Documentación Práctica final de módulo Emilio Pérez Arjona

A continuación se realizan anotaciones sobre las diferentes partes en las que se divide la práctica y se añaden capturas de pantalla.

Smart Contract.

Consiste en rellenar el código de una DAO. Se da una plantilla con varias funciones y modifiers y el objetivo es rellenar el contenido de las funciones. Hay comentarios explicando los requisitos de cada función.

En este apartado he utilizado prácticamente el mismo código visto ya en las clases.

Lo novedoso fue el uso de un bucle en la función “createProposal” para rellenar los mappings.

```
//iterar sobre el array de opciones para rellenar los mappings. Pista usar _options.length
for(uint8 i = 0; i < _options.length; i++){
    //inicializar las opciones
    proposal.optionsText[i] = _options[i];
    //inicializar los votos
    proposal.optionsVotes[i] = 0;
}
```

Así como la inicialización de un nuevo objeto “MultipleChoiceProposal” en la función “getProposalInfo”.

```
function getProposalInfo(uint256 _proposalId) doesProposalExist(_proposalId) public view returns(MultipleChoiceProposalInfo memory){
    //obtener la propuesta cuyo id es el parametro recibido
    MultipleChoiceProposal storage selectedProposal = proposals[_proposalId];
    //inicializar el objeto MultipleChoiceProposalInfo. Pista no es necesario usar storage, mejor usar memory
    MultipleChoiceProposalInfo memory proposalInfo;
    //inicializar el array optionsText dentro del struct
    proposalInfo.optionsText = new string[](selectedProposal.optionsNumber);//Explica por que no existia ya cuando se creó la propuesta:
    //inicializar el array optionsVotes dentro del struct
    proposalInfo.optionsVotes = new uint8[](selectedProposal.optionsNumber);

    //iterar sobre las opciones. Pista usar el campo optionsNumber como tope del bucle
    for(uint8 i = 0; i < proposals[_proposalId].optionsNumber; i++){
        //coger el optionsText
        proposalInfo.optionsText[i] = proposals[_proposalId].optionsText[i];
        //coger el optionsVotes
        proposalInfo.optionsVotes[i] = proposals[_proposalId].optionsVotes[i];
    }
    //devolver el objeto MultipleChoiceProposalInfo
    return proposalInfo;
}
```

Se añade un control a “voteProposal” ya que actualmente no se impedía que alguien pudiera votar las veces que quiera aún ya habiendo votado (de esto me dí cuenta más adelante, probando el front, vi que podía clicar las veces que quisiera y se enviaba la votación).

```
function voteProposal(uint256 _proposalId, uint8 _optionCode) public doesProposalExist(_proposalId) isProposalActive(_proposalId){
    // Verificar si la dirección ya votó
    if (hasAddressVoted(_proposalId, msg.sender)) {
    revert("You have already voted on this proposal");
    }
    //sumar el voto a su opcion
    proposals[_proposalId].optionsVotes[_optionCode]++;
    //incluir el address como que ya voto
    proposals[_proposalId].voters[msg.sender] = true;
    //emitir el evento ProposalVoted
    emit ProposalVoted(_proposalId, msg.sender);
}
```

```
eth_estimateGas
  Contract call:       KeepCodingDAO#voteProposal
  From:                0x8626f6940e2eb28930efb4cef49b2d1f2c9c1199
  To:                  0x5fbdb2315678afecb367f032d93f642f64180aa3
  Value:               0 ETH

  Error: reverted with reason string 'You have already voted on this proposal'
```

Test

Están divididos en dos partes. Primero hay unos test ya desarrollados que sirven para comprobar que la lógica implementada en las funciones está desarrollada correctamente. Si ejecutando los tests sale todo bien es que el desarrollo es correcto. Después hay una segunda parte en la que se pide desarrollar tests que comprueben que cuando las condiciones para ejecutar una función no se cumplen salte el error correspondiente. Tal y como hemos hecho en clase.

En esta parte fue de mucha ayuda el código que se aporta ya para los tests ya que, además de la sintaxis para el “deployDAOFixture” también pude fijarme en cómo simular que el tiempo se incrementa a la hora de desarrollar el test “Should revert when the proposal is not active”

```
it("Should revert when the proposal is not active", async function(){
    const {dao} = await loadFixture(deployDAOFixture)
    await time.increase(30 * 60)
    await expect(dao.voteProposal(1, vote)).to.be.revertedWithCustomError(
        dao,
        'ProposalDeadlineExceeded'
    )
})
```

Finalmente, se ejecutan y pasan todos los tests.

```
DAO Test Suite
  Create Proposal
    ✔ Should crete proposal (473ms)
  Get Proposal Info
    ✔ Should get the proposal properly
    ✔ Should revert when the proposal does not exist
  Vote Proposal
    ✔ Should vote the proposal properly
    ✔ Should revert when the proposal does not exist
    ✔ Should revert when the proposal is not active
  Execute Proposal
    ✔ Should execute the proposal properly
    ✔ Should revert when the proposal does not exist
    ✔ Should revert when the proposal cannot be executed


9 passing (495ms)
```

Front

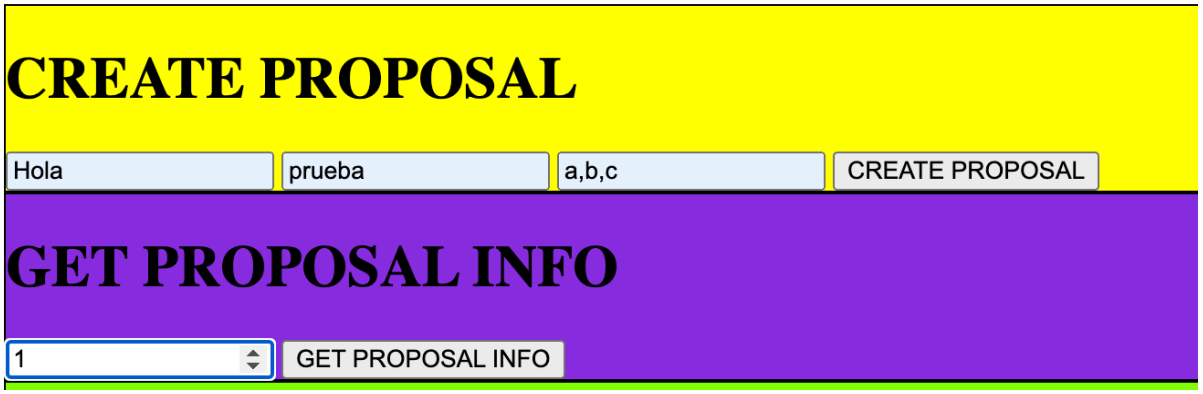
Por último, hay que desarrollar el front. Como en la parte de los smart contract, existe una parte de código ya hecho, el HTML, y hay que rellenar parte del JS para inicializar los contratos y construir las llamadas a la blockchain.

Aquí me quedé algo atascado probando la funcionalidad.

Creación de la proposal, bien.

```
eth_sendRawTransaction
  Contract call:       KeepCodingDAO#createProposal
  Transaction:         0x19ddb2fb114df66acfc7c1cc9264d899498805fa5356292234edb5163a2a8e79
  From:                0x8626f6940e2eb28930efb4cef49b2d1f2c9c1199
  To:                  0x5fbdb2315678afecb367f032d93f642f64180aa3
  Value:               0 ETH
  Gas used:            266732 of 266732
  Block #2:            0x7d6b7df00e5d195550ba39bb7dc356b59de6b7c4bd48b02b57271a14871879c4
```

Traer la info de la proposal, bien, pero en consola JS, “Undefined”




```
getButton.addEventListener("click", async () => {
    const response = await getProposalInfo(
        getInput.value
    )
    console.log(response)
})
```

```
Console was cleared                                    VM1799:1
undefined
undefined                                              app.js:78
>
```

Sin embargo, por la consola del Visual Studio Code, la operación parece que se realiza. Es más, cuando introduzco un Id que no existe, se revierte con el error que hemos planteado en el ejercicio. Y eso sí que sale por consola.

```
eth_chainId
eth_call
  Contract call:       KeepCodingDAO#getProposalInfo
  From:                0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266
  To:                  0x5fbdb2315678afecb367f032d93f642f64180aa3
```

```
▶ Uncaught (in promise) Error: execution reverted:                    ethers.umd.min.js:1
ProposalDoesNotExist(uint256)
    at makeError (ethers.umd.min.js:1:2851)
    at getBuiltinCallException (ethers.umd.min.js:1:227926)
    at AbiCoder.getBuiltinCallException (ethers.umd.min.js:1:229802)
    at Interface.makeError (ethers.umd.min.js:1:240456)
    at staticCallResult (ethers.umd.min.js:1:268772)
    at async staticCall (ethers.umd.min.js:1:267631)
    at async Proxy.getProposalInfo (ethers.umd.min.js:1:269005)
    at async getProposalInfo (app.js:33:5)
    at async HTMLInputElement.<anonymous> (app.js:75:22)
```

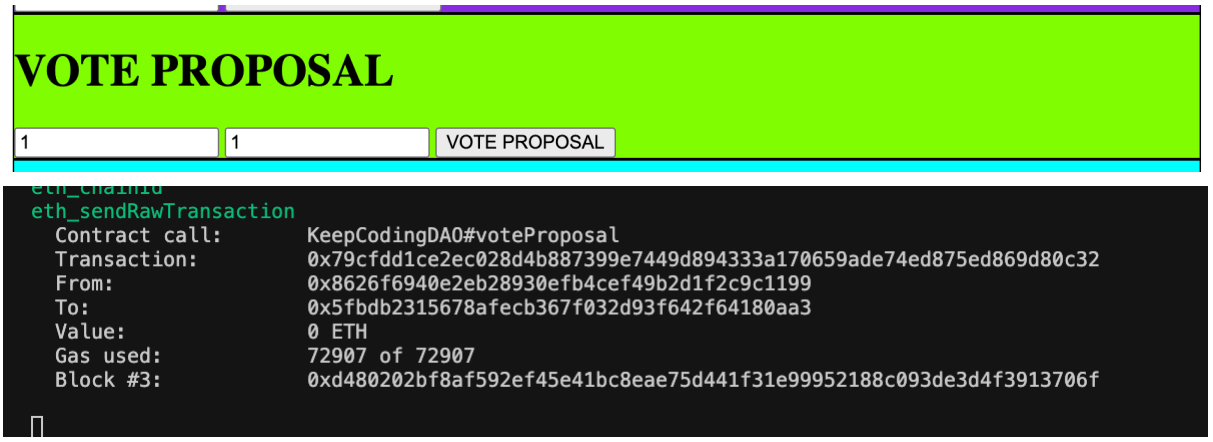
```
eth_chainId
eth_call
  Contract call:       KeepCodingDAO#getProposalInfo
  From:                0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266
  To:                  0x5fbdb2315678afecb367f032d93f642f64180aa3

  Error: VM Exception while processing transaction: reverted with an unrecognized custom error (return data: 0x4f017420000000
0000000000000000000000000000000000000000000000000000000000000d)
```

Metí un log en el contrato de solidity para estar más seguro y corroboré que la función para devolver la info de la proposal funciona correctamente.

```
console.log("Fetching Proposal ID: %s", _proposalId);
console.log("Options Number: %s", selectedProposal.optionsNumber);
for(uint8 i = 0; i < selectedProposal.optionsNumber; i++) {
    console.log("Option %s: %s - Votes: %s", i, selectedProposal.optionsText[i], selectedProposal.optionsVotes[i]);
}
```

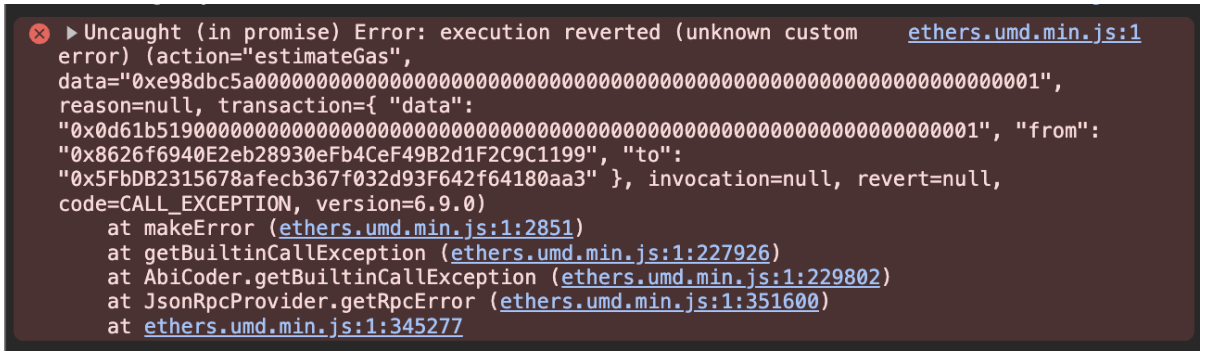
```
eth_call
  Contract call:       KeepCodingDAO#getProposalInfo
  From:                0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266
  To:                  0x5fbdb2315678afecb367f032d93f642f64180aa3

  console.log:
    Fetching Proposal ID: 1
    Options Number: 3
    Option 0: a - Votes: 0
    Option 1: b - Votes: 0
    Option 2: c - Votes: 0
```

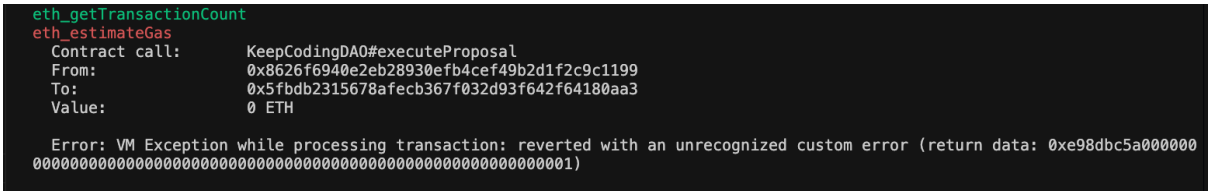
La votación de la propuesta, parece que también funciona bien, ya que por la consola del VSC se ve correcto.



Si haces click repetidamente el control funciona.

La ejecución del contrato, da error tal como se espera, porque aún no pasaron los 30 minutos. Pero sí es cierto, que el error que da es “unrecognized” cuando debería ser del tipo “ProposalCannotBeExecuted”.





De todas formas, en los tests, pasaba la prueba cuando aumentamos el tiempo. Aquí da error, solo que no el que nosotros esperamos.

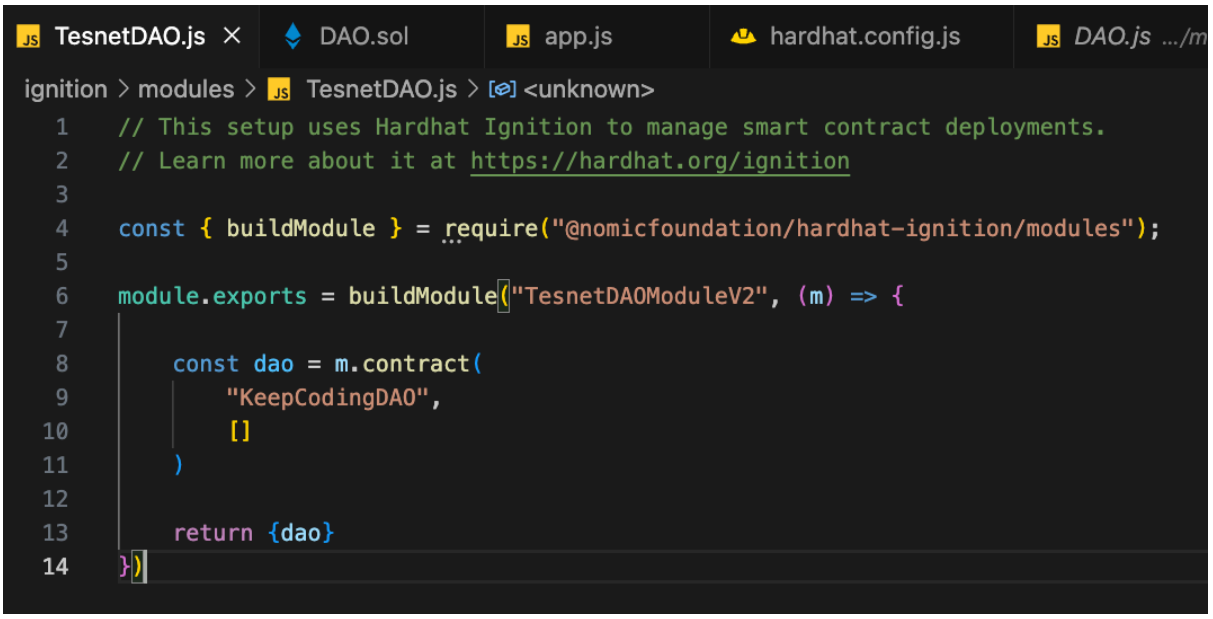
Con esto, la práctica estaría completada.

Extra, despliegue en Amoy tesnet.

Modificar el hardhat.config para añadir la network Amoy (punto de acceso y clave privada)

```
require("@nomicfoundation/hardhat-toolbox");
require('@nomicfoundation/hardhat-verify');

const AMOY_ENDPOINT = "https://polygon-amoy.g.alchemy.com/v2/_3UvND3xyxsHKYHV_zZ6MCJCYMsPiUSP"
const PRIVATE_KEY = "AQUÍ TU CLAVE PRIVADA PARA TESNET"
const POLYGONSACN_APIKEY = "AQUÍ TU API KEY"

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.28",
  networks:{
      amoy:{
        url: AMOY_ENDPOINT,
        accounts: [PRIVATE_KEY]
      },
  },
  etherscan: {
    apiKey: POLYGONSACN_APIKEY
  }
};
```

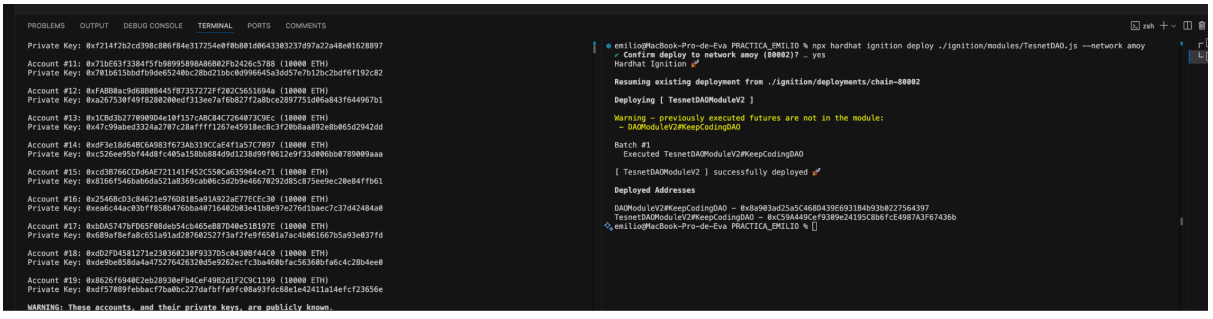
Creación de un nuevo módulo de despliegue (para no modificar el de la práctica), llamado TesnetDAO.js



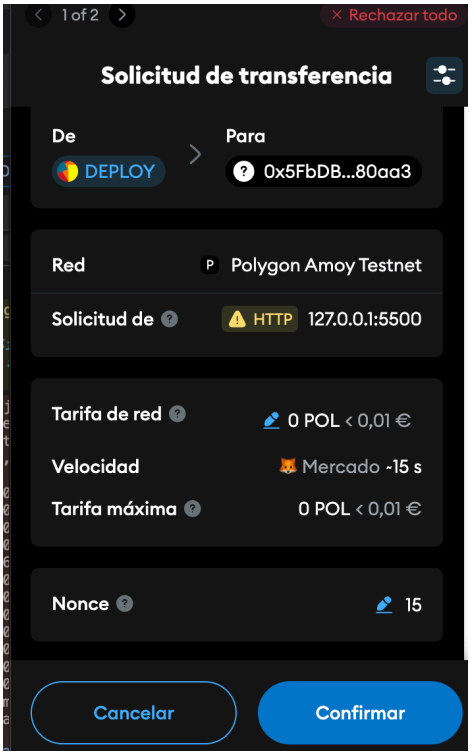
Modificar el Front en app.js (Dejo comentadas las líneas 40 y 47)

```
const HARDHAT_URL = "http://127.0.0.1:8545";
const DAO_ADDRESS_LOCALHOST = "0x5fbdb2315678afecb367f032d93f642f64180aa3"
const DAO_ADDRESS_TESTNET = ""
const SIGNER_PRIVATEKEY = "0xdf57089febbacf7ba0bc227dafbffa9fc08a93fdc68e1e42411a14efcf23656e"

const initializeMetamask = async () => {
    let signer = null;

    let provider;
    if (window.ethereum == null) {

        // If MetaMask is not installed, we use the default provider,
        // which is backed by a variety of third-party services (such
        // as INFURA). They do not have private keys installed,
        // so they only have read-only access
        console.log("MetaMask not installed; using read-only defaults")
        provider = ethers.getDefaultProvider()

    } else {

        // Connect to the MetaMask EIP-1193 object. This is a standard
        // protocol that allows Ethers access to make all read-only
        // requests through MetaMask.
        provider = new ethers.BrowserProvider(window.ethereum)

        // It also provides an opportunity to request access to write
        // operations, which will be performed by the private key
        // that MetaMask manages for the user.
        signer = await provider.getSigner();

        return {provider, signer}
    }
}

const initializeContract = async () => {
    const response = await fetch('../../artifacts/contracts/DAO.sol/KeepCodingDAO.json')
    const data = await response.json()

    //provider
    //const provider = new ethers.JsonRpcProvider(HARDHAT_URL)

    const {provider, signer} = await  initializeMetamask() //Alternativa para conexión con Metamask, utilizado para la tesnet

    //read contract
    const readContract = new ethers.Contract(DAO_ADDRESS_LOCALHOST, data.abi, provider)
    //signer
    //const signer = new ethers.Wallet(SIGNER_PRIVATEKEY, provider)

    //write contract
    const writeContract = readContract.connect(signer)
    //return provider, signer, write contract, read contract
    return {provider, signer, writeContract, readContract}
}
```

Despliegue en la tesnet



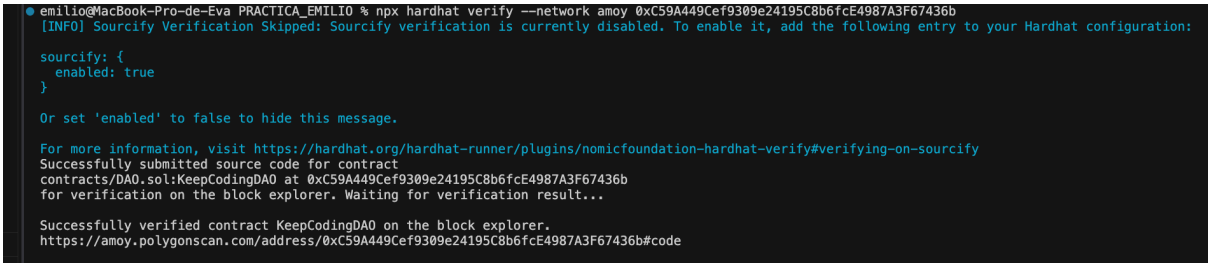
Creación de la proposal



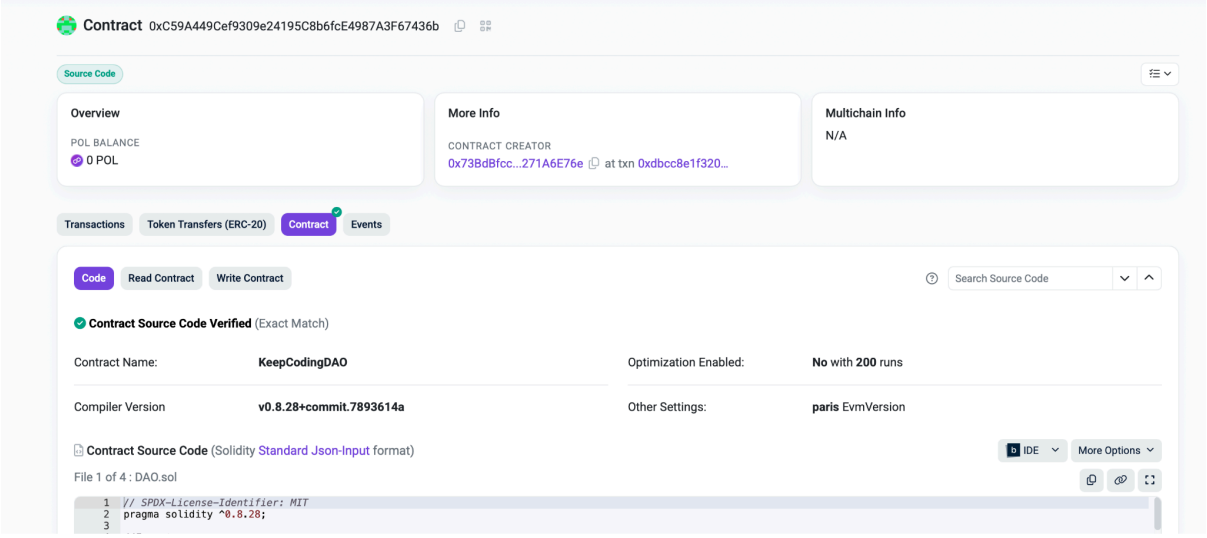
Se ha podido desplegar y ejecutar una función del contrato sin problemas.

También he intentado verificar el contrato

Para la verificación usé la API KEY que nos dejaste en las clases, la he borrado del repo para que no suba a git.



Contrato verificado en polygonscan



El repo está subido a git sin claves privadas (tan solo la clave privada que usas tú, del nodo simulado).