

**RAJALAKSHMI ENGINEERING COLLEGE**  
**RAJALAKSHMI NAGAR, THANDALAM – 602 105**



**RAJALAKSHMI**  
**ENGINEERING COLLEGE**

**CB23332**  
**SOFTWARE ENGINEERING LAB**

**Laboratory Record Note Book**

Name : .....

Year / Branch / Section : .....

Register No. : .....

Semester : .....

Academic Year : .....

**RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)**  
**RAJALAKSHMI NAGAR, THANDALAM – 602-105**

**BONAFIDE CERTIFICATE**

**NAME:** \_\_\_\_\_ **REGISTER NO.:** \_\_\_\_\_

**ACADEMIC YEAR:** 2024-25 **SEMESTER:** III **BRANCH:** \_\_\_\_\_ B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

**CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.**

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on \_\_\_\_\_

Internal Examiner

External Examiner



**REGULATION 2023**  
**CB23332 – SOFTWARE ENGINEERING – LAB MANUAL**

S.No.	Date	INDEX	Page No.	Signature
1		Writing the complete problem statement		
2		Writing the software requirement specification document		
3		Entity relationship diagram		
4		Data flow diagrams at level 0 and level 1		
5		Use case diagram		
6		Activity diagram of all use cases.		
7		State chart diagram of all use cases		
8		Sequence diagram of all use cases		
9		Collaboration diagram of all use cases		
10		Assign objects in sequence diagram to classes and make class diagram		
11		Sample Code and screenshots		

<b>EX NO:1</b>	<b>WRITE THE COMPLETE PROBLEM STATEMENT</b>
<b>DATE</b>	

**AIM:** To prepare a problem statement for a decentralized blockchain-based crowdfunding platform that eliminates intermediaries and provides transparency and security in fundraising processes.

**ALGORITHM:**

1. **Define the Problem Clearly**
  - Identify challenges in traditional crowdfunding platforms like high fees, fraud risks, and lack of transparency.
2. **High-Level Problem Description**
  - Summarize the inefficiencies in centralized systems and highlight the benefits of blockchain.
3. **Emphasize Improvement**
  - Address the need for trust, automation, and transparency in the crowdfunding ecosystem.
4. **Impact on Stakeholders**
  - Outline how the solution benefits project owners, donors, and regulators.
5. **Focus on What Needs to be Solved**
  - Avoid implementation details; focus on goals.

**INPUT:**

- Limitations of traditional crowdfunding platforms.
- Stakeholders' expectations (project creators, contributors, regulators).
- Analysis of current fundraising challenges.
- Overview of blockchain technology advantages.

**OUTPUT:**

**Problem:**

Traditional crowdfunding platforms are centralized, leading to issues like high transaction fees, delays in fund transfers, lack of transparency, and increased fraud risks. These limitations undermine trust between fundraisers and contributors.

**Background:**

The growing popularity of crowdfunding as a financial tool has exposed inefficiencies in centralized

platforms. Blockchain technology offers decentralized, immutable, and transparent solutions to address these challenges by automating processes and reducing reliance on third parties.

**Relevance:**

A decentralized crowdfunding platform can revolutionize the fundraising landscape by:

- Ensuring transparent fund management.
- Eliminating intermediaries, reducing costs.
- Leveraging smart contracts for secure and automated fund allocation.

**OBJECTIVE:**

1. **For Fundraisers (Project Owners):**
  - Enable secure, decentralized project creation and funding requests.
  - Automate fund allocation based on predefined milestones.
2. **For Donors (Contributors):**
  - Provide a transparent and tamper-proof transaction record.
  - Offer a secure platform for contributing to projects.
3. **General Objectives:**
  - Build a decentralized ecosystem that fosters trust and efficiency.
  - Integrate smart contracts to eliminate manual fund distribution.

**Result:**

The problem statement has been successfully created, addressing the inefficiencies in centralized crowdfunding platforms and proposing a decentralized solution that ensures transparency, trust, and efficiency for all stakeholders

<b>EX NO:2</b>	<b>WRITE THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT</b>
<b>DATE</b>	

### **Aim:**

To perform requirement analysis and develop a Software Requirement Specification (SRS) for a decentralized blockchain-based crowdfunding platform.

## **1. Introduction**

### **1.1 Purpose**

The purpose of this document is to develop a decentralized crowdfunding platform leveraging blockchain technology to provide transparency, security, and trust for fundraisers and contributors. This platform aims to automate the fundraising process using smart contracts, ensuring secure and transparent transactions.

### **1.2 Document Conventions**

This document uses the following conventions:

- **SC:** Smart Contract
- **TxID:** Transaction Identifier
- **UI:** User Interface

### **1.3 Intended Audience and Reading Suggestions**

This SRS is intended for:

- **Developers:** To understand the technical and functional requirements.
- **Fundraisers:** To know the platform's capabilities for campaign creation and management.
- **Contributors:** To understand how they can securely fund campaigns.

### **1.4 Project Scope**

The decentralized crowdfunding platform will eliminate intermediaries and ensure transparency in fundraising. Features include:

- Smart contract-driven fund management.
- Real-time transaction tracking.
- Secure contribution mechanisms with blockchain immutability.

### **1.5 References**

- Blockchain whitepapers (Ethereum, Hyperledger).
- Current centralized crowdfunding platforms (e.g., GoFundMe, Kickstarter).

## **2. Overall Description**

### **2.1 Product Perspective**

The platform will operate as a decentralized application (dApp), ensuring immutable and transparent records. It integrates with blockchain wallets for transactions and uses smart contracts for fund automation.

### **2.2 Product Features**

- **Campaign Management:** Create, update, and manage fundraising campaigns.
- **Secure Donations:** Enable contributors to fund projects securely via blockchain wallets.
- **Fund Release Automation:** Use smart contracts for milestone-based fund disbursement.
- **Transaction Records:** Immutable and real-time tracking of all financial transactions.

### **2.3 User Classes and Characteristics**

- **Fundraisers:** Non-technical users managing campaigns.
- **Contributors:** Individuals contributing funds, seeking transparency and security.
- **Administrators:** Monitor and ensure the platform's compliance and reliability.

### **2.4 Operating Environment**

- **Blockchain Network:** Ethereum or similar for transaction and contract execution.
- **Client Devices:** Web browsers and mobile devices for user interaction.

### **2.5 Design and Implementation Constraints**

- Must comply with global fundraising regulations.
- High dependency on blockchain uptime and gas fees.

### **2.6 Assumptions and Dependencies**

- Stable internet connectivity for all users.
- Secure wallets for transaction processing.

## **3. System Features**

### **3.1 Functional Requirements**

- **User Registration:** Authenticate users with blockchain wallets.

- **Campaign Creation:** Allow fundraisers to set goals, deadlines, and milestones.
- **Donations:** Enable secure contributions from blockchain wallets.
- **Fund Release:** Automate fund disbursement using milestone-based smart contracts.

### 3.2 Non-Functional Requirements

- **Performance:** Fast transaction processing with minimal latency.
- **Scalability:** Handle increasing users and campaigns without degrading performance.
- **Security:** Blockchain integration for tamper-proof transactions.

### Result:

The Software Requirement Specification (SRS) for the decentralized blockchain-based crowdfunding platform has been successfully developed, addressing functional and non-functional requirements, along with design constraint



<b>EX NO:3</b>	<b>DRAW THE ENTITY RELATIONSHIP DIAGRAM</b>
<b>DATE</b>	

**Aim:**

To draw the Entity Relationship Diagram (ERD) for the decentralized blockchain-based crowdfunding platform.

**Algorithm:**

**1. Mapping of Regular Entity Types**

- Identify the main entities in the platform:
  - Fundraiser
  - Contributor
  - Campaign
  - Administrator

**2. Mapping of Weak Entity Types**

- Determine if there are any weak entities (e.g., Transactions may depend on Campaigns and Contributors).

**3. Mapping of Binary 1:1 Relation Types**

- Identify relationships where each entity is related to only one instance of another entity.
- Example: **Administrator** ↔ **Platform** (One admin manages one platform).

**4. Mapping of Binary 1:N Relationship Types**

- Define relationships where one entity relates to multiple instances of another entity:
  - **Fundraiser** → **Campaign**: One fundraiser can create multiple campaigns.
  - **Campaign** → **Transactions**: One campaign has multiple transactions.

**5. Mapping of Binary M:N Relationship Types**

- Define relationships where many instances of one entity relate to many instances of another entity:
  - **Contributor** ↔ **Campaign**: Multiple contributors can fund multiple campaigns.

**Input:**

● **Entities:**

- Fundraiser
- Contributor
- Campaign
- Administrator
- Transaction

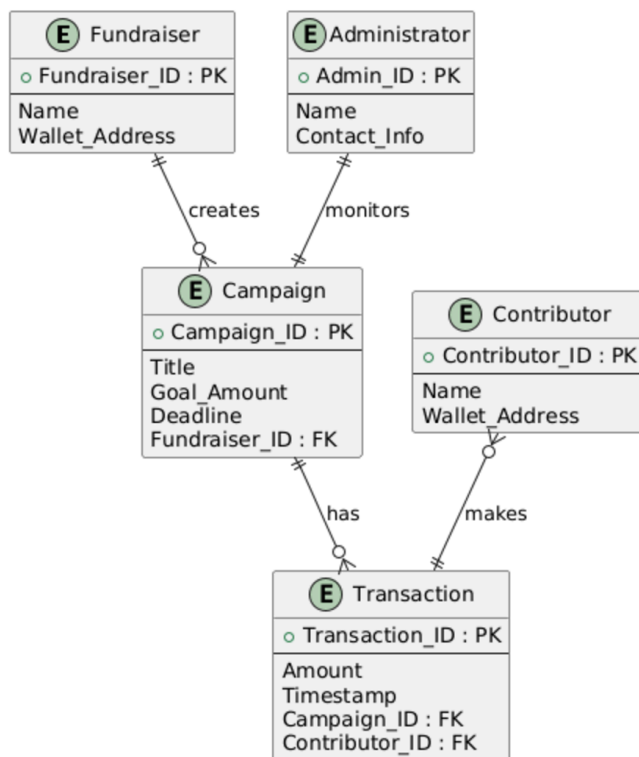
● **Entity Relationship Matrix:**

- Fundraiser ↔ Campaigns: **1:N** (One fundraiser can create multiple campaigns).
- Campaign ↔ Contributors: **M:N** (Many contributors can fund multiple campaigns).

- Campaign ↔ Transactions: **1:N** (One campaign can have multiple transactions).
- **Attributes:**
  - **Fundraiser:** Fundraiser\_ID, Name, Wallet\_Address
  - **Contributor:** Contributor\_ID, Name, Wallet\_Address
  - **Campaign:** Campaign\_ID, Title, Goal\_Amount, Deadline, Fundraiser\_ID (FK)
  - **Transaction:** Transaction\_ID, Amount, Timestamp, Campaign\_ID (FK), Contributor\_ID (FK)
  - **Administrator:** Admin\_ID, Name, Contact\_Info

### Output:

The entity relationship diagram includes:



### Result:

The Entity Relationship Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created, effectively illustrating the relationships and attributes of the platform's core entities.

<b>EX NO:4</b>	<b>DRAW THE DATA FLOW DIAGRAMS AT LEVEL 0 AND LEVEL 1</b>
<b>DATE</b>	

**Aim:**

To draw the Data Flow Diagram (DFD) for the decentralized blockchain-based crowdfunding platform and list the application modules.

**Algorithm:**

1. **Open Tool:**
  - Use tools such as Visual Paradigm, Lucidchart, or any DFD drawing software.
2. **Select Template:**
  - Start with a blank DFD template.
3. **Name the DFD:**
  - Title it appropriately (e.g., "Blockchain-Based Crowdfunding Platform DFD").
4. **Add External Entities:**
  - Include the main actors interacting with the system:
    - Fundraisers
    - Contributors
    - Administrators
5. **Add Processes:**
  - Define key processes in the platform:
    - Create Campaign
    - Donate to Campaign
    - Manage Campaign
    - Monitor Transactions
6. **Add Data Stores:**
  - Identify the key data repositories:
    - Campaign Database
    - User Database
    - Blockchain Ledger
7. **Add Data Flows:**
  - Connect entities, processes, and data stores with labeled arrows showing the flow of data.
8. **Customize the Diagram:**
  - Use different colors or fonts for clarity.
9. **Add Title and Share:**
  - Include a title and ensure the diagram is well-organized for presentation.

**Input:**

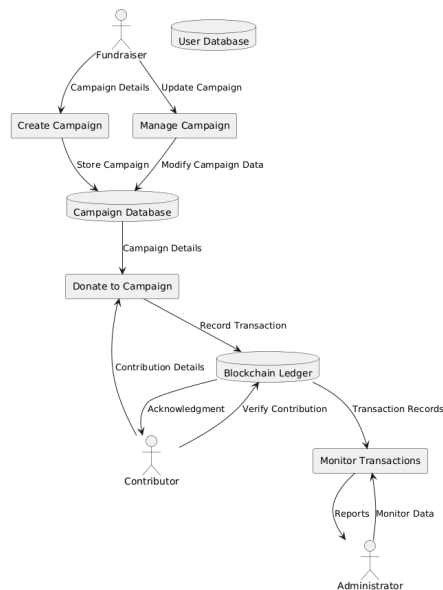
- **Processes:**
  - Create Campaign, Donate to Campaign, Manage Campaign, Monitor Transactions.
- **Data Stores:**
  - Campaign Database, User Database, Blockchain Ledger.
- **External Entities:**
  - Fundraisers, Contributors, Administrators.

### Modules in the Application:

1. **Campaign Management Module**
  - Allows fundraisers to create and manage campaigns.
2. **Donation Module**
  - Handles secure contributions from contributors.
3. **Fund Release Module**
  - Automates fund allocation using smart contracts.
4. **Admin Module**
  - Provides monitoring and administrative capabilities for campaigns and transactions.
5. **Transaction Tracking Module**
  - Ensures real-time tracking of all contributions and fund disbursement.

### Output:

### Data Flow Diagram (DFD):



### Result:

The Data Flow Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created, detailing the interaction between external entities, processes, and data stores. Application modules are clearly defined for efficient system design and development.

<b>EX NO:5</b>	<b>DRAW USE CASE DIAGRAM</b>
<b>DATE</b>	

**Aim:**

To draw the Use Case Diagram for the decentralized blockchain-based crowdfunding platform.

**Algorithm:**

**1. Identify Actors:**

- Determine the key actors interacting with the system:
  - **Fundraiser:** Creates and manages campaigns.
  - **Contributor:** Makes contributions to campaigns.
  - **Administrator:** Monitors and manages campaigns and users.

**2. Identify Use Cases:**

- List the key tasks each actor performs:
  - **Fundraiser:**
    - Create Campaign
    - Update Campaign
    - View Campaign Status
  - **Contributor:**
    - Register
    - Donate to Campaign
    - View Campaigns
  - **Administrator:**
    - Monitor Campaigns
    - Manage Transactions
    - Manage Users

**3. Define Relationships:**

- Identify the relationships between actors and use cases:
  - Use "include" if a use case is always executed as part of another.
  - Use "extend" if a use case can optionally extend another.
  - Use lines to connect actors with their respective use cases.

**4. Draw the Diagram:**

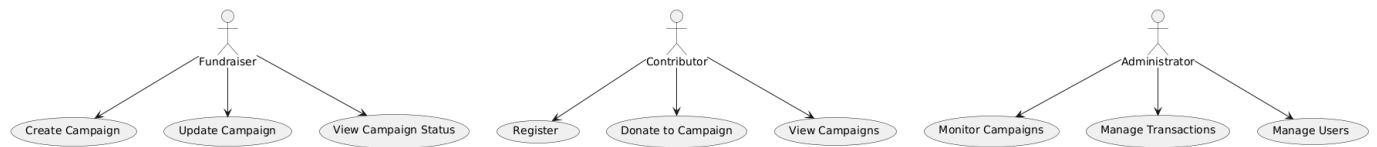
- Use appropriate UML notations to represent the system, actors, and use cases.

**Input:**

- **Actors:**
  - Fundraiser
  - Contributor
  - Administrator
- **Use Cases:**
  - Fundraiser: Create Campaign, Update Campaign, View Campaign Status
  - Contributor: Register, Donate to Campaign, View Campaigns
  - Administrator: Monitor Campaigns, Manage Transactions, Manage Users

## Output:

- **Use Case Diagram**



## Result:

The Use Case Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created, showing the relationships between the system's actors and their associated use cases. This will help in understanding the primary functions of the platform and the roles of each user group.

<b>EX NO:6</b>	<b>DRAW ACTIVITY DIAGRAM OF ALL USE CASES.</b>
<b>DATE</b>	

**Aim:**

To draw the Activity Diagram for the decentralized blockchain-based crowdfunding platform.

**Algorithm:**

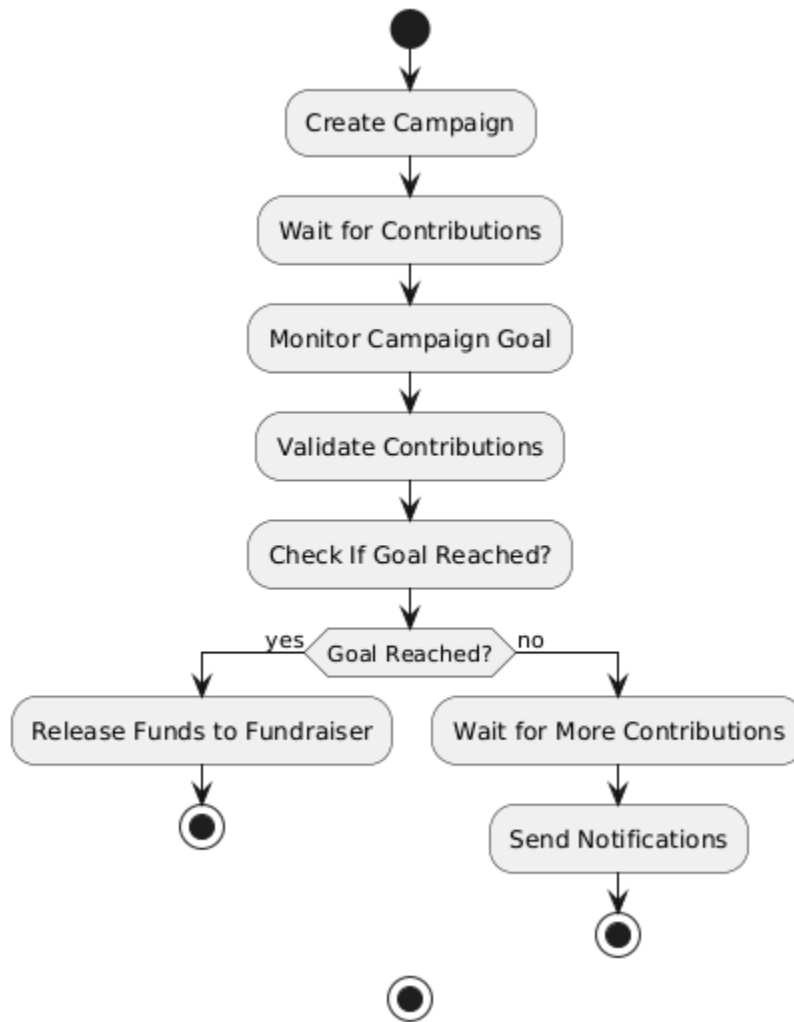
1. **Identify Initial State and Final State:**
  - Define the starting point (initial state) and the endpoint (final state) of the activity.
2. **Identify Intermediate Activities:**
  - List the main activities that occur between the initial and final states.  
For example:
    - Fundraiser creates a campaign.
    - Contributor registers and makes a donation.
3. **Identify Decision Points:**
  - Determine decision points that affect the flow of activities.  
For example:
    - "Is the goal amount reached?" for releasing funds.
4. **Identify Parallel Activities:**
  - Identify tasks that can occur in parallel, such as campaign creation and contribution.
5. **Draw the Diagram with Appropriate Notations:**
  - Use ovals for initial and final states.
  - Use rectangles for activities.
  - Use diamonds for decision points, and arrows for the flow of control.
6. **Review and Refine:**
  - Ensure that all activities, conditions, and decision points are captured accurately.

**Input:**

- **Activities:**
  - Create Campaign, Register for Campaign, Donate to Campaign, Monitor Fund Status, Fund Release.
- **Decision Points:**
  - Is the campaign goal amount reached?
  - Is the contributor's registration valid?
- **Parallel Activities:**
  - Contributors can donate while the campaign is active.
- **Conditions:**
  - Ensure mandatory fields are filled when creating a campaign.

- Check if the contributor's funds are verified before accepting donations.

**Output:**



**Result:**

The Activity Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created, illustrating the workflow of activities involved in campaign creation, donation, fund allocation, and monitoring. This diagram provides a clear representation of how users interact with the system and the various decision points that impact the flow of activities.



<b>EX NO:7</b>	<b>DRAW STATE CHART DIAGRAM OF ALL USE CASES.</b>
<b>DATE</b>	

**Aim:**

To draw the State Chart Diagram for the decentralized blockchain-based crowdfunding platform.

**Algorithm:**

1. **Identify Important Objects to be Analyzed:**
  - Determine the key objects involved in the system:
    - **Campaign**
    - **Fundraiser**
    - **Contributor**
2. **Identify the States:**
  - List the possible states of each object throughout its lifecycle.  
For example:
    - **Campaign:** Planned, Active, Funded, Completed, Canceled
    - **Fundraiser:** Active, Inactive, Managing Campaigns
    - **Contributor:** Registered, Contributing, Completed Donation
3. **Identify the Events:**
  - Define the events that trigger state changes (e.g., Campaign Created, Donation Made, Campaign Funded).
4. **Draw the Diagram:**
  - Use standard notations for states (rectangles), transitions (arrows), and events (labels on transitions).
5. **Review and Refine:**
  - Ensure that the diagram captures all the states and transitions accurately.

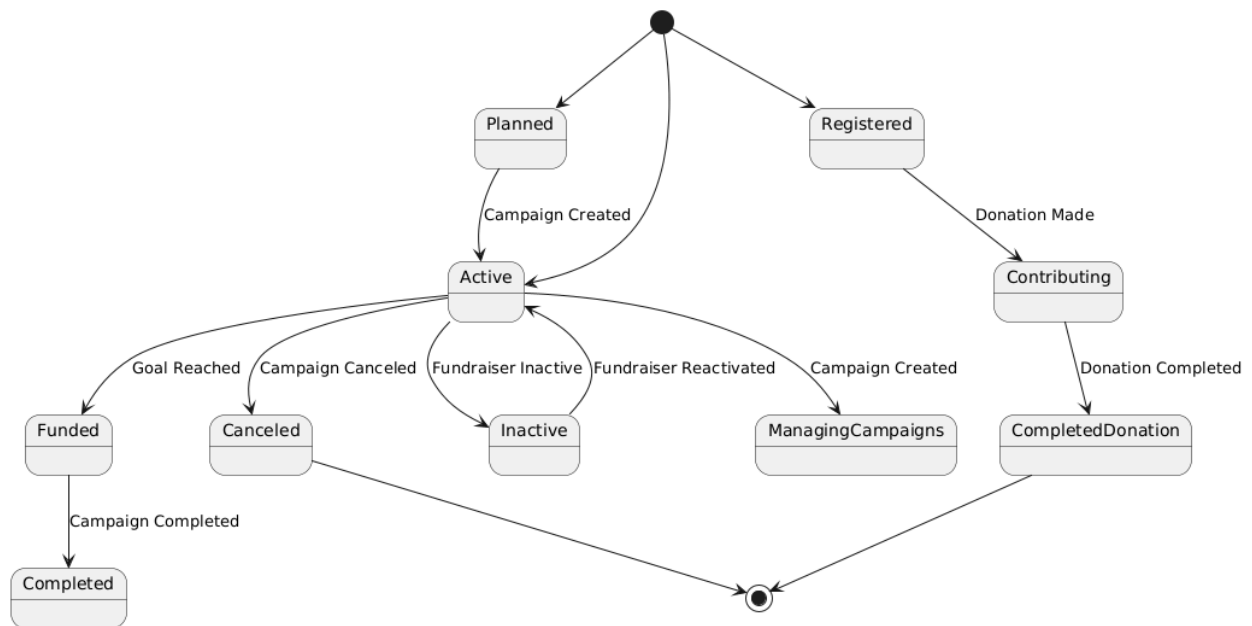
**Input:**

- **Objects:**
  - **Campaign**
  - **Fundraiser**
  - **Contributor**
- **States:**
  - **Campaign:**
    - Planned, Active, Funded, Completed, Canceled
  - **Fundraiser:**
    - Active, Inactive, Managing Campaigns

- **Contributor:**
  - Registered, Contributing, Completed Donation
- **Events:**
  - **Campaign:** Campaign Created, Campaign Started, Campaign Funded, Campaign Canceled
  - **Fundraiser:** Fundraiser Created, Fundraiser Inactive
  - **Contributor:** Contributor Registered, Donation Made, Donation Completed

**Output:**

**State Chart Diagram:**



**Result:**

The State Chart Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created. It shows the possible states of key entities and their transitions, including the events that trigger each transition. This diagram helps in understanding the lifecycle of objects and their interactions in the system.

<b>EX NO:8</b>	<b>DRAW SEQUENCE DIAGRAM OF ALL USE CASES.</b>
<b>DATE</b>	

**Aim:**

To draw the Sequence Diagram for the decentralized blockchain-based crowdfunding platform.

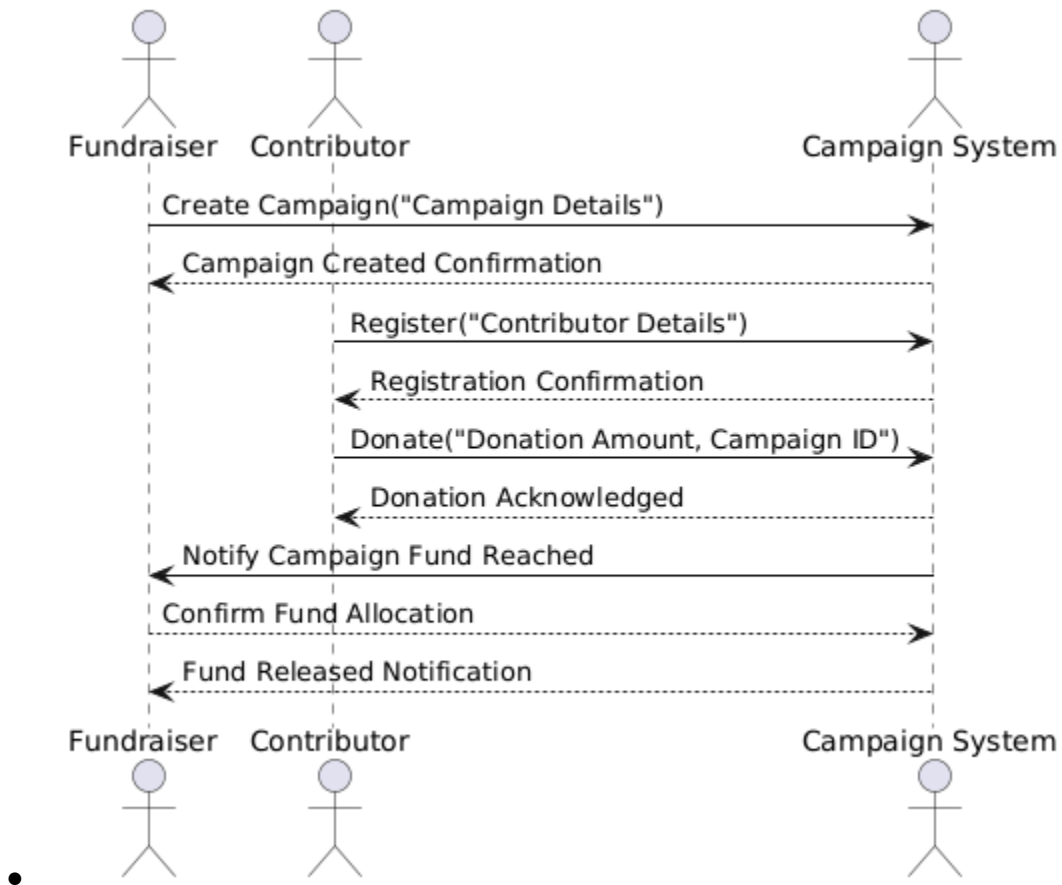
**Algorithm:**

1. **Identify the Scenario:**
  - Determine the scenario to be depicted in the sequence diagram (e.g., Contributor registering and donating to a campaign).
2. **List the Participants:**
  - Identify all participants involved in the interaction (e.g., Fundraiser, Contributor, Campaign System).
3. **Define Lifelines for Each Participant:**
  - Lifelines represent the timeline for each participant. Each lifeline is shown as a vertical dashed line.
4. **Add Activation Bars:**
  - Represent the period when a participant is active in the process.
5. **Draw Messages Between Participants:**
  - Show the interaction between participants with arrows. Label each arrow with the message or function being called.
6. **Include Return Messages:**
  - Include return arrows to represent the return of control after a message is processed.
7. **Include Timing and Order of Events:**
  - Ensure the sequence of events is depicted chronologically.
8. **Add Conditions and Loops (if applicable):**
  - If a process repeats, use loops or conditions to depict it.
9. **Review and Refine:**
  - Make sure the sequence is correct and all interactions are captured.

**Input:**

- **Scenario:**
  - A Contributor registers and donates to a campaign.
- **Participants:**
  - Fundraiser
  - Contributor
  - Campaign System
- **Messages:**
  - Registration Request, Confirmation Response, Donation Request, Acknowledgment, etc.

**Output:**



**Result:**

The Sequence Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created. It depicts the interactions between participants, such as the process of registration and donation by the Contributor, and the responses from the system.

<b>EX NO:9</b>	<b>DRAW COLLABORATION DIAGRAM OF ALL USE CASES</b>
<b>DATE</b>	

**Aim:**

To draw the Collaboration Diagram for the decentralized blockchain-based crowdfunding platform.

**Algorithm:**

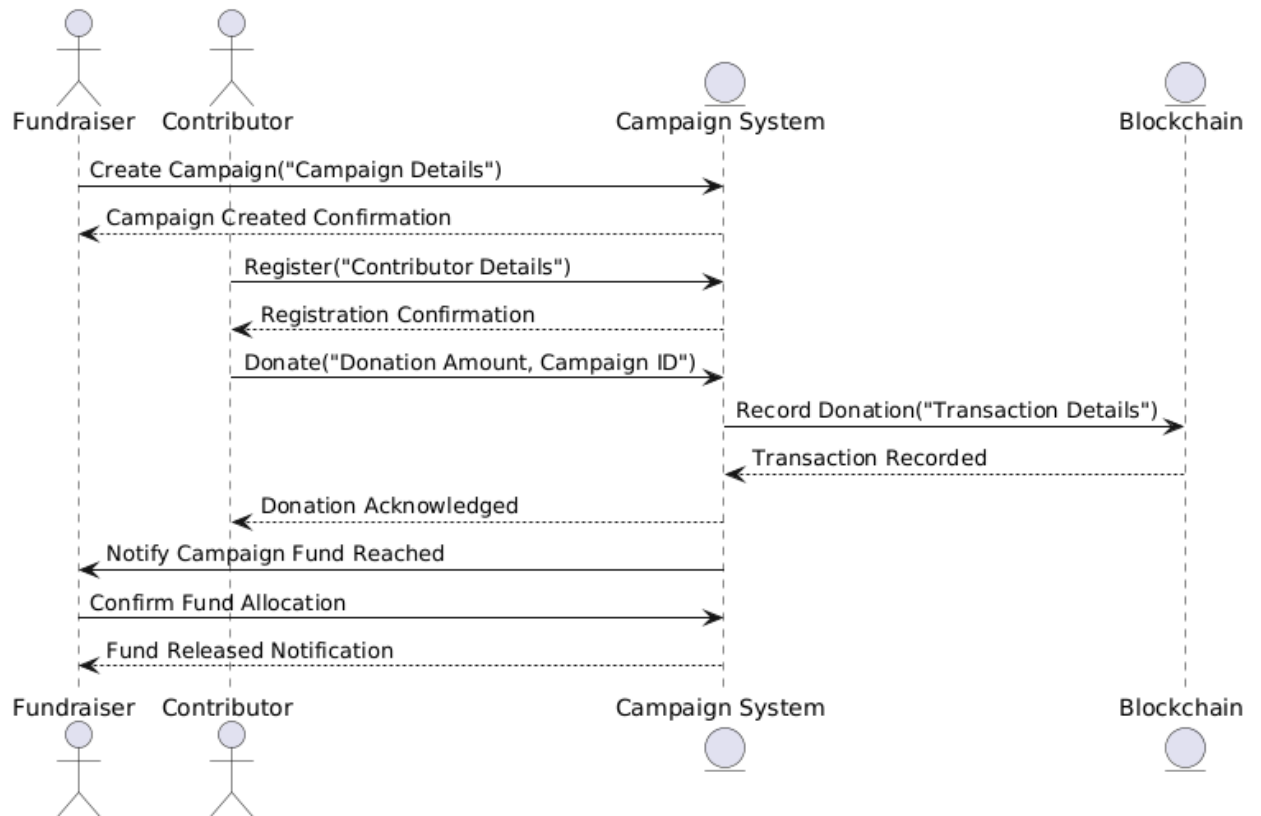
1. **Identify Objects/Participants:**
  - Identify the key participants (objects) in the system.
  - For example:
    - **Fundraiser**
    - **Contributor**
    - **Campaign System**
    - **Blockchain**
2. **Define Interactions:**
  - Determine how these participants interact with each other.
  - Identify what messages or requests are exchanged between objects.
3. **Add Messages:**
  - Document the messages that are exchanged, including the order of the messages.
4. **Consider Relationships:**
  - Define how objects relate to one another (associations).
  - Consider if any participants are directly dependent on others.
5. **Draw the Diagram:**
  - Use UML notations for objects (rectangles) and the messages exchanged (arrows).
6. **Document the Collaboration Diagram:**
  - Ensure the diagram is neat, clear, and understandable, with relevant annotations or comments.

**Input:**

- **Objects/Participants:**
  - **Fundraiser**
  - **Contributor**
  - **Campaign System**
  - **Blockchain**
- **Messages/Interactions:**
  - Fundraiser creates campaign, Contributor registers, Contributor donates, and the system verifies and processes the donation.
- **Relationships:**
  - Contributor interacts with Campaign System.

- Fundraiser interacts with Campaign System.
- Blockchain records the donation.

### Output:



### Result:

The Collaboration Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created. It illustrates how the Fundraiser, Contributor, Campaign System, and Blockchain interact to create and fund a campaign, ensuring transparency and traceability.

<b>EX NO:10</b>	<b>ASSIGN OBJECTS IN SEQUENCE DIAGRAM TO CLASSES AND MAKE CLASS DIAGRAM.</b>
<b>DATE</b>	

**Aim:**

To draw the Class Diagram for the decentralized blockchain-based crowdfunding platform.

**Algorithm:**

**1. Identify Classes:**

- Determine the main classes required for the system:

- **Campaign**
- **Fundraiser**
- **Contributor**
- **Transaction**
- **Administrator**

**2. List Attributes and Methods:**

- Define the attributes (data members) and methods (functions) for each class.

- Example:

- **Campaign Class:**

- Attributes: `CampaignID`, `Title`, `GoalAmount`, `Deadline`
- Methods: `createCampaign()`, `updateCampaign()`, `getDetails()`

**3. Identify Relationships:**

- Determine how classes interact with each other:

- **Fundraiser** can create **Campaign** (1:N)
- **Contributor** can donate to multiple **Campaigns** (M:N)
- **Transaction** stores details for donations made by **Contributors** to **Campaigns**.

**4. Create Class Boxes:**

- Draw the class boxes, including the class name, attributes, and methods.

**5. Draw Relationships:**

- Use lines to represent relationships between classes:

- **Associations:** One-to-many or many-to-many relations between classes.
- **Visibility:** Use + for public, - for private attributes/methods.
- **Methods:** Define the behavior and actions for each class.

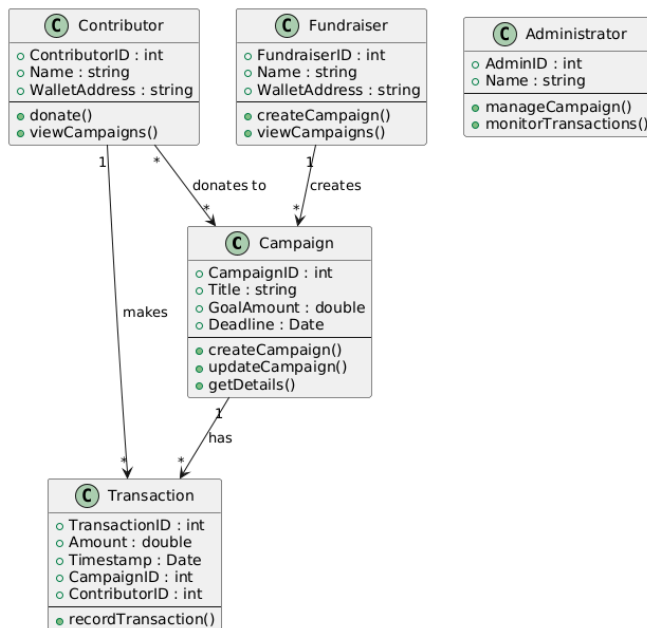
**6. Review and Refine:**

- Ensure that all necessary classes, attributes, methods, and relationships are captured.

**Input:**

- **Classes:**
  - **Campaign**
  - **Fundraiser**
  - **Contributor**
  - **Transaction**
  - **Administrator**
- **Attributes and Methods:**
  - **Campaign:** CampaignID, Title, GoalAmount, Deadline, createCampaign(), updateCampaign(), getDetails()
  - **Fundraiser:** FundraiserID, Name, WalletAddress, createCampaign(), viewCampaigns()
  - **Contributor:** ContributorID, Name, WalletAddress, donate(), viewCampaigns()
  - **Transaction:** TransactionID, Amount, Timestamp, CampaignID, ContributorID, recordTransaction()
  - **Administrator:** AdminID, Name, manageCampaign(), monitorTransactions()

### Output:



### Result:

The Class Diagram for the decentralized blockchain-based crowdfunding platform has been successfully created, showing the main classes, their attributes, methods, and the relationships between them.



<b>EX NO:11</b>	<b>MINI PROJECT- DECENTRALIZED BLOCKCHAIN-BASED CROWDFUNDING PLATFORM</b>
<b>DATE</b>	

### **Aim:**

To develop a decentralized crowdfunding platform using blockchain technology that enables transparent, secure, and efficient fundraising by eliminating intermediaries and providing real-time transaction tracking via smart contracts.

### **Algorithm:**

- 1. User Registration:**  
Users (fundraisers, contributors) register with blockchain wallets for secure access.
- 2. Campaign Creation:**  
Fundraisers create campaigns with details (goal, deadline). Campaigns are stored on the platform.
- 3. Donations:**  
Contributors donate to campaigns using blockchain wallets, recorded on the blockchain ledger.
- 4. Fund Allocation:**  
Smart contracts release funds to the fundraiser once the donation goal is met.
- 5. Transaction Tracking:**  
Blockchain tracks all donations and fund releases for transparency.
- 6. Campaign Monitoring:**  
Administrators monitor campaigns, ensuring compliance and tracking fund usage.
- 7. Notifications:**  
Users receive notifications for goal achievement, fund releases, and milestones.

### **PROGRAM:**

```
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;
}

```

```

    );
    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;
}

```

```

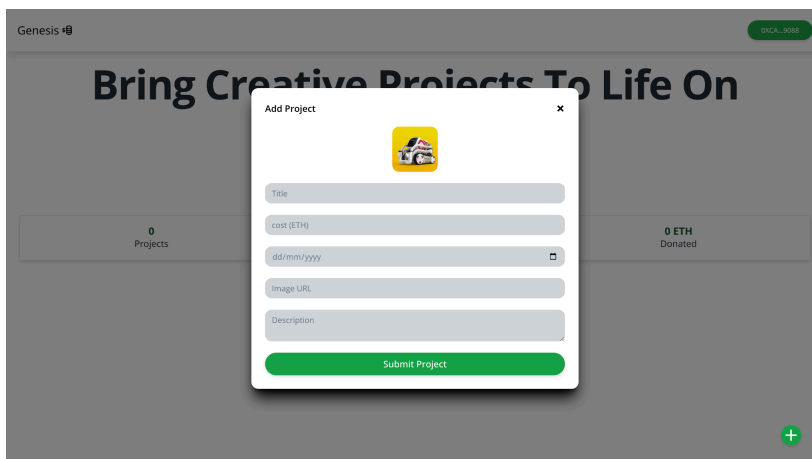
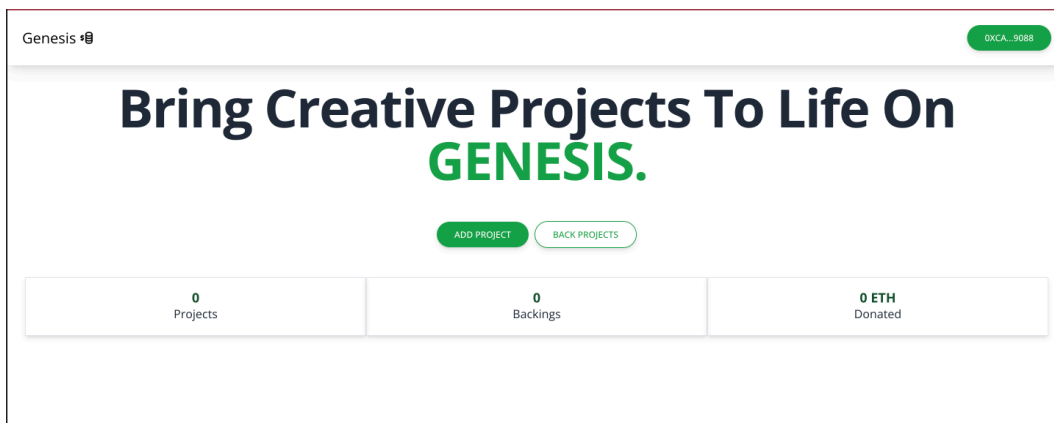
function changeTax(uint _taxPct) public ownerOnly {
    projectTax = _taxPct;
}
function getProject(uint id) public view returns (projectStruct memory) {
    require(projectExist[id], "Project not found");
    return projects[id];
}


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);
}
}


```

## OUTPUT:




Genesis 

0x70...79C8



**GREEN DISC | The smart bike chain care tool**

3 days left

 0x70...79C8 0 Backings

Open


GREEN DISC supports you maintaining your bike – uncomplicated, accurate and eco-friendly. Just press it slightly onto your chain and let your crank rotate backwards. Three turns on the outside, three on the inside of the chain – that's it.


---

0 ETH Raised

17 EHT

BACK THIS PROJECT EDIT DELETE

Genesis 


Account 4  0x5Fb...0aa3

New address detected! Click here to add to your address book.

\$5,054.20

DETAILS DATA HEX

EDIT

**Estimated gas fee**  \$0.48 **0.000382 ETH**

Site suggested Likely in < 30 seconds

Max fee: 0.00040428 ETH

---


**Total** \$5,054.68

**4.00038232 ETH**

Amount + gas fee Max amount: 4.00040428 ETH


Reject Confirm

**#GREEN DISC | The smart bike chain care tool**



4


Back Project

Genesis 


2 Projects

4 Backing

20.8 ETH Donated




**GREEN DISC | The smart bike chain care tool**


 0x70...79C8 3 days left

17.8 ETH Raised 17 EHT

3 Backings Paid



**Mini Pupper 2: Open-Source, ROS2 Robot Kit for Dreamers**

 0x90...b906 2 days left

3 ETH Raised 14 EHT

1 Backings Open