SCHOOL OF INFORMATION, COMPUTER AND COMMUNICATION TECHNOLOGY(ICT), SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY, THAMMASAT UNIVERSITY

Voting smart contract technical Report

Capstone Project

For CSS486 Blockchain Development

Group Member:

Peemawat Phosrinak 6222770164

Theerapat Cheeppanich 6222772202

Waranya Cheeppanich 6222771535

Table of Contents

Chapter 1

Introduction

Main Problem

The Voting Smart Contract System Technical Report provides a comprehensive analysis of a cutting-edge voting system based on blockchain technology and smart contracts. Traditional voting systems often face challenges related to transparency, security, and efficiency. In response, this technical report explores the potential of blockchain technology and smart contracts to address these issues and revolutionize the voting process.

Objective

To provide a comprehensive understanding of the voting smart contract system and its potential in revolutionizing the voting process.
To explore the advantages of utilizing blockchain technology and smart contracts in voting systems, including transparency, immutability, and automation.
To present a detailed analysis of the system architecture, including the integration of smart contracts, blockchain network, and user interface.

Chapter 2

Secure Voting System

A Blockchain based on Voting System

The voting smart contract system is an innovative solution that leverages blockchain technology and smart contracts to improve the transparency, security, and efficiency of the voting process. It aims to address the limitations of traditional voting systems, such as potential fraud, lack of transparency, and inefficiency.

At its core, the system utilizes blockchain technology, which is a decentralized and immutable ledger that records all transactions and actions within the network. This ensures transparency and prevents tampering with voting data. Smart contracts, which are self-executing contracts with predefined rules and conditions, play a crucial role in automating and securing the voting process.

The main contribution of the proposed system are as follows:

1. **Blockchain Infrastructure:** The voting system is built on a blockchain network, such as Ethereum, which provides a decentralized and tamper-resistant platform for recording votes. Each vote is treated as a transaction and recorded on the blockchain, ensuring transparency and immutability.
2. **Smart Contracts:** Smart contracts are used to define the rules and conditions of the voting process. They contain the logic for voter registration, ballot creation, vote casting, and result tabulation. Smart contracts help automate and enforce the voting rules, eliminating the need for intermediaries and reducing the possibility of fraud.
3. **Voter Registration:** The voting system allows eligible voters to register their identities on the blockchain. Once registered, voters receive a unique cryptographic key or digital identity that enables them to cast their votes securely.
4. **Ballot Creation:** The system generates the digital ballot, which includes the list of candidates or options for voters to choose from. The ballot is cryptographically secured to prevent tampering.
5. **Vote Casting:** Voters can securely cast their votes by digitally signing their ballots using their cryptographic keys. These votes are then broadcasted to the blockchain network, where they are validated and added to the blockchain.

6. **Vote Counting and Tabulation:** The votes recorded on the blockchain are transparently counted and tabulated using smart contracts. The blockchain's consensus mechanism ensures that the final result is agreed upon by all network participants, making it highly resistant to manipulation.
7. **Auditing and Transparency:** The use of blockchain technology enables real-time auditing and transparency of the voting process. Anyone can verify the integrity of the votes recorded on the blockchain, ensuring that the results are accurate and trustworthy.

Ganache

Ganache is a development and testing tool used in Ethereum blockchain development. It provides a personal Ethereum network that developers can use to simulate and test their smart contracts, decentralized applications (dApps), and other Ethereum-related projects. Ganache allows developers to interact with the blockchain locally, providing them with a sandbox environment to deploy and test their code without using real Ether or interacting with the main Ethereum network. It is a popular choice among Ethereum developers for its simplicity and ease of use during the development and debugging process.

Key features of Ganache:

1. **Local Ethereum Network:** Ganache provides a local, in-memory Ethereum blockchain that developers can use for testing and development purposes. This allows developers to interact with the blockchain without connecting to the main Ethereum network.
2. **Smart Contract Deployment:** Ganache enables developers to deploy their smart contracts to the local network, making it easy to test and debug contract functionality before deploying them to the live Ethereum network.
3. **Account Management:** Ganache generates a set of predefined accounts with associated private keys for developers to use during testing. These accounts can be used for simulating transactions, testing contract interactions, and debugging Ethereum applications.
4. **Transaction Simulation:** Developers can use Ganache to simulate various types of transactions, such as token transfers, contract invocations, and contract deployments. This helps in testing the behavior and performance of smart contracts and dApps.

5. **Gas Price Customization:** Ganache allows developers to customize the gas price for transactions, which is helpful in simulating different network conditions and optimizing gas usage in Ethereum applications.
6. **Block Mining Control:** Developers can control the mining behavior of the Ganache network, such as the block interval and mining speed. This provides flexibility for testing time-dependent functionalities and evaluating the impact of mining on transaction confirmation times.
7. **Network and Account Snapshotting:** Ganache supports the capability to take snapshots of the local network and individual accounts at specific points in time. This allows developers to revert to a particular state during testing, enabling them to easily recreate scenarios and debug issues.

Flask

Flask is a lightweight web framework written in Python. It is designed to be simple, flexible, and easy to use, making it a popular choice for developing web applications and APIs. Flask provides the necessary tools and features to handle routing, request handling, and template rendering, allowing developers to build web applications quickly and efficiently.

Key features of Flask:

1. **Lightweight and Minimalistic:** Flask follows a "micro" framework approach, providing only the essentials for building web applications. It has a small core and relies on extensions to add additional functionalities as needed. This lightweight design allows developers to have more control and flexibility in building their applications.
2. **Routing:** Flask provides a simple and intuitive routing system that allows developers to map URLs to specific functions or views. This makes it easy to define the routes and handle HTTP requests such as GET, POST, and more.
3. **Templating Engine:** Flask includes a built-in Jinja2 templating engine, which enables developers to create dynamic HTML templates. Templating allows for the separation of presentation and logic, making it easier to maintain and update web pages.
4. **HTTP Request Handling:** Flask provides convenient methods for handling and processing HTTP requests. It supports request handling for parameters, cookies, sessions, file uploads, and more.
5. **Flask Extensions:** Flask has a rich ecosystem of extensions that can be easily integrated into applications. These extensions offer additional functionalities such as database integration, authentication, form handling, caching, and more. The modular nature of Flask allows developers to choose and add only the required extensions, keeping the application lightweight.

6. **Development Server:** Flask includes a built-in development server that makes it easy to test and debug applications during the development process. This server automatically reloads the application when changes are made, making the development workflow more efficient.
7. **Flask-WTF:** Flask integrates seamlessly with the Flask-WTF extension, which simplifies form handling and validation. It provides CSRF protection, form rendering, data validation, and other form-related utilities.
8. **Testing Support:** Flask provides a testing framework that allows developers to write unit tests for their web applications. This makes it easier to ensure the correctness and reliability of the application's behavior.

WEB3

Web3 refers to a collection of libraries and protocols that enable developers to interact with the Ethereum blockchain and build decentralized applications (dApps). It provides a programming interface that allows applications to connect to and interact with the Ethereum network.

Key features of Web3:

1. **Blockchain Interaction:** Web3 enables developers to interact with the Ethereum blockchain and perform various operations such as reading data from the blockchain, sending transactions, and deploying smart contracts. It provides APIs for querying blockchain state, retrieving transaction details, and accessing historical data.
2. **Smart Contract Interaction:** Web3 allows developers to interact with smart contracts deployed on the Ethereum blockchain. It provides methods for calling smart contract functions, reading data from contract state variables, and sending transactions to modify the contract state.
3. **Wallet Integration:** Web3 supports integration with Ethereum wallets like MetaMask. This enables users to securely sign transactions and interact with decentralized applications (dApps) using their wallet's private keys. Wallet integration enhances the user experience by simplifying transaction signing and providing a seamless connection to the Ethereum network.
4. **Event Listening:** Web3 allows developers to listen to events emitted by smart contracts on the Ethereum blockchain. This feature enables applications to react to specific events triggered by smart contract actions, such as token transfers or contract state changes.
5. **Transaction Management:** Web3 provides functionalities for managing transactions, including estimating gas costs, setting gas prices, and handling transaction confirmation and receipt. These features assist developers in optimizing gas usage, estimating transaction fees, and ensuring the reliability of transaction processing.

6. **Blockchain Network Connectivity:** Web3 supports connectivity to different Ethereum networks, including the mainnet, testnets, and private networks. It allows developers to choose the appropriate network for their application's needs and seamlessly switch between networks during development and deployment.
7. **Integration with Web Technologies:** Web3 can be integrated with other web technologies and frameworks, such as JavaScript frameworks like React or Vue.js, enabling developers to build user-friendly and interactive decentralized applications.
8. **Ethereum Name Service (ENS) Integration:** Web3 provides support for the Ethereum Name Service, which allows users to associate human-readable names with Ethereum addresses. This makes it easier for users to interact with dApps and send transactions without relying on complex Ethereum addresses.

Chapter 3

Design and Implementation

Requirements

User Registration:

- The platform should allow user to sign up by providing required information such as their username and password

Add Candidate Management

- The platform require only the owner of the contract for doing this process adding ID,Name ,Party,Campaign Message of the candidate

Start voting

- To initialize the vote the owner of the contract have to press the button in order for doing the vote for process(if we not start before doing voting process it will cause an error)

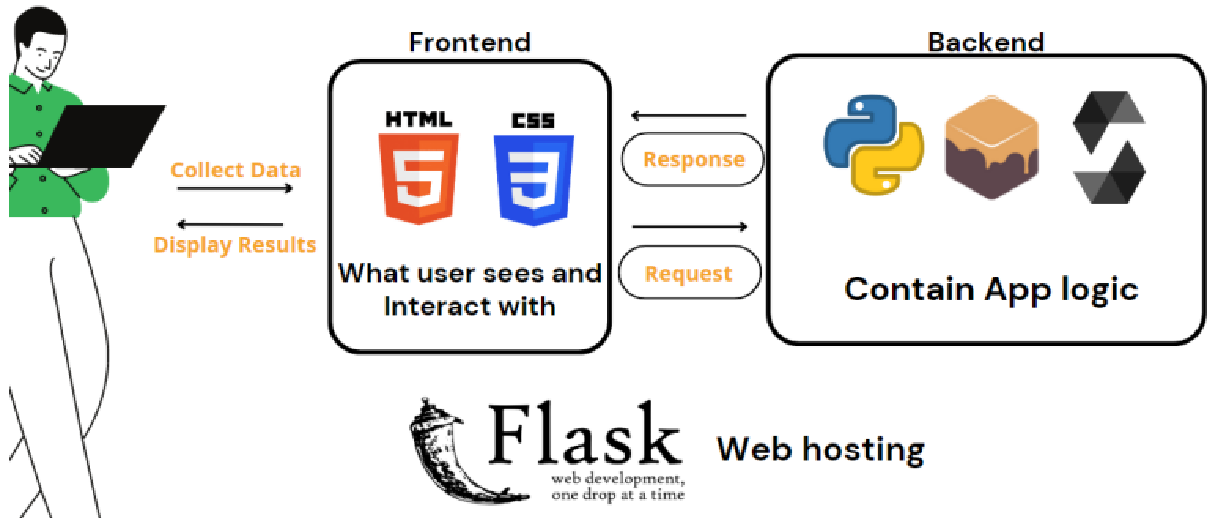
Voting Process

- The voting Process is for the voter to vote for candidate by make a thick symbol in the table everyone that register can vote once for one candidate .If the voter is already vote they can’t vote any more

Ending Vote

- The ending vote process requires the owner of the contract to do so in order to end the vote. After pressing the end vote it will go to show the result but only the owner of the contract can see this result data .

System Architecture



The architecture of the blockchain web application consists of three main components: the frontend, the backend, and the smart contracts.

The frontend, implemented using HTML and CSS, serves as the user interface of the web application. HTML structures the content, while CSS styles the presentation. Users interact with the frontend by inputting data and engaging with various elements.

The backend, powered by Python, Ganache, and Flask, acts as the brain of the web application. Python handles the core functionalities, while Ganache functions as a local Ethereum blockchain network for development and testing purposes. Flask serves as the web server and facilitates communication between the frontend and the blockchain network.

Smart contracts, written in Solidity, form another vital component. These self-executing contracts have predefined rules and conditions and define the business logic, data storage, and rules for the blockchain application.

In the interaction flow, users input data and interact with the frontend interface. The frontend captures the user input and initiates a request to the backend. The backend, implemented using Python and Flask, receives the user's data request, processes it, validates input, and prepares data for interaction with the blockchain network.

The backend then interacts with the local blockchain network (Ganache) using deployed Solidity smart contracts. It executes predefined actions by invoking functions in the smart contracts. Transactions recorded on the blockchain ensure transparency, immutability, and decentralization. Consensus mechanisms and cryptographic algorithms maintain data integrity, enhancing blockchain transparency and security.

Once a transaction is confirmed on the blockchain, the frontend updates the interface, reflecting the updated state of data. It provides feedback to the user, such as success messages or error notifications.

By combining HTML/CSS frontend, Python with Ganache and Flask in the backend, and Solidity smart contracts, a secure, transparent, and decentralized blockchain web application can be created.

Smart Contract

Structure

Candidate: Represents the information about a candidate (id, name, party, campaign message).
User: Represents the information about a user (username, password, first name, last name).

State Variable

owner: Stores the address of the contract owner.

state: Represents the current state of the voting process (Created, Voting, or Ended).

candidateList: An array that stores the names of the candidates.

candidateInfo: An array that stores the detailed information of the candidates.

votesReceived: A mapping that keeps track of the vote count for each candidate.

isVoted: A mapping that records whether an address has already voted.

users: A mapping to store user information.

userAddresses: An array to store user addresses.

Mapping

mapping(string => User) : This mapping is to map the register info of user Username with Username that user use in the login process

mapping(string => uint256) : This map is use map the candidate name with the vote count In order for function getVotecountForAllCandidates to use this for showing result

mapping(address => bool): This map is use for map associates Ethereum key with boolean value for checking in function IsVoted to check that is voter already vote or not

mapping(address => User) : This mapping associates an Ethereum address key with a value of type User struct. It is used to store user information based on the user's Ethereum address. Each user's address is mapped to their corresponding User struct containing their username, password, first name, and last name.

Function:

startVote(): Allows the contract owner to start the voting process.

endVote(): Allows the contract owner to end the voting process.

addCandidate(): Allows the contract owner to add a candidate.

voteForCandidate(): Allows a user to vote for a specific candidate.

getVoteCountsForAllCandidates(): Retrieves the vote count for all candidates after the voting process has ended.

registerUser(): Allows a user to register by providing their username, password, first name, and last name.

getUserCount(): Retrieves the total number of registered users.

getUserByIndex(): Retrieves the user information by providing the index.

getCandidate(): Retrieves the list of candidate names.

getCandidateInfo(): Retrieves the detailed information of all candidates.

get_user_info(): Retrieves the user information by providing the username.

Modifiers:

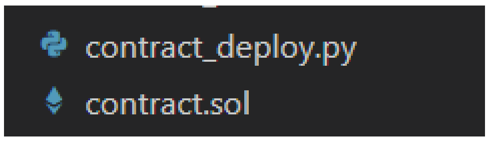
: **onlyOwner():** A modifier that restricts access to certain functions to the contract owner only.

User Interface and User Experience Design

Instruction manual

1.Getting started

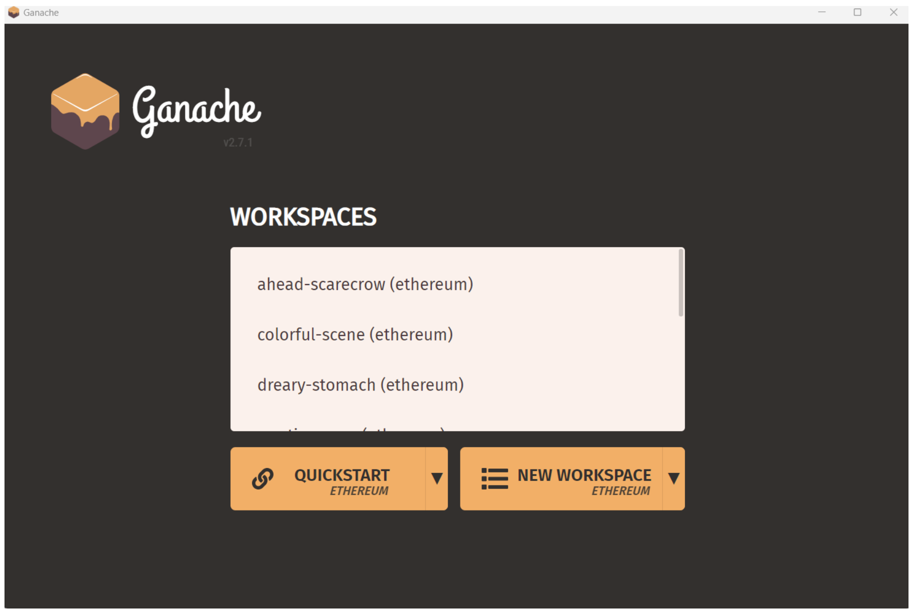
Open the folder and navigate to the smart contract directory. Look for these two files first.



contract_deploy.py
contract.sol

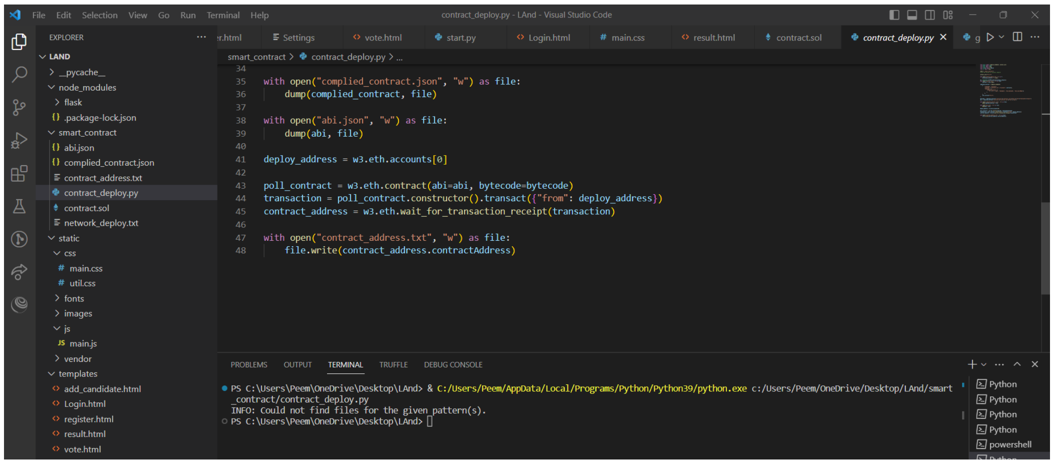
2.Open Ganache App (Download on internet)

Choose the Quick start Ethereum button.


Ganache
Ganache
v2.7.1
WORKSPACES
ahead-scarecrow (ethereum)
colorful-scene (ethereum)
dreary-stomach (ethereum)
QUICKSTART
ETHEREUM
NEW WORKSPACE
ETHEREUM

3.Deploy the contract

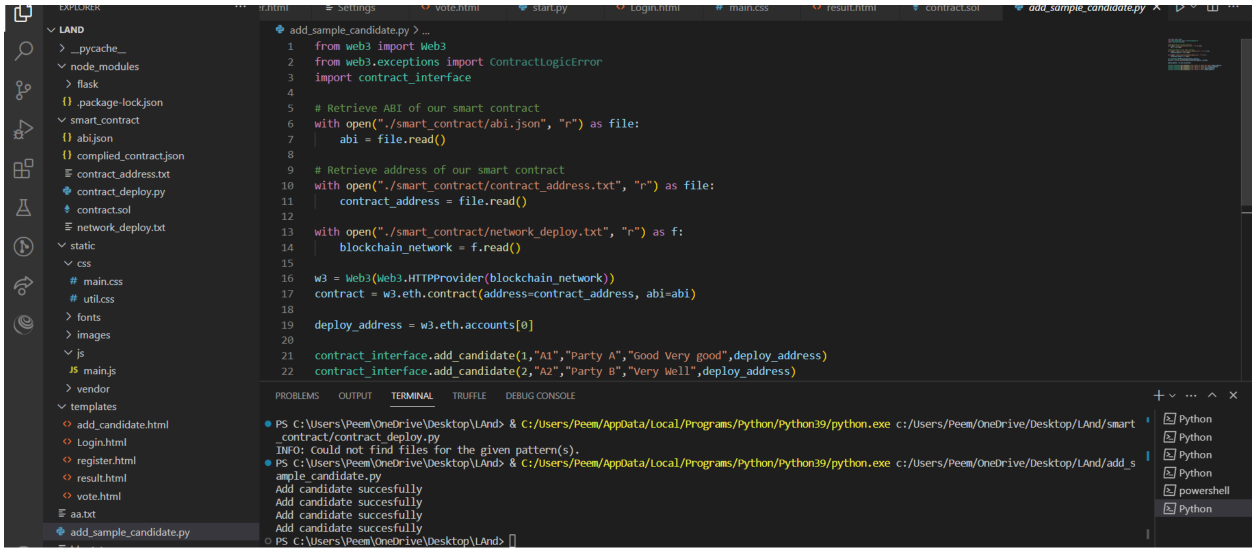
Open this file first and then press the run python file button (make sure you already open the Ganache).



```python
with open("complied_contract.json", "w") as file:
    dump(complied_contract, file)

with open("abi.json", "w") as file:
    dump(abi, file)

deploy_address = w3.eth.accounts[0]

poll_contract = w3.eth.contract(abi=abi, bytecode=bytecode)
transaction = poll_contract.constructor().transact({"from": deploy_address})
contract_address = w3.eth.wait_for_transaction_receipt(transaction)

with open("contract_address.txt", "w") as file:
    file.write(contract_address.contractAddress)
```

```
PS C:\Users\Peem\OneDrive\Desktop\LAnd> & C:/Users/Peem/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Peem/OneDrive/Desktop/LAnd/smart
_contract/contract_deploy.py
INFO: Could not find files for the given pattern(s).
PS C:\Users\Peem\OneDrive\Desktop\LAnd> 
```

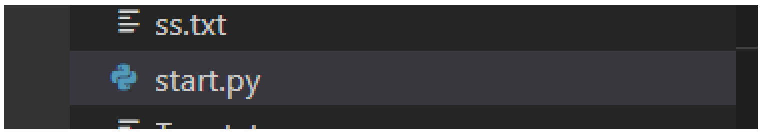
After that, go to the Sample file here in the picture then press the run button(this will add sample of candidate to the candidate list) If this contract is already connect with web app It should show the message “add candidate successfully” in the terminal



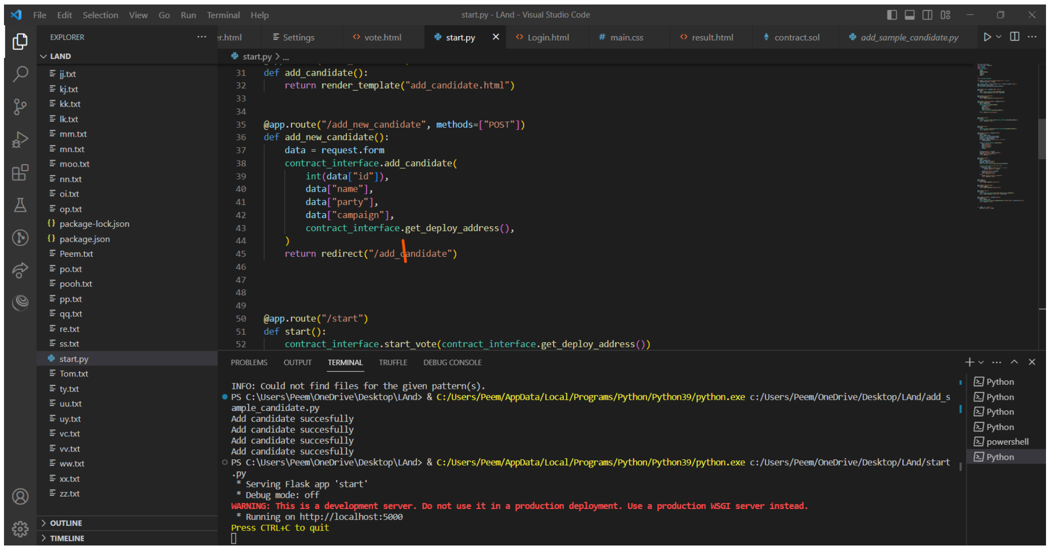
```python
from web3 import Web3
from web3.exceptions import ContractLogicError
import contract_interface

# Retrieve ABI of our smart contract
with open("./smart_contract/abi.json", "r") as file:
    abi = file.read()

# Retrieve address of our smart contract
with open("./smart_contract/contract_address.txt", "r") as file:
    contract_address = file.read()

with open("./smart_contract/network_deploy.txt", "r") as f:
    blockchain_network = f.read()

w3 = Web3(Web3.HTTPProvider(blockchain_network))
contract = w3.eth.contract(address=contract_address, abi=abi)

deploy_address = w3.eth.accounts[0]

contract_interface.add_candidate(1,"A1","Party A","Good Very good",deploy_address)
contract_interface.add_candidate(2,"A2","Party B","Very Well",deploy_address)
```

```
PS C:\Users\Peem\OneDrive\Desktop\LAnd> & C:/Users/Peem/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Peem/OneDrive/Desktop/LAnd/smart
_contract/contract_deploy.py
INFO: Could not find files for the given pattern(s).
PS C:\Users\Peem\OneDrive\Desktop\LAnd> & C:/Users/Peem/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Peem/OneDrive/Desktop/LAnd/add_s
ample_candidate.py
Add candidate succesfully
Add candidate succesfully
Add candidate succesfully
Add candidate succesfully
PS C:\Users\Peem\OneDrive\Desktop\LAnd> 
```

Go to the file start.py



Then, press the run button in the terminal, it should provide the localhost link. Use this localhost link and open it in Google Chrome.

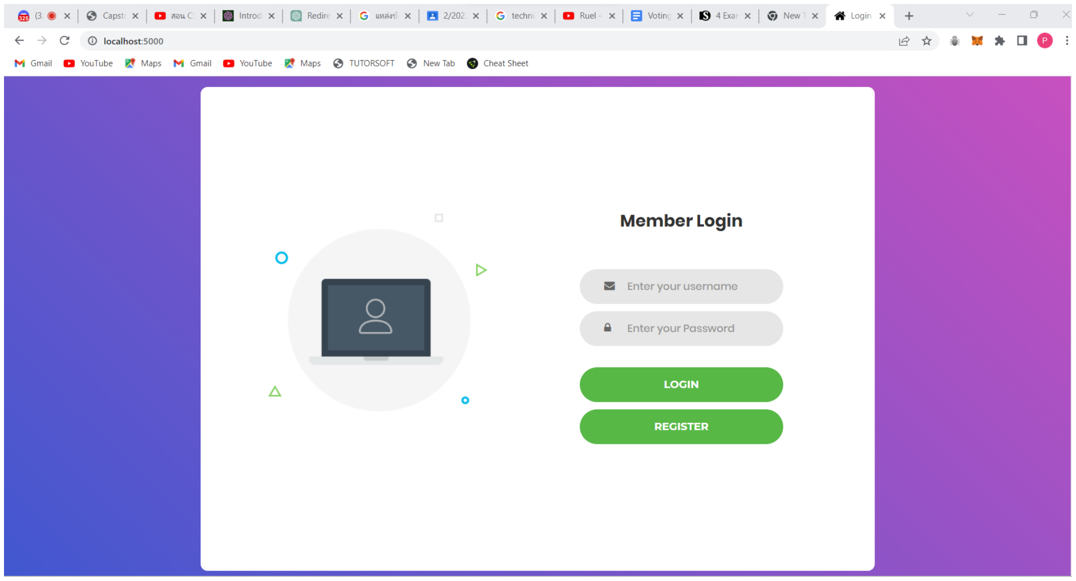


```python
def add_candidate():
    return render_template("add_candidate.html")


@app.route("/add_new_candidate", methods=["POST"])
def add_new_candidate():
    data = request.form
    contract_interface.add_candidate(
        int(data["id"]),
        data["name"],
        data["party"],
        data["campaign"],
        contract_interface.get_deploy_address(),
    )
    return redirect("/add_candidate")




@app.route("/start")
def start():
    contract_interface.start_vote(contract_interface.get_deploy_address())
```

```
INFO: Could not find files for the given pattern(s).
PS C:\Users\Peem\OneDrive\Desktop\LAnd> & C:/Users/Peem/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Peem/OneDrive/Desktop/LAnd/add_s
ample_candidate.py
Add candidate succesfully
Add candidate succesfully
Add candidate succesfully
Add candidate succesfully
PS C:\Users\Peem\OneDrive\Desktop\LAnd> & C:/Users/Peem/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Peem/OneDrive/Desktop/LAnd/start
.py
 * Serving Flask app 'start'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://localhost:5000
Press CTRL+C to quit
```

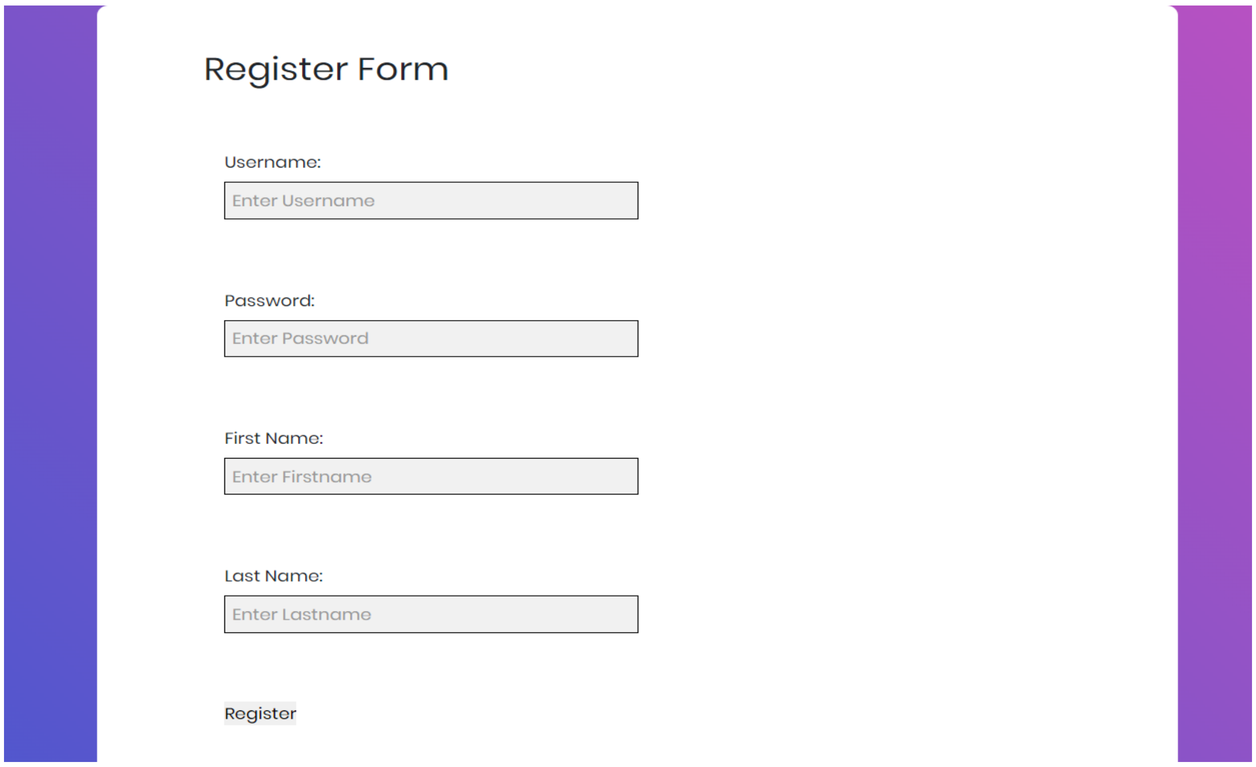
Home Page

First, go to the Login Page. Then, press the register button to create an account.



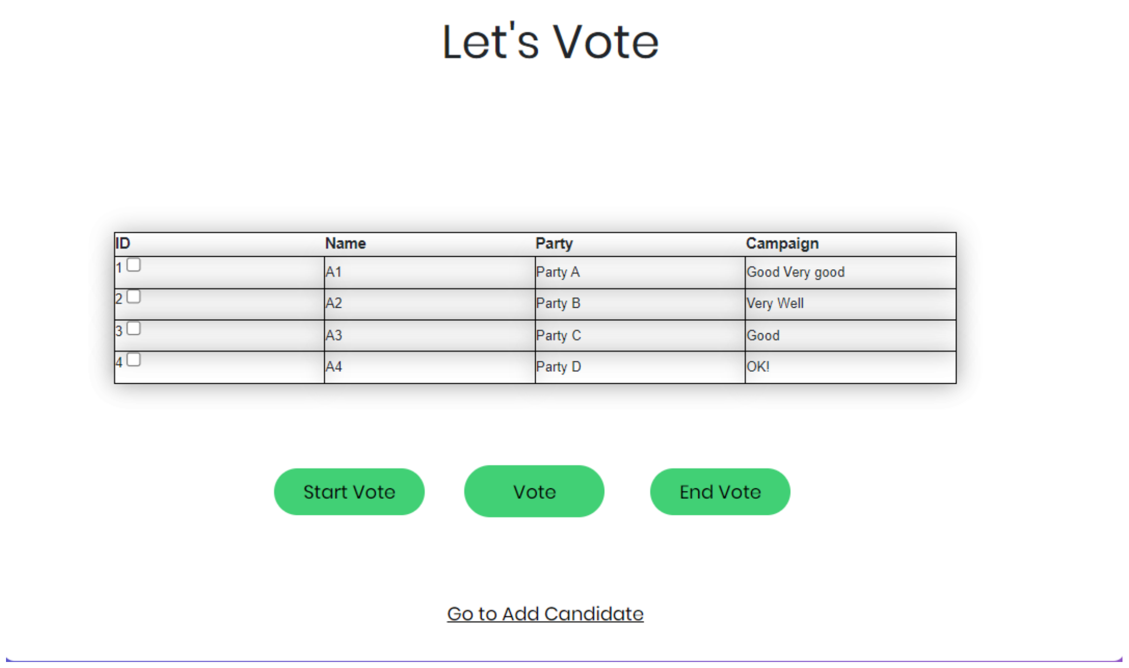
Register Page

Fill in the information and then press the Register button.



Vote Page

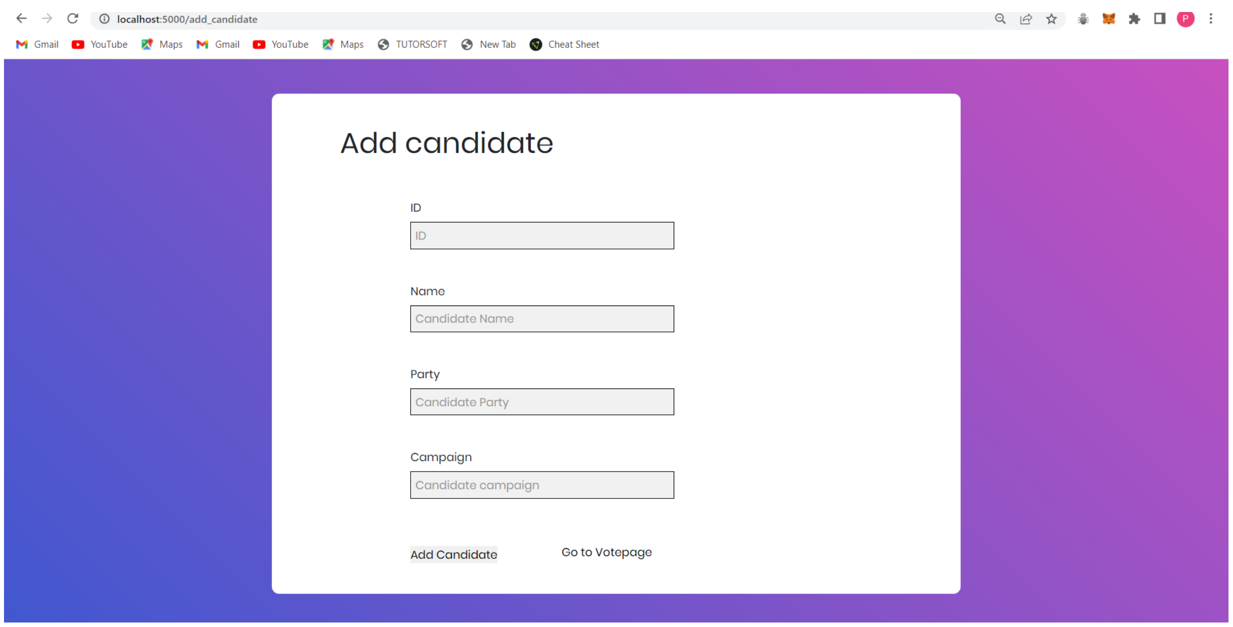
After that, go to the login page and you will be directed to the vote page (Note that the first account you register after deploying the contract becomes the owner of the contract, as per the code of my backend). In this vote page, since the account you registered is now the owner of the contract address, you can add a candidate and press the Start Vote button and End Vote button.



ID	Name	Party	Campaign
1 ☐	A1	Party A	Good Very good
2 ☐	A2	Party B	Very Well
3 ☐	A3	Party C	Good
4 ☐	A4	Party D	OK!

Add candidate Page

Fill the information of the candidate(only owner account can add(make your the account you are in is the owner address in that deploy round).After this Press Add Candidate Button to add and then click Go to VotePage button to redirect to vote page.

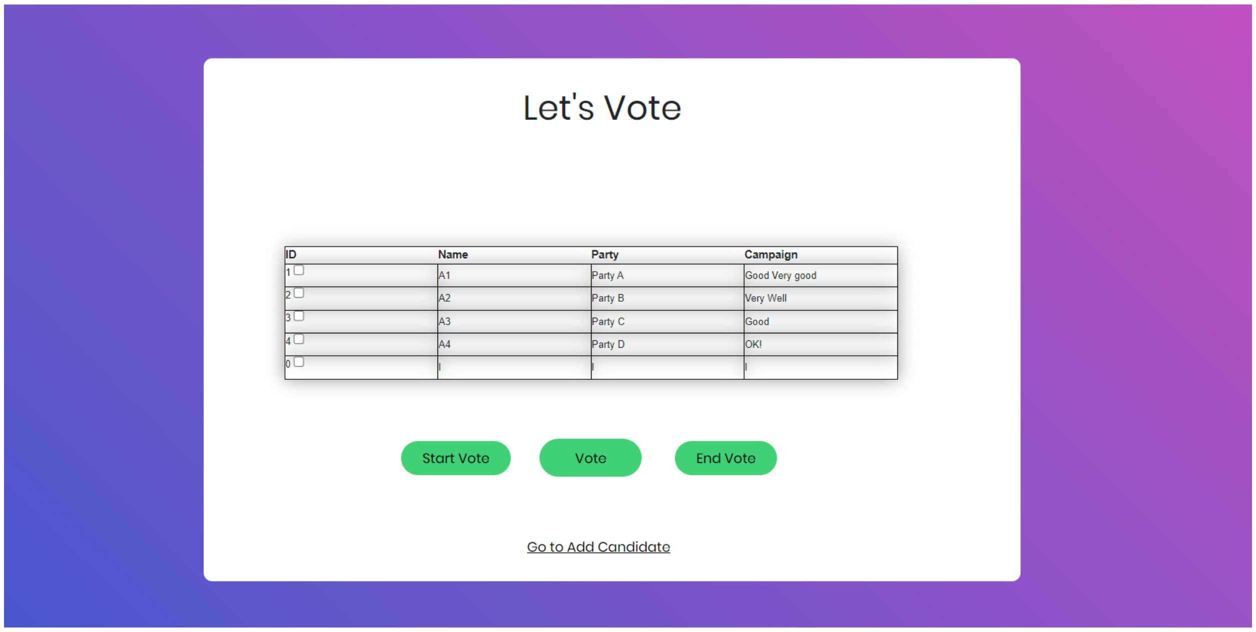


Start vote Process

After adding the candidate , In the vote Page the candidate you add will appear in the table . Press the start vote button to initial the vote state . The Page should refresh if there is nothing wrong.

vote Process

Thick the box for the candidate you chose for and then press the vote button .The page should be refresh is there is nothing wrong.

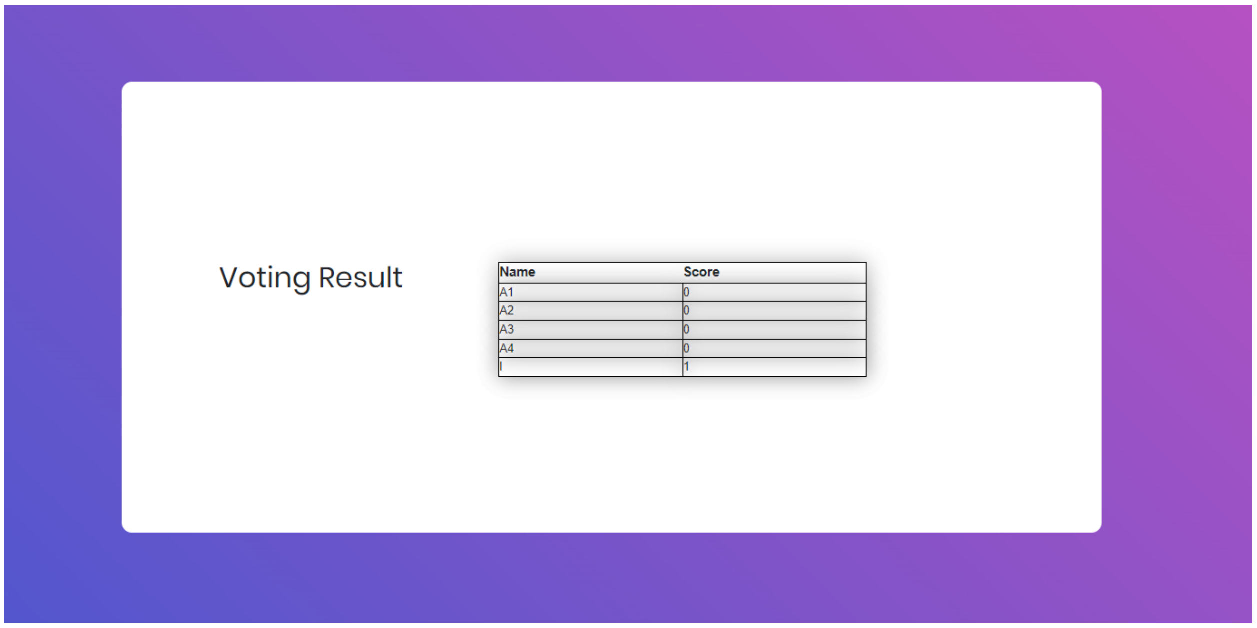


Let's Vote

ID	Name	Party	Campaign
1 ☐	A1	Party A	Good Very good
2 ☐	A2	Party B	Very Well
3 ☐	A3	Party C	Good
4 ☐	A4	Party D	OK!
0 ☐	l	l	l

End vote Process and result Page

After you press the end vote button It will go to the result page (make sure that you are login in the account that presses the starts vote (owner account) Otherwise the Page crashes and errors.

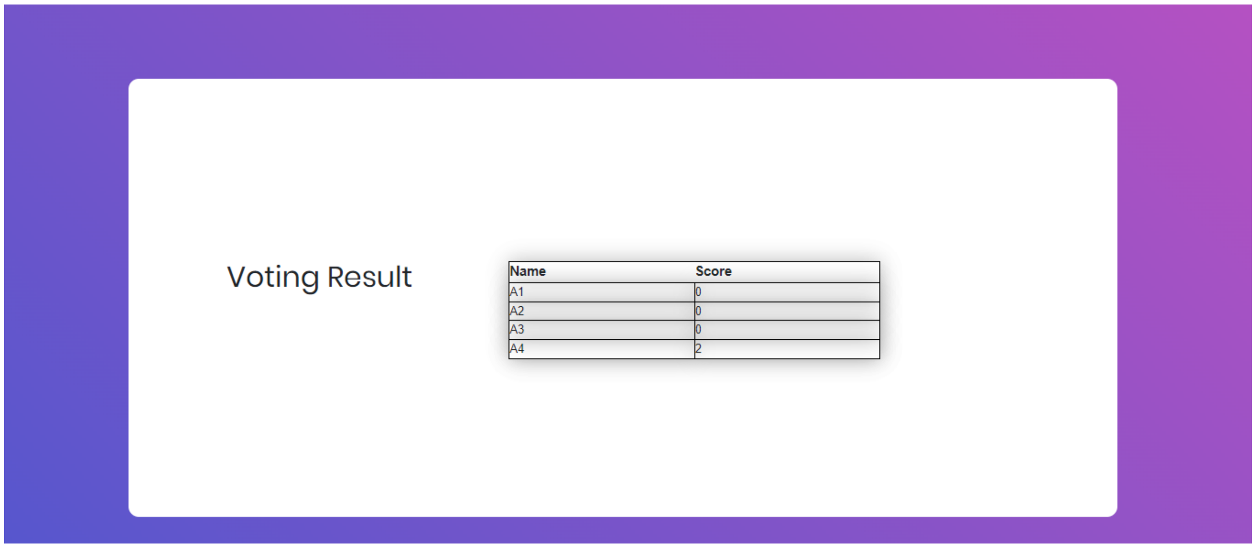


Voting Result

Name	Score
A1	0
A2	0
A3	0
A4	0
l	1

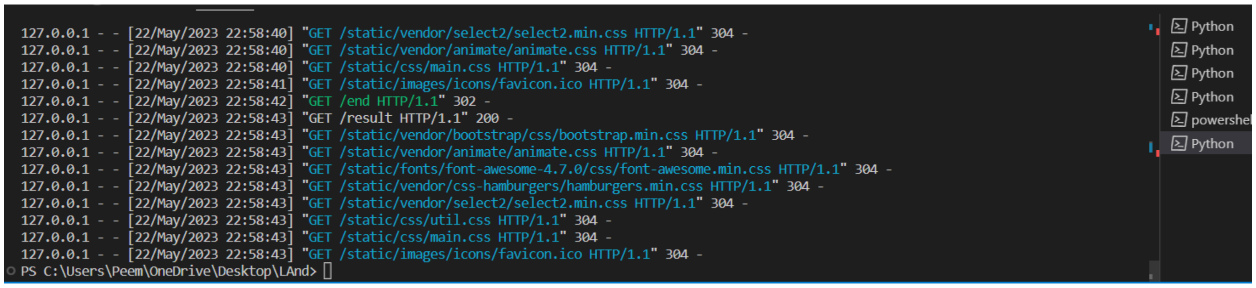
*Important Note

Because this web app doesn't have a logout button . If you want to test a case like ,For example after you use owner account (the first account that you register after the deploy contract of that round) and then you press the start vote button and doing the vote process . instead of going press the end vote for the other account to do their vote in that round(Normal account not Owner) . You just have to press back in the browser to the login page or close the web browser and then open the login Page. Registering the new account and doing the vote process after that do the same as previous for changing account back to the Owner account for doing the end vote Process. If you do the step right, if you use two accounts to vote for the same candidate the score should be 2. (This is my mistake, forget to think about this case for the web page and front end) ,but there is nothing wrong with My contract!!!



Name	Score
A1	0
A2	0
A3	0
A4	2

- If you want to deploy a contract again press switch or close Ganache and open it again in the vscode press ctrl+c in the terminal and then do the deploy process as mentioned earlier before. It should show in the terminal like this if you doing the step right



```
127.0.0.1 - - [22/May/2023 22:58:40] "GET /static/vendor/select2/select2.min.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:40] "GET /static/vendor/animate/animate.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:40] "GET /static/css/main.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:41] "GET /static/images/icons/favicon.ico HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:42] "GET /end HTTP/1.1" 302 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /result HTTP/1.1" 200 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/vendor/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/vendor/animate/animate.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/fonts/font-awesome-4.7.0/css/font-awesome.min.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/vendor/css-hamburgers/hamburgers.min.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/vendor/select2/select2.min.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/css/util.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/css/main.css HTTP/1.1" 304 -
127.0.0.1 - - [22/May/2023 22:58:43] "GET /static/images/icons/favicon.ico HTTP/1.1" 304 -
PS C:\Users\Peem\OneDrive\Desktop\LAnd> 
```

Chapter 4

Conclusion and Discussion

Conclusion

Implementing a voting dApp platform can empower individuals and organizations to conduct secure, transparent, and decentralized voting processes. By leveraging the features of blockchain technology and smart contracts, a voting dApp can ensure immutability, tamper resistance, and verifiability of votes.

The implementation of a voting dApp platform involves several key aspects. First, the use of a blockchain network, such as Ethereum, provides a decentralized and distributed infrastructure that enhances transparency and security. Smart contracts serve as the backbone of the voting process, governing the rules, recording votes, and enforcing the integrity of the system.

To ensure accessibility, a user-friendly interface can be developed, allowing voters to easily participate in the voting process using web or mobile applications. Wallet integration, through tools like Web3, enables users to securely sign and submit their votes, ensuring the authenticity and privacy of each vote.

Additionally, the integration of event listening allows for real-time monitoring and auditing of voting activities, ensuring transparency and accountability. Through data visualization and analytics, the platform can provide valuable insights into voting trends, patterns, and outcomes.

Implementing a voting dApp platform has the potential to revolutionize democratic processes by increasing trust, reducing fraud, and enhancing inclusivity. It empowers individuals to participate in decision-making while maintaining the integrity and privacy of the voting process. By leveraging the benefits of blockchain technology, a voting dApp platform can pave the way for a more transparent, secure, and democratic future.

Chapter 5

References

[1] M. Antonopoulos. “Mastering Bitcoin, 2nd Edition”, Publisher: O'Reilly Media, Inc., Release Date: June 2017, ISBN: 9781491954379.

[2] Andreas M. Antonopoulos, Gavin Wood, “Mastering Ethereum - Building Smart Contracts and Dapps”, Publisher: O'Reilly Media, Inc., Release Date: 2018, https://covers.oreillystatic.com/images/0636920056072/rc_lrg.jpg.

[3] Konstantinos CHRISTIDIS, Michael DEVETSIKIOTIS, “Blockchains and Smart Contracts for the Internet of Things”, IEEE Access, June 2016, Digital Object Identifier 10.1109/ACCESS.2016.2566339.

[4] Imran Bashir, “Mastering Blockchain - Master the theoretical and technical foundations of Blockchain technology and explore future of Blockchain technology”, Packt Publishing, March 2017, ISBN 9781787125445

[5] SEC 2: Recommended Elliptic Curve Domain Parameters: http://www.secg.org/sec2-v2.pdf

[6] Feng Hao, P.Y.A. Ryan and Piotr Zielinski. (2008). Anonymous voting by two-round public discussion. Available at: http://homepages.cs.ncl.ac.uk/feng.hao/files/OpenVote_IET.pdf

[7] Feng Hao and Piotr Zielinski. A 2-Round Anonymous Veto Protocol Available at: http://homepages.cs.ncl.ac.uk/feng.hao/files/av_net.pdf.

[8] Friðrik Þ. Hjálmarsson, Gunnlaugur K. Hreiðarsson, “Blockchain-Based E-Voting System”, Online: https://skemman.is/bitstream/1946/31161/1/Research-Paper-BBEVS.pdf

[9] Timothy Ko, “A guide to developing an Ethereum decentralized voting application”: https://medium.freecodecamp.org/developing-an-ethereum-decentralized-voting-application-a99de24992d9

[10] Jackson Ng, “Voting on a Blockchain: DApp Demonstration”: https://medium.com/coinmonks/voting-on-a-blockchain-dapp-demonstration-dfb5944a0c9e

[11] Mahesh Murthy, “Full Stack Hello World Voting Ethereum Dapp Tutorial”: https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-1-40d2d0d807c2