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

Building Dapp: Under the Hood

Lecture 10 (2023-04-05)

Jason Han, Ph.D

Adjunct Professor of KAIST School of Computing

Founder of Ground X & Klaytn

web3classdao@gmail.com

http://web3classdao.xyz/kaist/

Token Example
with hardhat, ethers.js & react

Two popular tools to develop contracts
truffle vs. hardhat

truffle

- The oldest tool, developed in 2016 by ConsenSys
- Comprehensive documentation and resources to learn

hardhat

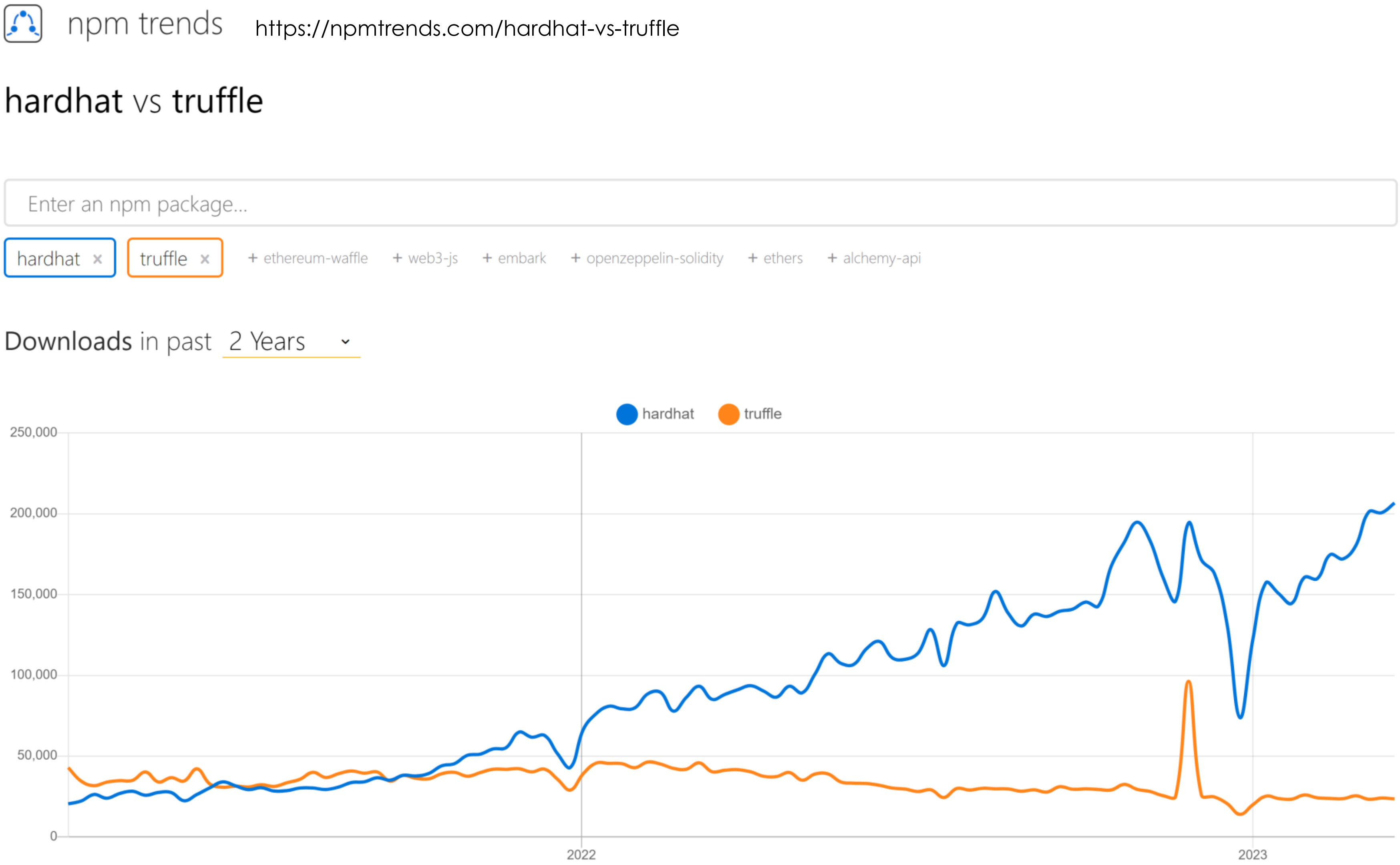
- A new tool released in 2019 by the Nomic Foundation, and is supported by the Ethereum Foundation

hardhat over truffle

- error handling and testing tools like console.log and stack traces
- easier default local network with Hardhat Network
- more flexible to customize with plugins and tasks
- use ethers.js as the default JavaScript library (more user-friendly and easier to use)
- easily integrate TypeScript into hardhat
- easier to fork the blockchain using Alchemy and Infura

https://guideofdapp.com/posts/truffle-vs-hardhat/
https://101blockchains.com/hardhat-vs-truffle/
https://trufflesuite.com/blog/truffle-vs-hardhat-breaking-down-the-difference-between-ethereums-top-development-environments/

https://npmtrends.com/hardhat-vs-truffle



npm trends

hardhat vs truffle

Enter an npm package...

hardhat × truffle × + ethereum-waffle + web3-js + embark + openzeppelin-solidity + ethers + alchemy-api

Downloads in past 2 Years

A simple token Dapp
from hardhat tutorial

https://hardhat.org/tutorial

It’s simple, well-documented, and comprehensive
It can be used as the starting point for your Ethereum project

Clone the code here!
https://github.com/NomicFoundation/hardhat-boilerplate
git clone https://github.com/NomicFoundation/hardhat-boilerplate.git

Toolsets that we will use

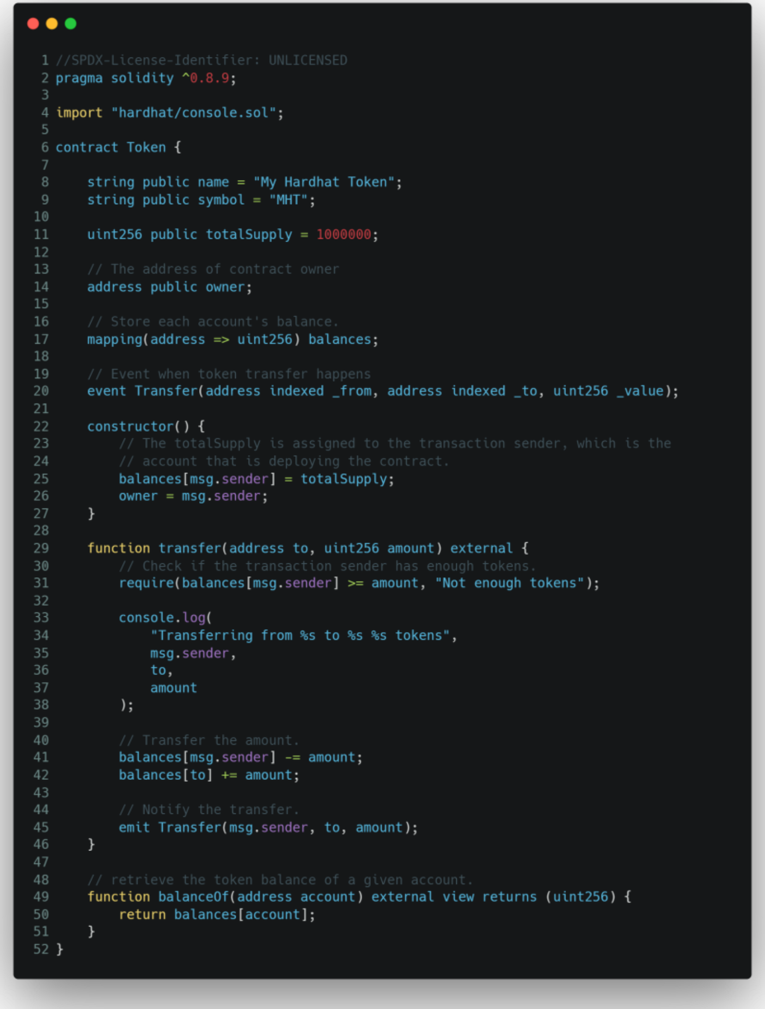
1. Package manager: npm
2. Web server for the web app: node.js & react
3. Smart contract IDE: hardhat & ethers.js
4. Web browser & wallet: Chrome & Metamask
5. Local testnet: Hardhat Network
6. Public testnet: Sepolia
7. Code Editor: VSCode

1. Design (Problem Statement)

Problem Statement

- There is a fixed total supply of tokens that can't be changed.
- The entire supply is assigned to the address that deploys the contract.
- Anyone can receive tokens.
- Anyone with at least one token can transfer tokens.
- The token is non-divisible. You can transfer 1, 2, 3 or 37 tokens but not 2.5.
- It's not compatible to ERC20.

2. Develop smart contract w/ Remix

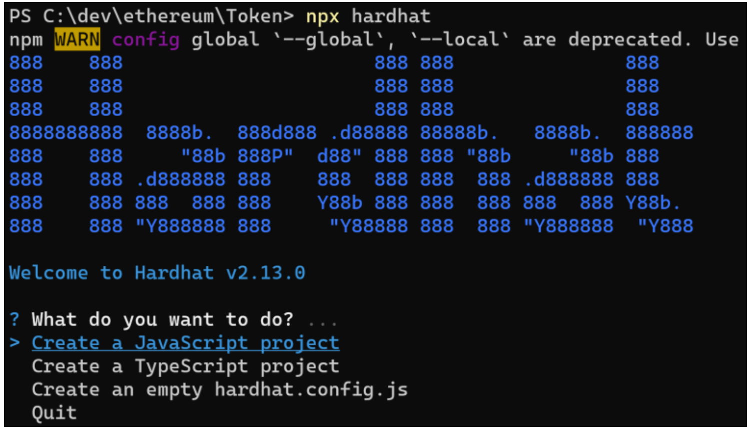


```
//SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

import "hardhat/console.sol";

contract Token {

    string public name = "My Hardhat Token";
    string public symbol = "MHT";

    uint256 public totalSupply = 1000000;

    // The address of contract owner
    address public owner;

    // Store each account's balance.
    mapping(address => uint256) balances;

    // Event when token transfer happens
    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    constructor() {
        // The totalSupply is assigned to the transaction sender, which is the
        // account that is deploying the contract.
        balances[msg.sender] = totalSupply;
        owner = msg.sender;
    }

    function transfer(address to, uint256 amount) external {
        // Check if the transaction sender has enough tokens.
        require(balances[msg.sender] >= amount, "Not enough tokens");

        console.log(
            "Transferring from %s to %s %s tokens",
            msg.sender,
            to,
            amount
        );

        // Transfer the amount.
        balances[msg.sender] -= amount;
        balances[to] += amount;

        // Notify the transfer.
        emit Transfer(msg.sender, to, amount);
    }

    // retrieve the token balance of a given account.
    function balanceOf(address account) external view returns (uint256) {
        return balances[account];
    }
}
```

3. Deploy & test smart contract (Local)

1) install required packages

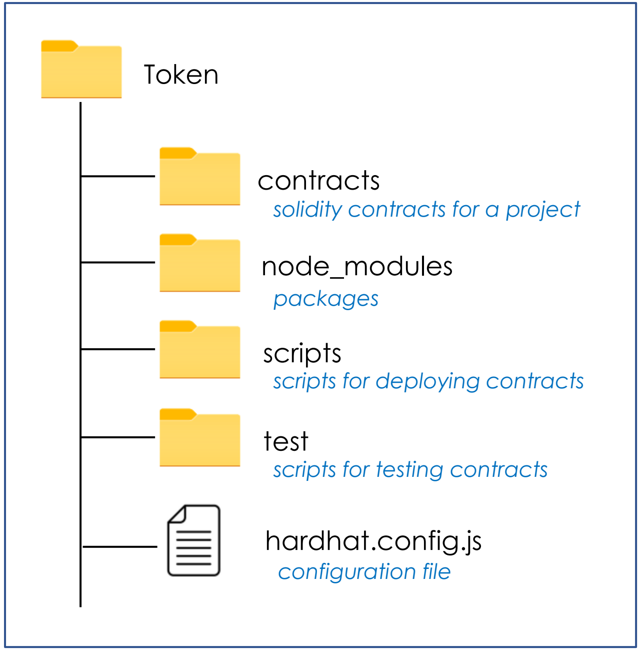
```
mkdir Token
cd Token
npm init
npm install --save-dev hardhat
npx hardhat
```



```
PS C:\dev\ethereum\Token> npx hardhat
npm WARN config global `--global`, `--local` are deprecated. Use
888    888                      888 888               888
888    888                      888 888               888
888    888                      888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888    888     "88b 888P"  d88" 888 888 "88b     "88b 888
888    888 .d888888 888    888  888 888  888 .d888888 888
888    888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888    888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

Welcome to Hardhat v2.13.0

? What do you want to do? ...
> Create a JavaScript project
  Create a TypeScript project
  Create an empty hardhat.config.js
  Quit
```

```
npm install --save-dev @nomicfoundation/hardhat-toolbox
```

Generated by npx hardhat



Token

contracts
solidity contracts for a project

node_modules
packages

scripts
scripts for deploying contracts

test
scripts for testing contracts

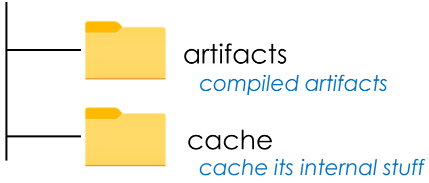
hardhat.config.js
configuration file

3. Deploy & test smart contract (Local)

2) create Token.sol to the contracts folder

3) compile the contract

```
npx hardhat compile
```



artifacts

compiled artifacts

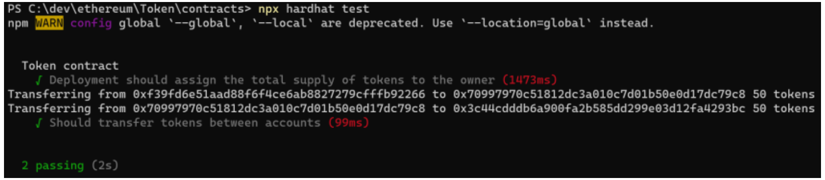
cache

cache its internal stuff

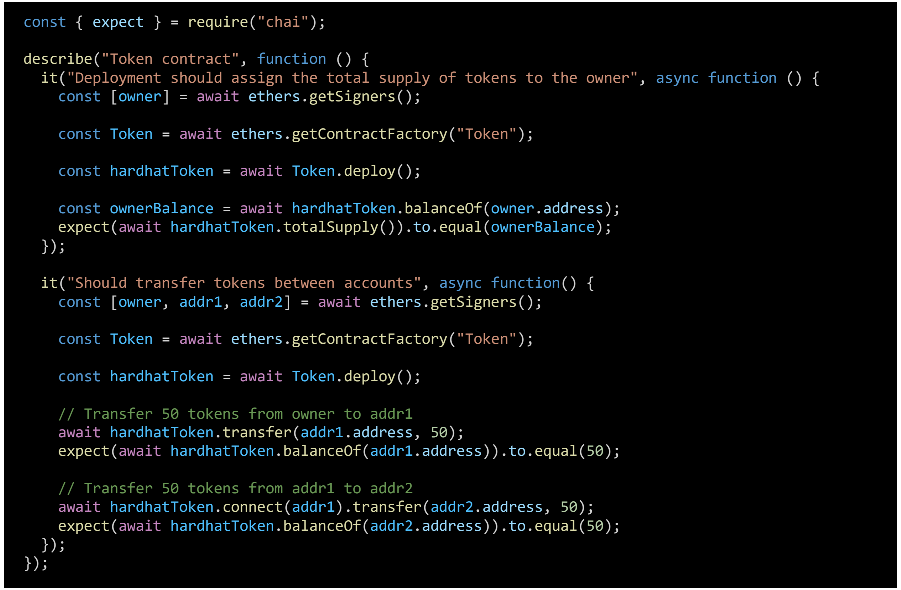
4) create a test script (Token.js) to the test folder → *Token.js*

5) test the contract

```
npx hardhat test
```



```
PS C:\dev\ethereum\Token\contracts> npx hardhat test
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.


  Token contract
    √ Deployment should assign the total supply of tokens to the owner (1473ms)
Transferring from 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266 to 0x70997970c51812dc3a010c7d01b50e0d17dc79c8 50 tokens
Transferring from 0x70997970c51812dc3a010c7d01b50e0d17dc79c8 to 0x3c44cdddb6a900fa2b585dd299e03d12fa4293bc 50 tokens
    √ Should transfer tokens between accounts (99ms)


  2 passing (2s)
```



```
const { expect } = require("chai");

describe("Token contract", function () {
  it("Deployment should assign the total supply of tokens to the owner", async function () {
    const [owner] = await ethers.getSigners();

    const Token = await ethers.getContractFactory("Token");

    const hardhatToken = await Token.deploy();

    const ownerBalance = await hardhatToken.balanceOf(owner.address);
    expect(await hardhatToken.totalSupply()).to.equal(ownerBalance);
  });

  it("Should transfer tokens between accounts", async function() {
    const [owner, addr1, addr2] = await ethers.getSigners();

    const Token = await ethers.getContractFactory("Token");

    const hardhatToken = await Token.deploy();

    // Transfer 50 tokens from owner to addr1
    await hardhatToken.transfer(addr1.address, 50);
    expect(await hardhatToken.balanceOf(addr1.address)).to.equal(50);

    // Transfer 50 tokens from addr1 to addr2
    await hardhatToken.connect(addr1).transfer(addr2.address, 50);
    expect(await hardhatToken.balanceOf(addr2.address)).to.equal(50);
  });
});
```

4. Develop Web App

1) create a template directory for a web app

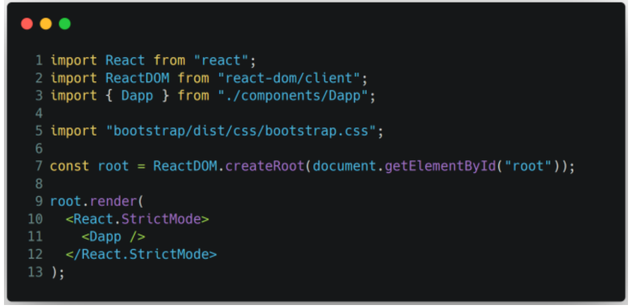
```
npx create-react-app frontend
```

frontend

web app files

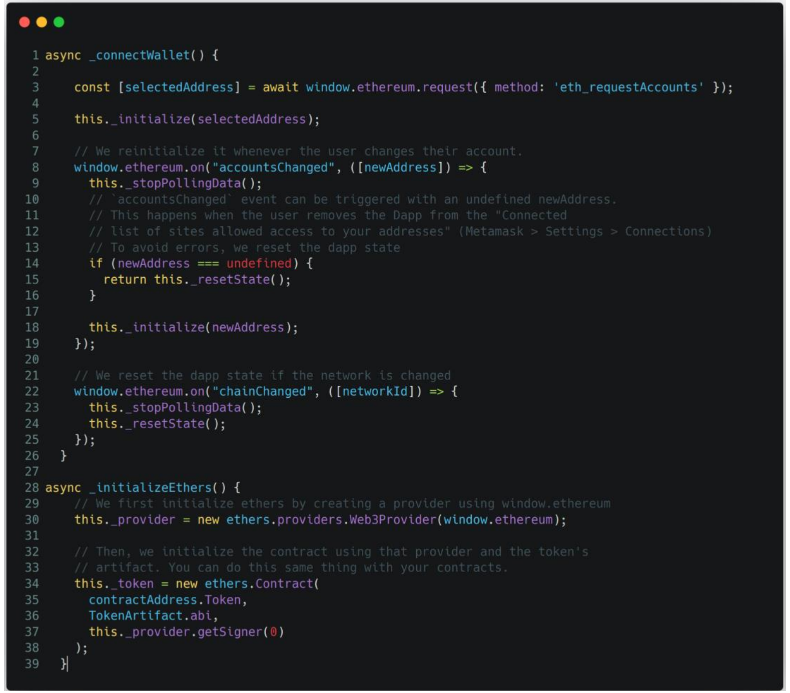
2) write web UI and web app

frontend/src/index.js



```
import React from "react";
import ReactDOM from "react-dom/client";
import { Dapp } from "./components/Dapp";

import "bootstrap/dist/css/bootstrap.css";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(
  <React.StrictMode>
    <Dapp />
  </React.StrictMode>
);
```

frontend/src/components/Dapp.js



```
async _connectWallet() {

    const [selectedAddress] = await window.ethereum.request({ method: 'eth_requestAccounts' });

    this._initialize(selectedAddress);

    // We reinitialize it whenever the user changes their account.
    window.ethereum.on("accountsChanged", ([newAddress]) => {
      this._stopPollingData();
      // `accountsChanged` event can be triggered with an undefined newAddress.
      // This happens when the user removes the Dapp from the "Connected
      // list of sites allowed access to your addresses" (Metamask > Settings > Connections)
      // To avoid errors, we reset the dapp state
      if (newAddress === undefined) {
        return this._resetState();
      }

      this._initialize(newAddress);
    });

    // We reset the dapp state if the network is changed
    window.ethereum.on("chainChanged", ([networkId]) => {
      this._stopPollingData();
      this._resetState();
    });
  }

async _initializeEthers() {
    // We first initialize ethers by creating a provider using window.ethereum
    this._provider = new ethers.providers.Web3Provider(window.ethereum);

    // Then, we initialize the contract using that provider and the token's
    // artifact. You can do this same thing with your contracts.
    this._token = new ethers.Contract(
      contractAddress.Token,
      TokenArtifact.abi,
      this._provider.getSigner(0)
    );
  }
```

5. Deploy & test all (Local)

1) run Hardhat Network (local blockchain)

```
cd Token
npm install
npx hardhat node
```

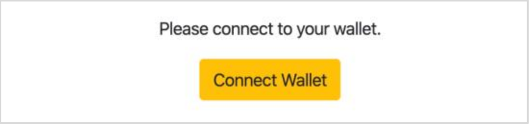
2) deploy contracts to Hardhat Network

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

3) start the react web app

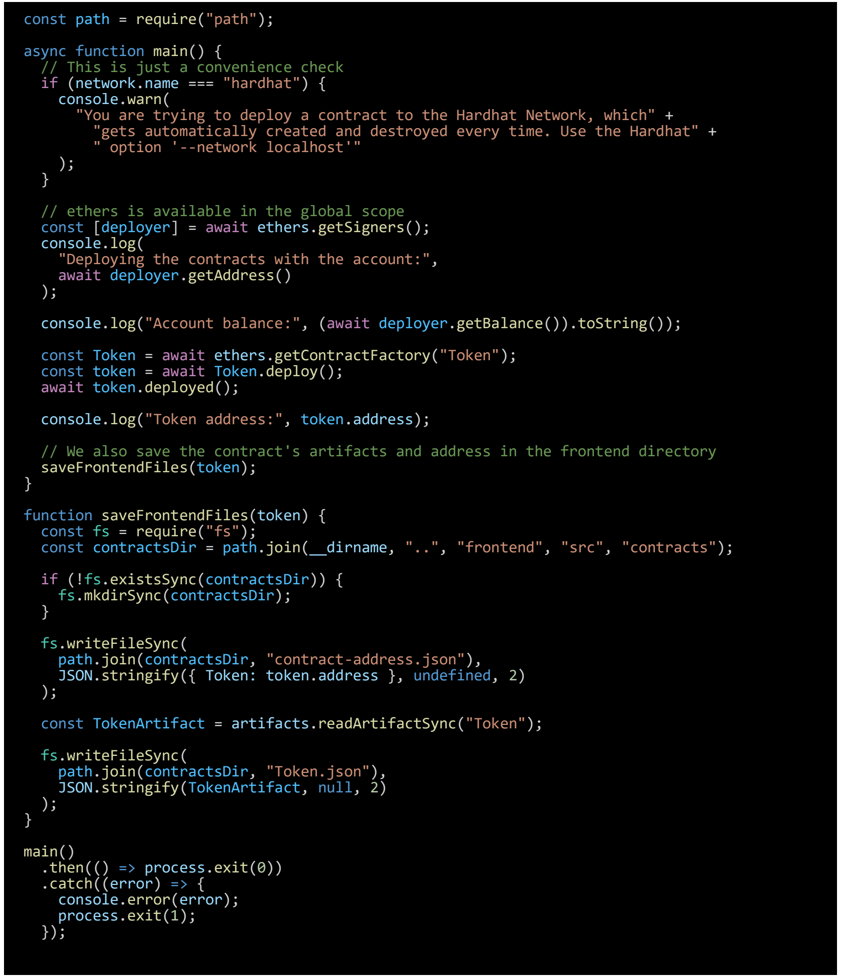
```
cd frontend
npm install
npm run start
```

4) open http://127.0.0.1:3000/ in a browser



Please connect to your wallet.

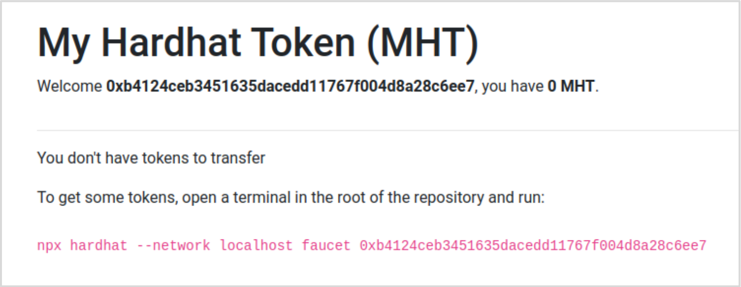
scripts/deploy.js



```
const path = require("path");

async function main() {
  // This is just a convenience check
  if (network.name === "hardhat") {
    console.warn(
      "You are trying to deploy a contract to the Hardhat Network, which" +
        "gets automatically created and destroyed every time. Use the Hardhat" +
        " option '--network localhost'"
    );
  }

  // ethers is available in the global scope
  const [deployer] = await ethers.getSigners();
  console.log(
    "Deploying the contracts with the account:",
    await deployer.getAddress()
  );

  console.log("Account balance:", (await deployer.getBalance()).toString());

  const Token = await ethers.getContractFactory("Token");
  const token = await Token.deploy();
  await token.deployed();

  console.log("Token address:", token.address);

  // We also save the contract's artifacts and address in the frontend directory
  saveFrontendFiles(token);
}

function saveFrontendFiles(token) {
  const fs = require("fs");
  const contractsDir = path.join(__dirname, "..", "frontend", "src", "contracts");

  if (!fs.existsSync(contractsDir)) {
    fs.mkdirSync(contractsDir);
  }

  fs.writeFileSync(
    path.join(contractsDir, "contract-address.json"),
    JSON.stringify({ Token: token.address }, undefined, 2)
  );

  const TokenArtifact = artifacts.readArtifactSync("Token");

  fs.writeFileSync(
    path.join(contractsDir, "Token.json"),
    JSON.stringify(TokenArtifact, null, 2)
  );
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

5. Deploy & test all (Local)

5) set your network in MetaMask to Localhost:8545
click "connect wallet"



My Hardhat Token (MHT)

Welcome **0xb4124ceb3451635dacedd11767f004d8a28c6ee7**, you have **0 MHT**.

You don't have tokens to transfer

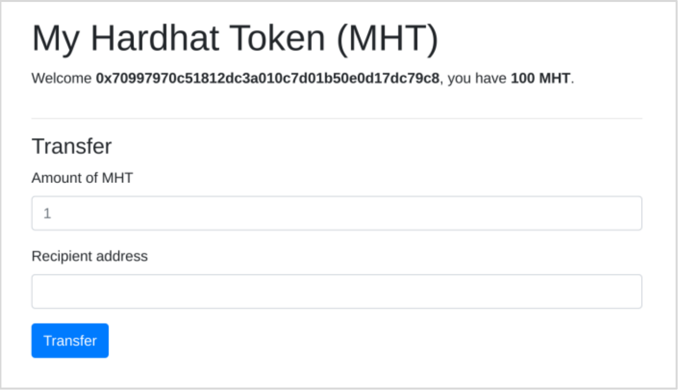
To get some tokens, open a terminal in the root of the repository and run:

```
npx hardhat --network localhost faucet 0xb4124ceb3451635dacedd11767f004d8a28c6ee7
```

6) create a custom hardhat task (tasks/faucet.js)
to send 100 MHT and 1 ETH to an address

7) run the faucet task

```
npx hardhat --network localhost faucet <address>
```



My Hardhat Token (MHT)

Welcome **0x70997970c51812dc3a010c7d01b50e0d17dc79c8**, you have **100 MHT**.

Transfer

Amount of MHT: 1

Recipient address:

Video-05
Deploying and testing contracts with Hardhat locally

6. Deploy & test all (Testnet)

Leave it as your challenge!

Using the previous example as a guide, give it a try

ethers.js

Two most popular Ethereum Javascript libraries

web3.js

- Original Ethereum JavaScript API library
- Launched in 2015 by the Ethereum Foundation
- LGPL-3.0 license

Pros

- Extremely Popular
- Easier to find tutorials, developers, community support, etc

Cons

- A few MBs, significantly larger than ethers

ethers.js

- Original Ethereum JavaScript API library
- Launched in 2015 by a Canadian software engineer named Richard Moore
- A lightweight alternative to Web3.js
- MIT license

Pros

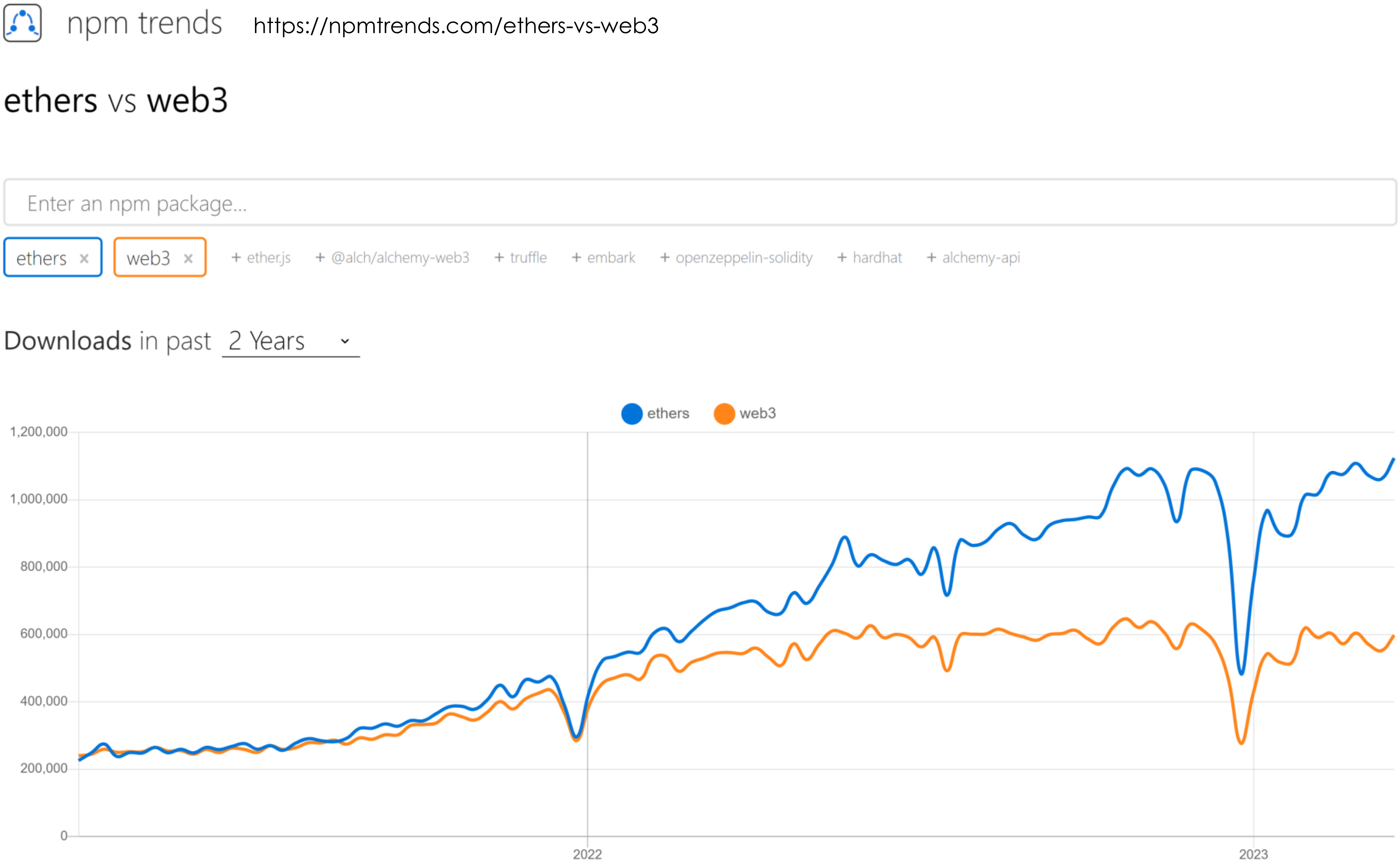
- Separating the wallet and the provider
- Extremely lightweight library, 77 KB compressed and 284 KB uncompressed
- User-friendly API structure

Cons

- Relatively new library, lack of foundational projects and companies

Our choice

https://docs.alchemy.com/docs/ethersjs-vs-web3js-sdk-comparison
https://guideofdapp.com/posts/ethers-vs-web3/

 https://npmtrends.com/ethers-vs-web3

npm trends

ethers vs web3

Enter an npm package...

ethers × web3 × + ether.js + @alch/alchemy-web3 + truffle + embark + openzeppelin-solidity + hardhat + alchemy-api

Downloads in past 2 Years ⌄

Account-related classes of ethers.js

Signer: an Ethereum account that allows for transactions to be signed

- Wrap all operations that interact with an account
- An account generally has a private key located somewhere
- Abstract and cannot be directly instantiated. Instead, use one of sub-classes, such as the Wallet, VoidSigner or JsonRpcSigner
- All important properties of a Signer are immutable
- getAddress( ), getBalance( ), signTransaction( ), sendTransaction( )

Wallet: Sub-class of signer as a standard Externally Owned Account (EOA)

- Sign transactions and messages using a private key
- new ethers.Wallet(privateKey): create a new Wallet instance
- ethers.Wallet.createRandom(): return a new Wallet with a random private key
- ethers.Wallet.fromMnemonic(mnemonic): create an instance from a mnemonic phrase

https://docs.ethers.org/v6/getting-started/
https://docs.ethers.org/v5/api/signer/
https://docs.alchemy.com/docs/ethers-js-signer

Contract-related classes of ethers.js

Contract: an abstraction of a contract deployed on the Ethereum

- contract.attach(): retrieve a new instance of a contract associated with an address
- contract.address(): retrieve the contract or ensName that created the contract
- ontract.queryFilter(): retrieve events that match a specific event

ContractFactory: a factory class to deploy a contract

- Sends a special type of transaction, an initcode transaction (i.e. the *to* field is null, and the *data* field is the initcode)
- new ethers.ContractFactory(interface , bytecode [, signer]): create a new instance of a ContractFactory for the contract
- contractFactory.attach(address): get an instance of a Contract attached to address
- contractFactory.deploy(...args): deploy the Contract with args

https://docs.ethers.org/v6/getting-started/
https://docs.ethers.org/v5/api/contract/
https://docs.alchemy.com/docs/ethers-js-contract

Blockchain-related classes of ethers.js

Provider: a read-only connection to the blockchain

- Abstraction to the Ethereum Network that allows developers to connect to a standard Ethereum node
- provider.getBalances() to retrieve the balances from specific addresses
- provider.getGasPrice() to retrieve the gas price for a transaction that is displayed to a user
- provider.call() to read from the blockchain and execute smart contracts, but cannot publish to the blockchain
- provider.getTransaction() to retrieve the transaction hash to confirm the completion of an execution by the user

Various sub-classes to implement the Provider class

- **DefaultProvider**: the safest, easiest way to begin developing on Ethereum
- **JsonRpcProvider**: a popular method for interacting with Ethereum and is available in all major Ethereum node implementations
- **Web3Provider**: an EIP-1193 Provider or an existing Web3Provider-compatible Provider, moving from a web3.js based application to ethers
- **API Providers**: providers from third-party services, InfuraProvider, AlchemyProvider, EtherscanProvider, etc. (Not recommended to use in order to mitigate the reliance on third-parties)

https://docs.ethers.org/v6/getting-started/
https://docs.ethers.org/v5/api/providers/
https://docs.alchemy.com/docs/ethers-js-provider

hardhat-ethers plugin

Interact with the Ethereum blockchain in a simple way
The same API as ethers.js with some extra Hardhat-specific functionality

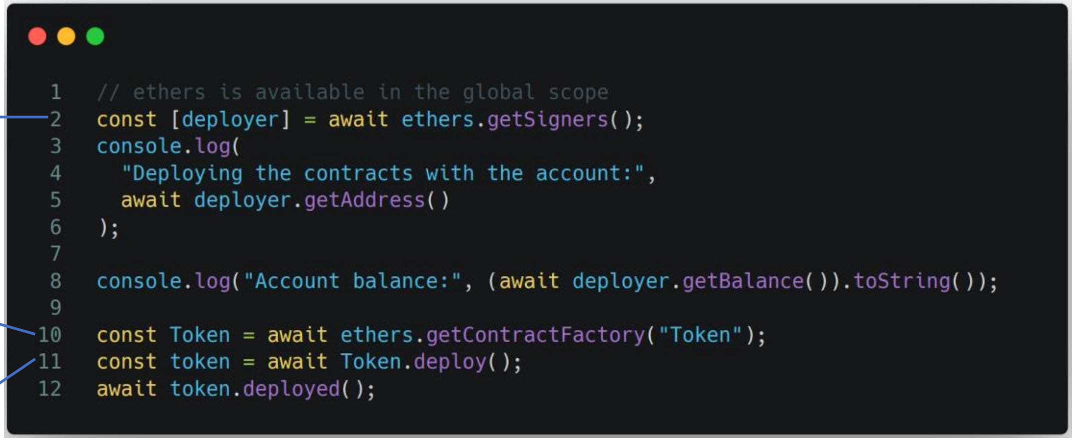
Helper functions

- getContractFactory(): return a new ContractFactory instance
- getContractAt(): return a new Contract instance
- getSigners(): return Signers (accounts) in the network
- getSigner(address): return a Signer (account) of the address

https://hardhat.org/hardhat-runner/plugins/nomiclabs-hardhat-ethers

Deploy the smart contract

scripts/deploy.js



```
// ethers is available in the global scope
const [deployer] = await ethers.getSigners();
console.log(
  "Deploying the contracts with the account:",
  await deployer.getAddress()
);

console.log("Account balance:", (await deployer.getBalance()).toString());

const Token = await ethers.getContractFactory("Token");
const token = await Token.deploy();
await token.deployed();
```

deployer: Signer
the first account of network

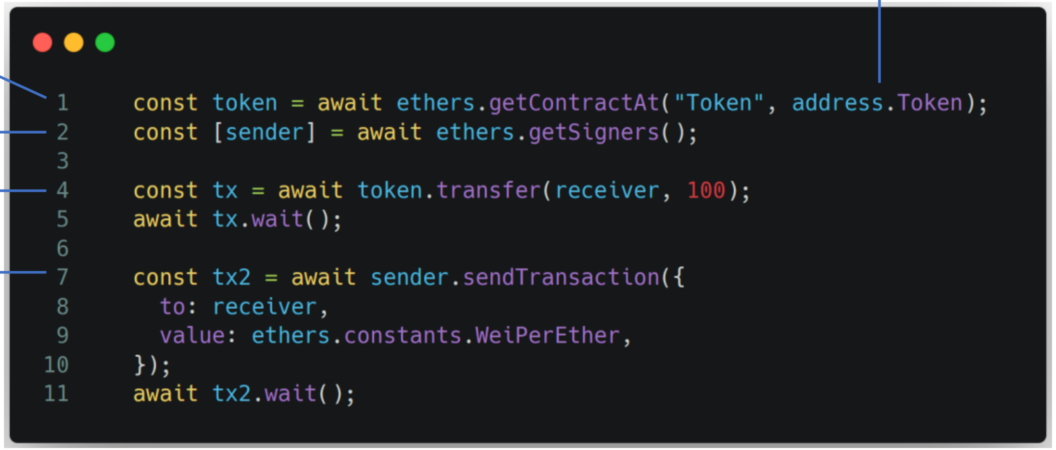
Token: ContractFactory
ContractFactory for Token contract

token: Contract
Token contract

Send initial tokens to an account (faucet)

contract address

tasks/faucet.js



```javascript
const token = await ethers.getContractAt("Token", address.Token);
const [sender] = await ethers.getSigners();

const tx = await token.transfer(receiver, 100);
await tx.wait();

const tx2 = await sender.sendTransaction({
  to: receiver,
  value: ethers.constants.WeiPerEther,
});
await tx2.wait();
```

token: Contract
Token contract

sender: Signer
the first account of network

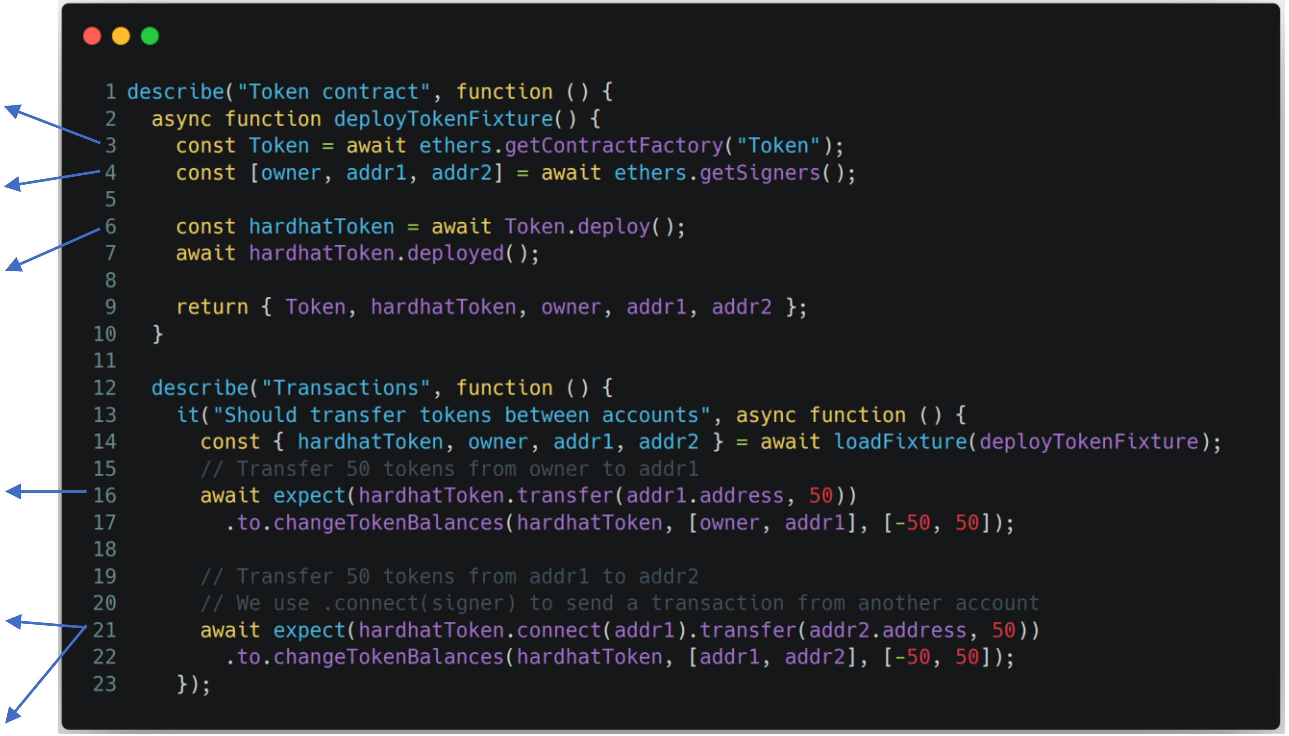
call the transfer function of Token

send tx
(send 1 ETH to receiver)

```
npx hardhat --network localhost faucet <address>
```

Test the smart contract

test/Token.js



```javascript
describe("Token contract", function () {
  async function deployTokenFixture() {
    const Token = await ethers.getContractFactory("Token");
    const [owner, addr1, addr2] = await ethers.getSigners();

    const hardhatToken = await Token.deploy();
    await hardhatToken.deployed();

    return { Token, hardhatToken, owner, addr1, addr2 };
  }

  describe("Transactions", function () {
    it("Should transfer tokens between accounts", async function () {
      const { hardhatToken, owner, addr1, addr2 } = await loadFixture(deployTokenFixture);
      // Transfer 50 tokens from owner to addr1
      await expect(hardhatToken.transfer(addr1.address, 50))
        .to.changeTokenBalances(hardhatToken, [owner, addr1], [-50, 50]);

      // Transfer 50 tokens from addr1 to addr2
      // We use .connect(signer) to send a transaction from another account
      await expect(hardhatToken.connect(addr1).transfer(addr2.address, 50))
        .to.changeTokenBalances(hardhatToken, [addr1, addr2], [-50, 50]);
    });
```

Token: ContractFactory
ContractFactory for Token contract

owner, addr1, addr2: Signer
the 1st, 2nd, 3rd account of network

hardhatToken: Contract
Token contract

call the transfer function of Token from owner account

new instance of Token contract for 2nd account (connect(addr1))

call the transfer function of Token from 2nd account

Interact with MetaMask in Web App

frontend/src/components/Dapp.js

window.ethereum
JavaScript Ethereum Provider API (EIP-1193)

request: wrapper function for RPCs
submit RPC requests via MetaMask

eth_requestAccounts
get accounts of MetaMask

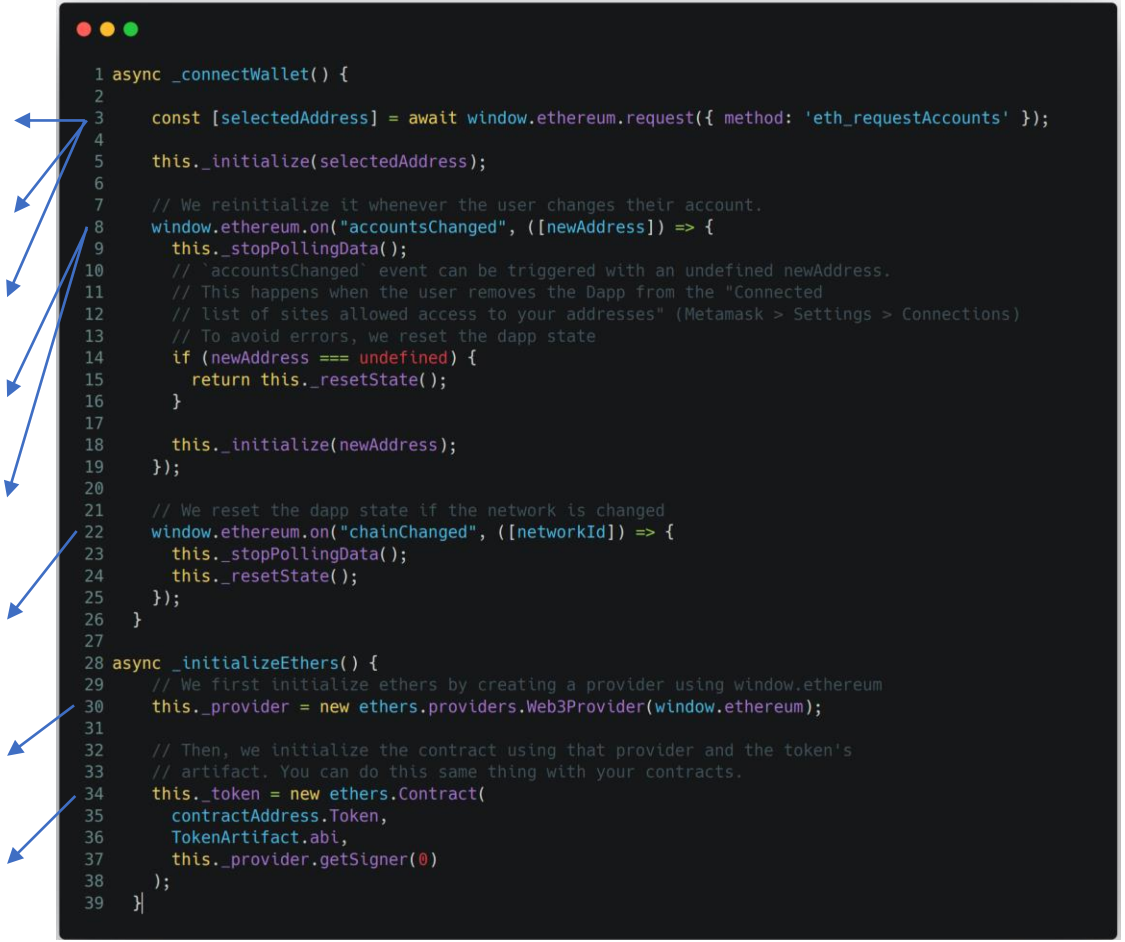
window.Ethereum.on()
listen for events

accountChanged
when MetaMask account changed, call handler to initialize with new address

chainChanged
when the network changed, call handler to reset the dapp state

Web3Provider
an EIP-1193 Provider or Web3Provider-compatible Provider as an ethers.js Provider

ethers.Contract
create a new instance of the contract



```
async _connectWallet() {

    const [selectedAddress] = await window.ethereum.request({ method: 'eth_requestAccounts' });

    this._initialize(selectedAddress);

    // We reinitialize it whenever the user changes their account.
    window.ethereum.on("accountsChanged", ([newAddress]) => {
      this._stopPollingData();
      // `accountsChanged` event can be triggered with an undefined newAddress.
      // This happens when the user removes the Dapp from the "Connected
      // list of sites allowed access to your addresses" (Metamask > Settings > Connections)
      // To avoid errors, we reset the dapp state
      if (newAddress === undefined) {
        return this._resetState();
      }

      this._initialize(newAddress);
    });

    // We reset the dapp state if the network is changed
    window.ethereum.on("chainChanged", ([networkId]) => {
      this._stopPollingData();
      this._resetState();
    });
  }

async _initializeEthers() {
    // We first initialize ethers by creating a provider using window.ethereum
    this._provider = new ethers.providers.Web3Provider(window.ethereum);

    // Then, we initialize the contract using that provider and the token's
    // artifact. You can do this same thing with your contracts.
    this._token = new ethers.Contract(
      contractAddress.Token,
      TokenArtifact.abi,
      this._provider.getSigner(0)
    );
  }
```

MetaMask Ethereum Provider API

MetaMask injects a global API into websites visited by its users at *window.ethereum*
This API allows websites to 1) request users' Ethereum accounts,
2) read data from blockchains the user is connected to (local, testnet, mainnet)
3) suggest that the user sign messages and transactions
The Ethereum JavaScript provider API is specified by EIP-1193

Methods

- **ethereum.request(args)**: submit RPC requests to Ethereum via MetaMask
  - methods: eth_requestAccounts, eth_accounts, eth_call, eth_getBalance, eth_sendTransaction, etc
- **ethereum.on(eventType, handler)**: listen for a specific event and call a handler
  - accountsChanged, chainChanged, connect, disconnect, message

Errors

- ethereum.request() throws errors (EIP-1474)
  - 4001: The request was rejected by the user
  - -32602: The parameters were invalid
  - -32603: Internal error

https://docs.metamask.io/guide/ethereum-provider.html
https://github.com/MetaMask/providers
https://eips.ethereum.org/EIPS/eip-1193
https://eips.ethereum.org/EIPS/eip-1474

Under the hood

What happen
when you call a smart contract function?

bytecode and ABI

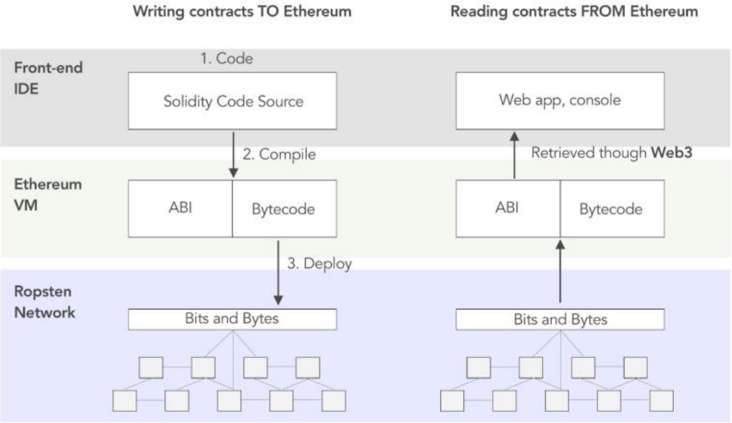
Token.sol → **solc** (Solidity Compiler)

bytecode *The smart contract information in a binary format on the Ethereum Virtual Machine*

```
0x60806040526040518060400160405280601381526020017f4b41495
354204861726468617420546f6b656e00000000000000000000000000
815250600090816200004a9190620003ae565b506040518060400160
405280600381526020017f4b485400000000000000000000000000000
000000000000000000000000000008152506001908162000091919062
0003ae565b50620f4240600255348015620000a657600080fd5b.......
```

ABI(Application Binary Interface)

An interpreter that facilitates communication with the EVM bytecode Human-readable JSON format

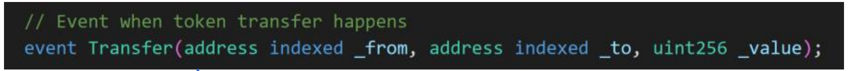


Source: https://hackernoon.com/hn-images/1*Sz1a7G2pQ62UnkHoieve4w.jpeg

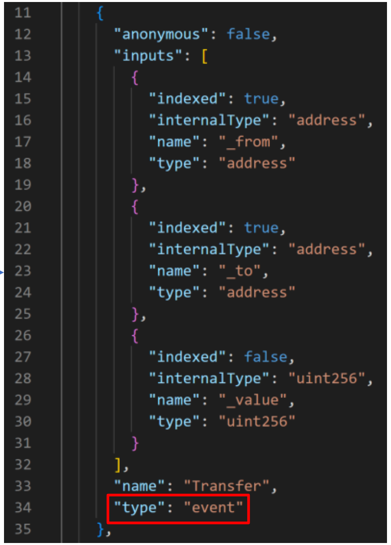
https://www.alchemy.com/overviews/solidity-abi
https://www.alchemy.com/overviews/what-is-an-abi-of-a-smart-contract-examples-and-usage
https://cypherpunks-core.github.io/ethereumbook/13evm.html

ABI from Token.sol

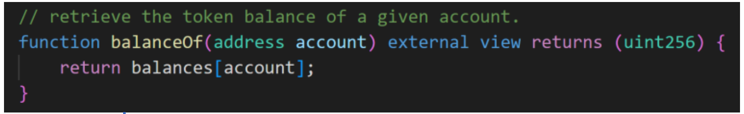
frontend/src/contracts/Token.json



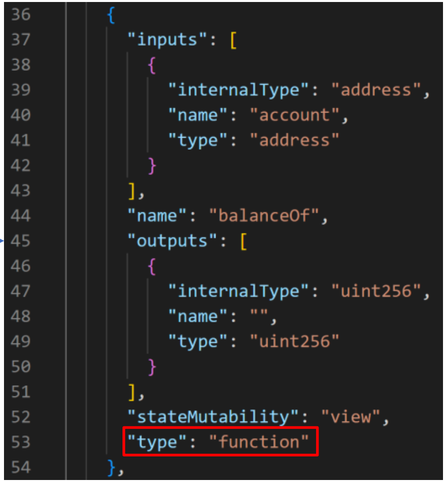
```
// Event when token transfer happens
event Transfer(address indexed _from, address indexed _to, uint256 _value);
```



```
    {
      "anonymous": false,
      "inputs": [
        {
          "indexed": true,
          "internalType": "address",
          "name": "_from",
          "type": "address"
        },
        {
          "indexed": true,
          "internalType": "address",
          "name": "_to",
          "type": "address"
        },
        {
          "indexed": false,
          "internalType": "uint256",
          "name": "_value",
          "type": "uint256"
        }
      ],
      "name": "Transfer",
      "type": "event"
    },
```



```
// retrieve the token balance of a given account.
function balanceOf(address account) external view returns (uint256) {
    return balances[account];
}
```

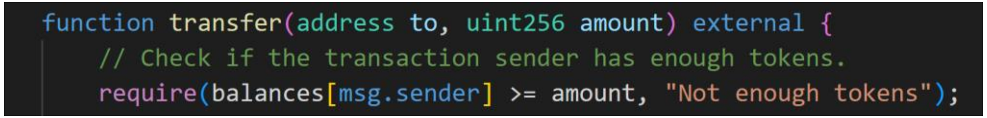


```
    {
      "inputs": [
        {
          "internalType": "address",
          "name": "account",
          "type": "address"
        }
      ],
      "name": "balanceOf",
      "outputs": [
        {
          "internalType": "uint256",
          "name": "",
          "type": "uint256"
        }
      ],
      "stateMutability": "view",
      "type": "function"
    },
```

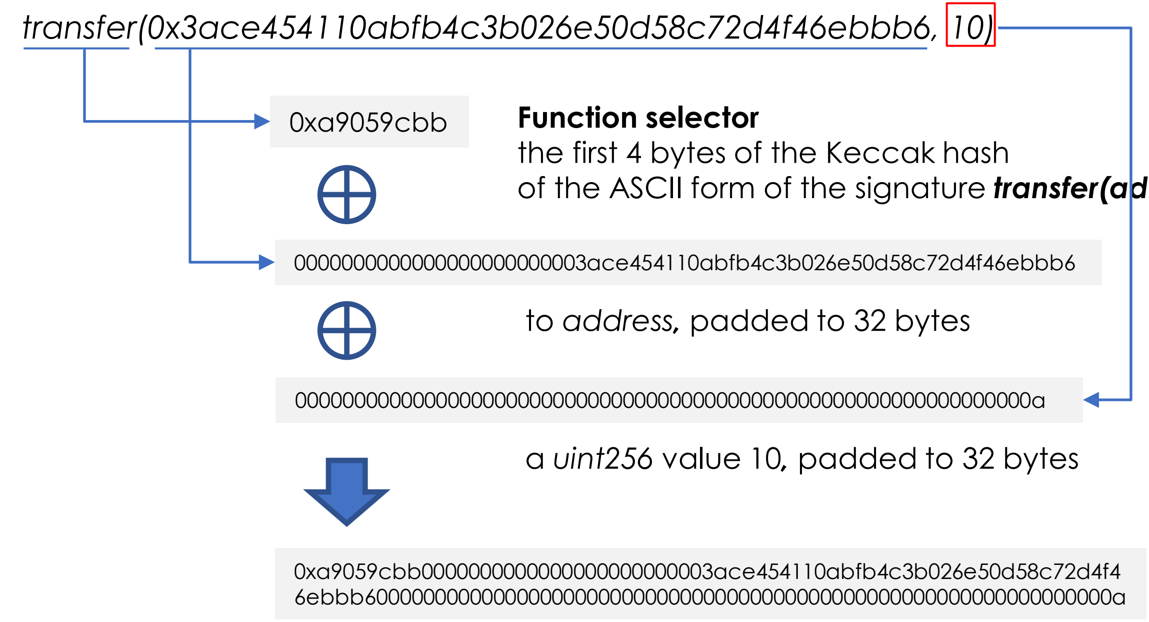
ABI JSON spec: https://docs.soliditylang.org/en/develop/abi-spec.html#json

ABI encoding

encode a smart contract function
in binary format to send to Ethereum



```
function transfer(address to, uint256 amount) external {
    // Check if the transaction sender has enough tokens.
    require(balances[msg.sender] >= amount, "Not enough tokens");
```



transfer(0x3ace454110abfb4c3b026e50d58c72d4f46ebbb6, 10)

0xa9059cbb

Function selector
the first 4 bytes of the Keccak hash
of the ASCII form of the signature ***transfer(address, uint256)***

0000000000000000000000003ace454110abfb4c3b026e50d58c72d4f46ebbb6

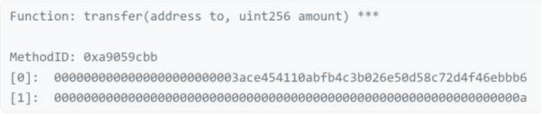
to *address*, padded to 32 bytes

000000000000000000000000000000000000000000000000000000000000000a

a *uint256* value 10, padded to 32 bytes

0xa9059cbb0000000000000000000000003ace454110abfb4c3b026e50d58c72d4f4
6ebbb6000000000000000000000000000000000000000000000000000000000000000a

Sepolia Etherscan



```
Function: transfer(address to, uint256 amount) ***

MethodID: 0xa9059cbb
[0]:  0000000000000000000000003ace454110abfb4c3b026e50d58c72d4f46ebbb6
[1]:  000000000000000000000000000000000000000000000000000000000000000a
```

Sepolia etherscan: https://sepolia.etherscan.io/tx/0xe76959281cf6caf46765d1b754b9c2a3da4fe0e23d0e1415dfa2abeb95da3f55
ABI encoding example: https://docs.soliditylang.org/en/develop/abi-spec.html#examples

Sending a transaction to a contract

eth_sendTransaction

```
{
  "id": 2,
  "jsonrpc": "2.0",
  "method": "eth_sendTransaction",
  "params": [
    {
      "from": "0x1923f626bb8dc025849e00f99c25fe2b2f7fb0db",
      "gas": "0x55555",
      "maxFeePerGas": "0x1234",
      "maxPriorityFeePerGas": "0x1234",
      "input": "0xabcd",
      "nonce": "0x0",
      "to": "0x07a565b7ed7d7a678680a4c162885bedbb695fe0",
      "value": "0x1234",
      "data":
"0xa9059cbb0000000000000000000000003ace454110abfb4c3b026e50d58c72d
4f46ebbb6000000000000000000000000000000000000000000000000000000000000000000
0000a"
    }
  ]
}
```

Return**: 32 bytes data of the **transaction hash

https://ethereum.org/en/developers/docs/transactions/
https://ethereum.org/en/developers/docs/apis/json-rpc/

eth_getTransactionReceipt [tx hash]

Return

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "blockHash":

"0xa957d47df264a31badc3ae823e10ac1d444b098d9b73d204c40426e57f47e8c
3",
    "blockNumber": "0xeff35f",
    "contractAddress": null, // string of the address if it was created
    "cumulativeGasUsed": "0xa12515",
    "effectiveGasPrice": "0x5a9c688d4",
    "from": "0x6221a9c005f6e47eb398fd867784cacfdcfff4e7",
    "gasUsed": "0xb4c8",
    "logs": [{
      // logs such as events
    }],
    "logsBloom": "0x00...0", // 256 byte bloom filter
    "status": "0x1",
    "to": "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2",
    "transactionHash":

"0x85d995eba9763907fdf35cd2034144dd9d53ce32cbec21349d4b12823c6860c5
",
    "transactionIndex": "0x66",
    "type": "0x2"
  }
}
```

Type of transactions

- **Regular transactions**: a transaction from one account to another
- **Execution of a contract**: a transaction that interacts with a deployed smart contract. In this case, 'to' address is the smart contract address
- **Contract deployment transactions**: a transaction without a 'to' address, where the data field is used for the contract code

https://ethereum.org/en/developers/docs/transactions/

Wrap-up

We Learned

Building Token Dapp
Hardhat-based smart contract development
ethers.js & MetaMask Web3Provider
Under the hood: what happen in calling a contract

Note.

Many people simply copy and paste contract code without understanding how it works.
Be careful when copying code,
and try to understand how it works.

Q & A