**Blockchain in Healthcare**

Project by Masoumeh Jafari

This code is a smart contract written in the Solidity programming language. It defines a PatientData contract that stores information about patients and their transactions using a blockchain. The contract utilizes the OpenZeppelin libraries to import the Ownable and SafeERC20 contracts. Additionally, it imports the IERC20 interface to interact with ERC20 tokens.

The contract includes functions to add, remove, and retrieve patient data, such as profiles, transaction history, and doctor lists. A function for transferring transaction fees is also defined, utilizing the SafeERC20 library's safeTransferFrom function to ensure secure execution. Overall, this contract aims to provide a secure and transparent method for storing and sharing medical data using blockchain technology.

**Getting Started**

1. Go to [Backend](./backend/) to access backend part

2. Go to [frontend](./frontend/) to access frontend part

3. To watch project's description video, click here > [ِWatch]( https://mega.nz/file/xusHmJwC#I0R8EoojYoSxZ7FMWUGn6OU8WBZPS7Rtrhy8T81cEpw)

Explain each function in detail:

1. `constructor(address \_feeToken)`:

This is the constructor function that initializes the `patient` and `feeToken` variables. It takes an address `\_feeToken` as input representing the address of the ERC20 token contract to be used as the transaction fee token.

2. `isPatientOwner(address \_address)`:

This function takes an address `\_address` and returns a boolean indicating whether the address corresponds to the owner of the patient data or not.

3. `isDoctor(address \_address)`:

This function takes an address `\_address` and returns a boolean indicating whether the address corresponds to a doctor or not.

4. `addDoctor(address \_doctor)`:

This function adds a new doctor to the contract. It takes the address `\_doctor` as input and only the patient can call this function. The function checks if the doctor already exists and adds the doctor if not already present.

5. `removeDoctor(address doctorAddress)`:

This function removes the doctor from the contract. It takes an address `doctorAddress` as input and deletes the doctor from the mapping of doctors.

6. `updatePatientProfile(string memory \_profile)`:

This function updates the patient's profile. It takes a string `\_profile` as input and can be called by either the patient or their doctors.

7. `addTransaction(string memory \_transactionHash)`:

This function adds a new transaction to the patient's transaction history. It takes a string `\_transactionHash` as input and can be called by either the patient or their doctors.

8. `getTransactionCount()`:

This function returns the total number of transactions for the patient who calls the function.

9. `getTransactionHash(uint256 index)`:

This function returns the hash of a transaction at a given index in the patient's transaction history. It takes an index value as an input and can only be called by the patient or their doctors.

10. `patientProfile()`:

This function returns the profile of the patient who calls the function.

11. `showDoctorList()`:

This function returns the list of all doctors added by the patient.

12. `getPatient(address \_patient)`:

This function returns a `Patient` struct containing all the data related to the specified patient address `\_patient`.

13. `getDoctorCount()`:

This function returns the total number of doctors added by the patient.

14. `transferFee(address \_doctor, uint256 \_amount)`:

This function transfers the transaction fee to a doctor's address. It takes the `\_doctor` address and `\_amount` as inputs and can only be called by the patient. The function checks if the doctor address is valid and if the patient has sufficient funds before transferring the amount using the SafeERC20 library's `safeTransferFrom()` function

**Our Stack**

For this project, we utilized the following stack:

- Solidity Smart Contracts

- IPFS for storing image data

- Hardhat, Binance Smart Chain Testnet, and Sepolia for our development and testing framework

- JavaScript and HTML for our front-end development

- MetaMask as our web3 provider

**Prerequisites**

Before proceeding, ensure you have the following:

1. Metamask installed

2. Node.js installed

**Steps to Deploy and Interact with the Contract in Visual Studio Code**

1. Install Hardhat:

yarn add --dev hardhat

2. Compile the smart contracts:

yarn hardhat compile

3. Run the test file:

yarn hardhat test

4. Deploy the smart contracts:

yarn hardhat deploy

5. Deploy the smart contracts on Sepolia:

yarn hardhat deploy --network sepolia

**Steps to Deploy and Interact with the Contract in Remix**

1. Copy and paste the contract code on the [Remix Ethereum IDE](https://remix.ethereum.org/).

2. Run and deploy FeeToken (select compiler 0.8.20 or 0.8.21).

3. Set deployed-address FeeToken in Deploy segment (address\_feeToken) in patient contract.

4. Run and deploy patient (select compiler 0.8.20 or 0.8.21).