Ethereum

Смарт-контракты

Andrey Makarov

Обо мне

* 10+ лет опыта

* HAQQ Wallet
* Смарт контракты для работы
* Индексеры и прочее

Оглавление

Оглавление

Оглавление

- Немного теории
- Смарт контракт
- Тестирование

Оглавление

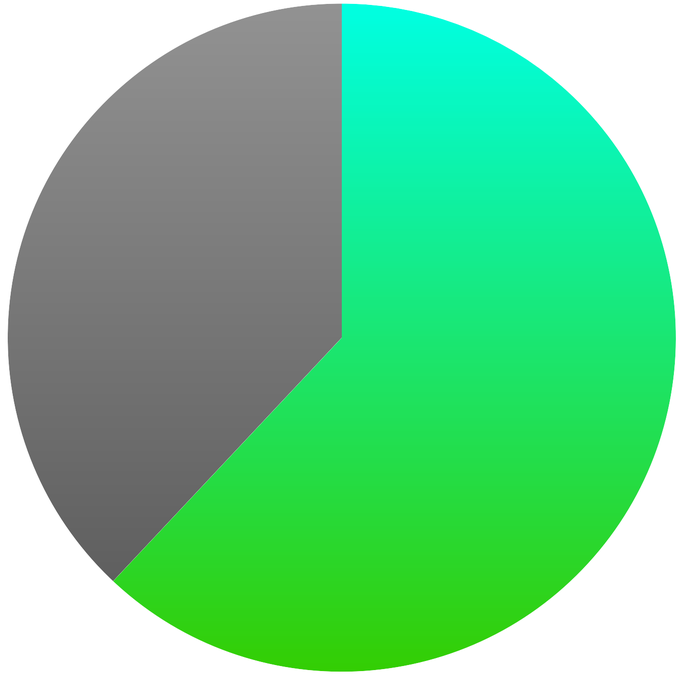
Оглавление

Оглавление

Смарт контракты

Роль смарт контрактов в современном мире

- Финансовая сфера



Роль смарт контрактов в современном мире

- Финансовая сфера
- Логистика и снабжение

Роль смарт контрактов в современном мире

- Финансовая сфера
- Логистика и снабжение
- Недвижимость

Роль смарт контрактов в современном мире

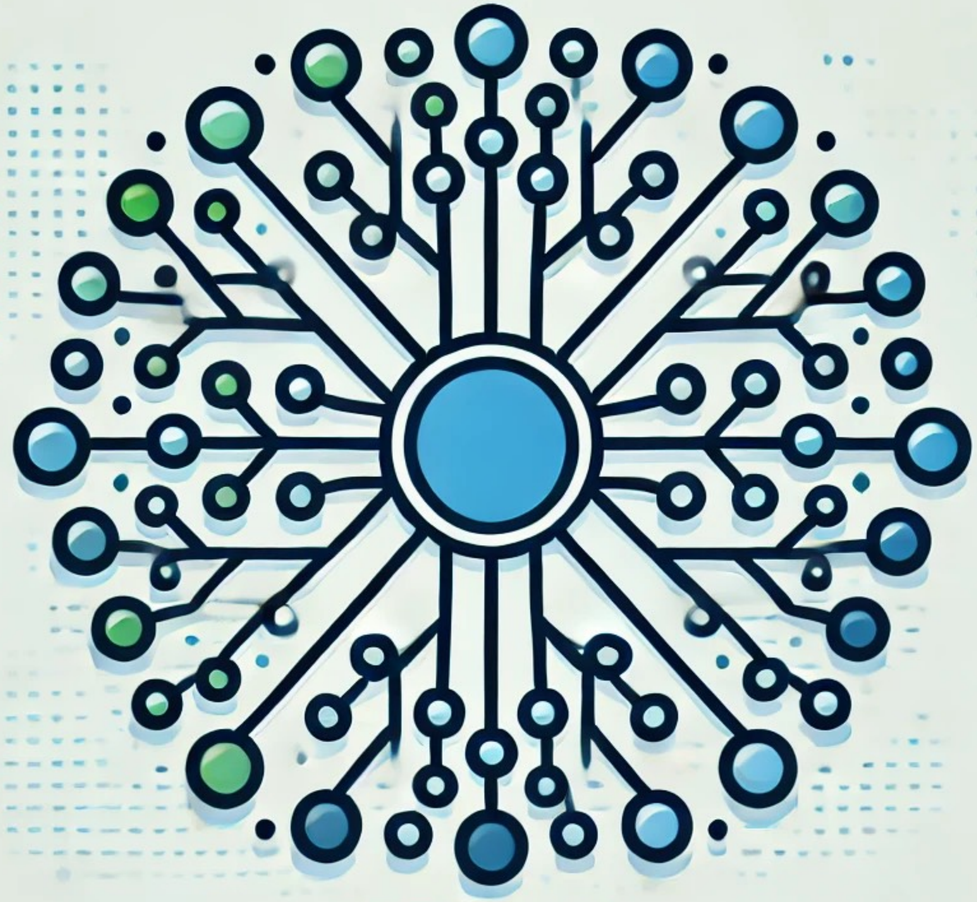
- Финансовая сфера
- Логистика и снабжение
- Недвижимость
- Здравоохранение

Роль смарт контрактов в современном мире

- Финансовая сфера
- Логистика и снабжение
- Недвижимость
- Здравоохранение
- Интеллектуальная собственность

Основные принципы смарт-контрактов

Децентрализация



Автоматизация

Безопасность

Прозрачность

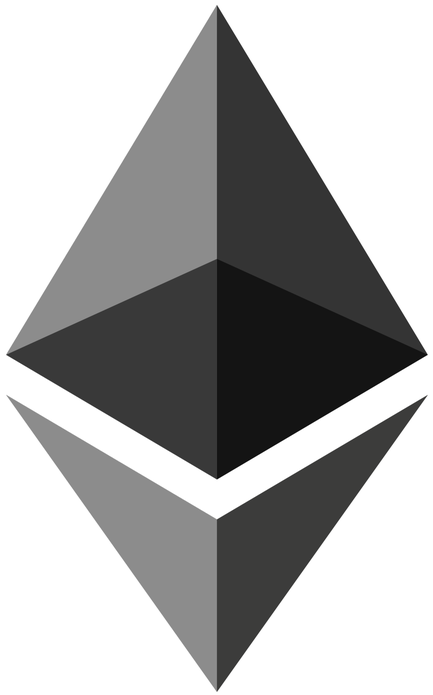
Надежность

Эффективность

Независимость от третьей стороны

Технологии и платформы для создания смарт-контрактов

Технологии и платформы для создания смарт-контрактов



Ethereum

Технологии и платформы для создания смарт-контрактов

Binance Smart Chain

Технологии и платформы для создания смарт-контрактов

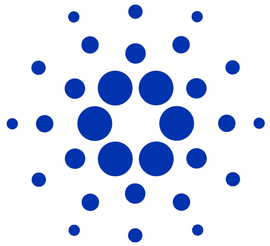
Solana

Технологии и платформы для создания смарт-контрактов

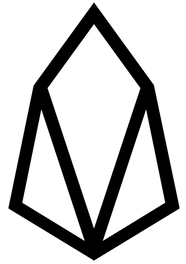
The Open Network (TON)

Технологии и платформы для создания смарт-контрактов

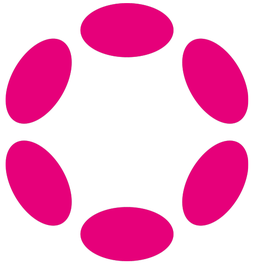
Tezor



Cardano



EOS



Polkadot

Avalanche

Преимущества смарт-контрактов

Снижение затрат

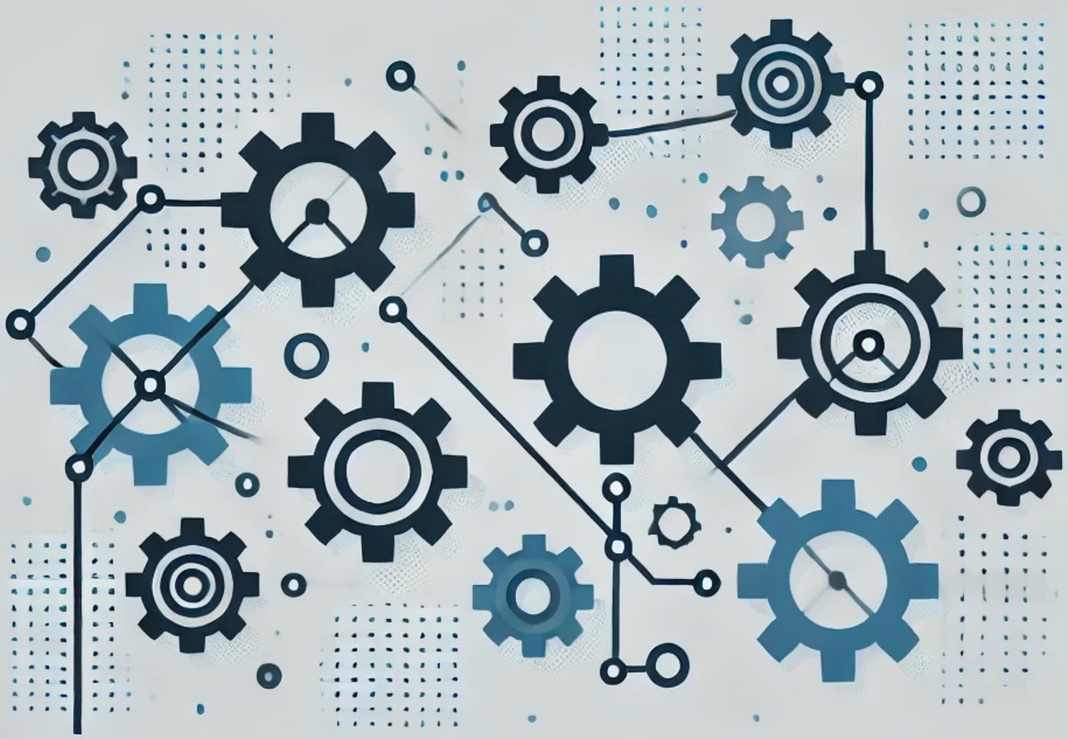
Глобальный доступ

Применимость в разных отраслях

Недостатки смарт-контрактов

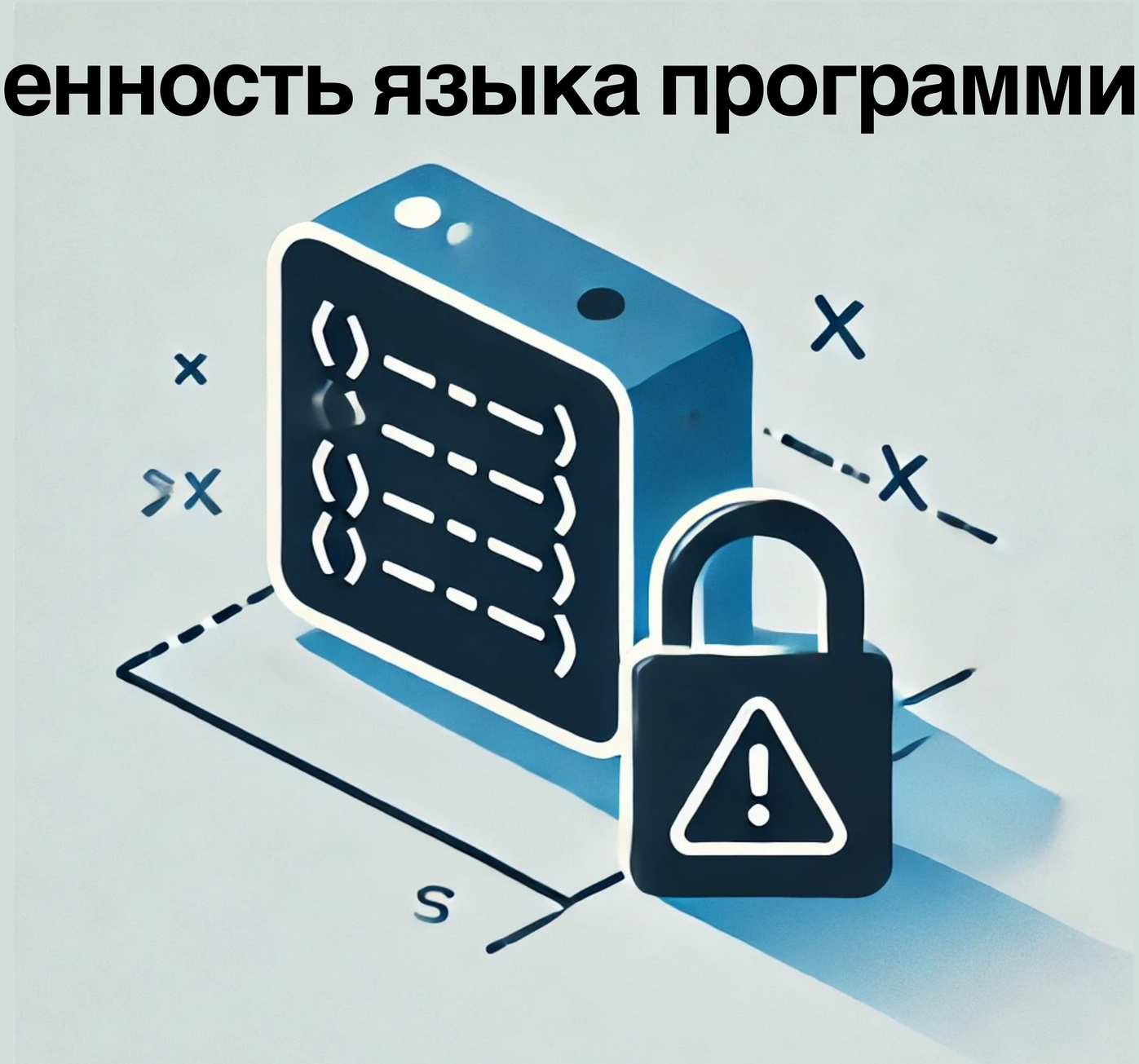
Невозможность исправления ошибок

Сложность разработки



Уязвимости

Ограниченность языка программирования



Скорость и масштабируемость

Юридические и регуляторные вопросы

Зависимость от оракулов

Низкая анонимность

Инструменты

Hardhat


OpenZeppelin

Разработка

```
❯ npx hardhat init
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 v2.21.0 👷

✔ What do you want to do? · Create a TypeScript project
✔ Hardhat project root: · /Users/.../contracts_tmo
✔ Do you want to add a .gitignore? (Y/n) · y
✔ Do you want to install this sample project's dependencies with npm (hardhat @nomicfoundation/hardhat-toolbox)?
(Y/n) · y

npm install --save-dev "hardhat@^2.21.0" "@nomicfoundation/hardhat-toolbox@^4.0.0"

added 549 packages, and audited 550 packages in 1m

90 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

✨ Project created ✨

See the README.md file for some example tasks you can run

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

     https://github.com/NomicFoundation/hardhat
```

Контракт

```
1  // SPDX-License-Identifier: MIT
2  // Compatible with OpenZeppelin Contracts ^5.0.0
3  pragma solidity ^0.8.20;
4
5  import "@openzeppelin/contracts/utils/Pausable.sol";
6  import "@openzeppelin/contracts/access/Ownable.sol";
7
8  contract Crowdfunding is Pausable, Ownable {
9      constructor(address initialOwner) Ownable(initialOwner) {
10
11     }
12 }
```

Типы данных

```
1  // SPDX-License-Identifier: MIT
2  // Compatible with OpenZeppelin Contracts ^5.0.0
3  pragma solidity ^0.8.20;
4
5  import "@openzeppelin/contracts/utils/Pausable.sol";
6  import "@openzeppelin/contracts/access/Ownable.sol";
7
8  contract Crowdfunding is Pausable, Ownable {
9      address payable public creator;
10     address payable[] public contributors;
11     mapping(address => uint) public contributions;
12
13     constructor(address initialOwner, address _creator) Ownable(initialOwner) {
14         creator = payable(_creator);
15     }
16 }
```

Типы данных. Перечисления

```
1  // SPDX-License-Identifier: MIT
2  // Compatible with OpenZeppelin Contracts ^5.0.0
3  pragma solidity ^0.8.20;
4
5  import "@openzeppelin/contracts/utils/Pausable.sol";
6  import "@openzeppelin/contracts/access/Ownable.sol";
7
8  contract Crowdfunding is Pausable, Ownable {
9      ...
10
11     State public state;
12
13     enum State {Verifying, Rejected, Fundraising, Expired, Successful}
14
15     constructor(address initialOwner, ...) Ownable(initialOwner) {
16         ...
17         state = State.Verifying;
18     }
19 }
```

Вызов функции

```
...
contract Crowdfunding is Pausable, Ownable {
    ...
    modifier beforeDeadline() {
        require(block.timestamp < deadline, "Deadline has passed");
        _;
    }
    ...
    function contribute() public payable beforeDeadline whenNotPaused {
        require(state == State.Fundraising, "Crowdfunding is not in fundraising state");
        require(msg.value > 0, "Contribution must be greater than 0");

        if (contributions[msg.sender] == 0) {
            contributors.push(payable(msg.sender));
        }

        contributions[msg.sender] += msg.value;
        totalContributions += msg.value;
    }
}
```

События

```
...
contract Crowdfunding is Pausable, Ownable {
    ...
    event Contribute(address contributor, uint amount);
    ...
    function contribute() public payable beforeDeadline whenNotPaused {
        ...
        emit Contribute(msg.sender, msg.value);
    }
}

```

Области видимости

```
...
contract Crowdfunding is Pausable, Ownable {
    ...
    function finish() public onlyOwner whenNotPaused afterDeadline {
        require(state == State.Fundraising, "Crowdfunding is not in fundraising state");
        if (totalContributions >= goal) {
            state = State.Successful;
            _withdraw();
        }

        emit StateChanged(state);
    }

    function _withdraw() private onlyOwner {
        uint amount = address(this).balance;
        creator.transfer(amount);

        emit Withdraw(creator, amount);
    }
}
```

Создание из контракта

```
1  contract CrowdfundingList is Initializable, PausableUpgradeable, OwnableUpgradeable {
2      ...
3
4      function create(uint _goal, uint _duration, string memory _metadata, uint256 _metadataHash)
external whenNotPaused returns (address)  {
5          Crowdfunding crowdfundingContract = new Crowdfunding(address(this), msg.sender, _goal,
_duration, _metadata, _metadataHash);
6          address crowdfundingAddress = address(crowdfundingContract);
7          crowdfunding[crowdfundingAddress] = crowdfundingContract;
8          verifyingCrowdfundingArray.push(crowdfundingAddress);
9
10         emit CrowdfundingCreated(crowdfundingAddress, msg.sender, _goal, _duration, _metadata);
11         return crowdfundingAddress;
12     }
13 }
```

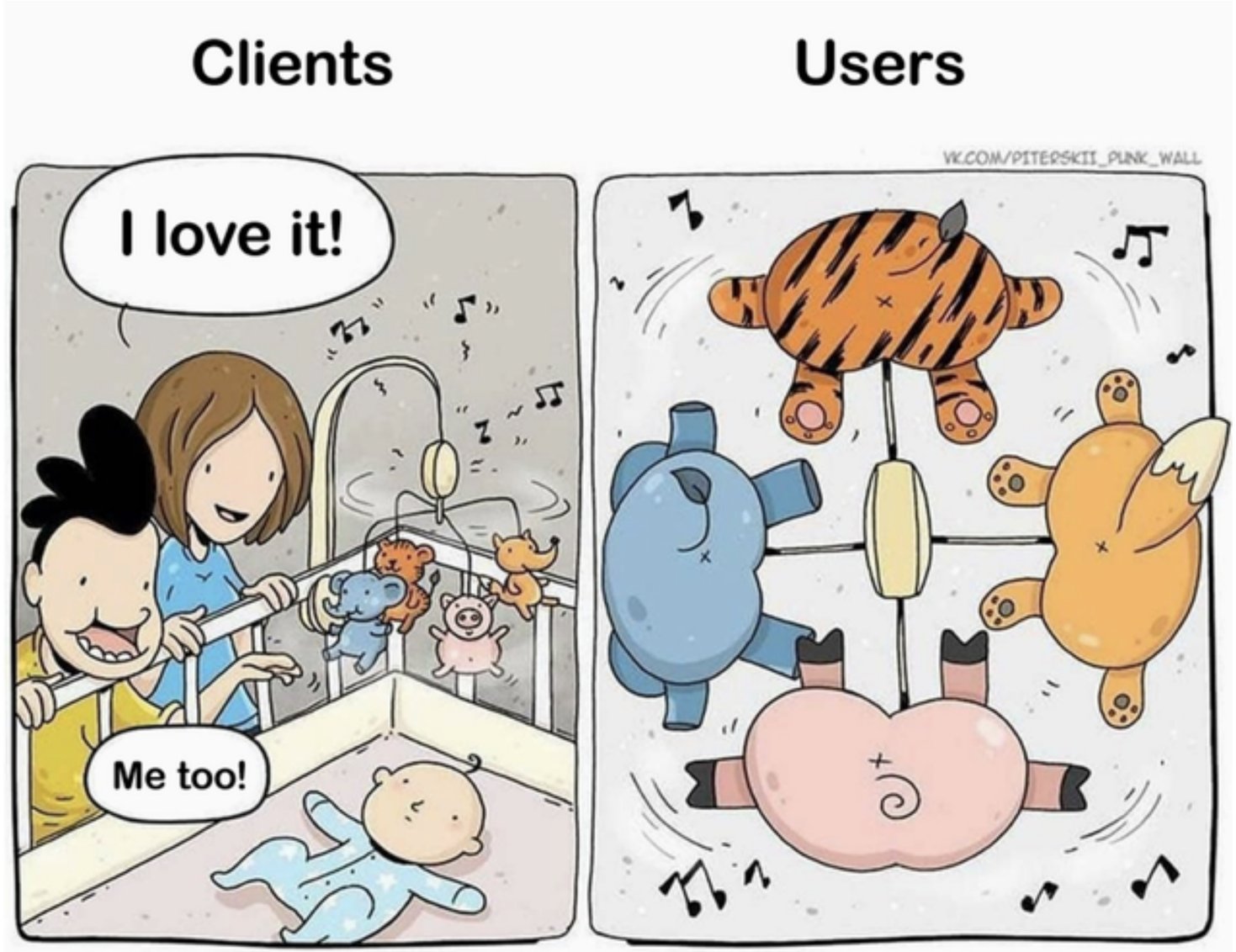

ГОТОВЬСЯ
МЫ ИДЁМ ТЕСТИРОВАТЬ

```
import {loadFixture, time,} from "@nomicfoundation/hardhat-toolbox/network-helpers";
import {expect} from "chai";
import {ethers} from "hardhat";

describe("Crowdfunding", function () {
    async function deployOneYearLockFixture() {
        const deadlineTime = (await time.latest()) + 60;
        const goal = ethers.parseEther("10");
        const [owner, account1, account2, account3] = await ethers.getSigners();

        const Crowdfunding = await ethers.getContractFactory("Crowdfunding");
        const contract = await Crowdfunding.deploy( owner.address, account1.address, goal,
deadlineTime, “Test", ethers.parseEther(“1") );

        return {contract, deadlineTime, owner, account1, account2, account3};
    }

    it('should create contract', async () => {
        const {contract} = await loadFixture(deployOneYearLockFixture);
        const info = await contract.info();
        expect(info.goal).to.equal(ethers.parseEther("10"));
        expect(info.deadline).to.equal(info.deadline);
        expect(info.metadata).to.equal("Test");
        expect(info.current).to.equal(0);
    });
```

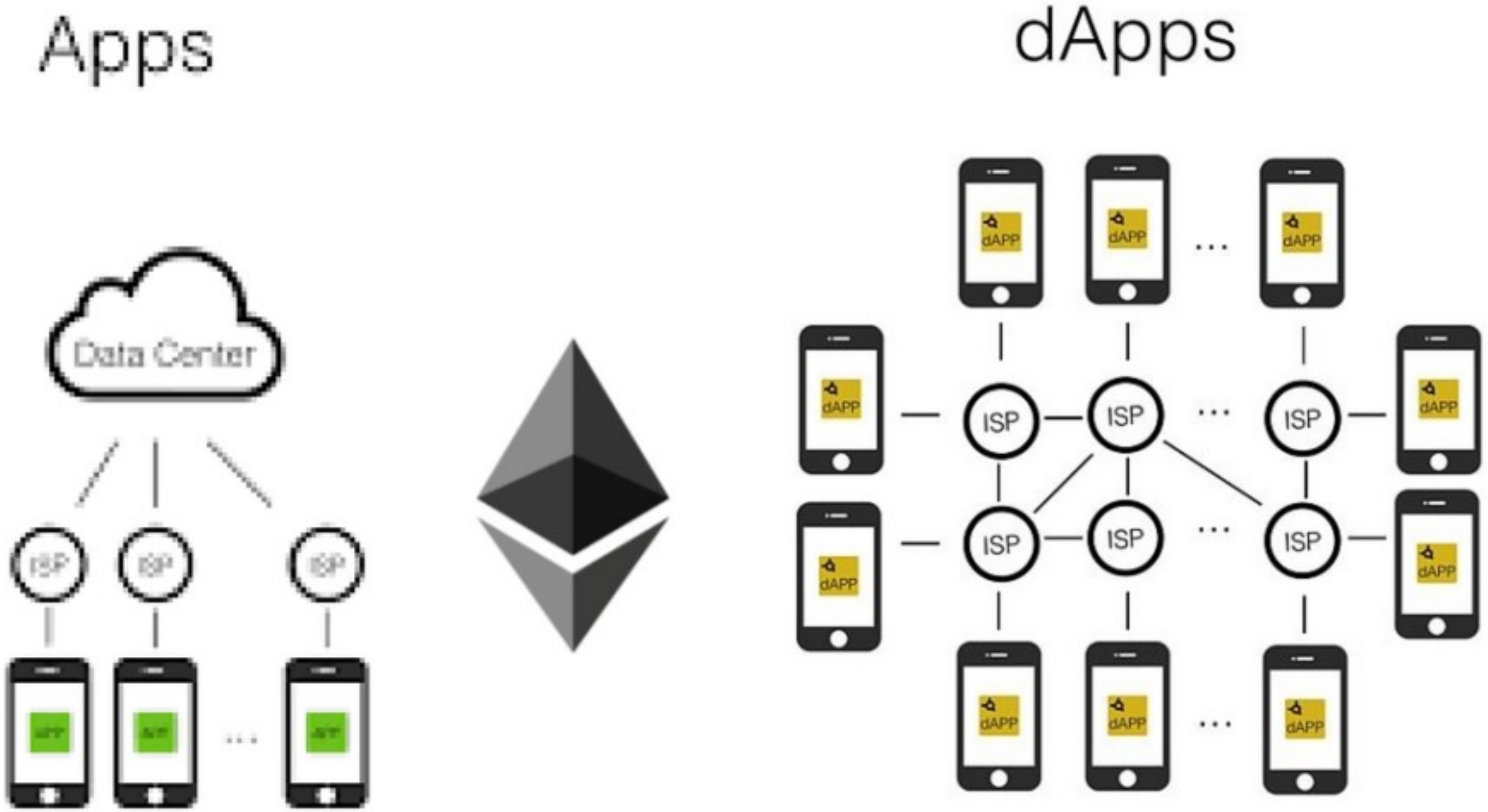
```
❯ npx hardhat test

  Crowdfunding
    ✔ should contribute
    ✔ should not contribute if not started
    ✔ should not contribute after deadline reached
    ✔ should withdraw if goal reached
    ✔ should refund if goal not reached
    ✔ should emit event on contribute
    ✔ should emit event on withdraw
    ✔ should emit event on refund

  CrowdfundingList
    ✔ should create crowdfunding

  CrowdfundingList
    ✔ should create text contract crowdfunding
```

Solc version: 0.8.24	Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas
Methods			
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
Crowdfunding	contribute	83758	117958	113072	14	-
Crowdfunding	finish	47001	52146	49574	10	-
Crowdfunding	start	-	-	47004	14	-
CrowdfundingList	create	-	-	890389	2	-
CrowdfundingList	start	-	-	107116	2	-
Deployments				**% of limit**		
Crowdfunding		-	-	897595	3 %	-
CrowdfundingList		-	-	2265089	7.6 %	-
CrowdfundingTest		-	-	2882789	9.6 %	-

UI



DApp


Apps
dApps
Data Center
ISP
ISP
ISP
ISP
ISP
ISP
ISP
ISP
ISP
ISP
dAPP
...

Explore Web3
Trending
Hot contracts NEW
Web3 ecosystem
Upcoming
Rankings
Rewards
Portfolio
RADAR
Research

PRO Upgrade to PRO
Boosting
Advertise
API

Explorer

DappRadar's Explorer is an all-in-one dashboard featuring bite-sized, modular insights of industry trends. Dive into the ever-changing dapp landscape with a dynamic overview, all at a glance.

Industry signals

NFTs

Signal: Volume

Sappy Seals	Liberty Cats	Bitcoin Frogs
+628.11%	+605.34%	+452.79%
24h Volume	24h Volume	24h Volume

Trending

Games DeFi NFTs

#	Name	UAW	24h%
1	Bitball Arbitrum	3.45k	+20,229.41%
	AtDawn		

Portfolio

Net worth

-

24h Change

-

GM fren

XP +5 | XP +5 | XP +5 | XP +5 | XP +10 | XP +10 | XP +50

GM

Come back each day to increase rewards.

Latest Articles

Show more

October 24 · 7min read

WAX Dapp Ecosystem Q3 2024 Report

October 24 · 2min read

QuickSwap Brings Gasless Perpetual

Как DApp используют смарт-контракты

Определение бизнес-логики

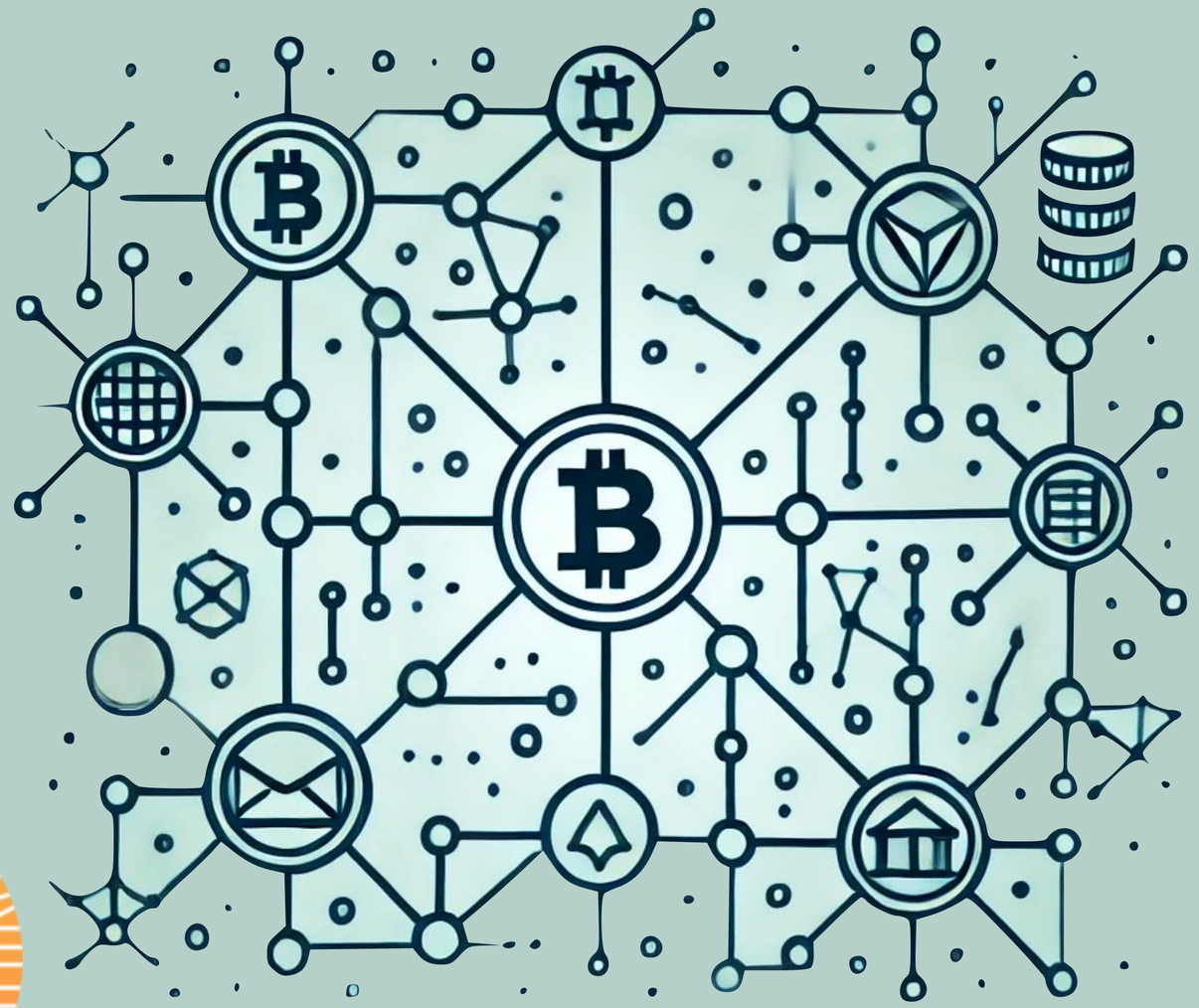
Взаимодействие с смарт-контрактами

Обновление состояния

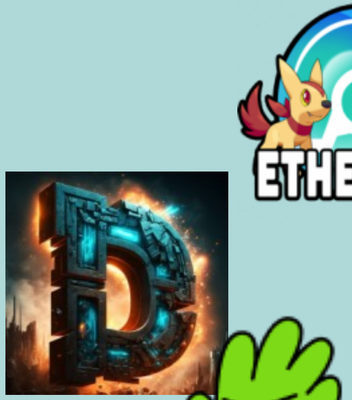
Учет и хранение данных

Примеры сценариев использования

Децентрализованные финансы (DeFi)



Игры



Социальные сети

Логистика и поставки

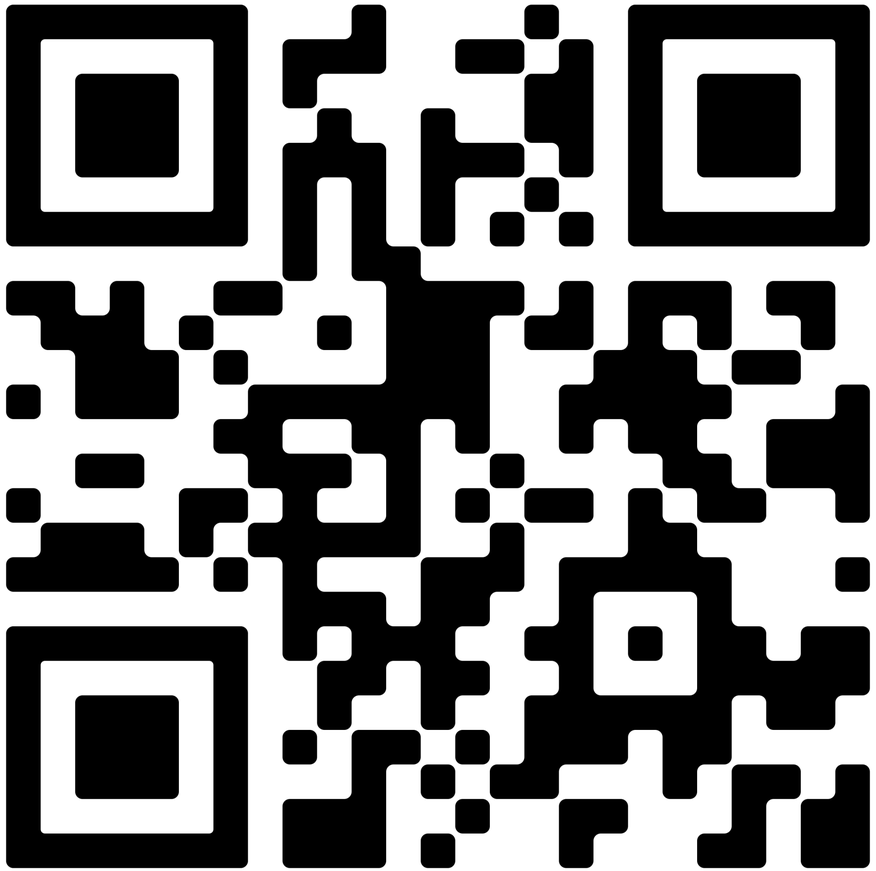
Здравоохранение

Образование

Голосования и управление

Разработка и хостинг

Примеры DApp в разных сферах



Подключение к провайдеру

```
import {useCallback, useState} from "react";
import {ethers} from "ethers";

const [accountData, setAccountData] = useState({});

const connect = useCallback(async () => {
    const ethereum = window.ethereum;
    if (typeof ethereum !== "undefined") {
        try {
            const accounts = await ethereum.request({ method: "eth_requestAccounts", });
            const provider = new ethers.BrowserProvider(ethereum);
            const network = await provider.getNetwork();
            const signer = await provider.getSigner();

            setAccountData({ address: accounts[0], chainId: network.chainId.toString(), network:
network.name, signer, provider });
        } catch (error: Error | any) {
            alert(`Error connecting to MetaMask: ${error?.message ?? error}`);
        }
    } else {
        alert("MetaMask not installed");
    }
}, []);
```

Подключение к провайдеру

```
import {useCallback, useState} from "react";
import {ethers} from "ethers";

const [accountData, setAccountData] = useState({});

const connect = useCallback(async () => {
    const ethereum = window.ethereum;
    if (typeof ethereum !== "undefined") {
        try {
            const accounts = await ethereum.request({ method: "eth_requestAccounts", });
            const provider = new ethers.BrowserProvider(ethereum);
            const network = await provider.getNetwork();
            const signer = await provider.getSigner();

            setAccountData({ address: accounts[0], chainId: network.chainId.toString(), network:
network.name, signer, provider });
        } catch (error: Error | any) {
            alert(`Error connecting to MetaMask: ${error?.message ?? error}`);
        }
    } else {
        alert("MetaMask not installed");
    }
}, []);
```

Подключение к провайдеру

```
import {useCallback, useState} from "react";
import {ethers} from "ethers";

const [accountData, setAccountData] = useState({});

const connect = useCallback(async () => {
    const ethereum = window.ethereum;
    if (typeof ethereum !== "undefined") {
        try {
            const accounts = await ethereum.request({ method: "eth_requestAccounts", });
            const provider = new ethers.BrowserProvider(ethereum);
            const network = await provider.getNetwork();
            const signer = await provider.getSigner();

            setAccountData({ address: accounts[0], chainId: network.chainId.toString(), network:
network.name, signer, provider });
        } catch (error: Error | any) {
            alert(`Error connecting to MetaMask: ${error?.message ?? error}`);
        }
    } else {
        alert("MetaMask not installed");
    }
}, []);
```

Подключение к провайдеру

```
import {useCallback, useState} from "react";
import {ethers} from "ethers";

const [accountData, setAccountData] = useState({});

const connect = useCallback(async () => {
    const ethereum = window.ethereum;
    if (typeof ethereum !== "undefined") {
        try {
            const accounts = await ethereum.request({ method: "eth_requestAccounts", });
            const provider = new ethers.BrowserProvider(ethereum);
            const network = await provider.getNetwork();
            const signer = await provider.getSigner();

            setAccountData({ address: accounts[0], chainId: network.chainId.toString(), network:
network.name, signer, provider });
        } catch (error: Error | any) {
            alert(`Error connecting to MetaMask: ${error?.message ?? error}`);
        }
    } else {
        alert("MetaMask not installed");
    }
}, []);
```

Подключение к провайдеру

```
import {useCallback, useState} from "react";
import {ethers} from "ethers";

const [accountData, setAccountData] = useState({});

const connect = useCallback(async () => {
    const ethereum = window.ethereum;
    if (typeof ethereum !== "undefined") {
        try {
            const accounts = await ethereum.request({ method: "eth_requestAccounts", });
            const provider = new ethers.BrowserProvider(ethereum);
            const network = await provider.getNetwork();
            const signer = await provider.getSigner();

            setAccountData({ address: accounts[0], chainId: network.chainId.toString(), network:
network.name, signer, provider });
        } catch (error: Error | any) {
            alert(`Error connecting to MetaMask: ${error?.message ?? error}`);
        }
    } else {
        alert("MetaMask not installed");
    }
}, []);
```

Контракт

```
import {EthereumContext} from "@/src/context";
import {useContext, useMemo} from "react";
import {Contract} from "ethers";
import CrowdfundingList from "../../../contracts/artifacts/contracts/Croudfunding.sol/CrowdfundingList.json"

export function useCrowdfunding() {
    const appData = useContext(EthereumContext);
    return useMemo(
        () =>
            appData.signer &&
            new Contract(process.env.NEXT_PUBLIC_CROUDFUNDING_LIST_ADDRESS, CrowdfundingList.abi, appData.signer),
        [appData.signer]
    );
}
```

Контракт

```
1  import {EthereumContext} from "@/src/context";
2  import {useContext, useMemo} from "react";
3  import {Contract} from "ethers";
4  import CrowdfundingList from "../../../contracts/artifacts/contracts/Croudfunding.sol/
CrowdfundingList.json"
5
6  export function useCrowdfunding() {
7      const appData = useContext(EthereumContext);
8      return useMemo(
9          () =>
10             appData.signer &&
11             new Contract(process.env.NEXT_PUBLIC_CROUDFUNDING_LIST_ADDRESS, CrowdfundingList.abi,
appData.signer),
12         [appData.signer]
13     );
14 }
```

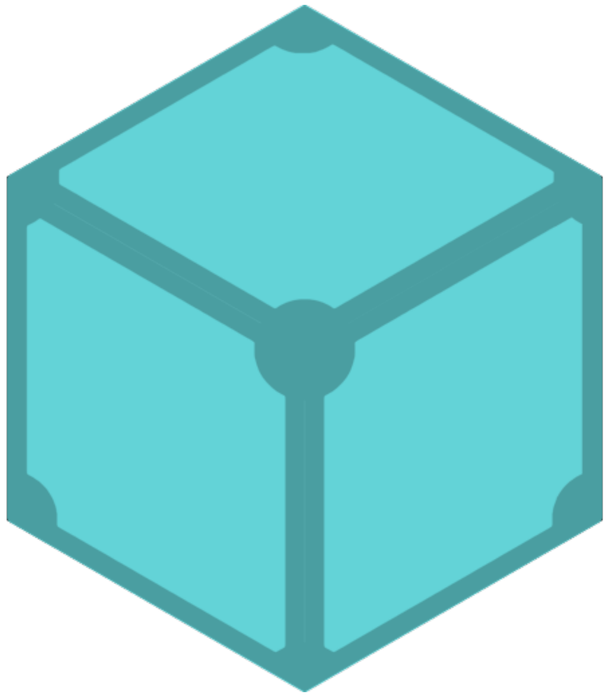
Чтение из контракта

```
1  import {Home} from "@/src/components";
2  import {useContext, useEffect, useState} from "react";
3  import {useCrowdfunding} from "@/src/hooks/useCroudfunding";
4  import {EthereumContext} from "@/src/context";
5
6  export default function HomeScreen() {
7      const appData = useContext(EthereumContext);
8      const crowdfunding = useCrowdfunding();
9      const [crowdfundingList, setCrowdfundingList] = useState([]);
10
11     useEffect(() => {
12         if (crowdfunding) {
13             crowdfunding.getActive(appData.address).then((list) => {
14                 setCrowdfundingList(list);
15             });
16         }
17     }, [crowdfunding, appData.address])
18
19     return <Home list={crowdfundingList}/>
20 }
```

Запись в контракт

```tsx
import {Create} from "@/src/components";
import {useCallback} from "react";
import {useCrowdfunding} from "@/src/hooks/useCroudfunding";
import {ethers} from "ethers";

export default function CreateScreen() {
    const crowdfunding = useCrowdfunding();
    const onCreate = useCallback(async (formData: Record<string, any>) => {
        if (!crowdfunding) return;
        const response = {cid: '', hash: ''}

        const goal = ethers.parseEther(formData.goal);
        const deadline = new Date(formData.deadline).getTime() / 1000;

        const receipt = await crowdfunding.create(
            goal, deadline, response.cid, BigInt(`0x${response.hash}`)
        );

        const resp = await receipt.wait();
        console.log('resp', resp);
    }, [crowdfunding])

    return <Create onCreate={onCreate}/>
}
```

Сохранение данных



IPFS

Запись в IPFS

```
export async function set(data: string): Promise<string> {
    const myBlob = new Blob([data], { type: 'application/json' });

    const form = new FormData();
    form.append('metadata.json', myBlob, 'metadata.json');

    const response = await fetch(
        `${process.env.NEXT_IPFS_NODE}/api/v0/add?pin=true`,
        {
            method: "POST",
            body: form,
            headers: {
                "Authorization": `Basic ${process.env.NEXT_IPFS_SECRET}`,
                "accept-type": "application/json",
            },
        },
    );

    const info = await response.json();
    return info.Hash
}
```

Чтение из IPFS

```
export async function get<T>(hash: string): Promise<T> {
    const response = await fetch(
        `${process.env.NEXT_IPFS_NODE}/api/v0/cat?arg=${hash}`,
        {
            method: "POST",
            headers: {
                "Authorization": `Basic ${process.env.NEXT_IPFS_SECRET}`,
                "accept-type": "application/json",
            },
        },
    );

    return await response.json()
}
```

Выводы

https://www.haqq.network/

https://github.com/vivalaakam/crowdfunding

Спасибо за внимание

https://www.openzeppelin.com/solidity-contracts

https://dappradar.com