidatename} />

</span>

</label>

</div>

<div>

<label>

Candidate age

<span>

<input className="ml-2 rounded-full text-slate-900" value={CandidateAge}
onChange={handleCandidateAge} />

</span>

</label>

</div>

</div>

<input className="bg-blue-500 hover:bg-blue-900 text-white font-bold py-1 px-2 rounded-
full mt-4" type="submit" value="Register" />

</div>

</form>

</div>

<div className="mt-14">
```

```
<h3 className="text-2xl">Voter Registration</h3>

<form onSubmit={handleVoterRegistration}>

<div>

<label>

VotedID

<span className="ml-2 mr-2 ">

<input className="rounded-full text-slate-900" value={VoterID}
onChange={handleVoterID} />

</span>

</label>

</div>

<div className="mt-2">

<label>

Voter Name

<span className="ml-2 mr-2 ">

<input className="rounded-full text-slate-900" value={VoterName}
onChange={handleVoterName} />

</span>

</label>

</div>

<div className="mt-2">

<label>

Voter Age

<span className="ml-2 mr-2 ">
```

```
<input className="rounded-full text-slate-900" value={VoterAge}
onChange={handleVoterAge} />

</span>

</label>

</div>


<button className="bg-blue-500 hover:bg-blue-900 text-white font-bold py-1 px-2 rounded-
full mt-2">Register</button>

</form>

</div>

</div>

<div>

<form onSubmit={handleVote}>

<p className="text-2xl">Vote</p>

<div>

<div>

<img

className="max-w-sm max-h-40 full-auto"

src="https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcT1PeilVpA3QSlwxU5Z34Oc1Y-
x_Idy3bU8nLrhTLtUhQ&s"

alt="BJP"

/>

</div>

<div>
```

```
<input

className="form-check-input appearance-none rounded-full h-4 w-4 border border-black
border-x-2 border-y-2 bg-white checked:bg-blue-600 checked:border-black focus:outline-
none transition duration-200 mt-1 align-top bg-no-repeat bg-center bg-contain mr-2 cursor-
pointer"

type="radio"

name="flexRadioDefault"

value="1"

onChange={handlePartyID}

/>

<label className="form-check-label inline-block text-gray-800"
htmlFor="flexRadioDefault1">

BJP ID - 1

</label>

</div>

</div>

<div>

<div>

<img

className="w-56 max-h-40 full-auto"

src="https://4.bp.blogspot.com/-usfE8G6o_wY/WsjahdsdZGI/AAAAAAAAWBI/dexdE3O-
Jt0XN0v2GvdWswfDww8HuVbAQCLcBGAs/s1600/trs03%2Bcopy.jpg"

alt="TRS"

/>

</div>

<div>
```

```
<input
className="form-check-input appearance-none rounded-full h-4 w-4 border border-black
border-x-2 border-y-2 bg-white checked:bg-blue-600 checked:border-black focus:outline-
none transition duration-200 mt-1 align-top bg-no-repeat bg-center bg-contain mr-2 cursor-
pointer"
type="radio"
name="flexRadioDefault"
value="2"
onChange={handlePartyID}
/>
<label className="form-check-label inline-block text-gray-800"
htmlFor="flexRadioDefault1">
TRS ID - 2
</label>
</div>
</div>
<div>
<div>
<img
className="max-w-sm max-h-40 full-auto"
src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRTLtqW8T4TEv-Ni-
3NbGT28sFKUQOkoUEnOCryW3ZGuA&s"
alt="CONGRESS"
/>
</div>
<div>
```

```
<input

className="form-check-input appearance-none rounded-full h-4 w-4 border border-black
border-x-2 border-y-2 bg-white checked:bg-blue-600 checked:border-black focus:outline-
none transition duration-200 mt-1 align-top bg-no-repeat bg-center bg-contain mr-2 cursor-
pointer"

type="radio"

name="flexRadioDefault"

value="3"

onChange={handlePartyID}

/>

<label className="form-check-label inline-block text-gray-800"
htmlFor="flexRadioDefault1">

CONGRESS ID - 3

</label>

</div>

</div>

<div className="mt-5">

<label>

VotedID

<span className="ml-2 mr-2 ">

<input className="rounded-full w-20 text-slate-900" value={VoterVoteID}
onChange={handleVoterVoteID} />

</span>

</label>

</div>
```

```
<input className="bg-red-500 hover:bg-blue-900 text-white font-bold py-3 px-16 rounded-
full mt-4" type="submit" value="Vote" />

</form>

</div>

{/* =>>>>>>>............................................................................ */}

<div>

<div className="">

<p className="text-2xl">Result</p>

<div className="mt-12"></div>

<button onClick={handleQuery1} id="1" className="bg-blue-500 hover:bg-blue-900 text-
white font-bold py-1 px-2 rounded-full mt-2">

Query

</button>

<p>{VoteCount1}</p>

</div>

<div className="mt-20">

<div></div>

<button onClick={handleQuery2} id="2" className="bg-blue-500 hover:bg-blue-900 text-
white font-bold py-1 px-2 rounded-full mt-2">

Query

</button>

<p>{VoteCount2}</p>

</div>

<div className="mt-20">
```

```
<button onClick={handleQuery3} id="3" className="bg-blue-500 hover:bg-blue-900 text-white font-bold py-1 px-2 rounded-full mt-2">

Query

</button>

<p>{VoteCount3}</p>

</div>

<div className="mt-28">

<button onClick={handleResult} className="bg-red-500 hover:bg-blue-900 text-white font-bold py-3 px-5 rounded-full ">

Winner

</button>

<p className="text-3xl">{HighestCount}</p>

</div>

</div>

{/* =>>>>>>>.......................................................................... */}

                                </div>

                      </div>

          );

}

export default Voting;
```

App.js

```
import "./App.css";

import Home from "./pages/Home";

import { Navbar } from "./components/Navbar";
```

```
}

function App() {
        return (
                <div className="App">
                        <Navbar />
                        <Home />
                </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;
    background-color: #d6dbe4;
}

code {

  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',

    monospace;

}

@tailwind base;

@tailwind components;

@tailwind utilities;
```

GITHUB LINK : https://github.com/Kabilanmkk/Block-chain.git