**HARDHAT install**

1-npm install --save-dev [hardhat@2.12.4](mailto:hardhat@2.12.4)

2-npx hardhat

**EXTRA dependencies**

3-npm install --save-dev "hardhat@^2.12.4" "@nomicfoundation/hardhat-toolbox@^2.0.0"

4-npm install create-react-app

5-npx create-react-app client

6-cd client / npm start

HARDHAT is like ganache. It gives environment to run local blockchain on our cli

npm install --save-dev @nomicfoundation/hardhat-toolbox @nomicfoundation/hardhat-network-helpers @nomicfoundation/hardhat-chai-matchers @nomiclabs/hardhat-ethers @nomiclabs/hardhat-etherscan chai ethers hardhat-gas-reporter solidity-coverage @typechain/hardhat typechain @typechain/ethers-v5 @ethersproject/abi @ethersproject/providers

To run it –

Npx hardhat node

--

When we will complile our code, artifacts folder will get created having ABI and bytecode. This is specified in hardhat.config specified in paths:

DEPLOY SMART CONTRACT

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

Upload.json has abi and bytecode

* AXIOS

Help in interaction with PINATA

* **const provider = new ethers.providers.Web3Provider(window.ethereum);**

The code snippet you provided initializes a new instance of the **Web3Provider** class from the **ethers.providers** module, using the **window.ethereum** object as the provider's source.

Here's a more detailed explanation of each part of the code:

* **const provider**: This line declares a new constant variable named **provider**. The **const** keyword indicates that the variable's value will not change throughout the program's execution.
* **new ethers.providers.Web3Provider()**: This line creates a new instance of the **Web3Provider** class, which is part of the **ethers.providers** module. This class is responsible for communicating with the Ethereum network via the Web3 API.
* **window.ethereum**: This is the parameter passed to the **Web3Provider** constructor. **window.ethereum** is a global variable provided by the MetaMask browser extension, which injects the Ethereum provider into the web page's JavaScript context. The provider is responsible for interacting with the Ethereum network on behalf of the user.

By combining these three lines of code, you are creating a new **Web3Provider** instance that will communicate with the Ethereum network via the **window.ethereum** provider. This is useful for building decentralized applications (dapps) that interact with the Ethereum blockchain, as it allows the dapp to securely connect to the user's MetaMask wallet and send/receive transactions.

* **Explain this code in detail useEffect(()=>{ const provider = new ethers.providers.Web3Provider(window.ethereum); const loadProvider=async()=>{ if(provider){ await provider.send("eth\_requestAccounts",[]); const signer = provider.getSigner(); const address = await signer.getAddress(); setAccount(address); let contractAddress = ""; const contract = new ethers.contract(contractAddress,Upload.abi,signer); console.log(contract) setContract(contract) setProvider(provider) }else{ console.error("Metamask is not installed") } }; provider && loadProvider() },[]);**

This code uses the **useEffect** hook in React, which runs a function when the component mounts.

Within the **useEffect** function, the code first creates an instance of the **Web3Provider** object from the **ethers** library. This object is used to interact with the Ethereum network.

Next, an asynchronous function called **loadProvider** is defined. This function checks if the **provider** object exists, which indicates that the user has the Metamask browser extension installed. If **provider** exists, the function sends an **eth\_requestAccounts** message to Metamask to request access to the user's Ethereum accounts. Once access is granted, the function gets the account address of the user's default account, sets it to the **account** state variable using the **setAccount** function, and sets an initial value for **contractAddress**.

The function then creates a new instance of the **ethers.contract** object using the **contractAddress**, the ABI of the **Upload** contract, and the signer object obtained from **provider.getSigner()**. The **ethers.contract** object is a wrapper around a smart contract on the Ethereum network, providing a convenient interface for interacting with the contract's functions and state variables.

Finally, the code sets the **contract** state variable and the **provider** state variable using the **setContract** and **setProvider** functions respectively. If **provider** does not exist, the code logs an error message to the console.

The second argument to **useEffect** is an empty array, indicating that the function should only be run once, when the component mounts. Overall, this code initializes the **provider**, **account**, and **contract** state variables, allowing the component to interact with the Ethereum network and the smart contract defined in the **Upload** ABI.

**signer** object is passed to the **ethers.Contract** constructor, which allows the **Contract** object to sign transactions on behalf of the user, using the **signer**'s private key. By signing transactions with the **signer**, the **Contract** object can perform actions on the Ethereum blockchain on behalf of the user who authorized the transaction.