**CrowdFunding DApp Documentation**

**1. Project Overview**

The CrowdFunding DApp is a decentralized crowdfunding platform built on Ethereum. It enables users to create fundraising campaigns, donate Ether to campaigns, and claim refunds if a campaign fails to meet its target by the deadline. The system consists of:

* **Smart Contract**: CrowdFunding.sol handles campaign creation, donations, and refund logic.
* **Frontend**: A React.js application (src/App.js) interacting with the smart contract via ethers.js and MetaMask.

**Key Components**

1. **Campaign Lifecycle**:
   * **Creation**: Users define campaign details — title, description, funding goal, deadline, and image URL.
   * **Donation**: Anyone can donate Ether to an active campaign before its deadline.
   * **Refund**: Donors can claim refunds if the campaign ends without reaching its target.
2. **Data Structures**:
   * Campaign struct stores campaign metadata, donations, and manages per-donor balances.
   * Mappings and arrays track campaigns and contributions securely on-chain.

**2. Functionality**

**Smart Contract (CrowdFunding.sol)**

* **createCampaign**
  + **Signature**: function createCampaign(address owner, string title, string description, uint256 target, uint256 deadline, string image) public returns (uint256)
  + **Behavior**: Registers a new campaign; returns its ID.
  + **Revert Conditions**: Deadline must be in the future.
* **donateToCampaign**
  + **Signature**: function donateToCampaign(uint256 id) public payable
  + **Behavior**: Accepts Ether; updates campaign’s amountCollected and donor’s balance.
  + **Revert Conditions**: Campaign must be active (before deadline); donation must be >0.
* **getDonators**
  + **Signature**: function getDonators(uint256 id) public view returns (address[] memory, uint256[] memory)
  + **Behavior**: Returns arrays of donor addresses and their contributed amounts for a campaign.
* **getCampaigns**
  + **Signature**: function getCampaigns() public view returns (Campaign[] memory)
  + **Behavior**: Returns list of all campaigns with their fields.
* **claimRefund**
  + **Signature**: function claimRefund(uint256 id) external
  + **Behavior**: Allows donors to reclaim contributions if the campaign deadline has passed without meeting the funding target.
  + **Revert Conditions**: Must be called after deadline; campaign must have failed; donor must have a positive balance.

**Frontend (React + ethers.js)**

* **Wallet Connection**: Uses ethers.BrowserProvider to connect MetaMask and obtain a signer.
* **Campaign Management**:
  + **Create Campaign**: Form collects campaign data; normalized Ether amounts using ethers.parseEther.
  + **List Campaigns**: Fetches campaign list and donation data; displays title, description, goal, collected amount, deadline, and image.
  + **Donate**: Prompt-based donation input; invokes donateToCampaign with specified Ether amount.
  + **Claim Refund**: Shows "Claim Refund" button for campaigns that failed; invokes claimRefund to return funds.

**3. Prerequisites**

* **Node.js** v16+ and npm
* **Hardhat** (or Truffle) for smart contract compilation and deployment
* **MetaMask** browser extension configured for your local or testnet environment

**4. Smart Contract Deployment**

1. **Install Dependencies**:

npm install --save-dev hardhat @nomiclabs/hardhat-ethers ethers

1. **Configure Hardhat**: Create hardhat.config.js with network settings (e.g., localhost, Goerli).
2. **Write Deployment Script** (scripts/deploy.js):

const { ethers } = require("hardhat");

async function main() {

    const CrowdFunding = await ethers.getContractFactory("CrowdFunding");

    const crowdFunding = await CrowdFunding.deploy();

    console.log("CrowdFunding deployed to:", crowdFunding.target);

}

main()

    .then(() => process.exit(0))

    .catch((error) => {

        console.error(error);

        process.exit(1);

    });

1. **Deploy**:

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

1. **Note Address**: Copy the deployed contract address (42-character hex starting with 0x).

**5. Frontend Setup and Run**

1. **Install Dependencies**:

cd src

npm install react react-dom ethers

1. **Configure src/App.js**:
   * Replace contractAddress with your deployed address.
   * Paste the complete ABI array (including claimRefund) into the abi constant.
2. **Run Application**:

npm start

1. **Use the DApp**:
   * Open in browser, click **Connect Wallet**.
   * **Create Campaign** by filling form and clicking **Create**.
   * **Donate** to listed campaigns; **Claim Refund** if eligible.

**6. Notes and Best Practices**

* **Security**: Use OpenZeppelin’s ReentrancyGuard if adding more complex features.
* **Gas Optimization**: For high traffic, consider batch fetching and event logs instead of looping on-chain.
* **Production**: Deploy to a testnet before mainnet; verify contract on Etherscan and use Infura/Alchemy for provider.

*Enjoy building decentralized crowdfunding!*