INTRODUCTION TO BLOCKCHAIN TECHNOLOGY
DECENTRALISED CROWDFUNDING

Under the Guidance of: -

Dr. Raghu Vamsi Potukuchi

Submitted By: -

Ananya Kapoor 20103104 B4
Dhairya Sachdeva 20103098 B4

Department of CSE / IT
Jaypee Institute of Information Technology, Noida

May 2023

Acknowledgement

I would like to place on record my deep sense of gratitude to Dr. Raghu Vamsi Potukuchi, Assistant Professor, Jaypee Institute of Information Technology, India for his generous guidance, help and useful suggestions.

I also wish to extend my thanks to friends and other classmates for their insightful comments and constructive suggestions to improve the quality of this project work.

Ananya Kapoor

20103104

Dhairya Sachdeva

20103098

Table of Contents

1. Abstract

This project aims to explore the growing potential of the Blockchain technology to replace centralised applications to provide an efficient and secure experience in context to Crowd Funding. The ultimate objective of the project is to develop, deploy and access the crowd funding platform using Ethereum blockchain, smart contracts, MetaMask, Hardhat, web3.js, JavaScript.
The methodology includes development of a smart contract in Solidity as the backbone of the Crowd Funding platform, a Graphical User interface using JavaScript, HTML, CSS, and Hardhat and web3.js for testing and deployment.

2. Problem statement

Traditionally Crowd Funding platforms are centralised, and often rely on 3rd party intermediaries to facilitate transactions between the sender and receiver. High Fee, Lack of trust, limited to no transparency and a potential for fraudulent behaviour are some of the many drawbacks of this centralised approach. Additionally, the load on centralised systems often causes delays and unforeseen circumstances disturbing the flow of transactional data.
To address and overcome these challenges, a need to explore a more decentralised approach in the form of a blockchain based Crowd Funding Application is felt.
By utilising the concepts of Smart contracts in the Ethereum blockchain an overall efficient, hassle free, transparent, secure and most importantly a decentralised application (dApp) can be designed to implement and deploy on the Ethereum Blockchain.

3. Introduction

3.1 General

Blockchain operates on a distributed network of computers, that collectively maintain and validate the integrity of the ledger. This distributed nature ensures that no single entity has complete control over the system.

Traditional crowdfunding platforms typically rely on centralized intermediaries to facilitate transactions and ensure trust between campaign creators and backers.

Smart contracts, which are self-executing contracts with predefined rules and conditions encoded on the blockchain, play a crucial role in enabling decentralized crowdfunding.

The use of tools such as Hardhat, MetaMask, JavaScript, and Solidity further empowers us to create robust and secure decentralized applications (DApps) on blockchain platforms like Ethereum. Hardhat provides a development environment and testing framework for Ethereum smart contracts, while MetaMask serves as a browser extension that enables users to interact with DApps seamlessly. JavaScript is a versatile programming language used for building interactive user interfaces, and Solidity is a programming language specifically designed for writing smart contracts on Ethereum.

3.2 Specific

The project focuses on the implementation of a crowdfunding platform using blockchain technology, specifically leveraging the Ethereum blockchain by utilizing the fundamentals of smart contract, deployment using web3.js and hardhat and the MetaMask chrome extension to sign the transactions processed by the smart contract.

4. Literature Survey

The papers we referred for the making of this project along with its summary is:

1. Title: " The Applications of Blockchain Technology in Crowdfunding Contract " Authors: Hongjiang Zhao and Cephas Paa Kwasi Coffie Summary: This paper investigates a conceptual framework that can provide the solution to the problems related to Crowdfunding contracts using blockchain technology. Considering the role of the intermediary platforms, they have examined how the foundational qualities of the blockchain technology may resolve the problems of these platforms.
2. Title: " Supportroops: Crowdfunding Using Blockchain" Authors: Moiyad Kaydawala, Abhinav Pandey, Parnika Roy. Summary: This paper provides a platform for businesses, entrepreneurs, NGOs, and customary people to host online fundraising campaigns with the

assistance of blockchain technology which may be of any nature (non-profit cause, personal cause, business related).

3. Title: "Crowd-Funding Using Blockchain Technology" Authors: Prof D. L. Falak, Soudagar Shanawaz, Jadhav Pranav, Katke Kajal, Shukla Utkarsh. Summary: The work of this paper is to provide interactive forms for campaign creation, donation and request approval through which both campaign creators and donors can easily create and fund the campaigns.
4. Title: "Cryptocurrency Rewards and Crowdsourcing Task Success" Authors: Shan Meng and Xia Zhao. Summary: This study investigates how the use of cryptocurrency rewards, i.e., the choices of stablecoins and unstable coins affects the crowdsourcing task success, and how the relationship depends on task difficulty.

5. Scope of the project

The project's scope includes the development and deployment of an Ethereum blockchain-based crowdfunding platform, covering a smart contract, frontend components and the connection between the two to produce a functional and user-friendly application.

6. System Architecture

6.1 Tools and Technology

1. Ethereum Blockchain
2. Solidity: Solidity is a programming language specifically designed for writing smart contracts on the Ethereum platform.
3. Hardhat: Hardhat is a development environment and testing framework for Ethereum smart contracts.
4. MetaMask: It serves as a digital wallet and connects to the frontend of project.

5. JavaScript: In this project, JavaScript is utilized to build the frontend interface of the crowdfunding platform. It enables the creation of an interactive and user-friendly interface that connects with the smart contract.
6. Web3.js: It allows to deploy the smart contract.

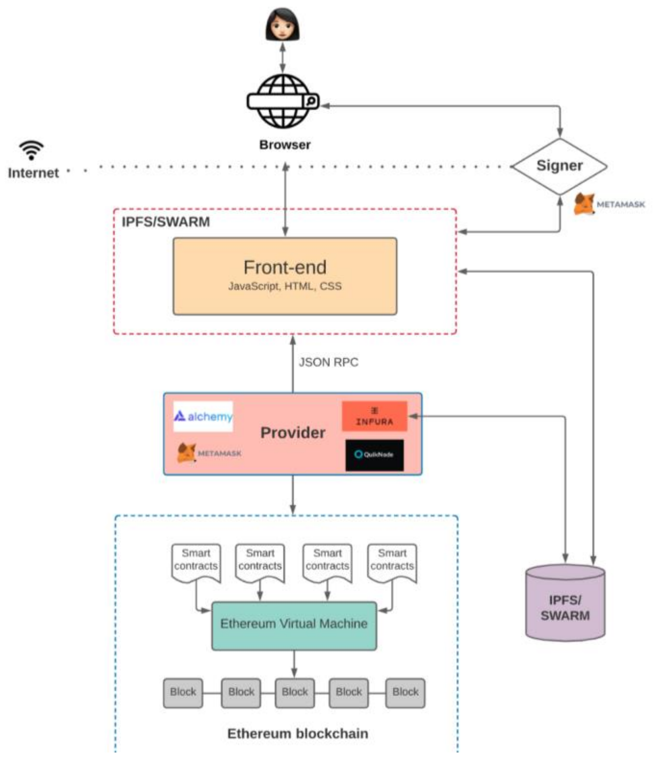
6.2 Design of the project

The key aspects of the project design are as follows:

1. Smart Contract Design: The smart contract is designed using Solidity. It includes functions for crowdfunding.
2. User Interface Design: It includes features such as project creation, backing projects, deleting projects, and viewing details of the project.
3. Integration with MetaMask: MetaMask enables us to connect to the Ethereum blockchain, connect to digital wallets and sign transactions in a secure manner.

6.3 System Architecture

1. Ethereum Blockchain: The project utilizes the Ethereum blockchain as the underlying distributed ledger.
2. Smart Contracts: They are written in Solidity and deployed on the Ethereum blockchain. The smart contracts define the logic for crowdfunding projects.



The Architecture of a Web 3.0 application

3. Frontend: The frontend interface connects to the Ethereum blockchain using Web3.js library and communicates with the deployed smart contracts to perform actions such as creating campaigns, contributing funds, and tracking campaign progress.
4. MetaMask: MetaMask browser extension, acts as a digital wallet and enables secure interactions with the Ethereum blockchain.
5. Testing and Deployment: The project utilizes Hardhat, a development environment and testing framework, for compiling, deploying, and testing the smart contracts.

6.4 Smart Contract Design

The smart contract design of the project is based on the SPDX-License-Identifier: MIT license. The contract has the following key components and functionalities:

1. Contract Variables: The contract includes various state variables such as **owner** (address of the contract owner), **projectTax** (percentage of project tax), **projectCount** (total count of projects), **balance** (contract balance), **stats** (a struct containing statistics about the project), and **projects** (an array to store project details).

2. Structs: The contract defines three structs - **statsStruct** (to store project statistics), **backerStruct** (to store backer details), and **projectStruct** (to store project details).
3. Enum: The contract includes an **statusEnum** enum that represents the status of a project, including OPEN, APPROVED, REVERTED, DELETED, and PAIDOUT.
4. Modifiers: The contract defines a modifier **ownerOnly** that restricts certain functions to be executed only by the contract owner.
5. Events: The contract emits an **Action** event when certain actions are performed, such as project creation, update, deletion, backing, and payout.
6. Constructor: The contract constructor initializes the contract owner and sets the initial value for the project tax.
7. Functions: The contract includes various functions to create projects, update project details, delete projects, back projects with contributions, perform refunds, perform payouts, change project tax, and retrieve project details.
8. Internal Functions: The contract includes internal functions **performRefund** and **performPayout** to handle refund and payout operations respectively.
9. Additional Function: The contract includes a utility function **payTo** to handle the transfer of funds to a specified address.

6. Implementation details

1. Importing ABI and contract address: The code imports the contract ABI (Application Binary Interface) and the contract address from separate JSON files. The ABI defines the interface of the smart contract, including its functions and events.
2. Authentication via MetaMask: The code checks for the presence of the MetaMask extension (**ethereum** object) in the browser window. It provides two functions, **connectWallet** and **isWalletConnected**, to connect to MetaMask and retrieve the connected Ethereum account.
3. Ethereum contract initialization: The **getEtheriumContract** function retrieves the connected Ethereum account and creates an instance of the Ethereum contract using the

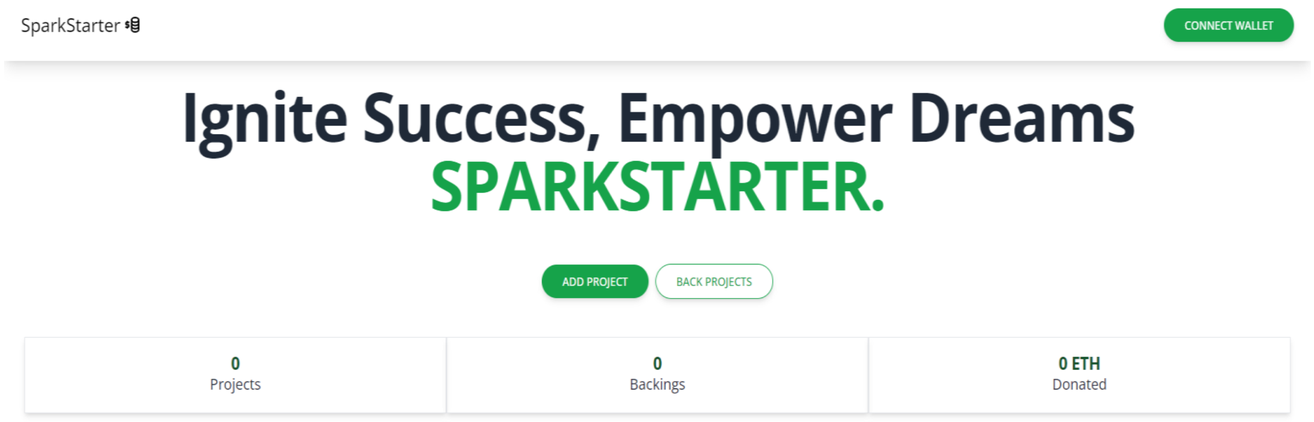
ethers.js library. It returns the contract object that can be used to interact with the smart contract functions.

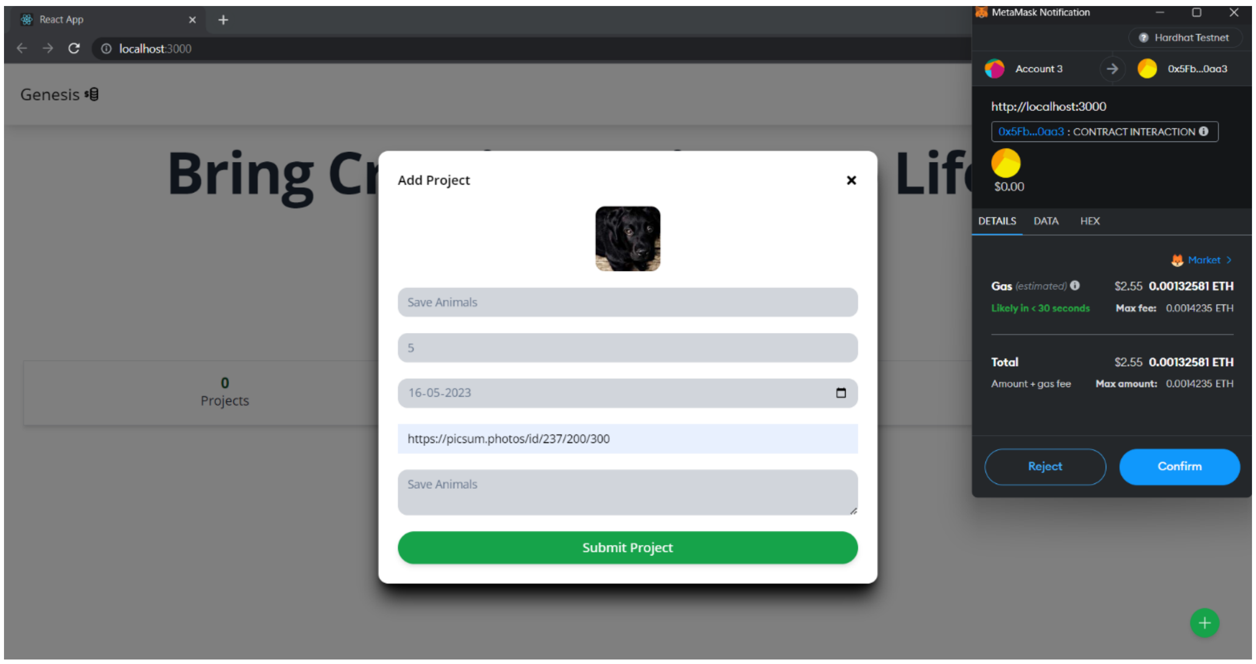
4. Contract interaction functions: The code provides various functions to interact with the smart contract. These functions include **createProject** for creating a new project, **updateProject** for updating an existing project, **deleteProject** for deleting a project, **loadProjects** for retrieving all projects, **loadProject** for retrieving a specific project, **backProject** for contributing to a project, and **payoutProject** for initiating a payout for a project.
5. Use of Structures: The code includes functions (**structuredBackers**, **structuredProjects**, **structureStats**, **toDate**) to represent the data received from the smart contract functions into a desired format.
6. Error handling: The code includes a **reportError** function that logs the error message and throws a new error if the **ethereum** object is not available.

7.1Code

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

contract Genesis {
  address public owner;
  uint public projectTax;
  uint public projectCount;
  uint public balance;
  statsStruct public stats;
  projectStruct[] projects;

  mapping(address => projectStruct[]) projectsOf;
  mapping(uint => backerStruct[]) backersOf;
  mapping(uint => bool) public projectExist;

  enum statusEnum {
    OPEN,
    APPROVED,
    REVERTED,
    DELETED,
    PAIDOUT
  }

  struct statsStruct {
    uint totalProjects;
    uint totalBacking;
    uint totalDonations;
  }

  struct backerStruct {
    address owner;
    uint contribution;
    uint timestamp;
    bool refunded;
  }

  struct projectStruct {
    uint id;
    address owner;
    string title;
    string description;
    string imageURL;
    uint cost;
    uint raised;
    uint timestamp;
    uint expiresAt;
    uint backers;
    statusEnum status;
  }

  modifier ownerOnly() {
    require(msg.sender == owner, "Owner reserved
only");
    _;
  }

  event Action(
    uint256 id,
    string actionType,
    address indexed executor,
    uint256 timestamp
  );

  constructor(uint _projectTax) {
    owner = msg.sender;
    projectTax = _projectTax;
  }

  function createProject(
    string memory title,
    string memory description,
    string memory imageURL,
    uint cost,
    uint expiresAt
  ) public returns (bool) {
    require(bytes(title).length > 0, "Title cannot be
empty");
    require(bytes(description).length > 0,
"Description cannot be empty");
    require(bytes(imageURL).length > 0, "ImageURL
cannot be empty");
    require(cost > 0 ether, "Cost cannot be zero");

    projectStruct memory project;
    project.id = projectCount;
    project.owner = msg.sender;
    project.title = title;
    project.description = description;
    project.imageURL = imageURL;
    project.cost = cost;
    project.timestamp = block.timestamp;
    project.expiresAt = expiresAt;

    projects.push(project);
    projectExist[projectCount] = true;
    projectsOf[msg.sender].push(project);
    stats.totalProjects += 1;

    emit Action(
      projectCount++,
      "PROJECT CREATED",
      msg.sender,
      block.timestamp
    );
    return true;
  }

  function updateProject(
    uint id,
    string memory title,
    string memory description,
    string memory imageURL,
    uint expiresAt
  ) public returns (bool) {
    require(msg.sender == projects[id].owner,
"Unauthorized Entity");
    require(bytes(title).length > 0, "Title cannot be
empty");
    require(bytes(description).length > 0,
"Description cannot be empty");
    require(bytes(imageURL).length > 0, "ImageURL
cannot be empty");

    projects[id].title = title;
    projects[id].description = description;
    projects[id].imageURL = imageURL;
    projects[id].expiresAt = expiresAt;

    emit Action(id, "PROJECT UPDATED",
msg.sender, block.timestamp);

    return true;
  }

  function deleteProject(uint id) public returns (bool)
{
    require(
      projects[id].status == statusEnum.OPEN,
      "Project no longer opened"
    );
    require(msg.sender == projects[id].owner,
"Unauthorized Entity");

    projects[id].status = statusEnum.DELETED;
    performRefund(id);

    emit Action(id, "PROJECT DELETED", msg.sender,
block.timestamp);

    return true;
  }

  function performRefund(uint id) internal {
    for (uint i = 0; i < backersOf[id].length; i++) {
      address _owner = backersOf[id][i].owner;
```

```
            uint _contribution = backersOf[id][i].contribution;

            backersOf[id][i].refunded = true;
            backersOf[id][i].timestamp = block.timestamp;
            payTo(_owner, _contribution);

            stats.totalBacking -= 1;
            stats.totalDonations -= _contribution;
        }
    }

    function backProject(uint id) public payable returns (bool) {
        require(msg.value > 0 ether, "Ether must be greater than zero");
        require(projectExist[id], "Project not found");
        require(
            projects[id].status == statusEnum.OPEN,
            "Project no longer opened"
        );

        stats.totalBacking += 1;
        stats.totalDonations += msg.value;
        projects[id].raised += msg.value;
        projects[id].backers += 1;

        backersOf[id].push(
            backerStruct(msg.sender, msg.value, block.timestamp, false)
        );

        emit Action(id, "PROJECT BACKED", msg.sender, block.timestamp);

        if (projects[id].raised >= projects[id].cost) {
            projects[id].status = statusEnum.APPROVED;
            balance += projects[id].raised;
            performPayout(id);
            return true;
        }

        if (block.timestamp >= projects[id].expiresAt) {
            projects[id].status = statusEnum.REVERTED;
            performRefund(id);
            return true;
        }

        return true;
    }

    function performPayout(uint id) internal {
        uint raised = projects[id].raised;
        uint tax = (raised * projectTax) / 100;

        projects[id].status = statusEnum.PAIDOUT;

        payTo(projects[id].owner, (raised - tax));
        payTo(owner, tax);

        balance -= projects[id].raised;

        emit Action(id, "PROJECT PAID OUT", msg.sender, block.timestamp);
    }

    function requestRefund(uint id) public returns (bool) {
        require(
            projects[id].status != statusEnum.REVERTED ||
                projects[id].status != statusEnum.DELETED,
            "Project not marked as revert or delete"
        );

        projects[id].status = statusEnum.REVERTED;
        performRefund(id);
        return true;
    }

    function payOutProject(uint id) public returns (bool) {
        require(
            projects[id].status == statusEnum.APPROVED,
            "Project not APPROVED"
        );
        require(
            msg.sender == projects[id].owner || msg.sender == owner,
            "Unauthorized Entity"
        );

        performPayout(id);
        return true;
    }

    function changeTax(uint _taxPct) public ownerOnly {
        projectTax = _taxPct;
    }

    function getProject(uint id) public view returns (projectStruct memory) {
        require(projectExist[id], "Project not found");
    return projects[id];
  }

  function getProjects() public view returns (projectStruct[] memory) {
    return projects;
  }

  function getBackers(uint id) public view returns (backerStruct[] memory) {
    return backersOf[id];
  }

  function payTo(address to, uint256 amount) internal {
    (bool success, ) = payable(to).call{value: amount}("");
    require(success);
  }
}
```





Save Animals

5

16-05-2023

https://picsum.photos/id/237/200/300

Save Animals

http://localhost:3000

0x5Fb...0aa3 : CONTRACT INTERACTION

$0.00

Gas (estimated) $2.55 0.00132581 ETH

Likely in < 30 seconds Max fee: 0.0014235 ETH

Total $2.55 0.00132581 ETH

Amount + gas fee Max amount: 0.0014235 ETH

7. Conclusion

The code provides functions for various interactions with the smart contract, such as creating, updating, and deleting projects, as well as loading project data, contributing to projects, and initiating project payouts.

Error handling is also implemented to report and handle errors when the required ethereum object is not available.

Overall, this implementation provides a foundation for building a user-friendly interface to interact with the smart contract. By leveraging MetaMask and the Ethereum network, users can perform actions such as managing projects, contributing funds, and retrieving project information.

8. Future Work

User Interface Enhancement: To create a complete and user-friendly application, a well-designed and intuitive user interface can be developed.

Authentication and Security: Currently, the code assumes that MetaMask is installed and handles the wallet connection. However, it's important to consider additional authentication and security measures

Smart Contract Optimization: Optimizing the smart contract can help reduce transaction costs and improve overall performance.

Testing: Thoroughly testing the smart contract code and the application as a whole is crucial to ensure its reliability, security, and compliance with the desired business logic.

Integration with IPFS or File Storage: If the application involves storing and displaying images or files related to projects, integrating with decentralized storage solutions like IPFS (InterPlanetary File System) can be considered.

9. Reference

[1]https://ethereum.org/en/developers/docs/smartcontracts/#:~:text=A%20%22smart%20contract%22%20is%20simply,be%20the%20target%20of%20transactions.
[2] Blockchain Based Crowdfunding. Medha Kulkarni, Pratik Tayad
[3] https://www.youtube.com/watch?v=gyMwXuJrbJQ&pp=ygUIc29saWRpdHk%3D