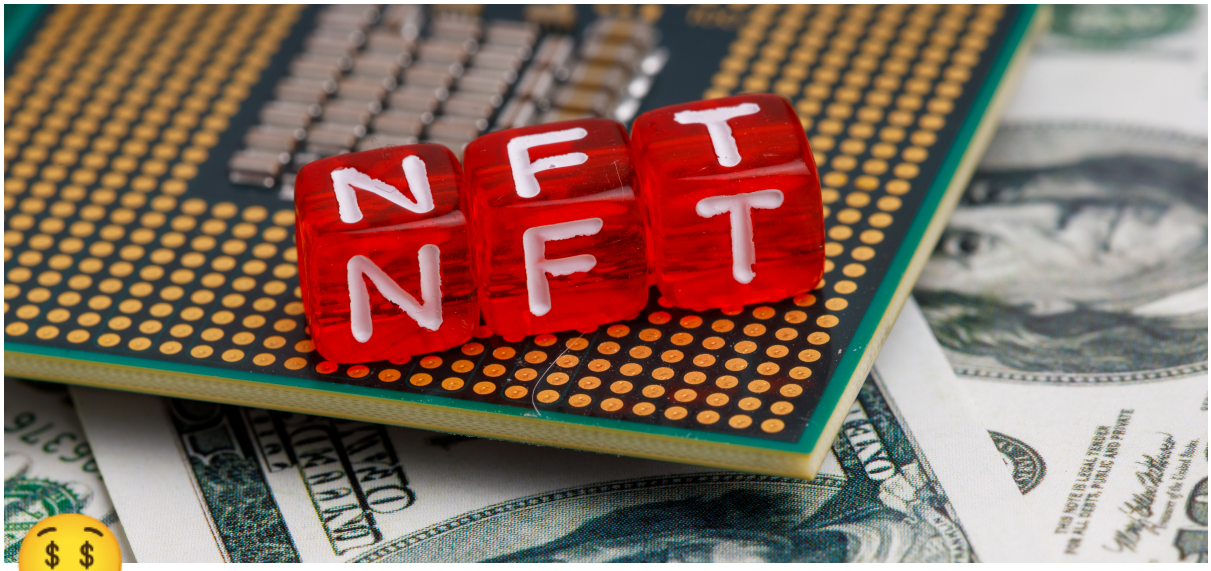


Web 3.0 Social Media App Like Twitter

Work done by: Ahmed Grati, Zeineb Labbane, Souha Ben Hassine, and Nour Belmabrouk

https://github.com/Souha-BH/Web3TwitterApp

1/ Goals and requirements:

In this project, we will build an **NFT-based social network**, a **decentralized version of Twitter**.

- To participate in the network, you must mint an NFT representing your avatar and username.
- Users can mint more than one NFT, and select the one they want to serve as their profile.
- NFT holders can post and tip other people’s posts.
- The post with the most tips will appear first on their feed.

2/ Strategy:

We will create a smart contract that will represent an NFT collection and the social network that the holders of the NFT can participate in. We will store each post and each NFT metadata on IPFS, and store the hash that points to it inside our smart contract. Once we coded our smart contract (in Solidity), and have written tests against them (in JS), we will deploy them to a local blockchain, then we'll build the client side using React.js, and connect it to the smart contract and the blockchain.

3/ Technology Stack & Tools:

What we will need is:

- Solidity (Writing Smart Contract)
- Javascript (React & Testing)
- Ethers (Blockchain Interaction)
- Hardhat (Development Framework)
- Ipfs (Metadata storage): we'll use Infura Ethereum API: Infura **provides the tools and infrastructure that allow developers to easily take their blockchain application from testing to scaled deployment** with simple, reliable access to Ethereum and IPFS.
- React routers (Navigational component)
- Metamask extension from Google Chrome
- NodeJS v 17>
- HardHat which is a smart contract development framework that comes with a solidity compiler.

4/ Development :

First thing first, we'll define our own data type for each post:

```
contract Decentratwitter is ERC721URIStorage {
...
struct Post {
        uint256 id;
        string hash;
        uint256 tipAmount;
        address payable author;
```

```
    }

    event PostCreated(
        uint256 id,
        string hash,
        uint256 tipAmount,
        address payable author
    );

    event PostTipped(
        uint256 id,
        string hash,
        uint256 tipAmount,
        address payable author
    );
...
}
```

Then we'll define our functions accordingly:

```


    function mint(string memory _tokenURI) external returns (uint256) {
        tokenCount++;
        _safeMint(msg.sender, tokenCount);
        _setTokenURI(tokenCount, _tokenURI);
        setProfile(tokenCount);
        return (tokenCount);
    }

    function setProfile(uint256 _id) public {
        require(
            ownerOf(_id) == msg.sender,
            "Must own the nft you want to select as your profile"
        );
        profiles[msg.sender] = _id;
    }

    function uploadPost(string memory _postHash) external {
        // Check that the user owns an nft
        require(
            balanceOf(msg.sender) > 0,
            "Must own a decentratwitter nft to post"
        );
        // Make sure the post hash exists
        require(bytes(_postHash).length > 0, "Cannot pass an empty hash");
        // Increment post count
        postCount++;
        // Add post to the contract
        posts[postCount] = Post(postCount, _postHash, 0, payable(msg.sender));
        // Trigger an event
        emit PostCreated(postCount, _postHash, 0, payable(msg.sender));
    }

    function tipPostOwner(uint256 _id) external payable {
        // Make sure the id is valid
```

```
        require(_id > 0 && _id <= postCount, "Invalid post id");
        // Fetch the post
        Post memory _post = posts[_id];
        require(_post.author != msg.sender, "Cannot tip your own post");
        // Pay the author by sending them Ether
        _post.author.transfer(msg.value);
        // Increment the tip amount
        _post.tipAmount += msg.value;
        // Update the image
        posts[_id] = _post;
        // Trigger an event
        emit PostTipped(_id, _post.hash, _post.tipAmount, _post.author);
    }

    // Fetches all the posts
    function getAllPosts() external view returns (Post[] memory _posts) {
        _posts = new Post[](postCount);
        for (uint256 i = 0; i < _posts.length; i++) {
            _posts[i] = posts[i + 1];
        }
    }

    // Fetches all of the users nfts
    function getMyNfts() external view returns (uint256[] memory _ids) {
        _ids = new uint256[](balanceOf(msg.sender));
        uint256 currentIndex;
        uint256 _tokenCount = tokenCount;
        for (uint256 i = 0; i < _tokenCount; i++) {
            if (ownerOf(i + 1) == msg.sender) {
                _ids[currentIndex] = i + 1;
                currentIndex++;
            }
        }
    }
```

PS: The state variables are stored to the blockchain and are the only data that a smart contract can manipulate directly, and through the use of functions defined in the smart contract that changes the state variables' values, then the state of the blockchain is changing which requires the user to pay a gas fee.

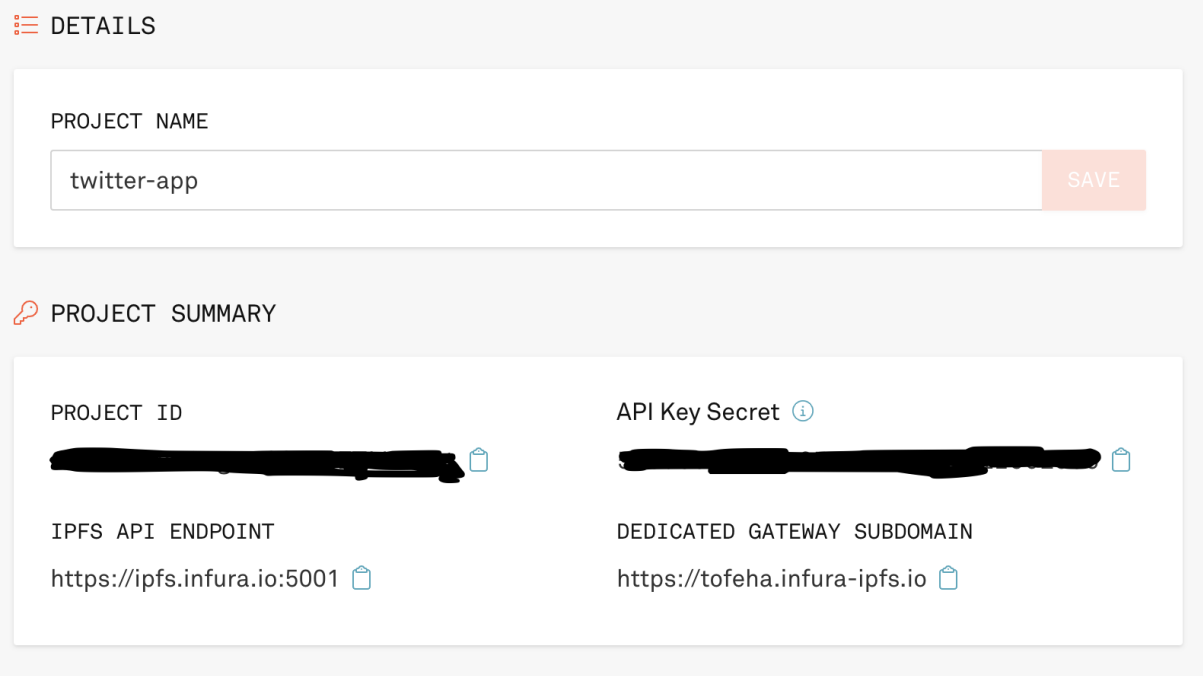
Next, we'll build the client side using React.js, and connect it to the smart contract and the blockchain, and write tests in JS.

Whenever we need to upload or fetch our NFTs, we'll use a third-party IPFS offered by INFURA.

```
const ipfsClient = require('ipfs-http-client');
const projectId = process.env.PROJECT_ID;
const projectSecret = process.env.PROJECT_SECRET;
const auth = 'Basic ' + Buffer.from(projectId + ':' + projectSecret).toString('base64');
const client = ipfsClient.create({
    host: 'ipfs.infura.io',
```

```
    port: 5001,
    protocol: 'https',
    headers: {
        authorization: auth,
    },
});
...
let posts = await Promise.all(results.map(async i => {
            // use hash to fetch the post's metadata stored on ipfs
            let response = await fetch(`https://tofeha.infura-ipfs.io/ipfs/${i.hash}`)
            const metadataPost = await response.json()
            // get authors nft profilean
            const nftId = await contract.profiles(i.author)
            // get uri url of nft profile
            const uri = await contract.tokenURI(nftId)
            // fetch nft profile metadata
            response = await fetch(uri)
            const metadataProfile = await response.json()
            // define author object
            const author = {
                address: i.author,
                username: metadataProfile.username,
                avatar: metadataProfile.avatar
            }
            // define post object
            let post = {
                id: i.id,
                content: metadataPost.post,
                tipAmount: i.tipAmount,
                author
            }
            return post
        }))
...
const uploadPost = async () => {
        if (!post) return
        let hash
        // Upload post to IPFS
        try {
            const result = await client.add(JSON.stringify({ post }))
            setLoading(true)
            hash = result.path
        } catch (error) {
            window.alert("ipfs image upload error: ", error)
        }
        // upload post to blockchain
        await (await contract.uploadPost(hash)).wait()
        loadPosts()
    }
```

And in INFURA Ethereum API, we add a dedicated gateway subdomain as such:



5/ Setting Up:

1. Clone/Download the Repository:

2. Install Dependencies:

```
npm install
```

3. Boot up local development blockchain:

`npx hardhat node`, and keep it running

4. Connect development blockchain accounts to Metamask:

- Copy private key of the first account and import to Metamask

```
souhabenhassine@Souhas-MacBook-Pro decentratwitter % npx hardhat node
You are using a version of Node.js that is not supported by Hardhat, and it may w

Please, make sure you are using a supported version of Node.js.

To learn more about which versions of Node.js are supported go to https://hardhat
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80
```

This account will be the deployer account and it'll allow us to sign transactions on our app.

Import as well account#1 and account#2.

- Connect your metamask to hardhat blockchain, network 127.0.0.1:8545.
- If you have not added hardhat to the list of networks on your metamask, open up a browser, click the fox icon, then click the top center dropdown button that lists all the available networks then click add networks. A form should pop up. For the "Network Name" field enter "Hardhat". For the "New RPC URL" field enter "http://127.0.0.1:8545". For the chain ID enter "31337". Then click save.

5. Run deploy script to migrate smart contracts:

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

Get the hash and run:

```
npx hardhat console --network localhost
```

```
>const contract= await ethers.getContractAt("Decentratwitter","your_hash")
```

`>contract` //displays the fetched contract

```
>const name = await contract.name()
```

```
>name
```

```
>.exit
```

6. Run Tests:

```
npx hardhat test
```

7. Launch Frontend

```
npm run start
```

6/ Demo Video:

BlockchainProjectDemo.mov https://drive.google.com/file/d/1b2lqcp5Q7g5jk1VJUXHAybWNXi-qjIV_/view?usp=drivesdk