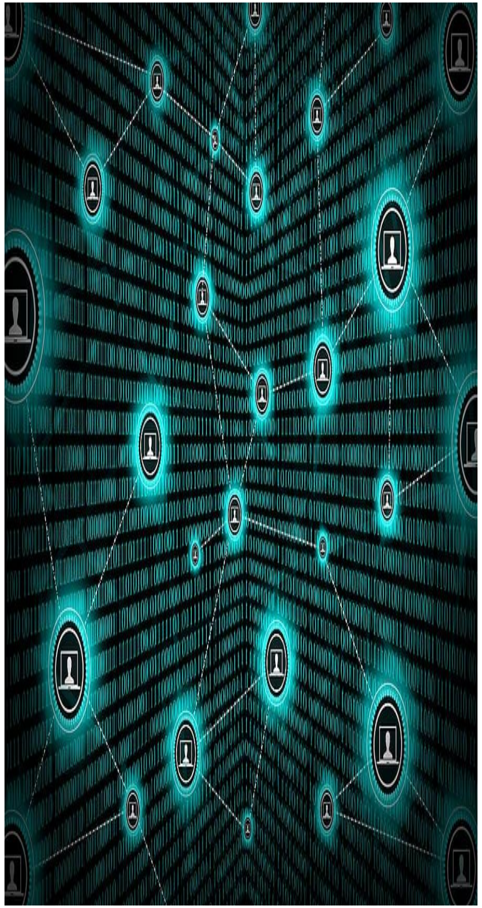
dApp Voting Project

From: Anthony Z., Raul N., Thomas L., Deep P.

Why dApp Voting

Why not?!

- ★ Have a public voting ledger which would ensure transparency and trust in the voting system in use.
- ★ Having a uniform voting application for all valid participants in an election, ensuring equal voting participation for all.
- ★ Ensure voter privacy in election process.
- ★ Minimize the cost and resources necessary to hold elections and ballot measure voting.



Project Objectives

Create a working application which:

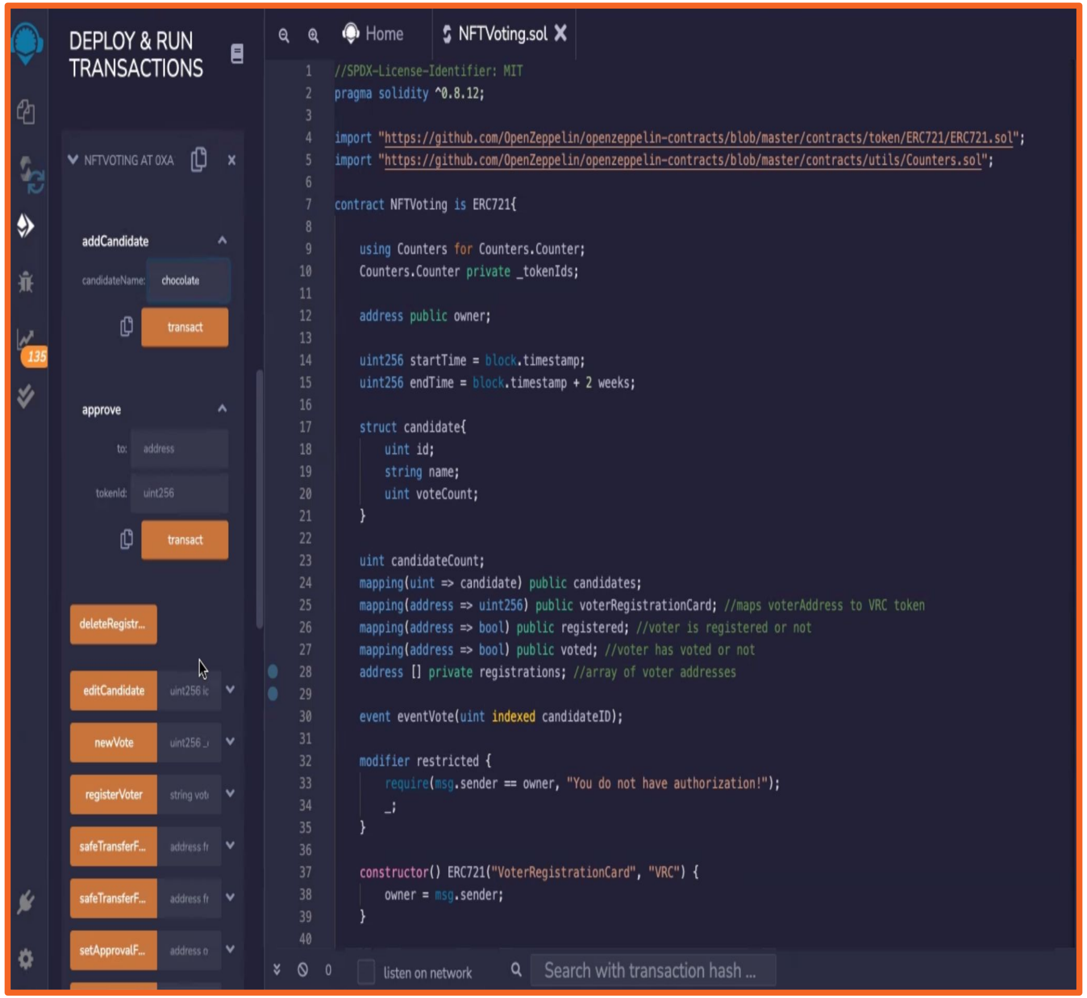
In a Streamlit interface…

- ❏ Accept only valid wallet addresses for verification
- ❏ Mints a “Vote” NFT for use to that address
- ❏ Processes the “Vote” NFT as having voted for candidate, single use only.
- ❏ Updates a plot visualization of the results of voting in real time.

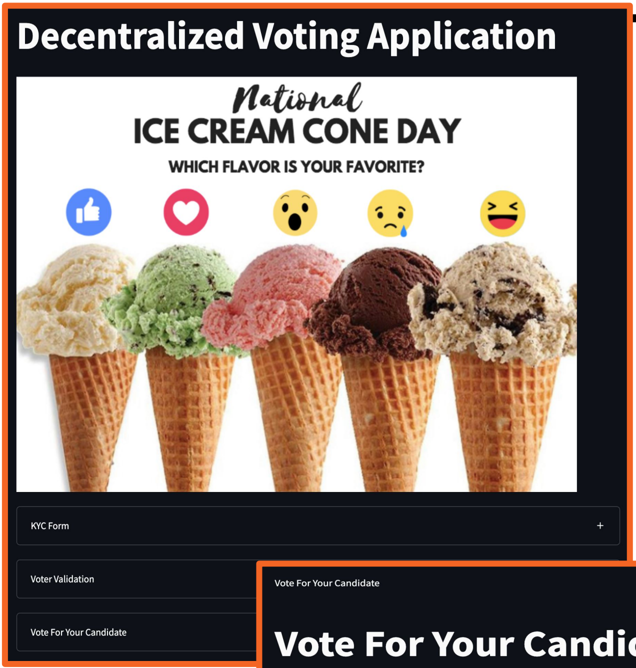
Contract Creation

Using Solidity contracts…

★ Create a contract which is deployed by the “Central Governance”.
★ Mint Voter NFTs which are be used for voting.
★ Create/edit/remove candidates.
★ *Return vote count for data processing and visualization.*
★ *Ability to create a new “election” on the same contract.*



```solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.12;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Counters.sol";

contract NFTVoting is ERC721{

    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    address public owner;

    uint256 startTime = block.timestamp;
    uint256 endTime = block.timestamp + 2 weeks;

    struct candidate{
        uint id;
        string name;
        uint voteCount;
    }

    uint candidateCount;
    mapping(uint => candidate) public candidates;
    mapping(address => uint256) public voterRegistrationCard; //maps voterAddress to VRC token
    mapping(address => bool) public registered; //voter is registered or not
    mapping(address => bool) public voted; //voter has voted or not
    address [] private registrations; //array of voter addresses

    event eventVote(uint indexed candidateID);

    modifier restricted {
        require(msg.sender == owner, "You do not have authorization!");
        _;
    }

    constructor() ERC721("VoterRegistrationCard", "VRC") {
        owner = msg.sender;
    }

```


Decentralized Voting Application

KYC Form +

Voter Validation

Vote For Your Candidate

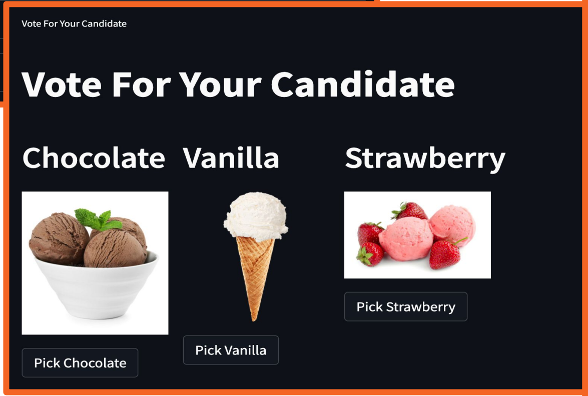
Python app w/Streamlit Integration

★ Pull contract data as needed.

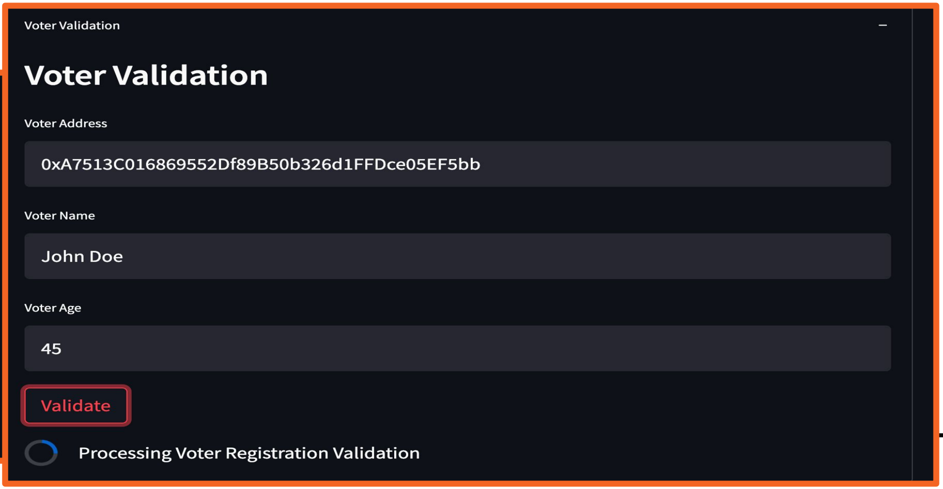
★ Authorize voter validity through 'voter verification process'

- Mint NFT to their verified address, through contract.

★ Let voter know if they are authorized to vote or not.


Vote For Your Candidate

Vote For Your Candidate


Voter Validation –

Voter Validation

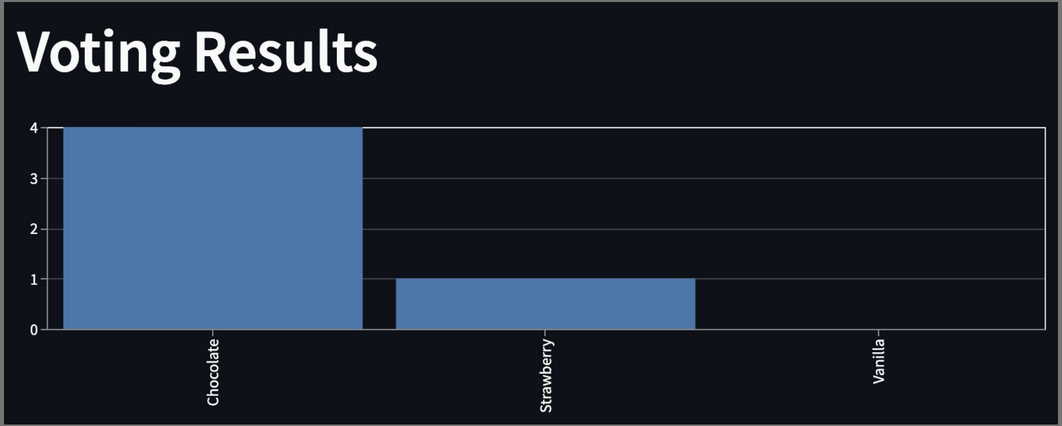
Voter Address: 0xA7513C016869552Df89B50b326d1FFDce05EF5bb

Voter Name: John Doe

Voter Age: 45

Processing Voter Registration Validation

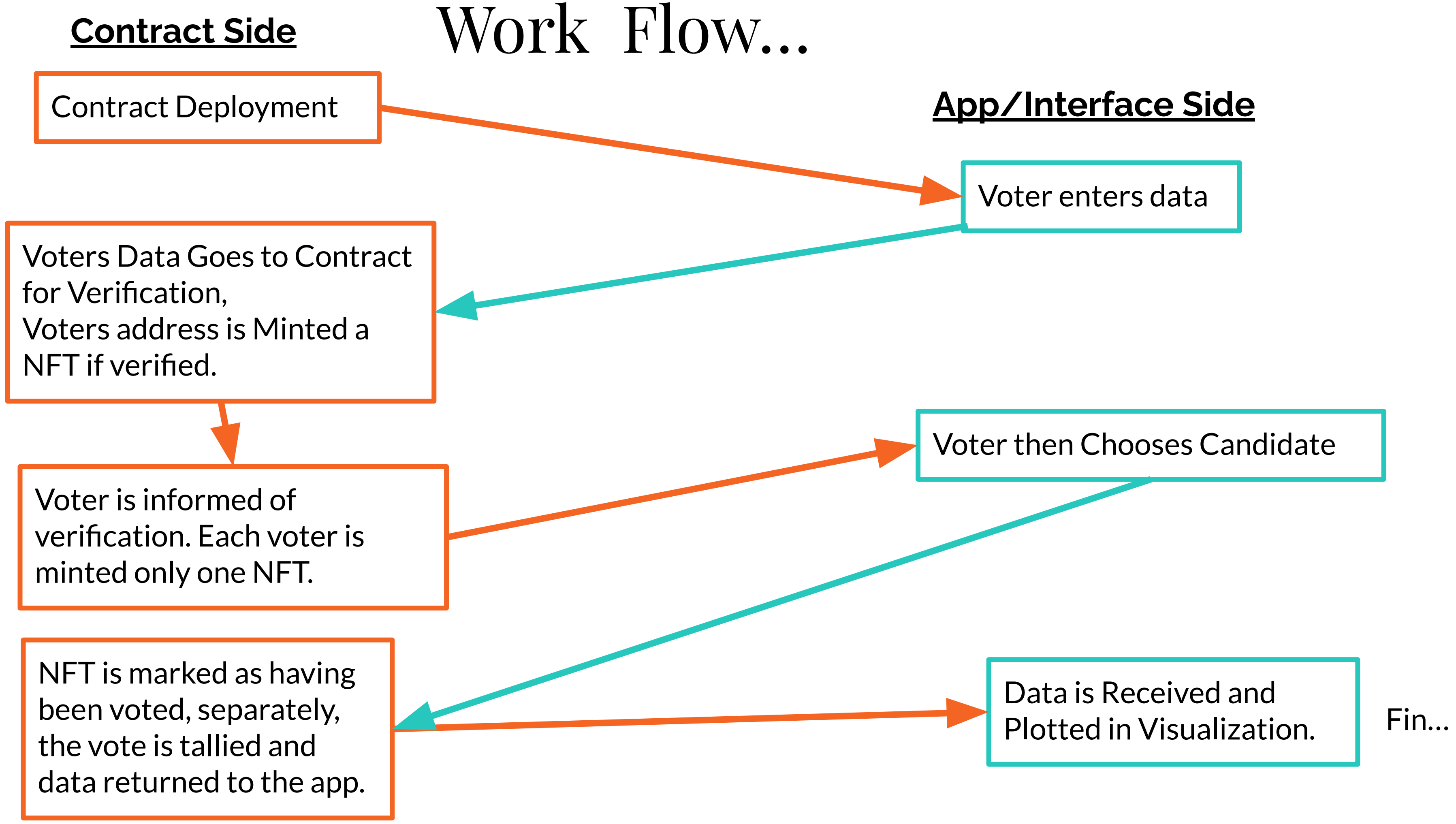
Processing...



★ Voter votes for candidate.
  ○ Processes the vote by marking the NFT as voted.
  ○ Returning data of processed vote and the vote count.

★ Display visualization of vote tally for public real time review.

Work Flow...



Contract Deployment

Voter enters data

Voters Data Goes to Contract for Verification,
Voters address is Minted a NFT if verified.

Voter is informed of verification. Each voter is minted only one NFT.

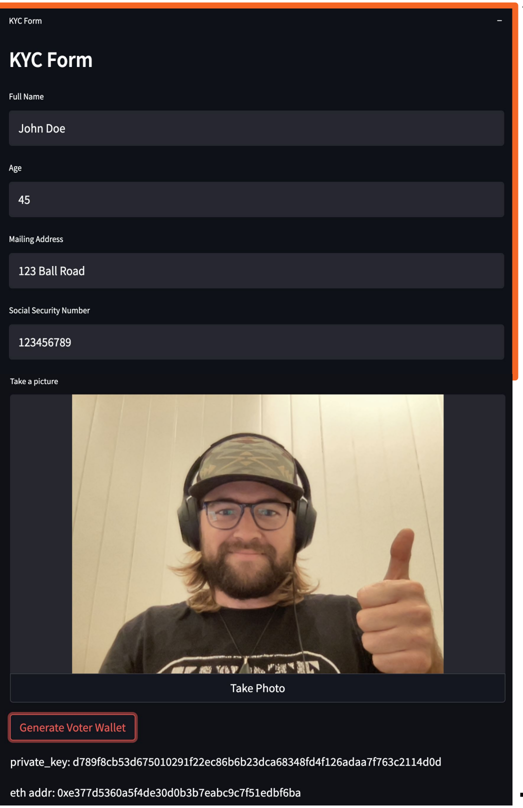
Voter then Chooses Candidate

NFT is marked as having been voted, separately, the vote is tallied and data returned to the app.

Data is Received and Plotted in Visualization.

Fin...

After Action Review



KYC Form

Full Name	John Doe
Age	45
Mailing Address	123 Ball Road
Social Security Number	123456789

Take a picture

```
private_key: d789f8cb53d675010291f22ec86b6b23dca68348fd4f126adaa7f763c2114d0d

eth addr: 0xe377d5360a5f4de30d0b3b7eabc9c7f51edbf6ba
```

Reflection and Challenges

★ Deciding on the amount of contracts necessary and how they would interact.

★ Deciding on the flow of the tokens themselves for both possible fees involved and simplicity.

★ Streamlit functionality for caching and input use.

Moving forward...

★ Improving/implementing Know Your Customer attributes would be ideal with valid photo kept for address accounts.

★ Using a more dynamic interface for actual deployment.

Project Breakdown

Solidity Contract Criteria-

Will mint a 'Vote' token.

Will accept only 'verified' wallet addresses.

Will distribute no more than 1 token per authorized wallet addresses.

(Potentially inside contract - token will automatically go to specific 'candidate' account 'voted' for)

Python app-

Integrate Smart Contract to app and Streamlit interface.

Create function for wallet address verification.

Create function for token minting to address.

Display via Streamlit syntax pictures of 'candidates' - Chocolate Ice Cream, Strawberry Ice Cream, Vanilla Ice Cream.

Create function which takes 'vote' for specific 'candidate', routes minted token to that 'candidates' wallet address.

Create function which gets balance from each candidate account. Then plots totals to a visualization after each 'vote' occurs.

Include a visualization of the blockchain on the streamlit interface.