CrowdChain Smart Contract Documentation

Overview

The CrowdChain smart contract is a decentralized crowdfunding platform that allows users to create and fund projects. The enhanced version now includes milestone-based funding, which enables project owners to receive funds in phases as they complete predefined milestones.

Contract Structure

Core Entities

1. **Projects**: The main entity representing a crowdfunding campaign with details like title, description, funding goal, etc.
2. **Backers**: Users who contribute funds to projects.
3. **Milestones**: Specific goals or phases within a project that must be completed to release portions of the funding.

Enums

1. **statusEnum**: Represents the status of a project
   - `OPEN`: Project is accepting contributions
   - `APPROVED`: Project has reached its funding goal
   - `REVERTED`: Project has been reverted (funds returned to backers)
   - `DELETED`: Project has been deleted
   - `PAIDOUT`: Project funds have been paid out to the owner
2. **milestoneStatusEnum**: Represents the status of a milestone
   - `PENDING`: Milestone is awaiting approval
   - `APPROVED`: Milestone has been approved but not executed
   - `REJECTED`: Milestone has been rejected
   - `EXECUTED`: Milestone has been executed and funds released

Structs

1. **statsStruct**: Tracks platform-wide statistics
   - `totalProjects`: Total number of projects created
   - `totalBacking`: Total number of backing transactions
   - `totalDonations`: Total amount of ETH donated
   - `totalMilestones`: Total number of milestones created
   - `totalMilestonesCompleted`: Total number of milestones completed
2. **backerStruct**: Represents a backer of a project
   - `owner`: Address of the backer
   - `contribution`: Amount contributed
   - `timestamp`: Time of contribution
   - `refunded`: Whether the contribution has been refunded
3. **milestoneStruct**: Represents a milestone within a project

    - `id`: Unique identifier for the milestone
    - `title`: Title of the milestone
    - `description`: Description of the milestone
    - `amount`: Amount of funds to be released upon completion
    - `yesVotes`: Number of votes in favor of the milestone
    - `noVotes`: Number of votes against the milestone
    - `createdAt`: Time when the milestone was created
    - `completedAt`: Time when the milestone was completed
    - `status`: Current status of the milestone

4. **projectStruct**: Represents a crowdfunding project
    - `id`: Unique identifier for the project
    - `owner`: Address of the project creator
    - `title`: Title of the project
    - `description`: Description of the project
    - `imageURL`: URL to the project image
    - `cost`: Total funding goal
    - `raised`: Amount raised so far
    - `timestamp`: Time when the project was created
    - `expiresAt`: Time when the project expires
    - `backers`: Number of backers
    - `milestoneCount`: Number of milestones
    - `milestonesCompleted`: Number of completed milestones
    - `hasMilestones`: Whether the project uses milestone-based funding
    - `status`: Current status of the project

Core Functions

Project Management

1. **createProject**: Creates a new crowdfunding project

    ```
    function createProject(
        string memory title,
        string memory description,
        string memory imageURL,
        uint cost,
        uint expiresAt,
        bool _hasMilestones
    ) public returns (bool)
    ```

2. **updateProject**: Updates an existing project's details

    ```
    function updateProject(
        uint id,
        string memory title,
        string memory description,
        string memory imageURL,
        uint expiresAt
    ```

```
) public returns (bool)
```

3. **deleteProject**: Deletes a project and refunds all backers

```
function deleteProject(uint id) public returns (bool)
```

Funding Operations

1. **backProject**: Allows a user to back (fund) a project

```
function backProject(uint id) public payable returns (bool)
```

2. **requestRefund**: Allows a user to request a refund for a project

```
function requestRefund(uint id) public returns (bool)
```

3. **payOutProject**: Pays out the project funds to the owner

```
function payOutProject(uint id) public returns (bool)
```

Milestone Management (New)

1. **createMilestone**: Creates a new milestone for a project

```
function createMilestone(
    uint projectId,
    string memory title,
    string memory description,
    uint amount
) public returns (bool)
```

2. **voteMilestone**: Allows a backer to vote on a milestone

```
function voteMilestone(
    uint projectId,
    uint milestoneId,
    bool approve
) public returns (bool)
```

3. **executeMilestone**: Executes a milestone and releases funds

```
function executeMilestone(
    uint projectId,
    uint milestoneId
) public returns (bool)
```

4. **rejectMilestone**: Rejects a milestone

```
function rejectMilestone(
    uint projectId,
    uint milestoneId
) public returns (bool)
```

View Functions

1. **getProject**: Returns details of a specific project

   ```
   function getProject(uint id) public view returns (projectStruct memory)
   ```

2. **getProjects**: Returns all projects

   ```
   function getProjects() public view returns (projectStruct[] memory)
   ```

3. **getBackers**: Returns all backers of a project

   ```
   function getBackers(uint id) public view returns (backerStruct[] memory)
   ```

4. **getMilestones**: Returns all milestones of a project

   ```
   function getMilestones(uint projectId) public view returns (milestoneStruct[] memory)
   ```

5. **getMilestone**: Returns details of a specific milestone

   ```
   function getMilestone(uint projectId, uint milestoneId) public view returns (milestoneS
   ```

Milestone-Based Funding Flow

1. **Project Creation with Milestones**:
   - Project owner creates a project with `hasMilestones` set to `true`
   - Backers contribute to the project
   - Once the funding goal is reached, the project status changes to `APPROVED`
2. **Milestone Creation**:
   - Project owner creates milestones for the project
   - Each milestone has a title, description, and amount to be released
3. **Milestone Voting**:
   - Backers vote on milestones (approve or reject)
   - A milestone can be executed if it has more yes votes than no votes
4. **Milestone Execution**:
   - When a milestone is executed, the specified amount is released to the project owner
   - The milestone status changes to `EXECUTED`
   - The project's `milestonesCompleted` counter is incremented
5. **Project Completion**:
   - A project with milestones is considered complete when all milestones are executed
   - The project owner can call `payOutProject` to receive any remaining funds only after all milestones are completed

Events

1. **Action**: Emitted for various project-related actions

   ```
   event Action(
   ```

```
    uint256 id,
    string actionType,
    address indexed executor,
    uint256 timestamp
)
```

2. **MilestoneAction**: Emitted for various milestone-related actions

```
event MilestoneAction(
    uint256 projectId,
    uint256 milestoneId,
    string actionType,
    address indexed executor,
    uint256 timestamp
)
```

Modifiers

1. **ownerOnly**: Restricts function access to the contract owner

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

2. **projectOwnerOnly**: Restricts function access to the project owner

```
modifier projectOwnerOnly(uint projectId) {
    require(msg.sender == projects[projectId].owner, "Project owner reserved only");
    _;
}
```

3. **validProject**: Ensures the project exists

```
modifier validProject(uint projectId) {
    require(projectExist[projectId], "Project not found");
    _;
}
```

4. **isContributor**: Ensures the caller is a contributor to the project

```
modifier isContributor(uint projectId) {
    bool isContributor = false;
    for(uint i = 0; i < backersOf[projectId].length; i++) {
        if(backersOf[projectId][i].owner == msg.sender) {
            isContributor = true;
            break;
        }
    }
    require(isContributor, "Only contributors can vote on milestones");
```

```
        _;
    }
```

Backward Compatibility

The enhanced contract maintains backward compatibility with the original contract by:

1. Making the milestone functionality optional through the `hasMilestones` flag
2. Preserving all original functions and their behavior
3. Only adding new functions and modifying existing ones in a way that doesn't break existing functionality

Projects created without milestones will behave exactly as they did in the original contract, while projects with milestones will use the new phased funding mechanism.

Security Considerations

1. **Fund Safety**: Funds are only released to project owners after milestones are approved by backers
2. **Voting Mechanism**: Each backer can only vote once on each milestone
3. **Milestone Amounts**: The total amount allocated to milestones cannot exceed the total raised amount
4. **Access Control**: Various modifiers ensure that only authorized users can perform certain actions