B.M.S. COLLEGE OF ENGINEERING

(Autonomous College under VTU, Approved by AICTE, Accredited by NAAC)

MASTER OF COMPUTER APPLICATIONS

(Accredited by NBA for 5 years 2019 - 2024)

Report is submitted for the fulfillment of the AAT Work in the subject

“Block Chain Technology”

(22MCA3PEBC)

Of

III Semester

MCA

Topic: Voting Management System Using Blockchain

By

Name of Student:	**Register Number:**
Sree Poojitha Varma	1BM22MC006
N Sahana	1BM22MC022
Sahana M	1BM22MC040

Under the Guidance

R V Raghavendra Rao

(Assistant Professor)

Table of Content

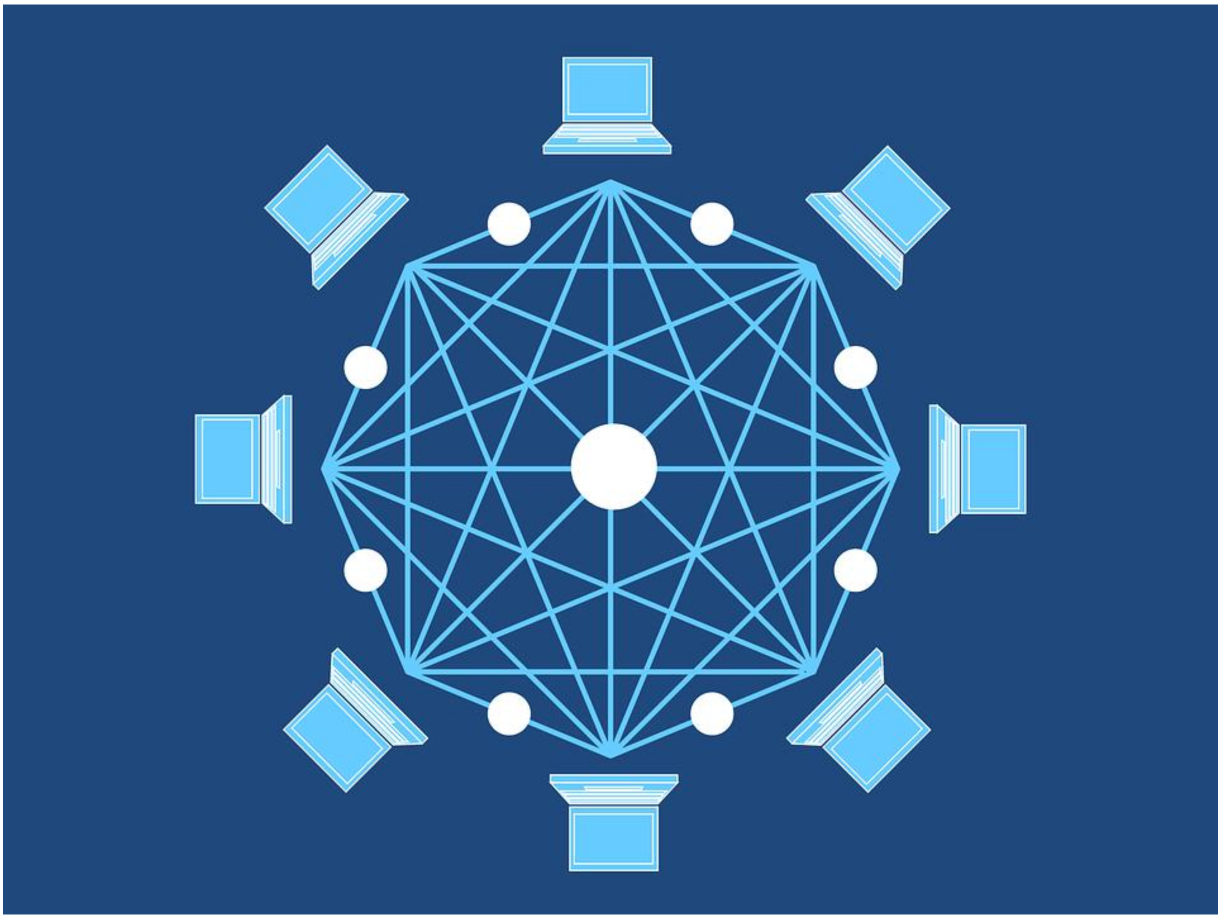
Introduction

A decentralized, distributed ledger technology.

Data is stored in blocks that are linked together in a chronological chain.

Each block contains a cryptographic hash of the previous block, ensuring immutability and tamper-resistance.

Decentralization



Objectives

Secure Authentication:

- User authentication using cryptographic methods

Transparent Voting Process:

- Every vote is recorded on the blockchain
- Immutable and tamper-proof

Real-time Results

- Instantaneous tallying of votes

Accessibility:

- Easy access through web interfaces

Auditable:

- Only admin can verify the integrity of the voting process

Why Blockchain For Voting?

Trust and Transparency: Traditional voting systems are vulnerable to various forms of fraud, including
tampering with ballots, voter impersonation, and hacking.

Security: Once a vote is recorded on the blockchain, it becomes practically impossible to alter or delete
without detection.

Immutable Record of Votes: Each vote is permanently recorded on the blockchain and cannot be altered or
deleted.

Decentralization: Blockchain technology decentralizes the voting process by distributing it across a network of nodes.Everyone can see the result of voting

Resistance to Tampering: Each block in the blockchain contains a cryptographic hash of the previous block, creating a chain of interconnected blocks.Any attempt to tamper with a block would require recalculating the hash of all subsequent blocks, making it computationally infeasible to alter past transactions without v detection

System Architecture

Overview of Technology Stack

- Solidity (Smart Contract Language)
- React JS (Frontend Framework)
- Typescript (Programming Language)
- MySQL (Database Management)
- Express (Backend Framework)
- Node.js (Server-side Runtime Environment)
- Web3 (Ethereum JavaScript API)
- Ganache

Components Of The System

Smart Contracts:

- Secure and transparent execution of voting logic
- Written in Solidity language

Frontend:

- User-friendly interface for voting operations
- Developed using React JS

Backend:

- Server-side logic for handling requests
- Built with Express and Node.js

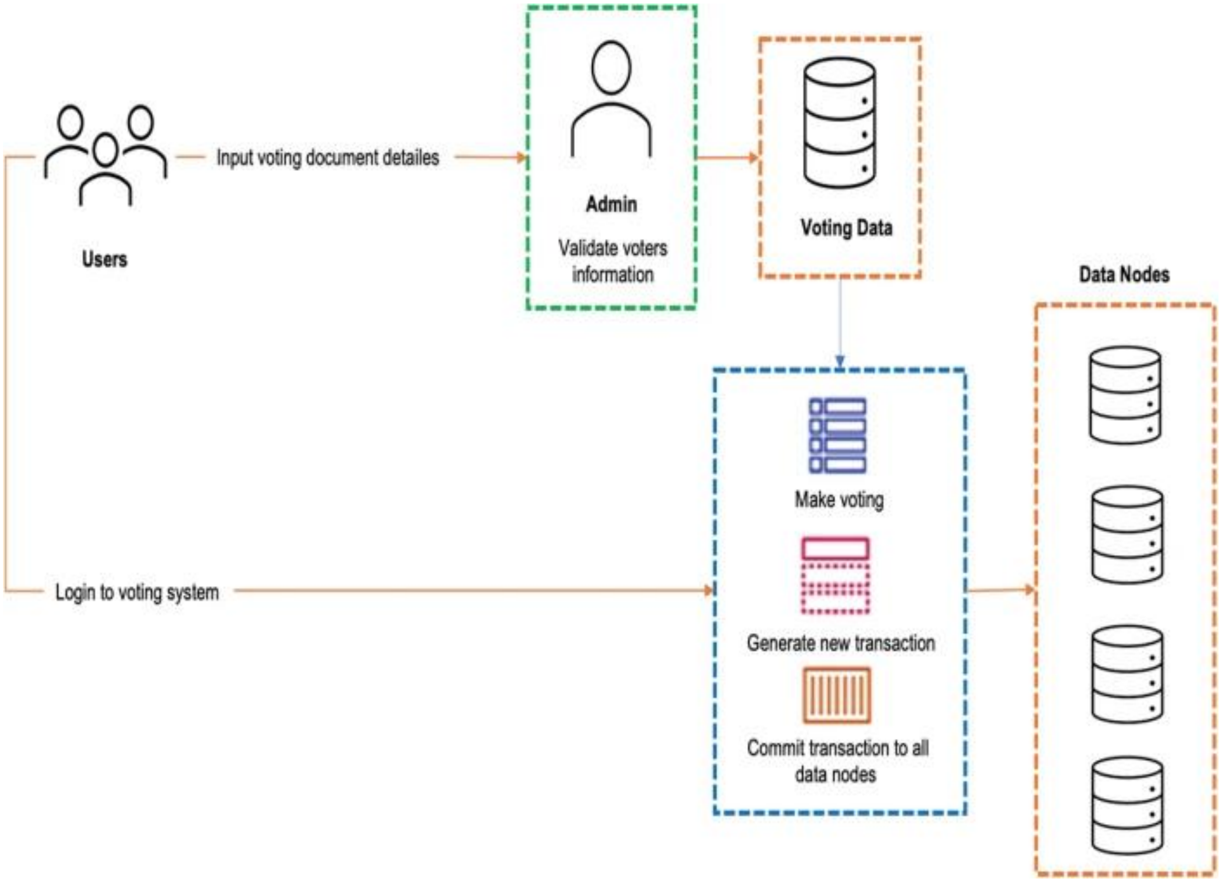
Database Management:

- MySQL used for storing non-sensitive data

Blockchain Integration:

- Web3 library for interaction with the Ethereum blockchain

Smart Contract


Users
Input voting document detailes
Admin
Validate voters information
Voting Data
Data Nodes
Make voting
Generate new transaction
Commit transaction to all data nodes
Login to voting system

Benefits Of The System

- Ensures fair and transparent elections
- Minimizes the risk of fraud and manipulation
- Increases voter participation through accessibility
- Simplifies the auditing process

Working

Solidity (Smart Contract Language):

- we will write smart contracts in Solidity to manage the voting process on the Ethereum blockchain. These contracts will handle tasks such as creating and closing polls, casting votes, and tallying results

React JS (Frontend Framework):

- Use React JS to build a user-friendly frontend interface for your voting system. This interface will allow users to view available polls, cast their votes, and see the results

Typescript (Programming Language):

- TypeScript can be used to write both frontend and backend code, providing type safety and scalability to your project

MySQL (Database Management):

- Although Ethereum is a blockchain-based solution and data storage primarily occurs on-chain, you might still need a traditional database like MySQL to store supplementary information such as user profiles, historical data, or metadata related to the voting process

Node.js (Server-side Runtime Environment):

- Node.js will be the runtime environment for your backend server, allowing you to write server-side code using JavaScript

Web3 (Ethereum JavaScript API):

- Web3.js is a JavaScript library that provides an interface for interacting with the Ethereum blockchain. we will use it in our backend server to communicate with Ethereum nodes, deploy smart contracts, and send transactions

Ganache:

- Ganache is a personal Ethereum blockchain that you can use for development and testing purposes. It allows you to deploy contracts, mine blocks instantly, and inspect state changes, all in a controlled environment

Working Procedure:

Contract Deployment:

- Deploy our smart contracts onto the Ethereum blockchain using tools like Truffle . These contracts will define the logic of your voting system, including creating polls, casting votes, and calculating results.

Frontend Development:

- Build the frontend interface using React JS. Users should be able to browse available polls, cast their votes, and view the results in real-time.

Backend Development:

- Set up an Express.js server to handle requests from the frontend and interact with the Ethereum blockchain using Web3.js. This server will authenticate users, validate their votes, and fetch data from the blockchain as needed.

Database Integration:

- Integrate MySQL with your backend server to store supplementary data such as user profiles or voting records.

Testing and Deployment:

- Test our application thoroughly, including both frontend and backend components. Once everything is working as expected, deploy your frontend to a web hosting service and your backend to a server provider.

User Interaction:

- Users can access the voting system through the frontend interface. They can browse available polls, cast their votes securely, and view the results in real-time.

Security and Scalability:

- Implement security measures such as user authentication and authorization to ensure the integrity of the voting process. Additionally, consider optimizing our application for scalability to handle a large number of users and votes.

Some Commands

1. create user bbvs@localhost identified with mysql_native_password by 'Password00$$' ;

2. grant all privileges on . to bbvs@localhost ;

3. create database bbvs;

4. truffle migrate

5. truffle migrate --reset

6. npm run typeorm migration:run

7. env file

   ```
   ACCESS_TOKEN_SECRET=976a66a5bd23b2050019f380c4decbbefdf8ff91cf502c68a3fe1ced91d7448cc54ce6c847657d53294e40889cef5bd996ec5b0fefc1f56270e06990657eeb6e

   REFRESH_TOKEN_SECRET=5f567afa6406225c4a759daae77e07146eca5df8149353a844fa9ab67fba22780cb4baa5ea508214934531a6f35e67e96f16a0328559111c597856c660f177c2
   ```

You must have pre installed:

1. **Node js**
   https://nodejs.org/en/

1. **Truffle**
   https://trufflesuite.com/docs/truffle...

1. **MySQL**
   https://www.mysql.com/

code:

contract file

election .sol

```
//SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.4.22 <0.9.0;

contract Election {
   mapping(address => bool) admins;
   string name; // name of the election. example: for president
   string description; // description of the election
   bool started;
   bool ended;

   constructor() {
      admins[msg.sender] = true;
      started = false;
      ended = false;
   }

   modifier onlyAdmin() {
      //require(admins[msg.sender] == true, "Only Admin");
      _;
   }

   function addAdmin(address _address) public onlyAdmin {
      admins[_address] = true;
   }
```

```
    /****************************CANDIDATES
SECTION****************************/

    struct Candidate {
        string name;
        string info;
        bool exists;
    }
    mapping(string => Candidate) public candidates;
    string[] candidateNames;

    function addCandidate(string memory _candidateName, string
memory _info)
        public
        onlyAdmin
    {
        Candidate memory newCandidate = Candidate({
            name: _candidateName,
            info: _info,
            exists: true
        });

        candidates[_candidateName] = newCandidate;
        candidateNames.push(_candidateName);
    }

    function getCandidates() public view returns (string[] memory) {
        return candidateNames;
    }

    /****************************CANDIDATES
SECTION****************************/

    /****************************ELECTION
SECTION****************************/

    function setElectionDetails(string memory _name, string memory
_description)
        public
        onlyAdmin
    {
        name = _name;
        description = _description;
        started = true;
        ended = false;
    }
```

```
        function getElectionName() public view returns (string memory) {
            return name;
        }

        function getElectionDescription() public view returns (string memory)
{
            return description;
        }

        function getTotalCandidates() public view returns (uint256) {
            return candidateNames.length;
        }

        /*****************************ELECTION
SECTION*****************************/

        /*****************************VOTER
SECTION*****************************/

        struct Vote {
            address voterAddress;
            string voterId;
            string voterName;
            string candidate;
        }
        Vote[] votes;
        mapping(string => bool) public voterIds;
        string[] votersArray;

        function vote(
            string memory _voterId,
            string memory _voterName,
            string memory _candidateName
        ) public {
            require(started == true && ended == false);
            require(candidates[_candidateName].exists, "No such candidate");
            require(!voterIds[_voterId], "Already Voted");

            Vote memory newVote = Vote({
                voterAddress: msg.sender,
                voterId: _voterId,
                voterName: _voterName,
                candidate: _candidateName
            });
            votes.push(newVote);
            voterIds[_voterId] = true;
            votersArray.push(_voterId);
        }

        function getVoters() public view returns (string[] memory) {
            return votersArray;
        }

        /*****************************VOTER
SECTION*****************************/

        function getVotes() public view onlyAdmin returns (Vote[] memory) {
            return votes;
        }

        function getTotalVoter() public view returns (uint256) {
            return votersArray.length;
        }

        function endElection() public onlyAdmin {
            require(started == true && ended == false);

            started = true;
            ended = true;
        }

        function resetElection() public onlyAdmin {
            require(started == true && ended == true);

            for (uint32 i = 0; i < candidateNames.length; i++) {
                delete candidates[candidateNames[i]];
            }

            for (uint32 i = 0; i < votersArray.length; i++) {
                delete voterIds[votersArray[i]];
            }

            name = "";
            description = "";

            delete votes;
            delete candidateNames;
            delete votersArray;

            started = false;
            ended = false;
        }

        function getStatus() public view returns (string memory) {
            if (started == true && ended == true) {
                return "finished";
            }

            if (started == true && ended == false) {
                return "running";
            }

            return "not-started";
        }
    }
```

migration.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.9.0;

contract Migrations {
  address public owner = msg.sender;
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

migration file

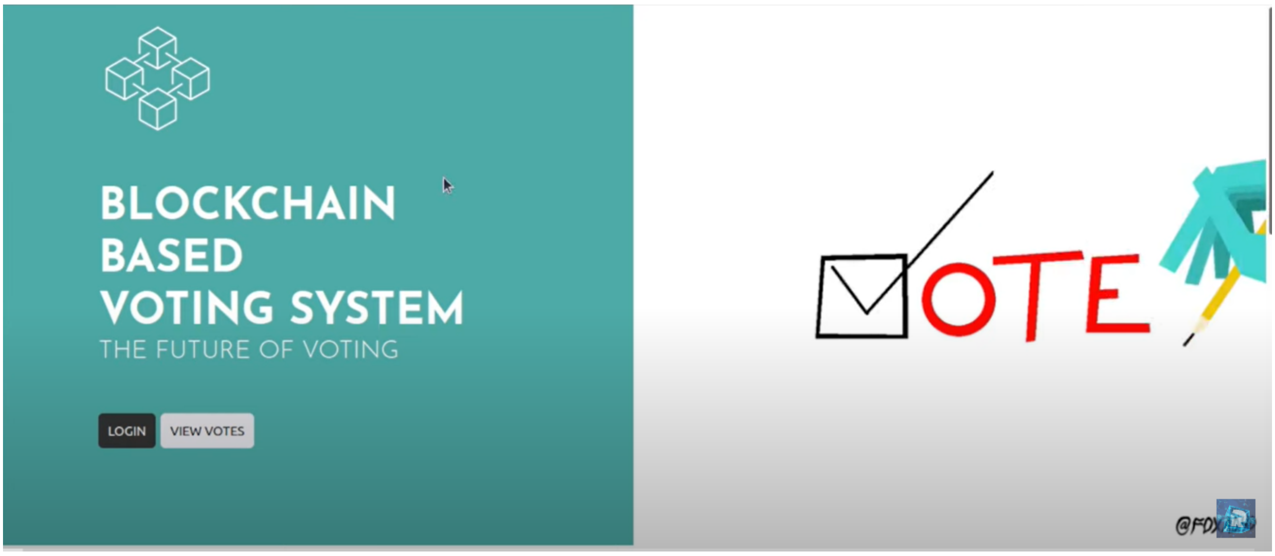
initial_migration.js

```
const Migrations = artifacts.require("Migrations");

module.exports = function (deployer) {
  deployer.deploy(Migrations);
};
```

election_migration.js

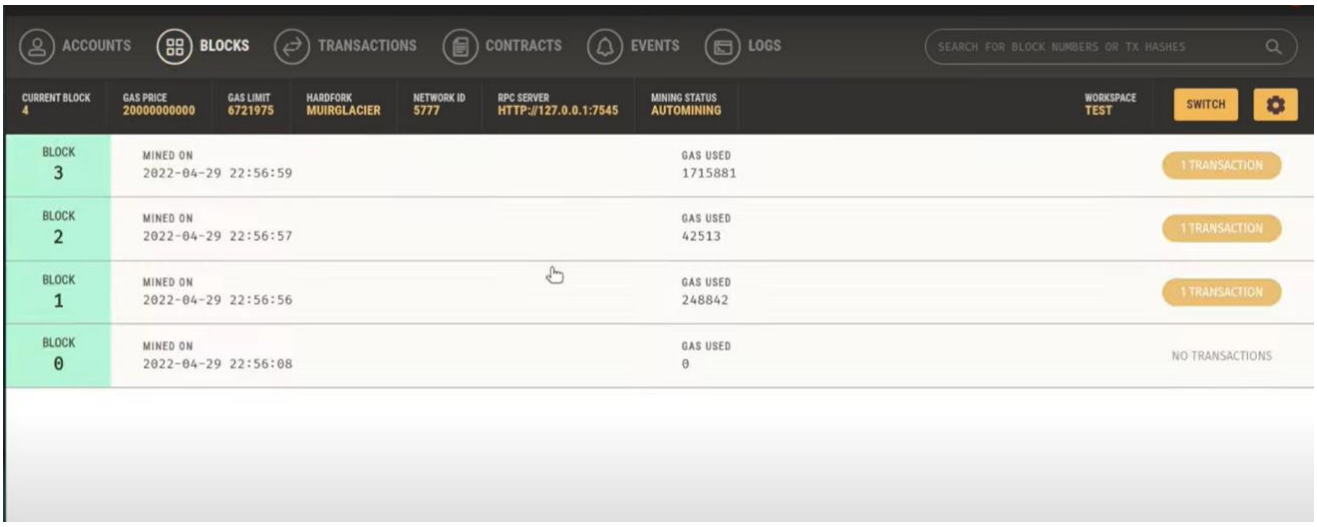
```
const Election = artifacts.require("Election");

module.exports = function (deployer) {
  deployer.deploy(Election);
};
```

Screenshots



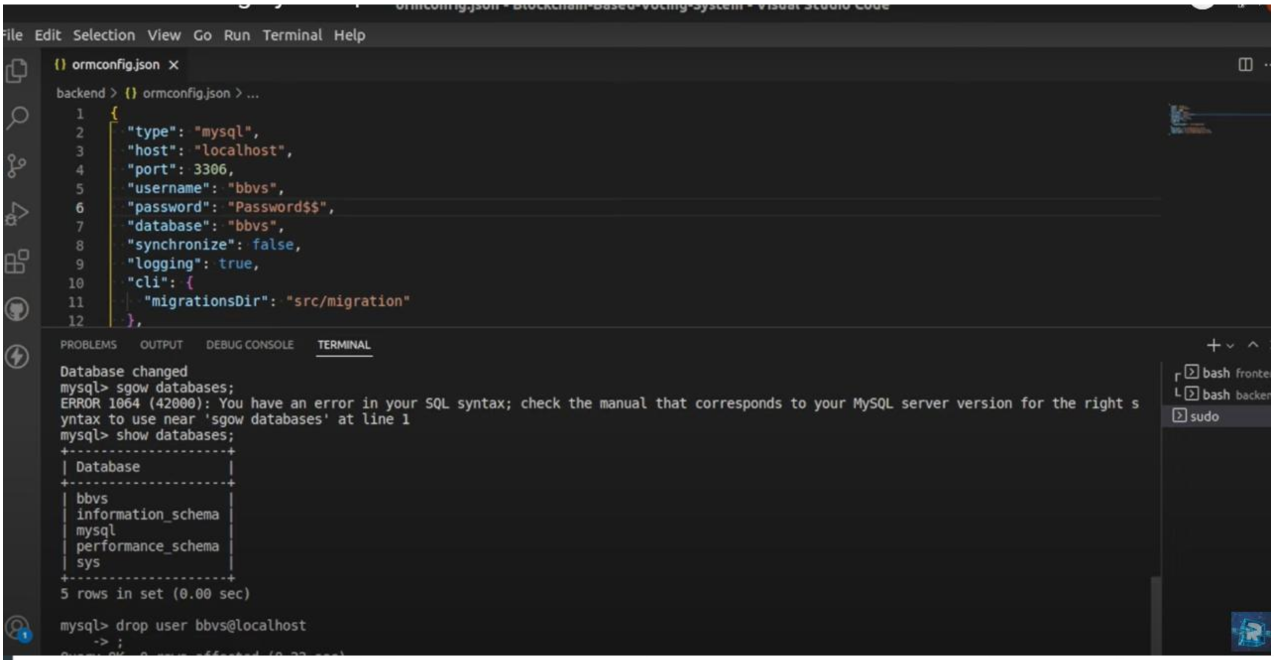
BLOCKCHAIN BASED VOTING SYSTEM

THE FUTURE OF VOTING



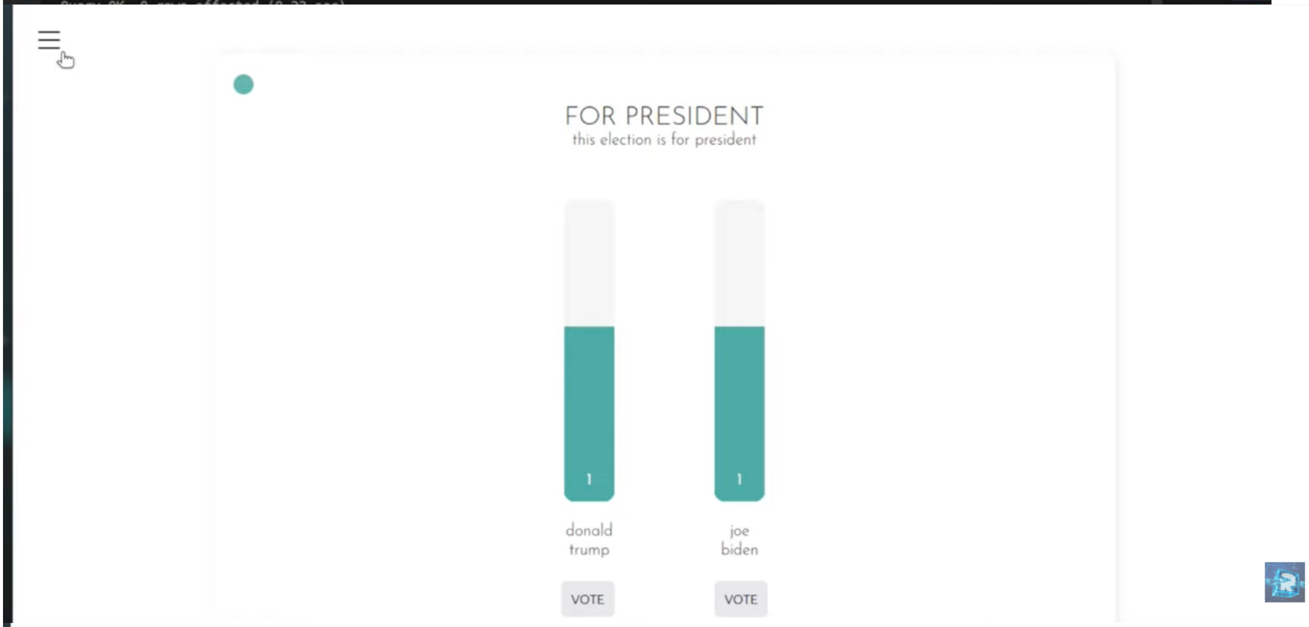
CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
4	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:7545	AUTOMINING	TEST

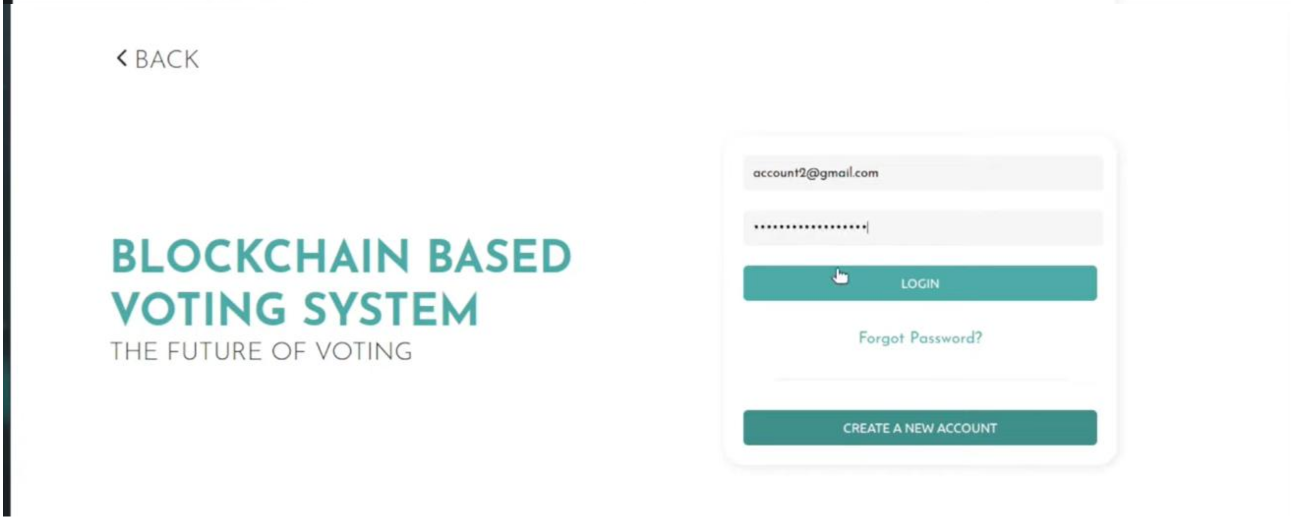
BLOCK	MINED ON	GAS USED	
3	2022-04-29 22:56:59	1715881	1 TRANSACTION
2	2022-04-29 22:56:57	42513	1 TRANSACTION
1	2022-04-29 22:56:56	248842	1 TRANSACTION
0	2022-04-29 22:56:08	0	NO TRANSACTIONS

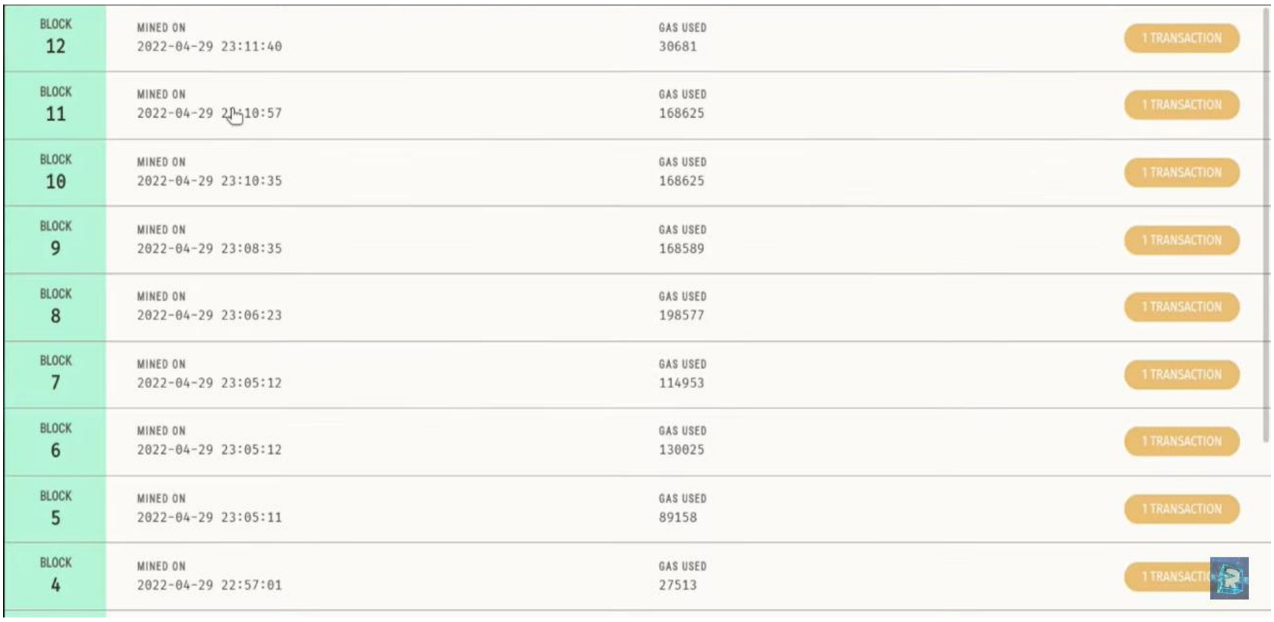

File Edit Selection View Go Run Terminal Help
ormconfig.json
backend > ormconfig.json > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash fronte
bash backer
sudo

```
{
  "type": "mysql",
  "host": "localhost",
  "port": 3306,
  "username": "bbvs",
  "password": "Password$$",
  "database": "bbvs",
  "synchronize": false,
  "logging": true,
  "cli": {
    "migrationsDir": "src/migration"
  },
```

```
Database changed
mysql> sgow databases;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right s
yntax to use near 'sgow databases' at line 1
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| bbvs               |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> drop user bbvs@localhost
    -> ;
```







BLOCK	MINED ON	GAS USED	
12	2022-04-29 23:11:40	30681	1 TRANSACTION
11	2022-04-29 23:10:57	168625	1 TRANSACTION
10	2022-04-29 23:10:35	168625	1 TRANSACTION
9	2022-04-29 23:08:35	168589	1 TRANSACTION
8	2022-04-29 23:06:23	198577	1 TRANSACTION
7	2022-04-29 23:05:12	114953	1 TRANSACTION
6	2022-04-29 23:05:12	130025	1 TRANSACTION
5	2022-04-29 23:05:11	89158	1 TRANSACTION
4	2022-04-29 22:57:01	27513	1 TRANSACTI

Conclusion

In conclusion, the development of a voting management system utilizing blockchain technology represents a significant advancement in ensuring the integrity, transparency, and accessibility of democratic processes. By leveraging a diverse set of technologies including Solidity, React JS, TypeScript, MySQL, Express, Node.js, and Web3, we have created a robust and secure platform for conducting elections.

References

https://www.typescriptlang.org/docs/

https://dev.mysql.com/doc/

https://nodejs.org/en/docs/

https://ethereum.org/en/developers/docs/

https://web3js.readthedocs.io/en/v1.5.2/