Crowdfunding Platform Design

1. Introduction

This document outlines the design and functionality of a crowdfunding platform that supports two main user types: *Project Managers* and *Contributors*[1]. The platform will enable project managers to start and manage campaigns to raise capital and contributors to fund these projects. Transactions on the platform will be exclusively conducted in Bitcoin, leveraging its security and blockchain transparency.

Key features of the platform include:

- **Campaign Management:** Project managers can create, edit, and manage campaigns, set fundraising goals and deadlines, and provide updates to contributors.
- **Contributions and Refunds:** Contributors can view ongoing campaigns, make donations, and receive refunds if the goal is not met or if a vote of no confidence is passed.
- **Installments:** Funded campaigns will receive funds in installments rather than a lump sum.
- **Voting Mechanism:** Contributors can participate in votes of no confidence regarding campaign progress, with weighted voting based on their contribution amounts.

[1] The terms "*Contributor(s)*" and "*Donor(s)*" are used interchangeably in this document. They are equivalent.

2. High-Level System Design

2.1 Frontend

The frontend will be built using modern web technologies such as React for a responsive and interactive user experience. The application will have two primary interfaces tailored for project managers and contributors.

Account Management Interface

- **Account Creation:** Form for new users to register by providing their email, password, and selecting their account type (*Project Manager* or *Contributor*).
- **Email Verification:** Inform users that a verification email has been sent and to prompt them to check their email to activate their account.
- **Login:** Form for existing users to log in using their email and password.
- **Password Reset Request:** Form for users to request a password reset by entering their email, triggering an email with a reset link that expires after a set period.
- **Password Reset:** Form for users to enter a new password after clicking the reset link in the email.

Project Manager Interface

- **Campaign Dashboard:** Overview of all campaigns created by the manager
- **Create/Edit Campaign:** Ability to input and modify campaign details including:
    1. Title
    2. Description
    3. Crowdfunding goal
    4. Crowdfunding deadline
    5. Campaign web URL (if applicable)
    6. Project manager email (ability to switch to different email account or reassign the campaign to another project manager if needed)
- **Update Campaign:** Ability to post updates for campaigns that meet the crowdfunding goal
- **Status:** View the campaign status which can be one of the following:
    - *Activation pending*: The manager has not published the campaign (e.g., still working on the campaign details). This is the initial status.
    - *Active*: The campaign is published, the crowdfunding goal is not met, and the deadline has not passed. Sequence: *Activation pending* $\rightarrow$ *Active*
    - *Vote*: The manager submitted a milestone for a funded campaign. The contributor can now vote. Sequence: *Activation pending* $\rightarrow$ *Active* $\rightarrow$ *Funded* $\rightarrow$ Vote
    - *Funded*: The campaign is published, and the crowdfunding goal is met before the deadline. Sequence: *Activation pending* $\rightarrow$ *Active* $\rightarrow$ *Funded*

      OR

      The campaign is funded and a vote of no confidence is passed. Sequence: *Activation pending* $\rightarrow$ *Active* $\rightarrow$ *Funded* $\rightarrow$ Vote $\rightarrow$ *Funded*

  - *Expired*: The campaign is published, the crowdfunding goal is not met and the deadline passed. Sequence: *Activation pending* → *Active* → *Expired*
  - *Canceled*: The campaign is funded and a vote of no confidence is passed. Sequence: *Activation pending* → *Active* → *Funded* → Vote → *Canceled*
- **Raised Percentage:** For every campaign show the target amount percentage that is raised

Contributor Interface

- **Campaign Dashboard:** List of all active campaigns with filtering and search capabilities.
- **Campaign Information:** Detailed view of each campaign including:
  1. Title
  2. Description
  3. Paid amount (sum of installments so far) for a funded campaign that the contributor donated to
  4. Donation amount (total amount that the contributor will pay amount if all installments complete)
  5. Raised Percentage (percentage of target amount that is raised)
  6. Crowdfunding goal
  7. Crowdfunding deadline
  8. Campaign web URL (if applicable)
  9. Campaign status (i.e., one of the following: Activation pending, Active, Funded, Expired, Canceled)
  10. Campaign updates by the manager (for funded campaigns)
  11. Project manager email (a way to contact the project manager)
- **Donation Form:** Interface to contribute a specific amount to a campaign. Ability to donate multiple times to the same campaign (a new donation will increment the donation total).
- **Voting Form:** Interface to participate in no-confidence votes for installment releases. The contributor will receive an email notification to vote. The email will grant access to the voting form for a set period (i.e., deadline to vote).

2.2 Backend

The backend will be composed of multiple microservices, each responsible for a specific domain. These services will communicate via REST APIs and use secure protocols for handling sensitive data.

2.2.1 Database

A relational database like PostgreSQL will be used for storing structured data such as:

- User Information
- Campaign Details
- Donor Contribution (to a given campaign)
- Donor Installments (for a given campaign)
- Donor Vote
- Project Manager Updates (for funded campaigns)

2.2.1.1 User Information

The table below summarizes the user information that is stored in the database:

User		
Variable Name	**Value Type**	**Description**
email	String	User email (unique identifier)
name	String	User's (display) name
address	String	User's address on the blockchain
password	String	Encrypted user password
isProjectManager	Boolean	*True* for a project manager *False* for a contributor
isVerified	Boolean	*True* if the user account is verified *False* if the user account is not verified
forgotPasswordToken	String	Randomly generated token (64 bytes) to reset the user password
forgotPasswordTokenExpiry	DateTime	Expiration date for the randomly generated token to reset the password
contributions	String[]	The contract addresses that the user has donated to

When a project manager creates and publishes a new campaign, a smart contract (see section 2.2.2) is deployed, and its address is returned. This smart contract manages all campaign transactions (donations and refunds), oversees voting once the campaign is funded, and updates the campaign status accordingly. The smart contract should store only the minimum necessary information. Additional details, such as the campaign description, campaign URL, milestone descriptions, text updates, etc., should be stored in the PostgreSQL database (see the Campaign table).

2.2.1.2 Campaign Details

The table below summarizes the campaign information is stored in the database:

Campaign		
Variable Name	**Value Type**	**Description**
campaign_id	String	Campaign unique identifier *
contract_addr	String	Campaign's address on the blockchain
managerEmail	String	Project manager email
title	String	Campaign title

info	String	Campaign description/info
updates	Update[ ]	Updates if the campaign is funded or an empty array otherwise
link	String	Campaign web url (if any)

Considerations:

1. One way to create the *campaign_id* is by concatenating the project manger's email and the campaign creation timestamp.
2. Although the campaign's address on the blockchain can be used as campaign_id it is advised to avoid that. For example, the campaign_id can be used in Web URLs to navigate to campaign pages. Addresses on the blockchain should not be exposed for security reasons.

2.2.1.3 Project Manager Updates

The table below summarizes the information about a funded campaign update stored in the database:

Update		
Variable Name	**Value Type**	**Description**
campaign	Campaign	Campaign the update is for
text	String	Update content/details
postedDate	Date	Update post timestamp

2.2.1.2 Schema Design

User Table		
id	Int	@id @default(autoincrement()) @map("_id")
email	String	@unique
address	String	@unique
name	String	
password	String	
isProjectManager	Boolean	@default(false)
isVerified	Boolean	@default(false)
forgotPasswordToken	String?	
forgotPasswordTokenExpiry	DateTime?	
contributions	String[]	

Campaign Table		
id	Int	@id @default(autoincrement()) @map("_id")
campaign_id	String	@unique
contract_addr	String	@unique
managerEmail	String	
title	String	
info	String	
updates	Update[]	
link	String?	

Update Table		
id	Int	@id @default(autoincrement()) @map("_id")
campaign	Campaign	@relation(fields: [cid], references: [id], onDelete: Cascade)
cid	Int	
text	String	
postedDate	DateTime	

2.2.2 Bitcoin Blockchain

The platform will integrate with the Bitcoin blockchain to manage all financial transactions. Bitcoin provides transparency, security, and immutability for transactions. This section details how transaction are managed.

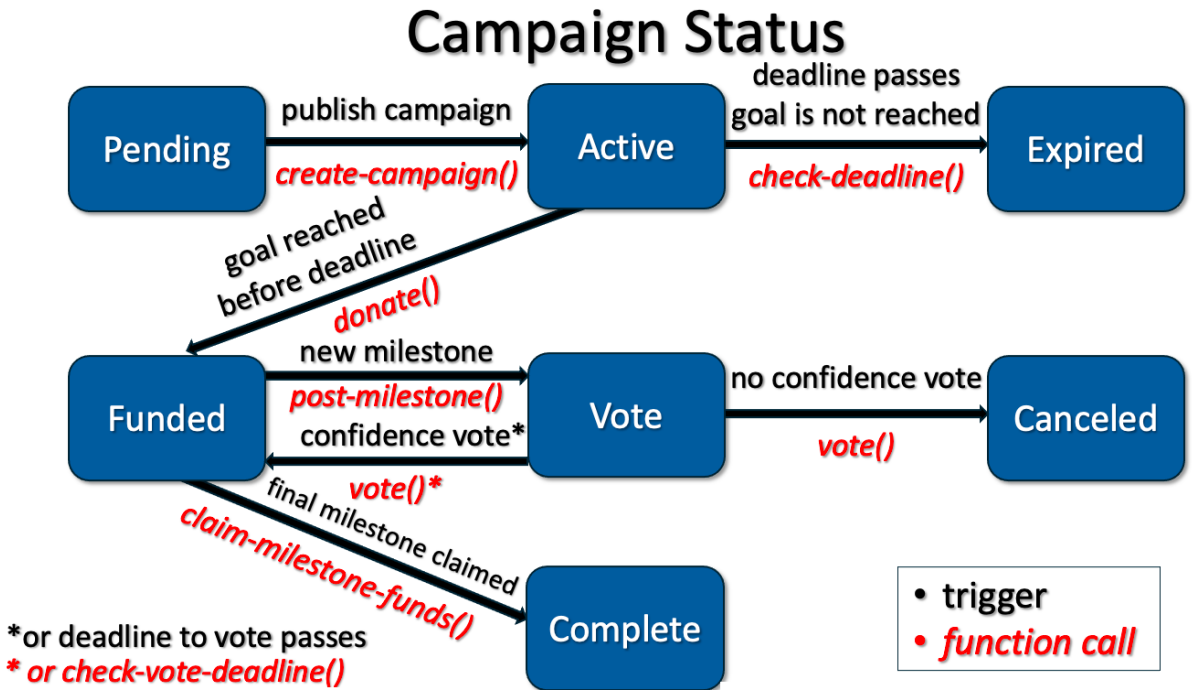
2.2.2.1 Donation Process and Fund Reservation with Smart Contracts

- **User Donation**
  - When a user decides to donate to a campaign, they send their funds to the campaign's smart contract address.
  - The smart contract deployed on the Stacks platform blockchain will hold the funds securely until the specified conditions for release are met.
- **Fund Holding**
  - The donated amount is held within the smart contract according to its predefined rules for how and when the funds can be released.
  - The funds are reserved and cannot be accessed by the project manager immediately.
- **Installment Releases**
  - Upon reaching a predefined milestone, the project manager or a trusted entity submits proof to the smart contract.
  - The smart contract verifies the proof and releases the specified installment amount to the project manager's address on the blockchain.
  - This process repeats for each milestone until all installments are released or the campaign is canceled.
- **No-Confidence Vote**
  - Contributors can initiate a no-confidence vote before each installment release.
  - Each contributor's vote weight is proportional to their contribution.
  - If the majority votes no-confidence, the smart contract will trigger the refund process.

- **Refund Conditions**
  - If the crowdfunding goal is not met by the deadline, the smart contract automatically refunds all contributors based on their contribution amount.
  - If a no-confidence vote passes, the smart contract calculates and distributes the remaining funds proportionally to each contributor.
- **Smart Contract Structure**

Constructor	Initializes the contract with campaign details, (crowdfunding goal, deadline, public key address of the project manager's account on the blockchain, installment milestones).
Donation Function	Allows users to send funds to the smart contract and records their contributions.
Milestone Verification Function	Verifies if a milestone has been achieved.
Installment Release Function	Releases a portion of the funds to the project manager upon milestone verification.
No-Confidence Vote Function	Allows contributors to vote on whether the campaign should continue or be canceled. The vote weight is proportional to their contribution.
Refund Function	Automatically refunds the remaining funds to contributors if the campaign is canceled or the crowdfunding goal is not met

2.2.2.3 Smart Contract State Machine



2.2.2.4 Smart Contract Public Functions

- create-campaign()

```
;; Creates and activates a new campaign. It is essentially the
;; initialization function which is the first one to call.
;;
;; @param uint target-goal The crowdfunding goal in micro-STX
;; @param uint duration The number of blocks till the deadline to
;;                      reach the crowdfunding goal
;; @param uint num-milestones The total number of the campaign
;;                            milestones
;;
;; @return an (ok true) response if this is the first time this
;;         function is called and the target goal, the duration and
;;         number of milestones are positve (i.e., non-zero) numbers
;;         or an error otherwise
define-public (create-campaign (target-goal    unit)
                               (duration       unit)
                               (num-milestones unit))
```

- donate()

```
;; Donates the specified amount in micro-STX to the campaign
;;
;; @return an (ok `amount`) response where amount is passed argument
;;         value if the function call succeeds or an error otherwise
define-public (donate (amount unit))
```

- vote()

```
;; Cast the user's vote for the specified milestone.
;; To vote the project manager must have posted that milestone and
;; voting should be enabled (i.e., the `can-vote` field is `true` and
;; the campaign status is STATUS_VOTE).
;; The user must have donated to the campaign and is not allowed to
;; vote more than once for the same milestone.
;; - If the user's vote results in a majority vote of confidence, the
;;   vote closes immediately and the campaign status is set to
;;   STATUS_FUNDED (i.e., the campaign continues). Only then can the
;;   owner claim the milestone funds.
;; - If the user's vote results in a majority vote of no confidence,
;;   the vote closes immediately and the campaign status is set to
;;   STATUS_CANCELED (i.e., the campaign is canceled). Only then can
;;   the user claim the proportional leftover funds for the funded
;;   campaign.
;;
;; @param uint index The index of the milestone to vote for
;; @param bool value `true` means vote of confidence while `false`
;;        means vote of no confidence
;; @return an (ok true) response if successful or an Err response
;;         otherwise
define-public (vote (index unit)
                    (value bool))
```

- post-milestone()

```
;; Called by the owner (i.e., project manager) to post a milestone.
;; The owner provides a short description about the accomplishment (or
;; a web URL to a page with the milestone details) which opens the
;; voting window for the
;; users to cast their votes.
;;
;; @param string-ascii details Information about the milestone
;;        accomplishment (can be a web URL) to help the users decide
;;        what to vote
;; @param uint index The milestone index
;;
;; @return an (ok true) response if successful or an Err response
;;         otherwise
define-public (post-milestone (details (string-ascii 100))
                              (index   (unit))
```

- claim-milestone-funds()

```
;; The owner claims the funds for the specified milestone provided that
;; the following requirements are met:
;;    1) The campaign status is STATUS_FUNDED which means the users
;;       are not still voting
;;    2) The users approved the campaign for the specified milestone
;; The installment amount is equal to the overall donation amount
;; divided by the number of milestones and is taken proportionally
;; from the donors.
;;
;; @param uint index The index of the milestone to claim the funds for
;;
;; @return an (ok `amount`) response where if the function call
;;         succeeds or an error in case of violation one of the
;;         conditions to claim the milestone or in case of an
;;         unexpected error during the STX refund transaction
define-public (donate (amount unit))
```

- claim-refund()

```
;; Refunds the `tx-sender` in any of the following two cases:
;; Case 1: The campaign did not reach the crowdfunding goal and the
;;         deadline expired. In this case, the `tx-sender` receives
;;         back the full donated amount.
;; Case 2: The campaign was funded but got cancelled later bacause a
;;         vote of no confidence was passed. In thie case, the `tx-
;;         sender` receives the a proportional refund from the
;;         leftover funds.
;;
;; @return an (ok `refunded amount`) response where if the function
;;         call succeeds or an error (e.g., if the campaign status
;;         does not fall under any of the two cases above or the
;;        `tx-sender` has not donated or in case of an unexpected
;;         error during the STX refund transaction
define-public (claim-refund)
```

- check-deadline()

```
;; Checks the campaign status and if it is other than STATUS-ACTIVE is
;; does nothing. Otherwise, it checks if the campaign deadline has
;; passed and if so it sets the status to STATUS-EXPIRED.
;;
;; @return an (ok `status`) response where `status` is the campaign
;;         status
define-public (check-deadline)
```

- check-vote-deadline()

```
;; Checks the campaign status and if it is other than STATUS-VOTE is
;; does nothing. Otherwise, it checks if the vote deadline has passed
;; and if so it sets the status to STATUS-FUNDED.
;;
;; @return an (ok `status`) response where `status` is the campaign
status
define-public (check-vote-deadline)
```