BLOCKCHAIN PROJECT

CRYPTODRIVE

SYSTEM MANUAL

Presented by

SWARNA KUMAR VUSA

14426869

Under the Guidance of

A S M Touhidul Hasan

At

UMKC

UNIVERSITY OF MISSOURI-KANSAS CITY

System Manual: CryptoDrive

Introduction

This project enables decentralized image upload and sharing on the blockchain, using Solidity for the smart contract and React for the front-end interface. Users can securely upload images to the InterPlanetary File System (IPFS) and share access with specific individuals through smart contract functionality. It is designed to allow for secure, transparent, and immutable image storage and sharing, with full ownership control via blockchain.

Features

1. Decentralized Storage:

- Images are uploaded to IPFS, ensuring they are securely stored on a decentralized network.
- Images cannot be tampered with or deleted by centralized authorities.

2. Smart Contract Integration:

- Solidity- based smart contracts on the Ethereum blockchain handle access control and ownership management.
- Users can control who accesses their uploaded images by interacting with the contract.

3. Access Control:

- Users can grant or revoke access to their images for specific Ethereum addresses through the smart contract.
- Access permissions are enforced via the smart contract's logic on the blockchain.

Technologies Used

1. **Solidity:** Smart contract development for decentralized access control.
2. **React:** Front-end application for user interaction, including uploading images and managing access.
3. **IPFS:** Decentralized storage for hosting images securely and immutably.
4. **Ethereum Blockchain:** For deploying and interacting with smart contracts.

Installation

1. Clone the Repository

To begin, clone the project repository to your local machine:

Bash

- ➔ git clone https://github.com/your-username/decentralized-image-upload.git

2. Install Hardhat Dependencies

Hardhat is a development environment for Solidity. In the root directory of your project, run:

Bash

- ➔ cd Dgdrive3.0
- ➔ npm install

3. Compile Smart Contracts

Compile the Solidity smart contracts to generate the necessary artifacts for deployment:

Bash

- ➔ npx hardhat compile

4. Deploy the Smart Contract

Deploy the smart contract to an Ethereum testnet (e.g., Rinkeby, Goerli) or a local development environment:

Bash

- ➔ npx hardhat run scripts/deploy.js --network <network-name>

Replace `<network-name>` with the name of your desired network.

5. Install React Dependencies

For the front-end interface, navigate to the client directory and install React dependencies:

Bash

- ➔ cd client
- ➔ npm install

6. Start the React Application

Once dependencies are installed, start the React application:

Bash

- ➔ npm start

This will launch the front-end app in your browser.

Configuration

1. Set Up Environment Variables

You will need an API key for Pinata (a service to interact with IPFS). Sign up for an account on [Pinata](https://www.pinata.cloud/) and obtain your API keys.

- In the project, navigate to the React component `FileUpload.js` and update the code with your Pinata API keys (replace the placeholders with your actual keys).

2. Update Smart Contract Address in React App

After deploying your smart contract to the Ethereum network, obtain the contract address. Then, in the App.js file within the React client directory, update the contract address with the correct value:

javascript

```
const contractAddress = "<YOUR_CONTRACT_ADDRESS>";
```

Usage

1. Install MetaMask

Ensure that MetaMask is installed and configured in your browser. MetaMask is a cryptocurrency wallet and gateway to blockchain apps. It will allow you to interact with your smart contract and manage Ethereum transactions.

- [Download MetaMask] (https://metamask.io/download/)

2. Upload an Image

To upload an image:

1. Click on the "Upload Image" button in the front-end interface.
2. The image will be uploaded to IPFS using Pinata, and the IPFS link will be stored.

3. Grant or Revoke Access to Image

Once an image is uploaded to IPFS:

- Grant Access: Use the smart contract's interface to grant access to a specific Ethereum address.
- Revoke Access: You can also revoke access for any specific address that you had granted permissions to.

4. Retrieve Image Data

To retrieve image data:

1. Click on the "Get Data" button.

2. Input the Ethereum address of the user whose images you want to access.

3. If you have been granted access via the smart contract, the image will be retrieved. If not, an error message will be displayed stating "You don't have access".

Note: You must upload an image to Pinata first before using the "Get Data" feature. Trying to retrieve data without an uploaded image will result in an error.

Troubleshooting

- "You don't have access" Error: This indicates that either:

  - You haven’t been granted access to the image through the smart contract, or

  - You are trying to retrieve data before uploading an image to IPFS.

  Ensure that the smart contract has granted you access, and that the image has been successfully uploaded to IPFS before attempting to retrieve it.

Security Considerations

- Private Keys: Never share your private keys or seed phrase for your Ethereum wallet. Always use MetaMask for handling transactions securely.

- Access Control: Ensure that only authorized users have access to your images by correctly using the smart contract to grant/revoke permissions.

Conclusion: This decentralized image upload and sharing system leverages blockchain technology to provide immutable, transparent, and secure image storage. By using Solidity smart contracts and IPFS, users can easily manage and share their images without relying on centralized cloud storage providers.