SCHOOL OF INFORMATION,COMPUTER AND COMMUNICATION TECHNOLOGY
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY THAMMASAT UNIVERSITY

DES484: Blockchain Development

Final Project

Title: Crowdfunding Decentralized application

BY

Smit Qiao	6322774363
Kittapat Trumjalern	6322771922
Warit Thongyoung	6322773779

Table of contents

Introduction to the Blockchain Crowdfunding dApp Project

Project Overview

This project is about making a simple app using blockchain technology for crowdfunding. The main goal is to create an app that makes raising funds easier, safer, and more open for everyone. Unlike usual fundraising methods, this app will use blockchain, which is a kind of digital record-keeping, to make things more trustworthy and straightforward.

Purpose

The idea behind this app is to make a space where people who want to start new projects can get money from others who are willing to support them. What makes this app special is that it uses blockchain, which means that every transaction is recorded clearly. This helps everyone involved to see where the money is going and builds trust.

Target Audience

This app is meant for two groups of people. First, it's for people with creative ideas or projects who need money to make them happen. Second, it's for people who want to give money to support these ideas. The app is designed to be easy to use, whether someone knows a lot about technology or not. It's not just for people in one place, but for users all over the world.

In short, this app is a new way to bring together people who need funding and those who want to fund them, using technology that's safe and easy to understand. It's all about making crowdfunding better and more accessible for everyone.

Understanding Blockchain Technology in the dApp

What is Blockchain?

Blockchain is a type of technology that keeps records of transactions on multiple computers at the same time. It's like a digital ledger that is very secure and hard to change once something is recorded. This technology is known for being very safe and open because everyone can see the transactions that have been made.

Why Use Blockchain for Crowdfunding?

Using blockchain for crowdfunding has many benefits. It makes the whole process very transparent, meaning everyone can see where the money is going. It's also very secure, which helps prevent fraud and makes sure that the money goes to the right place. Plus, because it doesn't need a central system or a middleman, it can be faster and cheaper than traditional ways of raising money.

Key Features of Blockchain in Our dApp

Decentralization: Our app doesn't rely on one single point of control. This means it's less likely to have problems like system failures or fraud.
Transparency: Every transaction is visible to everyone, which builds trust among users.
Security: Blockchain is known for being very secure. It uses complex cryptography, which makes it very hard for hackers to tamper with the data.
Efficiency: With blockchain, transactions can be processed faster and with lower costs compared to traditional methods.

Choosing the Right Blockchain Platform

For our crowdfunding app, we'll pick a blockchain platform that supports smart contracts (which are like automatic agreements) and is known for being stable and user-friendly. This could be a platform like Ethereum, which is popular and has a good track record. The choice will depend on factors like how easy it is to use, how secure it is, and how much it costs to use.

In summary, this section focuses on the benefits of using blockchain, like safety, openness, and efficiency, and talks about what we need to think about when picking a blockchain platform for the app.

The Crowdfunding Model of the dApp

Solidity Smart Contract for Crowdfunding

Environment for Development: We use a Solidity development environment like Remix IDE for writing and deploying this contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Crowdfunding {
    struct Campaign {
        address payable creator;
        string description;
        uint goal;
        uint pledged;
        uint end;
        bool claimed;
    }
    uint numCampaigns;
    mapping(uint => Campaign) public campaigns;
    event CampaignStarted(uint campaignId, address creator, string description, uint goal, uint duration);
    event Pledged(uint campaignId, address donor, uint amount);
    event Claimed(uint campaignId);

    function startCampaign(string memory description, uint goal, uint duration) public {
        require(goal > 0, "Goal should be more than 0");
        require(duration > 0 && duration <= 30 days, "Duration should be 1 to 30 days");

        uint end = block.timestamp + duration;
        campaigns[numCampaigns] = Campaign(payable(msg.sender), description, goal, 0, end, false);
        emit CampaignStarted(numCampaigns, msg.sender, description, goal, duration);
        numCampaigns++;
    }
    function pledge(uint campaignId) public payable {
        require(campaignId < numCampaigns, "Campaign does not exist");
        Campaign storage campaign = campaigns[campaignId];
        require(block.timestamp < campaign.end, "Campaign finished");
        require(msg.value > 0, "Pledge amount must be more than 0");

        campaign.pledged += msg.value;
        emit Pledged(campaignId, msg.sender, msg.value);
    }
    function claimFunds(uint campaignId) public {
        require(campaignId < numCampaigns, "Campaign does not exist");
        Campaign storage campaign = campaigns[campaignId];
        require(msg.sender == campaign.creator, "Not campaign creator");
        require(block.timestamp >= campaign.end, "Campaign not finished");
        require(!campaign.claimed, "Funds already claimed");
        require(campaign.pledged >= campaign.goal, "Goal not reached");

        campaign.claimed = true;
        campaign.creator.transfer(campaign.pledged);
        emit Claimed(campaignId);
    }

    function getCampaign(uint campaignId) public view returns (
        address creator,
        string memory description,
        uint goal,
        uint pledged,
        uint end,
        bool claimed
    ) {
        require(campaignId < numCampaigns, "Campaign does not exist");
        Campaign storage campaign = campaigns[campaignId];
        return (
            campaign.creator,
            campaign.description,
            campaign.goal,
            campaign.pledged,
            campaign.end,
            campaign.claimed
        );
    }
```

(Figure 1.1)

Development & Testing:
Open Remix IDE.
We write this code into Crowdfunding.sol. Located in the Remix IDE
Then compile and test the contract using Remix's built-in tools.
Deployment:
Deploy the contract to an Binance smartchain testnet using Remix due the cost of ETH is quite high..
After deployment, we note the contract address and ABI (Application Binary Interface) for use in the front-end part of the DApp.
Interaction:
Users can start campaigns, pledge funds, and claim funds through this contract.
The contract includes necessary checks and balances to ensure the correct flow of actions.

HTML and CSS for the Front-End Interface

The following code is used in Visual Studio Code.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Crowdfunding DApp</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f4;
      text-align: center;
      padding: 20px;
    }
    .container {
      width: 80%;
      margin: auto;
      overflow: hidden;
    }
    .campaign {
      background: #fff;
      border: #ddd solid 1px;
      border-radius: 10px;
      padding: 15px;
      margin-bottom: 10px;
    }
    .campaign h3 {
      margin-top: 0;
    }
    .campaign-info {
      text-align: left;
    }
    button {
      background: #333;
      color: #fff;
      border: none;
      padding: 10px 20px;
      margin: 10px 0;
      border-radius: 5px;
      cursor: pointer;
    }
    button:hover {
      background: #555;
    }
  </style>
```

```
</head>
<body>
  <div class="container">
    <h1>Crowdfunding DApp</h1>
    <p>Connect to your Ethereum wallet to start using the DApp</p>
    <button id="walletButton">Connect Wallet</button>

    <h2>Start a Campaign</h2>
    <input type="text" id="campaignDescription" placeholder="Campaign Description">
    <input type="number" id="campaignGoal" placeholder="Campaign Goal (in ETH)">
    <input type="number" id="campaignDuration" placeholder="Duration (in days)">
    <button id="startCampaignButton">Start Campaign</button>

    <h2>Current Campaigns</h2>
    <div id="campaigns">
      <!-- Campaigns will be loaded here -->
    </div>
  </div>

  <script src="https://cdn.jsdelivr.net/npm/ethers/dist/ethers.min.js"></script>
  <script src="app.js"></script>
</body>
</html>
```

(Figure 1.2)

Explanation and Usage:
HTML Structure:
The page includes a button to connect to an Ethereum wallet (like MetaMask).
There are input fields and a button to start a new campaign.
A section is dedicated to displaying current campaigns.
CSS Styling:
The CSS is included within the <style> tag in the head of the HTML document.
It styles the page and elements for a basic, clean layout.
Running the Front-End:

We save this code in a file named index.html in the project directory.
This HTML file will be the main interface for users to interact with your crowdfunding platform.
The JavaScript part, which will be the final piece, will handle connecting this front-end with the Ethereum smart contract, enabling users to interact with the blockchain.

JavaScript Code for Front-End Logic

The following code is used in Visual Studio Code.

```
const contractAddress = "YOUR_CONTRACT_ADDRESS";
const contractABI = [/* Your Contract ABI Here */];

let provider;
let signer;
let contract;

window.onload = function() {
  document.getElementById('walletButton').addEventListener('click', connectWallet);
  document.getElementById('startCampaignButton').addEventListener('click', startCampaign);
};
async function connectWallet() {
  if (typeof window.ethereum !== 'undefined') {
    try {
      await window.ethereum.request({ method: 'eth_requestAccounts' });
      provider = new ethers.providers.Web3Provider(window.ethereum);
      signer = provider.getSigner();
      contract = new ethers.Contract(contractAddress, contractABI, signer);
      console.log("Wallet connected!");
    } catch (error) {
      console.error("Could not connect to wallet", error);
    }
  } else {
    console.error("Metamask not found");
  }
}
async function startCampaign() {
  const description = document.getElementById('campaignDescription').value;
  const goal = ethers.utils.parseEther(document.getElementById('campaignGoal').value);
  const duration = document.getElementById('campaignDuration').value * 24 * 60 * 60; // Convert days to seconds
  try {
    const tx = await contract.startCampaign(description, goal, duration);
    await tx.wait();
    console.log("Campaign started!");
  } catch (error) {
    console.error("Error starting campaign", error);
  }
}

async function loadCampaigns() {
}
```

(Figure 1.3)

<u>*Connecting to the Wallet:*</u>

The script connects to a user's Ethereum wallet using MetaMask or similar.
connectWallet function is called when the user clicks the 'Connect Wallet' button.
Starting a New Campaign:
The startCampaign function is executed when the 'Start Campaign' button is clicked.
It reads the values from the input fields and sends a transaction to the smart contract.
Interacting with the Contract:
The contract variable represents your deployed smart contract and is used to call its functions.
Loading Campaigns:
Implement loadCampaigns to fetch and display active campaigns from the smart contract. This part of the code will depend on how you decide to structure the display of campaigns on your webpage.
Running the Script:
Save this code in a file named app.js in the same directory as your index.html.
Replace YOUR_CONTRACT_ADDRESS with the actual address of your deployed contract.
Replace /* Your Contract ABI Here */ with the ABI from your Solidity contract.
Testing and Deployment:
Test this thoroughly in a local environment and on Ethereum test networks before deploying to the mainnet.

Ensure MetaMask or another web3 provider is installed in the browser for interaction.
This JavaScript code, along with the HTML/CSS from Part 2 and the Solidity contract, completes the crowdfunding DApp.

Smart Contracts in the Crowdfunding dApp - A Code-Based Explanation

The crowdfunding decentralized application (dApp) uses a smart contract, written in Solidity, to manage its campaigns. The previous code (Figure 1.1) snippet outlines the structure and functionality of this smart contract.

Key Components of the Smart Contract

Campaign Structure: The Campaign struct defines the basic structure of a crowdfunding campaign. It includes:

creator: The Ethereum address of the campaign creator.
description: A brief overview of the campaign.
goal: The fundraising goal in terms of Ether.
pledged: The amount of Ether currently pledged.
end: The timestamp when the campaign ends.
claimed: A boolean to check if the funds have been claimed.
Campaign Storage: mapping(uint => Campaign) public campaigns creates a way to store and access campaigns using an ID. Each campaign is assigned a unique ID.

Core Functions of the Contract

<u>*startCampaign:*</u>
Creators use this to start a new campaign.
It checks that the goal and duration are valid.
The campaign is saved with details like creator, description, goal, and duration.
An event CampaignStarted is emitted for transparency.
<u>*pledge:*</u>
Allows donors to pledge Ether to campaigns.
Checks that the campaign exists and is still active.
The pledged amount is added to the campaign's total.
An event Pledged is emitted for each donation.
<u>*claimFunds:*</u>
Enables the creator to claim funds after the campaign ends.
Checks that the campaign exists, the caller is the creator, the campaign has ended, the goal is reached, and funds haven't been claimed before.
Transfers the pledged funds to the creator.
An event Claimed is emitted once funds are claimed.

getCampaign:
A public view function to get details about a specific campaign.
Validates the campaign ID and returns the campaign details.
Events for Transparency and Tracking
The contract uses Ethereum events (CampaignStarted, Pledged, Claimed) to log significant actions. These events are crucial for transparency and can be monitored by the front-end of the dApp to update users about campaign progress in real-time.

Security and Efficiency

The contract includes checks (require statements) to prevent invalid actions, such as pledging to a finished campaign or claiming funds prematurely. This ensures that the contract behaves as intended and safeguards users' funds.

User Experience

From the user's perspective, campaign creators can easily start and manage campaigns, while backers can pledge and track the progress of their contributions. The contract automates most processes, ensuring a smooth and secure crowdfunding experience.
In summary, the smart contract outlined in the provided code is a fundamental component of the crowdfunding dApp. It automates key processes, enforces rules, and ensures transparency and security in crowdfunding activities. This functionality is central to providing a reliable and user-friendly platform for both project creators and backers.

User Interface and Experience (UI/UX)

Overview of the Web Interface Design

The previous HTML and JavaScript code (Figure 1.2, Figure 1.3) gives a clear structure to the user interface (UI) of the crowdfunding dApp. It outlines a simple, user-friendly web page designed for interacting with the blockchain-based crowdfunding application.

Design Aesthetics and Layout

Simple and Clean Design: The UI uses a minimalist design with a light background and a straightforward layout. This makes it easy for users to focus on the key elements without being overwhelmed.

Consistent Styling: The use of Arial font and consistent color scheme (gray, black, and white) across the page ensures a cohesive look and feel.

Responsive Container: The .container class indicates that the UI is designed to be responsive, adjusting to different screen sizes for optimal viewing on various devices.

Campaign Display and Interaction

Campaign Section: The .campaign class styles each campaign box, making it visually distinct. This helps users to easily differentiate between different campaigns.

Clear Call to Action: Buttons are styled to be prominent and inviting, encouraging user interaction, such as connecting a wallet or starting a campaign.

Functionality Based on JavaScript Code

Wallet Connection: The JavaScript code handles the connection to the user's Ethereum wallet, which is essential for interacting with the dApp.

Starting a Campaign: Users can start a campaign by entering its description, goal, and duration. The startCampaign function communicates with the Ethereum blockchain to record these details.

Error Handling: The code includes error handling for wallet connection and campaign creation, ensuring that users are aware of any issues that occur.

User Interaction and Experience

Intuitive Navigation: The page layout, with clear headings and input fields, makes it easy for users to understand how to interact with the dApp.

Real-time Feedback: The JavaScript console logs provide real-time feedback on actions like wallet connection and campaign creation, though these could be enhanced with user-friendly notifications or alerts in a production environment.

Placeholder Text: Input fields contain placeholder text guiding users on the required information, improving usability.

Areas for Improvement and Expansion

Load Campaign Functionality: The loadCampaigns function is yet to be implemented. Once completed, it would dynamically display ongoing campaigns, enhancing user engagement.

Accessibility Features: Additional features could be added for accessibility, such as alt text for images (if any) and keyboard navigation support.

Educational Content: Given the technical nature of blockchain and smart contracts, integrating tooltips or help sections explaining terms and processes would be beneficial for users new to the technology.

The HTML and JavaScript code provide a solid foundation for the UI/UX of the crowdfunding dApp. The design is clean and user-friendly, and the functionality for wallet connection and campaign creation is well-implemented. Further development, particularly in campaign display and user education, would enhance the overall experience and accessibility of the application.

Regulatory Compliance in the Crowdfunding dApp

Importance of Compliance

For the crowdfunding decentralized application (dApp), adhering to regulatory compliance is essential. It ensures the platform operates legally and maintains user trust.

Key Compliance Areas

Know Your Customer (KYC) and Anti-Money Laundering (AML):
Implementing KYC procedures to verify user identities, crucial for preventing fraud and meeting AML regulations.

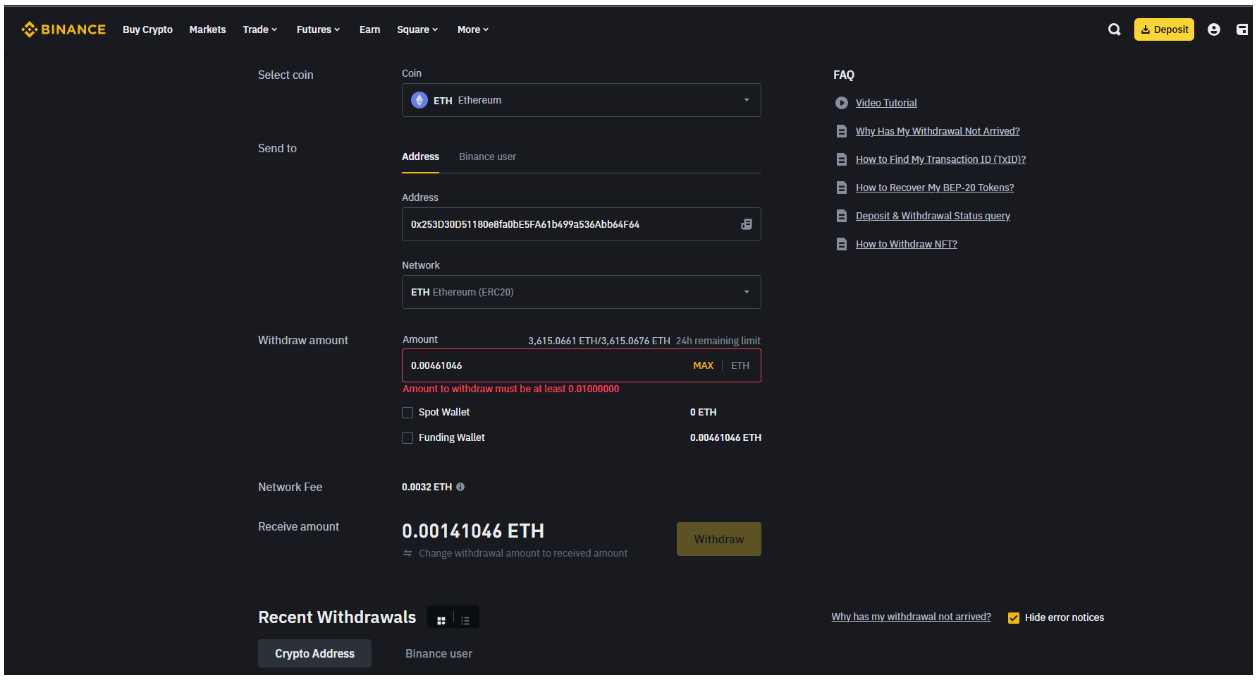
Crowdfunding Regulations:
Complying with specific rules in different countries related to fundraising, investor qualifications, and disclosures.

Data Privacy and Protection:
Ensuring user data privacy, in line with regulations like the GDPR, through robust encryption and secure data handling practices.

Maintaining regulatory compliance is vital for the dApp's legitimacy and user confidence. By focusing on critical areas like KYC, AML, crowdfunding laws, and data protection, the platform can operate effectively within legal frameworks.

Final Product

Challenge of High Ethereum Gas Fees



Select coin — Coin: ETH Ethereum

Send to — Address | Binance user

Address: 0x253D30D51180e8fa0bE5FA61b499a536Abb64F64

Network: ETH Ethereum (ERC20)

Withdraw amount — Amount: 0.00461046 ETH — 3,615.0661 ETH/3,615.0676 ETH 24h remaining limit

Amount to withdraw must be at least 0.01000000

Spot Wallet 0 ETH

Funding Wallet 0.00461046 ETH

Network Fee: 0.0032 ETH

Receive amount: 0.00141046 ETH

Change withdrawal amount to received amount

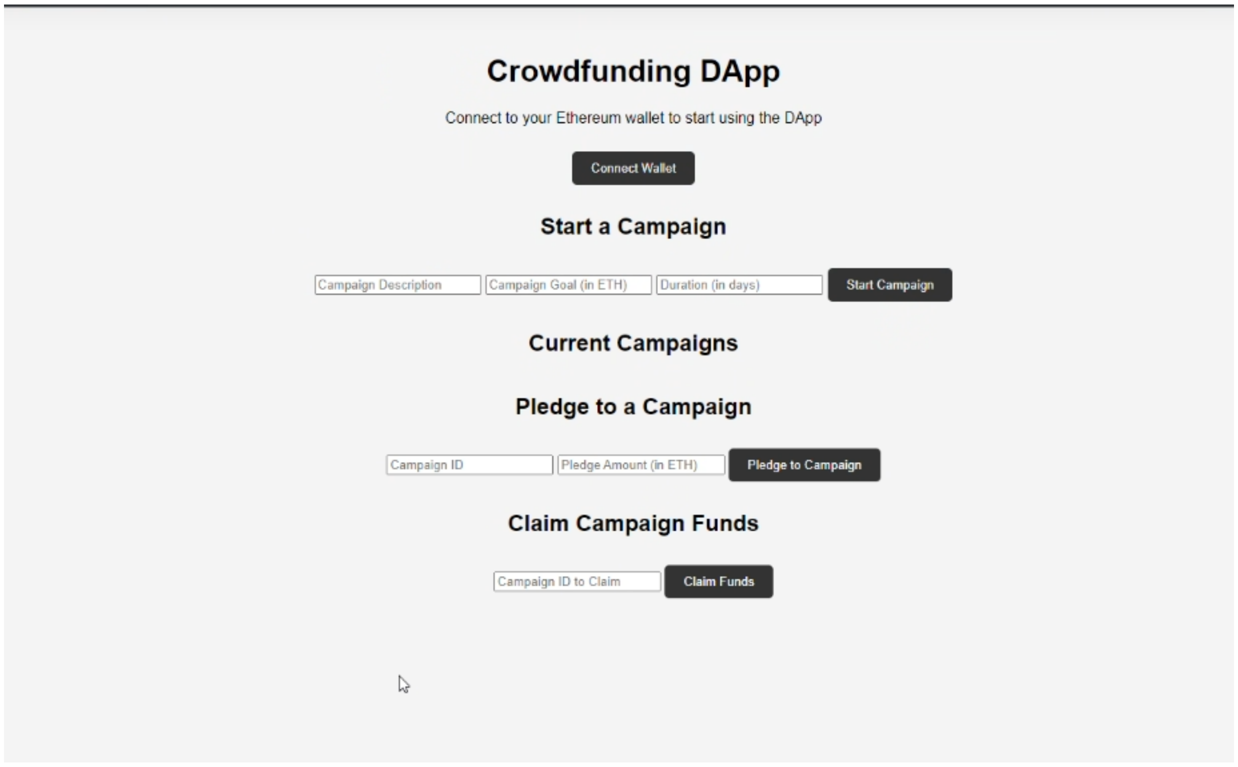
Recent Withdrawals

FAQ
- Video Tutorial
- Why Has My Withdrawal Not Arrived?
- How to Find My Transaction ID (TxID)?
- How to Recover My BEP-20 Tokens?
- Deposit & Withdrawal Status query
- How to Withdraw NFT?

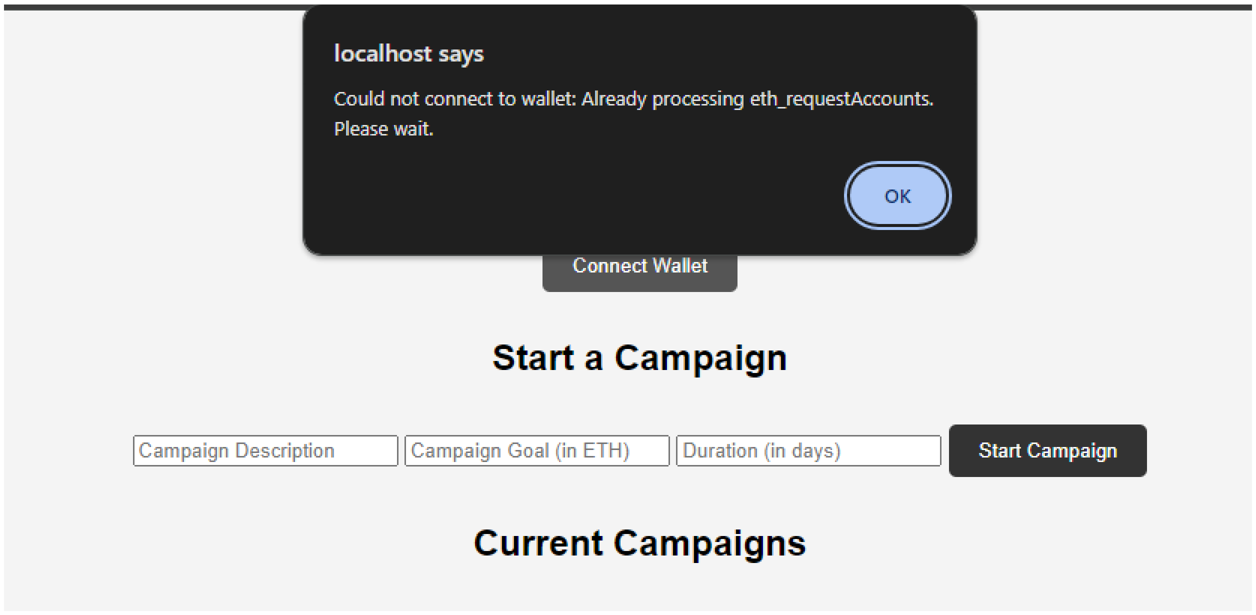
Why has my withdrawal not arrived? [✓] Hide error notices

(Figure 2.1)

While the deployment of the Crowdfunding Wallet contract was successful on the testnet, a significant hurdle has been encountered in its transition to the Ethereum mainnet. This issue arises due to the substantially higher gas fees on the mainnet, which are currently too expensive for transactions as small as 0.01 ETH. The escalated cost of gas on the mainnet, compared to the testnet, has presented a considerable obstacle to deploying our contract effectively in a real-world scenario.

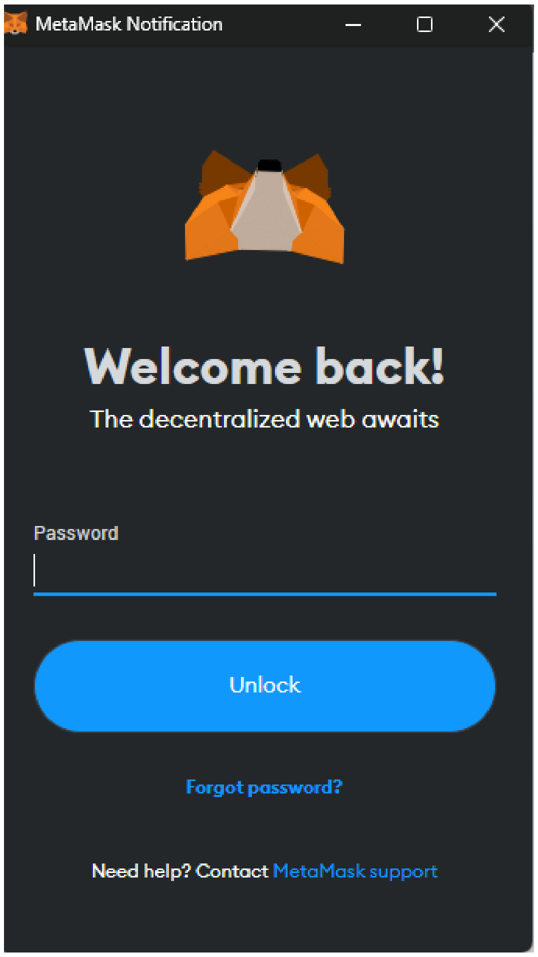


(Figure 2.2)

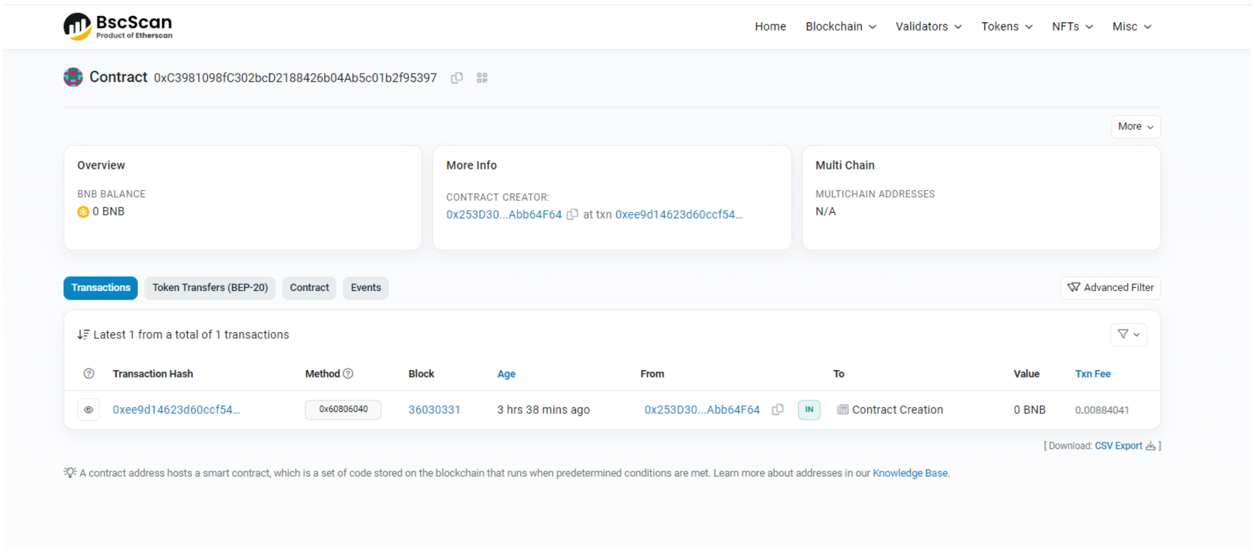


Could not connect to wallet: Already processing eth_requestAccounts. Please wait.

(Figure 2.3)



(Figure 2.4)



Contract 0xC3981098fC302bcD2188426b04Ab5c01b2f95397

Overview

BNB BALANCE
0 BNB

More Info

CONTRACT CREATOR:
0x253D30...Abb64F64 at txn 0xee9d14623d60ccf54...

Multi Chain

MULTICHAIN ADDRESSES
N/A

Latest 1 from a total of 1 transactions

	Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
	0xee9d14623d60ccf54...	0x60806040	36030331	3 hrs 38 mins ago	0x253D30...Abb64F64	IN	Contract Creation	0 BNB	0.00884041

[Download: CSV Export]

A contract address hosts a smart contract, which is a set of code stored on the blockchain that runs when predetermined conditions are met. Learn more about addresses in our Knowledge Base.

(Figure 2.5)

Our crowd funding wallet:
https://testnet.bscscan.com/address/0xC3981098fC302bcD2188426b04Ab5c01b2f95397 (Figure 2.5)

As outlined in figure 2.2, the first step is to connect your Ethereum wallet by clicking the 'Connect to Wallet' button. This easy step is crucial for secure transactions and interactions on our platform. If there's an issue, an error pop-up, as shown in figure 2.3, will guide you through resolving it. For first-time users, a pop-up shown in figure 2.4 will facilitate a seamless connection to your MetaMask wallet. Returning users will be greeted with a welcome back screen each time they reconnect their wallet. Once your wallet is connected, you're all set to initiate your own crowdfunding campaign.

Creating a campaign is straightforward. Begin by filling in the 'Campaign Description' field with a detailed and compelling narrative of your project and how the funds will be utilized. It's essential to be clear and honest here, as transparency fosters trust among potential backers. Next, set your 'Campaign Goal' in Ethereum (ETH) to specify the amount of funding you're aiming to raise. It's important to be realistic in this aspect, as achievable goals are more likely to attract contributions.

Additionally, determine the 'Duration' of your campaign in days. This should reflect the time you believe will be needed to reach your funding goal, while also keeping your backers interested and engaged.

After providing all the necessary details, click the 'Start Campaign' button to submit your project for review. Our platform will then verify that your campaign meets our community guidelines and standards before it goes live.

Once your campaign is live, it becomes visible to a broad audience of potential backers. They can contribute directly using their Ethereum wallets. To make a pledge to a campaign, backers simply choose a project they believe in and contribute the desired amount of ETH. Each contribution is recorded and reflected in the campaign's progress.

As the campaign progresses, the project creator can claim the raised funds if the campaign reaches its goal within the specified duration. This is done through the 'Claim Funds' feature, which securely transfers the accumulated contributions to the project creator's wallet.

It's important to actively manage and update your campaign throughout its duration, engaging with backers and keeping them informed. This active engagement is key to a successful crowdfunding experience, as it builds trust and fosters a community around your project.

Challenges and Solutions in Developing the Crowdfunding dApp

While developing a blockchain-based crowdfunding dApp offers numerous benefits, it also presents unique challenges. Identifying and addressing these challenges is crucial for the successful implementation and adoption of the platform.

Challenge 1: Scalability

Issue: Blockchain networks, especially those handling numerous transactions, can face scalability issues, leading to slower transaction times and higher costs.

Solution: Implementing scalability solutions like Layer 2 protocols or choosing a blockchain platform known for handling high transaction volumes efficiently.

Challenge 2: User Adoption

Issue: Convincing users accustomed to traditional crowdfunding platforms to switch to a blockchain-based solution can be challenging.

Solution: Educating potential users about the benefits of blockchain, such as increased security and transparency. Simplifying the user interface to make the transition as seamless as possible for all users.

Challenge 3: Technical Complexity

Issue: The technical nature of blockchain and smart contracts might be daunting for some users and project creators.

Solution: Providing comprehensive user guides, tutorials, and customer support to assist users in navigating the platform. Including features in the dApp that simplify complex processes.

Challenge 4: Regulatory Uncertainty

Issue: The regulatory landscape for blockchain and crowdfunding is still evolving, which can create uncertainty.

Solution: Staying informed about regulatory changes and being flexible to adapt the platform accordingly. Engaging with legal experts to ensure ongoing compliance.

Challenge 5: Security Concerns

Issue: Ensuring the security of smart contracts and user funds is critical, given the risks of hacks and vulnerabilities in blockchain platforms.

Solution: Conducting thorough security audits and regular vulnerability assessments of the smart contracts. Implementing robust security measures for the platform.

Addressing these challenges is essential for the success of the crowdfunding dApp. By focusing on scalability, user adoption, simplifying technical complexities, staying updated with regulations, and ensuring top-notch security, the dApp can provide a reliable, efficient, and user-friendly crowdfunding platform.

Conclusion

Summary of the dApp's Journey

The development of the blockchain-based crowdfunding decentralized application (dApp) marks a significant milestone in leveraging technology to enhance and democratize the funding landscape. This journey has involved careful planning and execution in various aspects, including technology adoption, user interface design, regulatory compliance, and addressing potential challenges.

Key Achievements

Innovative Use of Blockchain: Integrating blockchain technology to provide a secure, transparent, and efficient platform for crowdfunding.

User-Centric Design: Focusing on an intuitive and accessible user interface to cater to both seasoned blockchain users and newcomers.

Addressing Regulatory Compliance: Ensuring that the dApp adheres to existing legal frameworks and is prepared to adapt to future regulatory changes.

Final Thoughts

The crowdfunding dApp represents not just a technological innovation but a commitment to empowering individuals and organizations by providing them with a more accessible and transparent funding platform. As the dApp continues to grow and evolve, it will undoubtedly face new challenges and opportunities, but its core mission will remain the same: to revolutionize the way we think about and engage in crowdfunding.

In conclusion, this dApp is set to make a significant impact in the crowdfunding domain, offering a unique blend of technological sophistication and user-friendly features. It stands as a testament to the potential of blockchain technology in creating more open, fair, and efficient systems for financial transactions and community engagement.

Reference

https://chat.openai.com/