DEPARTMENT OF COMPUTER SCIENCE
Data Privacy and Securituy - CS 528

Project Report:

Decentralized Voting System

By:

Arvind Sai Dooda - A20553046
Aidhitha K - A20550006
Rahul Tatikonda - A20546868

Contents

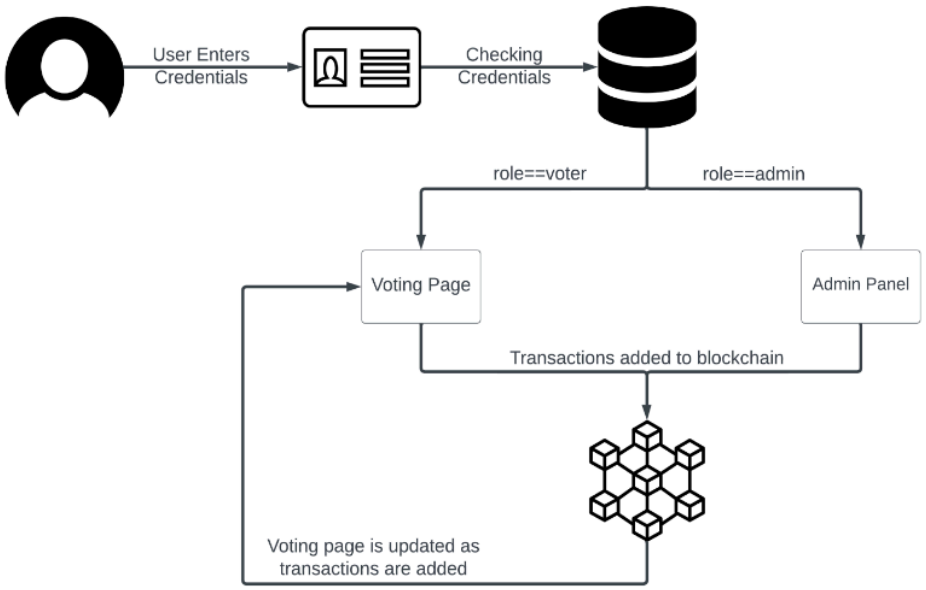
1 Team Description

The team in this project is composed of 3 members. Arvind, Aidhitha and Rahul are pursuing their master's degree in Cybersecurity as well as Computer Science at Illinois Tech starting from the fall semester 2023. In addition to that, Arvind has worked as a Software engineer Intern at Renault Nissan. Rahul has worked as an Cyber Risk and Regulatory Intern at PwC. Since two of the teammates are pursuing their Cybersecurity master's degree this project is a crucial one for us. And coming to Aidhitha she has done two internships during her undergrad as a Web Development Intern and Data Science Intern. The work for this project has been split equally between 3 of them and we have worked collaboratively to develop the ideas, frameworks, code and report.

2 Application

The proposed application is a decentralized voting system built on the Ethereum blockchain. It provides a secure and transparent platform for conducting elections, ensuring tamper-proof vote records, voter anonymity, and real-time result tracking. The application includes features such as voter authentication using JWT (JSON Web Tokens), candidate management through an admin panel, a user-friendly voting interface, and real-time result tracking.

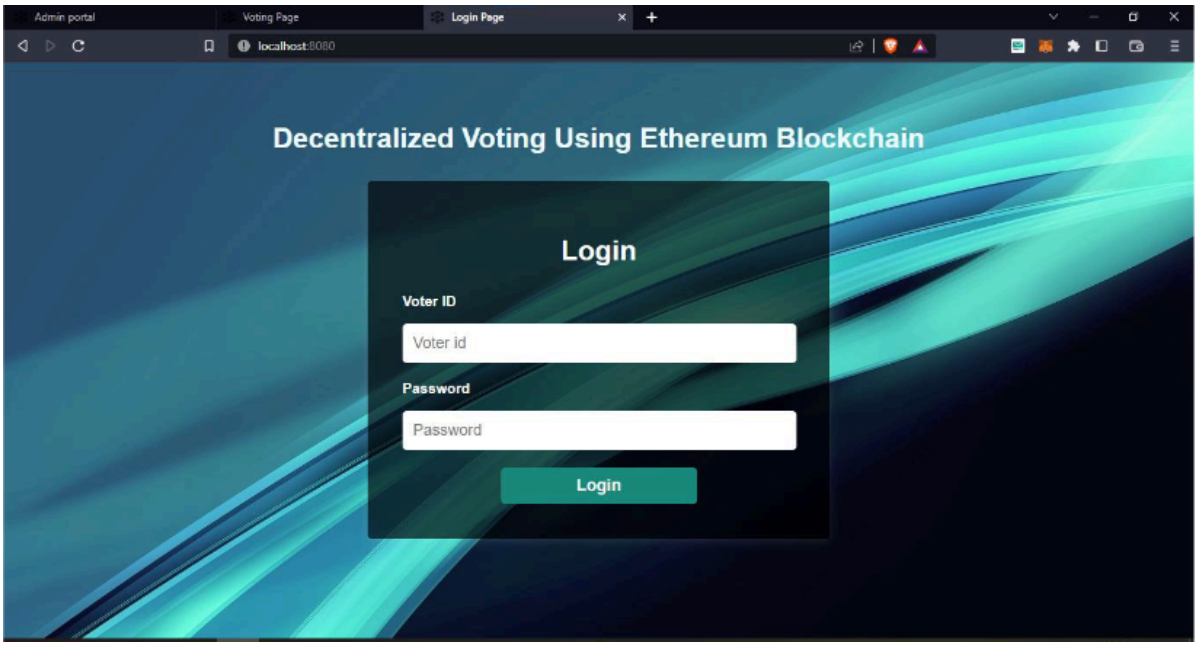
System Architecture

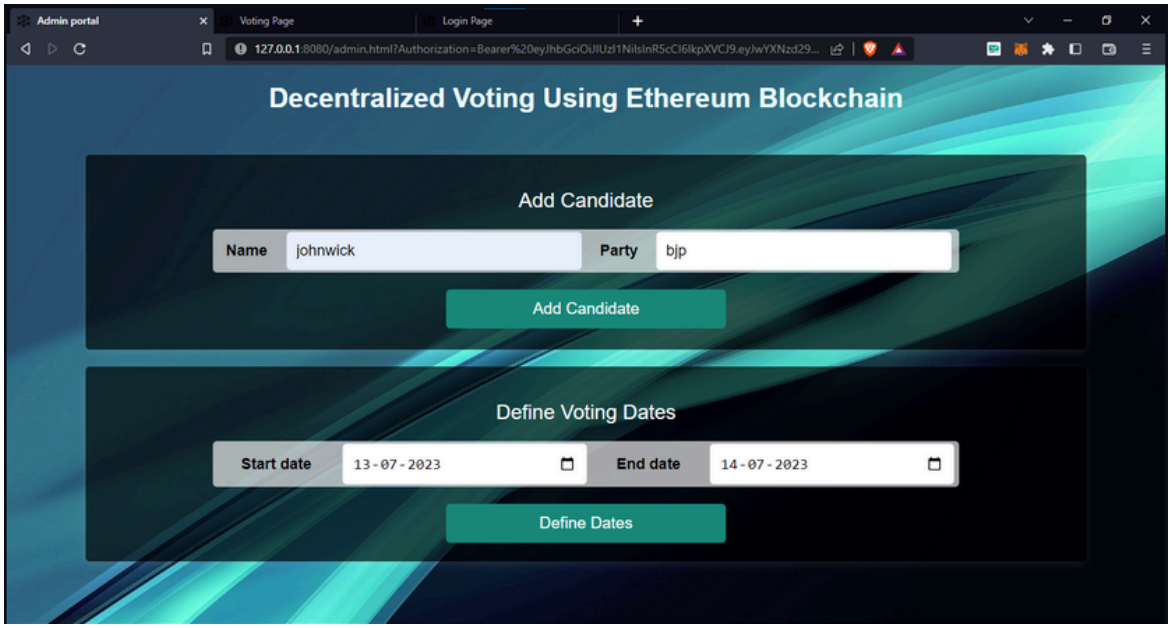


User enters the credentials (voter id & password) and they are matched with the database. If the match is found, the user is either redirected to admin page or voter page as per their role corresponding to the credentials in the database. Once the admin is logged in he/she can start the voting process by adding candidates and defining dates. Voters can vote once the voting process has been started. Once the voter has voted the transaction is recorded to the blockchain and the voting page is updated with real-time votes.

User Interface

You will be able to use something like the screenshot below as a Voters Login Page.





You can see something like the above interface once you login to the portal with the correct credentials.

3 Dataset

The system will generate and store voting data, including voter information (anonymized), candidate details, and vote records on the Ethereum blockchain. The voter information will be anonymized using techniques like hashing and encryption to protect voter privacy. The candidate details will include information such as names, parties, and other relevant information. The dataset is dynamic and get's stored onto the blockchain based on the people who vote.

The vote records will be securely stored on the Ethereum blockchain, ensuring transparency and immutability. The blockchain's decentralized nature and consensus mechanisms will prevent any tampering or manipulation of the voting data.

4 Privacy and Security Techniques

The project aims to address potential attack scenarios such as voter impersonation, tampering with vote records, denial of service attacks, and privacy concerns regarding voter anonymity. It leverages techniques like Ethereum blockchain for decentralization and transparency, Solidity smart contracts for implementing voting logic, and JWT

authentication for secure voter authorization. Since this is a project based on Ethereum Blockchain the ledger that we used to store the is tamper proof, the transaction that takes place happens on block chain, which is a dummy ethereum chain. We during our undergrad used to have a habit of investing in Cryptocurrencies, that's how we got familiar with Metamask and other stuff.

The project aims to address potential security threats like voter impersonation, vote record tampering, denial of service attacks, and voter privacy concerns. The techniques employed include:

1. Ethereum blockchain for decentralization and transparency
2. Solidity smart contracts for implementing the voting logic
3. JWT (JSON Web Tokens) for secure voter authentication

The use of blockchain technology and smart contracts ensures transparency and immutability of vote records, while JWT authentication provides a secure way to authorize voters.

5 Risk Management

Implementing a decentralized voting system based on blockchain technology introduces various risks that need to be carefully managed. While blockchain technology offers enhanced security and transparency, it is not immune to potential vulnerabilities and threats. Effective risk management strategies are crucial to ensure the integrity and reliability of the voting process. Here are some key risk management considerations for the decentralized voting system:

1. **Security risks:**

- **Blockchain vulnerabilities:** Although blockchain technology is generally considered secure, vulnerabilities in the underlying protocols, consensus mechanisms, or smart contract code could potentially be exploited by malicious actors. Regular security audits and proactive vulnerability management are essential.
- **Cryptographic risks:** The system relies heavily on cryptographic algorithms and key management for voter authentication, data encryption, and digital signatures. Proper key management practices, strong encryption algorithms, and regular updates to address potential cryptographic vulnerabilities should be implemented.
- **Distributed Denial of Service (DDoS) attacks:** As a decentralized system, the voting platform could be targeted by DDoS attacks aimed at disrupting the

network or hindering voter access. Robust DDoS mitigation strategies, load balancing, and redundancy measures should be in place.

2. **Operational risks:**

- **System availability and reliability:** Ensuring the continuous availability and reliability of the voting system during critical periods, such as elections, is crucial. Comprehensive disaster recovery and business continuity plans should be developed and tested regularly.
- **User errors and misconfigurations:** Improper usage, misconfigurations, or mistakes by administrators or voters could lead to unintended consequences or security breaches. Extensive user training, clear documentation, and user-friendly interfaces can help mitigate these risks.
- **Third-party dependencies:** The system may rely on third-party services, libraries, or infrastructure components. Thorough due diligence, monitoring, and contingency plans should be in place to manage risks associated with these dependencies.

3. **Legal and regulatory risks:**

- **Compliance with laws and regulations:** The decentralized voting system must comply with relevant laws, regulations, and policies related to elections, data privacy, and cybersecurity. Regular assessments and updates should be conducted to ensure ongoing compliance.
- **Liability and accountability:** Clear governance models, roles, and responsibilities should be established to address potential legal liabilities and ensure accountability in case of incidents or disputes.

4. **Risk management process:**

- Establish a risk management framework: Develop a comprehensive risk management framework that outlines the processes, roles, and responsibilities for identifying, assessing, mitigating, and monitoring risks throughout the system's lifecycle.
- Conduct regular risk assessments: Perform periodic risk assessments to identify and prioritize potential risks, considering factors such as likelihood, impact, and existing controls.

- Implement risk mitigation strategies: Based on the risk assessments, implement appropriate mitigation strategies, such as security controls, redundancy measures, incident response plans, and contingency plans.
- Continuous monitoring and review: Continuously monitor the system for potential risks, changes in the threat landscape, and the effectiveness of implemented controls. Regularly review and update the risk management strategies as needed.

By implementing a robust risk management approach, the decentralized voting system can proactively address potential threats, vulnerabilities, and operational challenges, ensuring the integrity, reliability, and trustworthiness of the voting process.

6 Code Implementation

In this section we will get to know in detail about the code structure, smart contracts (Migrations.sol and Voting.sol), JavaScript files (app.js and login.js), Python files (main.py for the database API), and other supporting files and directories.

Migrations.sol and Voting.sol are the core smart contracts responsible for managing the voting process and storing vote records on the Ethereum blockchain. The Migrations.sol contract handles the deployment and upgradation of the Voting contract, while the Voting.sol contract contains the logic for adding candidates, casting votes, and retrieving voting results.

migrations.sol

```
pragma solidity ^0.5.15;

contract Migrations {

address public owner;

uint public last_completed_migration;

modifier restricted() {

require(msg.sender == owner, "Access restricted to owner");

_;

}

constructor() public {
```

```
owner = msg.sender;
}
function setCompleted(uint completed) public restricted {
last_completed_migration = completed;
}
function upgrade(address new_address) public restricted {
Migrations upgraded = Migrations(new_address);
upgraded.setCompleted(last_completed_migration);
}
}
```

voting.sol

```
pragma solidity ^0.5.15;
contract Voting {
struct Candidate {
uint id;
string name;
string party;
uint voteCount;
}
mapping (uint => Candidate) public candidates;
mapping (address => bool) public voters;
uint public countCandidates;
uint256 public votingEnd;
uint256 public votingStart;
function addCandidate(string memory name, string memory party)
public returns(uint) {
```

```
countCandidates ++;
candidates[countCandidates] = Candidate(countCandidates,
name, party, 0);
return countCandidates;
}
function vote(uint candidateID) public {
require((votingStart <= now) && (votingEnd > now));
require(candidateID > 0 && candidateID <= countCandidates);
require(!voters[msg.sender]);
voters[msg.sender] = true;
candidates[candidateID].voteCount ++;
}
function checkVote() public view returns(bool){
return voters[msg.sender];
}
function getCountCandidates() public view returns(uint) {
return countCandidates;
}
function getCandidate(uint candidateID) public view returns
(uint,string memory, string memory,uint) {
return
(candidateID,candidates[candidateID].name,candidates[candidateID].party,
candidates[candidateID].voteCount);
}
function setDates(uint256 _startDate, uint256 _endDate) public{
require((votingEnd == 0) && (votingStart == 0) && (_startDate +
```

```
1000000 > now) && (_endDate > _startDate));
votingEnd = _endDate;
votingStart = _startDate;
}
function getDates() public view returns (uint256,uint256) {
return (votingStart,votingEnd);
}
}
```

app.js

```
const Web3 = require('web3');
const contract = require('@truffle/contract');
const votingArtifacts = require('../../build/contracts/Voting.json');
var VotingContract = contract(votingArtifacts)
window.App = {
eventStart: function() {
window.ethereum.request({ method: 'eth_requestAccounts' });
VotingContract.setProvider(window.ethereum)
VotingContract.defaults({from:
window.ethereum.selectedAddress,gas:6654755})
// Load account data
App.account = window.ethereum.selectedAddress;
$("#accountAddress").html("Your Account: " +
window.ethereum.selectedAddress);
VotingContract.deployed().then(function(instance){
instance.getCountCandidates().then(function(countCandidates){
$(document).ready(function(){
```

```
$('#addCandidate').click(function() {
var nameCandidate = $('#name').val();
var partyCandidate = $('#party').val();
instance.addCandidate(nameCandidate,partyCandidate).the
n(function(result){ })
});
$('#addDate').click(function(){
var startDate =
Date.parse(document.getElementById("startDate").value)/1000;
var endDate
= Date.parse(document.getElementById("endDate").value)/1000;
instance.setDates(startDate,endDate).then(function(rsl
t){
console.log("tarihler verildi");
});
});
instance.getDates().then(function(result){
var startDate = new Date(result[0]*1000);
var endDate = new Date(result[1]*1000);
$("#dates").text(
startDate.toDateString(("#DD#/#MM#/#YYYY#")) + " - " +
endDate.toDateString("#DD#/#MM#/#YYYY#"));
}).catch(function(err){
console.error("ERROR! " + err.message)
});
});
```

```
for (var i = 0; i < countCandidates; i++ ){

instance.getCandidate(i+1).then(function(data){

var id = data[0];

var name = data[1];

var party = data[2];

var voteCount = data[3];

var viewCandidates = `<tr><td> <input class="form-check-

input" type="radio" name="candidate" value="${id}" id=${id}>` + name +

"</td><td>" + party + "</td><td>" + voteCount + "</td></tr>"

$("#boxCandidate").append(viewCandidates)

})

}

window.countCandidates = countCandidates

});

instance.checkVote().then(function (voted) {

console.log(voted);

if(!voted) {

$("#voteButton").attr("disabled", false);

}

});

}).catch(function(err){

console.error("ERROR! " + err.message)

})

},

vote: function() {

var candidateID = $("input[name='candidate']:checked").val();
```

```
if (!candidateID) {
$("#msg").html("<p>Please vote for a candidate.</p>")
return
}
VotingContract.deployed().then(function(instance){
instance.vote(parseInt(candidateID)).then(function(result){
$("#voteButton").attr("disabled", true);
$("#msg").html("<p>Voted</p>");
window.location.reload(1);
})
}).catch(function(err){
console.error("ERROR! " + err.message)
})
}
}
window.addEventListener("load", function() {
if (typeof web3 !== "undefined") {
console.warn("Using web3 detected from external source like
Metamask")
window.eth = new Web3(window.ethereum)
} else {
console.warn("No web3 detected. Falling back to
http://localhost:9545. You should remove this fallback when you deploy
live, as it's inherently insecure. Consider switching to Metamask for
deployment. More info here:
http://truffleframework.com/tutorials/truffle-and-metamask")
```

```
window.eth = new Web3(new
Web3.providers.HttpProvider("http://127.0.0.1:9545"))
}
window.App.eventStart()
}
```

login.js

```
const loginForm = document.getElementById('loginForm');
loginForm.addEventListener('submit', (event) => {
event.preventDefault();
const voter_id = document.getElementById('voter-id').value;
const password = document.getElementById('password').value;
const token = voter_id;
const headers = {
'method': "GET",
'Authorization': `Bearer ${token}`,
};
fetch(`http://127.0.0.1:8000/login?voter_id=${voter_id}&password=${pas
sword}`, { headers })
.then(response => {
if (response.ok) {
return response.json();
} else {
throw new Error('Login failed');
}
})
.then(data => {
```

```
if (data.role === 'admin') {
console.log(data.role)
localStorage.setItem('jwtTokenAdmin', data.token);
window.location.replace(`http://127.0.0.1:8080/admin.html?Authoriz
ation=Bearer ${localStorage.getItem('jwtTokenAdmin')}`);
} else if (data.role === 'user'){
localStorage.setItem('jwtTokenVoter', data.token);
window.location.replace(`http://127.0.0.1:8080/index.html?Authoriz
ation=Bearer ${localStorage.getItem('jwtTokenVoter')}`);
}
})
.catch(error => {
console.error('Login failed:', error.message);
});
});
```

main.py

```
# Import required modules
import dotenv
import os
import mysql.connector
from fastapi import FastAPI, HTTPException, status, Request
from fastapi.middleware.cors import CORSMiddleware
from fastapi.encoders import jsonable_encoder
from mysql.connector import errorcode
import jwt
# Loading the environment variables
```

```
dotenv.load_dotenv()

# Initialize the todoapi app

app = FastAPI()

# Define the allowed origins for CORS

origins = [

"http://localhost:8080",

"http://127.0.0.1:8080",

]

# Add CORS middleware

app.add_middleware(

CORSMiddleware,

allow_origins=origins,

allow_credentials=True,

allow_methods=["*"],

allow_headers=["*"],

)

# Connect to the MySQL database

try:

cnx = mysql.connector.connect(

user=os.environ['MYSQL_USER'],

password=os.environ['MYSQL_PASSWORD'],

host=os.environ['MYSQL_HOST'],

database=os.environ['MYSQL_DB'],

)

cursor = cnx.cursor()

except mysql.connector.Error as err:
```

```
if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
print("Something is wrong with your user name or password")
elif err.errno == errorcode.ER_BAD_DB_ERROR:
print("Database does not exist")
else:
print(err)
# Define the authentication middleware
async def authenticate(request: Request):
try:
api_key = request.headers.get('authorization').replace("Bearer
", "")
cursor.execute("SELECT * FROM voters WHERE voter_id = %s",
(api_key,))
if api_key not in [row[0] for row in cursor.fetchall()]:
raise HTTPException(
status_code=status.HTTP_401_UNAUTHORIZED,
detail="Forbidden"
)
except:
raise HTTPException(
status_code=status.HTTP_401_UNAUTHORIZED,
detail="Forbidden"
)
# Define the POST endpoint for login
@app.get("/login")
async def login(request: Request, voter_id: str, password: str):
```

```
await authenticate(request)

role = await get_role(voter_id, password)

# Assuming authentication is successful, generate a token

token = jwt.encode({'password': password, 'voter_id': voter_id,

'role': role}, os.environ['SECRET_KEY'], algorithm='HS256')

return {'token': token, 'role': role}

# Replace 'admin' with the actual role based on authentication

async def get_role(voter_id, password):

try:

cursor.execute("SELECT role FROM voters WHERE voter_id = %s AND

password = %s", (voter_id, password,))

role = cursor.fetchone()

if role:

return role[0]

else:

raise HTTPException(

status_code=status.HTTP_401_UNAUTHORIZED,

detail="Invalid voter id or password"

)

except mysql.connector.Error as err:

print(err)

raise HTTPException(

status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,

detail="Database error"

)
```

package.json

```
{
"name": "decentralized-voting",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"author": "",
"license": "ISC",
"dependencies": {
"jsonwebtoken": "^9.0.0",
"@truffle/contract": "^4.6.18",
"browserify": "^17.0.0",
"dotenv": "^16.0.3",
"express": "^4.18.2",
"web3": "^1.9.0"
}
}
```

7 Synthetic Data Generation

Synthetic data generation can be a valuable technique for testing and evaluating the decentralized voting system, as well as for training machine learning models or conducting security analyses. In the context of this project, synthetic data could be generated to simulate various scenarios, such as voter behavior, voting patterns, and potential security threats.

One approach to generating synthetic data could involve using generative adversarial networks (GANs) or other deep learning techniques to create realistic voter profiles and voting records. GANs can be trained on existing voter data (with appropriate anonymization and privacy measures) to learn the underlying patterns and distributions. Once trained, the GAN can generate synthetic data that closely resembles the real data but does not contain any identifiable information about actual voters.

Another approach could be to use statistical techniques, such as Markov chain Monte Carlo (MCMC) methods or copula-based simulations, to generate synthetic data based on specified probability distributions and correlations. These methods can be particularly useful for generating synthetic data that captures the complex relationships between various variables, such as voter demographics, geographic locations, and voting preferences.

Synthetic data generation can also be combined with differential privacy techniques to ensure that the generated data does not leak any sensitive information about individual voters. Differential privacy involves adding controlled noise to the data, which preserves the overall statistical properties while protecting individual privacy.

Regardless of the specific technique used, it is essential to validate the synthetic data to ensure that it accurately reflects the real-world characteristics and patterns of interest. This validation can be done by comparing the synthetic data to real data (if available) or by involving domain experts to assess the plausibility and realism of the generated data.

Once generated, synthetic data can be used for various purposes, such as:

1. Testing and validation: Synthetic data can be used to test the decentralized voting system under different scenarios, including edge cases and potential security threats, without compromising the privacy of real voters.
2. Training machine learning models: Synthetic data can be used to train machine learning models for tasks such as voter behavior prediction, anomaly detection, or fraud detection, without the need for large amounts of real voter data.
3. Security analysis: Synthetic data can be used to simulate potential security attacks or vulnerabilities, allowing researchers and developers to investigate and mitigate these risks proactively.
4. Privacy-preserving data sharing: Synthetic data can be shared with third parties, such as researchers or auditors, without compromising the privacy of real voters, enabling collaboration and transparency while maintaining data confidentiality.

Our model is able to defend against tampering since it's built on blockchain technology. The data that we

Overall, synthetic data generation can be a powerful tool for enhancing the security, robustness, and transparency of the decentralized voting system, while also enabling data-driven analyses and collaborations without compromising voter privacy.

8 Testing

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. It includes a set of techniques and methods to identify defects, bugs, performance issues and providing a reliable and quality product. The goal is to identify issues as early as possible and improve the overall quality of the system.

Types of Testing

Unit Testing

Unit testing is a type of testing that is used to evaluate the individual units or components of a software system. This type of testing helps ensure that each unit or component of the system is working correctly and is able to perform its intended function.

Integration Testing

Integration testing is a type of testing that is used to evaluate how well the different units or components of a software system work together. This type of testing helps to identify and resolve issues related to compatibility, performance, and data flow between the different units or components.

Functional Testing

Functional testing is a type of testing that is used to evaluate how well a system or software performs the specific functions or tasks that it is designed to perform. It is done by testing the system or software with various inputs and verifying that the outputs are correct. This type of testing ensures that the system or software is working as intended and is able to perform the functions it was designed to perform.

White Box Testing

White box testing, also known as structural testing or glass-box testing, is a type of testing that examines the internal structure and implementation of a software system. It involves testing the code itself and checking that it is functioning correctly and adhering to coding standards. This type of testing helps to identify and resolve issues related to logic, control flow, and data structures within the system.

Black Box Testing

Black box testing, also known as functional testing, is a type of testing that examines the external behavior and interfaces of a software system. It involves testing the system from the user's perspective, without looking at the internal structure or implementation, and checking that it is functioning correctly and meeting the requirements. This type of testing helps to identify and resolve issues related to usability, compatibility, and performance.

9 Results

Test Case 1

Test Case No.	1
Test Type	Unit Test
Name of Test	Checking JWT Authorization
Test Case Description	The objective of this test case is to check jwt authorization.
Input	Login and Password
Expected Output	User should not be able to login without proper authorization.
Actual Output	User cannot access voting or admin page without authorization.
Result	Pass
Comments	Working properly.

Test Case 2

Test Case No.	2
Test Type	Functional Test
Name of Test	Verify user login
Test Case Description	The objective of this test case is to verify that user can login to the voting portal.
Input	Voter_id and password
Expected Output	User must be able to login if credentials match the database, else unauthorized error is shown.
Actual Output	User is able to login with correct credentials only.
Result	Pass
Comments	Working properly.

Test Case 3

Test Case No.	3
Test Type	Unit Test
Name of Test	Verify candidate registration
Test Case Description	The objective of this test case is to verify that candidate can be registered by admin.
Input	Candidate name and party.
Expected Output	Registration transaction should be successful.
Actual Output	Registration transaction is successful.
Result	Pass
Comments	Working properly.

Test Case 4

Test Case No.	4
Test Type	Unit Test
Name of Test	Verify date registration
Test Case Description	The objective of this test case is to verify that date of voting can be specified by admin.
Input	Starting and ending date
Expected Output	Date transaction should be successful.
Actual Output	Date transaction is successful.
Result	Pass
Comments	Working properly.

Test Case 5

Test Case No.	5
Test Type	Functional Test
Name of Test	Verify voting
Test Case Description	The objective of this test case is to verify that voter is able to cast their vote.
Input	Select a candidate and click “Vote” button.
Expected Output	Vote transaction should be successful.
Actual Output	Vote transaction is successful.
Result	Pass
Comments	Working properly.

10 Summary

Decentralized Voting with Ethereum Blockchain offers a robust and transparent solution for secure elections. By leveraging blockchain technology, it ensures the integrity of votes and provides a tamper-proof platform. With continued enhancements, including improved user experience, scalability, and integration with other cutting-edge technologies, it has the potential to revolutionize the democratic process and empower citizens to participate in a trusted and efficient voting system. It represents a significant step towards building a more democratic and accountable society.

Future Enhancement:

In future iterations, the decentralized voting system can be enhanced by implementing additional features such as real-time vote counting, secure voter identification mechanisms, advanced data analytics for voter insights, and integration with emerging technologies like artificial intelligence and biometrics. These enhancements will further enhance the efficiency, security, and accessibility of the voting process, making it more inclusive and trustworthy.