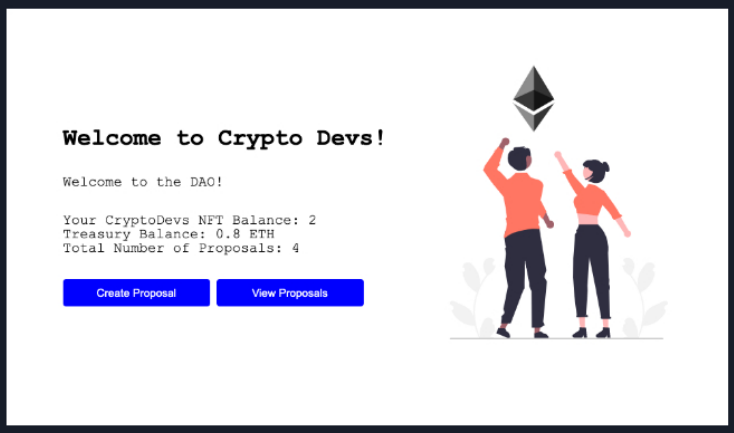
Lesson Type: Practical | Estimated Time: 8-12 hours | Current Score: 0%

Build a DAO for your NFT holders



What is a DAO?

DAO stands for **D**ecentralized **A**utonomous **O**rganization. You can think of DAOs as analogous to companies in the real world. Essentially, DAOs allow for members to create and vote on governance decisions.

In traditional companies, when a decision needs to be made, the board of directors or executives of the company are in charge of making that decision. In a DAO, however, this process is democratized, and any member can create a proposal, and all other members can vote on it. Each proposal created has a deadline for voting, and after the deadline the decision is made in favour of the voting outcome (YES or NO).

Membership in DAOs is typically restricted either by ownership of ERC20 tokens, or by ownership of NFTs. Examples of DAOs where membership and voting power is proportional to how many tokens you own include **Uniswap** and **ENS**. Examples of DAOs where they are based on NFTs include **Meebits DAO**.

Building our DAO

You want to launch a DAO for holders of your `CryptoDevs` NFTs. From the ETH that was gained through the ICO, you built up a DAO Treasury. The DAO now has a lot of ETH, but currently does nothing with it.

You want to allow your NFT holders to create and vote on proposals to use that ETH for purchasing other NFTs from an NFT marketplace, and speculate on price. Maybe in the future when you sell the NFT back, you split the profits among all members of the DAO.

Requirements

- Anyone with a `CryptoDevs` NFT can create a proposal to purchase a different NFT from an NFT marketplace
- Everyone with a `CryptoDevs` NFT can vote for or against the active proposals
- Each NFT counts as one vote for each proposal
- Voter cannot vote multiple times on the same proposal with the same NFT
- If majority of the voters vote for the proposal by the deadline, the NFT purchase is automatically executed

What we will make

- To be able to purchase NFTs automatically when a proposal is passed, you need an on-chain NFT marketplace that you can call a `purchase()` function on. There exist a lot of NFT marketplaces out there, but to avoid overcomplicating things, we will create a simplified fake NFT marketplace for this tutorial as the focus is on the DAO.
- We will also make the actual DAO smart contract using Hardhat.
- We will make the website using Next.js to allow users to create and vote on proposals

Prerequisites

- You have completed the `NFT Collection` tutorial from earlier.
- You must have some ETH to give to the DAO Treasury

BUIDL IT

Smart Contract Development

We will start off with first creating the smart contracts. We will be making two smart contracts:

- `FakeNFTMarketplace.sol`
- `CryptoDevsDAO.sol`

To do so, we will use the **Hardhat** development framework we have been using for the last few tutorials.

Create a folder for this project named `DAO-Tutorial`, and open up a Terminal window in that folder.

Setup a new hardhat project by running the following commands in your terminal:

```
mkdir hardhat-tutorial
cd hardhat-tutorial
npm init --yes
npm install --save-dev hardhat
```

Now that you have installed Hardhat, we can setup a project. Execute the following command in your terminal.

In the same directory where you installed Hardhat run:

```
npx hardhat
```

Make sure you select `Create a Javascript Project` and then follow the steps in the terminal to complete your Hardhat setup.

Now, let's install the `@openzeppelin/contracts` package from NPM as we will be using **OpenZeppelin's Ownable Contract** for the DAO contract.

```
npm install @openzeppelin/contracts
```

First, let's make a simple Fake NFT Marketplace. Create a file named `FakeNFTMarketplace.sol` under the `contracts` directory within `hardhat-tutorial`, and add the following code.

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

contract FakeNFTMarketplace {
    /// @dev Maintain a mapping of Fake TokenID to Owner addresses
    mapping(uint256 => address) public tokens;
    /// @dev Set the purchase price for each Fake NFT
    uint256 nftPrice = 0.1 ether;

    /// @dev purchase() accepts ETH and marks the owner of the given tokenId as the caller address
    /// @param _tokenId - the fake NFT token Id to purchase
    function purchase(uint256 _tokenId) external payable {
```

```
        require(msg.value == nftPrice, "This NFT costs 0.1 ether");
        tokens[_tokenId] = msg.sender;
    }

    /// @dev getPrice() returns the price of one NFT
    function getPrice() external view returns (uint256) {
        return nftPrice;
    }

    /// @dev available() checks whether the given tokenId has already been sold or not
    /// @param _tokenId - the tokenId to check for
    function available(uint256 _tokenId) external view returns (bool) {
        // address(0) = 0x0000000000000000000000000000000000000000
        // This is the default value for addresses in Solidity
        if (tokens[_tokenId] == address(0)) {
            return true;
        }
        return false;
    }
}
```

The `FakeNFTMarketplace` exposes some basic functions that we will be using from the DAO contract to purchase NFTs if a proposal is passed. A real NFT marketplace would be more complicated - as not all NFTs have the same price.

Let's make sure everything compiles before we start writing the DAO Contract. Run the following command inside the `hardhat-tutorial` folder from your Terminal.

```
npx hardhat compile
```

and make sure there are no compilation errors.

Now, we will start writing the `CryptoDevsDAO` contract. Since this is mostly a completely custom contract, and relatively more complicated than what we have done so far, we will explain this one bit-by-bit. First, let's write the boilerplate code for the contract. Create a new file named `CryptoDevsDAO.sol` under the `contracts` directory in `hardhat-tutorial` and add the following code to it.

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

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

// We will add the Interfaces here

contract CryptoDevsDAO is Ownable {
    // We will write contract code here
}
```

Now, we will need to call functions on the `FakeNFTMarketplace` contract, and your previously deployed `CryptoDevs NFT` contract. Recall from the `Advanced Solidity Topics` tutorial that we need to provide an interface for those contracts, so this contract knows which functions are available to call and what they take as parameters and what they return.

Add the following two interfaces to your code by adding the following code

```
/**
 * Interface for the FakeNFTMarketplace
 */
interface IFakeNFTMarketplace {
    /// @dev getPrice() returns the price of an NFT from the FakeNFTMarketplace
    /// @return Returns the price in Wei for an NFT
    function getPrice() external view returns (uint256);

    /// @dev available() returns whether or not the given _tokenId has already been purchased
    /// @return Returns a boolean value - true if available, false if not
    function available(uint256 _tokenId) external view returns (bool);

    /// @dev purchase() purchases an NFT from the FakeNFTMarketplace
    /// @param _tokenId - the fake NFT tokenID to purchase
    function purchase(uint256 _tokenId) external payable;
}

/**
 * Minimal interface for CryptoDevsNFT containing only two functions
 * that we are interested in
 */
```

```
interface ICryptoDevsNFT {
    /// @dev balanceOf returns the number of NFTs owned by the given address
    /// @param owner - address to fetch number of NFTs for
    /// @return Returns the number of NFTs owned
    function balanceOf(address owner) external view returns (uint256);

    /// @dev tokenOfOwnerByIndex returns a tokenID at given index for owner
    /// @param owner - address to fetch the NFT TokenID for
    /// @param index - index of NFT in owned tokens array to fetch
    /// @return Returns the TokenID of the NFT
    function tokenOfOwnerByIndex(address owner, uint256 index)
        external
        view
        returns (uint256);
}
```

Now, let's think about what functionality we need in the DAO contract.

- Store created proposals in contract state
- Allow holders of the CryptoDevs NFT to create new proposals
- Allow holders of the CryptoDevs NFT to vote on proposals, given they haven't already voted, and that the proposal hasn't passed it's deadline yet
- Allow holders of the CryptoDevs NFT to execute a proposal after it's deadline has been exceeded, triggering an NFT purchase in case it passed

Let's start off by creating a `struct` representing a `Proposal`. In your contract, add the following code:

```
// Create a struct named Proposal containing all relevant information
struct Proposal {
    // nftTokenId - the tokenID of the NFT to purchase from FakeNFTMarketplace if the proposal passes
    uint256 nftTokenId;
    // deadline - the UNIX timestamp until which this proposal is active. Proposal can be executed after the deadline
    uint256 deadline;
    // yayVotes - number of yay votes for this proposal
    uint256 yayVotes;
    // nayVotes - number of nay votes for this proposal
    uint256 nayVotes;
    // executed - whether or not this proposal has been executed yet. Cannot be executed before the deadline has been
    bool executed;
    // voters - a mapping of CryptoDevsNFT tokenIDs to booleans indicating whether that NFT has already been used to
    mapping(uint256 => bool) voters;
}
```

Let's also create a mapping from Proposal IDs to Proposals to hold all created proposals, and a counter to count the number of proposals that exist.

```
// Create a mapping of ID to Proposal
mapping(uint256 => Proposal) public proposals;
// Number of proposals that have been created
uint256 public numProposals;
```

Now, since we will be calling functions on the `FakeNFTMarketplace` and `CryptoDevsNFT` contract, let's initialize variables for those contracts.

```
IFakeNFTMarketplace nftMarketplace;
ICryptoDevsNFT cryptoDevsNFT;
```

Create a `constructor` function that will initialize those contract variables, and also accept an ETH deposit from the deployer to fill the DAO ETH treasury. (In the background, since we imported the `Ownable` contract, this will also set the contract deployer as the owner of this contract)

```
// Create a payable constructor which initializes the contract
// instances for FakeNFTMarketplace and CryptoDevsNFT
// The payable allows this constructor to accept an ETH deposit when it is being deployed
constructor(address _nftMarketplace, address _cryptoDevsNFT) payable {
    nftMarketplace = IFakeNFTMarketplace(_nftMarketplace);
    cryptoDevsNFT = ICryptoDevsNFT(_cryptoDevsNFT);
}
```

Now, since we want pretty much all of our other functions to only be called by those who own NFTs from the `CryptoDevs NFT` contract, we will create a `modifier` to avoid duplicating code.

```
// Create a modifier which only allows a function to be
// called by someone who owns at least 1 CryptoDevsNFT
modifier nftHolderOnly() {
    require(cryptoDevsNFT.balanceOf(msg.sender) > 0, "NOT_A_DAO_MEMBER");
    _;
}
```

We now have enough to write our `createProposal` function, which will allow members to create new proposals.

```
/// @dev createProposal allows a CryptoDevsNFT holder to create a new proposal in the DAO
/// @param _nftTokenId - the tokenID of the NFT to be purchased from FakeNFTMarketplace if this proposal passes
/// @return Returns the proposal index for the newly created proposal
function createProposal(uint256 _nftTokenId)
    external
    nftHolderOnly
    returns (uint256)
{
    require(nftMarketplace.available(_nftTokenId), "NFT_NOT_FOR_SALE");
    Proposal storage proposal = proposals[numProposals];
    proposal.nftTokenId = _nftTokenId;
    // Set the proposal's voting deadline to be (current time + 5 minutes)
    proposal.deadline = block.timestamp + 5 minutes;

    numProposals++;

    return numProposals - 1;
}
```

Now, to vote on a proposal, we want to add an additional restriction that the proposal being voted on must not have had it's deadline exceeded. To do so, we will create a second modifier.

```
// Create a modifier which only allows a function to be
// called if the given proposal's deadline has not been exceeded yet
modifier activeProposalOnly(uint256 proposalIndex) {
    require(
        proposals[proposalIndex].deadline > block.timestamp,
        "DEADLINE_EXCEEDED"
    );
    _;
}
```

Note how this modifier takes a parameter!

Additionally, since a vote can only be one of two values (YAY or NAY) - we can create an `enum` representing possible options.

```
// Create an enum named Vote containing possible options for a vote
enum Vote {
    YAY, // YAY = 0
    NAY // NAY = 1
}
```

Let's write the `voteOnProposal` function

```
/// @dev voteOnProposal allows a CryptoDevsNFT holder to cast their vote on an active proposal
/// @param proposalIndex - the index of the proposal to vote on in the proposals array
/// @param vote - the type of vote they want to cast
function voteOnProposal(uint256 proposalIndex, Vote vote)
    external
    nftHolderOnly
    activeProposalOnly(proposalIndex)
{
    Proposal storage proposal = proposals[proposalIndex];

    uint256 voterNFTBalance = cryptoDevsNFT.balanceOf(msg.sender);
    uint256 numVotes = 0;

    // Calculate how many NFTs are owned by the voter
    // that haven't already been used for voting on this proposal
    for (uint256 i = 0; i < voterNFTBalance; i++) {
        uint256 tokenId = cryptoDevsNFT.tokenOfOwnerByIndex(msg.sender, i);
        if (proposal.voters[tokenId] == false) {
            numVotes++;
            proposal.voters[tokenId] = true;
```

```
            }
        }
        require(numVotes > 0, "ALREADY_VOTED");

        if (vote == Vote.YAY) {
            proposal.yayVotes += numVotes;
        } else {
            proposal.nayVotes += numVotes;
        }
    }
```

We're almost done! To execute a proposal whose deadline has exceeded, we will create our final modifier.

```
// Create a modifier which only allows a function to be
// called if the given proposals' deadline HAS been exceeded
// and if the proposal has not yet been executed
modifier inactiveProposalOnly(uint256 proposalIndex) {
    require(
        proposals[proposalIndex].deadline <= block.timestamp,
        "DEADLINE_NOT_EXCEEDED"
    );
    require(
        proposals[proposalIndex].executed == false,
        "PROPOSAL_ALREADY_EXECUTED"
    );
    _;
}
```

Note this modifier also takes a parameter!

Let's write the code for `executeProposal`

```
/// @dev executeProposal allows any CryptoDevsNFT holder to execute a proposal after it's deadline has been exceeded
/// @param proposalIndex - the index of the proposal to execute in the proposals array
function executeProposal(uint256 proposalIndex)
    external
    nftHolderOnly
    inactiveProposalOnly(proposalIndex)
{
    Proposal storage proposal = proposals[proposalIndex];

    // If the proposal has more YAY votes than NAY votes
    // purchase the NFT from the FakeNFTMarketplace
    if (proposal.yayVotes > proposal.nayVotes) {
        uint256 nftPrice = nftMarketplace.getPrice();
        require(address(this).balance >= nftPrice, "NOT_ENOUGH_FUNDS");
        nftMarketplace.purchase{value: nftPrice}(proposal.nftTokenId);
    }
    proposal.executed = true;
}
```

We have at this point implemented all the core functionality. However, there are a couple of additional features we could and should implement.

- Allow the contract owner to withdraw the ETH from the DAO if needed
- Allow the contract to accept further ETH deposits

The `Ownable` contract we inherit from contains a modifier `onlyOwner` which restricts a function to only be able to be called by the contract owner. Let's implement `withdrawEther` using that modifier.

```
/// @dev withdrawEther allows the contract owner (deployer) to withdraw the ETH from the contract
function withdrawEther() external onlyOwner {
    payable(owner()).transfer(address(this).balance);
}
```

This will transfer the entire ETH balance of the contract to the owner address

Finally, to allow for adding more ETH deposits to the DAO treasury, we need to add some special functions. Normally, contract addresses cannot accept ETH sent to them, unless it was through a `payable` function. But we don't want users to call functions just to deposit money, they should be able to tranfer ETH directly from their wallet. For that, let's add these two functions:

```
// The following two functions allow the contract to accept ETH deposits
// directly from a wallet without calling a function
```

```
receive() external payable {}

fallback() external payable {}
```

Smart Contract Deployment

Now that we have written both our contracts, let's deploy them to the **Goerli Testnet**. Ensure you have some ETH on the Goerli Testnet.

Install the `dotenv` package from NPM to be able to use environment variables specified in `.env` files in the `hardhat.config.js`. Execute the following command in your Terminal in the `hardhat-tutorial` directory.

```
npm install dotenv
```

Now create a `.env` file in the `hardhat-tutorial` folder and add the following lines. Follow the instructions below.

Go to **Quicknode** and sign up for an account. If you already have an account, log in. Quicknode is a node provider that lets you connect to various different blockchains. We will be using it to deploy our contract through Hardhat. After creating an account, `Create an endpoint` on Quicknode, select `Ethereum`, and then select the `Goerli` network. Click `Continue` in the bottom right and then click on `Create Endpoint`. Copy the link given to you in `HTTP Provider` and add it to the `.env` file below for `QUICKNODE_HTTP_URL`.

> NOTE: If you previously set up a Goerli Endpoint on Quicknode during the Freshman Track, you can use the same URL as before. No need to delete it and set up a new one.

To get your private key, you need to export it from Metamask. Open Metamask, click on the three dots, click on `Account Details` and then `Export Private Key`. MAKE SURE YOU ARE USING A TEST ACCOUNT THAT DOES NOT HAVE MAINNET FUNDS FOR THIS. Add this Private Key below in your `.env` file for `PRIVATE_KEY` variable.

```
QUICKNODE_HTTP_URL="add-quicknode-http-provider-url-here"

PRIVATE_KEY="add-the-private-key-here"
```

Now, let's write a deployment script to automatically deploy both our contracts for us. Create a new file, or replace the existing default one, named `deploy.js` under `hardhat-tutorial/scripts`, and add the following code:

```
const { ethers } = require("hardhat");
const { CRYPTODEVS_NFT_CONTRACT_ADDRESS } = require("../constants");

async function main() {
  // Deploy the FakeNFTMarketplace contract first
  const FakeNFTMarketplace = await ethers.getContractFactory(
    "FakeNFTMarketplace"
  );
  const fakeNftMarketplace = await FakeNFTMarketplace.deploy();
  await fakeNftMarketplace.deployed();

  console.log("FakeNFTMarketplace deployed to: ", fakeNftMarketplace.address);

  // Now deploy the CryptoDevsDAO contract
  const CryptoDevsDAO = await ethers.getContractFactory("CryptoDevsDAO");
  const cryptoDevsDAO = await CryptoDevsDAO.deploy(
    fakeNftMarketplace.address,
    CRYPTODEVS_NFT_CONTRACT_ADDRESS,
    {
      // This assumes your account has at least 1 ETH in it's account
      // Change this value as you want
      value: ethers.utils.parseEther("1"),
    }
  );
  await cryptoDevsDAO.deployed();

  console.log("CryptoDevsDAO deployed to: ", cryptoDevsDAO.address);
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
```

```
    process.exit(1);
  });
```

As you may have noticed, `deploy.js` imports a variable called `CRYPTODEVS_NFT_CONTRACT_ADDRESS` from a file named `constants`. Let's make that. Create a new file named `constants.js` in the `hardhat-tutorial` directory.

```
// Replace the value with your NFT contract address
const CRYPTODEVS_NFT_CONTRACT_ADDRESS =
  "YOUR_CRYPTODEVS_NFT_CONTRACT_ADDRESS_HERE";

module.exports = { CRYPTODEVS_NFT_CONTRACT_ADDRESS };
```

Now, let's add the Goerli Network to your Hardhat Config so we can deploy to Goerli. Open your `hardhat.config.js` file and replace it with the following:

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config({ path: ".env" });

const QUICKNODE_HTTP_URL = process.env.QUICKNODE_HTTP_URL;
const PRIVATE_KEY = process.env.PRIVATE_KEY;

module.exports = {
  solidity: "0.8.4",
  networks: {
    goerli: {
      url: QUICKNODE_HTTP_URL,
      accounts: [PRIVATE_KEY],
    },
  },
};
```

Let's make sure everything compiles before proceeding. Execute the following command from your Terminal within the `hardhat-tutorial` folder.

```
npx hardhat compile
```

and make sure there are no compilation errors. If you do face compilation errors, try comparing your code against the **final version present here**

Let's deploy! Execute the following command in your Terminal from the `hardhat-tutorial` directory

```
npx hardhat run scripts/deploy.js --network goerli
```

- Save the `FakeNFTMarketplace` and `CryptoDevsDAO` contract addresses that get printed in your Terminal. You will need those later.

Frontend Development

Whew! So much coding!

We've successfully developed and deployed our contracts to the Goerli Testnet. Now, it's time to build the Frontend interface so users can create and vote on proposals from the website.

To develop the website, we will be using **Next.js** as we have so far, which is a meta-framework built on top of **React**.

Let's get started by creating a new `next` app. Your folder structure should look like this after setting up the `next` app:

```
- DAO-Tutorial
    - hardhat-tutorial
    - my-app
```

To create `my-app`, execute the following command in your Terminal within the `DAO-Tutorial` directory

```
npx create-next-app@latest
```

and press `Enter` for all the question prompts. This should create the `my-app` folder and setup a basic Next.js project. Let's see if everything works. Run the following in your Terminal

```
cd my-app
npm run dev
```

Your website should be up and running at `http://localhost:3000`. However, this is a basic starter Next.js project and we need to add code for it to do what we want. Let's install the `web3modal` and `ethers` library. Web3Modal will allow us to support connecting to wallets in the browser, and Ethers will be used to interact with the blockchain. Run this in your Terminal from the `my-app` directory.

```
npm install web3modal ethers
```

Download and save the following file as `0.svg` in `my-app/public/cryptodevs`. We will display this image on the webpage. NOTE: You need to create the `cryptodevs` folder inside `public`. **Download Image**

Add the following CSS styles in `my-app/styles/Home.modules.css`

```
.main {
  min-height: 90vh;
  display: flex;
  flex-direction: row;
  justify-content: center;
  align-items: center;
  font-family: "Courier New", Courier, monospace;
}

.footer {
  display: flex;
  padding: 2rem 0;
  border-top: 1px solid #eaeaea;
  justify-content: center;
  align-items: center;
}

.image {
  width: 70%;
  height: 50%;
  margin-left: 20%;
}

.title {
  font-size: 2rem;
  margin: 2rem 0;
}

.description {
  line-height: 1;
  margin: 2rem 0;
  font-size: 1.2rem;
}

.button {
  border-radius: 4px;
  background-color: blue;
  border: none;
  color: #ffffff;
  font-size: 15px;
  padding: 10px;
  width: 200px;
  cursor: pointer;
  margin-right: 2%;
}

.button2 {
  border-radius: 4px;
  background-color: indigo;
  border: none;
  color: #ffffff;
  font-size: 15px;
  padding: 10px;
  cursor: pointer;
  margin-right: 2%;
  margin-top: 1rem;
}

.proposalCard {
```

```
.proposalCard {
  width: fit-content;
  margin-top: 0.25rem;
  border: black 2px solid;
  flex: 1;
  flex-direction: column;
}

.container {
  margin-top: 2rem;
}

.flex {
  flex: 1;
  justify-content: space-between;
}

@media (max-width: 1000px) {
  .main {
    width: 100%;
    flex-direction: column;
    justify-content: center;
    align-items: center;
  }
}
```

The website also needs to read/write data from two smart contracts - `CryptoDevsDAO` and `CryptoDevsNFT`. Let's store their contract addresses and ABIs in a constants file. Create a `constants.js` file in the `my-app` directory.

```
export const CRYPTODEVS_DAO_CONTRACT_ADDRESS = "";
export const CRYPTODEVS_NFT_CONTRACT_ADDRESS = "";

export const CRYPTODEVS_DAO_ABI = [];
export const CRYPTODEVS_NFT_ABI = [];
```

Replace the contract address and ABI values with your relevant contract addresses and ABIs.

Now for the actual cool website code. Open up `my-app/pages/index.js` and write the following code. Explanation of the code can be found in the comments.

```
import { Contract, providers } from "ethers";
import { formatEther } from "ethers/lib/utils";
import Head from "next/head";
import { useEffect, useRef, useState } from "react";
import Web3Modal from "web3modal";
import {
  CRYPTODEVS_DAO_ABI,
  CRYPTODEVS_DAO_CONTRACT_ADDRESS,
  CRYPTODEVS_NFT_ABI,
  CRYPTODEVS_NFT_CONTRACT_ADDRESS,
} from "../constants";
import styles from "../styles/Home.module.css";

export default function Home() {
  // ETH Balance of the DAO contract
  const [treasuryBalance, setTreasuryBalance] = useState("0");
  // Number of proposals created in the DAO
  const [numProposals, setNumProposals] = useState("0");
  // Array of all proposals created in the DAO
  const [proposals, setProposals] = useState([]);
  // User's balance of CryptoDevs NFTs
  const [nftBalance, setNftBalance] = useState(0);
  // Fake NFT Token ID to purchase. Used when creating a proposal.
  const [fakeNftTokenId, setFakeNftTokenId] = useState("");
  // One of "Create Proposal" or "View Proposals"
  const [selectedTab, setSelectedTab] = useState("");
  // True if waiting for a transaction to be mined, false otherwise.
  const [loading, setLoading] = useState(false);
  // True if user has connected their wallet, false otherwise
  const [walletConnected, setWalletConnected] = useState(false);
  const web3ModalRef = useRef();

  // Helper function to connect wallet
  const connectWallet = async () => {
    try {
      await getProviderOrSigner();
      setWalletConnected(true);
    } catch (error) {
      console.error(error);
    }
```

```
};

// Reads the ETH balance of the DAO contract and sets the `treasuryBalance` state variable
const getDAOTreasuryBalance = async () => {
  try {
    const provider = await getProviderOrSigner();
    const balance = await provider.getBalance(
      CRYPTODEVS_DAO_CONTRACT_ADDRESS
    );
    setTreasuryBalance(balance.toString());
  } catch (error) {
    console.error(error);
  }
};

// Reads the number of proposals in the DAO contract and sets the `numProposals` state variable
const getNumProposalsInDAO = async () => {
  try {
    const provider = await getProviderOrSigner();
    const contract = getDaoContractInstance(provider);
    const daoNumProposals = await contract.numProposals();
    setNumProposals(daoNumProposals.toString());
  } catch (error) {
    console.error(error);
  }
};

// Reads the balance of the user's CryptoDevs NFTs and sets the `nftBalance` state variable
const getUserNFTBalance = async () => {
  try {
    const signer = await getProviderOrSigner(true);
    const nftContract = getCryptodevsNFTContractInstance(signer);
    const balance = await nftContract.balanceOf(signer.getAddress());
    setNftBalance(parseInt(balance.toString()));
  } catch (error) {
    console.error(error);
  }
};

// Calls the `createProposal` function in the contract, using the tokenId from `fakeNftTokenId`
const createProposal = async () => {
  try {
    const signer = await getProviderOrSigner(true);
    const daoContract = getDaoContractInstance(signer);
    const txn = await daoContract.createProposal(fakeNftTokenId);
    setLoading(true);
    await txn.wait();
    await getNumProposalsInDAO();
    setLoading(false);
  } catch (error) {
    console.error(error);
    window.alert(error.data.message);
  }
};

// Helper function to fetch and parse one proposal from the DAO contract
// Given the Proposal ID
// and converts the returned data into a Javascript object with values we can use
const fetchProposalById = async (id) => {
  try {
    const provider = await getProviderOrSigner();
    const daoContract = getDaoContractInstance(provider);
    const proposal = await daoContract.proposals(id);
    const parsedProposal = {
      proposalId: id,
      nftTokenId: proposal.nftTokenId.toString(),
      deadline: new Date(parseInt(proposal.deadline.toString()) * 1000),
      yayVotes: proposal.yayVotes.toString(),
      nayVotes: proposal.nayVotes.toString(),
      executed: proposal.executed,
    };
    return parsedProposal;
  } catch (error) {
    console.error(error);
  }
};

// Runs a Loop `numProposals` times to fetch all proposals in the DAO
// and sets the `proposals` state variable
const fetchAllProposals = async () => {
  try {
    const proposals = [];
    for (let i = 0; i < numProposals; i++) {
      const proposal = await fetchProposalById(i);
      proposals.push(proposal);
    }
```

```
    setProposals(proposals);
    return proposals;
  } catch (error) {
    console.error(error);
  }
};

// Calls the `voteOnProposal` function in the contract, using the passed
// proposal ID and Vote
const voteOnProposal = async (proposalId, _vote) => {
  try {
    const signer = await getProviderOrSigner(true);
    const daoContract = getDaoContractInstance(signer);

    let vote = _vote === "YAY" ? 0 : 1;
    const txn = await daoContract.voteOnProposal(proposalId, vote);
    setLoading(true);
    await txn.wait();
    setLoading(false);
    await fetchAllProposals();
  } catch (error) {
    console.error(error);
    window.alert(error.data.message);
  }
};

// Calls the `executeProposal` function in the contract, using
// the passed proposal ID
const executeProposal = async (proposalId) => {
  try {
    const signer = await getProviderOrSigner(true);
    const daoContract = getDaoContractInstance(signer);
    const txn = await daoContract.executeProposal(proposalId);
    setLoading(true);
    await txn.wait();
    setLoading(false);
    await fetchAllProposals();
  } catch (error) {
    console.error(error);
    window.alert(error.data.message);
  }
};

// Helper function to fetch a Provider/Signer instance from Metamask
const getProviderOrSigner = async (needSigner = false) => {
  const provider = await web3ModalRef.current.connect();
  const web3Provider = new providers.Web3Provider(provider);

  const { chainId } = await web3Provider.getNetwork();
  if (chainId !== 5) {
    window.alert("Please switch to the Goerli network!");
    throw new Error("Please switch to the Goerli network");
  }

  if (needSigner) {
    const signer = web3Provider.getSigner();
    return signer;
  }
  return web3Provider;
};

// Helper function to return a DAO Contract instance
// given a Provider/Signer
const getDaoContractInstance = (providerOrSigner) => {
  return new Contract(
    CRYPTODEVS_DAO_CONTRACT_ADDRESS,
    CRYPTODEVS_DAO_ABI,
    providerOrSigner
  );
};

// Helper function to return a CryptoDevs NFT Contract instance
// given a Provider/Signer
const getCryptodevsNFTContractInstance = (providerOrSigner) => {
  return new Contract(
    CRYPTODEVS_NFT_CONTRACT_ADDRESS,
    CRYPTODEVS_NFT_ABI,
    providerOrSigner
  );
};

// piece of code that runs everytime the value of `walletConnected` changes
// so when a wallet connects or disconnects
// Prompts user to connect wallet if not connected
// and then calls helper functions to fetch the
```

```
// DAO Treasury Balance, User NFT Balance, and Number of Proposals in the DAO
useEffect(() => {
  if (!walletConnected) {
    web3ModalRef.current = new Web3Modal({
      network: "goerli",
      providerOptions: {},
      disableInjectedProvider: false,
    });

    connectWallet().then(() => {
      getDAOTreasuryBalance();
      getUserNFTBalance();
      getNumProposalsInDAO();
    });
  }
}, [walletConnected]);

// Piece of code that runs everytime the value of `selectedTab` changes
// Used to re-fetch all proposals in the DAO when user switches
// to the 'View Proposals' tab
useEffect(() => {
  if (selectedTab === "View Proposals") {
    fetchAllProposals();
  }
}, [selectedTab]);

// Render the contents of the appropriate tab based on `selectedTab`
function renderTabs() {
  if (selectedTab === "Create Proposal") {
    return renderCreateProposalTab();
  } else if (selectedTab === "View Proposals") {
    return renderViewProposalsTab();
  }
  return null;
}

// Renders the 'Create Proposal' tab content
function renderCreateProposalTab() {
  if (loading) {
    return (
      <div className={styles.description}>
        Loading... Waiting for transaction...
      </div>
    );
  } else if (nftBalance === 0) {
    return (
      <div className={styles.description}>
        You do not own any CryptoDevs NFTs. <br />
        <b>You cannot create or vote on proposals</b>
      </div>
    );
  } else {
    return (
      <div className={styles.container}>
        <label>Fake NFT Token ID to Purchase: </label>
        <input
          placeholder="0"
          type="number"
          onChange={(e) => setFakeNftTokenId(e.target.value)}
        />
        <button className={styles.button2} onClick={createProposal}>
          Create
        </button>
      </div>
    );
  }
}

// Renders the 'View Proposals' tab content
function renderViewProposalsTab() {
  if (loading) {
    return (
      <div className={styles.description}>
        Loading... Waiting for transaction...
      </div>
    );
  } else if (proposals.length === 0) {
    return (
      <div className={styles.description}>No proposals have been created</div>
    );
  } else {
    return (
      <div>
        {proposals.map((p, index) => (
          <div key={index} className={styles.proposalCard}>
            <p>Proposal ID: {p.proposalId}</p>
```

```
            <p>Fake NFT to Purchase: {p.nftTokenId}</p>
            <p>Deadline: {p.deadline.toLocaleString()}</p>
            <p>Yay Votes: {p.yayVotes}</p>
            <p>Nay Votes: {p.nayVotes}</p>
            <p>Executed?: {p.executed.toString()}</p>
            {p.deadline.getTime() > Date.now() && !p.executed ? (
              <div className={styles.flex}>
                <button
                  className={styles.button2}
                  onClick={() => voteOnProposal(p.proposalId, "YAY")}
                >
                  Vote YAY
                </button>
                <button
                  className={styles.button2}
                  onClick={() => voteOnProposal(p.proposalId, "NAY")}
                >
                  Vote NAY
                </button>
              </div>
            ) : p.deadline.getTime() < Date.now() && !p.executed ? (
              <div className={styles.flex}>
                <button
                  className={styles.button2}
                  onClick={() => executeProposal(p.proposalId)}
                >
                  Execute Proposal{" "}
                  {p.yayVotes > p.nayVotes ? "(YAY)" : "(NAY)"}
                </button>
              </div>
            ) : (
              <div className={styles.description}>Proposal Executed</div>
            )}
          </div>
        ))}
      </div>
    );
  }
}

return (
  <div>
    <Head>
      <title>CryptoDevs DAO</title>
      <meta name="description" content="CryptoDevs DAO" />
      <link rel="icon" href="/favicon.ico" />
    </Head>

    <div className={styles.main}>
      <div>
        <h1 className={styles.title}>Welcome to Crypto Devs!</h1>
        <div className={styles.description}>Welcome to the DAO!</div>
        <div className={styles.description}>
          Your CryptoDevs NFT Balance: {nftBalance}
          <br />
          Treasury Balance: {formatEther(treasuryBalance)} ETH
          <br />
          Total Number of Proposals: {numProposals}
        </div>
        <div className={styles.flex}>
          <button
            className={styles.button}
            onClick={() => setSelectedTab("Create Proposal")}
          >
            Create Proposal
          </button>
          <button
            className={styles.button}
            onClick={() => setSelectedTab("View Proposals")}
          >
            View Proposals
          </button>
        </div>
        {renderTabs()}
      </div>
      <div>
        <img className={styles.image} src="/cryptodevs/0.svg" />
      </div>
    </div>

    <footer className={styles.footer}>
      Made with &#10084; by Crypto Devs
    </footer>
  </div>
);
```

```
}
```

Let's run it! In your terminal, from the `my-app` directory, execute:

```
npm run dev
```

to see your website in action. It should look like the screenshot at the beginning of this tutorial.

Congratulations! Your CryptoDevs DAO website should now be working.

Testing

- Create a couple of proposals
- Try voting `YAY` on one, and `NAY` on the other
- Wait for 5 minutes for their deadlines to pass
- Execute both of them.
- Watch the balance of the DAO Treasury go down by `0.1 ETH` due to the proposal which passed as it bought an NFT upon execution.

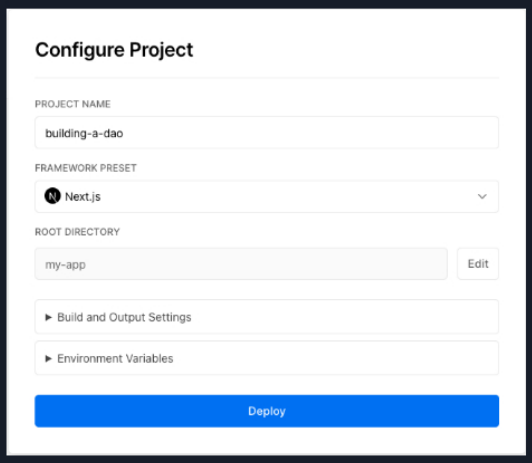
Push to Github

Make sure to push all this code to Github before proceeding to the next step.

Website Deployment

What good is a website if you cannot share it with others? Let's work on deploying your dApp to the world so you can share it with all your LearnWeb3DAO frens.

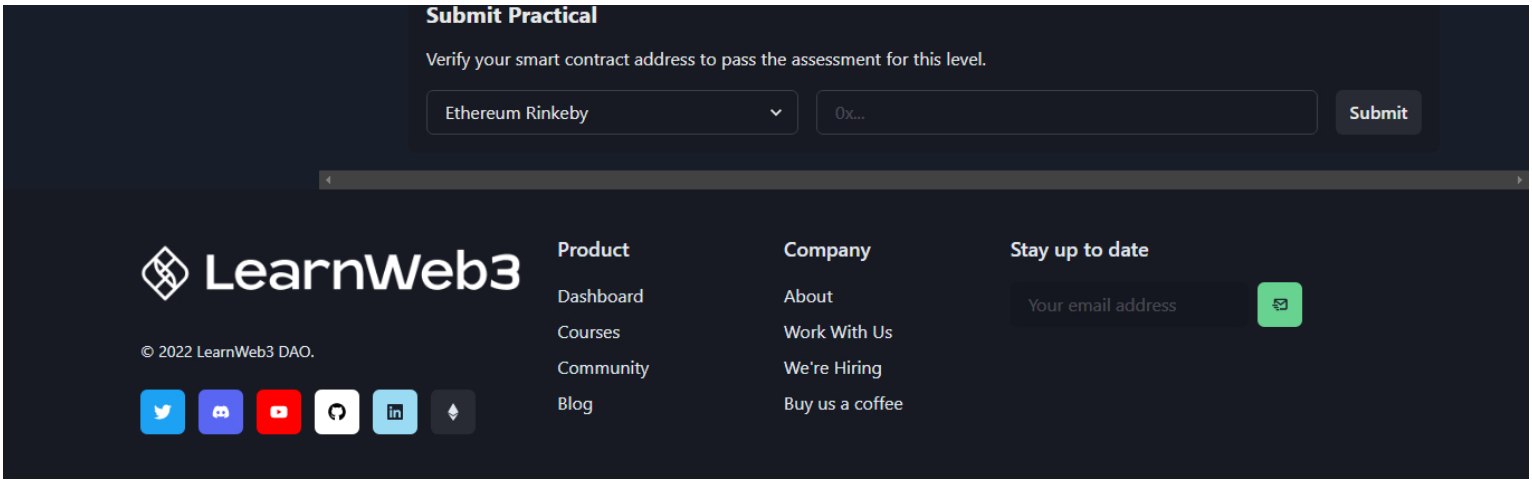
- Go to **Vercel Dashboard** and sign in with your GitHub account.
- Click on the `New Project` button and select your `DAO-Tutorial` repo.
- When configuring your new project, Vercel will allow you to customize your `Root Directory`
- Since our Next.js application is within a subfolder of the repo, we need to modify it.
- Click `Edit` next to `Root Directory` and set it to `my-app`.
- Select the framework as `Next.js`
- Click `Deploy`



- Now you can see your deployed website by going to your Vercel Dashboard, selecting your project, and copying the domain from there!

CONGRATULATIONS! You're all done!

Hopefully you enjoyed this tutorial. Don't forget to share your DAO website in the `#showcase` channel on Discord :D



LearnWeb3

Product

- Dashboard
- Courses
- Community
- Blog

Company

- About
- Work With Us
- We're Hiring
- Buy us a coffee

Stay up to date

Your email address