Project Report

On

Genesis

Blockchain Application

Using React Js

Designed and Developed

By

Mr. Naqi Ansari

TYBSc CS 26

2022-2023

And Guided By

Prof. Pramitha Santhumayor

Submitted in partial fulfillment of academic project

[Bachelor of Computer Science]

University of Mumbai

Since 1989

Devoted to Serve

ROYAL COLLEGE
OF ARTS, SCIENCE & COMMERCE

Empowerment through Value Education

DEPARTMENT OF COMPUTER SCIENCE

Class: TYBSc Roll. No. 26

Seat No._____

Certificate

Certified that **Naqi Ansari** of T.Y.BSc Semester-VI has successfully completed the project as prescribed by the University of Mumbai on **Genesis** as partial fulfillment of requirement for completing Bachelor's Degree in Computer Science during the academic year 2022-2023.

Signature of Project Guide
Date: _____

Signature of Examiner
Date: _____

H.O.D

Dept. of Computer Science

ACKNOWLEDGEMENT

I would also like to extend my appreciation to all the faculty members of the Computer Science department who have played a vital role in shaping my academic career. Their insightful guidance, valuable feedback, and constructive criticism have been instrumental in enabling me to develop a better understanding of the subject matter and to enhance my problem-solving skills.

I would like to express my gratitude to my classmates who have been a constant source of motivation and support throughout the project. Their valuable inputs and suggestions have helped me to refine my ideas and to improve the quality of my work.

I would like to thank my friends and colleagues for their encouragement and support during the project. Their unwavering belief in my abilities and their willingness to lend a helping hand whenever needed have been crucial in keeping me motivated and focused.

I would like to acknowledge the invaluable support and assistance provided by the staff of the Royal College of Arts, Commerce & Science. Their cooperation and prompt response to my requests have made the project a smooth and hassle-free experience.

I would also like to extend my heartfelt thanks to **Prof**. **Pramitha Santhumayor** , who has been a constant source of inspiration, guidance, and support throughout the project. Her unwavering faith in my abilities and her constructive feedback have been instrumental in shaping my ideas and refining my work. Her invaluable inputs and suggestions have helped me to develop a better understanding of the subject matter and to enhance my skills.

I would also like to express my gratitude to the entire Computer Science department for their support and encouragement during the project. Their expertise and experience have been invaluable in helping me to overcome challenges and to achieve my goals. Their willingness to share their knowledge and to provide me with the resources I needed has been crucial in enabling me to complete this project successfully.

I would like to thank my family for their unwavering support, motivation, and encouragement throughout the project. Their belief in me and their constant encouragement have been my guiding light, and I could not have completed this project without their love and support. Their sacrifices and unwavering faith in me have been the cornerstone of my success, and I am truly grateful to them for everything they have done for me.

Finally, I would like to thank the almighty for blessing me with the strength, perseverance, and determination to complete this project successfully. Without His grace and guidance, this accomplishment would not have been possible.

Naqi Ansari

DECLARATION

I, Mr.Naqi Ansari hereby confirm that the project titled "Design and Implementation of Genesis" is the result of my independent work and that I have not used any unauthorized assistance or material in the completion of this project. The project has been undertaken as part of my course curriculum for the Bachelor's Degree in Computer Science at Mumbai University.

Throughout the project, I have personally designed the system architecture, developed the programming logic, and performed the required testing and validation. I have made sure to incorporate all the necessary features and functionalities required to meet the project objectives and to ensure a seamless user experience.

Moreover, I acknowledge that the project may require modifications in the future as per the user's requirements or due to changes in the technological landscape. In this regard, I have incorporated flexibility in the system design to enable any necessary modifications or updates. I am confident that I can make the required changes by modifying the file design or the program code, if necessary, to ensure the system's continued smooth operation.

I would also like to state that I have taken all necessary precautions to ensure that the system design and implementation are secure and comply with industry standards. I have followed best practices in data protection, authentication to ensure that user data remains secure and confidential.

I have developed a deep understanding of the system's underlying technology and functionality throughout the project's development, and I am confident that I can handle any modifications or updates that may be required in the future. I have created the system with a modular and scalable design, enabling me to incorporate changes or enhancements without disrupting the system's overall functioning.

Finally, I affirm that this project report represents my own work, and all sources used have been duly cited and referenced. Any resemblance to other works is purely coincidental.

Index

INDEX

Preliminary Investigation

Introduction

The Genesis Crowd Funding Application is a decentralized crowdfunding platform built on blockchain technology that provides individuals and organizations with a transparent and secure way to raise funds for their projects, campaigns, or businesses without the need for intermediaries such as banks, payment gateways, or traditional crowdfunding platforms. The application uses Solidity, React JS, Hardhat, and Metamask on the Goerli network to ensure the highest level of security and transparency in the fundraising process.

Objectives:

The main objective of the Genesis Crowd Funding Application is to provide a decentralized crowdfunding platform that empowers individuals and organizations to raise funds for their projects, campaigns, or businesses without the need for intermediaries. The application aims to eliminate the risks of fraud and manipulation by third parties, increase transparency and accountability, provide a higher level of security for the funds raised, and reduce fees significantly. The application also aims to provide global access to fundraising campaigns and simplify the process of starting and contributing to a crowdfunding campaign.

Need for the App:

The need for the Genesis Crowd Funding Application arises from the limitations of traditional crowdfunding platforms that rely on intermediaries such as banks and payment gateways, which can be costly, slow, and prone to fraud and manipulation. Additionally, traditional crowdfunding platforms often have limited access to global participation, making it challenging for individuals and organizations to reach a broader audience. The Genesis Crowd Funding Application addresses these limitations by providing a decentralized, transparent, and secure platform that is accessible to anyone with an internet connection, enabling global participation in fundraising campaigns.

Proposed System and Advantages.

The proposed system is a decentralized crowdfunding platform that uses blockchain technology to enable individuals and organizations to raise funds for their projects, campaigns, or businesses without the need for intermediaries such as banks and payment gateways. The platform will allow users to create and manage crowdfunding campaigns using Ether (ETH) on the blockchain network. The system will use smart contracts to enforce the rules of the campaign and release the funds raised to the campaign owner when the fundraising goal is reached before the deadline. The system will provide transparency, security, and lower fees than traditional crowdfunding platforms.

Advantages over Traditional Crowdfunding with Centralized Banking and Intermediary Fees:

1) Decentralization: The proposed system is decentralized, which means it is not controlled by any central authority or intermediary. This eliminates the risk of fraud or manipulation by third parties.

2) Transparency: The transactions and activities on the blockchain are transparent and can be viewed by anyone, providing greater accountability and trust in the crowdfunding process.

3) Security: The blockchain technology is secure and tamper-proof, providing a higher level of security for the funds raised.

4) Lower Fees: Traditional crowdfunding platforms charge high fees for their services. With this blockchain-based crowdfunding platform, there are no intermediaries, so fees are significantly lower.

5) Global Access: The proposed system is accessible to anyone with an internet connection, enabling global participation in fundraising campaigns.

6) Faster Transaction Processing: The proposed system uses blockchain technology, which can process transactions faster than traditional crowdfunding platforms, reducing the time it takes to receive funds.

7) More Control for Campaign Owners: The proposed system gives more control to campaign owners by enabling them to manage their campaigns directly without the need for intermediaries. This allows for greater customization and flexibility in the fundraising process.

System Requirements.

Hardware Requirements:

- A computer or laptop with a minimum of 4 GB RAM and a dual-core processor.
- An internet connection with a minimum speed of 5 Mbps.

Software Requirements:

- Operating System: Windows, macOS, or Linux.
- Node.js: version 12 or later.
- NPM: version 6 or later.
- Metamask wallet: version 10 or later.
- Hardhat: version 2.0 or later.
- Alchemy API key.

Note: The above software requirements are necessary for both the development and deployment of the Genesis Crowdfunding Application.

Deployment Requirements:

- Goerli network: The Genesis Crowdfunding Application is deployed on the Goerli network, so you will need to connect to this network using your Metamask wallet.
- Ethereum (ETH): You will need some Ethereum (ETH) on the Goerli network to interact with the smart contract.

TYBSC Computer Science Semester 6
2022 -2023

Phase Title	Expected Date of Completion	Actual Time of Completion with Guide's Signature	Remarks
I. Preliminary Investigation			
(i) Organizational Overview			
(ii) Present System and its advantages	10/12/2022		
(iii) System Requirements			
(iv) Feasibility Study			
(v) Fact Finding Methods			
(vi) Phase Title	20/12/2022		
(vii) Gantt Chart			
II. System Analysis			
(i) Event Table			
(ii) Use Case Diagram	09/01/2023		
(iii) Class Diagram			
III. System Design			
(i) Sequence Diagram			
IV. System Coding			
(i) System Coding	06/02/2023		
(ii) Form Layouts			
(iii) Report Layouts			
V. Future Enhancements			
VI. Reference and Bibliography	20/03/2023		

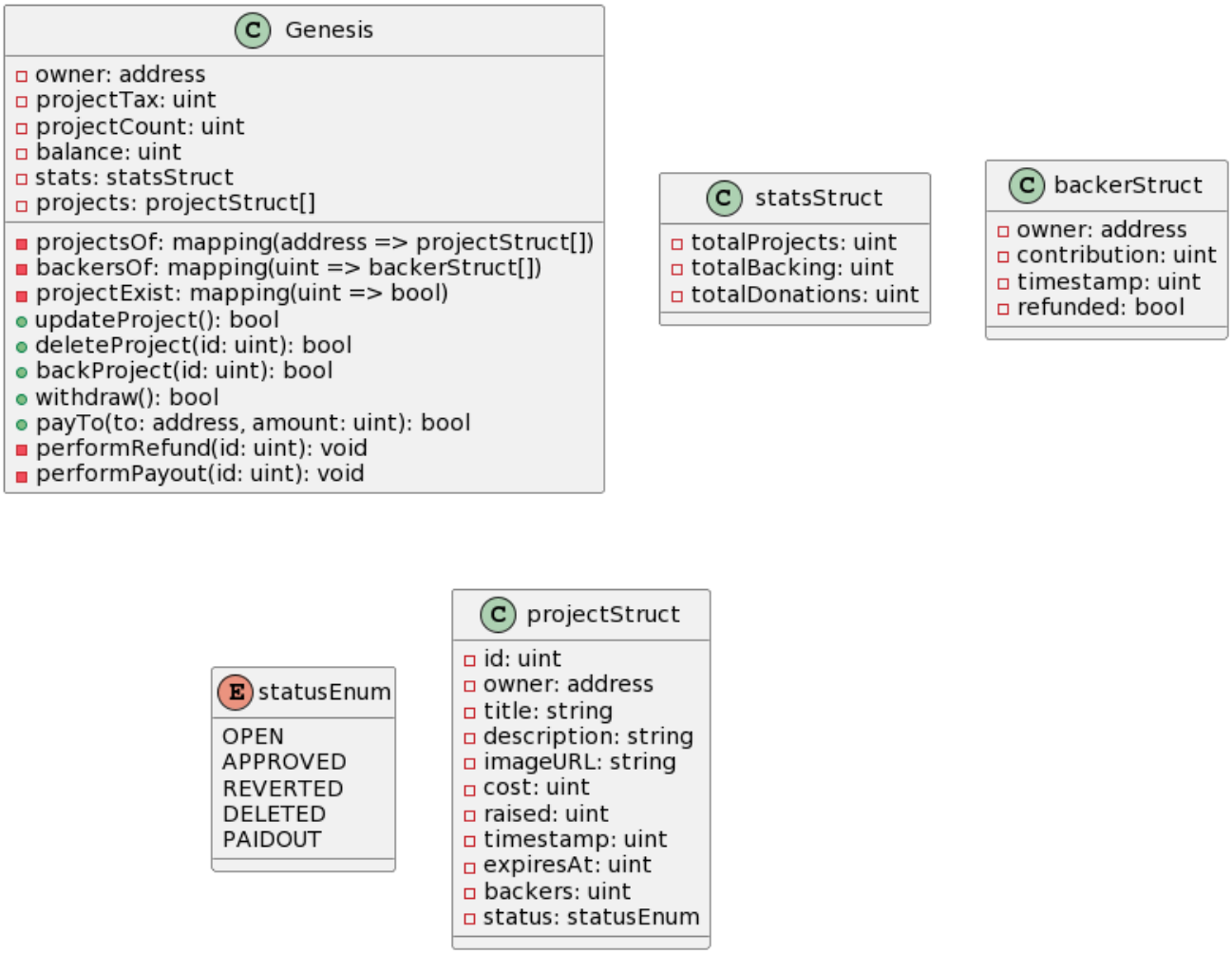
Gantt Chart

TYBSc Computer Science Semester 6 Project Gantt Chart		Time Requirement	Year 2019 - 20												
			Weeks												
			December		January				February				March		
			W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3
I	**Preliminary Investigation**	Estimated													
		Actual													
II	**Requirement Gathering**	Estimated													
		Actual													
III	**System Analysis**	Estimated													
		Actual													
IV	**System Design**	Estimated													
		Actual													
V	**System Coding**	Estimated													
		Actual													
VI	**Testing**	Estimated													
		Actual													
VII	**Implementation**	Estimated													
		Actual													
VIII	**Deployment**	Estimated													
		Actual													

Estimated	
Actual	

System Analysis

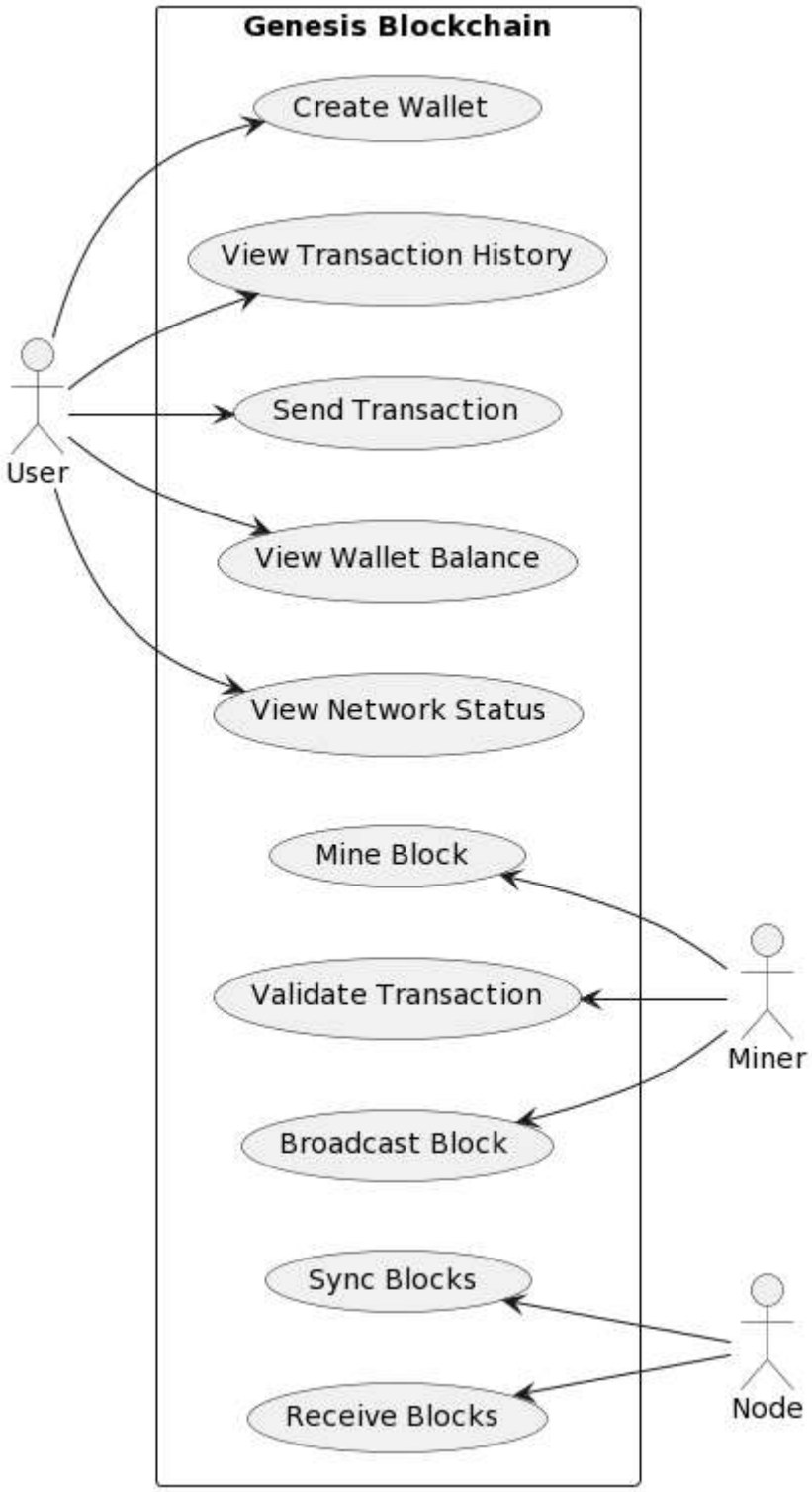
Class Diagram



Event Table

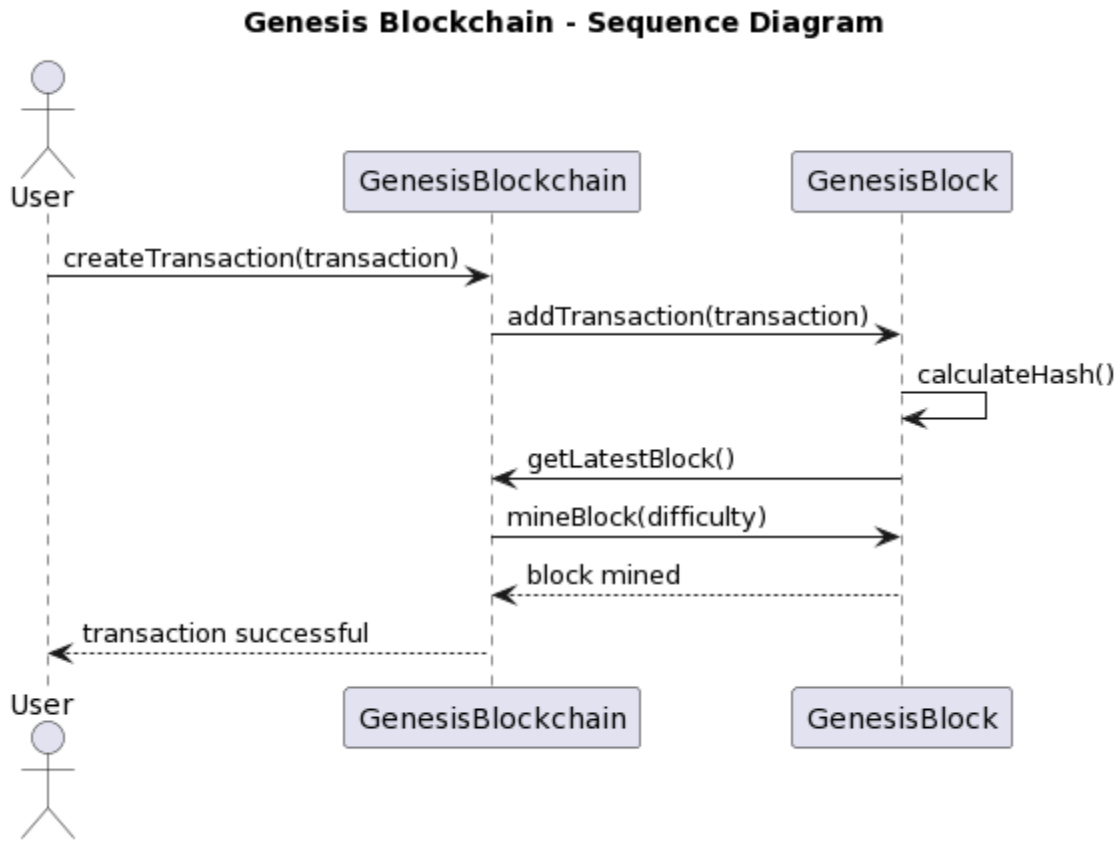
Event	Trigger	Precondition	Post condition
User creates campaign	User specifies project details, fundraising goal, and deadline	User has a Metamask wallet on the Goerli network	Smart contract is created on the blockchain
User contributes to campaign	User sends Ether (ETH) through Metamask wallet	Campaign smart contract exists on the blockchain	Funds are added to the campaign balance
Campaign reaches fundraising goal	Campaign balance reaches or exceeds fundraising goal	Campaign smart contract exists on the blockchain	Campaign owner can withdraw funds
Campaign deadline passes	Campaign deadline is reached	Campaign smart contract exists on the blockchain	If fundraising goal is not met, funds are returned to contributors
Campaign owner withdraws funds	Campaign owner initiates withdrawal	Campaign balance is greater than 0 and fundraising goal has been met	Funds are transferred from the campaign smart contract to the campaign owner's wallet

Use Case


Genesis Blockchain
Create Wallet
View Transaction History
Send Transaction
View Wallet Balance
View Network Status
Mine Block
Validate Transaction
Broadcast Block
Sync Blocks
Receive Blocks
User
Miner
Node

System Design

Sequence Diagram



System Coding

App.jsx

```
import { useEffect, useState } from 'react'
import { Route, Routes } from 'react-router-dom'
import Header from './components/Header'
import Home from './views/Home'
import Project from './views/Project'
import { isWallectConnected } from './services/blockchain'
import { ToastContainer } from 'react-toastify'

const App = () => {
  const [loaded, setLoaded] = useState(false)

  useEffect(async () => {
    await isWallectConnected()
    console.log('Blockchain loaded')
    setLoaded(true)
  }, [])

  return (
    <div className="min-h-screen relative">
      <Header />
      {loaded ? (
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/projects/:id" element={<Project />} />
        </Routes>
      ) : null}

      <ToastContainer
        position="bottom-center"
        autoClose={5000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
        pauseOnFocusLoss
        draggable
        pauseOnHover
        theme="dark"
      />
    </div>
  )
}
```

```
export default App
```

Home.jsx

```
import { useEffect } from 'react'
import AddButton from '../components/AddButton'
import CreateProject from '../components/CreateProject'
import Hero from '../components/Hero'
import Projects from '../components/Projects'
import { loadProjects } from '../services/blockchain'
import { useGlobalState } from '../store'

const Home = () => {
  const [projects] = useGlobalState('projects')

  useEffect(async () => {
    await loadProjects()
  }, [])
  return (
    <>
      <Hero />
      <Projects projects={projects} />
      <CreateProject />
      <AddButton />
    </>
  )
}

export default Home
```

Projects.jsx

```
import { useEffect, useState } from 'react'
import { useParams } from 'react-router-dom'
import BackProject from '../components/BackProject'
import DeleteProject from '../components/DeleteProject'
import ProjectBackers from '../components/ProjectBackers'
import ProjectDetails from '../components/ProjectDetails'
import UpdateProject from '../components/UpdateProject'
import { getBackers, loadProject } from '../services/blockchain'
import { useGlobalState } from '../store'

const Project = () => {
  const { id } = useParams()
```

```
  const [loaded, setLoaded] = useState(false)
  const [project] = useGlobalState('project')
  const [backers] = useGlobalState('backers')

  useEffect(async () => {
    await loadProject(id)
    await getBackers(id)
    setLoaded(true)
  }, [])
  return loaded ? (
    <>
      <ProjectDetails project={project} />
      <UpdateProject project={project} />
      <DeleteProject project={project} />
      <BackProject project={project} />
      <ProjectBackers backers={backers} />
    </>
  ) : null
}

export default Project
```

blockchain.jsx

```
import abi from '../abis/src/contracts/Genesis.sol/Genesis.json'
import address from '../abis/contractAddress.json'
import { getGlobalState, setGlobalState } from '../store'
import { ethers } from 'ethers'

const { ethereum } = window
const contractAddress = address.address
const contractAbi = abi.abi

const connectWallet = async () => {
  try {
    if (!ethereum) return alert('Please install Metamask')
    const accounts = await ethereum.request({ method: 'eth_requestAccounts' })
    setGlobalState('connectedAccount', accounts[0]?.toLowerCase())
  } catch (error) {
    reportError(error)
  }
}

const isWallectConnected = async () => {
  try {
```

```
    if (!ethereum) return alert('Please install Metamask')
    const accounts = await ethereum.request({ method: 'eth_accounts' })
    setGlobalState('connectedAccount', accounts[0]?.toLowerCase())

    window.ethereum.on('chainChanged', (chainId) => {
      window.location.reload()
    })

    window.ethereum.on('accountsChanged', async () => {
      setGlobalState('connectedAccount', accounts[0]?.toLowerCase())
      await isWallectConnected()
    })

    if (accounts.length) {
      setGlobalState('connectedAccount', accounts[0]?.toLowerCase())
    } else {
      alert('Please connect wallet.')
      console.log('No accounts found.')
    }
  } catch (error) {
    reportError(error)
  }
}

const getEtheriumContract = async () => {
  const connectedAccount = getGlobalState('connectedAccount')

  if (connectedAccount) {
    const provider = new ethers.providers.Web3Provider(ethereum)
    const signer = provider.getSigner()
    const contract = new ethers.Contract(contractAddress, contractAbi, signer)

    return contract
  } else {
    return getGlobalState('contract')
  }
}

const createProject = async ({
  title,
  description,
  imageURL,
  cost,
  expiresAt,
}) => {
```

```
  try {
    if (!ethereum) return alert('Please install Metamask')

    const contract = await getEtheriumContract()
    cost = ethers.utils.parseEther(cost)
    await contract.createProject(title, description, imageURL, cost, expiresAt)
  } catch (error) {
    reportError(error)
  }
}

const updateProject = async ({
  id,
  title,
  description,
  imageURL,
  expiresAt,
}) => {
  try {
    if (!ethereum) return alert('Please install Metamask')

    const contract = await getEtheriumContract()
    await contract.updateProject(id, title, description, imageURL, expiresAt)
  } catch (error) {
    reportError(error)
  }
}

const deleteProject = async (id) => {
  try {
    if (!ethereum) return alert('Please install Metamask')
    const contract = await getEtheriumContract()
    await contract.deleteProject(id)
  } catch (error) {
    reportError(error)
  }
}

const loadProjects = async () => {
  try {
    if (!ethereum) return alert('Please install Metamask')

    const contract = await getEtheriumContract()
    const projects = await contract.getProjects()
    const stats = await contract.stats()
```

```
    setGlobalState('stats', structureStats(stats))
    setGlobalState('projects', structuredProjects(projects))
  } catch (error) {
    reportError(error)
  }
}

const loadProject = async (id) => {
  try {
    if (!ethereum) return alert('Please install Metamask')
    const contract = await getEtheriumContract()
    const project = await contract.getProject(id)

    setGlobalState('project', structuredProjects([project])[0])
  } catch (error) {
    alert(JSON.stringify(error.message))
    reportError(error)
  }
}

const backProject = async (id, amount) => {
  try {
    if (!ethereum) return alert('Please install Metamask')
    const connectedAccount = getGlobalState('connectedAccount')
    const contract = await getEtheriumContract()
    amount = ethers.utils.parseEther(amount)

    await contract.backProject(id, {
      from: connectedAccount,
      value: amount._hex,
    })
  } catch (error) {
    reportError(error)
  }
}

const getBackers = async (id) => {
  try {
    if (!ethereum) return alert('Please install Metamask')
    const contract = await getEtheriumContract()
    let backers = await contract.getBackers(id)

    setGlobalState('backers', structuredBackers(backers))
  } catch (error) {
```

```
    reportError(error)
  }
}

const payoutProject = async (id) => {
  try {
    if (!ethereum) return alert('Please install Metamask')
    const connectedAccount = getGlobalState('connectedAccount')
    const contract = await getEtheriumContract()

    await contract.payOutProject(id, {
      from: connectedAccount,
    })
  } catch (error) {
    reportError(error)
  }
}

const structuredBackers = (backers) =>
  backers
    .map((backer) => ({
      owner: backer.owner.toLowerCase(),
      refunded: backer.refunded,
      timestamp: new Date(backer.timestamp.toNumber() * 1000).toJSON(),
      contribution: parseInt(backer.contribution._hex) / 10 ** 18,
    }))
    .reverse()

const structuredProjects = (projects) =>
  projects
    .map((project) => ({
      id: project.id.toNumber(),
      owner: project.owner.toLowerCase(),
      title: project.title,
      description: project.description,
      timestamp: new Date(project.timestamp.toNumber()).getTime(),
      expiresAt: new Date(project.expiresAt.toNumber()).getTime(),
      date: toDate(project.expiresAt.toNumber() * 1000),
      imageURL: project.imageURL,
      raised: parseInt(project.raised._hex) / 10 ** 18,
      cost: parseInt(project.cost._hex) / 10 ** 18,
      backers: project.backers.toNumber(),
      status: project.status,
    }))
    .reverse()
```

```
const toDate = (timestamp) => {
  const date = new Date(timestamp)
  const dd = date.getDate() > 9 ? date.getDate() : `0${date.getDate()}`
  const mm =
    date.getMonth() + 1 > 9 ? date.getMonth() + 1 : `0${date.getMonth() + 1}`
  const yyyy = date.getFullYear()
  return `${yyyy}-${mm}-${dd}`
}

const structureStats = (stats) => ({
  totalProjects: stats.totalProjects.toNumber(),
  totalBacking: stats.totalBacking.toNumber(),
  totalDonations: parseInt(stats.totalDonations._hex) / 10 ** 18,
})

const reportError = (error) => {
  console.log(error.message)
  throw new Error('No ethereum object.')
}

export {
  connectWallet,
  isWallectConnected,
  createProject,
  updateProject,
  deleteProject,
  loadProjects,
  loadProject,
  backProject,
  getBackers,
  payoutProject,
}
```

Contract/Genesis.sol

```
//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);
    }
}
```

Components/AddButton.jsx

```jsx
import { setGlobalState } from '../store'
import { BsPlusLg } from 'react-icons/bs'

const AddButton = () => {
  return (
    <div className="fixed right-10 bottom-10 flex space-x-2 justify-center">
      <button
        type="button"
        className="flex justify-center items-center w-9 h-9 bg-green-600
        text-white font-medium text-xs leading-tight uppercase
        rounded-full shadow-md hover:bg-green-700"
        onClick={() => setGlobalState('createModal', 'scale-100')}
      >
        <BsPlusLg className='font-bold' size={20} />
      </button>
    </div>
  )
}

export default AddButton
```

Components/BackProject.jsx

```
import { useState } from 'react'
import { FaTimes } from 'react-icons/fa'
import { toast } from 'react-toastify'
import { backProject } from '../services/blockchain'
import { useGlobalState, setGlobalState } from '../store'

const BackProject = ({ project }) => {
  const [backModal] = useGlobalState('backModal')
  const [amount, setAmount] = useState('')

  const handleSubmit = async (e) => {
    e.preventDefault()
    if (!amount) return

    await backProject(project?.id, amount)
    toast.success('Project backed successfully, will reflect in 30sec.')
    setGlobalState('backModal', 'scale-0')
  }

  return (
    <div
      className={`fixed top-0 left-0 w-screen h-screen flex
    items-center justify-center bg-black bg-opacity-50
    transform transition-transform duration-300 ${backModal}`}
    >
      <div
        className="bg-white shadow-xl shadow-black
        rounded-xl w-11/12 md:w-2/5 h-7/12 p-6"
      >
        <form onSubmit={handleSubmit} className="flex flex-col">
          <div className="flex justify-between items-center">
            <p className="font-semibold">{project?.title}</p>
            <button
              onClick={() => setGlobalState('backModal', 'scale-0')}
              type="button"
              className="border-0 bg-transparent focus:outline-none"
            >
              <FaTimes />
            </button>
          </div>

          <div className="flex justify-center items-center mt-5">
            <div className="rounded-xl overflow-hidden h-20 w-20">
              <img
                src={
```

```
                project?.imageURL ||
                'https://media.wired.com/photos/5926e64caf95806129f50fde/master
/pass/AnkiHP.jpg'
              }
              alt={project?.title}
              className="h-full w-full object-cover cursor-pointer"
            />
          </div>
        </div>

        <div
          className="flex justify-between items-center
        bg-gray-300 rounded-xl mt-5"
        >
          <input
            className="block w-full bg-transparent
          border-0 text-sm text-slate-500 focus:outline-none
          focus:ring-0"
            type="number"
            step={0.01}
            min={0.01}
            name="amount"
            placeholder="Amount (ETH)"
            onChange={(e) => setAmount(e.target.value)}
            value={amount}
            required
          />
        </div>

        <button
          type="submit"
          className="inline-block px-6 py-2.5 bg-green-600
          text-white font-medium text-md leading-tight
          rounded-full shadow-md hover:bg-green-700 mt-5"
        >
          Back Project
        </button>
      </form>
    </div>
  </div>
 )
}

export default BackProject
```

hardhat.config.js

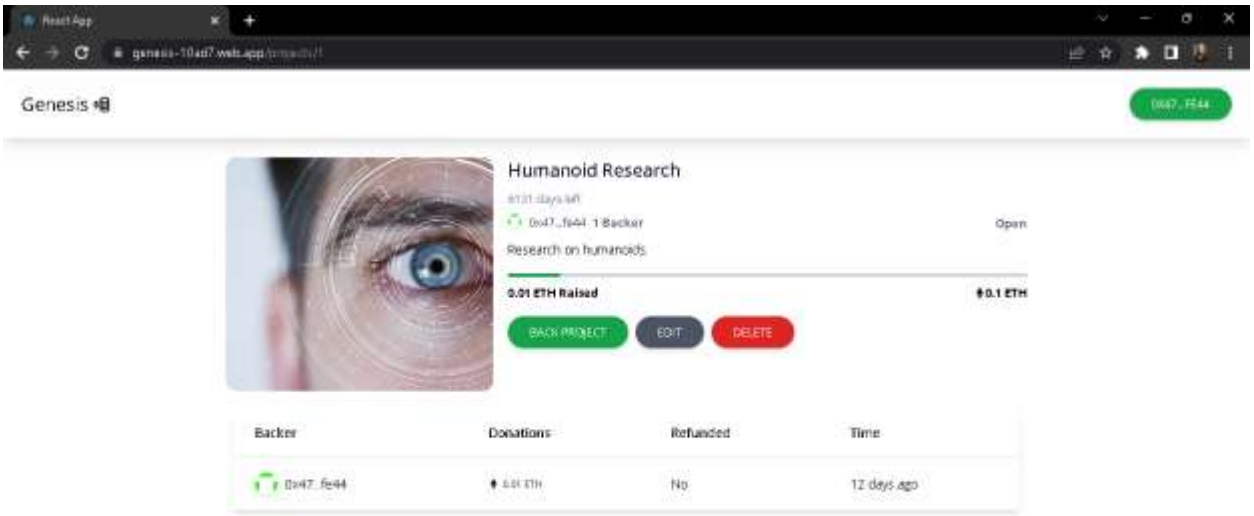
```
require('@nomiclabs/hardhat-waffle')
require('dotenv').config()

module.exports = {
  defaultNetwork: 'localhost',
  networks: {
    localhost: {
      url: 'http://127.0.0.1:8545',
    },
    goerli: {
      url: process.env.ENDPOINT_URL,
      accounts: [process.env.DEPLOYER_KEY]
    }
  },
  solidity: {
    version: '0.8.11',
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
  paths: {
    sources: './src/contracts',
    artifacts: './src/abis',
  },
  mocha: {
    timeout: 40000,
  },
}
```

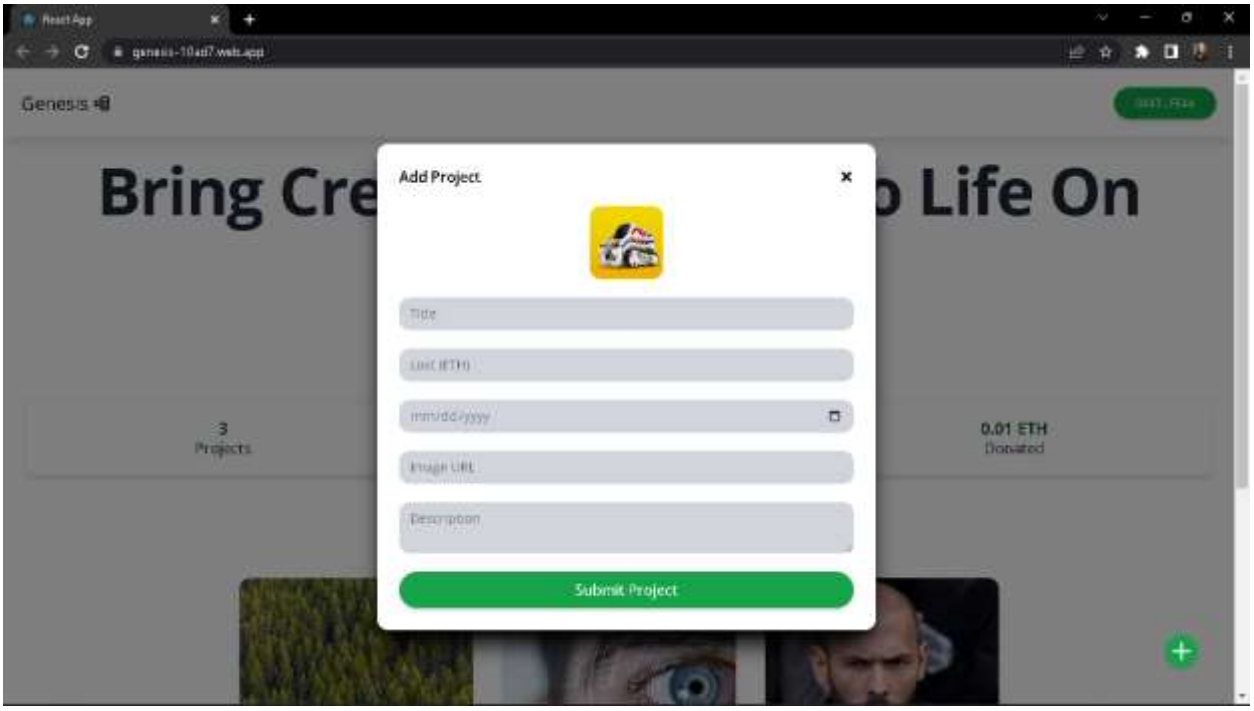
Test Cases

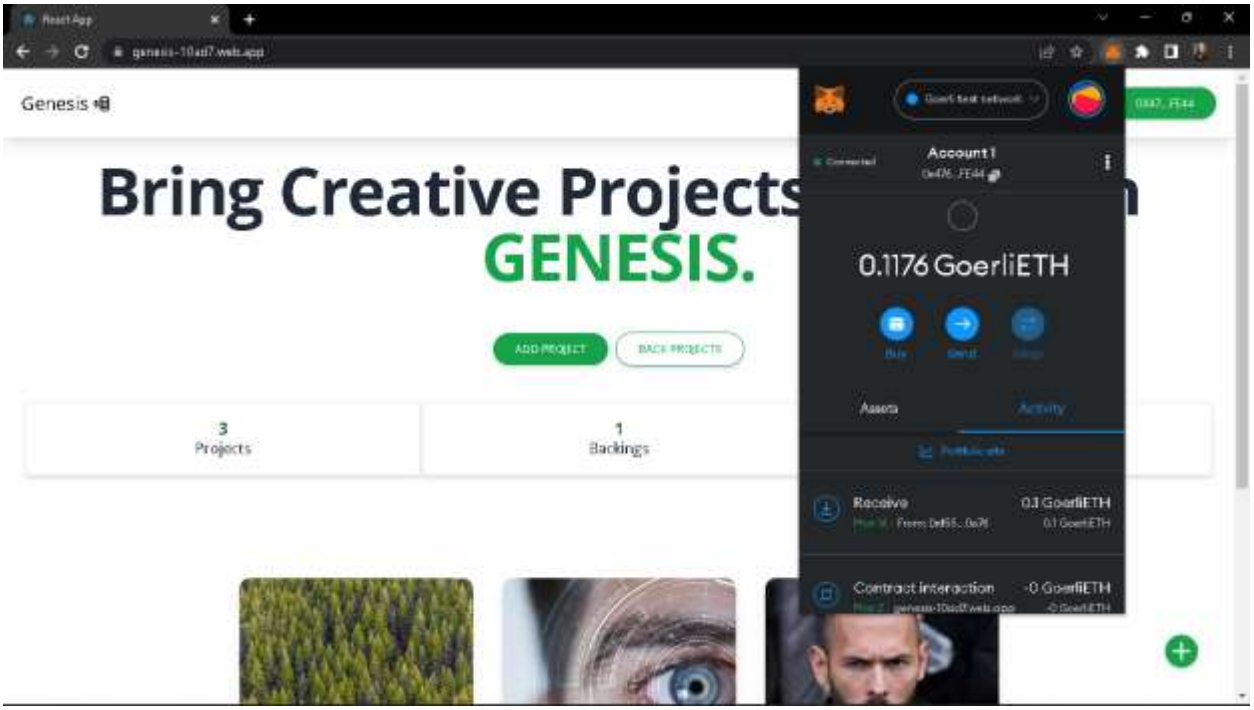
Test Case	Description	Data	Result
Creating a Campaign	Tests if a user can successfully create a campaign	Campaign details, fundraising goal, deadline, and Metamask wallet	A new smart contract is created on the blockchain and the campaign is added to the list of active campaigns
Contributing to a Campaign	Tests if a user can successfully contribute to a campaign	Campaign smart contract address, Metamask wallet, and amount of ETH	The campaign balance is increased by the amount of ETH sent
Reaching Fundraising Goal	Tests if the smart contract correctly detects when a campaign has reached its fundraising goal	Campaign smart contract address, amount of ETH contributed, fundraising goal, and deadline	If the campaign balance reaches or exceeds the fundraising goal before the deadline, the campaign owner is able to withdraw the funds raised
Deadline Passes	Tests if the smart contract correctly detects when a campaign deadline has passed	Campaign smart contract address and deadline	If the deadline passes and the fundraising goal has not been reached, the campaign balance is returned to the contributors
Withdrawing Funds	Tests if the campaign owner can successfully withdraw the funds raised	Campaign smart contract address and Metamask wallet	The funds are transferred from the campaign smart contract to the campaign owner's wallet

Snapshot


Genesis
Bring Creative Projects To Life On
GENESIS.
3
Projects
1
Backings
0.01 ETH
Donated
Test
Humanoid Research
Free Top G


Genesis
Humanoid Research
Open
Research on humanoids
0.01 ETH Raised
0.1 ETH
BACK PROJECT
EDIT
DELETE
Backer
Donations
Refunded
Time
No
12 days ago


Genesis
Add Project
Submit Project
3
Projects
0.01 ETH


Genesis
Bring Creative Projects
GENESIS.
ADD PROJECT
3
Projects
1
Backings
Account 1
0.1176 GoerliETH
Assets
Activity
Receive
0.1 GoerliETH
Contract Interaction
-0 GoerliETH

Future Enhancement

- Integration with other cryptocurrencies: Currently, Genesis only accepts Ether (ETH) as a form of contribution to campaigns. In the future, the application can be enhanced to accept other cryptocurrencies like Bitcoin, Litecoin, or Dogecoin.

- Improved user interface: The current user interface of Genesis is basic and can be improved to make it more user-friendly. Enhancements can include better campaign search, improved filtering options, and an interactive dashboard for campaign owners.

- Social media integration: To improve campaign awareness and reach, Genesis can be integrated with social media platforms like Twitter, Facebook, and LinkedIn. This will allow campaign owners to promote their campaigns to a larger audience.

- Smart contract templates: Currently, campaign smart contracts are created for each campaign, which can be time-consuming. In the future, smart contract templates can be created to simplify the process of creating new campaigns.

- Tokenization of campaigns: Genesis can be enhanced to allow campaigns to create and issue their own tokens. This will allow campaigns to create unique incentives for their supporters and can help in promoting their campaigns.

- Integration with analytics tools: Genesis can be enhanced to provide campaign owners with analytics tools to track the performance of their campaigns. This can include data on campaign views, contributor demographics, and other important metrics.

Reference and Bibliography

1. Antonopoulos, A. M. (2014). Mastering Bitcoin: Unlocking Digital Cryptocurrencies. O'Reilly Media.

2. Buterin, V. (2014). A Next-Generation Smart Contract and Decentralized Application Platform. Ethereum.

3. Christidis, K., & Devetsikiotis, M. (2016). Blockchains and Smart Contracts for the Internet of Things. IEEE Access, 4, 2292-2303.

4. Dannen, C. (2017). Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. Apress.

5. Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press.