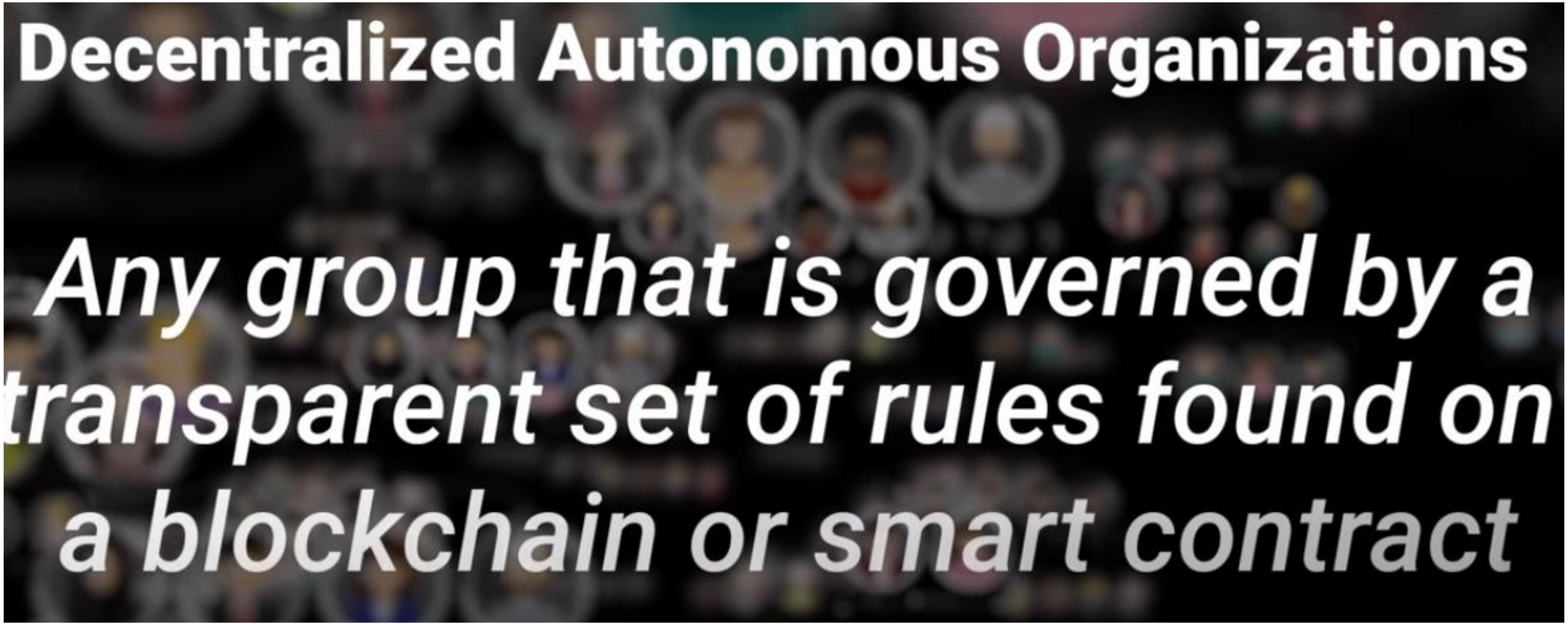
Build a DAO 1.05.45.26

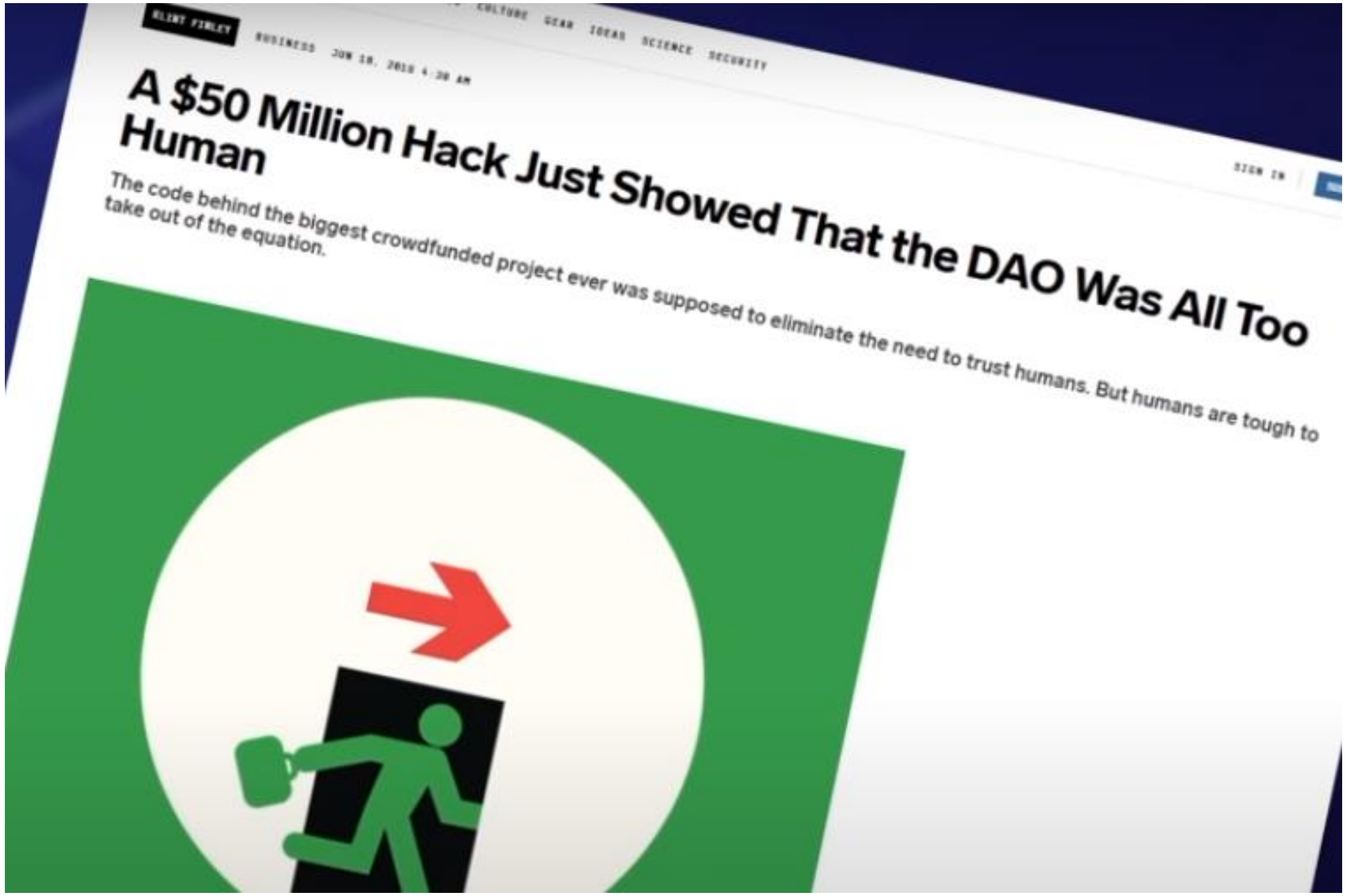
What is a DAO 1.05.46.36

Decentralized
Autonomous
Organizations



Decentralized Autonomous Organizations

Any group that is governed by a transparent set of rules found on a blockchain or smart contract



KLINT FINLEY

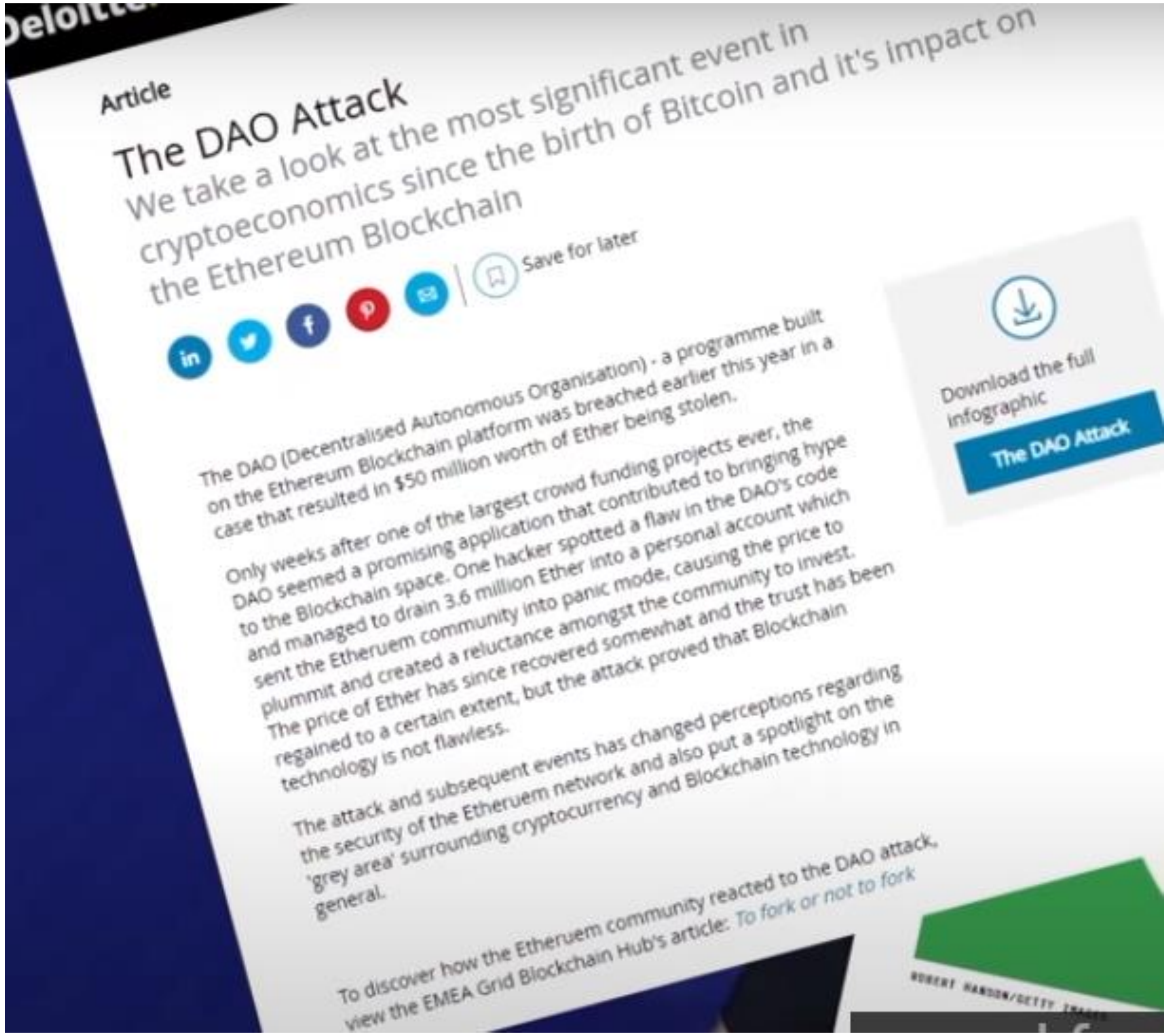
CULTURE GEAR IDEAS SCIENCE SECURITY

BUSINESS JUN 18, 2016 4:30 AM

SIGN IN

A $50 Million Hack Just Showed That the DAO Was All Too Human

The code behind the biggest crowdfunded project ever was supposed to eliminate the need to trust humans. But humans are tough to take out of the equation.



Deloitt

Article

The DAO Attack

We take a look at the most significant event in cryptoeconomics since the birth of Bitcoin and it's impact on the Ethereum Blockchain

Save for later

The DAO (Decentralised Autonomous Organisation) - a programme built on the Ethereum Blockchain platform was breached earlier this year in a case that resulted in $50 million worth of Ether being stolen.

Only weeks after one of the largest crowd funding projects ever, the DAO seemed a promising application that contributed to bringing hype to the Blockchain space. One hacker spotted a flaw in the DAO's code and managed to drain 3.6 million Ether into a personal account which sent the Etheruem community into panic mode, causing the price to plummit and created a reluctance amongst the community to invest. The price of Ether has since recovered somewhat and the trust has been regained to a certain extent, but the attack proved that Blockchain technology is not flawless.

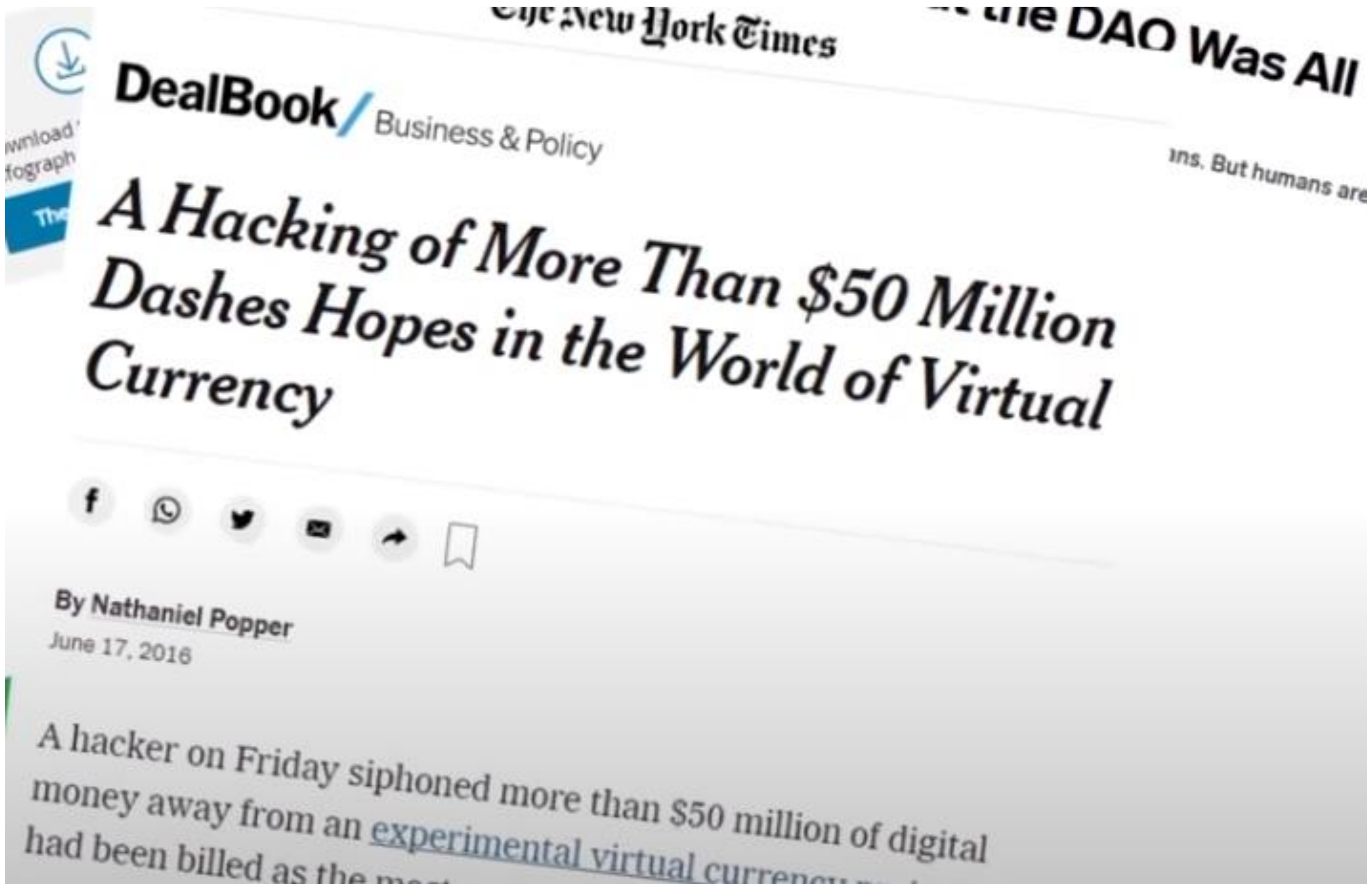
The attack and subsequent events has changed perceptions regarding the security of the Etheruem network and also put a spotlight on the 'grey area' surrounding cryptocurrency and Blockchain technology in general.

To discover how the Etheruem community reacted to the DAO attack, view the EMEA Grid Blockchain Hub's article: *To fork or not to fork*

Download the full infographic

The DAO Attack

ROBERT HANSON/GETTY IMAGES



wnload
fograph
The

The New York Times

the DAO Was All

ns. But humans are

DealBook/ Business & Policy

A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency

By Nathaniel Popper
June 17, 2016

A hacker on Friday siphoned more than $50 million of digital money away from an experimental virtual currency
had been billed as the

Imagine if all of the users of Google were given voting power into what Google should do next. And the rules of the voting was immutable, transparent and decentralized?

This solves an age old problem of trust, centrality and transparency and giving the power to the users of different protocols and applications instead of everything happening behind closed doors.

And this voting piece is a cornerstone of how thse operate. This Decentralized Governance, if you will. And it summarized by company or organizations operated exclusively through code.

Company / Organization
Operated
Exclusively
Through
Code

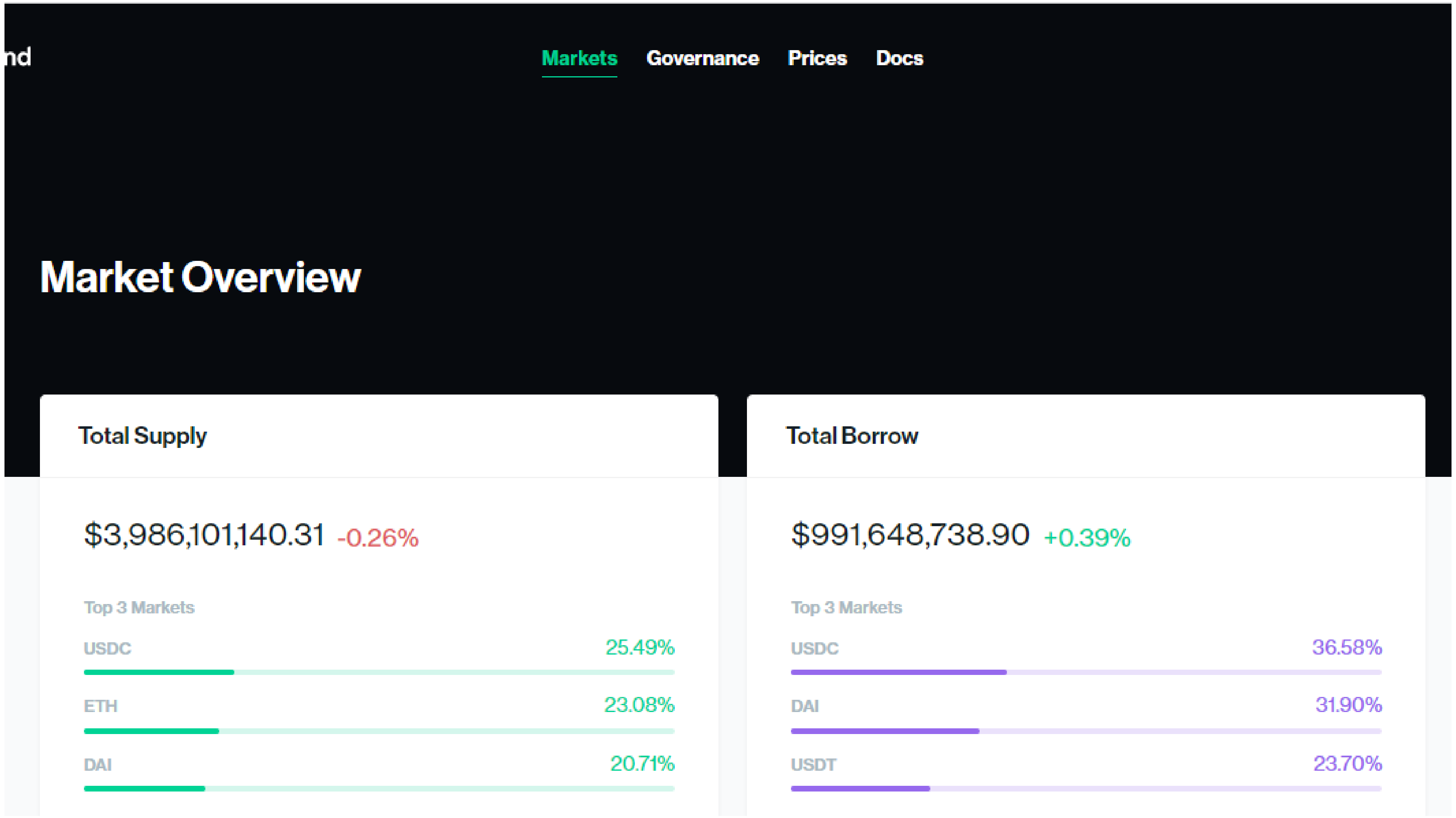
DeGov

Compound Protocol	1.05.47.57

To really understand all this were going to look under the hood of the protocol thats setting the precedent for all other doubs and compound.

https://compound.finance/

Its a borrowing and lending application that allows users to borrow and lend their assets.


nd

Markets Governance Prices Docs

Market Overview

Total Supply

$3,986,101,140.31 -0.26%

Top 3 Markets

USDC	25.49%
ETH	23.08%
DAI	20.71%

Total Borrow

$991,648,738.90 +0.39%

Top 3 Markets

USDC	36.58%
DAI	31.90%
USDT	23.70%

And everything about this app is built in smart contracts.

Now, oftentimes they are gonna want to do a lot new things.

Maybe they want to add a new tken to allow borrowing and lending.

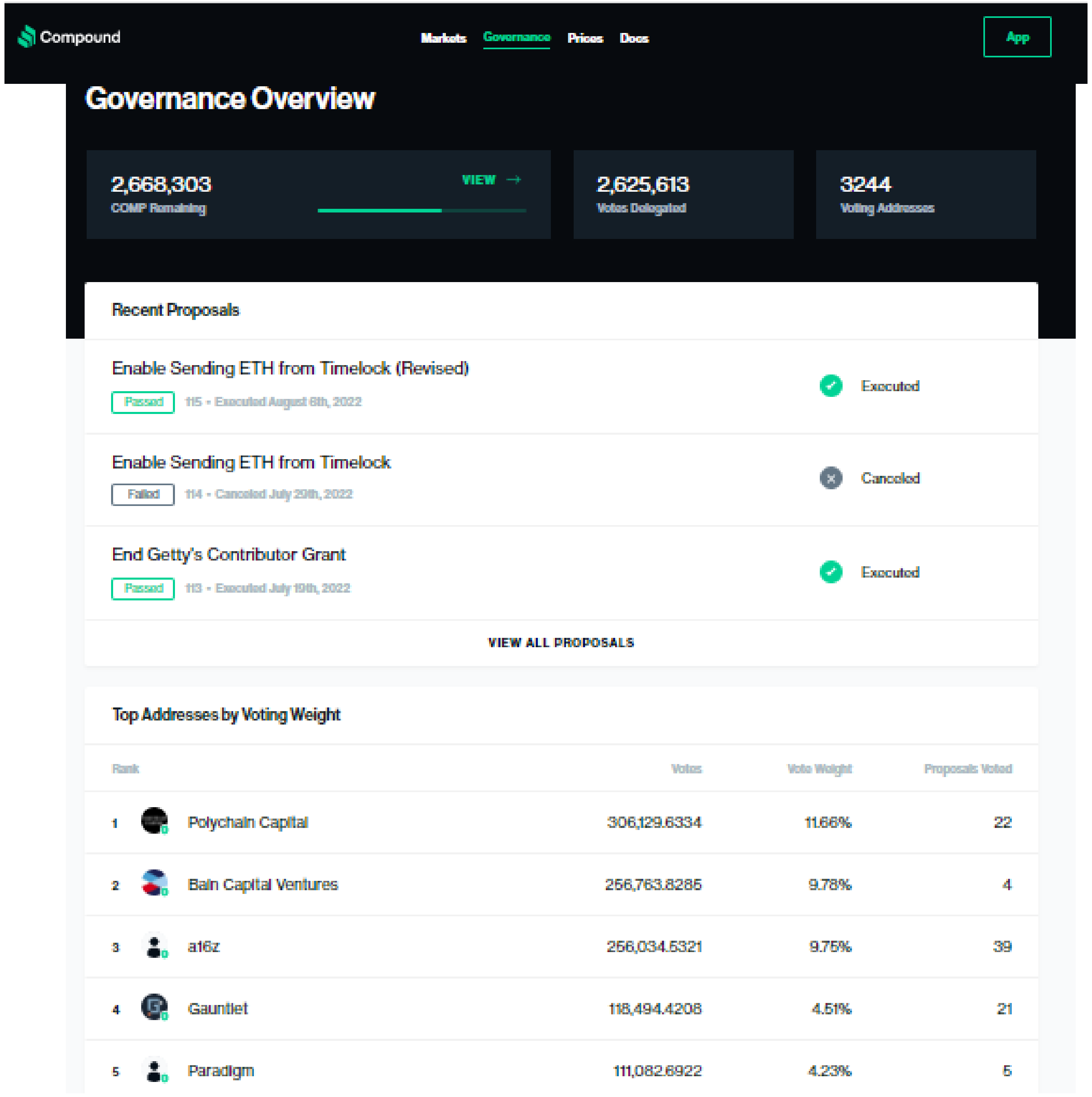
Maybe they re gonna want to change some of the API parameters.

Maybe theyre gonna want to block certain coins.

There is alot of different things that they might want to do.

So that where we are gonna go ahead to Governance.

Governance is where you can find a user interface for a list of all the proposals and all the different ballot s that came to be. So here is a list of some of the governance proposals that this protocol has actually been making to imporove.



Governance Overview

2,668,303
COMP Remaining
VIEW →

2,625,613
Votes Delegated

3244
Voting Addresses

Recent Proposals

Enable Sending ETH from Timelock (Revised)
Passed 115 • Executed August 6th, 2022 — Executed

Enable Sending ETH from Timelock
Failed 114 • Canceled July 29th, 2022 — Canceled

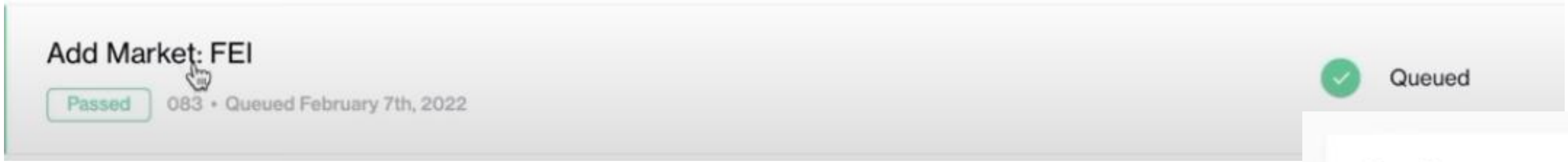
End Getty's Contributor Grant
Passed 113 • Executed July 19th, 2022 — Executed

VIEW ALL PROPOSALS

Top Addresses by Voting Weight

Rank		Votes	Vote Weight	Proposals Voted
1	Polychain Capital	306,129.6334	11.66%	22
2	Bain Capital Ventures	256,763.8285	9.78%	4
3	a16z	256,034.5321	9.75%	39
4	Gauntlet	118,494.4208	4.51%	21
5	Paradigm	111,082.6922	4.23%	5

So lets look at one of these actually in process:


Add Market: FEI

Passed 083 • Queued February 7th, 2022

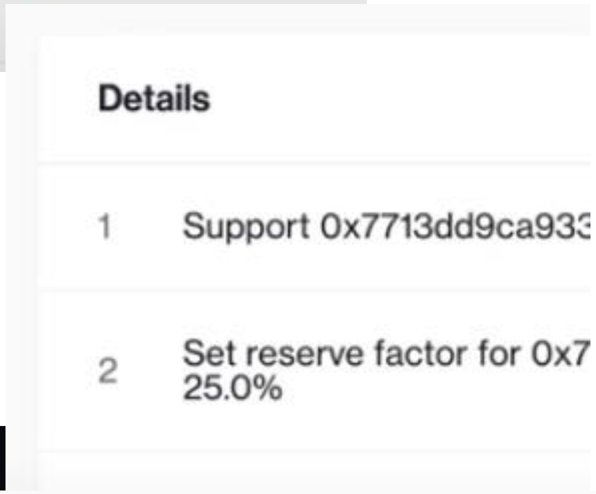
Queued

When click proposal actually see everything about this proposals;

who voted for

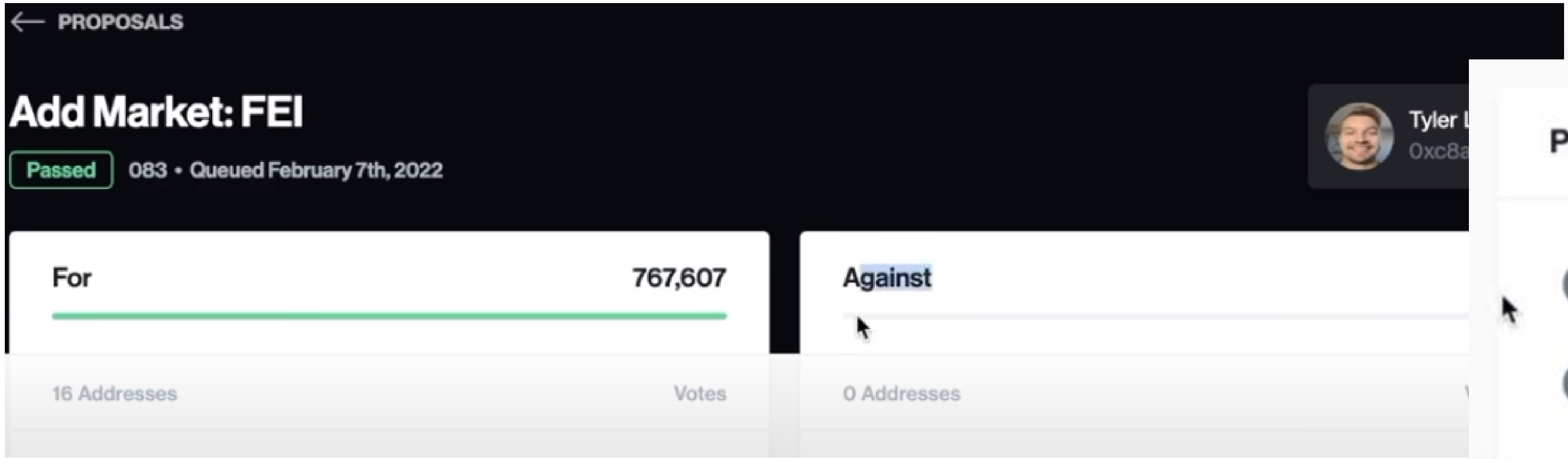
who voted against

the proposal histhory


Details

1 Support 0x7713dd9ca933

2 Set reserve factor for 0x7 25.0%


PROPOSALS

Add Market: FEI

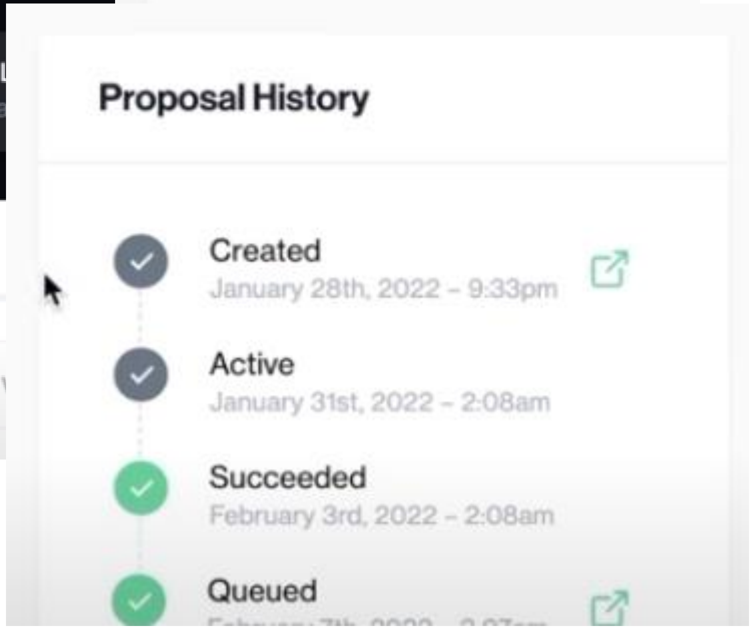
Passed 083 • Queued February 7th, 2022

Tyler

0xc8a

For	767,607
16 Addresses	Votes

Against
0 Addresses


Proposal History

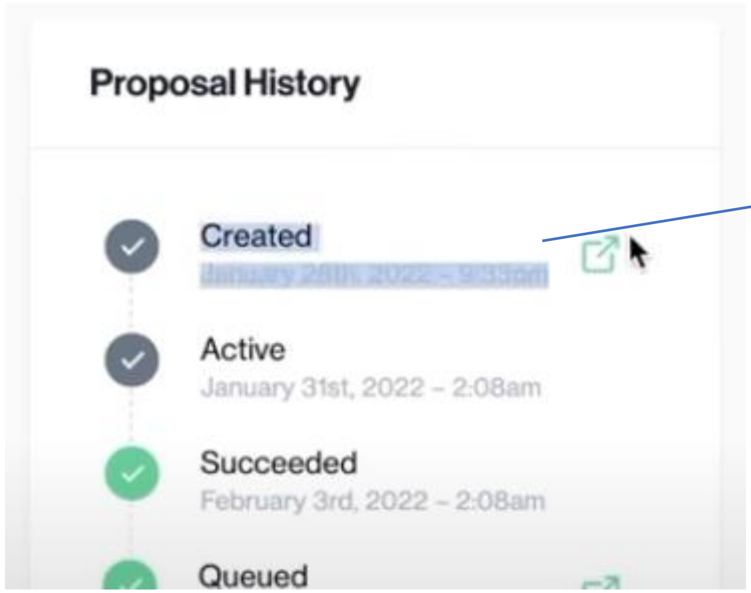
Created
January 28th, 2022 – 9:33pm

Active
January 31st, 2022 – 2:08am

Succeeded
February 3rd, 2022 – 2:08am

Queued

The first thing to one of these proposals is somebody has to actually create the proposal in a proposed transaction.



Proposal History

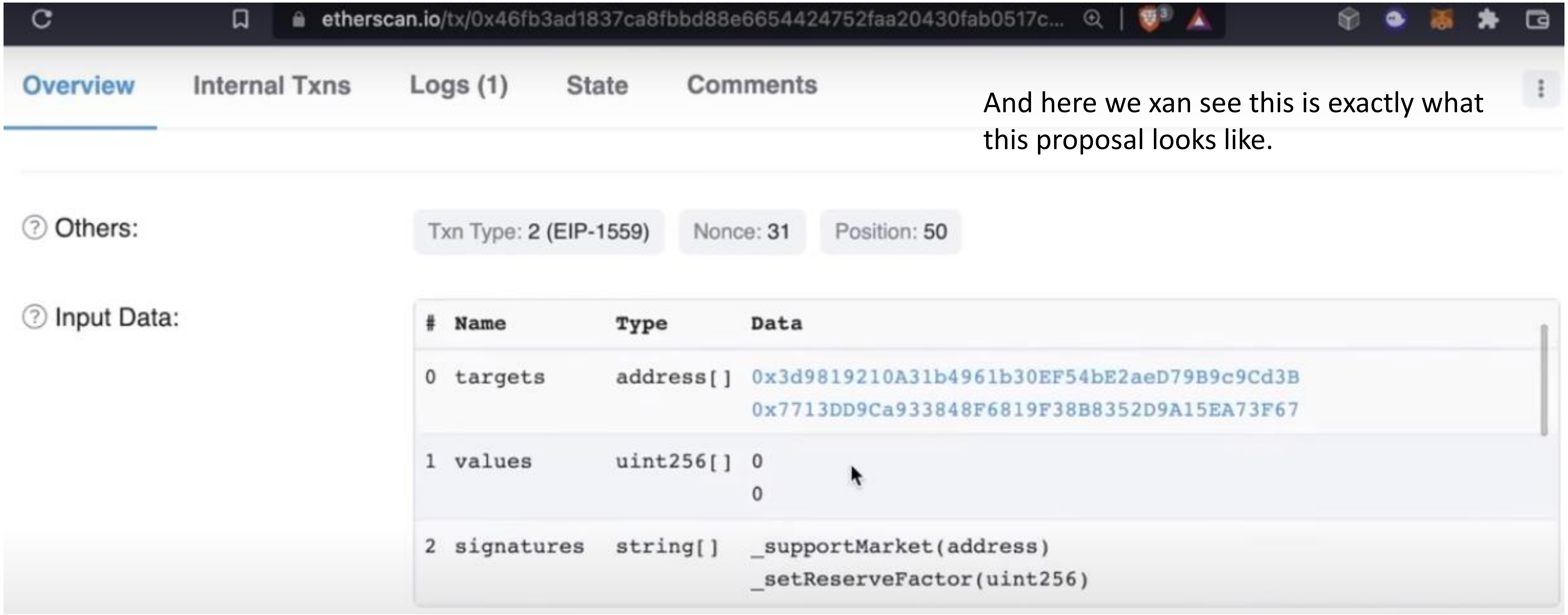
Created
January 28th, 2022 – 9:33pm

Active
January 31st, 2022 – 2:08am

Succeeded
February 3rd, 2022 – 2:08am

Queued

Click transaction


etherscan.io/tx/0x46fb3ad1837ca8fbbd88e6654424752faa20430fab0517c...

Others: Txn Type: 2 (EIP-1559) | Nonce: 31 | Position: 50

Input Data:

#	Name	Type	Data
0	targets	address[]	0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B 0x7713DD9Ca933848F6819F38B8352D9A15EA73F67
1	values	uint256[]	0 0

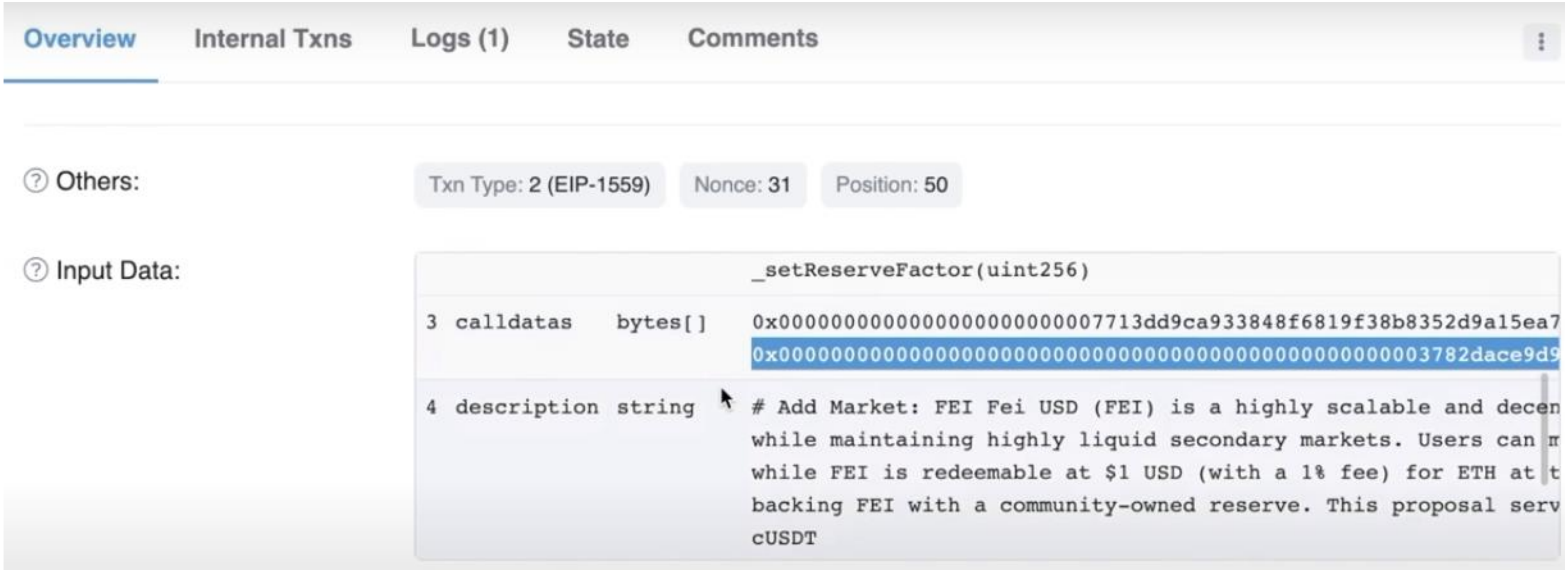
2	signatures	string[]	_supportMarket(address)
			_setReserveFactor(uint256)

And here we xan see this is exactly what this proposal looks like.

The way that theyare typically divided is they have alist of addresses and the list of functions to call on those addresses. And then obviously the parameters to pass those addresses. So this proposal is saying , "hey, i would like to call supportmarket() on this address, setReserveFactor on this address. Here are the paremeters we are going to pass.

They are obviously encoded with bytes and then there is the description string of what this is doing and why we are actually doing this.

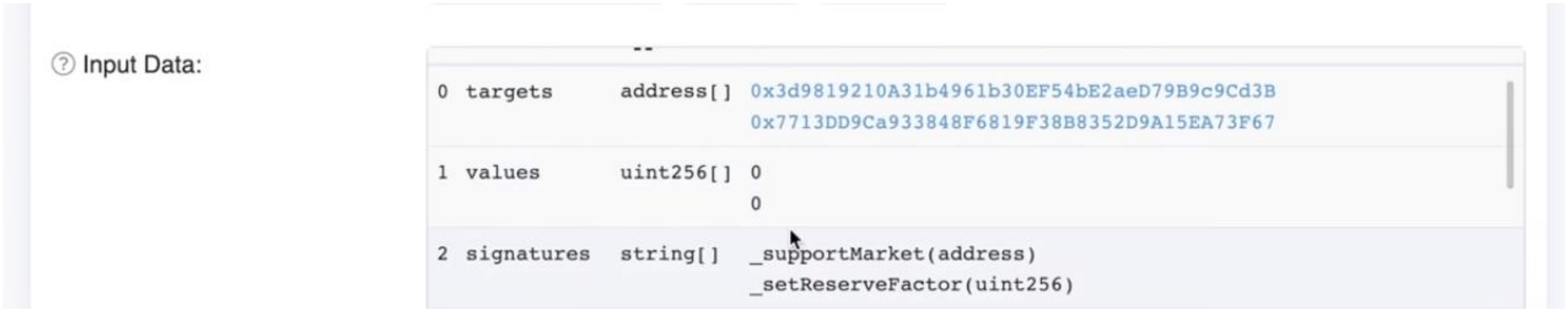
The reason to do this proposal governance process is that these contracts likely have Access controls where only th owner of these contracts can actully call these two functions and the owner of these two contracts (addresses are the contract addresses) is likely going to be this governance style.



Others: Txn Type: 2 (EIP-1559) Nonce: 31 Position: 50

Input Data:

```
                              _setReserveFactor(uint256)
3  calldatas    bytes[]       0x0000000000000000000000007713dd9ca933848f6819f38b8352d9a15ea7
                              0x00000000000000000000000000000000000000000000000003782dace9d9
4  description  string        # Add Market: FEI Fei USD (FEI) is a highly scalable and decen
                              while maintaining highly liquid secondary markets. Users can m
                              while FEI is redeemable at $1 USD (with a 1% fee) for ETH at t
                              backing FEI with a community-owned reserve. This proposal serv
                              cUSDT
```


Input Data:

```
0  targets     address[]  0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B
                          0x7713DD9Ca933848F6819F38B8352D9A15EA73F67
1  values      uint256[]  0
                          0
2  signatures  string[]   _supportMarket(address)
                          _setReserveFactor(uint256)
```

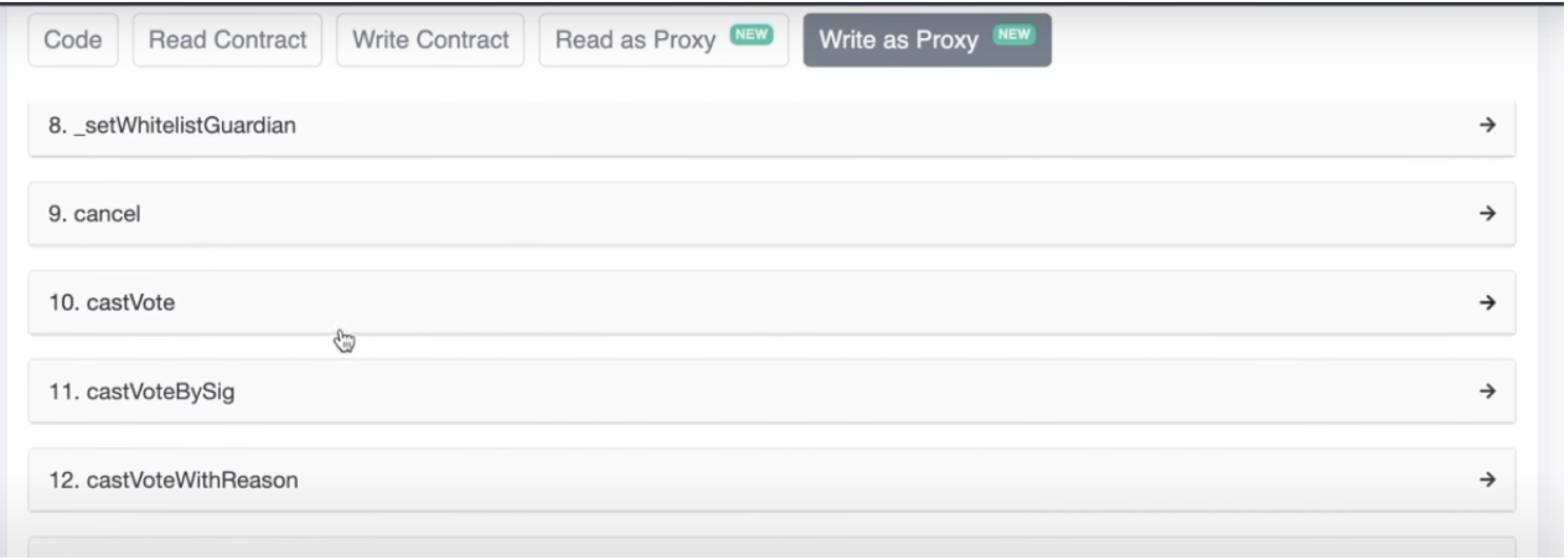
And values zero just means that we re not going to send any eth alon with hese transactions.

Once the proposal has been created afer ashort delay to becomes active. And this is when people can actually start voting on them.

This delay between a proposal and an act of vote can be cahnged or modified depending on your doubt.

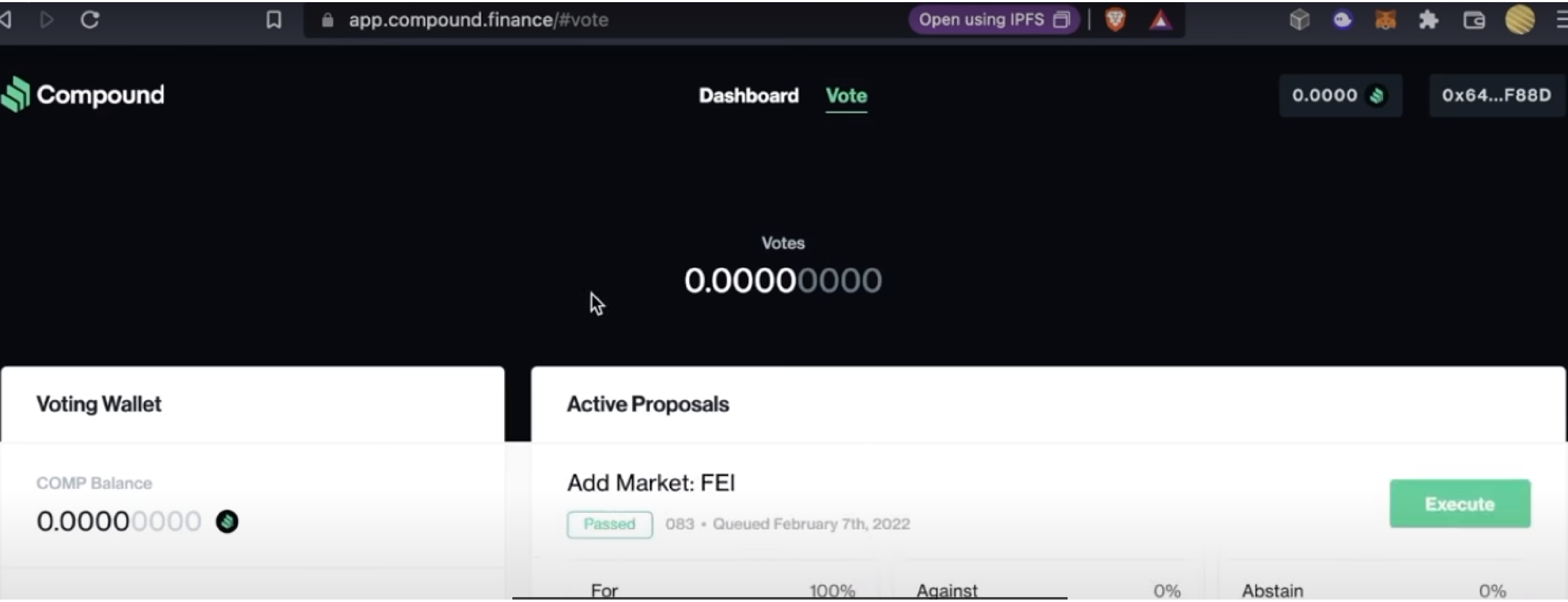
And if it passes it reaches succeeed.

In contract you can see the exact function that the people call to vote, namely castVote, castVoteBySig, castBoteWithReason



8. _setWhitelistGuardian
9. cancel
10. castVote
11. castVoteBySig
12. castVoteWithReason

Then go to the compound App and connect with wallet:



Votes
0.00000000

Voting Wallet

COMP Balance
0.00000000

Active Proposals

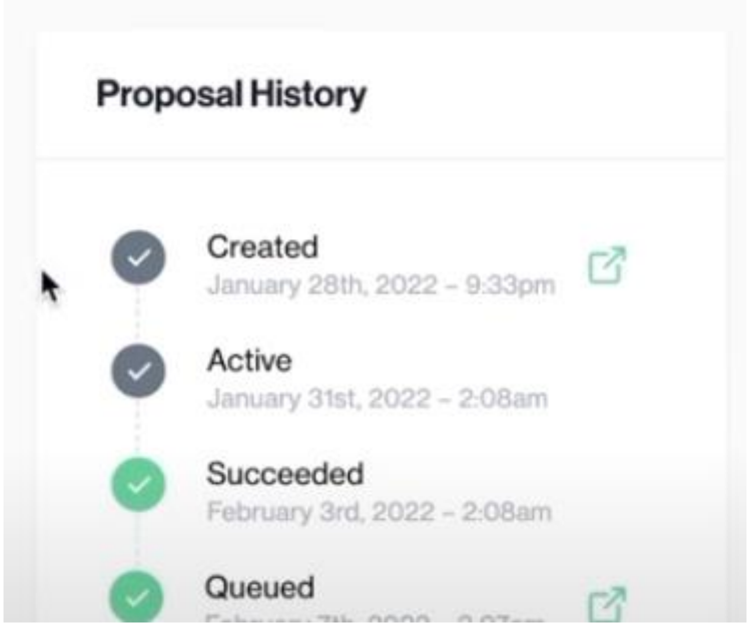
Add Market: FEI
Passed 083 • Queued February 7th, 2022
Execute

For 100% Against 0% Abstain 0%

This is the ui to vote

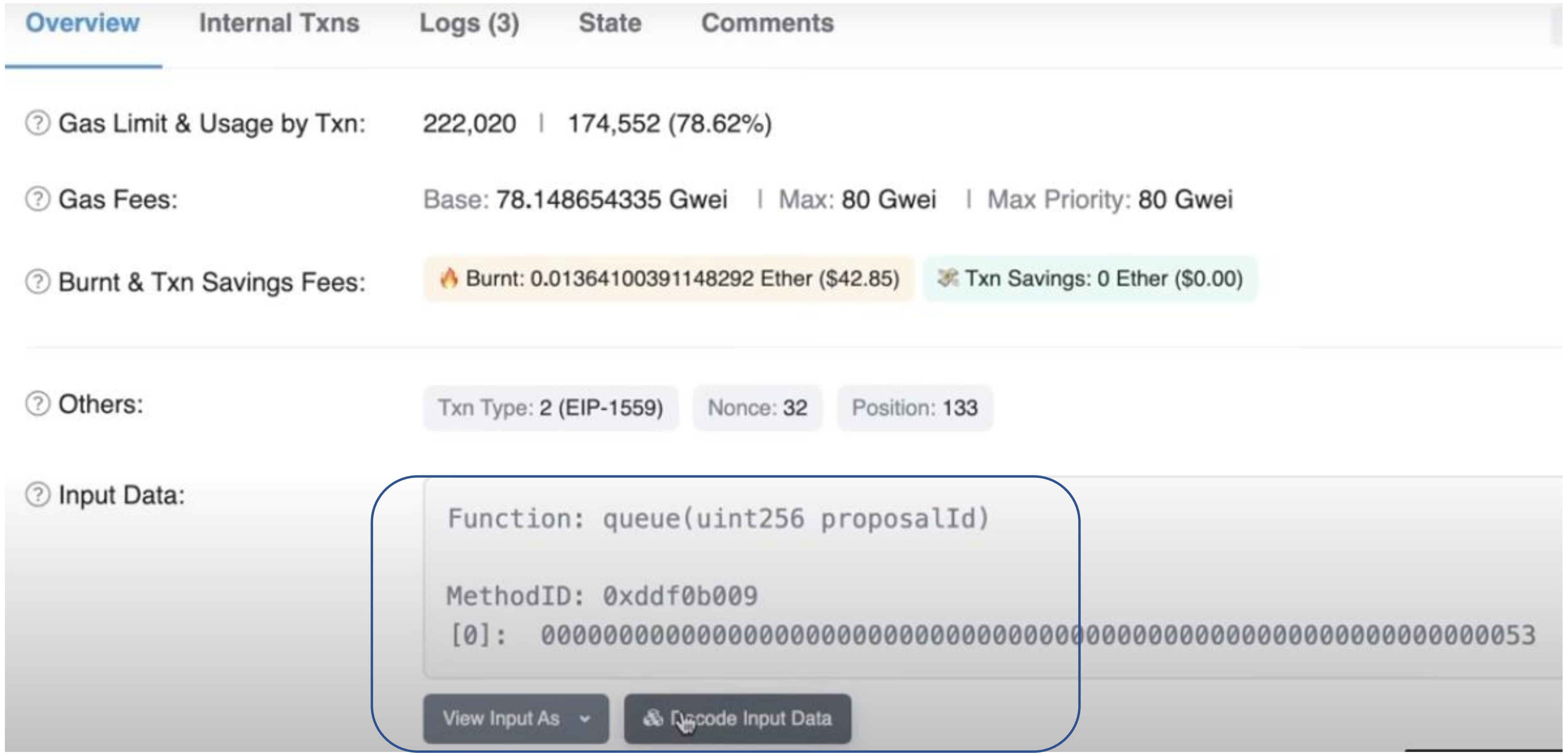
Once all those votes happen it reaches QUEUED STAGE.

Before a proposal actually becomes active, there is a minimum delay between a proposal Passing and a proposal being executed. So somebody has to call this queue function.



Proposal History

- Created — January 28th, 2022 – 9:33pm
- Active — January 31st, 2022 – 2:08am
- Succeeded — February 3rd, 2022 – 2:08am
- Queued



Overview | Internal Txns | Logs (3) | State | Comments

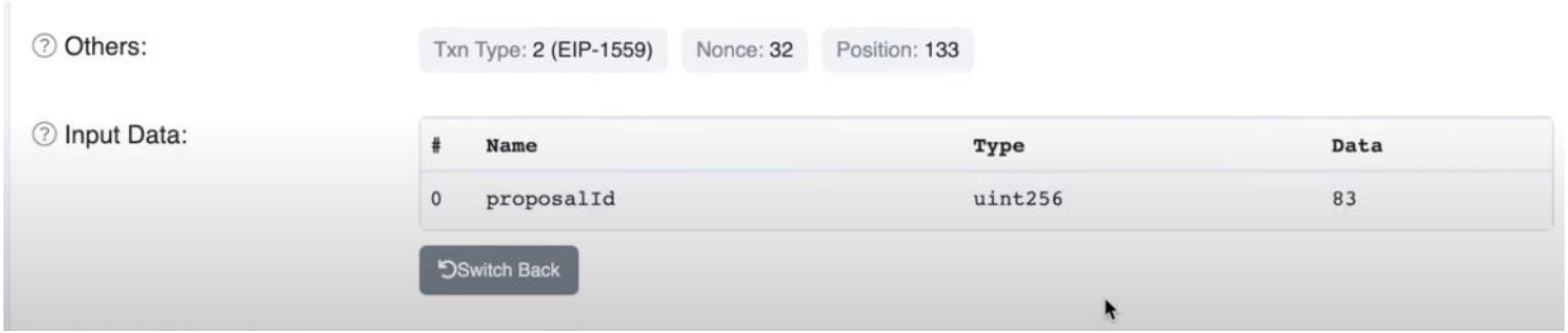
Gas Limit & Usage by Txn:	222,020 | 174,552 (78.62%)
Gas Fees:	Base: 78.148654335 Gwei | Max: 80 Gwei | Max Priority: 80 Gwei
Burnt & Txn Savings Fees:	Burnt: 0.01364100391148292 Ether ($42.85) Txn Savings: 0 Ether ($0.00)
Others:	Txn Type: 2 (EIP-1559) Nonce: 32 Position: 133

Input Data:

```
Function: queue(uint256 proposalId)

MethodID: 0xddf0b009
[0]:  0000000000000000000000000000000000000000000000000000000000000053
```

View Input As | Decode Input Data

And it only can be called if a vote passed. And it says ok that proposal Id has been queued and we are going to execute it soon.



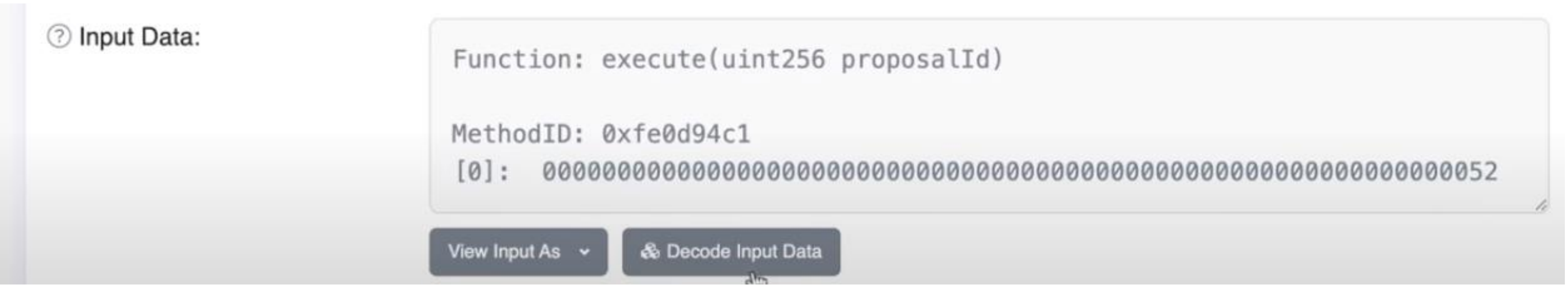
Others: Txn Type: 2 (EIP-1559) Nonce: 32 Position: 133

Input Data:

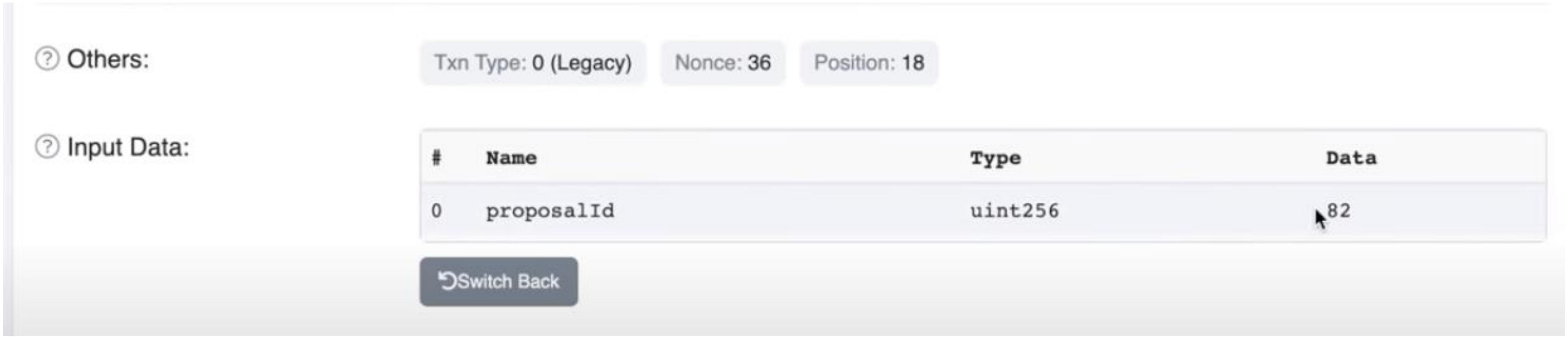
#	Name	Type	Data
0	proposalId	uint256	83

Now if we to to a different proposal, we can see it has been executed. We can seel somebody called this executed function. And they executed proposal detail.

And this is the full lifecyle example of proposal


Input Data:

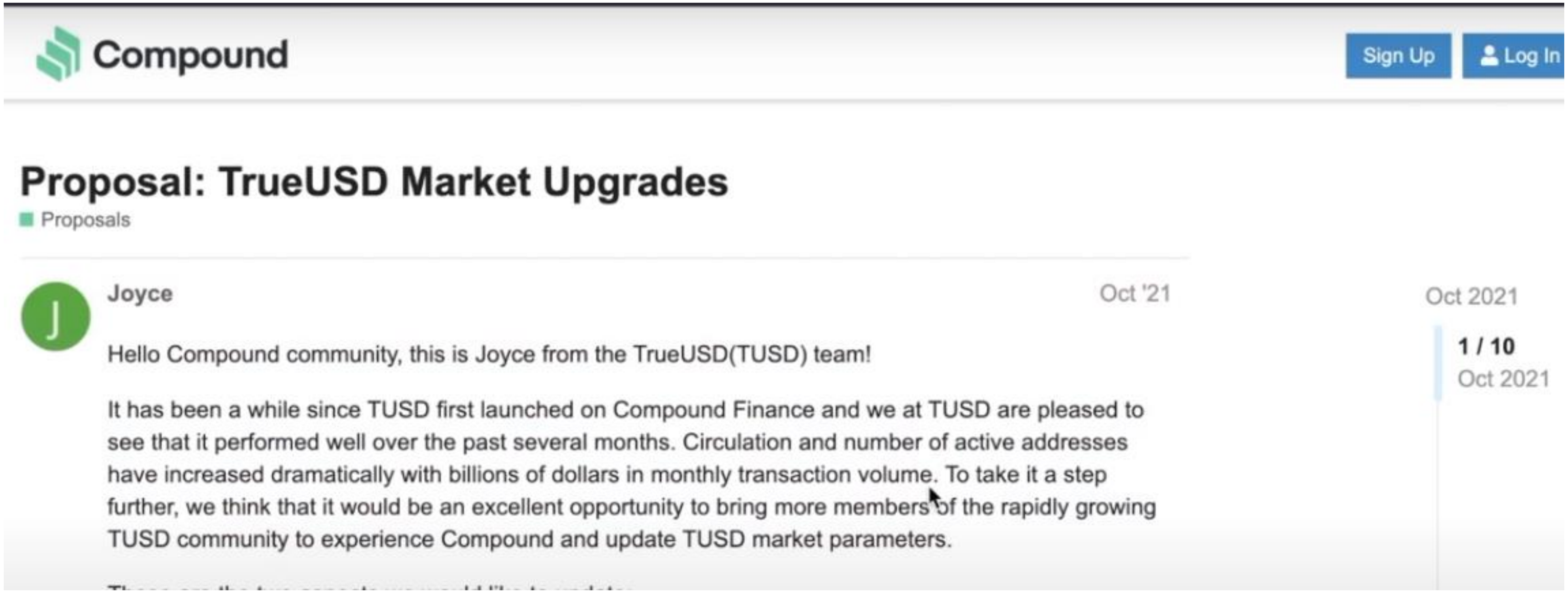
```
Function: execute(uint256 proposalId)

MethodID: 0xfe0d94c1
[0]:  0000000000000000000000000000000000000000000000000000000000000052
```


Others: Txn Type: 0 (Legacy) Nonce: 36 Position: 18

Input Data:

#	Name	Type	Data
0	proposalId	uint256	82

Sometimes you need to add a forum or something like forum or disscussion for the proposal.



Proposal: TrueUSD Market Upgrades

Proposals

Joyce Oct '21

Oct 2021

1 / 10
Oct 2021

Hello Compound community, this is Joyce from the TrueUSD(TUSD) team!

It has been a while since TUSD first launched on Compound Finance and we at TUSD are pleased to see that it performed well over the past several months. Circulation and number of active addresses have increased dramatically with billions of dollars in monthly transaction volume. To take it a step further, we think that it would be an excellent opportunity to bring more members of the rapidly growing TUSD community to experience Compound and update TUSD market parameters.

https://snapshot.org/#/ is a tool that you use to figüre out if your community event wants something before it even goes to vote.

Architecture and Tools for DAO:

1. Voting Mechanism

The problem:

More token → More voteing power

Skin In The Game

The decision leads to a bad outcome, your tokens are axed.

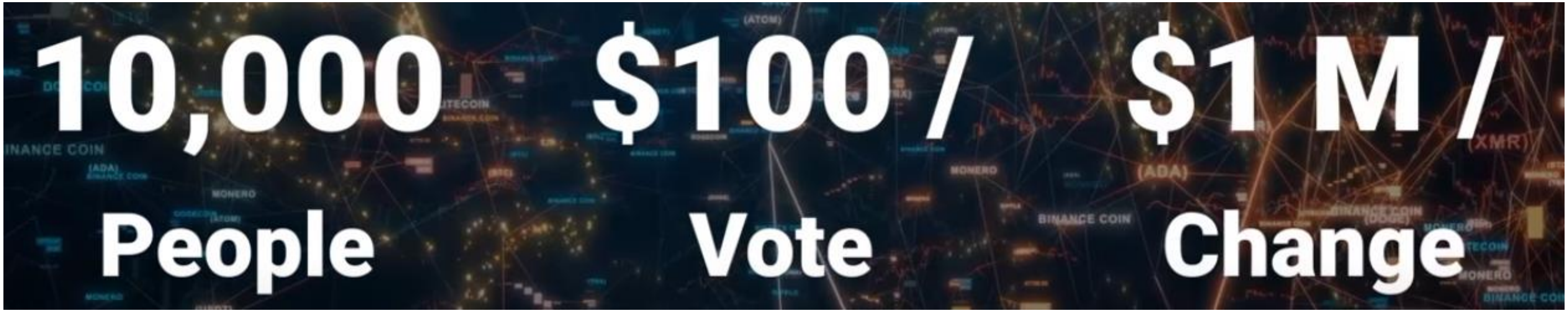
Proof of Personhood or Participation

Sybil resistance

Proof of personhood is basically just off chain data

IMPLEMENTATİON OF VOTING;

On Chain Voting


10,000 People

$100 / Vote

$1 M / Change

Governor C

Off Chain Voting

Using ORACLE

Replay side transactions in a single transaction

Saves Gas ✓

Efficient ✓

TOOLS THAT YOUCAN USE TO HELP GET YOU UP TO SPEED GUICKER:

There are a number of no code solutions that can go into building one of these daos.

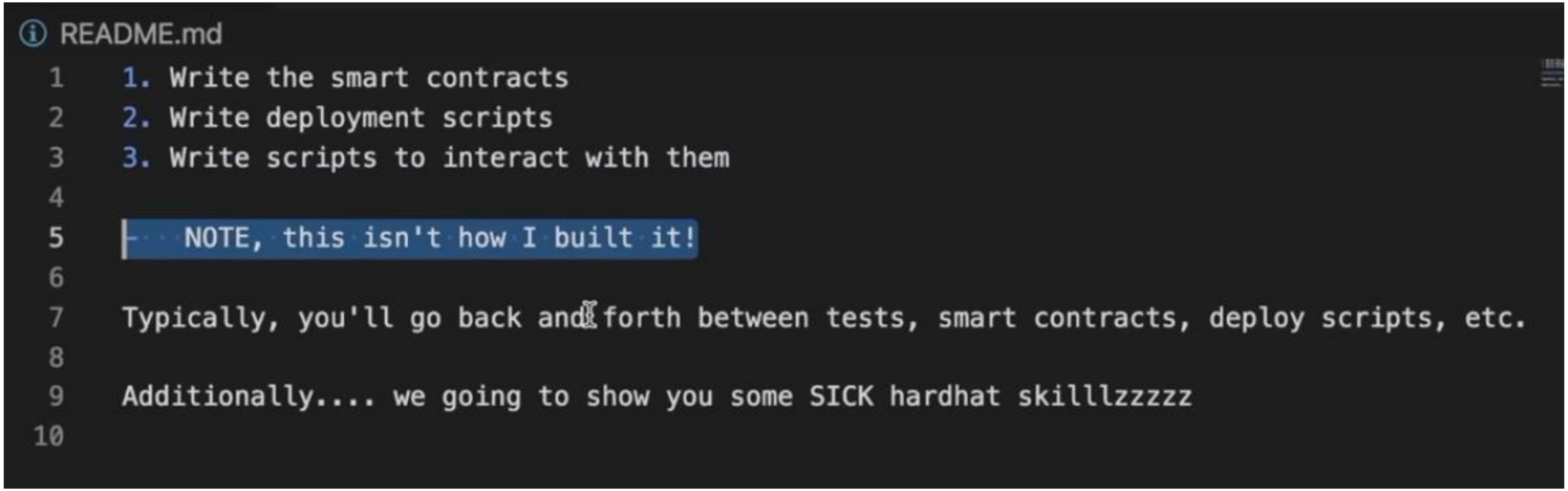
Gnosis Safe
Multisig

COLONY

Zodiac

Legality

How to Build a DAO 1.06.02.39



README.md

```
1. Write the smart contracts
2. Write deployment scripts
3. Write scripts to interact with them

-   NOTE, this isn't how I built it!

Typically, you'll go back and forth between tests, smart contracts, deploy scripts, etc.

Additionally.... we going to show you some SICK hardhat skilllzzzzz
```

Project folder is dao-template

yarn add --dev hardhat (if not install anything ---- with this waning → package-lock.json found. Your project contains lock files generated by tools other than Yarn.------ use yarn init command to create package.json)

```
yarn hardhat
```

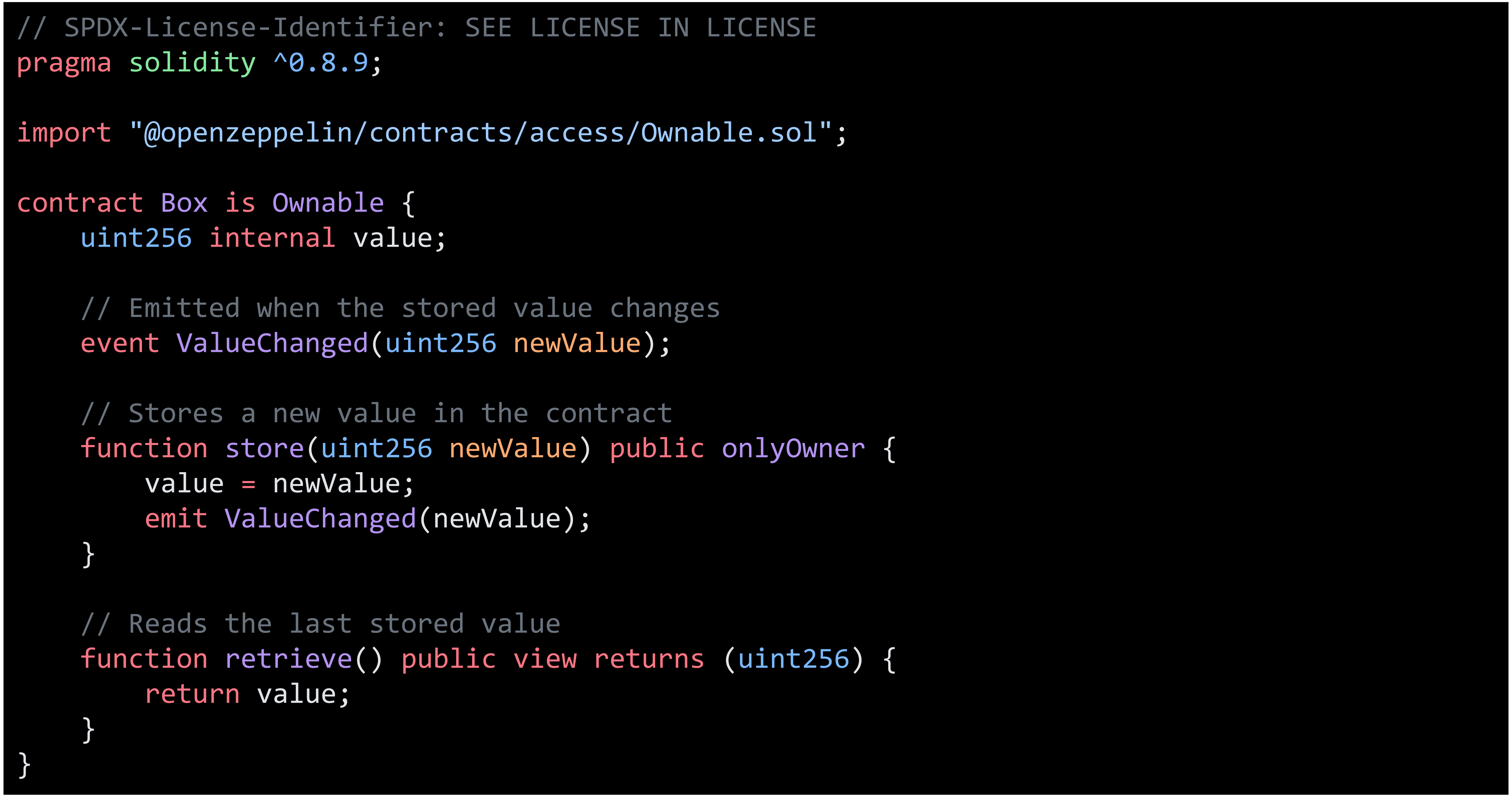
```
✔ What do you want to do? · Create an empty hardhat.config.js
Config file created

Give Hardhat a star on Github if you're enjoying it!

    https://github.com/NomicFoundation/hardhat
Done in 31.96s.
```

contracts/Box.sol

```
yarn add --dev @openzeppelin/contracts
```

Install solidity extension (Juan Blanco) in vs code.

```
yarn hardhat compile
```



```solidity
// SPDX-License-Identifier: SEE LICENSE IN LICENSE
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/access/Ownable.sol";

contract Box is Ownable {
    uint256 internal value;

    // Emitted when the stored value changes
    event ValueChanged(uint256 newValue);

    // Stores a new value in the contract
    function store(uint256 newValue) public onlyOwner {
        value = newValue;
        emit ValueChanged(newValue);
    }

    // Reads the last stored value
    function retrieve() public view returns (uint256) {
        return value;
    }
}
```

contracts/GovernanceToken.sol

```solidity
// SPDX-License-Identifier: SEE LICENSE IN LICENSE
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract GovernanceToken is ERC20 {
    uint256 public s_maxSupply = 1000000000000000000000000; // 1 milyon adet 1000000x10^18

    constructor() ERC20("GovernanceToken", "GT") {
        _mint(msg.sender, s_maxSupply);
    }
}
```

```
This is not a normal ERC20. See, when we do votes we need to make sure that its fair.
Imagine this for a second, someone knows a hot proposal is comng up. So they just buy a ton
of tokens and then they dump it after the votes over.

We want to avoid this.
We want to avoid people just buying and selling tokens to get in on governance.
So what we do is we actually;
       Create a snapshop of how many tokens people have at a certain block.
       And we want to make sure once a proposal goes throught we actually pick a snapshot
from the past that we want to use. This kind of incentivizes people to not just jump in when
its a proposal and jump out because once a proposal hits. It uses a block snapshot from the
past. So we are actually going to need to change this a little bit.

We ar gonna change this from ERC20 to ERC20 votes and we can actually see this in
openzeppelin extensions.
```

```
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";

contract GovernanceToken is ERC20Votes {
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Votes.sol

In IRC20Votes.sol some of the main functions are it has these checkpoints().

```
function checkpoints(address account, uint32 pos) public view virtual returns (Checkpoint memory) {
        return _checkpoints[account][pos];
    }
```

These checkpoints are basically Hey what is the snapshot? There is a numCheckpoints()

You can also delegate your tokens to different people. So maybe you are not going to be avaliable to actually vode.

So you say hey i am gonna give my tokens to somebody else .

You can get how many vodes somebody has passed with getPastVotes()

Or was apply it has all these functions that make this token much better as a voting tool.

```
    constructor()
        ERC20("GovernanceToken", "GT")
        ERC20Permit("GovernanceToken")
    {
```

```
function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override(ERC20Votes) {
        super._afterTokenTransfer(from, to, amount);
    }
```

```
Anytime we do this aftertokentransfer and the time we transfer a token we want to make
sure that we call the after token transfer of the ERC20Votes. And the reason that we do
this is because we want to make sure that the snapshops are updated. We want to make sure
we know how how many people have how many tokens at each block.
```

```
    function _mint(address to, uint256 amount) internal override(ERC20Votes) {
        super._mint(to, amount);
    }

    function _burn(address account, uint256 amount)
        internal
        override(ERC20Votes)
    {
        super._burn(account, amount);
    }
```

Same thing with the mint and burning. We want to make sure we always know how many tokens people have at different blocks or can be at different checkkpoints I should say.

And thats the most important bit at which checkpoint are you going to use for your token voting.

```
AND NOW WE HAVE A GOVERNANCE TOKEN AND erc20 TOKEN THAT WE CAN USE FOR GOVERNANCE.
```

```
yarn hardhat compile
```

```
NOW
       CREATE OUR GOVERNANCE CONTRACTS:
```

```
This is going to be the standart governance model. This is going to be this on chain ERC20
and I plan on updating this in the futere with no a governance off chain or something right.

In here we need two contract.
```

```
contracts/governance_standard/GovernorContract.sol
```

```
contracts/governance_standard/TimeLock.sol
```

contracts/governance_standard/GovernorContract.sol

```
This is going to be the contract that has all the voting code, all the voting logic
that our governance token is going to use.
```

contracts/governance_standard/TimeLock.sol

```
This is actually going to be an additional contract that is actually the owner. So the
timelock and the governor contract are sort of one in the same but the difference is
the timelock is actually going to be the owner of the box contract.

And this is important because whenever we propose or queue something to a proposal to
go through we want to wait. We want to wait for a new vote to be "executed".
```

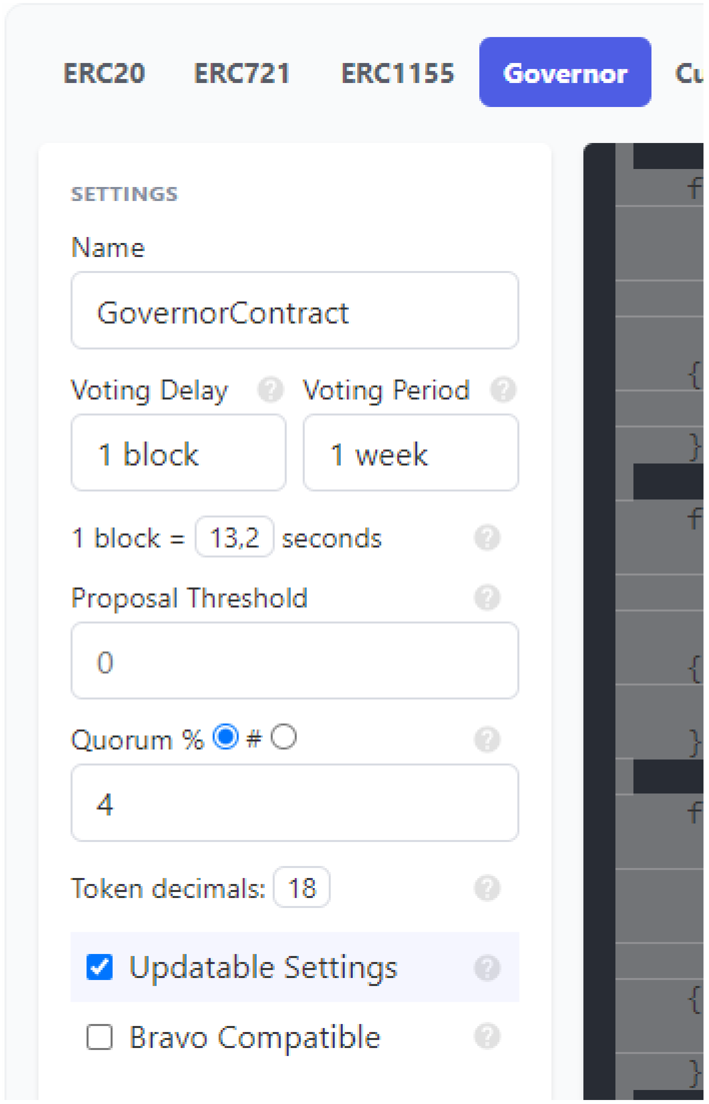
```
Why do we want to do that:
       lets say some proposal goes through thats bad. So like, lets say we have a box
contract and then a proposal goes through that says everyone who holds the governance token
has to pay five tokens or something like that or whatever or who knows?.

Maybe thats something that you dont really want to be a part of. So all of these governance
contracts give tive to users to get out if they dont like a governance update.
```

Sow we always want to have some type of timeline. So once a proposal passes it wont go in effect right away, it will have to wait some duration and then go in effect.

S thats what the timelin is gonna be for.

Governore contract is going to have all our actual code.

We can use openzeppelin wizard to create Governor contract.
https://docs.openzeppelin.com/contracts/4.x/wizard



ERC20 ERC721 ERC1155 **Governor** Cu

SETTINGS

Name: GovernorContract

Voting Delay: 1 block

Voting Period: 1 week

1 block = 13,2 seconds

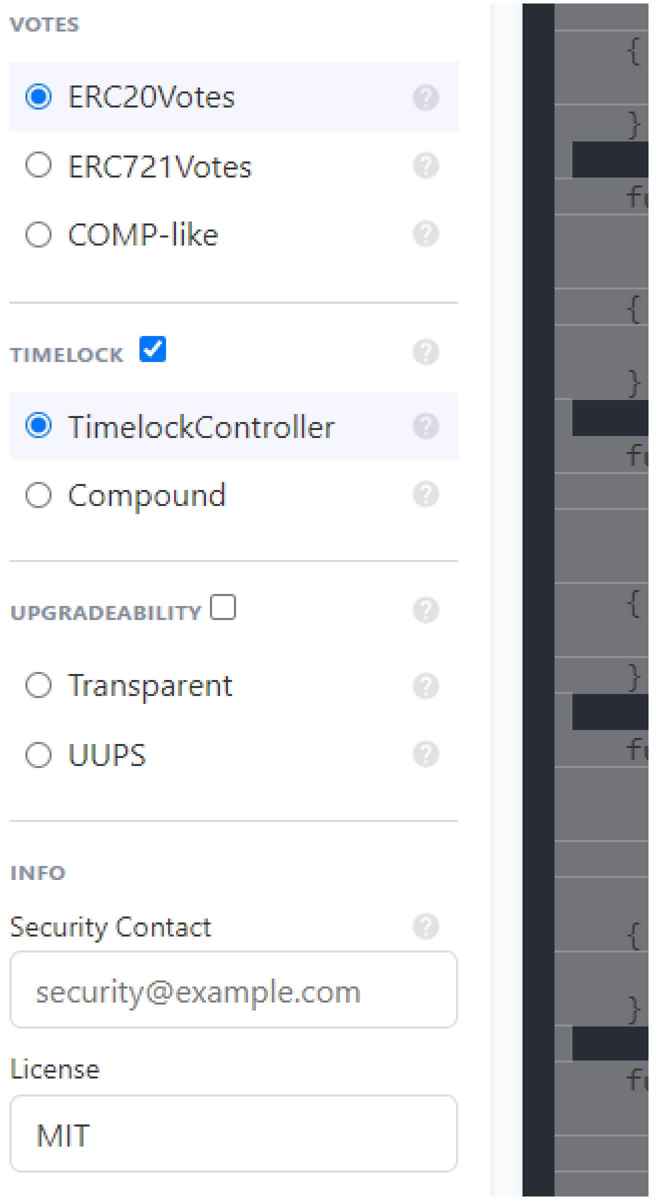
Proposal Threshold: 0

Quorum [✓ %] #: 4

Token decimals: 18

[✓] Updatable Settings

[ ] Bravo Compatible



VOTES

[✓ ERC20Votes]

ERC721Votes

COMP-like

TIMELOCK [✓]

[✓ TimelockController]

Compound

UPGRADEABILITY [ ]

Transparent

UUPS

INFO

Security Contact: security@example.com

License: MIT

Then copy codes to vs code

```
contract GovernorContract is
    Governor, // base contract
    GovernorSettings,
    GovernorCountingSimple, // is a way of counting votes
    GovernorVotes, // is a way of integrating with that ERC20 contract
    GovernorVotesQuorumFraction, // is a way to understand quorum time lock
    GovernorTimelockControl // time lock
{
```

```
All these are just implementations to make it easier to be governor.
```

```
votingDelay() → This is exactly the voting delay which we are gonna do süper.votingDelay().
We are gonna get from this governer settings contract that we are going to set in a minute
your voting period that we are going to set in our governer settings.
```

We will make contract a little bit more customizable.

```
constructor(
        IVotes _token,
        TimelockController _timelock,
    )
        Governor("GovernorContract")
        GovernorSettings(
            1, /* 1 block */
            45818, /* 1 week */
            0
        )
```

We have Ivotes token, this is going to be our governance token; TimeLockController timelock, this is going to be timeline controller that we make an amended.

We need this because we dont want to let any proposal just go through once it passes, we want to give people time to get out. And add votingdelay parameter and others

```
    constructor(
         IVotes _token,
         TimelockController _timelock,
         uint256 _voteingDelay,
         uint256 _votingPeriod,
         uint256 _quorumPercentage
     )
         Governor("GovernorContract")
         GovernorSettings(
             _voteingDelay, /* 1 block */
             _votingPeriod, /* 45818 blocks = ~ 1 week */
             0
         )
         GovernorVotes(_token)
         GovernorVotesQuorumFraction(_quorumPercentage)
         GovernorTimelockControl(_timelock)
     {}
```

```
Its ok for governance contract. Now we will code timelock.sol
```

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/governance/TimelockController.sol";

contract TimeLock is TimelockController {
    // minDelay : How long you have to wait before executing
    // proposers : list of addresses that can propose
    // executors : who can execute when a proposal passed
    constructor(
        uint256 minDelay,
        address[] memory proposers,
        address[] memory executors
    ) TimelockController(minDelay, proposers, executors) {}
}
```

TimeLock contract is going to be owning box contract, not the governance contract. The governor contract is where we are going to send our votes and stuff. But at the time lock that actually everything needs to flow through in order for governance to actually happen. Because we want to make sure we have mindelay, we go through the right process and everything.

```
yarn hardhat compile
```

```
Deploy tools instalation: npm:hardhat-deploy-ethers overrides hardhat-ethers
```

```
yarn add --dev @nomiclabs/hardhat-ethers@npm:hardhat-deploy-ethers ethers
```

```
yarn add --dev hardhat-deploy
```

```
DEPOLY SCRIPTS  IN TYPESCRIPT
```

```
deploy/01-deploy-governor-token.ts
```

```
CHANGE hardhat.config.js o ts for typescript
```

```
yarn add --dev typescript typechain ts-node @typechain/ethers-v5 @typechain/hardhat @types/chai @types/node
```

deploy/01-deploy-governor-token.ts

```
import { HardhatRuntimeEnvironment } from "hardhat/types";
import { DeployFunction } from "hardhat-deploy/types";
```

```
These are the two main things you need to create a deploy function with hardhat-deploy
```

```
const deployGovernanceToken: DeployFunction = async function (
  hre: HardhatRuntimeEnvironment
) {
  console.log("hello");
};
export default deployGovernanceToken;
```

hardhat.config.ts

```
import "hardhat-deploy";
import "@nomiclabs/hardhat-ethers";
import "@thpechain/hardhat";
/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.9",
};
```

```
Test deploy script
       yarn hardhat deploy
```

```
To list hardhat tasks:
       yarn hardhat --help
```

deploy/01-deploy-governor-token.ts

```
  console.log("hello");
  const { getNamedAccounts, deployments, network } = hre;
};
```

hardhat.config.ts

```
import { HardhatUserConfig } from "hardhat/config";
/** @type import('hardhat/config').HardhatUserConfig */
// module.exports = {
//   solidity: "0.8.9",
// };

const config: HardhatUserConfig = {
  defaultNetwork: "hardhat",
  networks: {
    hardhat: {
      chainId: 31337,
    },
    localhost: {
      chainId: 31337,
    },
  },
  solidity: "0.8.9",
  namedAccounts: {
    deployer: {
      default: 0,
    },
  },
};

export default config;
```

Hardhat network runs for tests,
localhost newwork runs for fake chain

deploy/01-deploy-governor-token.ts

```
  const { getNamedAccounts, deployments, network } = hre;
  const { deploy, log } = deployments;
  const { deployer } = await getNamedAccounts();
  log("Deploying Governance Token...");
  const governanceToken = await deploy("GovernanceToekn", {
    from: deployer,
    args: [],
    log: true,
    // waitConfirmations:
  });
  log(`Deployed governance token to address ${governanceToken.address}`);
  // verify
};
```

```
Yarn hardhat deploy
```

```
We are going to add one more thing here. We are going to add something called DELEGATE
FUNCTION.

Now when you actually deploy this contract nobody has voting power yet. The reason is because
noboby has the token delegated to them. We want to delegate this token to our deployer.

So we are going to call delegate function.
```

deploy/01-deploy-governor-token.ts

```
  log(`Deployed governance token to address ${governanceToken.address}`);
  // verify
};

const delegate = async (
  governanceTokenAddress: string,
  delegatedAccount: string
) => {
  const governanceToken = await ethers.getContractAt(
    "GovernanceToekn",
    governanceTokenAddress
  );
  const tx = await governanceToken.delegate(delegatedAccount);
  await tx.wait(1);
  console.log(
    `Checkpoints ${await governanceToken.numCheckpoints(delegatedAccount)}`
  );
};
```

With this delegate function call, we are doing, we have this delegate function that we havent used it. But when somebody calls us, we are saying, hey, you can use my vote, take my votes and vote however you want.

And with numcheckpoints in openzeppelin contract, we can see how many checkpoint that account actually has.

```
function getVotes(address account) public view virtual override returns (uint256) {
        uint256 pos = _checkpoints[account].length;
        return pos == 0 ? 0 : _checkpoints[account][pos - 1].votes;
    }
```

 Reason this is go important is because once again, like ı was saying when people do a vote, they do it based off some checkpoints. And anytime you transfer a token or delegate a token, basically call this function (_moveVotingPower),

```
 function _moveVotingPower(
        address src,
        address dst,
        uint256 amount
    ) private {
        if (src != dst && amount > 0) {
            if (src != address(0)) {
```

```
(uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[src], _subtract,
amount);
```

whick happens with the back end, which writes the checkpoint and says hey, at checkpoint x, heres what everybody has for voting Powers. And thats what theese are so important.

And ı know ı said before, its every block, but its acutally just every checkpoint, whenever these checkpoints are updated. Thats gonna be alot cheaper on gas than if we just did every single bock.

Now we will call delegate function in deploy script.

```
  log(`Deployed governance token to address ${governanceToken.address}`);
  // verify
  await delegate(governanceToken.address, deployer);
  log("Delegated");
};
```

```
Yarn hardhat deploy
```

```
yarn hardhat deploy
yarn run v1.22.19
warning ../../package.json: No license field
$ /home/eemcs/freecodecamp/dao-template/node_modules/.bin/hardhat deploy
Nothing to compile
No need to generate any newer typings.
hello
Deploying Governance Token...
deploying "GovernanceToken" (tx: 0xb49d2f92f8ba1131bbd0b79a9260a6c0cbb8faf85f906385ad3ba653767dc414)...:
deployed at 0x5FbDB2315678afecb367f032d93F642f64180aa3 with 3326235 gas
Deployed governance token to address 0x5FbDB2315678afecb367f032d93F642f64180aa3
Checkpoints 1
Delegated
Done in 6.25s.
```

```
Now we have one checkpoint. Because we just deployed , it was just delegated and
this address has one checkpoint.

The reason we checked for this is because if you see zero checkpoints here, it means
you havent delegated correctluy. So be sure to check for checkpoints. That s it
```

DEPLOY TIMELOCK CONTRACT:

deploy/02-deploy-time-lock.ts

```typescript
import { HardhatRuntimeEnvironment } from "hardhat/types";
import { DeployFunction } from "hardhat-deploy/types";
// @ts-ignore
import { ethers } from "hardhat";
import { MIN_DELAY } from "../helper-hardhat-config";

const deployTimeLock: DeployFunction = async function (
  hre: HardhatRuntimeEnvironment
) {
  //@ts-ignore
  const { getNamedAccounts, deployments } = hre;
  const { deploy, log } = deployments;
  const { deployer } = await getNamedAccounts();
  log("Deploying timelock....");

  const timeLock = await deploy("TimeLock", {
    from: deployer,
    args: [MIN_DELAY, [], []],
    log: true,
  });
};

export default deployTimeLock;
```

helper-hardhat-config.ts

```
export const MIN_DELAY = 3600;
```

```
yarn hardhat deploy
```

deploy/03-deploy-governor-contract.ts

```
import { HardhatRuntimeEnvironment } from "hardhat/types";
import { DeployFunction } from "hardhat-deploy/types";

const deployGovernorContract: DeployFunction = async function (
  hre: HardhatRuntimeEnvironment
) {
  //@ts-ignore
  const { getNamedAccounts, deployments } = hre;
  const { deploy, log, get } = deployments;
  const { deployer } = await getNamedAccounts();
  const governanceToken = await get("GovernanceToken");
  const timeLock = await get("TimeLock");
  log("Deploying Governor Contract....");
  const governorContract = await deploy("GovernorContract", {
    from: deployer,
    args: [governanceToken.address, timeLock.address],
  });
};
```

helper-hardhat-config.ts

```
export const MIN_DELAY = 3600;
export const VOTING_PERIOD = 5;
export const VOTING_DELAY = 1;
export const QUORUM_PERCENTAGE = 4;
```

deploy/03-deploy-governor-contract.ts

```
    args: [
      governanceToken.address,
      timeLock.address,
      VOTING_DELAY,
      VOTING_PERIOD,
      QUORUM_PERCENTAGE,
    ],
    log: true,
  });
  log(`Deployed governance token to address ${governorContract.address}`);
};

export default deployGovernorContract;
```

```
yarn hardhat deploy
```

```
HATA OLUŞTU
```

```
Error: ERROR processing /home/eemcs/freecodecamp/dao-template/deploy/03-deploy-governor-contract.ts:
Error: cannot estimate gas; transaction may fail or may require manual gas limit [ See: https://links.ethers.org/v5-errors-UNPREDICTABLE_GAS_LIMIT ] (reason="Transaction reverted: trying to deploy a contract whose code is too large",
```

```
TO FIX EROR:
```

```
hardhat.config.ts
```

```
solidity: {
    version: "0.8.9",
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
```

SET UP GOVERNANCE CONTRACTS:

This ones really important. Right now out timelock contract has no proposers and no executors.

We want to change that we want to only allow for the proposer to be the governor.

The governor contract should be the only one that proposes things the timelock and then anybody should be able to execute.

The way that this Works, we say the governance contract proposes something to the timelock once its in the timelock,and it waits that period, anybody can go ahead and execute it.

So governor contract everybody votes at everything, once a vote passes, governor says hey, can you please propose this? Timelock goes yeah, sure, but we got to wait this minimum delay. Once its been in delay happens anybody can execute it.

This would be really coll to do and integration with chain link keepers, by the way, for the chain link keeepes to automatically execute.

deploy/04-setup-governance-contracts.ts

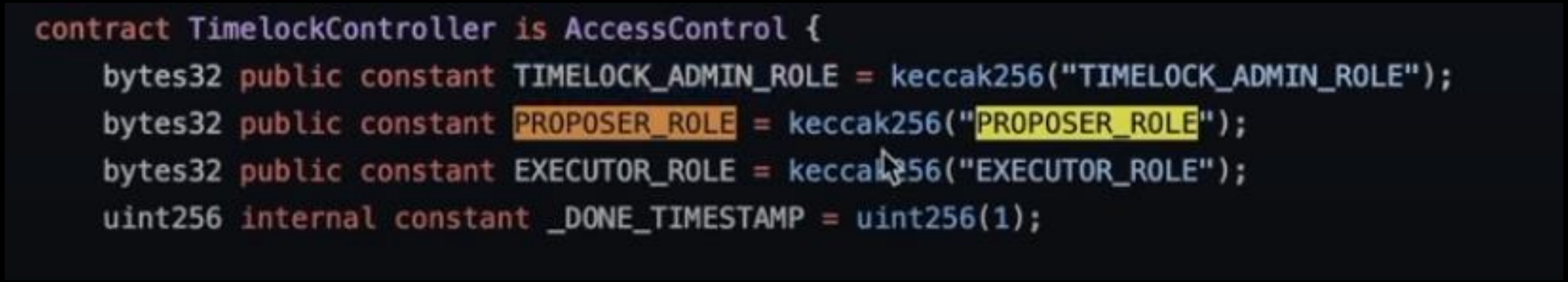
This is gonna be the code that does all the setting up.

```typescript
import { HardhatRuntimeEnvironment } from "hardhat/types";
import { DeployFunction } from "hardhat-deploy/types";
// @ts-ignore
import { ethers } from "hardhat";

const setupContracts: DeployFunction = async function (
  hre: HardhatRuntimeEnvironment
) {
  //@ts-ignore
  const { getNamedAccounts, deployments } = hre;
  const { deploy, log, get } = deployments;
  const { deployer } = await getNamedAccounts();
  const timeLock = await ethers.getContract("TimeLock", deployer); // to call functions
  const governor = await ethers.getContract("GovernorContract",deployer);
  log("Setting up roles....");
};

export default setupContracts;
```

And we are going to set up the roles again, we are setting it up so that only the governor can send things to this timelock, because the timelock is going to be you can almost think of the timelock as like president. So everything goes to the Senate the house representative which is the governor and then th President just says yeah sure, we just got to wait this minimum delay but the president will be the one to actually execute everything.

So the way that this Works is we are actually going to get the byte codes of different roles.

İf you look at Timelockcontroller.sol



```
contract TimelockController is AccessControl {
    bytes32 public constant TIMELOCK_ADMIN_ROLE = keccak256("TIMELOCK_ADMIN_ROLE");
    bytes32 public constant PROPOSER_ROLE = keccak256("PROPOSER_ROLE");
    bytes32 public constant EXECUTOR_ROLE = keccak256("EXECUTOR_ROLE");
    uint256 internal constant _DONE_TIMESTAMP = uint256(1);
```

There are some roles.These are just hashes of these strings here. Bu these are bytes32. Anybody who has this bytes32 is a proposer or executor etc.

Right now our deployer account is the timelock_admin. And that is bad. We dont want that we dont want anyone to be al timelock_admin. We dont want anyone to have power over this timelock. We dont want any centralized force her.

So we are going to get those roles.

```
 log("Setting up roles....");
  const proposerRole = await timeLock.PROPOSER_ROLE();
  const executorRole = await timeLock.EXECUTOR_ROLE();
  const adminRole = await timeLock.TIMELOCK_ADMIN_ROLE();
```

And we need to fix

```
const proposerTx = await timeLock.grantRole(proposerRole, governor.address);
```

So saying, okey, Governor, you are the only one who can actually do anything. Once you tell the timelock to do something we will wait for the timelock preiod tobe over and then we will be done.

```
  await proposerTx.wait(1);
  const executorTx = await timeLock.grantRole(executorRole, ADDRESS_ZERO);
// giving executor role to nobody
};
```

helper-hardhat-config.ts

```
export const ADDRESS_ZERO = "0x0000000000000000000000000000000000000000"; // 40
```

```
import { ADDRESS_ZERO } from "../helper-hardhat-config";
```

```
await executorTx.wait(1);
```

Now we need to revoke role right now out deployer count owns that time lock controller. And that s how we can actually do these transaction. We can actually grant role because our deployer account owns it now that we are given everybody Access and given all the decentralized Access we need. We want to revoke that role.

```
  const revokeTx = await timeLock.revokeRole(adminRole, deployer);
  await revokeTx.wait(1)
```

```
Anything thats i am like wants to do has to go through governance and noboy ownds the
timelockcontroller. It is currently after this runs it is impossible for anyone to do
anything with the timelock without governance happening.
```

DEPLOY BOX CONTRACT:

deploy/05-deploy-box.ts

```
import { HardhatRuntimeEnvironment } from "hardhat/types";
import { DeployFunction } from "hardhat-deploy/types";

const deployBox: DeployFunction = async function (
  hre: HardhatRuntimeEnvironment
) {
  //@ts-ignore
  const { getNamedAccounts, deployments } = hre;
  const { deploy, log, get } = deployments;
  const { deployer } = await getNamedAccounts();
  log("Deploying Box....")
  const box = await deploy("Box", {
    from: deployer,
    args: [],
    log: true,
  })
}
```

Our deployer has actually deployed this contract, not our time lock. So we want to give the boxes ownership over to our governance process.

```
    log: true,
  });

  const timeLock = await ethers.getContract("TimeLock");
```

```
//@ts-ignore
import { ethers } from "hardhat";
```

We are going to transfer the ownership of our box to this timelock, okay, and now so this is actually whats known as box deployment.Before we do that we have to get the box contract. So this is a box deployment object (const box = …) whick doesnt have contract functions we want to get the box contract object.

```
  const timeLock = await ethers.getContract("TimeLock");
  const boxContract = await ethers.getContractAt("Box", box.address);
  const transferOwnerTx = await boxContract.transferOwnership(timeLock.address);
  await transferOwnerTx.wait(1);
  log("YOU DONE IT !!!!");
};
export default deployBox;
```

We just did everything. We are deploying the governor token, deploying timelock which owns governance process, we are deployed the governance process, we are setting up the governance process so that its totally decentralized. And then we deployed and set up our box so that it only can be updated through a governance process.

TEST IT

```
yarn hardhat deploy
```

SUCCESSFULL.

```
NOW WE WILL WRITE THE SCRIPTS TO INTERACT WITH THIS DAO. And we can acctually do a
governance we can actually see exactly what the governance process looks like.
```

```
FIRST going to PROPOSE
       We are going to propose that our box contract stores the value 77 or something.
       Because when it first gets initialized its going to start with zero. So maybe
       we will propose it could start at 77.

Once proposing is done we are VOTEING ( SECOND ) on it. Once proposals it and we are going
to vote on whether or not we want the proposal to go throught right yes or no.

THIRD. And then if it passes we go to queue and execute. We queue first. And then we
execute.
```

scripts/propose.ts

We are actually going to propose on our governor contract.

```
//@ts-ignore
import { ethers } from "hardhat";

export async function propose() {
  const governor = await ethers.getContract("GovernorContract");
}
```

We are going to need the box contract. We are gonna say, hey, we want to propose the box contract changes the store value.

```
  const box = await ethers.getContract("Box");
}
```

In https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/governance/Governor.sol

```
function propose(
      address[] memory targets,
      uint256[] memory values,
      bytes[] memory calldatas,
      string memory description
   ) public virtual override returns (uint256) {
      require(
         getVotes(_msgSender(), block.number - 1) >= proposalThreshold(),
         "Governor: proposer votes below proposal threshold"
      );
```

First, in box contract, we are going to call the store function with new value. So we need to encode. And we also need to encode what we want to upgrade it to right so we have to code all the function parameters.

```
  const box = await ethers.getContract("Box");
  const encodedFunctionCall = box.interface.encodeFunctionData(
    functionToCall,
    args
  );
}
```

```
encodecFuncitonCallData() is turns it to being the bytes calldata in propose funciton.
```

```
function propose(
      address[] memory targets,
      uint256[] memory values,
      bytes[] memory calldatas,
```

```
Now we want to get args:
```

```
export async function propose(args: any[],
functionToCall: string) {
```

Now we will call the propose function in script:

helper-hardhat-config.ts

```
export const NEW_STORE_VALUE = 77;
export const FUNC = "store";
```

scripts/propose.ts

```
//propose([77], "store");
propose([NEW_STORE_VALUE], FUNC);
```

```
import { NEW_STORE_VALUE, FUNC } from
"../helper-hardhat-config";
```

```
    args
  );
  console.log(encodedFunctionCall);
}

//propose([77], "store");
propose([NEW_STORE_VALUE], FUNC)
  .then(() => process.exit(0))
  .catch((error) => {
    console.log(error);
    process.exit(1);
  });
```

```
TO TEST;
RUN LOCAL CHAIN
        yarn hardhat node
OPEN NEW CONSOLE
        yarn hardhat run scripts/propose.ts --network localhost
```

```
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$ yarn hardhat run scripts/propose.ts --network localhost
yarn run v1.22.19
warning ../../package.json: No license field
$ /home/eemcs/freecodecamp/dao-template/node_modules/.bin/hardhat run scripts/propose.ts --network localhost

// this is the endcodedfFunctionCAll output ========================

0x6057361d000000000000000000000000000000000000000000000000000000000000004d

Done in 7.95s.
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$
```

Going to add other arguments for proposal

```
export async function propose(
  args: any[],
  functionToCall: string,
  proposalDescription: string
) {
```

```
    args
  );
  console.log(encodedFunctionCall);
  console.log(`Proposing ${functionToCall} on ${box.address} with ${args}`);
  console.log(`Proposal Description: \n ${proposalDescription}`);
}
```

helper-hardhat-config.ts

```
export const PROPOSAL_DESCRIPTION = "Proposal
#1: Store 77 in the Box!";
```

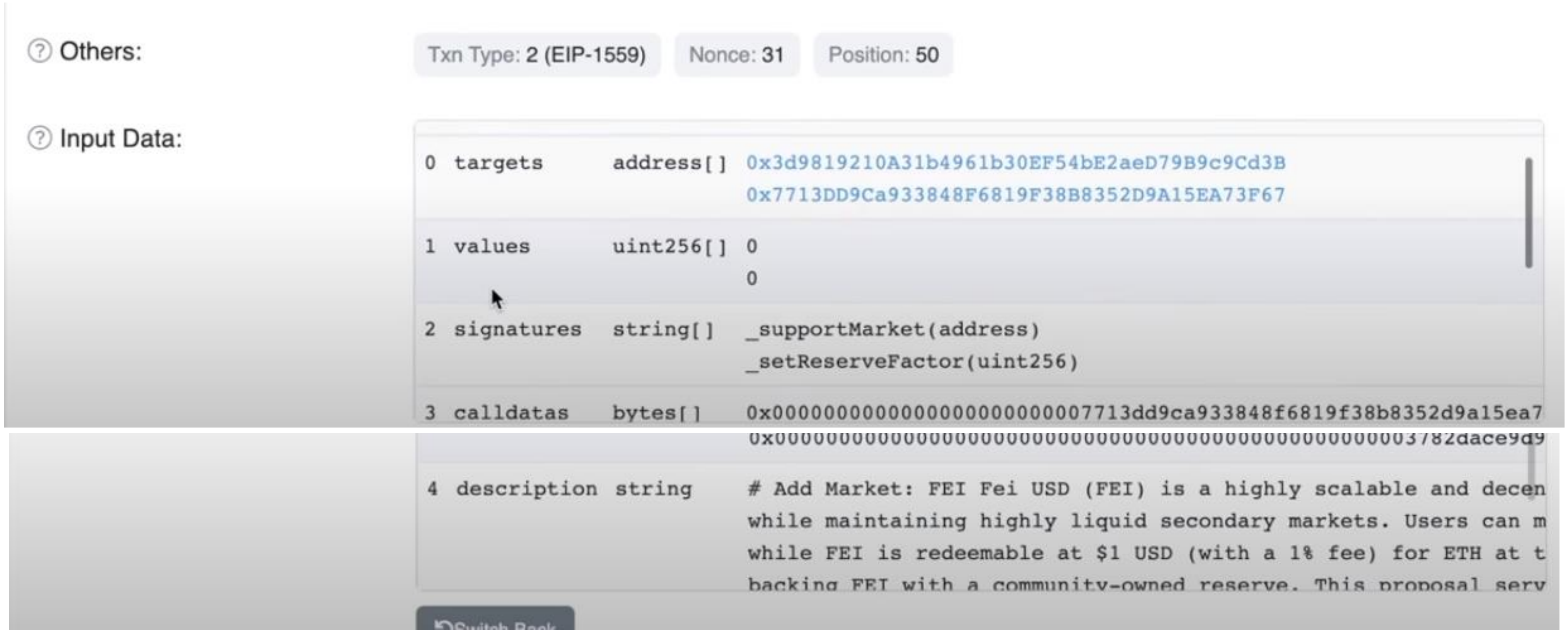
```
import {
  NEW_STORE_VALUE,
  FUNC,
  PROPOSAL_DESCRIPTION,
} from "../helper-hardhat-config";
```

```
//propose([77], "store");
propose([NEW_STORE_VALUE], FUNC,
PROPOSAL_DESCRIPTION)
  .then(() => process.exit(0))
  .catch((error) => {
    console.log(error);
    process.exit(1);
  });
```

MAKE TRANSACTION:

```
  console.log(`Proposal Description: \n ${proposalDescription}`);

  const proposeTx = await governor.propose(
    [box.address],
    [0],
    [encodedFunctionCall],
    proposalDescription
  );
  proposeTx.wait(1);
}
```

If you remember from compount this is going to be te exact same.



Others: Txn Type: 2 (EIP-1559) Nonce: 31 Position: 50

Input Data:

```
0 targets      address[] 0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B
                         0x7713DD9Ca933848F6819F38B8352D9A15EA73F67
1 values       uint256[] 0
                         0
2 signatures   string[]  _supportMarket(address)
                         _setReserveFactor(uint256)
3 calldatas    bytes[]   0x0000000000000000000000007713dd9ca933848f6819f38b8352d9a15ea7
                         0x00000000000000000000000000000000000000000000000003782dace9d9
4 description  string    # Add Market: FEI Fei USD (FEI) is a highly scalable and decen
                         while maintaining highly liquid secondary markets. Users can m
                         while FEI is redeemable at $1 USD (with a 1% fee) for ETH at t
                         backing FEI with a community-owned reserve. This proposal serv
```

Since we have a voting delay, people actually cant vote until the voting delay passes. Now with a local chain nobody actually processing blocks and time doesnt really pass as quick as we want.

We are just going to speed things up for our own testing purposes.

utils/move-blocks.ts

```
import { network } from "hardhat";
export async function moveBlocks(amount: number) {
  console.log("Moving blocks.....");
  for (let index = 0; index < amount; index++) {
    await network.provider.request({
      method: "evm_mine",
      params: [],
    });
  }
}
```

helper-hardhat-config.ts

```
export const developmentChains = ["hardhat", "localhost"];
```

scripts/propose.ts

```
import { ethers, network } from "hardhat";
import { moveBlocks } from "../utils/move-blocks";
import {
  NEW_STORE_VALUE,
  FUNC,
  PROPOSAL_DESCRIPTION,
  developmentChains, VOTING_DELAY
} from "../helper-hardhat-config";
```

```
  proposeTx.wait(1);
  if (developmentChains.includes(network.name)) {
    await moveBlocks(VOTING_DELAY + 1);
  }
}
```

This proposed transaction does some stuff that we actually want. So one of the big things that it wants is, it has this proposalId.

```
uint256 proposalId = hashProposal(targets, values, calldatas, keccak256(bytes(description)));
```

And event that it iemits, it ends up emitting this proposal ID, we actually need the proposalId for later on when we actually go to vote.

```
emit ProposalCreated(
        proposalId,
        _msgSender(),
        targets,
        values,
        new string[](targets.length),
        calldatas,
        snapshot,
        deadline,
        description
    );
```

To get proposal id

```
  const proposeReceipt = await proposeTx.wait(1);
  if (developmentChains.includes(network.name)) {
    await moveBlocks(VOTING_DELAY + 1);
  }
  const porposalId = proposeReceipt.event[0].args.porposalId;
}
```

Something else that we want to do is maybe we want to see what the deadline is with a snapshot, you can go ahead and check github repo.

We want to save this proposalId. Our other scripts know what this proposal id is going to be when we run those.

Create proposal.json in Project root folder.

proposals.json

```
{ "31337": [] }
```

helper-hardhat-config.ts

```
export const proposalsFile = "proposals.json";
```

scripts/propose.ts

```
import {
……………………………..
  proposalsFile,
} from "../helper-hardhat-config";
```

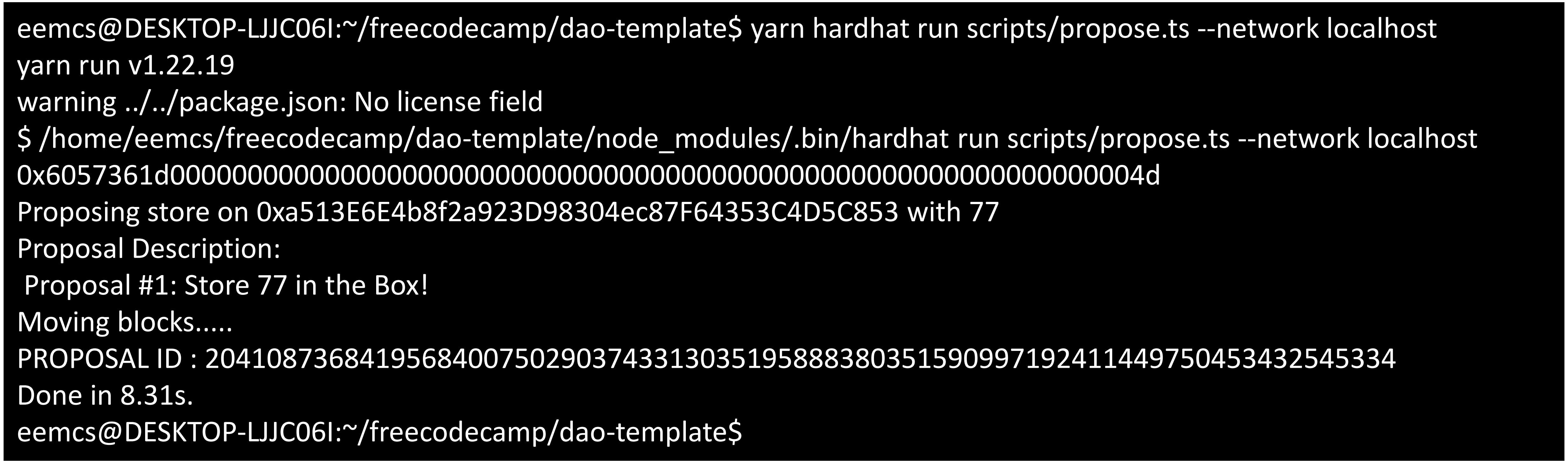
```
yarn add fs
```

```
import * as fs from "fs";
```

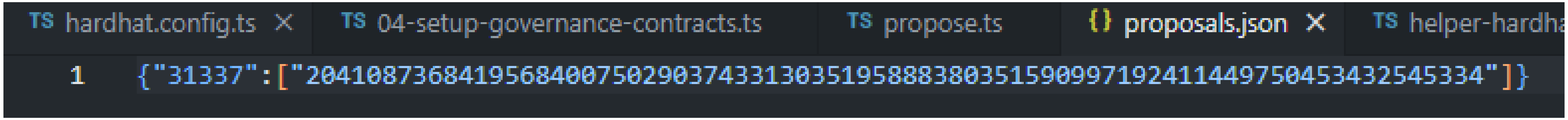
```
  let proposals = JSON.parse(fs.readFileSync(proposalsFile, "utf8"));
}
```

```
  const porposalId = proposeReceipt.events[0].args.proposalId;

  let proposals = JSON.parse(fs.readFileSync(proposalsFile, "utf8"));
  proposals[network.config.chainId!.toString()].push(porposalId.toString());
  fs.writeFileSync(proposalsFile, JSON.stringify(proposals));
}
```

```
TEST IT:
       yarn hardhat node

ANOTHER CONSOLE
       yarn hardhat run scripts/propose.ts --network localhost
```



```
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$ yarn hardhat run scripts/propose.ts --network localhost
yarn run v1.22.19
warning ../../package.json: No license field
$ /home/eemcs/freecodecamp/dao-template/node_modules/.bin/hardhat run scripts/propose.ts --network localhost
0x6057361d000000000000000000000000000000000000000000000000000000000000004d
Proposing store on 0xa513E6E4b8f2a923D98304ec87F64353C4D5C853 with 77
Proposal Description:
 Proposal #1: Store 77 in the Box!
Moving blocks.....
PROPOSAL ID : 20410873684195684007502903743313035195888380351590997192411449750453432545334
Done in 8.31s.
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$
```



```
{"31337":["20410873684195684007502903743313035195888380351590997192411449750453432545334"]}
```

CREATE VOTE SCRIPT:

```
import * as fs from "fs";
import { proposalsFile } from "../helper-hardhat-config";
import { network } from "hardhat";

const index = 0;

async function main(proposalIndex: number) {
  const proposals = JSON.parse(fs.readFileSync(proposalsFile, "utf8"));
  const proposalId = proposals[network.config.chainId!][proposalIndex];
  // 0 = Against, 1 = for, 2 = abstain-this cost gas
  const voteWay = 1;
}
main(index)
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

There is three type of vote

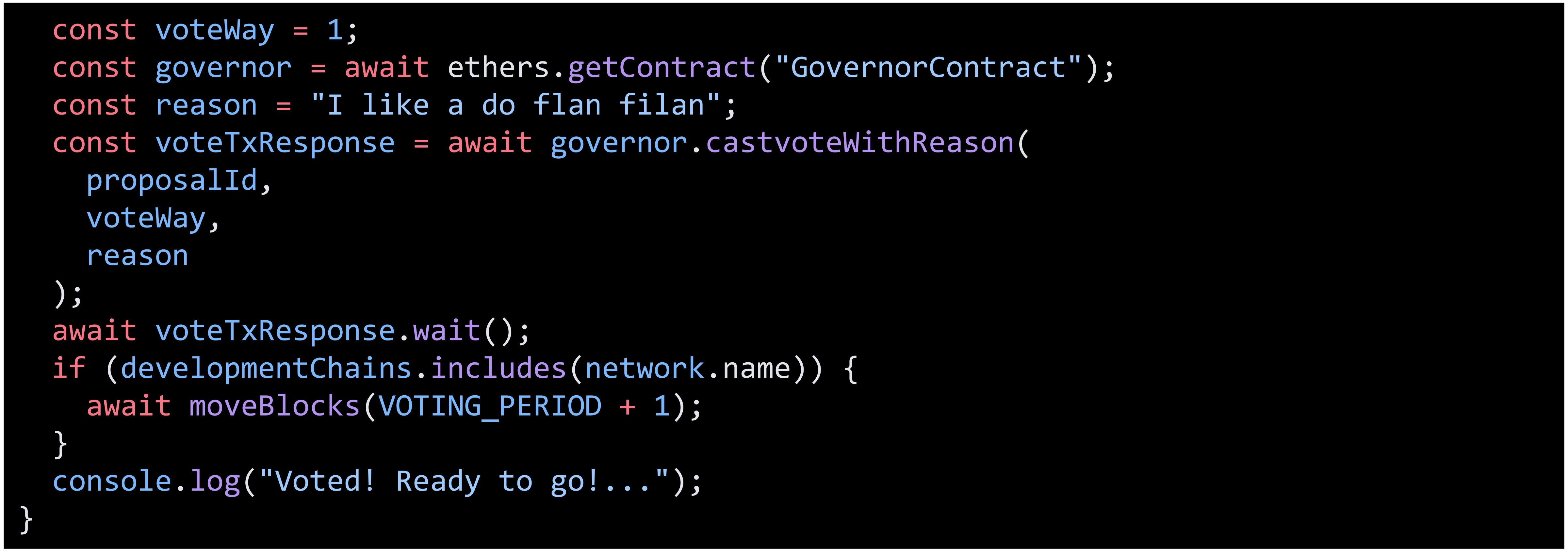
In the contract there is three vote type . castVote() castVoteWithReason() and castVoteBySig() in

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/governance/Governor.sol

https://forum.openzeppelin.com/t/what-is-votecastbysig/17069

My hunch is that anyone could then execute this vote on behalf of me if I didn't send the tx or something?

Yes, exactly! This method implements a meta-transaction, and allows a project to subsidize voting fees. The voters can generate a signature for free, and the project can then submit those and pay for the gas.



```javascript
  const voteWay = 1;
  const governor = await ethers.getContract("GovernorContract");
  const reason = "I like a do flan filan";
  const voteTxResponse = await governor.castvoteWithReason(
    proposalId,
    voteWay,
    reason
  );
  await voteTxResponse.wait();
  if (developmentChains.includes(network.name)) {
    await moveBlocks(VOTING_PERIOD + 1);
  }
  console.log("Voted! Ready to go!...");
}
```

```
import * as fs from "fs";
import {
  developmentChains,
  proposalsFile,
  VOTING_PERIOD,
} from "../helper-hardhat-config";
import { network } from "hardhat";
//@ts-ignore
import { ethers } from "hardhat";
import { moveBlocks } from "../utils/move-blocks";
const index = 0;
```

I checked the proposal state is because there is this state function in the governor contract.

function state(uint256 proposalId) public view virtual override returns (ProposalState) {

What this does is it tells us what the state of the proposals.

```
 if (proposal.executed) {
            return ProposalState.Executed;
        }

        if (proposal.canceled) {
            return ProposalState.Canceled;
        }

 if (snapshot == 0) {
            revert("Governor: unknown proposal id");
        }

        if (snapshot >= block.number) {
            return ProposalState.Pending;
        }
```

```
And what you are usulally looking for this worm reached and vote succedded. If both of
these happen the proposal state succeeded. Otherwise it defeated or its not there yet.

if (_quorumReached(proposalId) && _voteSucceeded(proposalId)) {
            return ProposalState.Succeeded; // accept this is 1
        } else {
            return ProposalState.Defeated; // accept htis is 0
        }
```

```
TEST IT;
       be sure local node is running
       be sure runned propose script
```

```
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$ yarn hardhat run scripts/vote.ts --network localhost
yarn run v1.22.19
warning ../../package.json: No license field
$ /home/eemcs/freecodecamp/dao-template/node_modules/.bin/hardhat run scripts/vote.ts --network localhost
Moving blocks.....
Voted! Ready to go!...
Done in 8.26s.
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$
```

Go to the hardhat console

```
       yarn hardhat console --network localhost
```

And check the state

Paste this line to console

```
> const governor = await ethers.getContract("GovernorContract");
Output : undefined
```

And run; (copy proposal id from proposals.json file)

```
> await
governor.state("2041087368419568400750290374331303519588838035159099719241144975045343254533
4")
Output : 4
```

THE STATE IS 4

The proposal state is actually in the Igovernor.sol interface of the governor. We can see 0 is pendig, 1 is active, two is cancelled….

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/governance/IGovernor.sol

```
enum ProposalState {
      Pending, // 0
      Active, // 1
      Canceled,
      Defeated,
      Succeeded, // 4
      Queued,
      Expired,
      Executed
  }
```

```
So four-4 is succeded
```

Our proposal in succeded state. And we have actually moved the blocks along the voting priod. So voting is now over because we cheated.

No we are going to queue and execute this.

scripts/queue-and-execute.ts

```
export async function queueAndExecute() {}
queueAndExecute()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Queue() function in the https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/governance/extensions/GovernorTimelockControl.sol

```
function queue(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        bytes32 descriptionHash
    ) public virtual override returns (uint256) {
```

It is excatly same as porpose

```
export async function queueAndExecute() {
  const args = [NEW_STORE_VALUE];
  const box = await ethers.getContract("Box");
  const encodedFunctionCall = box.interface.encodeFunctionData(FUNC, args);
  const descriptionHash = ethers.utils.keccak256(
    ethers.utils.toUtf8Bytes(PROPOSAL_DESCRIPTION)
  );

  const governor = await ethers.getContract("GovernorContract");
  console.log("Queueing....");
  const queueTx = await governor.queue(
    [box.address],
    [0],
    [encodedFunctionCall],
    descriptionHash
  );
  await queueTx.wait(1);
  if (developmentChains.includes(network.name)) {
    await moveTime(MIN_DELAY + 1);
    await moveBlocks(1);
  }
}
```

utils/move-time.ts

```
import { network } from "hardhat";
export async function moveTime(amount: number) {
  console.log("Moving time.....");
  await network.provider.send("evm_increaseTime", [amount]);
  console.log(`Moved forward ${amount} seconds`);
}
```

```
import {
  NEW_STORE_VALUE,
  FUNC,
  PROPOSAL_DESCRIPTION,
  developmentChains,
  MIN_DELAY,
} from "../helper-hardhat-config";
//@ts-ignore
import { ethers, network } from "hardhat";
import { moveTime } from "../utils/move-time";
import { moveBlocks } from "../utils/move-blocks";
```

ADD EXECUTE :

```
console.log("Executing....");
const executeTx = await governor.execute(
  [box.address],
  [0],
  [encodedFunctionCall],
  descriptionHash
);
await executeTx.wait(1);
const boxNewValue = await box.retrieve();
console.log(`New Box Value: ${boxNewValue.toString()}`);
```

```
TEST IT;
        be sure local node is running
                yarn hardhat localhost
        be sure runned propose script
                yarn hardhat run scripts/propose.ts --network localhost
        be sure runned vote script
                yarn hardhat run scripts/vote.ts --network localhost

Then test
        yarn hardhat run scripts/queue-and-execute.ts --network localhost
```

```
WARNING : If queue was runned
but execute was not, in second
try be sure  to passive queue
just be sure to run execute.
```

```
yarn hardhat run scripts/queue-and-execute.ts --network localhost
yarn run v1.22.19
warning ../../package.json: No license field
$ /home/eemcs/freecodecamp/dao-template/node_modules/.bin/hardhat run scripts/queue-and-execute.ts --network
localhost
Queueing....
Moving time.....
Moved forward 3601 seconds
Moving blocks.....
Executing....
New Box Value: 77
Done in 8.50s.
eemcs@DESKTOP-LJJC06I:~/freecodecamp/dao-template$
```

THE END OF LESSON