KAIST
School of Computing

Web3@KAIST
Building Web3 Apps to Solve Real Problems

Building Web3 & Blockchain Applications
(CS492 Special Topics in Computer Science)
Spring 2023

Developing Governance

Lecture 19 (2023-05-17)

Jason Han, Ph.D

Adjunct Professor of KAIST School of Computing
Founder of Ground X & Klaytn

web3classdao@gmail.com
http://web3classdao.xyz/kaist/

Today’s Lecture 17 Overview

- **Lecture Objective**

  - Understanding access control in a smart contract

  - Understanding on-chain and off-chain governance

  - Learning how to develop on-chain governance in Solidity

  - Learning how to upgrade a smart contract

- **Lecture will cover**

  - Access Control

  - Governance

  - Upgradeable contracts

References for the lecture

- Dapp University: Governance (Github)
- Ultimate Web3, Full Stack Solidity, and Smart Contract Course by Patrick Collins

  - Lesson 17: Hardhat DAOs

  - Lesson 16: Hardhat Upgrades
- OpenZeppelin access control
- OpenZeppelin governance
- OpenZeppelin governance code
- How to Create a DAO Governance Token (Alchemy)
- Compound governance
- Tally: Explore DAOs
- Open Zeppelin Upgradeable Contracts (Github)
- Open Zeppelin Upgradeable ERC20
- Writing Upgradeable Contracts (OpenZeppelin)
- OpenZeppelin Upgrades: Step by Step Tutorial for Hardhat

A governance & upgradeable contracts

Examples from various sites
with some modification

Clone the code here!

git clone https://github.com/web3classdao/onchain-governance.git
git clone https://github.com/web3classdao/upgradeable-contracts.git

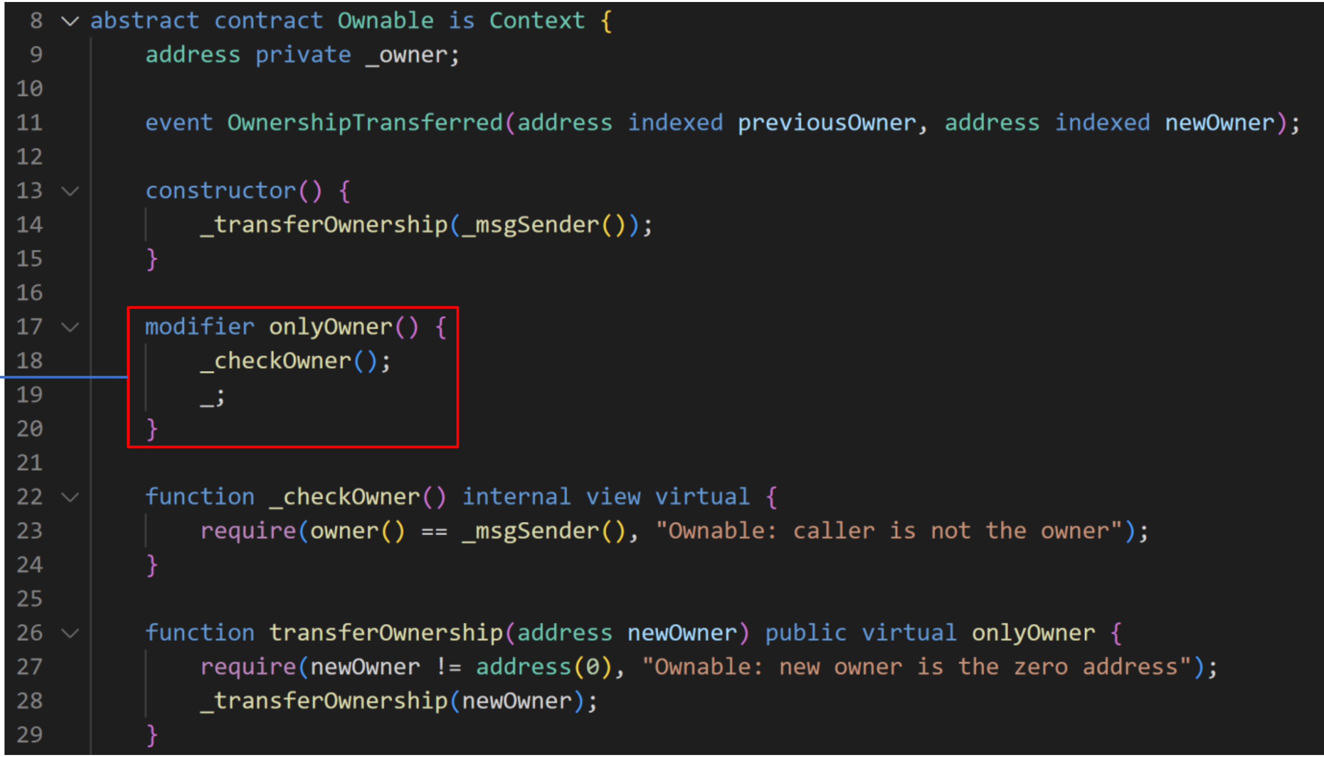
Access Control

Access Control so far

- *The contract deployer (owner) by EOA*
- *The contract deployer (owner) by another contract*
- *msg.sender (usually EOA/contracts calling a function)*

Need more granular access control

OpenZeppelin Ownable.sol



```
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor() {
        _transferOwnership(_msgSender());
    }

    modifier onlyOwner() {
        _checkOwner();
        _;
    }

    function _checkOwner() internal view virtual {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _transferOwnership(newOwner);
    }
```

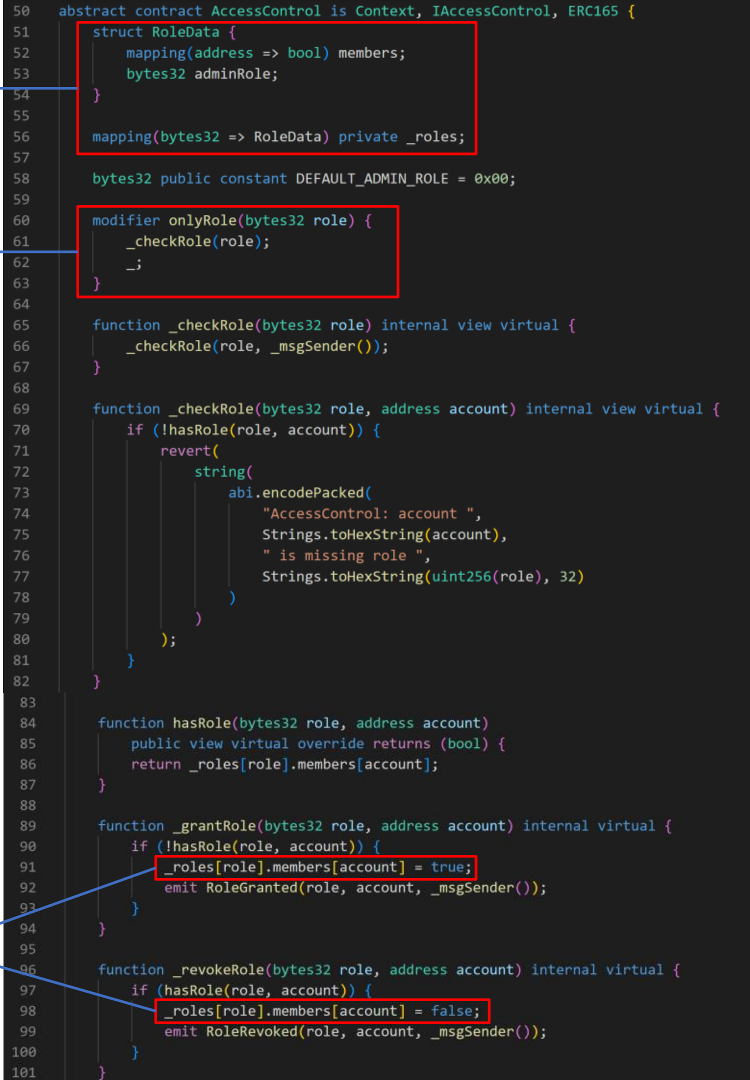
Use the modifier to check the calling address is the owner

OpenZeppelin AccessControl.sol

addresses that have a role
the admin of a role

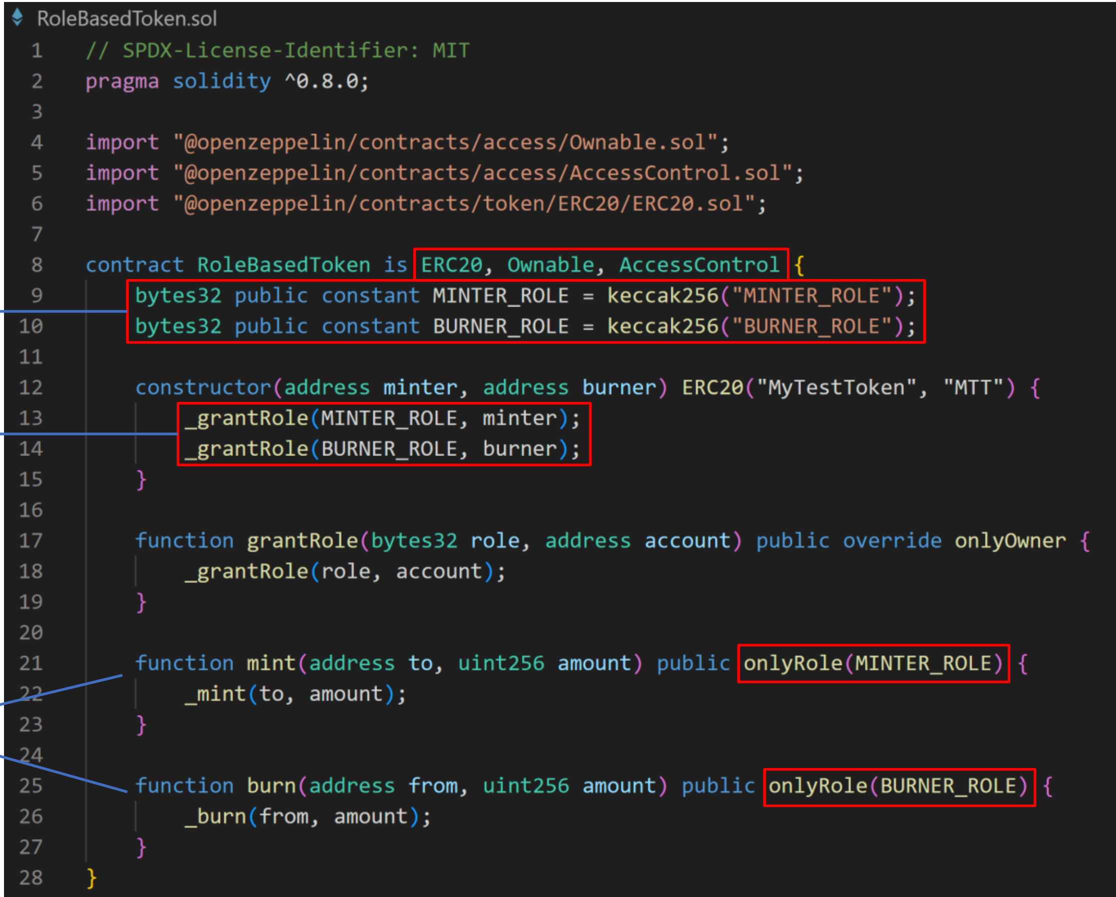
Use the modifier
to check the calling account has the role

Granting and revoking a role are
the same as changing the _roles variable



```solidity
abstract contract AccessControl is Context, IAccessControl, ERC165 {
    struct RoleData {
        mapping(address => bool) members;
        bytes32 adminRole;
    }

    mapping(bytes32 => RoleData) private _roles;

    bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

    modifier onlyRole(bytes32 role) {
        _checkRole(role);
        _;
    }

    function _checkRole(bytes32 role) internal view virtual {
        _checkRole(role, _msgSender());
    }

    function _checkRole(bytes32 role, address account) internal view virtual {
        if (!hasRole(role, account)) {
            revert(
                string(
                    abi.encodePacked(
                        "AccessControl: account ",
                        Strings.toHexString(account),
                        " is missing role ",
                        Strings.toHexString(uint256(role), 32)
                    )
                )
            );
        }
    }

    function hasRole(bytes32 role, address account)
        public view virtual override returns (bool) {
        return _roles[role].members[account];
    }

    function _grantRole(bytes32 role, address account) internal virtual {
        if (!hasRole(role, account)) {
            _roles[role].members[account] = true;
            emit RoleGranted(role, account, _msgSender());
        }
    }

    function _revokeRole(bytes32 role, address account) internal virtual {
        if (hasRole(role, account)) {
            _roles[role].members[account] = false;
            emit RoleRevoked(role, account, _msgSender());
        }
    }
```

Role-based ERC20 Token Example



RoleBasedToken.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract RoleBasedToken is ERC20, Ownable, AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

    constructor(address minter, address burner) ERC20("MyTestToken", "MTT") {
        _grantRole(MINTER_ROLE, minter);
        _grantRole(BURNER_ROLE, burner);
    }

    function grantRole(bytes32 role, address account) public override onlyOwner {
        _grantRole(role, account);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }

    function burn(address from, uint256 amount) public onlyRole(BURNER_ROLE) {
        _burn(from, amount);
    }
}
```

A role is set as bytes32 constant to save the storage

Use _grantRole internal function of AccessControl

Use the onlyRole modifier with a role argument

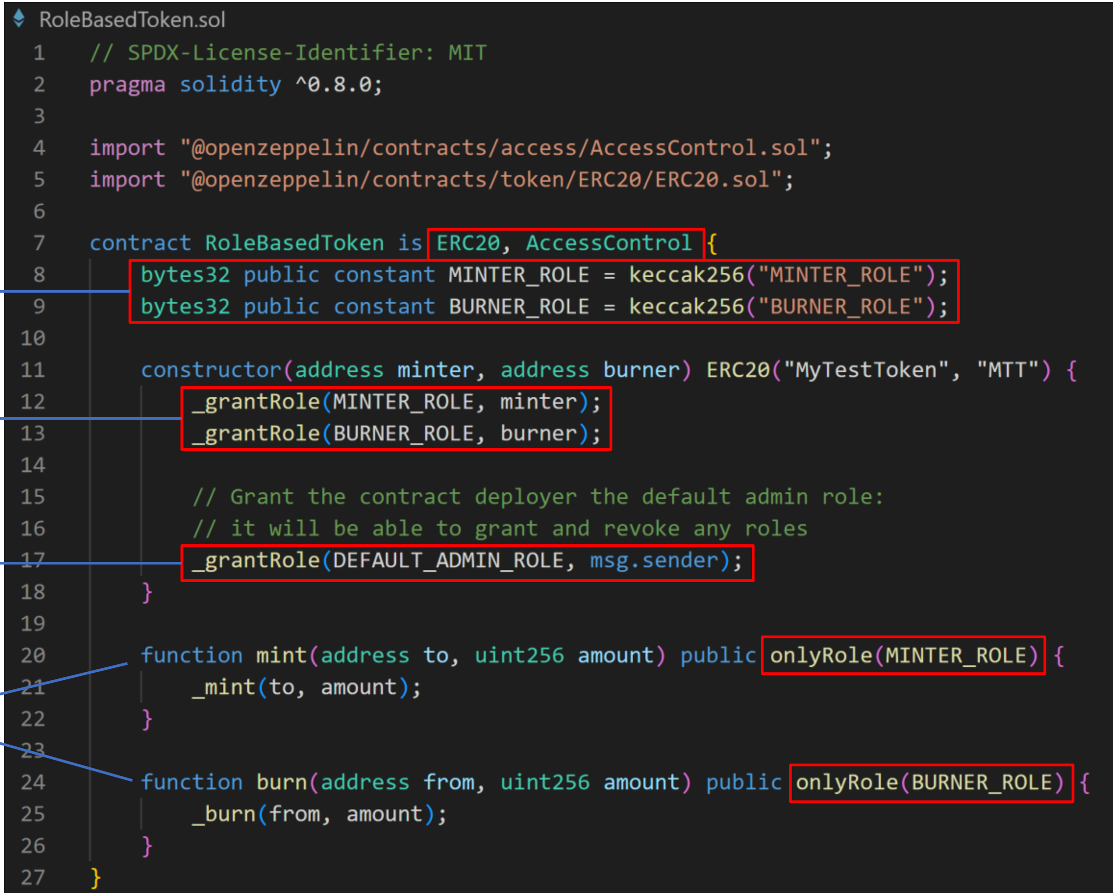
Role-based ERC20 Token Example

A role is set as bytes32 constant to save the storage

Use _grantRole internal function of AccessControl

Grant the deployer the admin role If granting roles more accounts, the deployer can call the grantRole()

Use the onlyRole modifier with a role argument



RoleBasedToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

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

contract RoleBasedToken is ERC20, AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
    bytes32 public constant BURNER_ROLE = keccak256("BURNER_ROLE");

    constructor(address minter, address burner) ERC20("MyTestToken", "MTT") {
        _grantRole(MINTER_ROLE, minter);
        _grantRole(BURNER_ROLE, burner);

        // Grant the contract deployer the default admin role:
        // it will be able to grant and revoke any roles
        _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }

    function burn(address from, uint256 amount) public onlyRole(BURNER_ROLE) {
        _burn(from, amount);
    }
}
```

Governance

What is **Governance?**

Governance

- **Governance is the systems in place that allow decisions to be made**

  - In a typical organizational structure, the executive team or a board of directors may have the final say in decision-making, or shareholders vote on proposals to enact change

  - In a political system, elected officials may enact legislation that attempts to represent their constituents' desires

- **Decentralized governance**: No one person or authority owns or controls the governance

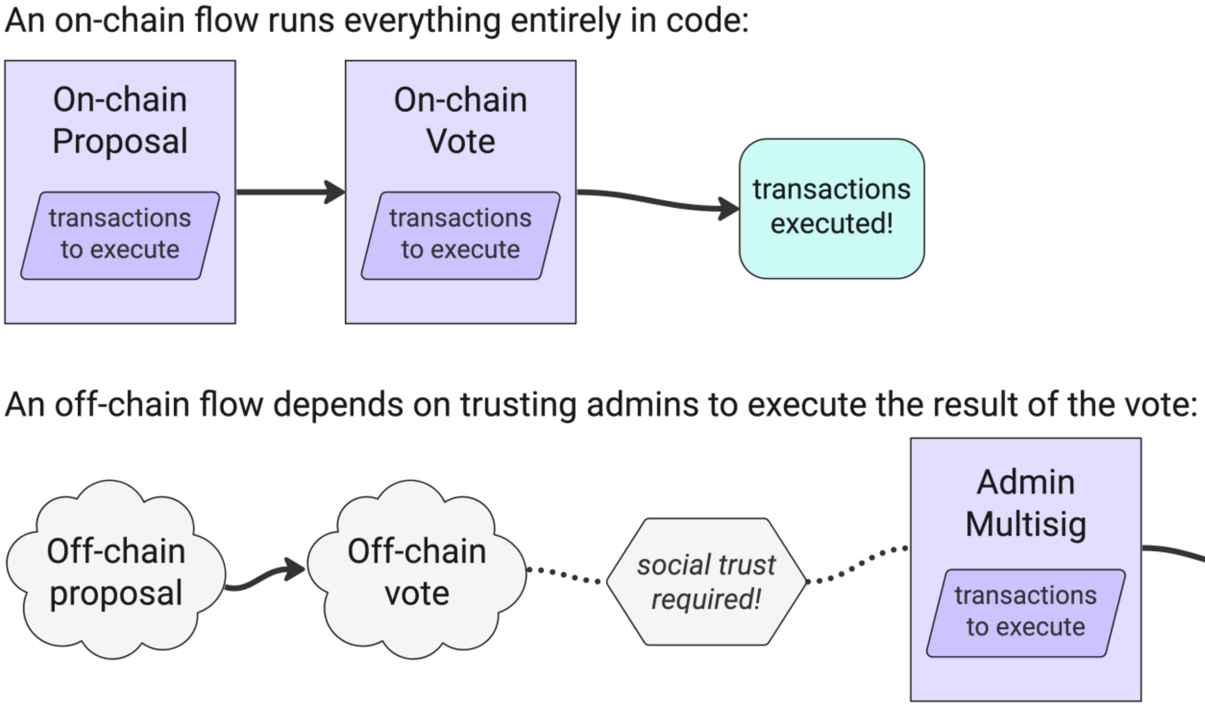
  - Blockchain is used to implement decentralized governance

  - **On-chain governance**: when proposed protocol changes are decided by a stakeholder vote, usually by holders of a governance token, and voting happens on the blockchain. Examples are Compound and Uniswap.

  - **Off-chain governance**: where any protocol change decisions happen through an informal process of social discussion, which, if approved, would be implemented in code. An example is Ethereum.

https://ethereum.org/en/governance/

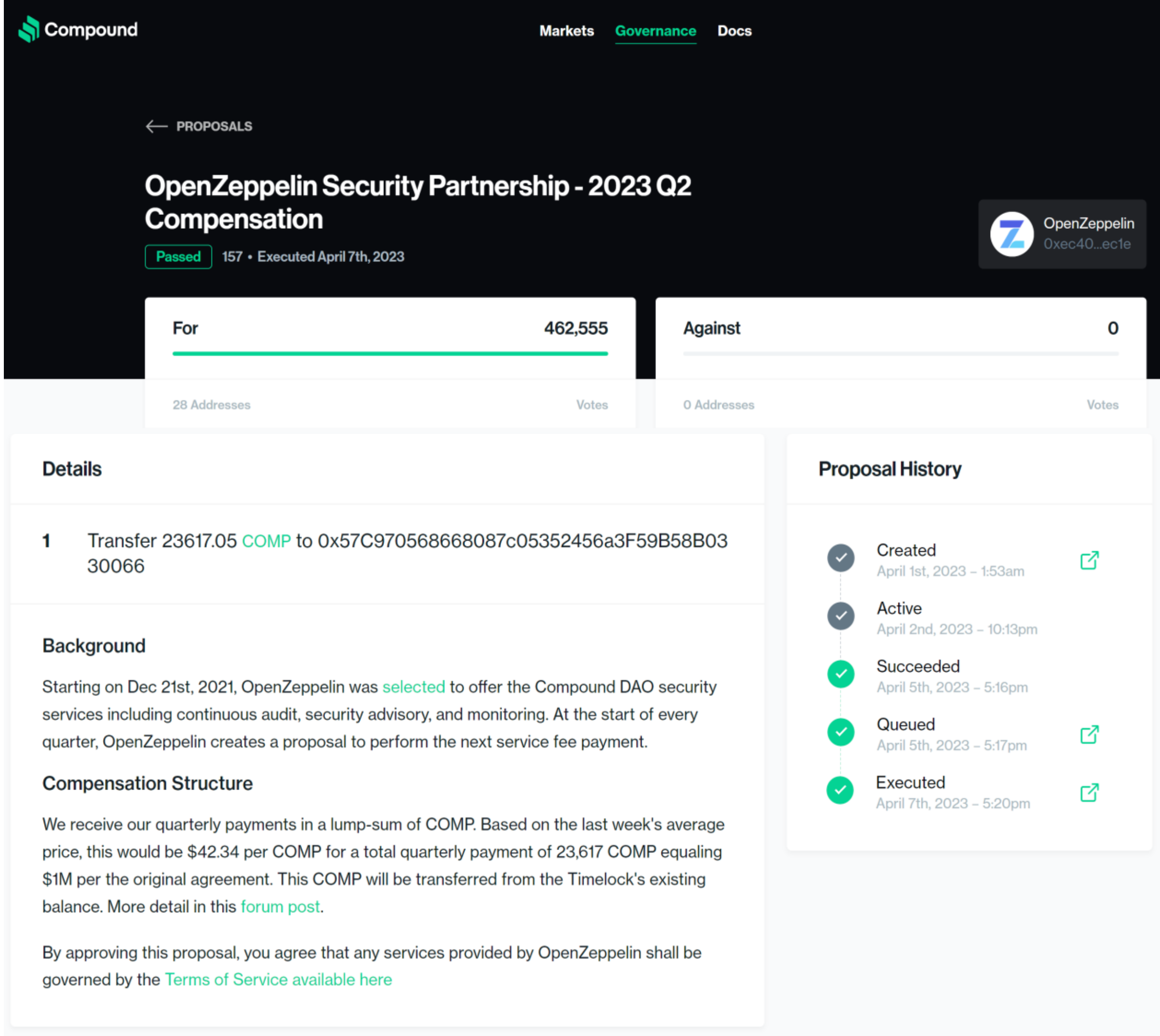
On-chain vs. Off-chain governance



https://docs.tally.xyz/knowledge-base/tally/on-chain-vs-off-chain

On-chain Governance Example

Compound
Service fee payment



OpenZeppelin Security Partnership - 2023 Q2 Compensation

Passed 157 • Executed April 7th, 2023

OpenZeppelin
0xec40...ec1e

For	462,555	Against	0
28 Addresses	Votes	0 Addresses	Votes

Details

1 Transfer 23617.05 COMP to 0x57C970568668087c05352456a3F59B58B0330066

Background

Starting on Dec 21st, 2021, OpenZeppelin was selected to offer the Compound DAO security services including continuous audit, security advisory, and monitoring. At the start of every quarter, OpenZeppelin creates a proposal to perform the next service fee payment.

Compensation Structure

We receive our quarterly payments in a lump-sum of COMP. Based on the last week's average price, this would be $42.34 per COMP for a total quarterly payment of 23,617 COMP equaling $1M per the original agreement. This COMP will be transferred from the Timelock's existing balance. More detail in this forum post.

By approving this proposal, you agree that any services provided by OpenZeppelin shall be governed by the Terms of Service available here

Proposal History

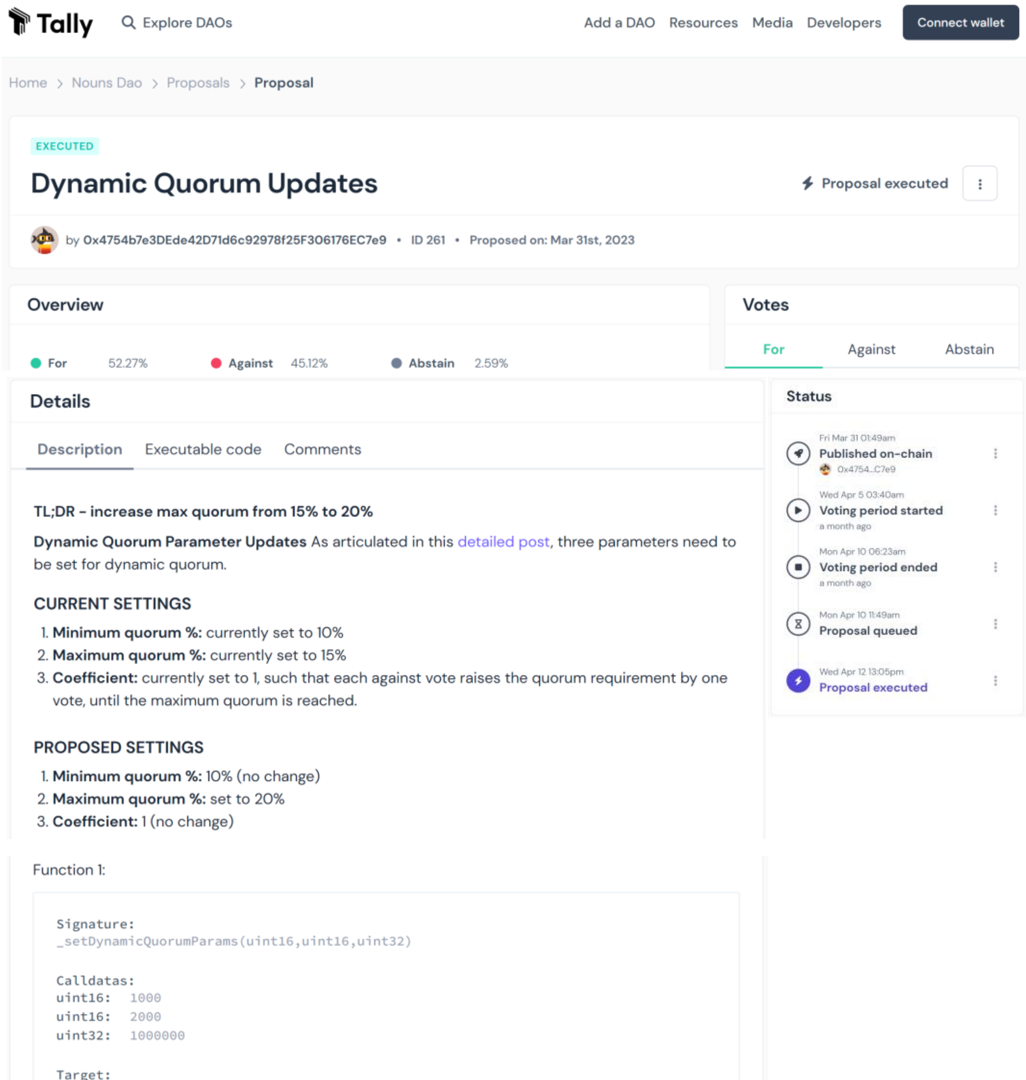
- ✓ Created — April 1st, 2023 – 1:53am
- ✓ Active — April 2nd, 2023 – 10:13pm
- ✓ Succeeded — April 5th, 2023 – 5:16pm
- ✓ Queued — April 5th, 2023 – 5:17pm
- ✓ Executed — April 7th, 2023 – 5:20pm

https://compound.finance/governance/proposals/157

On-chain Governance Example

Nouns DAO

Governance parameter update



Tally · Explore DAOs · Add a DAO · Resources · Media · Developers · Connect wallet

Home › Nouns Dao › Proposals › Proposal

EXECUTED

Dynamic Quorum Updates

Proposal executed

by 0x4754b7e3DEde42D71d6c92978f25F306176EC7e9 • ID 261 • Proposed on: Mar 31st, 2023

Overview

● For 52.27% ● Against 45.12% ● Abstain 2.59%

Votes

For · Against · Abstain

Details

Description · Executable code · Comments

TL;DR - increase max quorum from 15% to 20%

Dynamic Quorum Parameter Updates As articulated in this detailed post, three parameters need to be set for dynamic quorum.

CURRENT SETTINGS

1. **Minimum quorum %:** currently set to 10%
2. **Maximum quorum %:** currently set to 15%
3. **Coefficient:** currently set to 1, such that each against vote raises the quorum requirement by one vote, until the maximum quorum is reached.

PROPOSED SETTINGS

1. **Minimum quorum %:** 10% (no change)
2. **Maximum quorum %:** set to 20%
3. **Coefficient:** 1 (no change)

Function 1:

```
Signature:
_setDynamicQuorumParams(uint16,uint16,uint32)

Calldatas:
uint16:  1000
uint16:  2000
uint32:  1000000

Target:
```

Status

Fri Mar 31 01:49am
Published on-chain
0x4754...C7e9

Wed Apr 5 03:40am
Voting period started
a month ago

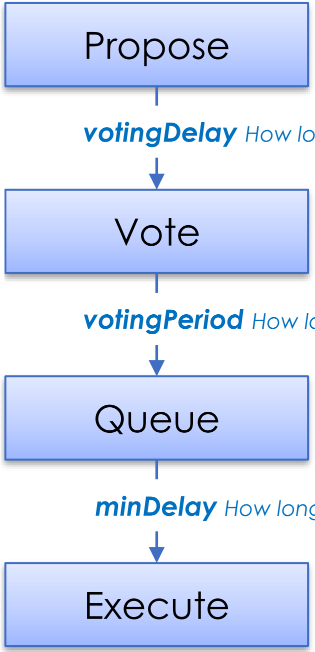
Mon Apr 10 06:23am
Voting period ended
a month ago

Mon Apr 10 11:49am
Proposal queued

Wed Apr 12 13:05pm
Proposal executed

https://www.tally.xyz/gov/nounsdao/proposal/261

On-chain governance lifecycle



Optional parameter
proposalThreshold

Propose

Create a proposal which is ***a sequence of actions***
(specifying functions with arguments in a smart contract)

votingDelay *How long after a proposal is created should voting power be fixed*

quorum
a percentage of
the total supply
at the block a proposal's
voting power is retrieved

Vote

Cast votes with tokens (ERC20, ERC721, ERC1155) by delegates
Two options 1) a holder can become a delegate themselves
2) a holder can set a trusted representative as their delegate

votingPeriod *How long does a proposal remain open to votes*

Queue

If quorum was reached (enough voting power participated) and the majority voted in favor the proposal is considered successful and can proceed to be executed

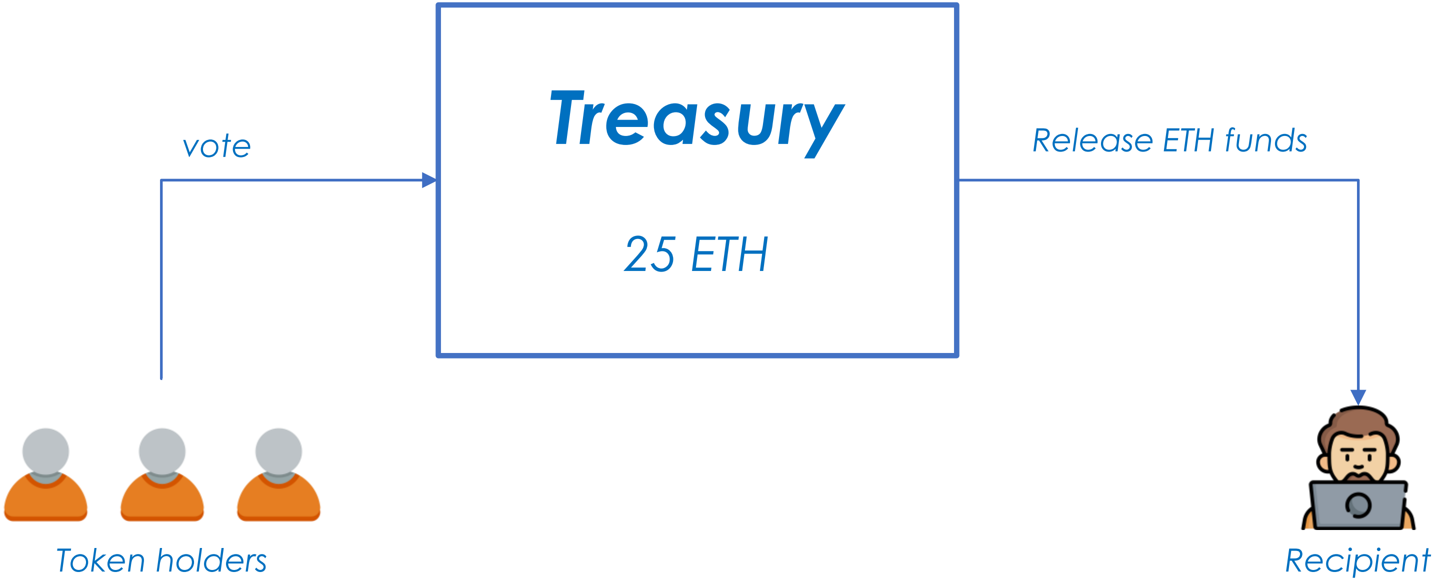
minDelay *How long should a proposal be delayed to execute after it passes*

Execute

Execute the encoded function call with arguments to the target address

On-chain governance example

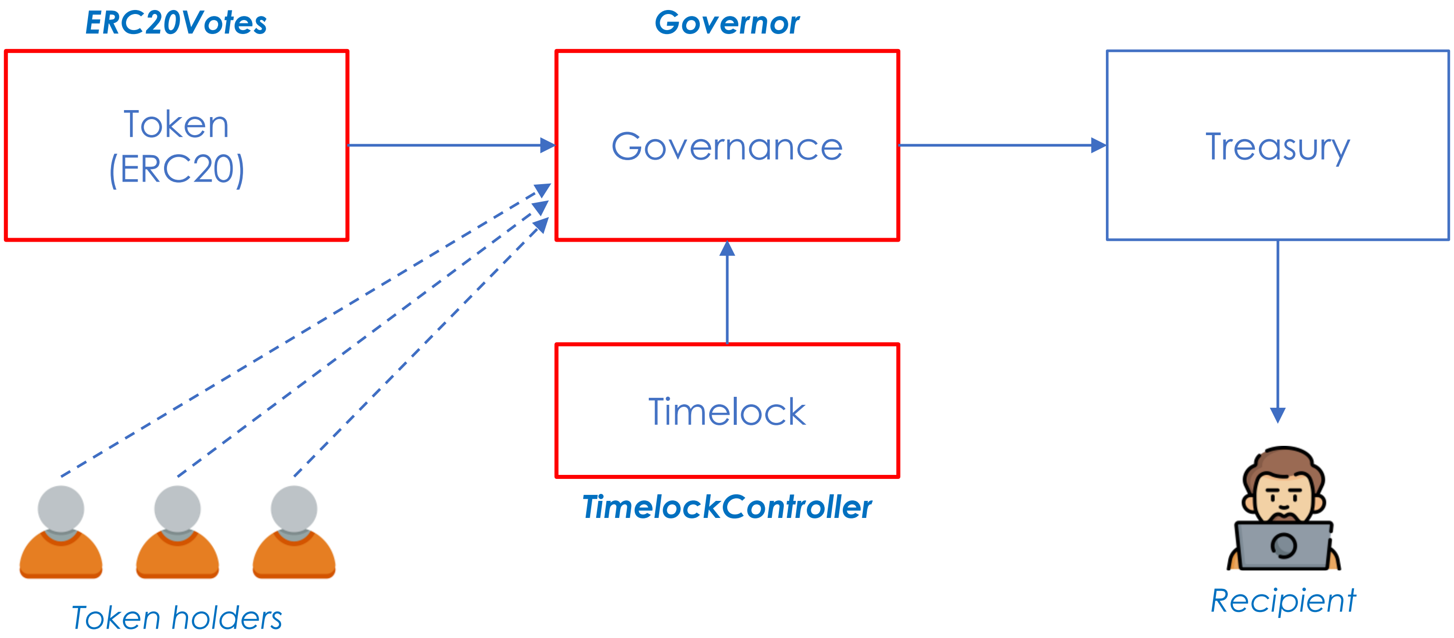
Proposal: the ETH funds of treasury will be released to the recipient



https://github.com/dappuniversity/governance

On-chain governance architecture

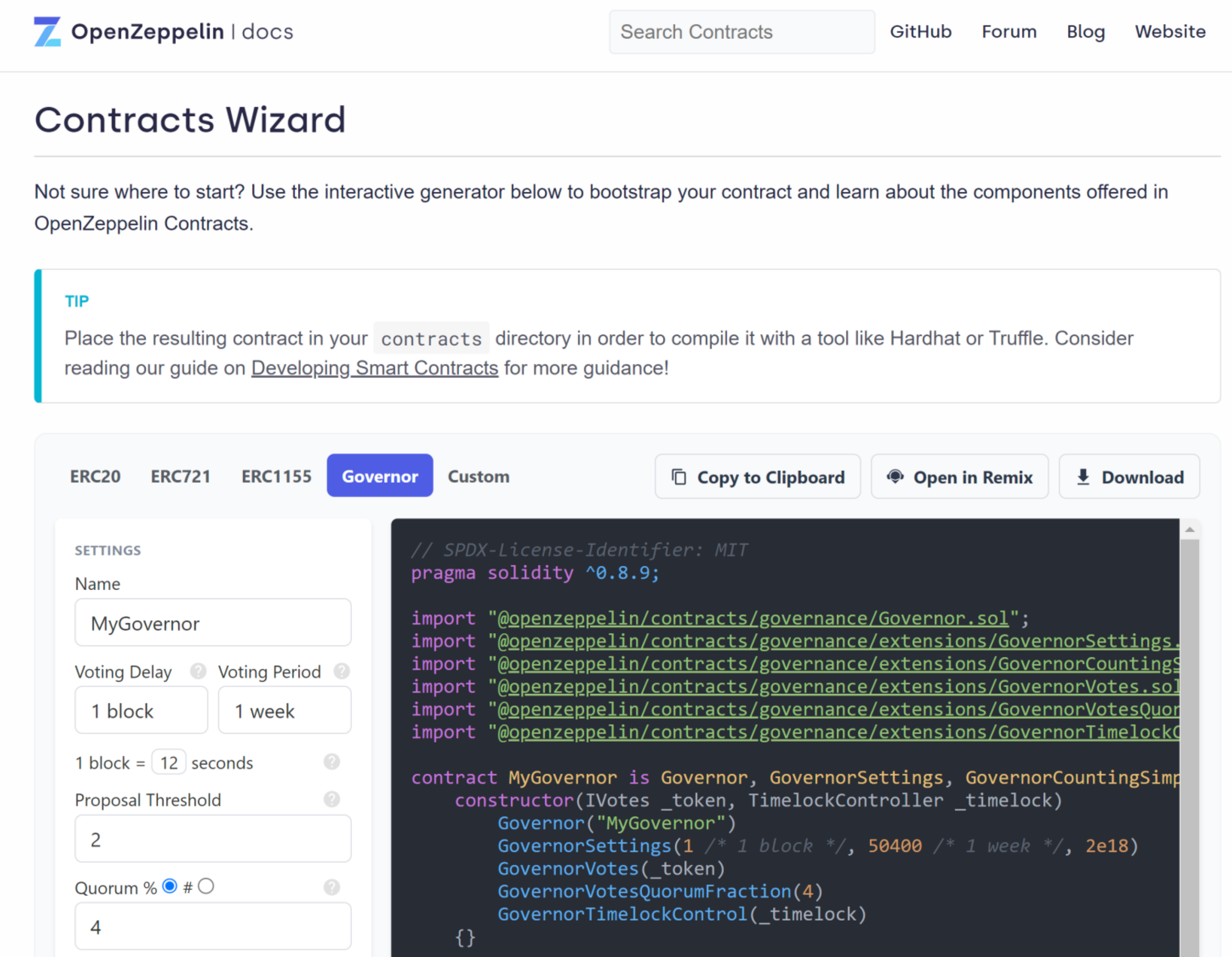
You have to implement 4 contracts:
3 of them just inherit OpenZeppelin governance contracts (red boxes)



https://github.com/dappuniversity/governance

OpenZeppelin Wizard

The easiest way to get template codes for tokens and governance



Contracts Wizard

Not sure where to start? Use the interactive generator below to bootstrap your contract and learn about the components offered in OpenZeppelin Contracts.

TIP

Place the resulting contract in your `contracts` directory in order to compile it with a tool like Hardhat or Truffle. Consider reading our guide on Developing Smart Contracts for more guidance!

SETTINGS

Name: MyGovernor

Voting Delay: 1 block

Voting Period: 1 week

1 block = 12 seconds

Proposal Threshold: 2

Quorum % ● # ○

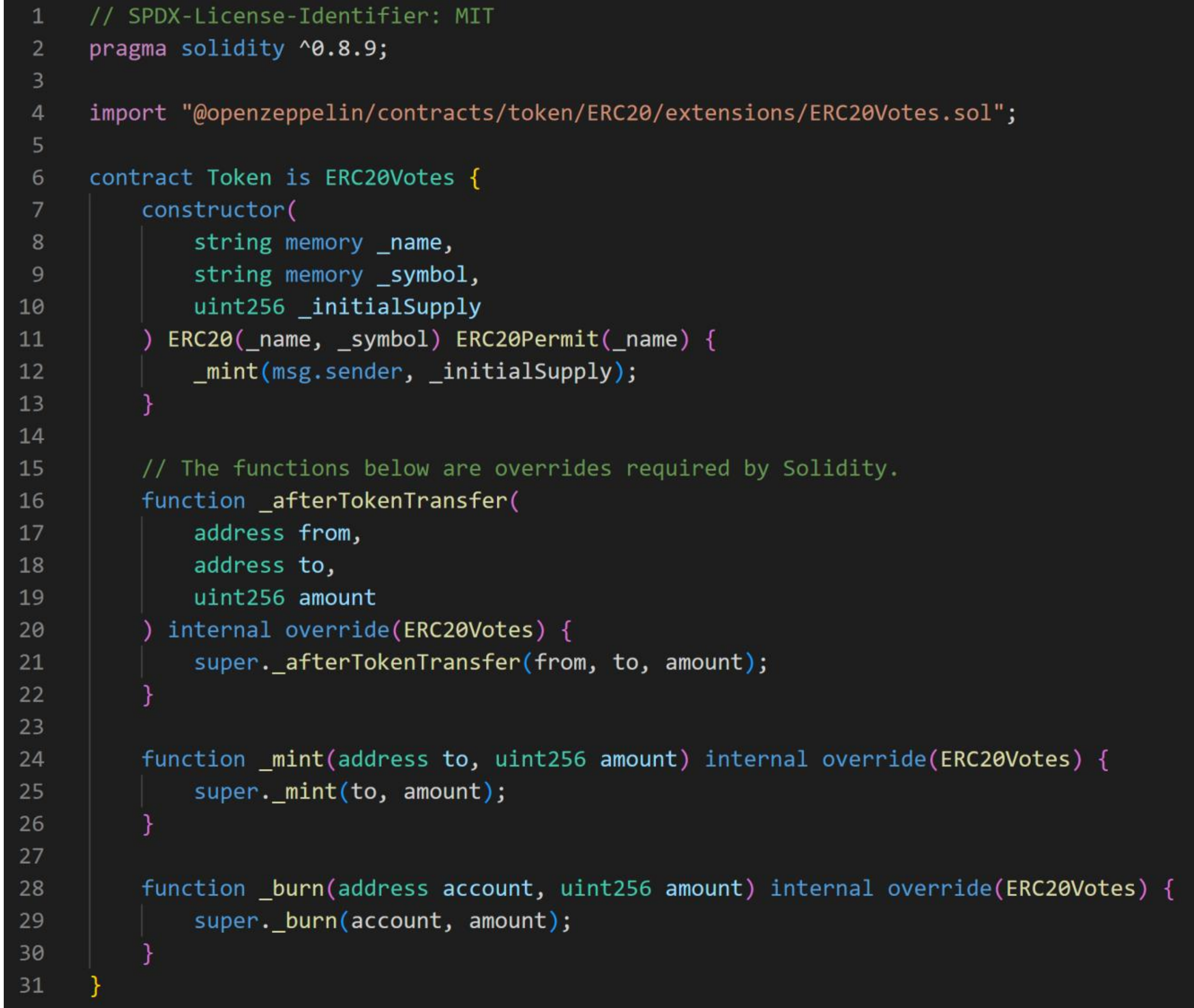
4

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

import "@openzeppelin/contracts/governance/Governor.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorSettings.
import "@openzeppelin/contracts/governance/extensions/GovernorCountingS
import "@openzeppelin/contracts/governance/extensions/GovernorVotes.sol
import "@openzeppelin/contracts/governance/extensions/GovernorVotesQuor
import "@openzeppelin/contracts/governance/extensions/GovernorTimelockC

contract MyGovernor is Governor, GovernorSettings, GovernorCountingSimp
    constructor(IVotes _token, TimelockController _timelock)
        Governor("MyGovernor")
        GovernorSettings(1 /* 1 block */, 50400 /* 1 week */, 2e18)
        GovernorVotes(_token)
        GovernorVotesQuorumFraction(4)
        GovernorTimelockControl(_timelock)
    {}
```

https://docs.openzeppelin.com/contracts/4.x/wizard

Governance Token inherited by ERC20Votes

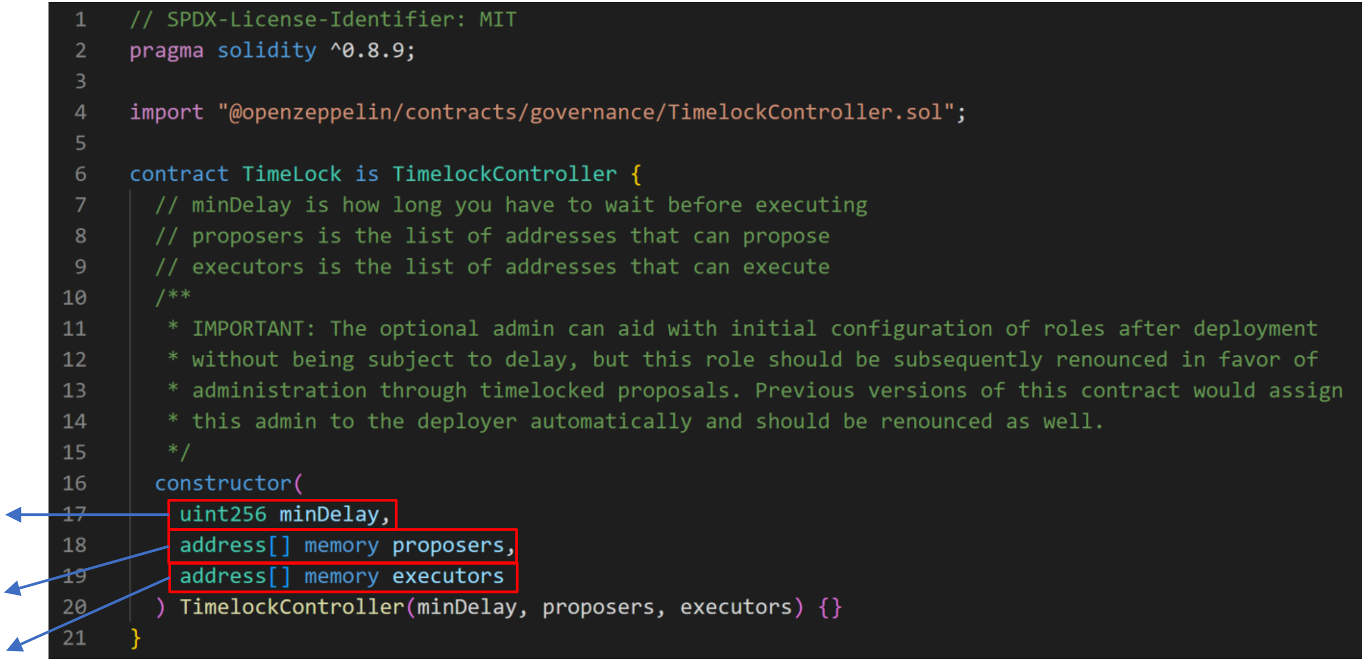
*The voting power of each account is **retrieved from past snapshots** rather than current balance, which is an important protection that **prevents double voting***



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

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

contract Token is ERC20Votes {
    constructor(
        string memory _name,
        string memory _symbol,
        uint256 _initialSupply
    ) ERC20(_name, _symbol) ERC20Permit(_name) {
        _mint(msg.sender, _initialSupply);
    }

    // The functions below are overrides required by Solidity.
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal override(ERC20Votes) {
        super._afterTokenTransfer(from, to, amount);
    }

    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);
    }
}
```

Timelock inherited by TimelockController

Defining the role of proposers and executors and setting up minDelay



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

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

contract TimeLock is TimelockController {
    // minDelay is how long you have to wait before executing
    // proposers is the list of addresses that can propose
    // executors is the list of addresses that can execute
    /**
     * IMPORTANT: The optional admin can aid with initial configuration of roles after deployment
     * without being subject to delay, but this role should be subsequently renounced in favor of
     * administration through timelocked proposals. Previous versions of this contract would assign
     * this admin to the deployer automatically and should be renounced as well.
     */
    constructor(
        uint256 minDelay,
        address[] memory proposers,
        address[] memory executors
    ) TimelockController(minDelay, proposers, executors) {}
}
```

How long should a proposal be delayed to be executed after it passes

Who can propose?

Who can execute a proposal?

Governance inherited by Governor

what options people have when casting a vote and how those votes are counted

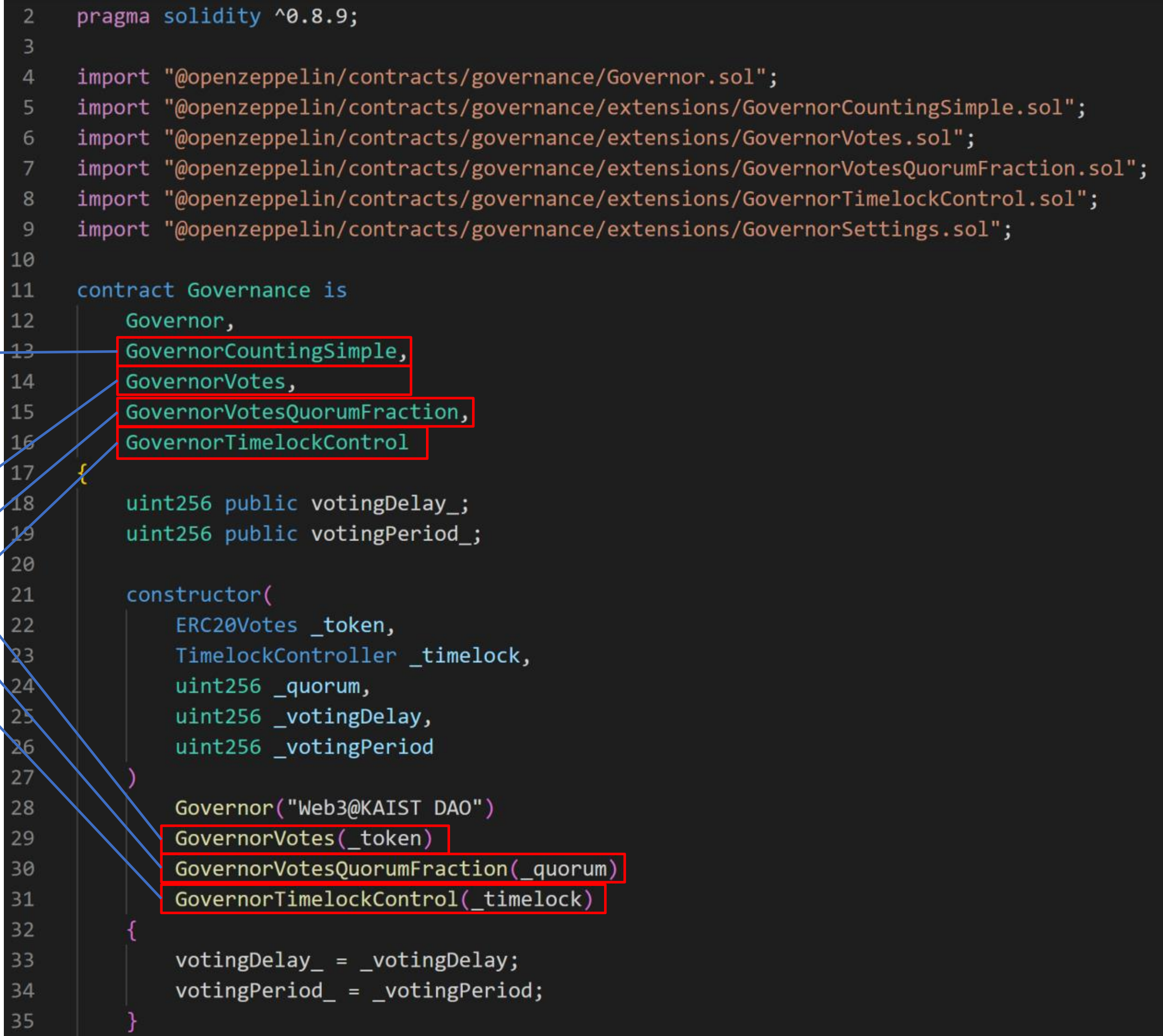
GovernorCountingSimple

For, Against, and Abstain only For and Abstain votes are counted towards quorum

how voting power is determined

how many votes are needed for quorum

set up roles and a timelock for execution

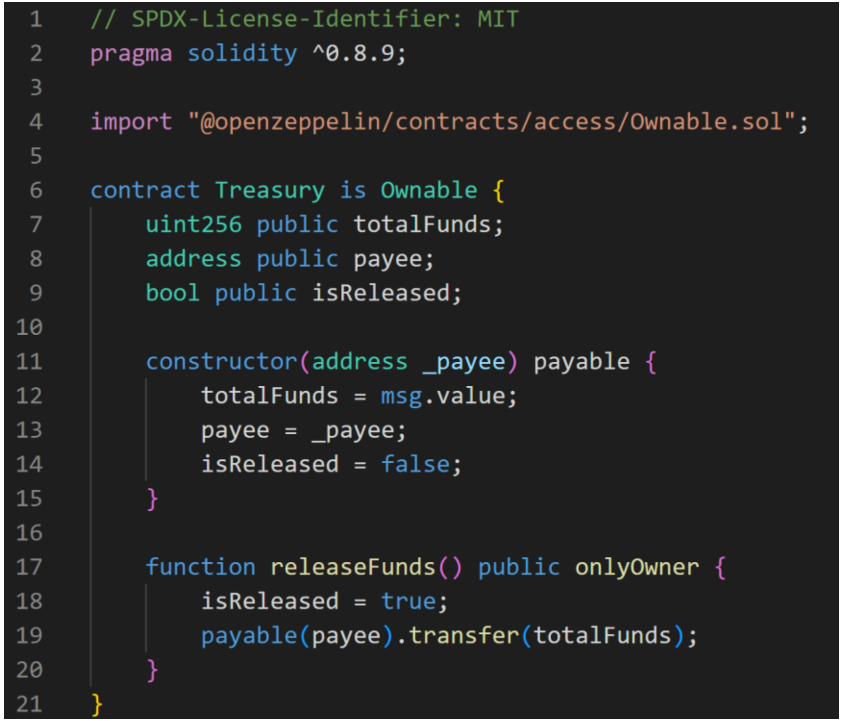


```
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/governance/Governor.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorCountingSimple.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorVotes.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorVotesQuorumFraction.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorTimelockControl.sol";
import "@openzeppelin/contracts/governance/extensions/GovernorSettings.sol";

contract Governance is
    Governor,
    GovernorCountingSimple,
    GovernorVotes,
    GovernorVotesQuorumFraction,
    GovernorTimelockControl
{
    uint256 public votingDelay_;
    uint256 public votingPeriod_;

    constructor(
        ERC20Votes _token,
        TimelockController _timelock,
        uint256 _quorum,
        uint256 _votingDelay,
        uint256 _votingPeriod
    )
        Governor("Web3@KAIST DAO")
        GovernorVotes(_token)
        GovernorVotesQuorumFraction(_quorum)
        GovernorTimelockControl(_timelock)
    {
        votingDelay_ = _votingDelay;
        votingPeriod_ = _votingPeriod;
    }
```

Treasury contract

ETH fund treasury

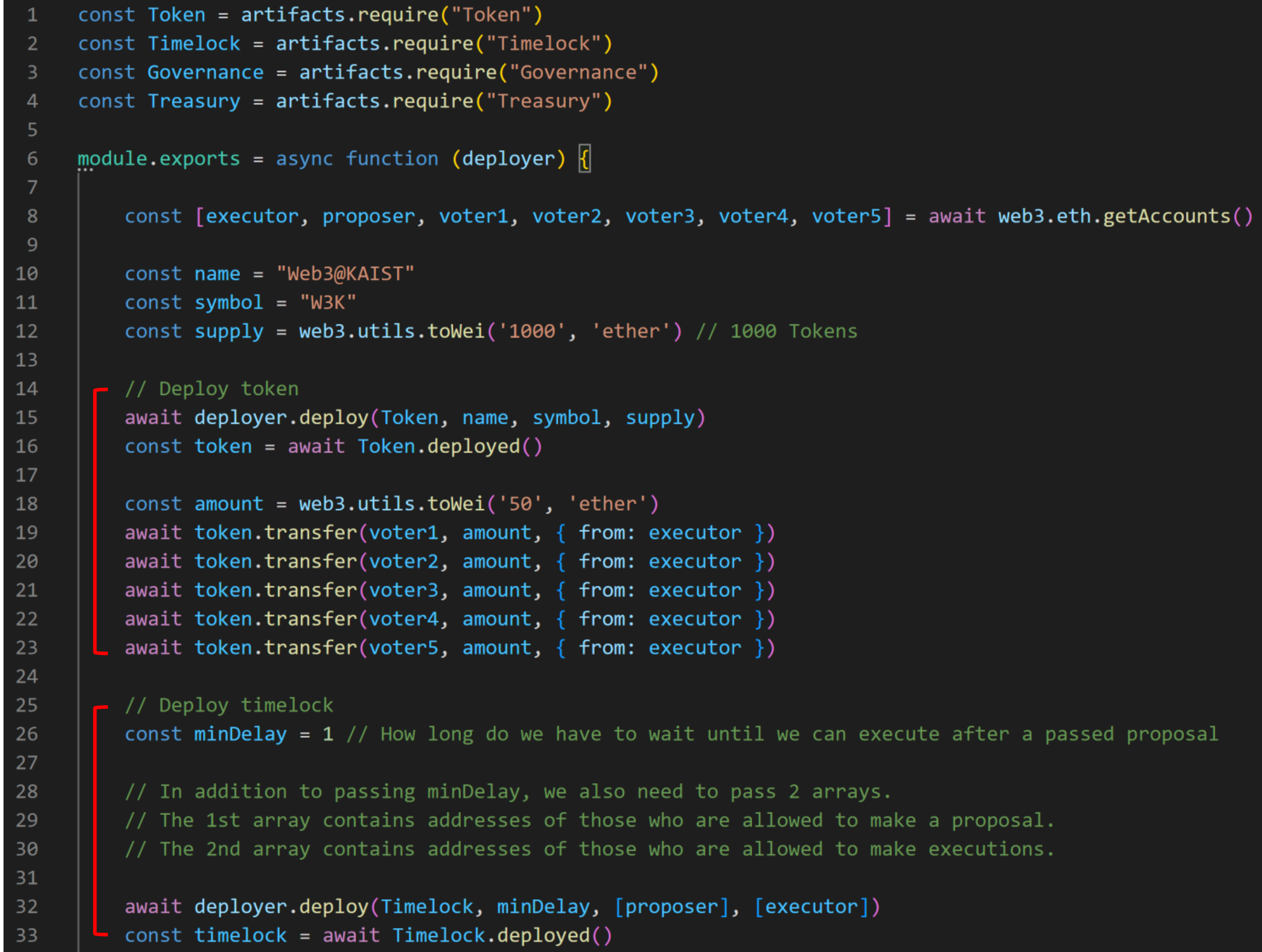
- *The fund will be set when the contract is created*
- *Only the owner of the contract can release the fund*



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

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

contract Treasury is Ownable {
    uint256 public totalFunds;
    address public payee;
    bool public isReleased;

    constructor(address _payee) payable {
        totalFunds = msg.value;
        payee = _payee;
        isReleased = false;
    }

    function releaseFunds() public onlyOwner {
        isReleased = true;
        payable(payee).transfer(totalFunds);
    }
}
```

Deployment Script (1/2)

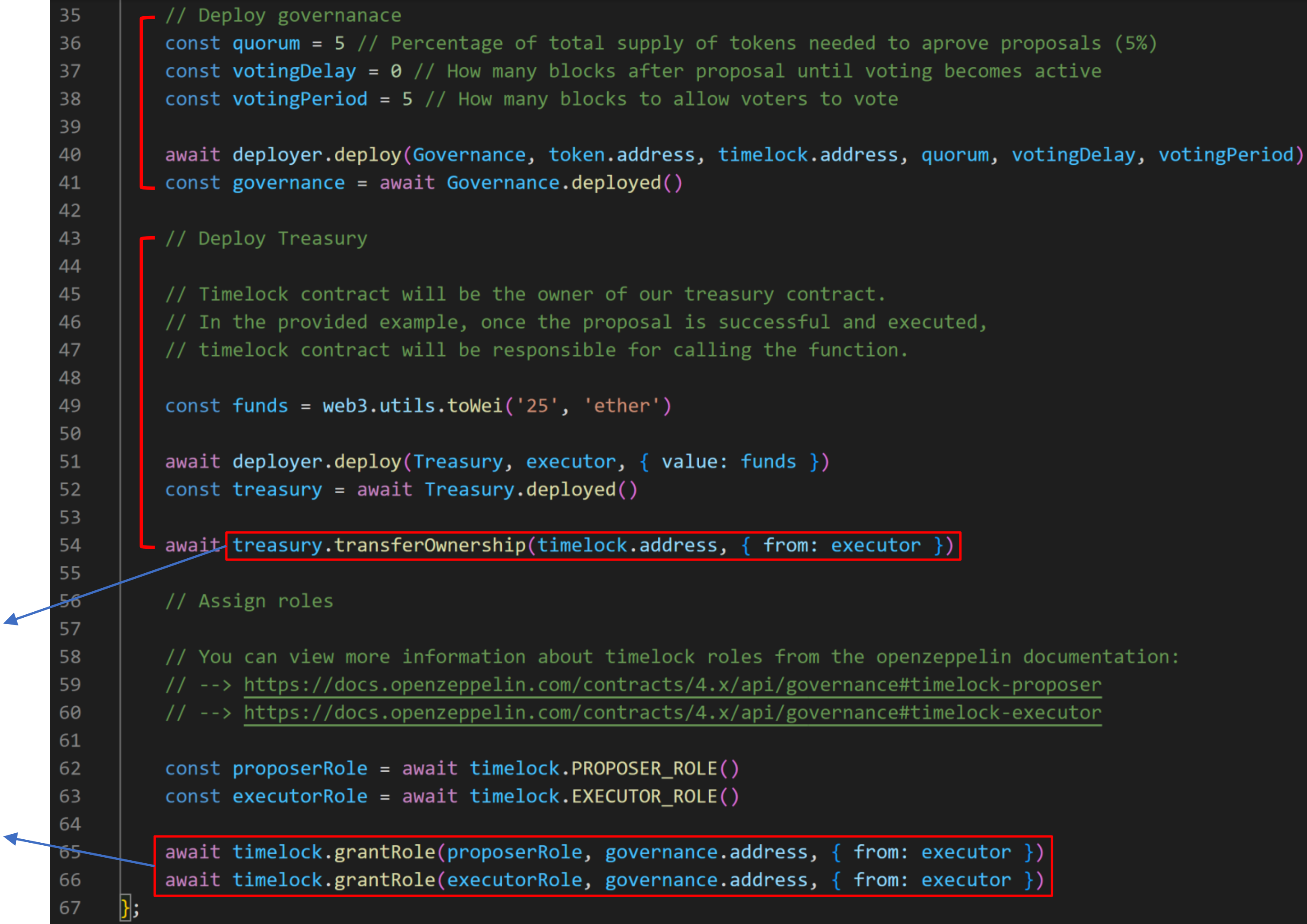
2_deploy_contracts.js



```
const Token = artifacts.require("Token")
const Timelock = artifacts.require("Timelock")
const Governance = artifacts.require("Governance")
const Treasury = artifacts.require("Treasury")

module.exports = async function (deployer) {

    const [executor, proposer, voter1, voter2, voter3, voter4, voter5] = await web3.eth.getAccounts()

    const name = "Web3@KAIST"
    const symbol = "W3K"
    const supply = web3.utils.toWei('1000', 'ether') // 1000 Tokens

    // Deploy token
    await deployer.deploy(Token, name, symbol, supply)
    const token = await Token.deployed()

    const amount = web3.utils.toWei('50', 'ether')
    await token.transfer(voter1, amount, { from: executor })
    await token.transfer(voter2, amount, { from: executor })
    await token.transfer(voter3, amount, { from: executor })
    await token.transfer(voter4, amount, { from: executor })
    await token.transfer(voter5, amount, { from: executor })

    // Deploy timelock
    const minDelay = 1 // How long do we have to wait until we can execute after a passed proposal

    // In addition to passing minDelay, we also need to pass 2 arrays.
    // The 1st array contains addresses of those who are allowed to make a proposal.
    // The 2nd array contains addresses of those who are allowed to make executions.

    await deployer.deploy(Timelock, minDelay, [proposer], [executor])
    const timelock = await Timelock.deployed()
```

Deployment Script (2/2)

2_deploy_contracts.js



```
    // Deploy governanace
    const quorum = 5 // Percentage of total supply of tokens needed to aprove proposals (5%)
    const votingDelay = 0 // How many blocks after proposal until voting becomes active
    const votingPeriod = 5 // How many blocks to allow voters to vote

    await deployer.deploy(Governance, token.address, timelock.address, quorum, votingDelay, votingPeriod)
    const governance = await Governance.deployed()

    // Deploy Treasury

    // Timelock contract will be the owner of our treasury contract.
    // In the provided example, once the proposal is successful and executed,
    // timelock contract will be responsible for calling the function.

    const funds = web3.utils.toWei('25', 'ether')

    await deployer.deploy(Treasury, executor, { value: funds })
    const treasury = await Treasury.deployed()

    await treasury.transferOwnership(timelock.address, { from: executor })

    // Assign roles

    // You can view more information about timelock roles from the openzeppelin documentation:
    // --> https://docs.openzeppelin.com/contracts/4.x/api/governance#timelock-proposer
    // --> https://docs.openzeppelin.com/contracts/4.x/api/governance#timelock-executor

    const proposerRole = await timelock.PROPOSER_ROLE()
    const executorRole = await timelock.EXECUTOR_ROLE()

    await timelock.grantRole(proposerRole, governance.address, { from: executor })
    await timelock.grantRole(executorRole, governance.address, { from: executor })
};
```

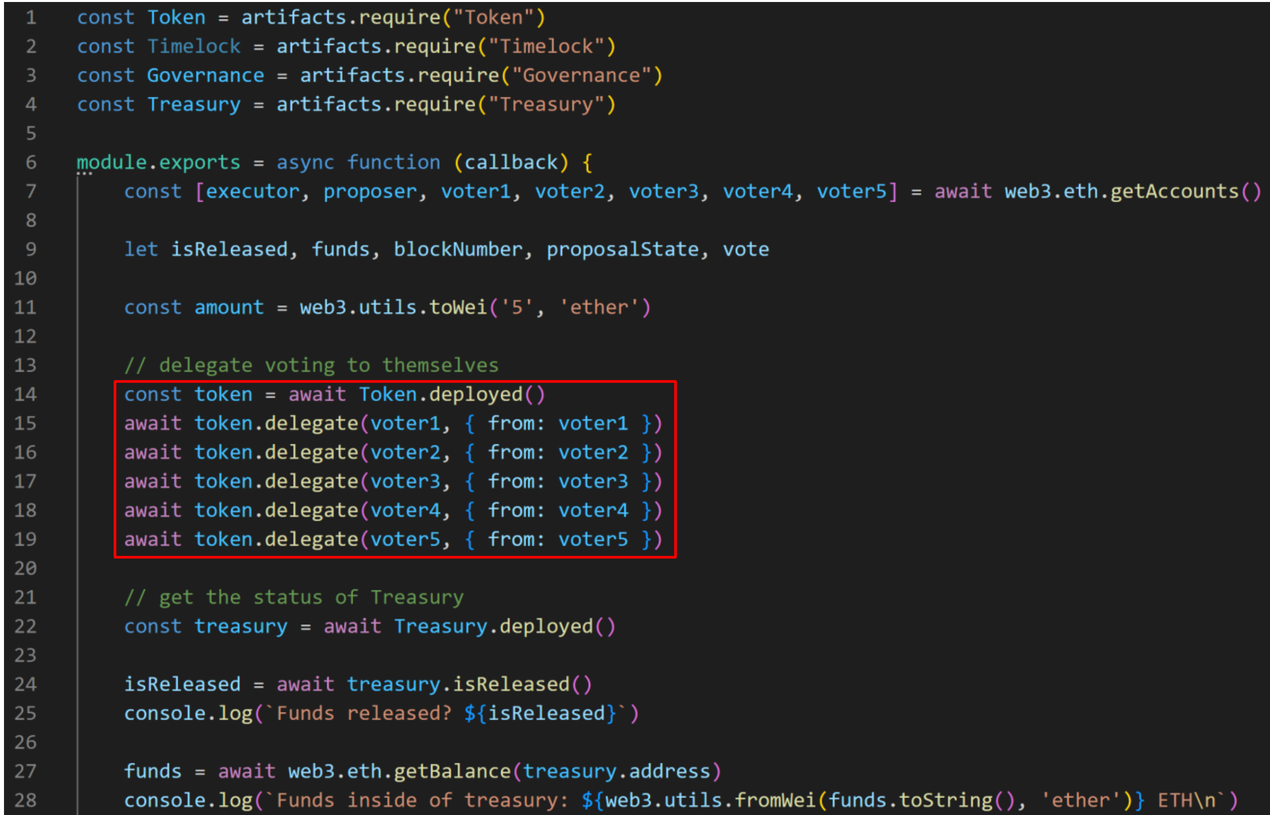
The ownership of contract should be transferred to the timelock contract in order to be executed by the timelock after a proposal passed

Grant a proposer role and an executor role to the governance contract

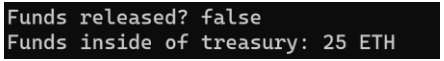
Execution Script (1/4)

scripts/1_create_proposal.js

Prepare voters as delegators



```
const Token = artifacts.require("Token")
const Timelock = artifacts.require("Timelock")
const Governance = artifacts.require("Governance")
const Treasury = artifacts.require("Treasury")

module.exports = async function (callback) {
    const [executor, proposer, voter1, voter2, voter3, voter4, voter5] = await web3.eth.getAccounts()

    let isReleased, funds, blockNumber, proposalState, vote

    const amount = web3.utils.toWei('5', 'ether')

    // delegate voting to themselves
    const token = await Token.deployed()
    await token.delegate(voter1, { from: voter1 })
    await token.delegate(voter2, { from: voter2 })
    await token.delegate(voter3, { from: voter3 })
    await token.delegate(voter4, { from: voter4 })
    await token.delegate(voter5, { from: voter5 })

    // get the status of Treasury
    const treasury = await Treasury.deployed()

    isReleased = await treasury.isReleased()
    console.log(`Funds released? ${isReleased}`)

    funds = await web3.eth.getBalance(treasury.address)
    console.log(`Funds inside of treasury: ${web3.utils.fromWei(funds.toString(), 'ether')} ETH\n`)
```

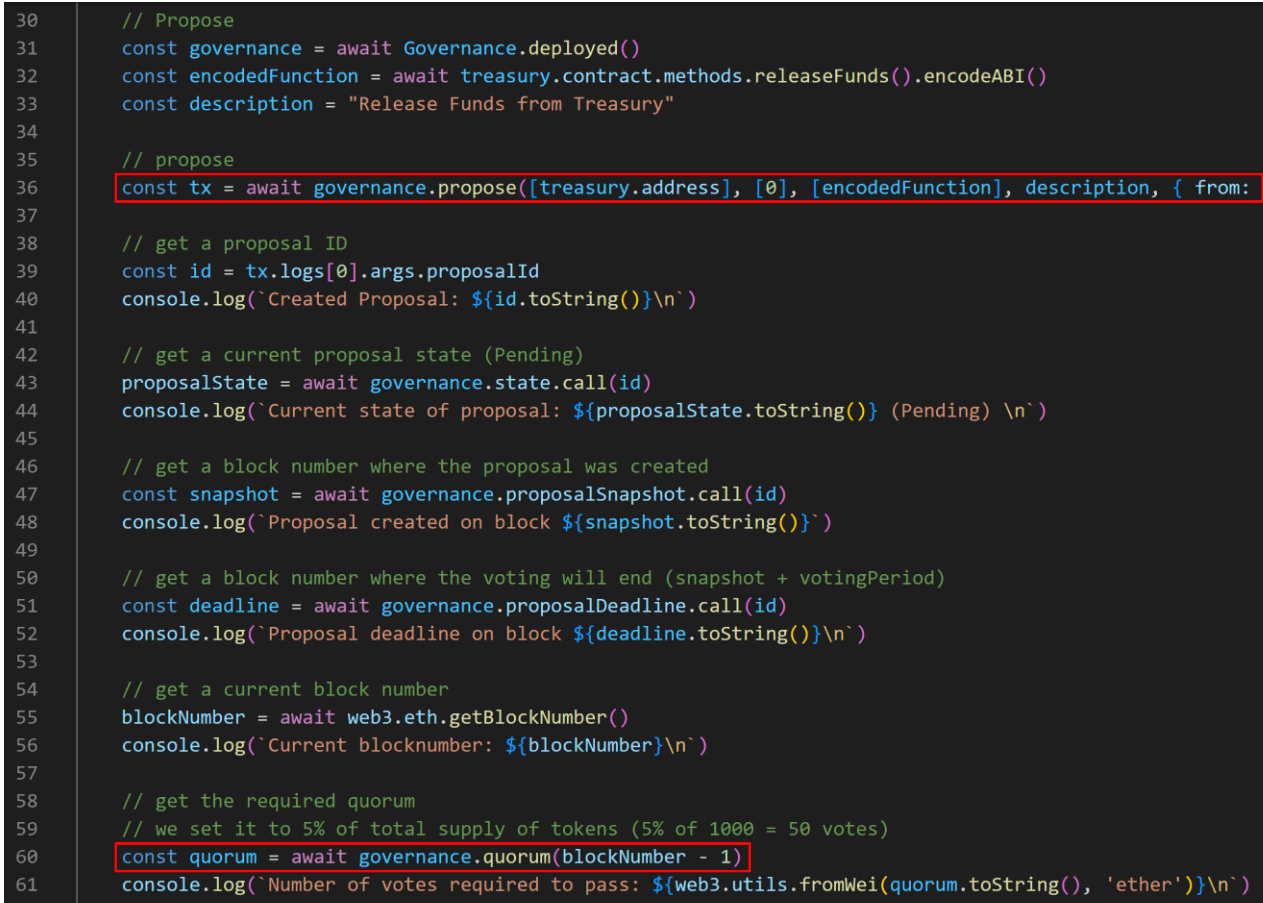


```
Funds released? false
Funds inside of treasury: 25 ETH
```

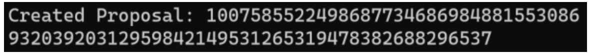
Execution Script (2/4)

scripts/1_create_proposal.js

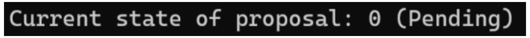
Propose phase



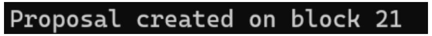
```
// Propose
const governance = await Governance.deployed()
const encodedFunction = await treasury.contract.methods.releaseFunds().encodeABI()
const description = "Release Funds from Treasury"

// propose
const tx = await governance.propose([treasury.address], [0], [encodedFunction], description, { from:

// get a proposal ID
const id = tx.logs[0].args.proposalId
console.log(`Created Proposal: ${id.toString()}\n`)

// get a current proposal state (Pending)
proposalState = await governance.state.call(id)
console.log(`Current state of proposal: ${proposalState.toString()} (Pending) \n`)

// get a block number where the proposal was created
const snapshot = await governance.proposalSnapshot.call(id)
console.log(`Proposal created on block ${snapshot.toString()}`)

// get a block number where the voting will end (snapshot + votingPeriod)
const deadline = await governance.proposalDeadline.call(id)
console.log(`Proposal deadline on block ${deadline.toString()}\n`)

// get a current block number
blockNumber = await web3.eth.getBlockNumber()
console.log(`Current blocknumber: ${blockNumber}\n`)

// get the required quorum
// we set it to 5% of total supply of tokens (5% of 1000 = 50 votes)
const quorum = await governance.quorum(blockNumber - 1)
console.log(`Number of votes required to pass: ${web3.utils.fromWei(quorum.toString(), 'ether')}\n`)
```



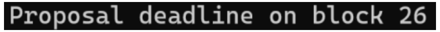
```
Created Proposal: 1007585522498687734686984881553086
93203920312959842149531265319478382688296537
```



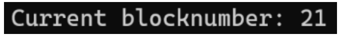
```
Current state of proposal: 0 (Pending)
```



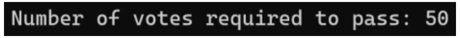
```
Proposal created on block 21
```



```
Proposal deadline on block 26
```



```
Current blocknumber: 21
```

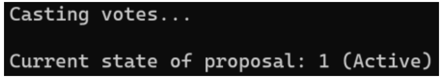


```
Number of votes required to pass: 50
```

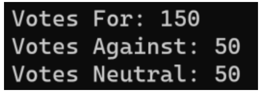
Execution Script (3/4)

scripts/1_create_proposal.js

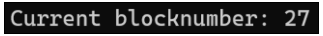
Vote phase



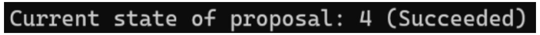
```
Casting votes...

Current state of proposal: 1 (Active)
```



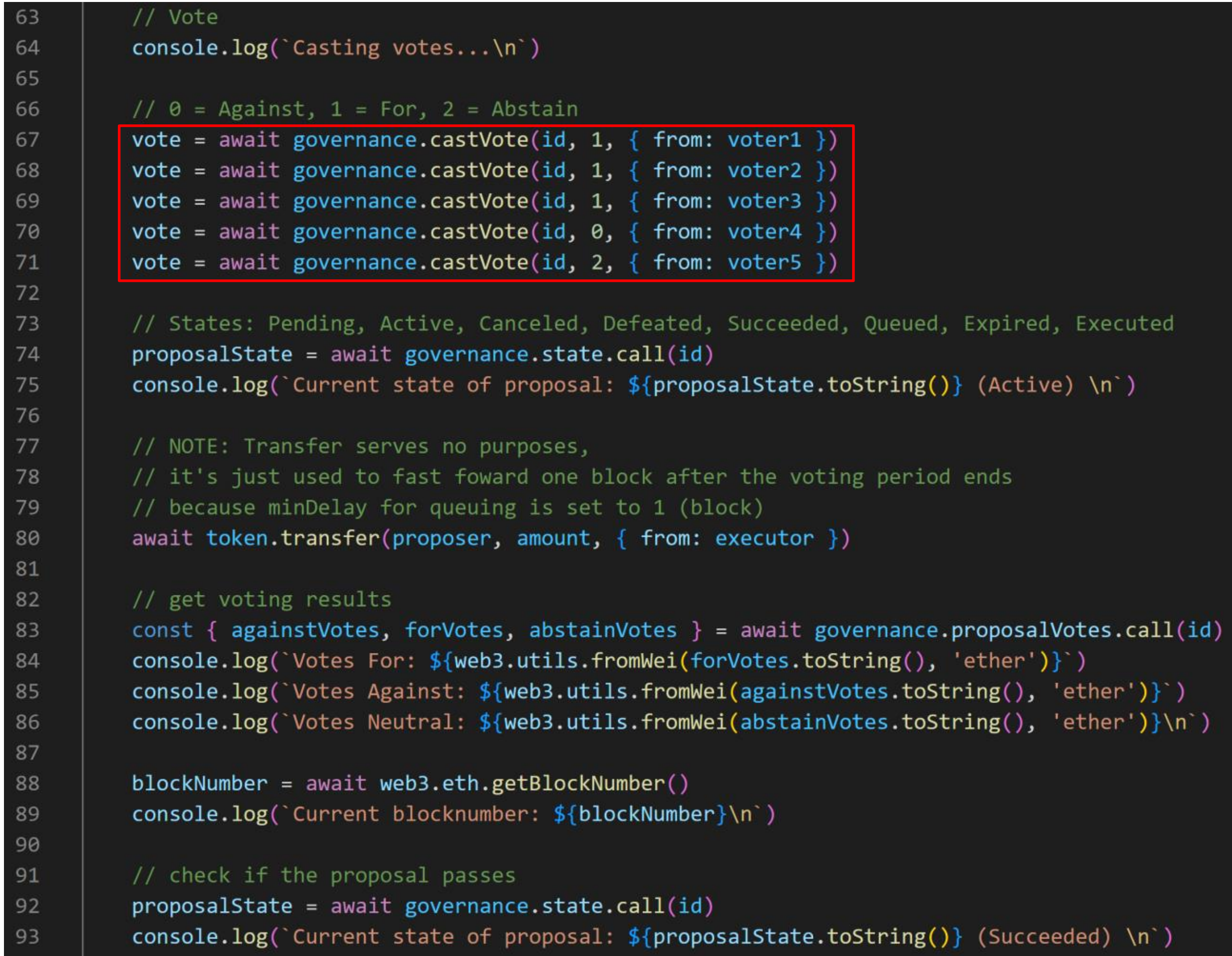
```
Votes For: 150
Votes Against: 50
Votes Neutral: 50
```



```
Current blocknumber: 27
```



```
Current state of proposal: 4 (Succeeded)
```

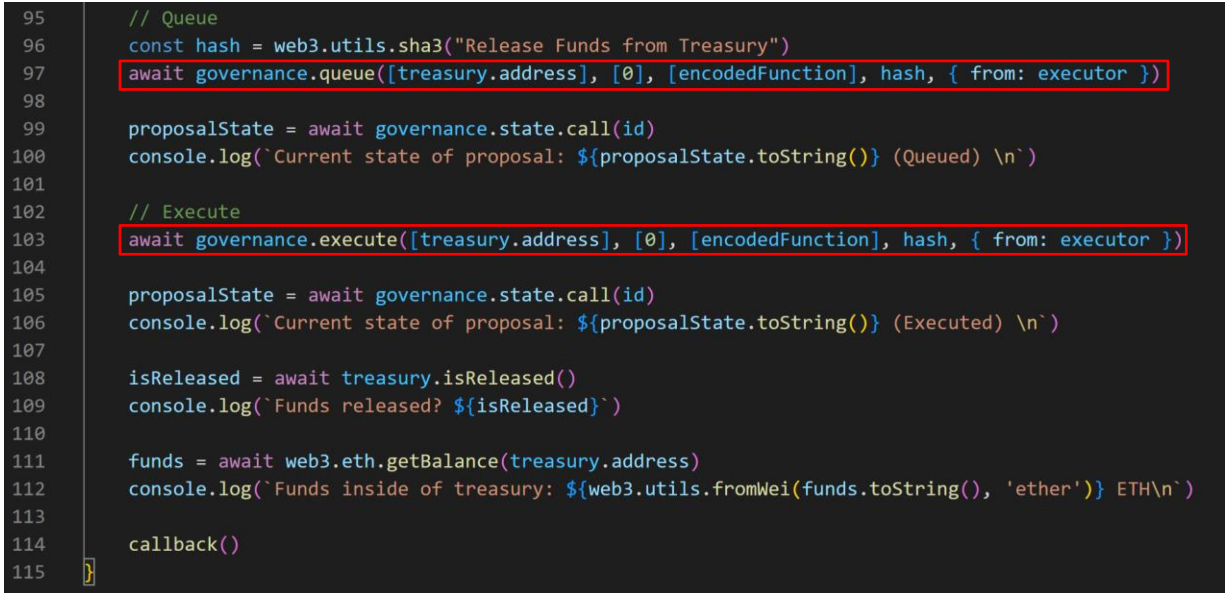


```
    // Vote
    console.log(`Casting votes...\n`)

    // 0 = Against, 1 = For, 2 = Abstain
    vote = await governance.castVote(id, 1, { from: voter1 })
    vote = await governance.castVote(id, 1, { from: voter2 })
    vote = await governance.castVote(id, 1, { from: voter3 })
    vote = await governance.castVote(id, 0, { from: voter4 })
    vote = await governance.castVote(id, 2, { from: voter5 })

    // States: Pending, Active, Canceled, Defeated, Succeeded, Queued, Expired, Executed
    proposalState = await governance.state.call(id)
    console.log(`Current state of proposal: ${proposalState.toString()} (Active) \n`)

    // NOTE: Transfer serves no purposes,
    // it's just used to fast foward one block after the voting period ends
    // because minDelay for queuing is set to 1 (block)
    await token.transfer(proposer, amount, { from: executor })

    // get voting results
    const { againstVotes, forVotes, abstainVotes } = await governance.proposalVotes.call(id)
    console.log(`Votes For: ${web3.utils.fromWei(forVotes.toString(), 'ether')}`)
    console.log(`Votes Against: ${web3.utils.fromWei(againstVotes.toString(), 'ether')}`)
    console.log(`Votes Neutral: ${web3.utils.fromWei(abstainVotes.toString(), 'ether')}\n`)

    blockNumber = await web3.eth.getBlockNumber()
    console.log(`Current blocknumber: ${blockNumber}\n`)

    // check if the proposal passes
    proposalState = await governance.state.call(id)
    console.log(`Current state of proposal: ${proposalState.toString()} (Succeeded) \n`)
```

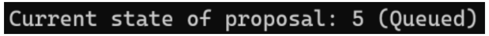
Execution Script (4/4)

scripts/1_create_proposal.js

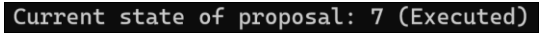
Queue & Execute phase



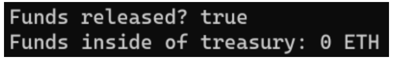
```
    // Queue
    const hash = web3.utils.sha3("Release Funds from Treasury")
    await governance.queue([treasury.address], [0], [encodedFunction], hash, { from: executor })

    proposalState = await governance.state.call(id)
    console.log(`Current state of proposal: ${proposalState.toString()} (Queued) \n`)

    // Execute
    await governance.execute([treasury.address], [0], [encodedFunction], hash, { from: executor })

    proposalState = await governance.state.call(id)
    console.log(`Current state of proposal: ${proposalState.toString()} (Executed) \n`)

    isReleased = await treasury.isReleased()
    console.log(`Funds released? ${isReleased}`)

    funds = await web3.eth.getBalance(treasury.address)
    console.log(`Funds inside of treasury: ${web3.utils.fromWei(funds.toString(), 'ether')} ETH\n`)

    callback()
}
```



```
Current state of proposal: 5 (Queued)
```



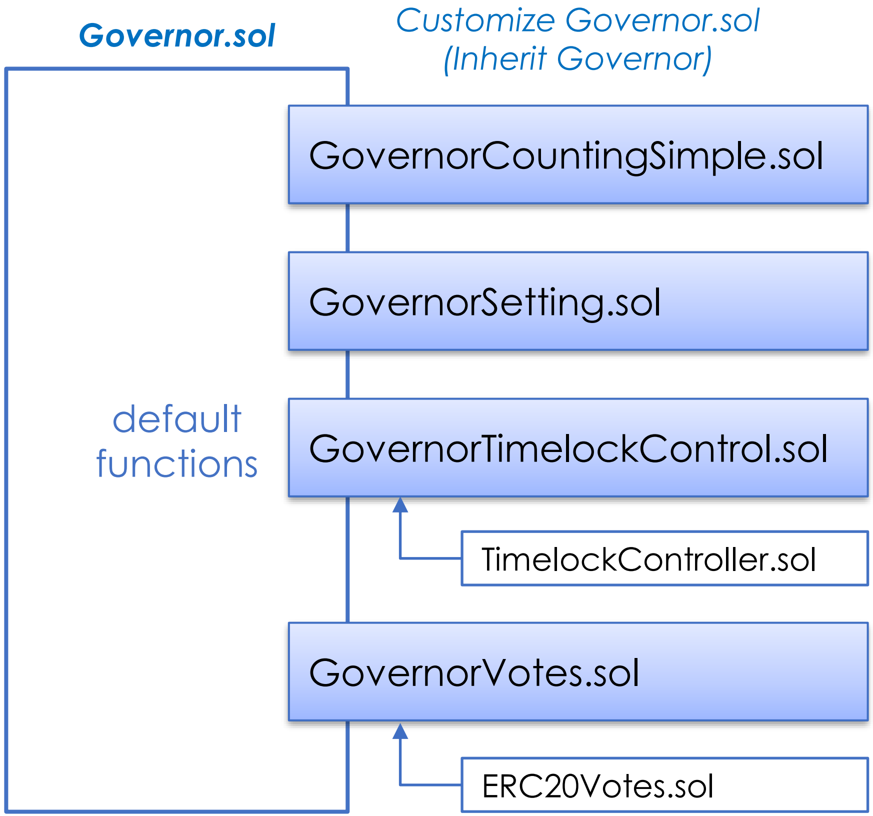
```
Current state of proposal: 7 (Executed)
```



```
Funds released? true
Funds inside of treasury: 0 ETH
```

Behind
OpenZeppelin Governance

Governor Architecture



Governor.sol

Customize Governor.sol
(Inherit Governor)

default
functions

GovernorCountingSimple.sol

GovernorSetting.sol

GovernorTimelockControl.sol

TimelockController.sol

GovernorVotes.sol

ERC20Votes.sol

Defining voting options (3 options) &
counting votes (_countVote(),
_quorumReached(), _voteSucceeded())

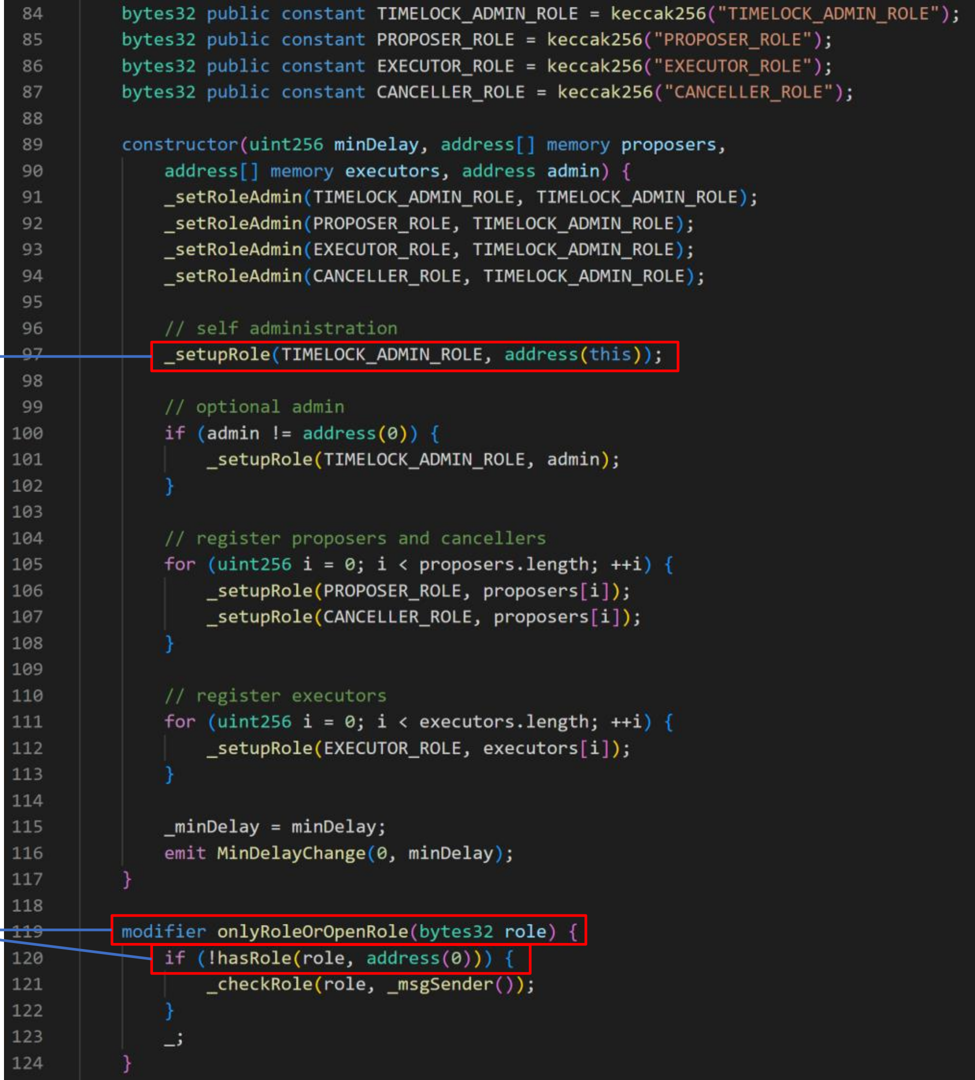
Setting parameters
votingDelay, votingPeriod, proposalThreshold

Binding the execution to TimelockController
enforcing the minDelay before the execution
Overriding Governor's functions such as
state(), _execute(), _cancel()

Wrapper contract for tokens (ERC20, ERC721)
Binding the token to ERC20Votes
Getting the voting weight (_getVotes())

Setting up the roles

TimelockController.sol



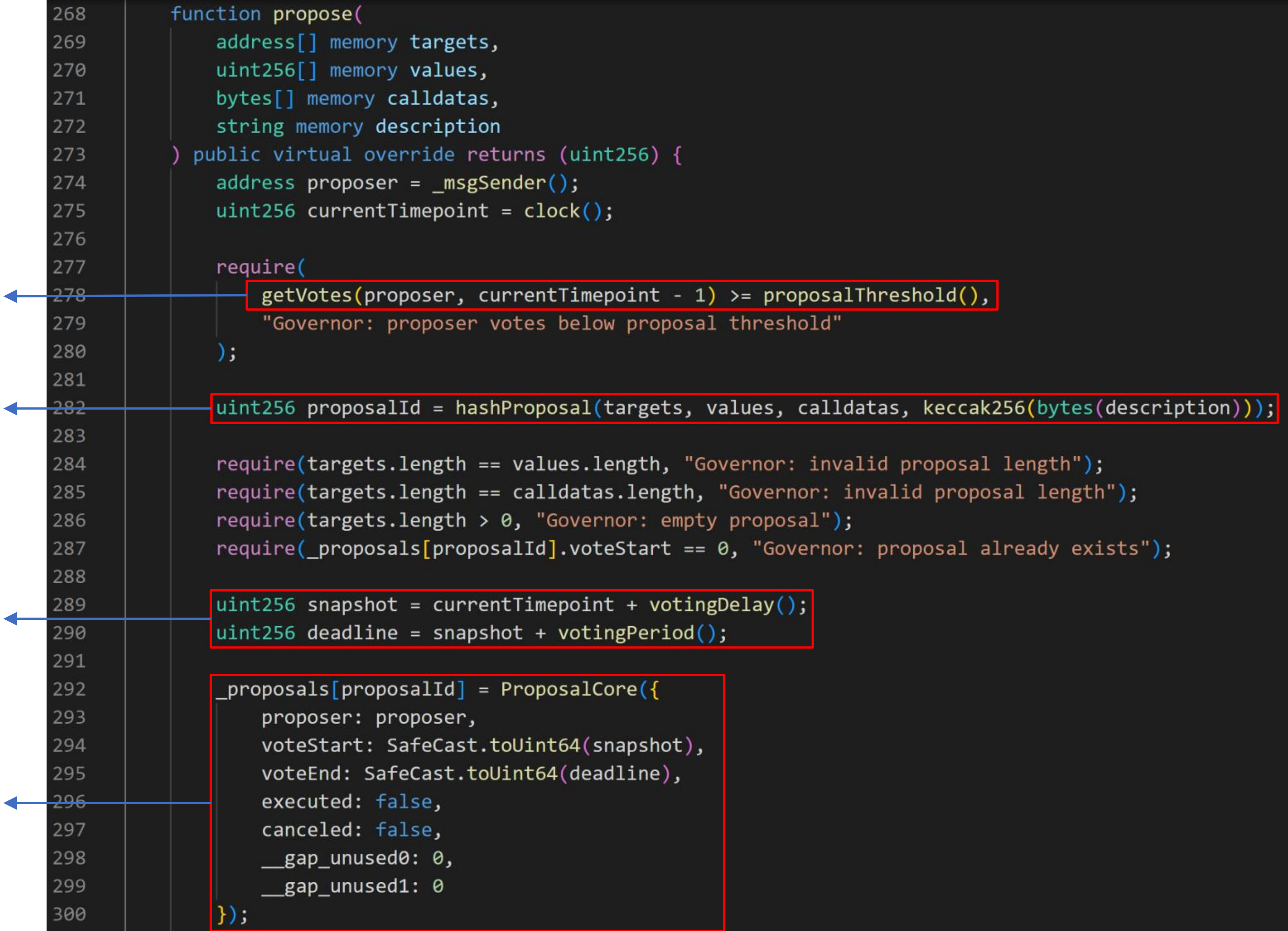
```
bytes32 public constant TIMELOCK_ADMIN_ROLE = keccak256("TIMELOCK_ADMIN_ROLE");
bytes32 public constant PROPOSER_ROLE = keccak256("PROPOSER_ROLE");
bytes32 public constant EXECUTOR_ROLE = keccak256("EXECUTOR_ROLE");
bytes32 public constant CANCELLER_ROLE = keccak256("CANCELLER_ROLE");

constructor(uint256 minDelay, address[] memory proposers,
    address[] memory executors, address admin) {
    _setRoleAdmin(TIMELOCK_ADMIN_ROLE, TIMELOCK_ADMIN_ROLE);
    _setRoleAdmin(PROPOSER_ROLE, TIMELOCK_ADMIN_ROLE);
    _setRoleAdmin(EXECUTOR_ROLE, TIMELOCK_ADMIN_ROLE);
    _setRoleAdmin(CANCELLER_ROLE, TIMELOCK_ADMIN_ROLE);

    // self administration
    _setupRole(TIMELOCK_ADMIN_ROLE, address(this));

    // optional admin
    if (admin != address(0)) {
        _setupRole(TIMELOCK_ADMIN_ROLE, admin);
    }

    // register proposers and cancellers
    for (uint256 i = 0; i < proposers.length; ++i) {
        _setupRole(PROPOSER_ROLE, proposers[i]);
        _setupRole(CANCELLER_ROLE, proposers[i]);
    }

    // register executors
    for (uint256 i = 0; i < executors.length; ++i) {
        _setupRole(EXECUTOR_ROLE, executors[i]);
    }

    _minDelay = minDelay;
    emit MinDelayChange(0, minDelay);
}

modifier onlyRoleOrOpenRole(bytes32 role) {
    if (!hasRole(role, address(0))) {
        _checkRole(role, _msgSender());
    }
    _;
}
```

Set up the admin role
to this contract

execute() function has the modifier
to check if the caller has a role of executor
or the executor role open to anyone (address(0))

Proposing

Governor.sol



```
    function propose(
        address[] memory targets,
        uint256[] memory values,
        bytes[] memory calldatas,
        string memory description
    ) public virtual override returns (uint256) {
        address proposer = _msgSender();
        uint256 currentTimepoint = clock();

        require(
            getVotes(proposer, currentTimepoint - 1) >= proposalThreshold(),
            "Governor: proposer votes below proposal threshold"
        );

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

        require(targets.length == values.length, "Governor: invalid proposal length");
        require(targets.length == calldatas.length, "Governor: invalid proposal length");
        require(targets.length > 0, "Governor: empty proposal");
        require(_proposals[proposalId].voteStart == 0, "Governor: proposal already exists");

        uint256 snapshot = currentTimepoint + votingDelay();
        uint256 deadline = snapshot + votingPeriod();

        _proposals[proposalId] = ProposalCore({
            proposer: proposer,
            voteStart: SafeCast.toUint64(snapshot),
            voteEnd: SafeCast.toUint64(deadline),
            executed: false,
            canceled: false,
            __gap_unused0: 0,
            __gap_unused1: 0
        });
```

Check if the proposer has more voting power than the proposal threshold

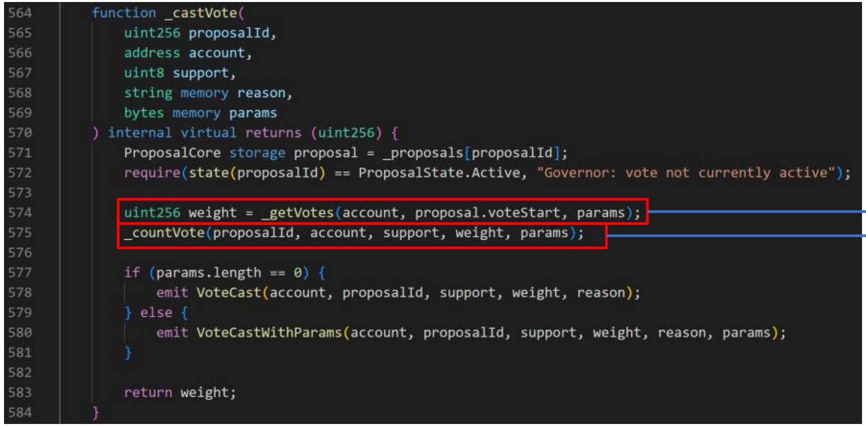
The proposal ID is the hash value of all input of the proposal

Setting the voting start time and end time

Adding a proposal to the proposals variable ***Note. it doesn't store the proposal content to save storage***

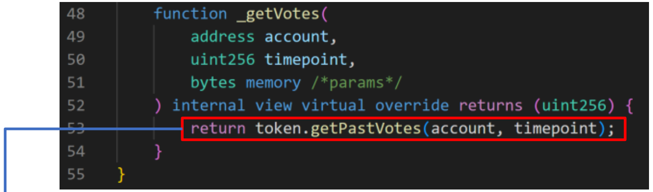
Casting a vote

Governor.sol



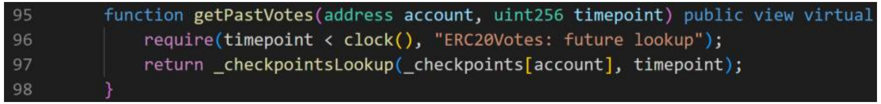
```
    function _castVote(
        uint256 proposalId,
        address account,
        uint8 support,
        string memory reason,
        bytes memory params
    ) internal virtual returns (uint256) {
        ProposalCore storage proposal = _proposals[proposalId];
        require(state(proposalId) == ProposalState.Active, "Governor: vote not currently active");

        uint256 weight = _getVotes(account, proposal.voteStart, params);
        _countVote(proposalId, account, support, weight, params);

        if (params.length == 0) {
            emit VoteCast(account, proposalId, support, weight, reason);
        } else {
            emit VoteCastWithParams(account, proposalId, support, weight, reason, params);
        }

        return weight;
    }
```

GovernorVotes.sol



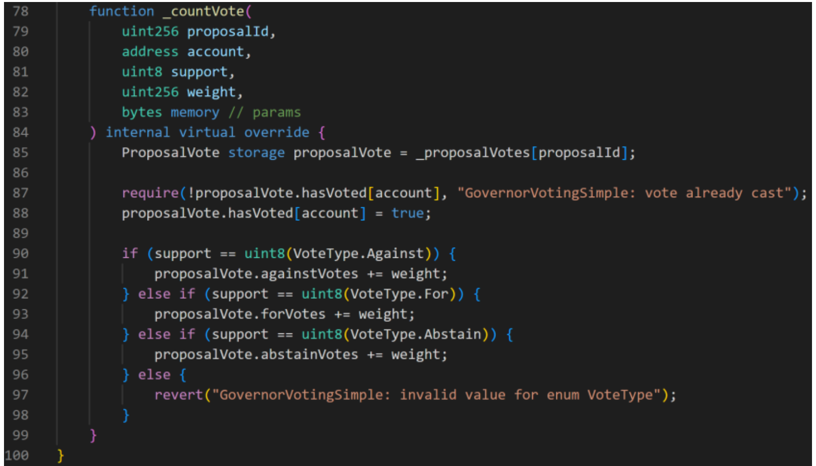
```
    function _getVotes(
        address account,
        uint256 timepoint,
        bytes memory /*params*/
    ) internal view virtual override returns (uint256) {
        return token.getPastVotes(account, timepoint);
    }
}
```

ERC20Votes.sol



```
    function getPastVotes(address account, uint256 timepoint) public view virtual
        require(timepoint < clock(), "ERC20Votes: future lookup");
        return _checkpointsLookup(_checkpoints[account], timepoint);
    }
```

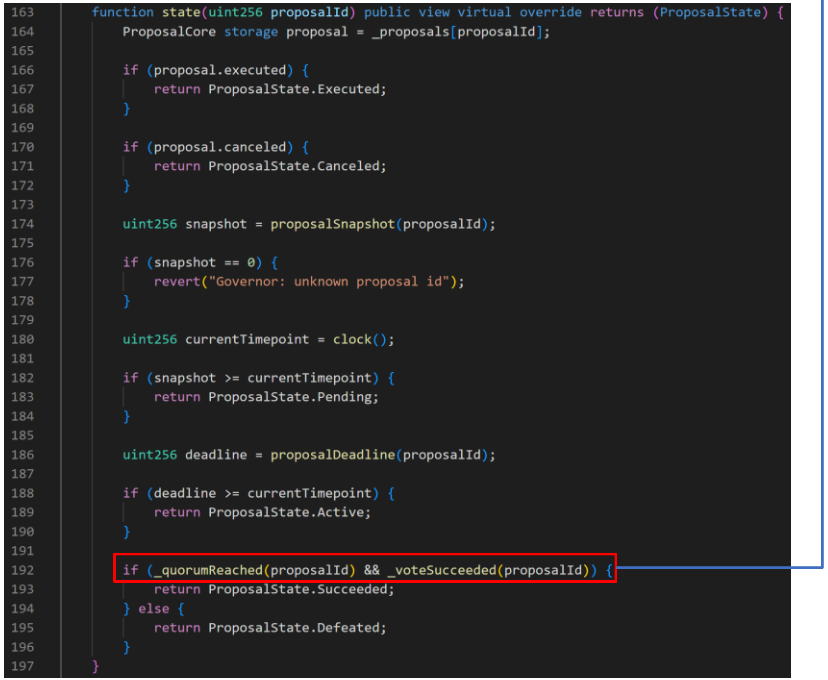
GovernorCountingSimple.sol



```
    function _countVote(
        uint256 proposalId,
        address account,
        uint8 support,
        uint256 weight,
        bytes memory // params
    ) internal virtual override {
        ProposalVote storage proposalVote = _proposalVotes[proposalId];

        require(!proposalVote.hasVoted[account], "GovernorVotingSimple: vote already cast");
        proposalVote.hasVoted[account] = true;

        if (support == uint8(VoteType.Against)) {
            proposalVote.againstVotes += weight;
        } else if (support == uint8(VoteType.For)) {
            proposalVote.forVotes += weight;
        } else if (support == uint8(VoteType.Abstain)) {
            proposalVote.abstainVotes += weight;
        } else {
            revert("GovernorVotingSimple: invalid value for enum VoteType");
        }
    }
}
```

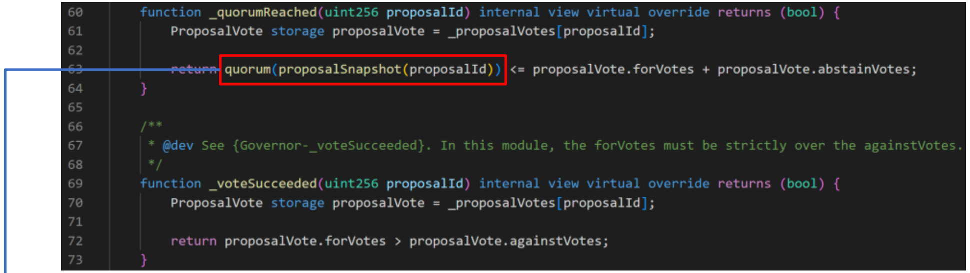
Checking the vote succeeded

Governor.sol



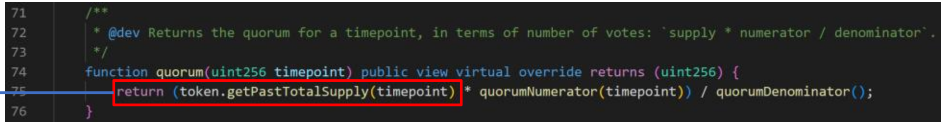
```
function state(uint256 proposalId) public view virtual override returns (ProposalState) {
    ProposalCore storage proposal = _proposals[proposalId];

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

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

    uint256 snapshot = proposalSnapshot(proposalId);

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

    uint256 currentTimepoint = clock();

    if (snapshot >= currentTimepoint) {
        return ProposalState.Pending;
    }

    uint256 deadline = proposalDeadline(proposalId);

    if (deadline >= currentTimepoint) {
        return ProposalState.Active;
    }

    if (_quorumReached(proposalId) && _voteSucceeded(proposalId)) {
        return ProposalState.Succeeded;
    } else {
        return ProposalState.Defeated;
    }
}
```

GovernorCountingSimple.sol



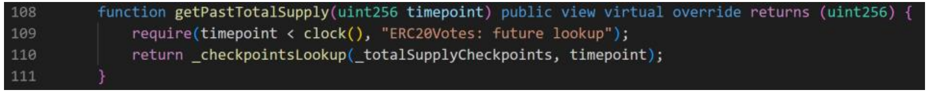
```
function _quorumReached(uint256 proposalId) internal view virtual override returns (bool) {
    ProposalVote storage proposalVote = _proposalVotes[proposalId];

    return quorum(proposalSnapshot(proposalId)) <= proposalVote.forVotes + proposalVote.abstainVotes;
}

/**
 * @dev See {Governor-_voteSucceeded}. In this module, the forVotes must be strictly over the againstVotes.
 */
function _voteSucceeded(uint256 proposalId) internal view virtual override returns (bool) {
    ProposalVote storage proposalVote = _proposalVotes[proposalId];

    return proposalVote.forVotes > proposalVote.againstVotes;
}
```

GovernorVotesQuorumFraction.sol



```
/**
 * @dev Returns the quorum for a timepoint, in terms of number of votes: `supply * numerator / denominator`.
 */
function quorum(uint256 timepoint) public view virtual override returns (uint256) {
    return (token.getPastTotalSupply(timepoint) * quorumNumerator(timepoint)) / quorumDenominator();
}
```

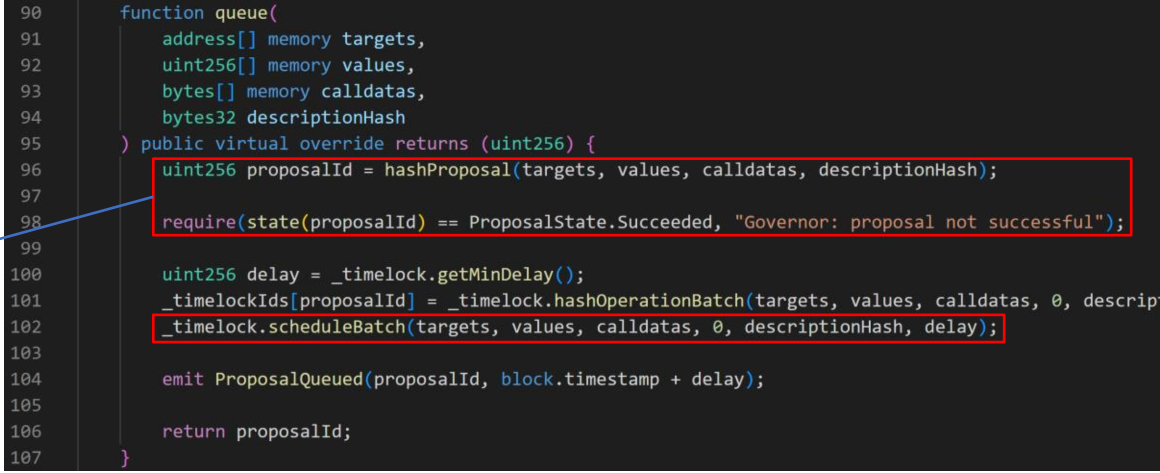
ERC20Votes.sol



```
function getPastTotalSupply(uint256 timepoint) public view virtual override returns (uint256) {
    require(timepoint < clock(), "ERC20Votes: future lookup");
    return _checkpointsLookup(_totalSupplyCheckpoints, timepoint);
}
```

Queuing

GovernorTimelockControl.sol

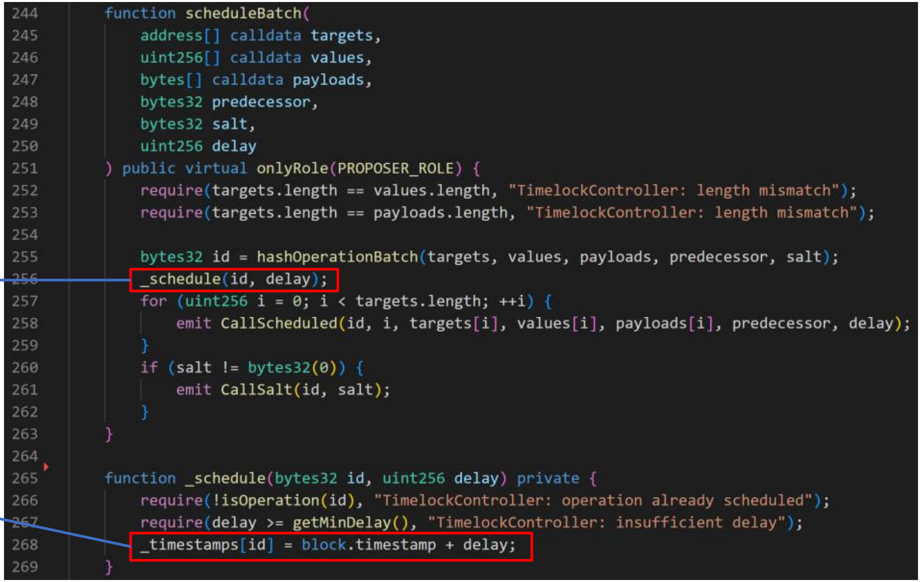


```
function queue(
    address[] memory targets,
    uint256[] memory values,
    bytes[] memory calldatas,
    bytes32 descriptionHash
) public virtual override returns (uint256) {
    uint256 proposalId = hashProposal(targets, values, calldatas, descriptionHash);

    require(state(proposalId) == ProposalState.Succeeded, "Governor: proposal not successful");

    uint256 delay = _timelock.getMinDelay();
    _timelockIds[proposalId] = _timelock.hashOperationBatch(targets, values, calldatas, 0, descrip
    _timelock.scheduleBatch(targets, values, calldatas, 0, descriptionHash, delay);

    emit ProposalQueued(proposalId, block.timestamp + delay);

    return proposalId;
}
```

Check the state of a proposal with the proposal ID

By the proposal ID, the proposal can be verified without saving the proposal content

TimelockController.sol



```
function scheduleBatch(
    address[] calldata targets,
    uint256[] calldata values,
    bytes[] calldata payloads,
    bytes32 predecessor,
    bytes32 salt,
    uint256 delay
) public virtual onlyRole(PROPOSER_ROLE) {
    require(targets.length == values.length, "TimelockController: length mismatch");
    require(targets.length == payloads.length, "TimelockController: length mismatch");

    bytes32 id = hashOperationBatch(targets, values, payloads, predecessor, salt);
    _schedule(id, delay);
    for (uint256 i = 0; i < targets.length; ++i) {
        emit CallScheduled(id, i, targets[i], values[i], payloads[i], predecessor, delay);
    }
    if (salt != bytes32(0)) {
        emit CallSalt(id, salt);
    }
}

function _schedule(bytes32 id, uint256 delay) private {
    require(!isOperation(id), "TimelockController: operation already scheduled");
    require(delay >= getMinDelay(), "TimelockController: insufficient delay");
    _timestamps[id] = block.timestamp + delay;
}
```

Schedule a proposal to execute after minDelay

Set the timestamp when the proposal is able to be executed after

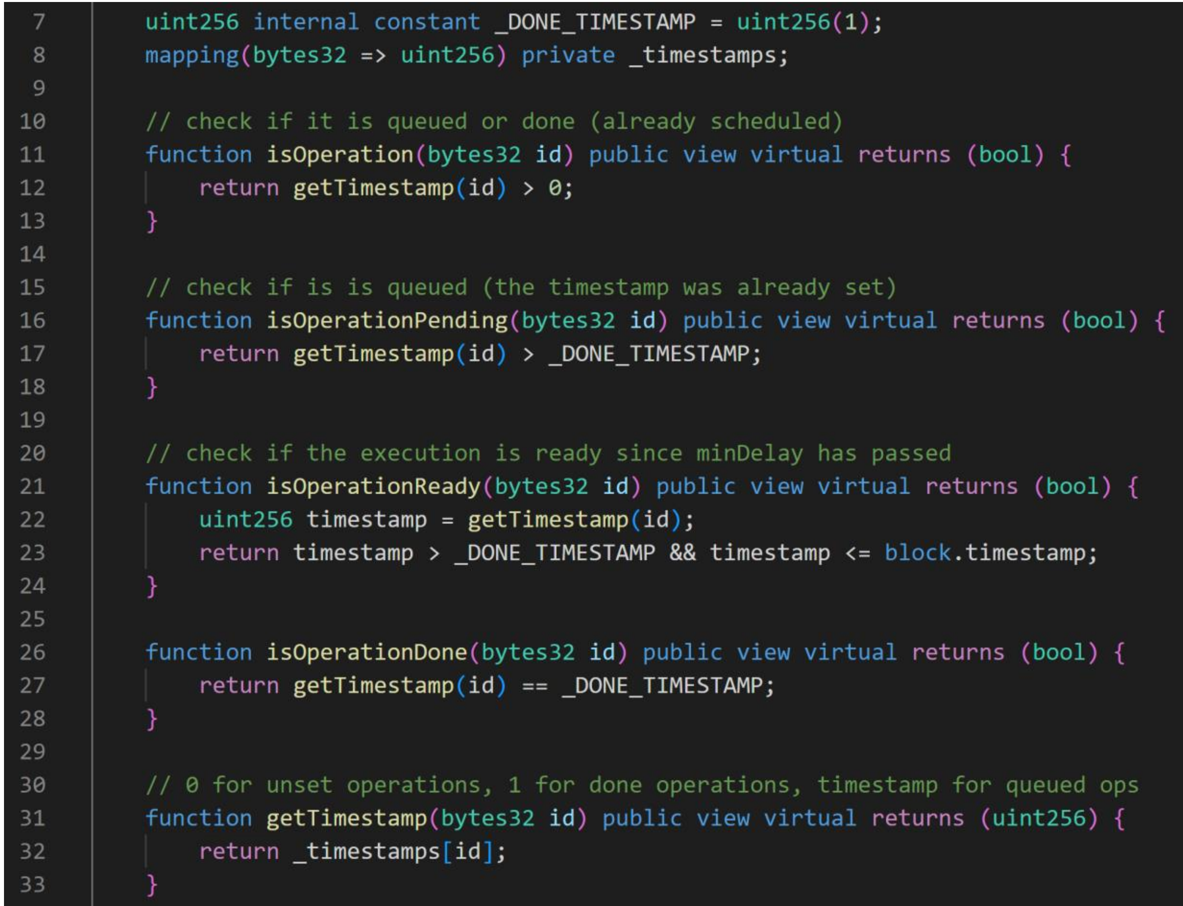
Checking the state of a proposal

TimelockController.sol

if _timestamp[id] is
0: unset (Vote)
>1: queued (Queue)
1: done (Execute)

Note.
queue() and execute() function will not be called automatically even after the condition is satisfied.

Someone should call the functions explicitly and then check if they can be executed



```
    uint256 internal constant _DONE_TIMESTAMP = uint256(1);
    mapping(bytes32 => uint256) private _timestamps;

    // check if it is queued or done (already scheduled)
    function isOperation(bytes32 id) public view virtual returns (bool) {
        return getTimestamp(id) > 0;
    }

    // check if is is queued (the timestamp was already set)
    function isOperationPending(bytes32 id) public view virtual returns (bool) {
        return getTimestamp(id) > _DONE_TIMESTAMP;
    }

    // check if the execution is ready since minDelay has passed
    function isOperationReady(bytes32 id) public view virtual returns (bool) {
        uint256 timestamp = getTimestamp(id);
        return timestamp > _DONE_TIMESTAMP && timestamp <= block.timestamp;
    }

    function isOperationDone(bytes32 id) public view virtual returns (bool) {
        return getTimestamp(id) == _DONE_TIMESTAMP;
    }

    // 0 for unset operations, 1 for done operations, timestamp for queued ops
    function getTimestamp(bytes32 id) public view virtual returns (uint256) {
        return _timestamps[id];
    }
```

Executing

TimelockController.sol

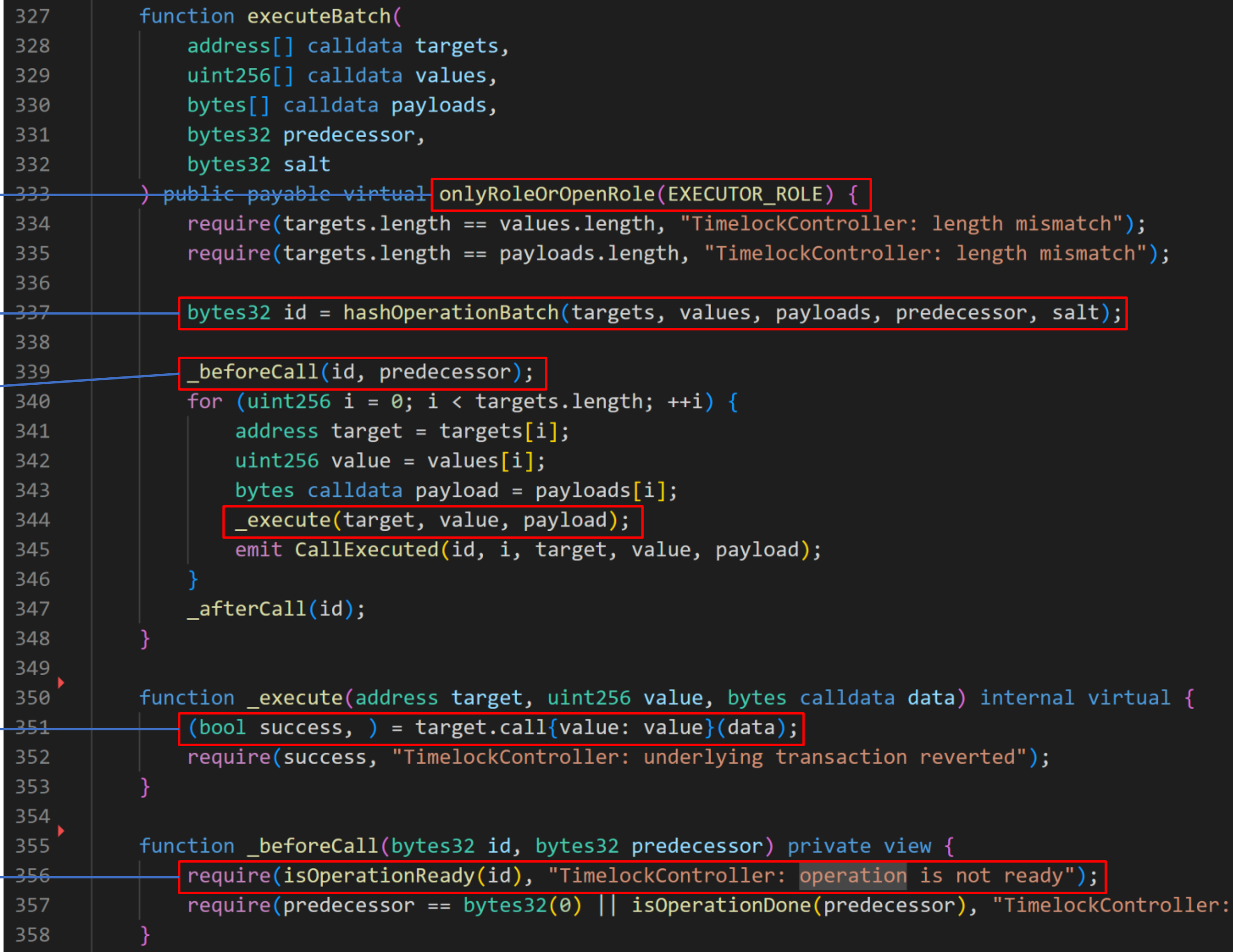
Only the executor role can run the function

Only the executor role can run the function

Check if the execution is ready

Send a tx request to the target contract with the ETH value and calldata

Check if minDelay has passed since the voting ended



```
function executeBatch(
    address[] calldata targets,
    uint256[] calldata values,
    bytes[] calldata payloads,
    bytes32 predecessor,
    bytes32 salt
) public payable virtual onlyRoleOrOpenRole(EXECUTOR_ROLE) {
    require(targets.length == values.length, "TimelockController: length mismatch");
    require(targets.length == payloads.length, "TimelockController: length mismatch");

    bytes32 id = hashOperationBatch(targets, values, payloads, predecessor, salt);

    _beforeCall(id, predecessor);
    for (uint256 i = 0; i < targets.length; ++i) {
        address target = targets[i];
        uint256 value = values[i];
        bytes calldata payload = payloads[i];
        _execute(target, value, payload);
        emit CallExecuted(id, i, target, value, payload);
    }
    _afterCall(id);
}

function _execute(address target, uint256 value, bytes calldata data) internal virtual {
    (bool success, ) = target.call{value: value}(data);
    require(success, "TimelockController: underlying transaction reverted");
}

function _beforeCall(bytes32 id, bytes32 predecessor) private view {
    require(isOperationReady(id), "TimelockController: operation is not ready");
    require(predecessor == bytes32(0) || isOperationDone(predecessor), "TimelockController:
}
```

Best practices to learn

- Create a main contract (Governor) and make it customizable through contract inheritance and wrapper contracts.

- Set up roles to control access to functions and set up states to control that only functions that fit the state are executed.

- To save blockchain storage, don't store large input parameters(proposal content). Instead, store only the hash value(proposalId) of the parameters. By hashing them in function calls and checking if it is the same value as before, we can verify if the parameter is the same in a series of function calls. The large parameters may be stored offchain.

Upgradeable Contracts

A smart contract is **immutable**.

Can we **upgrade** a smart contract?

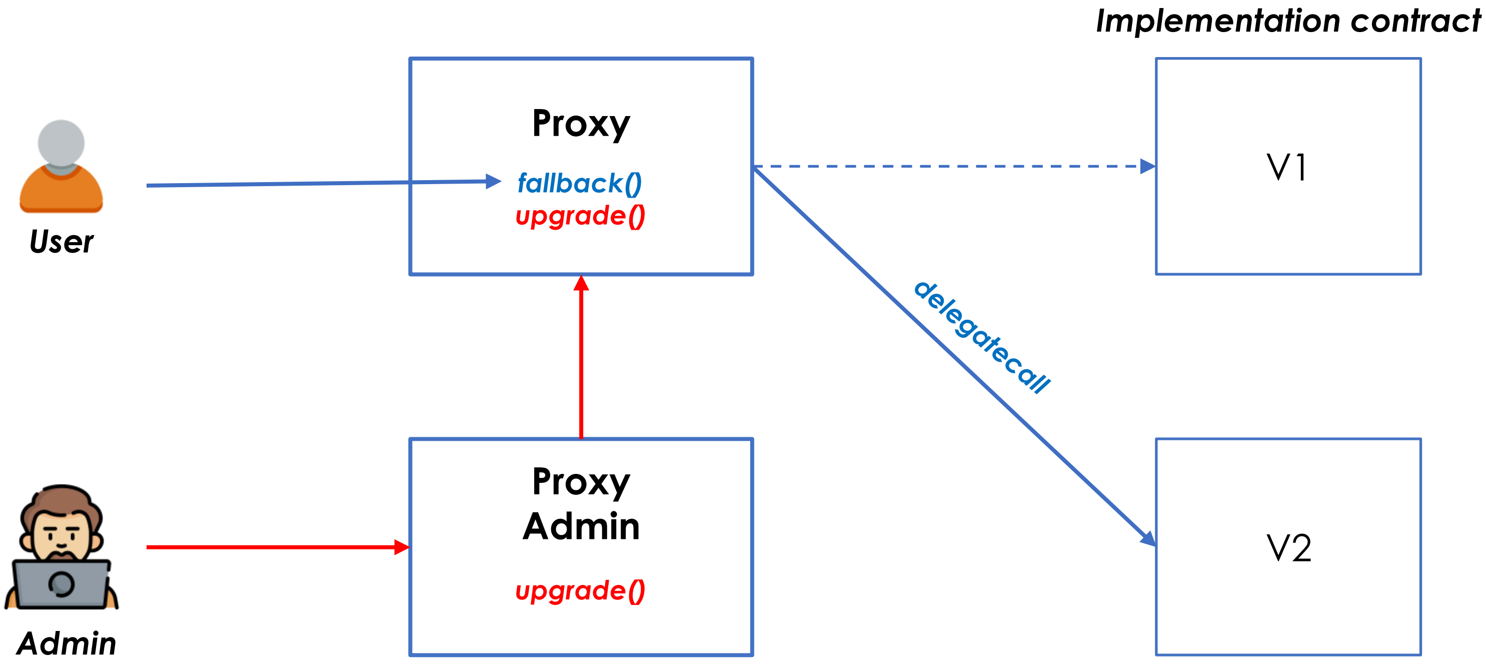
Three options of upgrading

- Set upgradable parameters
  e.g., setMiningRewards()

- Migrate all states and users from the old contract to the new one
  e.g., Uniswap V1, V2, V3 (Migrate from V2 to V3)

- Use a proxy

How a proxy works to upgrade a contract



https://www.youtube.com/watch?v=JgSj7IiE4jA

DelegateCall

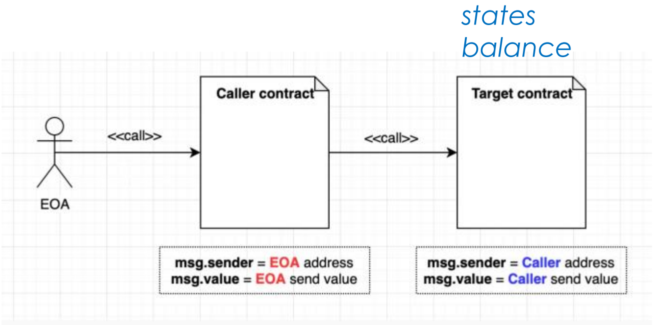
- **DelegateCall** is a low-level Solidity opcode that allows a contract to execute code from another contract, but it **using the state and the storage of the calling contract**.

- The syntax for DelegateCall

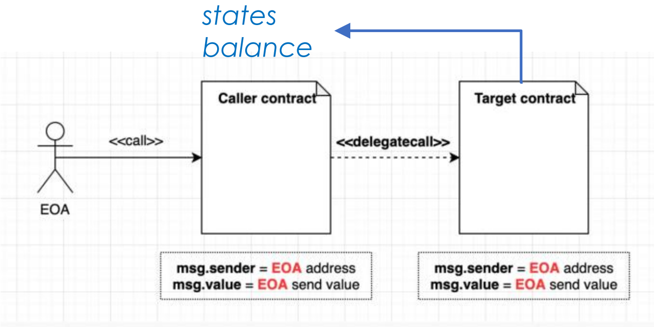
```
(bool success, bytes memory returnData) = address.delegatecall(bytes memory data);
```

the address is the address of contract to execute, and the data is the encoded function call to execute

- Call vs. DelegateCall



Context when the contract calls another contract

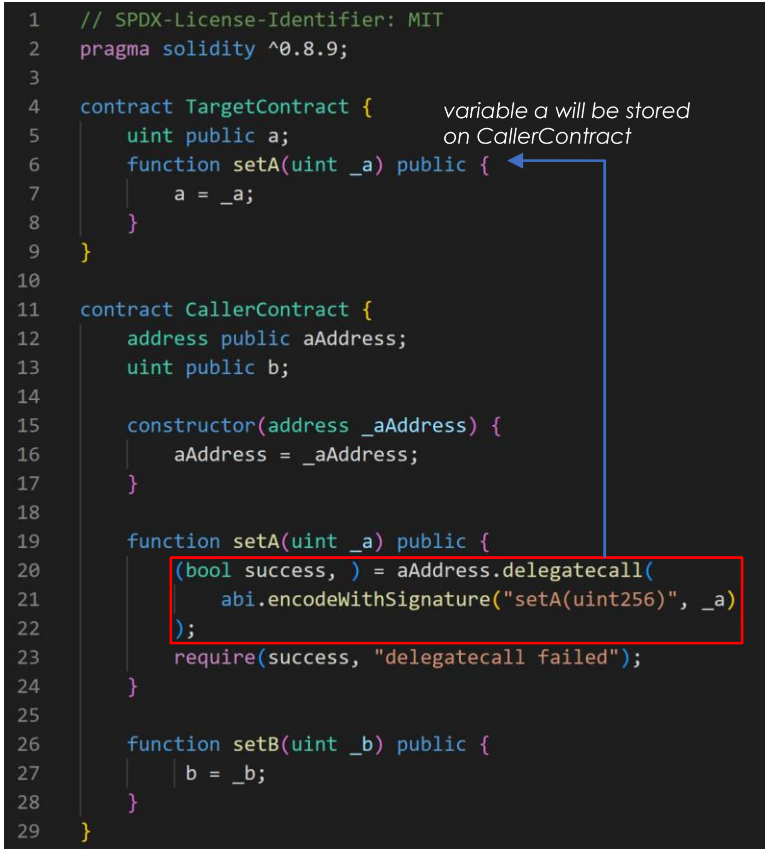


Context when contract delegatecall another contract

https://https://www.linkedin.com/pulse/delegatecall-solidity-some-code-examples-johnny-time/

DelegateCall Example

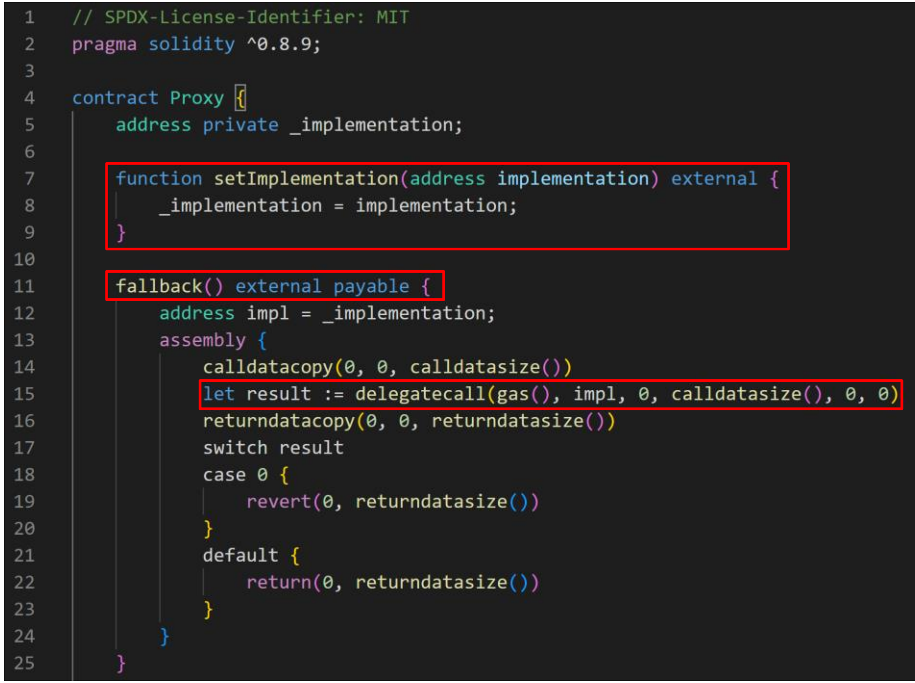
DelegateCallExample.sol



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

contract TargetContract {
    uint public a;
    function setA(uint _a) public {
        a = _a;
    }
}

contract CallerContract {
    address public aAddress;
    uint public b;

    constructor(address _aAddress) {
        aAddress = _aAddress;
    }

    function setA(uint _a) public {
        (bool success, ) = aAddress.delegatecall(
            abi.encodeWithSignature("setA(uint256)", _a)
        );
        require(success, "delegatecall failed");
    }

    function setB(uint _b) public {
         b = _b;
    }
}
```

variable a will be stored on CallerContract

Proxy.sol



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

contract Proxy {
    address private _implementation;

    function setImplementation(address implementation) external {
        _implementation = implementation;
    }

    fallback() external payable {
        address impl = _implementation;
        assembly {
            calldatacopy(0, 0, calldatasize())
            let result := delegatecall(gas(), impl, 0, calldatasize(), 0, 0)
            returndatacopy(0, 0, returndatasize())
            switch result
            case 0 {
                revert(0, returndatasize())
            }
            default {
                return(0, returndatasize())
            }
        }
    }
```

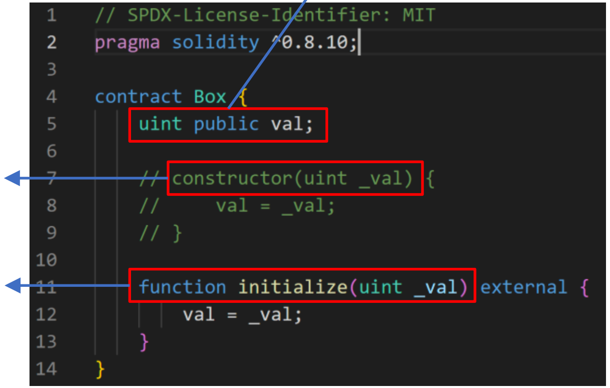
https://https://www.linkedin.com/pulse/delegatecall-solidity-some-code-examples-johnny-time/

Implementing Upgradeable contracts with OpenZeppelin & Hardhat

Implementation contracts

Same variable

Box.sol

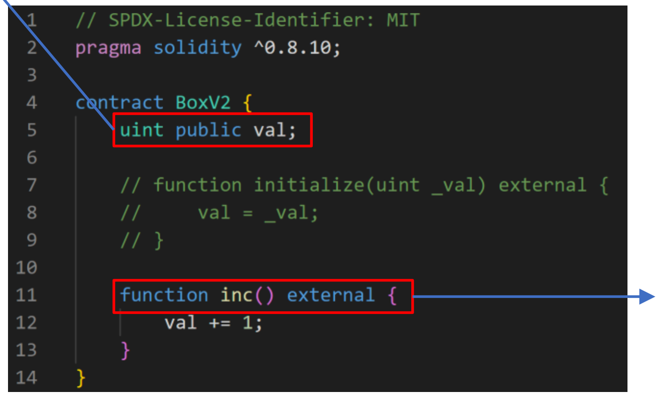


```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract Box {
    uint public val;

    // constructor(uint _val) {
    //     val = _val;
    // }

    function initialize(uint _val) external {
        val = _val;
    }
}
```

Shouldn't define a constructor

Instead of a constructor, use initialize()

BoxV2.sol



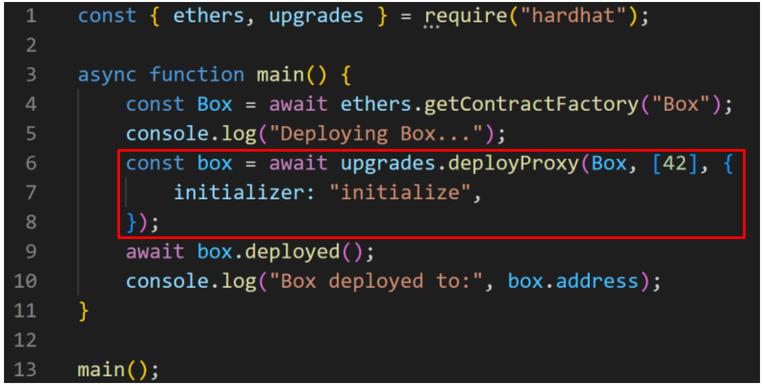
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract BoxV2 {
    uint public val;

    // function initialize(uint _val) external {
    //     val = _val;
    // }

    function inc() external {
        val += 1;
    }
}
```

New function

https://www.youtube.com/watch?v=JgSj7liE4jA
https://github.com/t4sk/hello-oz-upgradeable

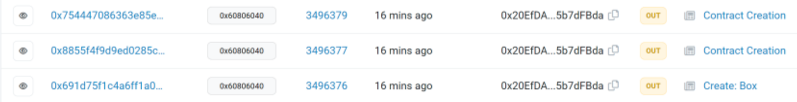
Deploy & Upgrade contracts

scripts/deploy_box_v1.js



```javascript
const { ethers, upgrades } = require("hardhat");

async function main() {
    const Box = await ethers.getContractFactory("Box");
    console.log("Deploying Box...");
    const box = await upgrades.deployProxy(Box, [42], {
        initializer: "initialize",
    });
    await box.deployed();
    console.log("Box deployed to:", box.address);
}

main();
```

TransparentUpgradeableProxy



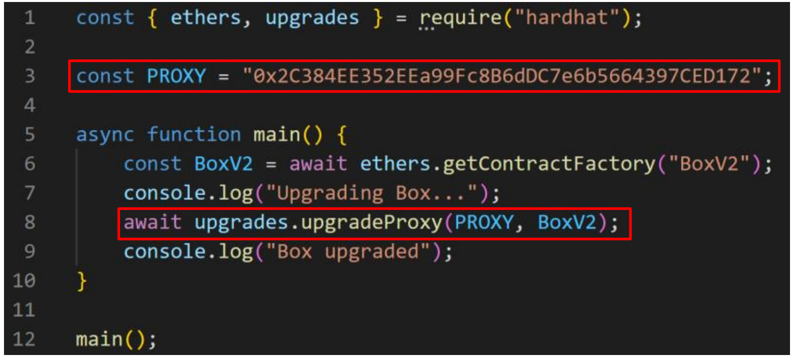
0x754447086363e85e...	0x60806040	3496379	16 mins ago	0x20EfDA...5b7dFBda	OUT	Contract Creation
0x8855f4f9d9ed0285c...	0x60806040	3496377	16 mins ago	0x20EfDA...5b7dFBda	OUT	Contract Creation
0x691d75f1c4a6ff1a0...	0x60806040	3496376	16 mins ago	0x20EfDA...5b7dFBda	OUT	Create: Box

ProxyAdmin

Box

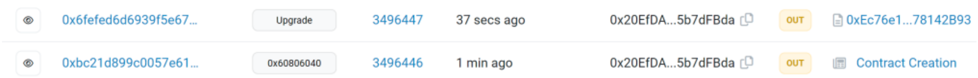
Etherscan: https://sepolia.etherscan.io/address/0x20efda938e7c1bf25ba7dc6b7a4ac8075b7dfbda

scripts/upgrade_box_v2.js



```javascript
const { ethers, upgrades } = require("hardhat");

const PROXY = "0x2C384EE352EEa99Fc8B6dDC7e6b5664397CED172";

async function main() {
    const BoxV2 = await ethers.getContractFactory("BoxV2");
    console.log("Upgrading Box...");
    await upgrades.upgradeProxy(PROXY, BoxV2);
    console.log("Box upgraded");
}

main();
```

ProxyAdmin.upgrade()



0x6fefed6d6939f5e67...	Upgrade	3496447	37 secs ago	0x20EfDA...5b7dFBda	OUT	0xEc76e1...78142B93
0xbc21d899c0057e61...	0x60806040	3496446	1 min ago	0x20EfDA...5b7dFBda	OUT	Contract Creation

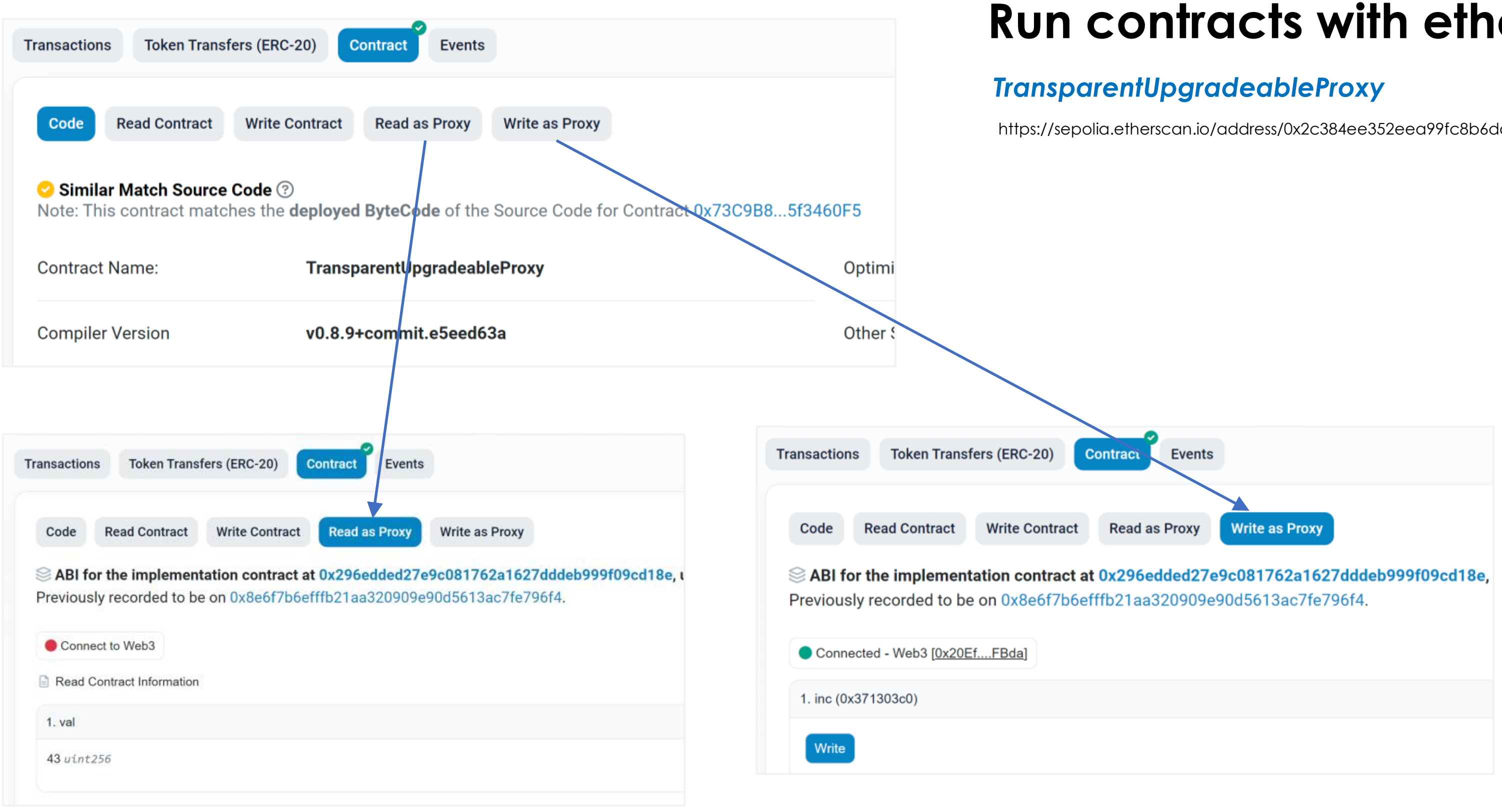
BoxV2

Etherscan: https://sepolia.etherscan.io/address/0x20efda938e7c1bf25ba7dc6b7a4ac8075b7dfbda

Run contracts with etherscan

TransparentUpgradeableProxy

https://sepolia.etherscan.io/address/0x2c384ee352eea99fc8b6ddc7e6b5664397ced172#code



Similar Match Source Code
Note: This contract matches the **deployed ByteCode** of the Source Code for Contract 0x73C9B8...5f3460F5

Contract Name: **TransparentUpgradeableProxy** Optimi

Compiler Version **v0.8.9+commit.e5eed63a** Other S

ABI for the implementation contract at 0x296edded27e9c081762a1627dddeb999f09cd18e,
Previously recorded to be on 0x8e6f7b6efffb21aa320909e90d5613ac7fe796f4.

Connect to Web3

Read Contract Information

1. val

43 *uint256*

ABI for the implementation contract at 0x296edded27e9c081762a1627dddeb999f09cd18e,
Previously recorded to be on 0x8e6f7b6efffb21aa320909e90d5613ac7fe796f4.

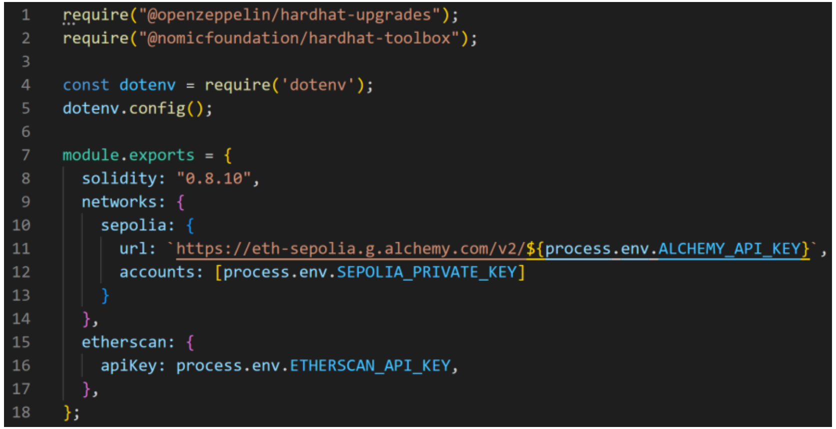
Connected - Web3 [0x20Ef....FBda]

1. inc (0x371303c0)

Write

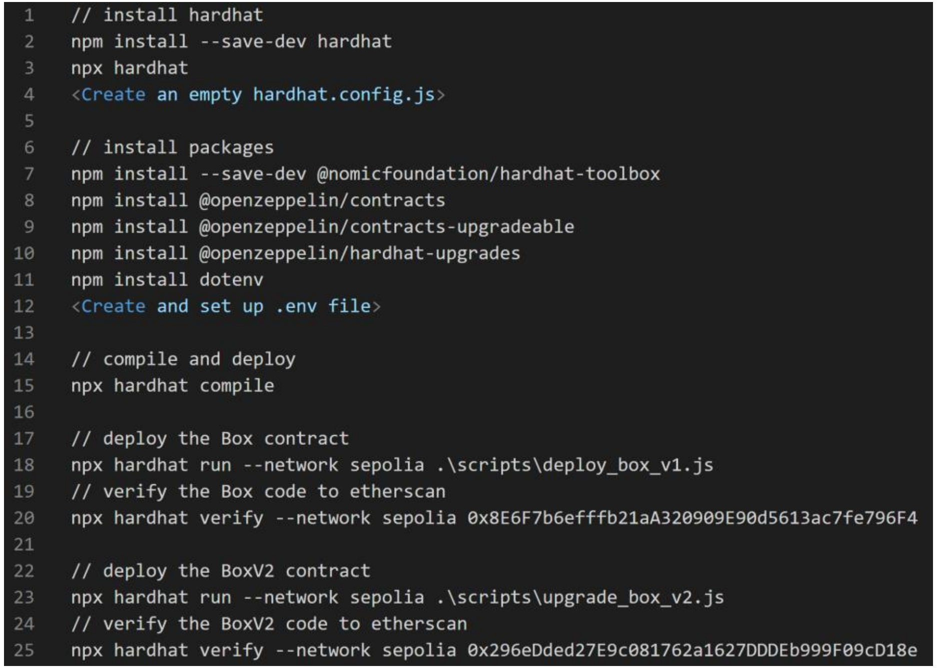
hardhat config file and commands

hardhat.config.js



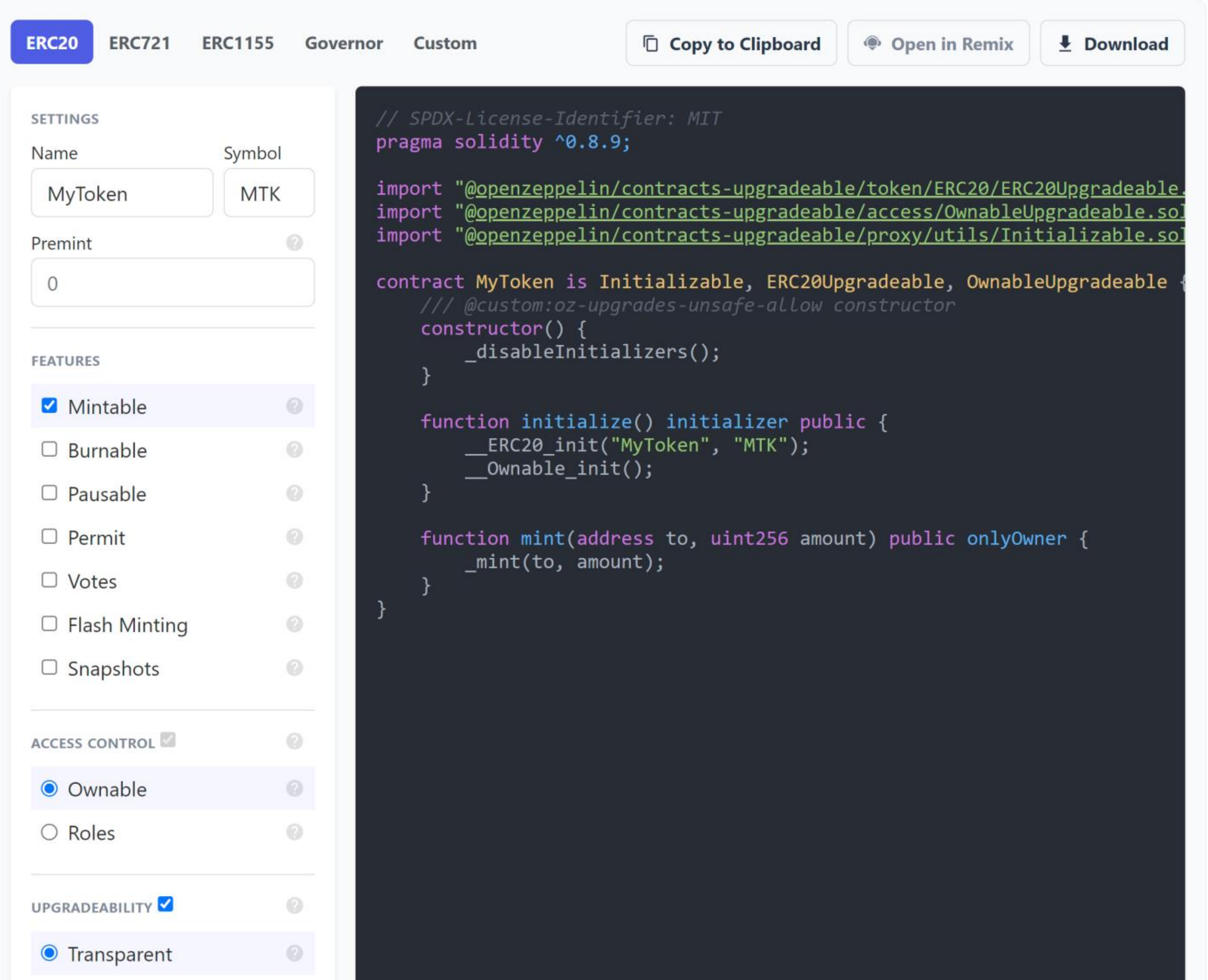
```
require("@openzeppelin/hardhat-upgrades");
require("@nomicfoundation/hardhat-toolbox");

const dotenv = require('dotenv');
dotenv.config();

module.exports = {
  solidity: "0.8.10",
  networks: {
    sepolia: {
      url: `https://eth-sepolia.g.alchemy.com/v2/${process.env.ALCHEMY_API_KEY}`,
      accounts: [process.env.SEPOLIA_PRIVATE_KEY]
    }
  },
  etherscan: {
    apiKey: process.env.ETHERSCAN_API_KEY,
  },
};
```

commands



```
// install hardhat
npm install --save-dev hardhat
npx hardhat
<Create an empty hardhat.config.js>

// install packages
npm install --save-dev @nomicfoundation/hardhat-toolbox
npm install @openzeppelin/contracts
npm install @openzeppelin/contracts-upgradeable
npm install @openzeppelin/hardhat-upgrades
npm install dotenv
<Create and set up .env file>

// compile and deploy
npx hardhat compile

// deploy the Box contract
npx hardhat run --network sepolia .\scripts\deploy_box_v1.js
// verify the Box code to etherscan
npx hardhat verify --network sepolia 0x8E6F7b6efffb21aA320909E90d5613ac7fe796F4

// deploy the BoxV2 contract
npx hardhat run --network sepolia .\scripts\upgrade_box_v2.js
// verify the BoxV2 code to etherscan
npx hardhat verify --network sepolia 0x296eDded27E9c081762a1627DDDEb999F09cD18e
```

Upgradeable ERC20 tokens



SETTINGS

Name: MyToken

Symbol: MTK

Premint: 0

FEATURES

☑ Mintable

☐ Burnable

☐ Pausable

☐ Permit

☐ Votes

☐ Flash Minting

☐ Snapshots

ACCESS CONTROL ☑

● Ownable

○ Roles

UPGRADEABILITY ☑

● Transparent

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

import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol

contract MyToken is Initializable, ERC20Upgradeable, OwnableUpgradeable {
    /// @custom:oz-upgrades-unsafe-allow constructor
    constructor() {
        _disableInitializers();
    }

    function initialize() initializer public {
        __ERC20_init("MyToken", "MTK");
        __Ownable_init();
    }

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
}
```

https://www.youtube.com/watch?v=Vt20jCu8OC8
https://github.com/t4sk/hello-oz-upgradeable

Restrictions of upgradeable contracts

- **Use initialize(), a regular function to run all the setup logic instead of a constructor**
- Inherit an Initializable contract and use an initializer modifier in order not to call initialize() multiple times
- Avoid initial values in field declarations. Make sure that all initial values are set in an initializer function (ok to define constant state variables)
- Invoke the _disableInitializers function in the constructor to automatically lock it when it is deployed

- Use @openzeppelin/contracts-upgradeable for libraries and contracts instead of @openzeppelin/contracts

- You cannot change the order in which the contract state variables are declared, nor their type. **If you need to introduce a new variable, make sure you always do so at the end.**

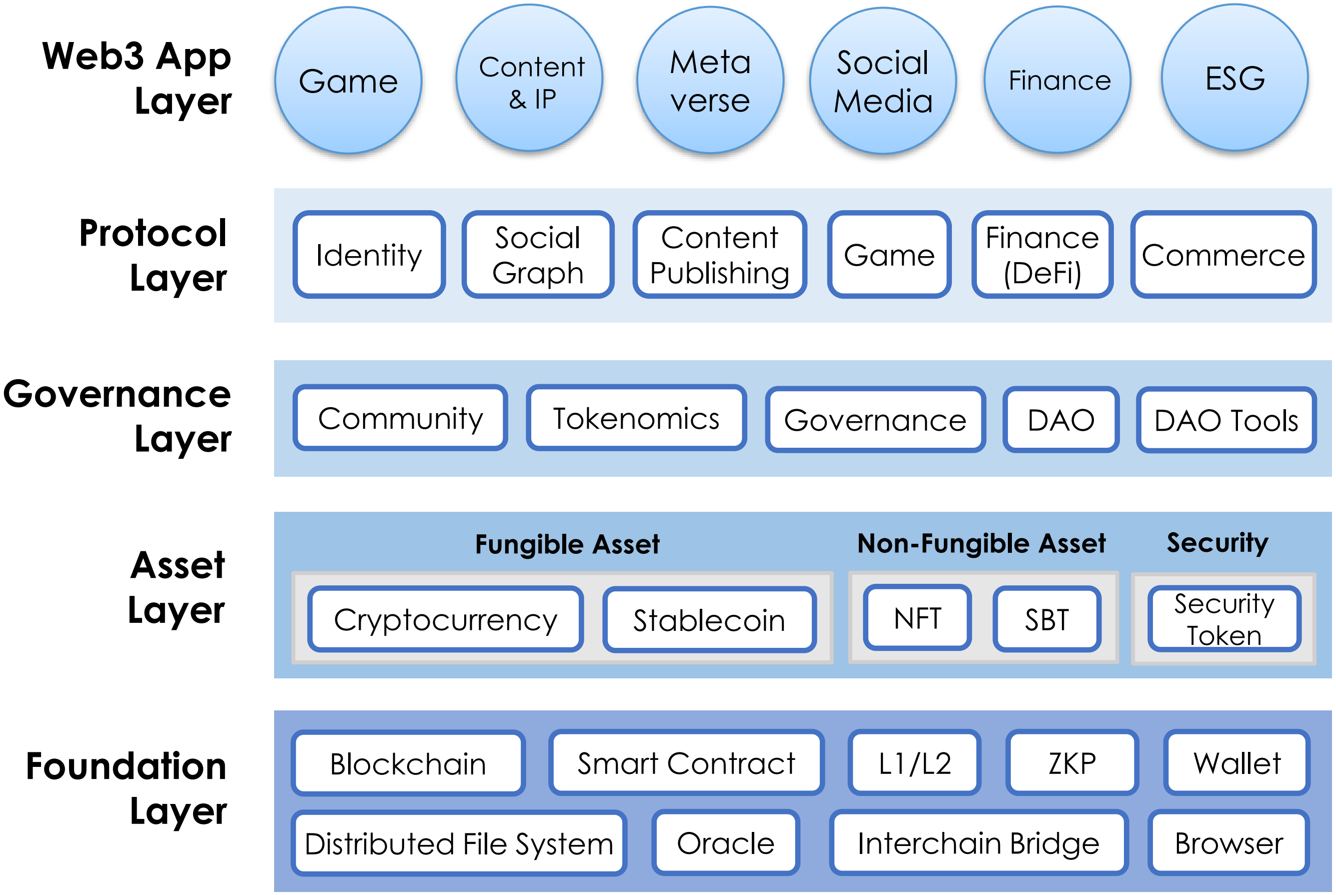
- Read OpenZeppelin docs for more restrictions

Wrap-up

We Learned

- Access Control
- Onchain governance
- Developing onchain governance with OpenZeppelin
- Technical detail on OpenZeppelin governance
- Upgradeable contract
- Developing upgradeable contract with OpenZeppelin

Revisit: Web3 Stack from the first lecture



Web3 App Layer

Game, Content & IP, Meta verse, Social Media, Finance, ESG

Protocol Layer

Identity, Social Graph, Content Publishing, Game, Finance (DeFi), Commerce

Governance Layer

Community, Tokenomics, Governance, DAO, DAO Tools

Asset Layer

Fungible Asset: Cryptocurrency, Stablecoin

Non-Fungible Asset: NFT, SBT

Security: Security Token

Foundation Layer

Blockchain, Smart Contract, L1/L2, ZKP, Wallet

Distributed File System, Oracle, Interchain Bridge, Browser

In terms of web3 technology, we covered three layers from the bottom.
You had a skill to develop your own basic web3 apps

Lecture 19
Access control,
on-chain governance,
upgradeable contracts

Lecture 15, 17
non-standard token, ERC20,
ERC721(NFT), ERC1155,
ERC5192(SBT), Oracle, IPFS

Lecture 1 – 12
Web3 staaks, blockchain tech,
programming solidity,
smart contracts, building dapps,
web3 security