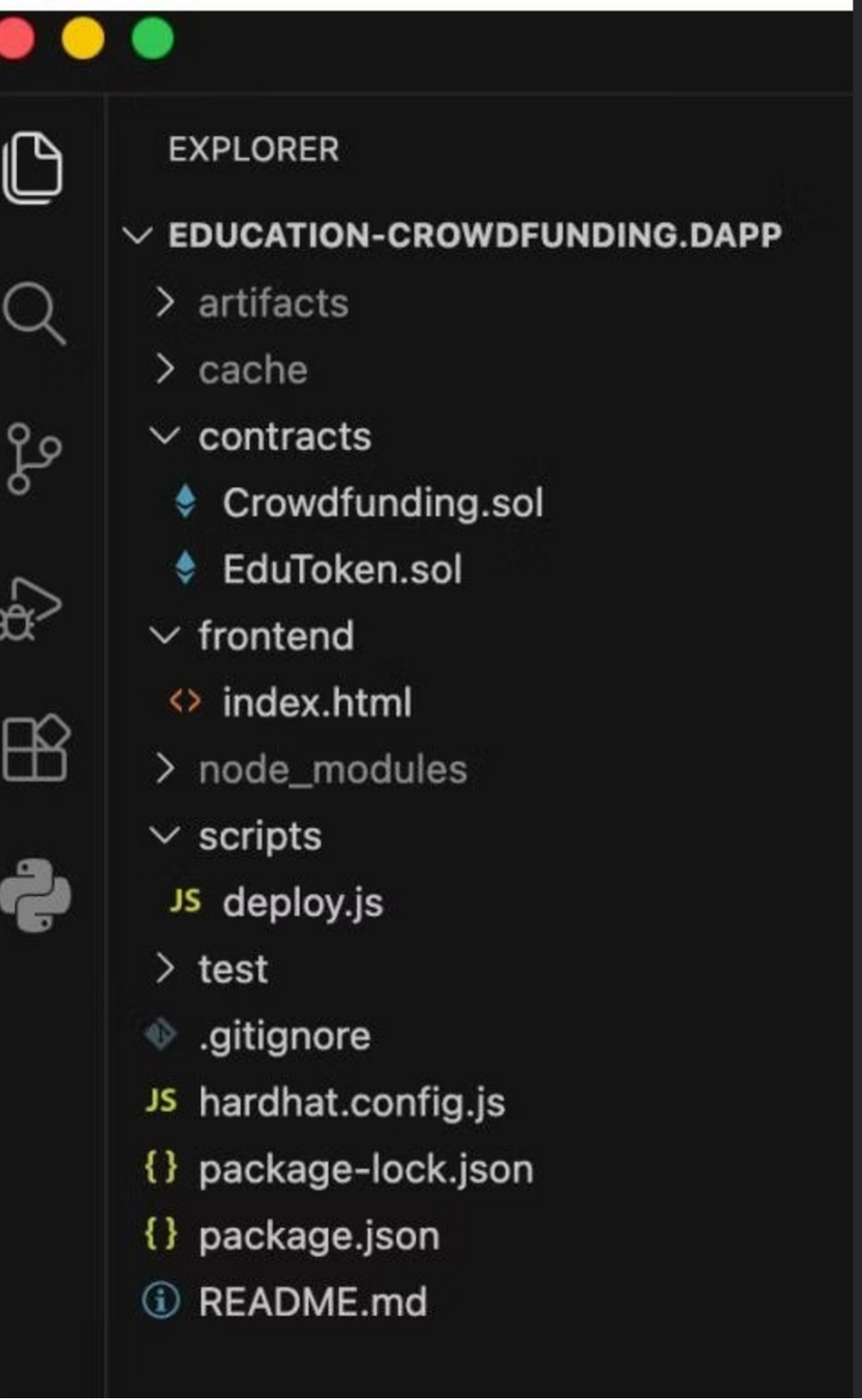
Blockchain Technologies: Education Crowdfunding DApp

Presented by: Zarina Kerimbay, Akbota Bekturgan, Rustam Saduakas



Application Architecture: Three-Tier DApp

Ourdecentralizedapplication (DApp)followsastandard three-tier architecture,ensuringrobust interactionbetweenusers andthe blockchain.

1

Frontend (The Client)

Built with frameworks like React, Vue, or Next.js, handling the UI and user input.

2

Provider/Wallet (The Bridge)

Tools like MetaMask or WalletConnect act as the signer and gateway to the blockchain.

3

Smart Contracts (The Backend)

Written in Solidity and deployed on a blockchain (e.g., Ethereum Sepolia), serving as the immutable database and logic layer.

Project Structure: EDUCATION-CROWDFUNDING.DAPP

TheVisual StudioCodeExplorerpanelillustratestheorganizedstructureofourDApp, with key directories for contracts, frontend, anddeploymentscripts.

- artifacts
- cache
- contracts (Crowdfunding.sol, EduToken.sol)
- frontend (index.html)
- node_modules
- scripts (deploy.js)
- test
- .gitignore
- hardhat.config.js
- package-lock.json
- package.json
- README.md

Design & Implementation Decisions

Our DApp prioritizes transparency, efficiency, and user-friendliness through strategic design choices.

Smart Contract Core

Core logic implemented in smart contracts for transparency and immutability.

Modular & Minimal

Contracts are kept modular and minimal to reduce gas costs.

Test Network Usage

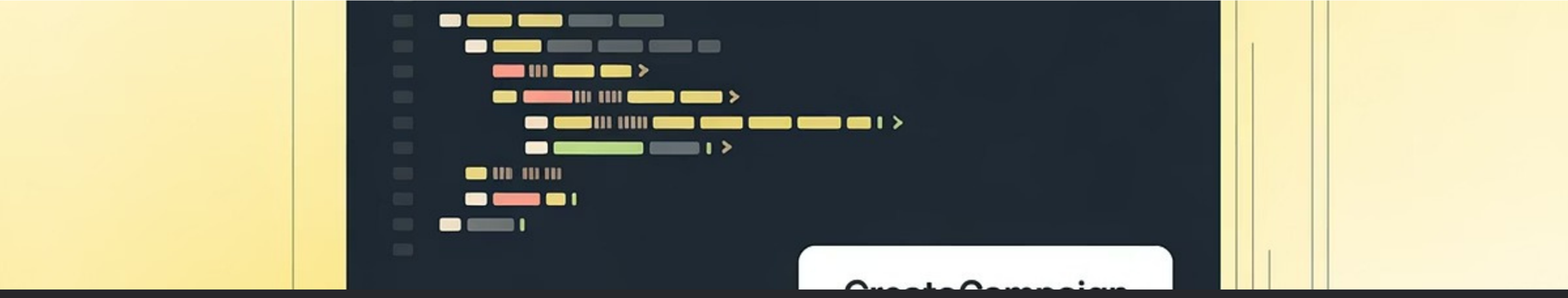
A test network (Sepolia) is used for safe and cost-effective testing.

MetaMask Integration

MetaMask selected for wallet integration due to its wide adoption and ease of use.

Secure Frontend

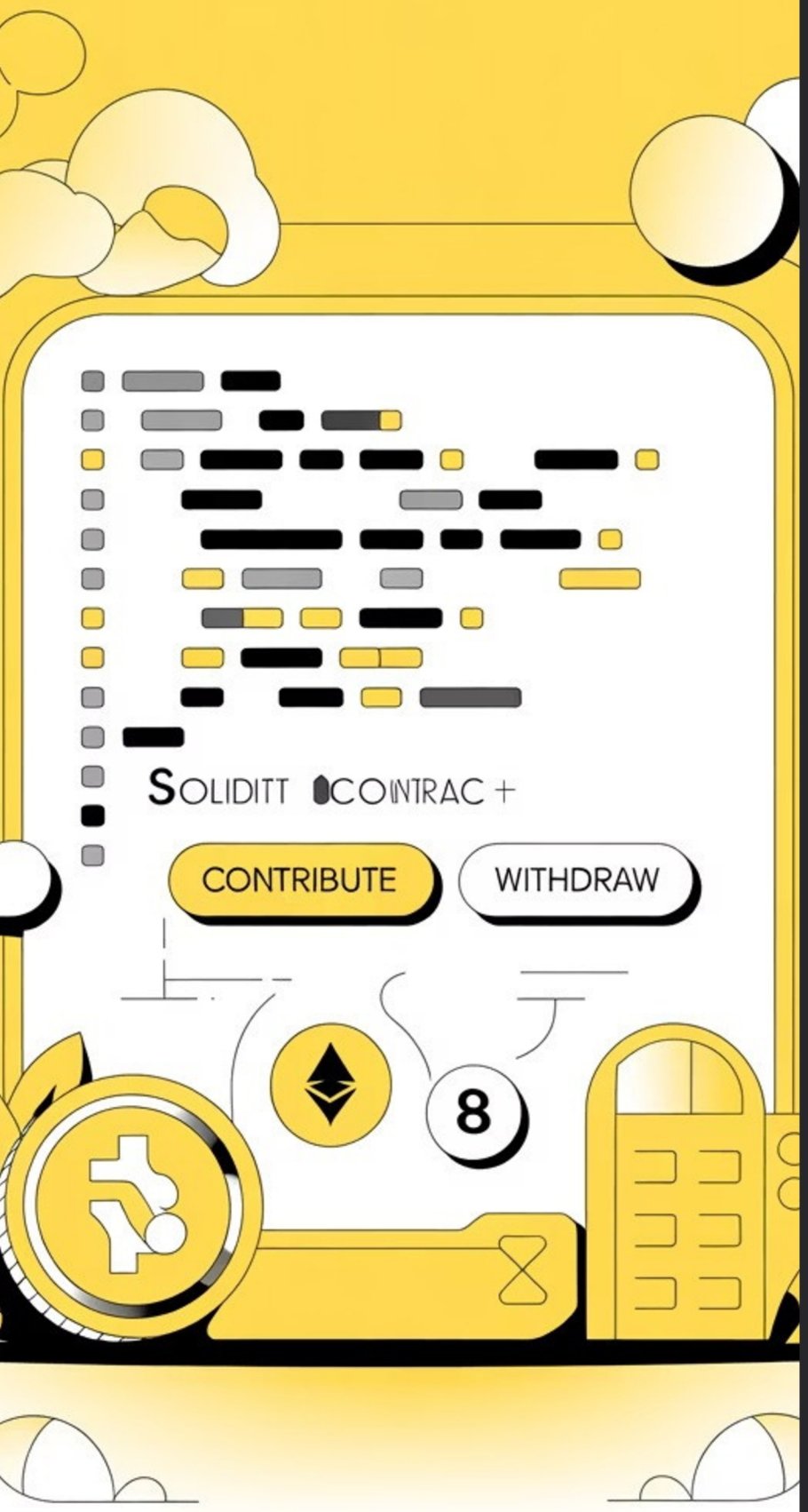
Frontend does not store sensitive data; all critical data is retrieved directly from the blockchain.



Crowdfunding Smart Contract: createCampaign

The `Crowdfunding.sol` contractdefinesthecorelogicforcampaigncreation,allowinguserstoinitiatenewfundraisingefforts on the blockchain.

```
// SPDX-License-Identifier: MIT
pragma solidity "0.8.20";

import "./EduToken.sol";

contract Crowdfunding {
 struct Campaign {
 address owner;
 string title;
 uint goal;
 uint deadline;
 uint amountRaised;
 bool withdrawn;
 }

 EduToken public token;
 Campaign[] public campaigns;

 mapping(uint => mapping(address => uint)) public contributions;

constructor(address tokenAddress) {
token = EduToken(tokenAddress);
}

function createCampaign(string memory _title, uint _goal, uint _duration) public {
campaigns.push(Campaign({
owner: msg.sender,
title: _title,
goal: _goal,
deadline: block.timestamp + _duration,
amountRaised: 0,
withdrawn: false
}));
}
```



Crowdfunding Smart Contract: contribute & withdraw

The contribute functionenablesuserstodonatetocampaigns, while the withdrawfunction allowscampaignownerstoretrievefunds once the goal is met and the deadline passed.

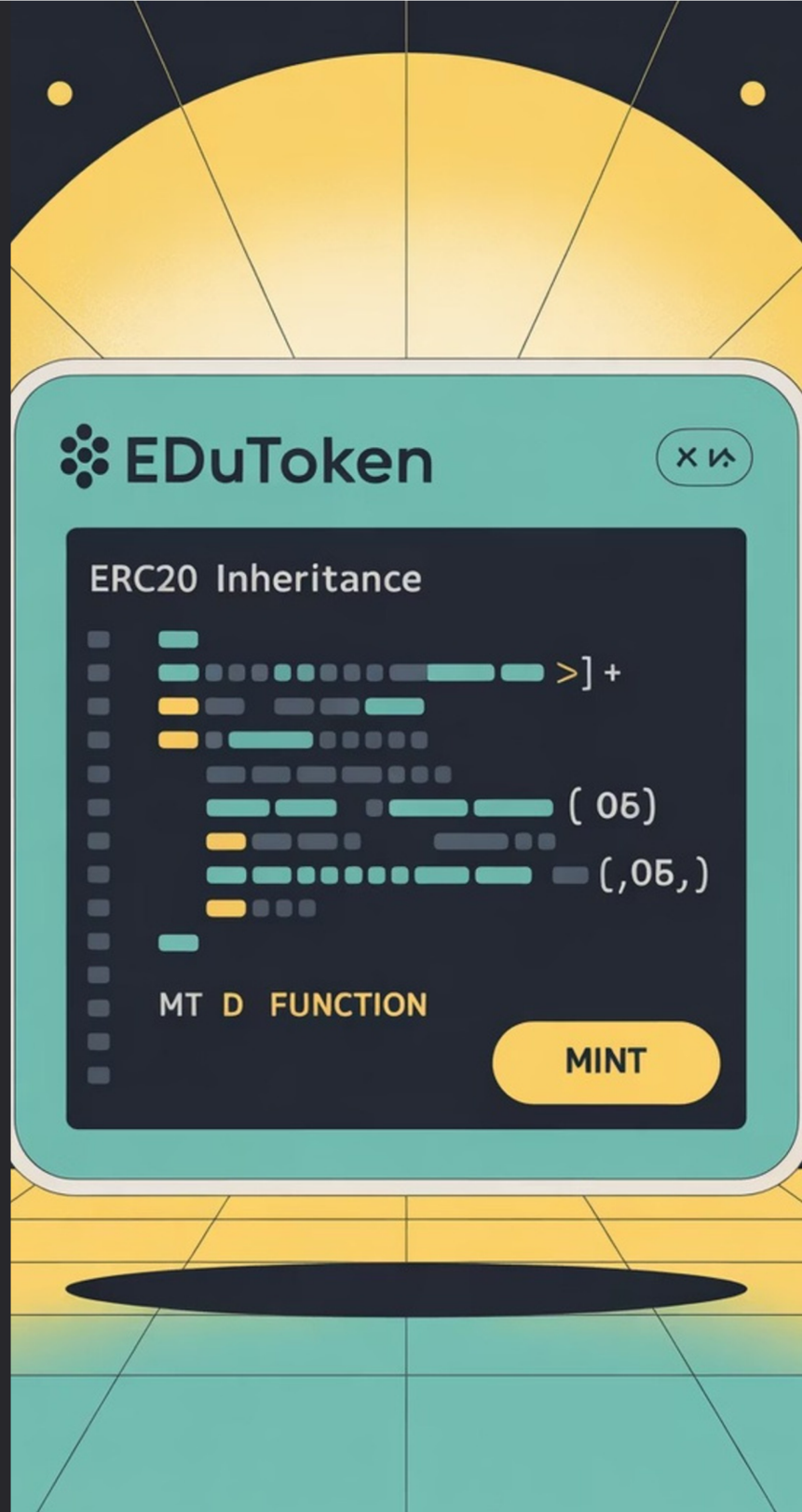
```
function contribute(uint id) public payable {
Campaign storage c = campaigns[id];
require(block.timestamp < c.deadline, "Ended!");

c.amountRaised += msg.value;
contributions[id][msg.sender] += msg.value;

token.mint(msg.sender, msg.value * 100);
}

function withdraw(uint id) public {
Campaign storage c = campaigns[id];
require(msg.sender == c.owner, "Not owner!");
require(c.amountRaised >= c.goal, "Goal not reached!");
require(block.timestamp >= c.deadline, "Not finished!");
require(!c.withdrawn, "Already!");

c.withdrawn = true;
payable(c.owner).transfer(c.amountRaised);
}
```

EduToken Smart Contract & Logic

The `EduToken.sol` contract, inheritingfrom ERC20andOwnable,defines our customeducationtoken. Itincludesa `mint`function,controlledbythe contract owner, to issue new tokens.

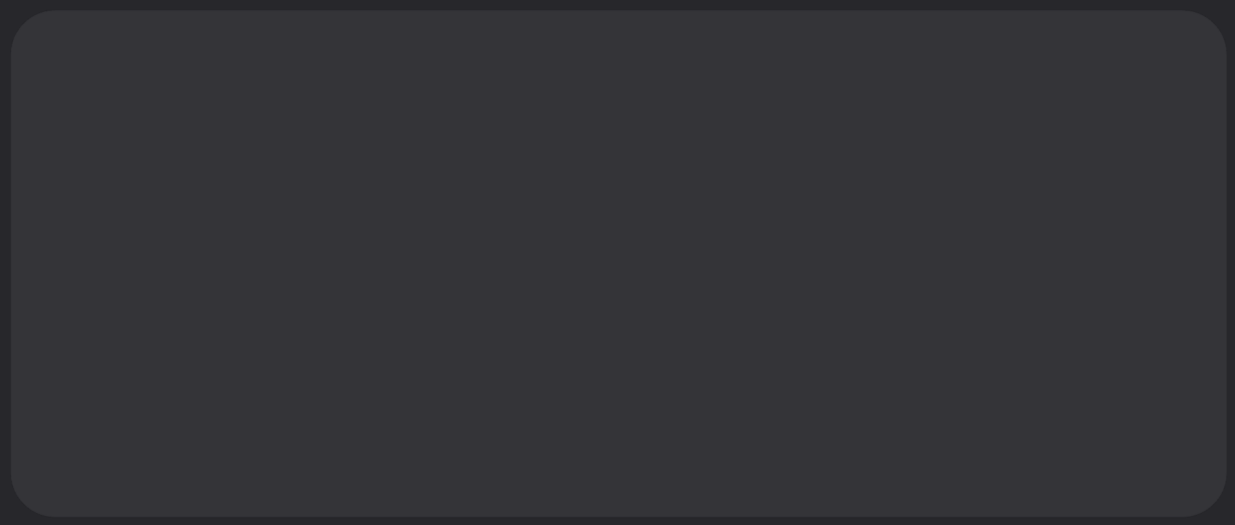
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract EduToken is ERC20, Ownable {
    constructor() ERC20("Education Token", "EDU")
Ownable(msg.sender) {}

    function mint(address to, uint amount) public onlyOwner {
        _mint(to, amount);
    }
}
```

The smart contract logic ensures data integrity, validates transactions, and manages access control, emitting events for frontend notifications.



Frontend Interaction: index.html

The `index.html` file formsthe userinterface,enablinginteraction with the DApp.ItincludesMetaMask connection,balancedisplays, and forms for campaign creation and donations.



Education Crowdfunding DApp

Connect MetaMask

Address: Not connected

ETH Balances: 0

EDU Token Balance: 0

Create Campaign

Create

Donate

Donate

Campaign List

Refresh List

```
index.html ×   Crowdfunding.sol
ntend > <> index.html > html > body >
<html>
<body>
<script>

async function createCampaign() {
  try {
    const title = document.getEleme
    const goalEth = document.getEle
    const goalWei = ethers.utils.pa
    const duration = 86400 * 7; //

    const tx = await crowdfundingCo
    console.log("Pending...");
    await tx.wait();
    alert("Campaign Created!");
  } catch (err) {
    console.error(err);
    alert("Error: " + err.reason);
  }
}

async function donate() {
  try {
    const id = document.getElementB
    const amountEth = document.getE

    const tx = await crowdfundingCo
      value: ethers.utils.parseEthe
    });
    console.log("Donating...");
    await tx.wait();

    alert("Donation Successful!");
    const address = await signer.ge
    updateBalances(address, signer.
  } catch (err) {
    console.error(err);
    alert("Error: " + err.reason);
  }
}
```

Frontend-to-Blockchain Interaction Flow

The DAppfacilitates seamlessinteraction betweentheuserinterface and the blockchain, from wallet connection to transaction confirmation.

This structured flow ensures secure and transparent operations, with MetaMask acting as the crucial bridge for user authentication and transaction signing.

Deployment & Execution Instructions

Follow these steps to deploy the smart contracts and run the frontend application.

01

Install Dependencies

Run `npm install` in both the root and frontend directories.

02

CompileSmart Contracts

Execute `npm hardhat compile` to compile the Solidity contracts.

03

Deploy Contracts

Deploy to a test network using `npm hardhat run scripts/deploy.js --network sepolia`.

04

Start Frontend Server

Navigate to the frontend directory and run `npm run dev`.

05

Connect MetaMask

Open the application in a browser and connect your MetaMask wallet.

06

Obtain Test ETH

Switch to a test network (e.g., Sepolia) in MetaMask and request test ETH from a faucet.