CrowdCoin

Ethereum-Based Decentralized Crowd Funding

Technical Documentation

Written By: *Theresa Whynot*

Contents

Technical Specifications

Introduction to CrowdCoin

CrowdCoin allows users to create and fund custom crowdfunding campaigns using an Ether wallet (e.g. MetaMask). All CrowdCoin transactions are housed on the Ethereum blockchain, governed by the application's smart contract code – making this application completely decentralized.

Campaigns can vary from tech products, charity initiatives, artistic ideas, etc. – the same way popular forums such as Kickstarter allow users to create their own custom crowdfunding projects. The campaign owner communicates a clear description of their project, funding needs and a deadline for that funding. Users can browse projects and contribute Ether to campaigns they believe in.

Why Blockchain?

Crowdfunding holds particular significance within the realm of blockchain technology due to the transparency, immutability, and security blockchain encompasses. On traditional 3rd party crowdfunding applications, such as Kickstarter, contributors and campaign owners are subject to any and all rules that are imposed by the 3rd party platform. This can include but not limited to: additional and/or unexpected fees required from users to the 3rd party, and/or changing rules, terms and conditions, and fee structures that interfere with existing user processes and experience. Not only this, but users are subject to any security risks and vulnerabilities the 3rd party may face, given that all data is stored in the 3rd party's centralized databases. This can include but not limited to: potential breaches of sensitive information, theft of users' funds, and/or missing or manipulated transaction/activity logs. CrowdCoin, powered by blockchain technology, presents a secure and trustworthy solution to the issues of traditional 3rd party crowdfunding platforms:

- *Transparency:* all CrowdCoin transactions (creating a campaign, contributing to a campaign, approving funding requests, etc.) are housed and can be identified on the Ethereum blockchain. This includes block number, timestamp, recipient, sender, Ether amount, etc.
    - Additionally, CrowdCoin boasts a blockchain-inspired solution, where Contributors must approve various funding requests by more than a 50% approval rating for the campaign to actually spend the funds. Funding requests are transparent in the nature by which the description, exact Ether amount, and recipient of the funds are communicated to all users. The approval process is inherently decentralized by giving power back to the Contributors as well.
- *Immutability:* all CrowdCoin transactions on the Ethereum blockchain cannot be changed or tampered with.
- *Security:* all CrowdCoin transactions are stored on the highly secure Ethereum blockchain, which is fortified by cryptography and a decentralized node network of transaction validators.

CrowdCoin Terms and Definitions

Before delving into CrowdCoin's main features and functionality, it is important to understand the below terms and definitions of the application:

Term	Definition
Manager	The Ether wallet address associated with the user who creates a campaign. The Manager has special privileges to create and finalize funding requests.
Contributor	The Ether wallet address associated with user who contributes the minimum contribution to a campaign.
Campaign Title	The title of the campaign, specified by the Manager. For example: "Dog Behavior Documentary"
Campaign Description	The description of the campaign, specified by the Manager. For example: "The documentary explores the intriguing and amusing behaviors of dogs, shedding light on their evolution, psychology, and interactions with humans. Dive into the world of our beloved companions and discover the secrets behind man's best friend."
Minimum Contribution	The minimum contribution for the campaign. Contributors must contribute this amount to a) actually contribute to the campaign, and b) be eligible to be a funding request Approver.
Minimum Balance	A campaign's minimum funding goal: the minimum balance that a campaign must meet in order to succeed.
Campaign Deadline	The deadline a campaign must meet its minimum balance, or funding goal, by.
Campaign Balance (remaining)	The remaining campaign balance after funding has been spent on requests. Campaign Balance (remaining) = Total Sum Contributions – Request Funds Spent
Total Sum Contributions	The total contributions to a campaign, regardless of what has been spent on funding requests.
Open Campaign	A campaign that has either not exceeded its deadline yet OR has met the minimum balance funding goal. In this case, the campaign allows further requests and contributions.
Closed Campaign	A campaign that has passed its deadline AND has not met the minimum balance funding goal. In this case, the Manager is given privileges to send the Campaign Balance (remaining) back to the Contributors.
Requests	Funding requests that can be created by the Manager for specific purposes. A description, amount, and recipient are defined. For example: "description: camera equipment for dog documentary amount: 100 wei recipient: 0x1aeFb08ee6ea3bcdf539e826707af6217b7763e6"
Request Recipient	The address that receives the amount from the funding request upon the Manager finalizing the request.
Approvers	All Contributors – who met the minimum campaign contribution – are eligible to approve a funding request and automatically are labeled as 'Approvers'.
Approval Count	The number of approvals a request has received. A funding request can only be finalized if the total number of approvals divided by the total number of Approvers (valid Contributors who met the minimum contribution) is greater than 50%.

Term (cont.)	Definition (cont.)
Approved Request	A request is considered completely approved once the total number of approvals divided by the total number of Approvers (valid Contributors who met the minimum contribution) is greater than 50%.
Finalized Request	Once a request is approved, the Manager can finalize the request. This sends the funding amount from the campaign balance (remaining) to the specified request recipient.
Ether (ETH)	The native cryptocurrency of the Ethereum blockchain, used for transactions and smart contract execution.
Wei	The smallest unit of Ether, representing the base denomination of value in the Ethereum network. 1 Wei = 0.000000000000000001 Ether (Wei is the unit used in CrowdCoin given limited developer test Ether available)

Main Features and Functionality

CrowdCoin boasts several key features:

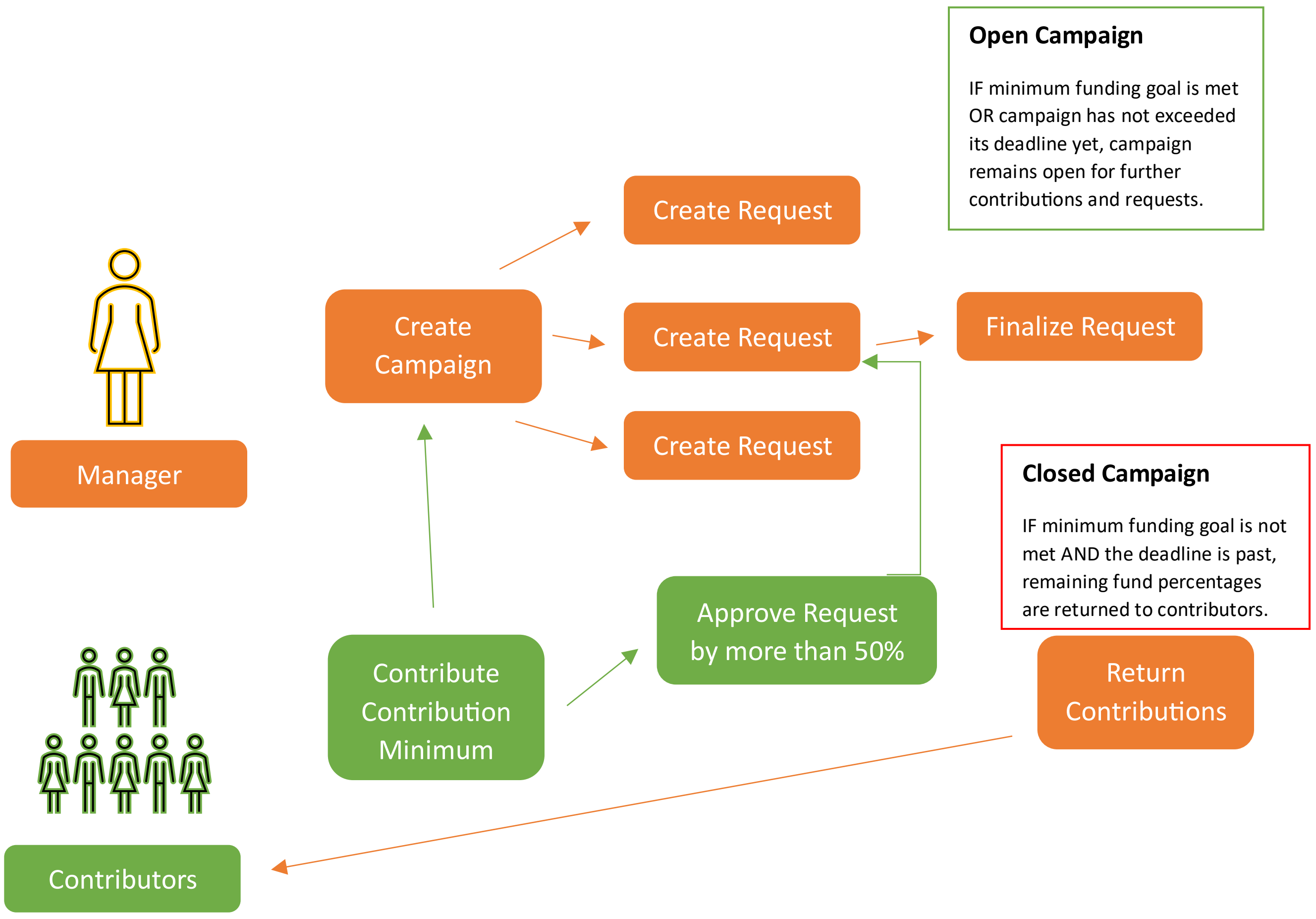
Feature/Functionality	Function Type	User Type	User Interface Details
Ability to create a campaign – specifying campaign title, description, minimum contribution from contributors, minimum balance for campaign to succeed, and deadline for campaign to succeed.	Add	Any user can create a campaign. Once a user does, the associated Ether wallet address is considered the Manager of that campaign.	'Create Campaign' button on the left navigation bar. 'Create Campaign' button on the bottom right home page.
Ability to view both open and closed campaigns per the navigation bar.	View	All users	'Open Campaigns' and 'Closed campaigns' buttons on the right navigation bar.
Ability to contribute to an open campaign.	Payable	All users that meet the minimum contribution – except the Manager of that campaign.	'Contribute' button on the right campaign home page.
Ability to create a funding request on an open campaign – specifying the description, amount and fund recipient. Taking from the blockchain-inspired philosophy, requests are a way manage transparent uses of campaign funds.	Add	Manager	'Add Request' button on the right of the campaign requests page.

Feature/Functionality (cont.)	Function Type (cont.)	User Type (cont.)	User Interface Details (cont.)
Ability to approve a funding request on an open campaign. Again taking from the blockchain-inspired philosophy, Contributors are given a decentralized format for voting on funding requests.	Add	Contributors/Approvers	'Approve' button in the table of the campaign requests page.
Ability to finalize a funding request on an open campaign if approval count % exceeds 50%. Funding is transferred to the specified recipient.	Payable	Manager	'Finalize' button in the table of the campaign requests page
Ability to pay Contributors back their unique % of the campaign balance (remaining) IF the campaign fails meet the minimum balance funding goal by the deadline (closed campaign). Contributors receive back whatever the % they originally contributed is from the remaining balance. So, funds returned once the campaign fails does not include funds already spent on requests.	Payable	Manager	'Return Contributions to Contributors' button in the bottom right of the closed campaign home page* *Campaign must be closed (failed to meet its goal by the deadline) in order for this button to appear.
Ability to view the FAQs	View	All users	'FAQs' button on the right navigation bar.
Ability to view summary of entire application: number of open campaigns, total campaigns funded, and total wei contributed	View	All users	Top banner on the CrowdCoin home page.

Feature/Functionality (cont.)	Function Type (cont.)	User Type (cont.)	User Interface Details (cont.)
Ability to view summary of campaign: campaign title, description, address of the manager, minimum contribution, number of requests, number of approvers, campaign balance (remaining), minimum balance, total contributions, and campaign deadline.	View	All users	All metrics on the campaign home page.
Ability to view open and closed requests: description, recipient, and approval count.	View	All users	All metrics of the campaign requests page.

High-Level Architecture

The below flow-chart offers a high-level summary at the main functions of CrowdCoin, from the perspective of the Manager and the Contributors/Approvers:


Open Campaign
IF minimum funding goal is met OR campaign has not exceeded its deadline yet, campaign remains open for further contributions and requests.
Create Request
Manager
Create Campaign
Create Request
Finalize Request
Create Request
Closed Campaign
IF minimum funding goal is not met AND the deadline is past, remaining fund percentages are returned to contributors.
Approve Request by more than 50%
Contribute Contribution Minimum
Return Contributions
Contributors

Technologies

The technologies used for the frontend of the CrowdCoin application include React for component construction, Next.JS for page routing, Node.JS for server set-up, Web3.JS for React component interaction with the Ethereum Blockchain and Semantic UI React for styling.

The technologies for the smart contract include Solidity for smart contract creation, JavaScript for compile and deployment scripts, and Web3.JS for contract interface scripts.

The technologies used for testing include Remix and Mocha.JS.

All the above technologies will be discussed in more detail in the following sections.

Smart Contract Details

Smart Contract Infrastructure

The embedded smart contract "Campaign.sol" file is the full source code for CrowdCoin's smart contract. The contract's solidity version is 0.4.17 (*Please note this is an outdated version, and a potential future update for the contract is to refactor the code and upgrade the solidity version. This is discussed further in the 'Limitations and Future Updates" section*).

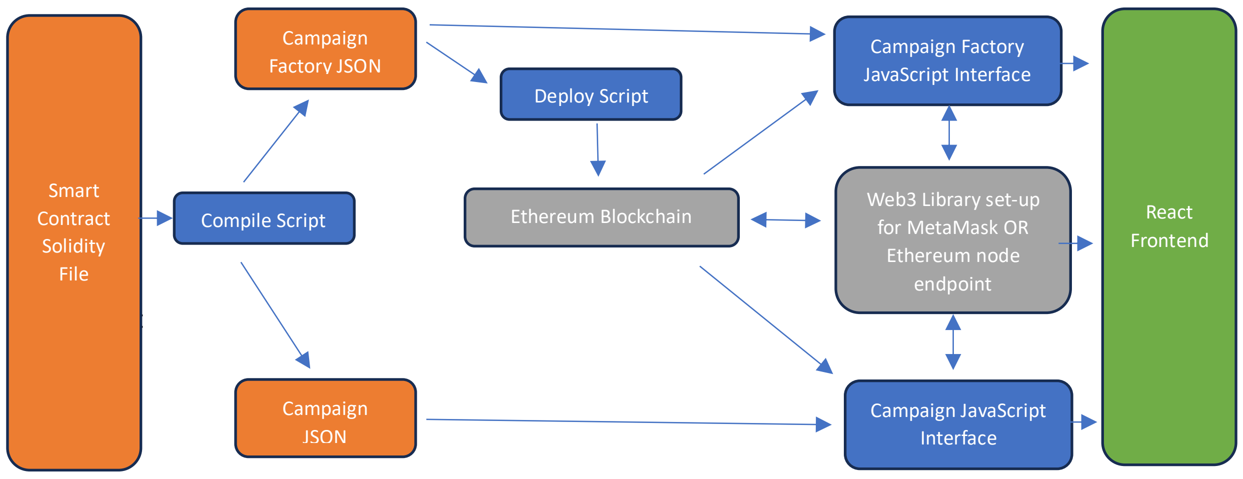
Campaign.sol

The contract contains 2 constructors for 2 unique contracts:

1. Campaign Factory: This allows the application to deploy multiple campaigns with fields such as name, description, minimum balance, deadline, etc. Without this initial contract, there could not be multiple campaigns created using the parameters specified in the Campaign contract.
2. Campaign: This allows all details of the individual campaign (title, description, minimum contribution, etc.) and transactions associated with the individual campaign (contribute to campaign create request, approve request, etc.) to be feasible.

After compilation, the smart contract transforms into 2 distinct JSON files: 1 for the Campaign Factory, and 1 for the Campaign. The Campaign Factory ABI and bytecode from the Campaign Factory JSON file are referenced in the deployment script and used to deploy the Campaign Factory to the Ethereum blockchain. When the Campaign Factory is deployed, it obtains a specific Ethereum contract address. This address is then integrated into a JavaScript interface file, which combines with the web3.js library (a library containing tools and utilities to connect front-end applications to the blockchain) to create an interface that can easily be applied to the React frontend. The Campaign contract has a similar JavaScript interface set-up for any Campaign Ethereum addresses the Campaign Factory contract deploys (the Campaign Factory deploys Campaigns to the Ethereum blockchain with the create campaign function, which will be discussed in further sections).

It is important to note that CrowdCoin's web3 set-up is designed to accommodate users with and without the MetaMask browser extension. MetaMask users can fully engage with the application's features. However, even non MetaMask users can interact with the app in a read-only capacity, as the web3 set-up connects to an Ethereum node endpoint ensuring that blockchain data is still accessible and retrievable for the frontend. The below diagram shows the high-level smart contract infrastructure:


Smart Contract Solidity File
Compile Script
Campaign Factory JSON
Campaign JSON
Deploy Script
Ethereum Blockchain
Campaign Factory JavaScript Interface
Web3 Library set-up for MetaMask OR Ethereum node endpoint
Campaign JavaScript Interface
React Frontend

The application's contracts contain several different types of variables, modifiers, and functions to ensure CrowdCoin intended architecture is working as expected. These elements are discussed in the below sections.

Smart Contract Variables and Modifiers

The smart contract code for both Campaign and Campaign Factory contains the below key variables:

Variable Name	Variable Description	Usage in Smart Contract
deployedCampaigns	Array of addresses for deployed campaigns	Used in getDeployedCampaigns function
manager	Address of the campaign manager, or creator of the campaign	Used as a state variable, modifier parameter, and condition in various functions and modifiers
minimumContribution	Minimum contribution required to participate in campaign	Used as a state variable and condition in various functions
campaignTitle	Title of the campaign	Used as a state variable and input parameter in the Campaign constructor
campaignDescription	Description of the campaign	Used as a state variable and input parameter in the Campaign constructor
minimumBalance	Minimum balance required for campaign success	Used as a state variable and input parameter in the Campaign constructor
targetDeadline	Deadline for reaching the campaign goal, or minimum balance	Used as a state variable and input parameter in the Campaign constructor
approvers	Mapping of participant addresses to approval status	Used as a state variable and condition in various functions and modifiers

Variable Name (cont.)	Variable Description (cont.)	Usage in Smart Contract (cont.)
approversAddressList	Array of addresses of campaign approvers (contributors who met the minimum contribution)	Used as a state variable, input parameter, and condition in various functions
contributions	Mapping of participant addresses to contribution amount	Used as a state variable, modifier parameter, and condition in various functions and modifiers
requests	Array of request structures for campaign expenditure	Used as a state variable and condition in various functions
approversCount	Number of participants who contributed minimum balance	Used as a state variable and condition in various functions and modifiers
sumContribution	Total amount contributed by all participants	Used as a state variable and modifier parameter in various functions and modifiers

The smart contract code for Campaign contains the below modifiers:

Modifier Name	Modifier Description	Usage in Smart Contract
activeCampaign	Modifier to check if the campaign is active/opn Current timestamp <= targetDeadline OR sumContribution >= minimumBalance	Used as a modifier to conditionally execute certain functions
inactiveCampaign	Modifier to check if the campaign is inactive/closed Current timestamp > targetDeadline AND sumContribution < minimumBalance	Used as a modifier to conditionally execute certain functions
restricted	Modifier to restrict access to manager-only functions	Used as a modifier to restrict access to specific functions

Smart Contract Functions

The smart contract code for both Campaign and Campaign Factory contain the below key functions:

Function Name	Function Description	Usage in Smart Contract
CampaignFactory (Constructor)	Constructor function to initialize the CampaignFactory contract	Initializes the state variables of the CampaignFactory contract
createCampaign	Manager creates a new campaign with specified parameters	Deploys a new instance of the Campaign contract using input parameters
getDeployedCampaigns	Returns an array of addresses for all deployed campaigns	Retrieves the list of addresses of all deployed campaigns
Campaign (Constructor)	Constructor function to initialize the Campaign contract with specified parameters	Initializes the state variables of the Campaign contract with input parameters

Function Name (cont.)	Function Description (cont.)	Usage in Smart Contract (cont.)
contribute	Allows a user to contribute funds to the campaign	Records the contribution of the user, updates sumContribution and contributions mapping
createRequest	Manager creates a spending request for campaign funds with specified parameters	Adds a new request to the requests array
approveRequest	Allows contributors to approve a spending request	Updates the approval status for a request, increments approvalCount
finalizeRequest	Finalizes a spending request after receiving sufficient approvals	Transfers the requested amount to the recipient and marks the request as complete
contributionReturn	Returns contributions to participants in an inactive campaign	Returns contributions proportionally to contributors based on their original contribution percentage
getSummary	Returns summary data about the campaign	Retrieves the summary data including minimum contribution, campaign balance, number of requests, etc.
getRequestsCount	Returns the count of spending requests in the campaign	Retrieves the number of spending requests in the campaign

Smart Contract Security

CrowdCoin’s smart contract code employs several best practices to mitigate potential security vulnerabilities. The code employs role-based access mechanisms through the use of modifiers like ‘restricted’, ensuring that only authorized users, such as the Manager, can execute certain functions.

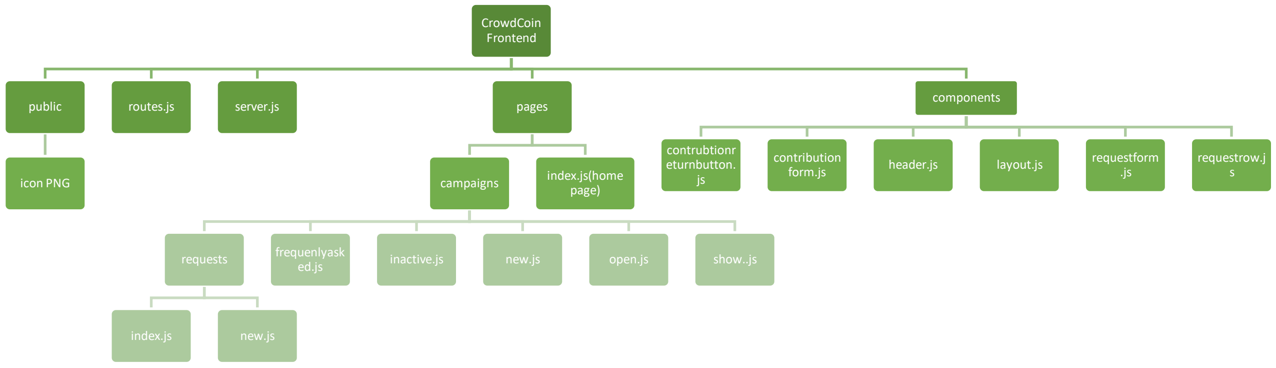
Additionally, the use of ‘require’ statements before executing critical actions, such as creating a request or finalizing a request, enforces conditions that must be met before proceeding, adding an extra layer of protection.

Furthermore, the separation of concerns between the CampaignFactory and Campaign contracts enhances code organization and readability, promoting maintainability and reducing the potential for errors.

Frontend Details

Frontend Infrastructure

CrowdCoin's frontend is built upon the dynamic synergy of React and Next.js. The component based architecture of React combined with the pages routing system of Next.JS makes for a seamless frontend infrastructure. Below is the file directory structure particular to the frontend, minus the general set-up React ‘node modules’ and and Next.JS ‘.next’ folders.



The public folder contains the CrowdCoin icon image:

The routes file contains all routes relative to the ‘pages’ directory. The routes file also incorporates the Ethereum blockchain account addresses in the URLs (see highlighted):

```
const routes = require('next-routes')();

routes
    .add('/campaigns/new', '/campaigns/new')
    .add('/campaigns/open', '/campaigns/open')
    .add('/campaigns/frequentlyasked', '/campaigns/frequentlyasked')
    .add('/campaigns/inactive', '/campaigns/inactive')
    .add('/campaigns/:address', '/campaigns/show')
    .add('/campaigns/:address/requests', '/campaigns/requests/index')
    .add('/campaigns/:address/requests/new', '/campaigns/requests/new');

module.exports = routes;
```

The server file code sets up an HTTP server using Node.js and the Next.js framework to handle incoming requests based on CrowdCoin’s above defined routes. The application is prepared to listen on port 8080 for incoming HTTP requests, and displays error messages given any issues:

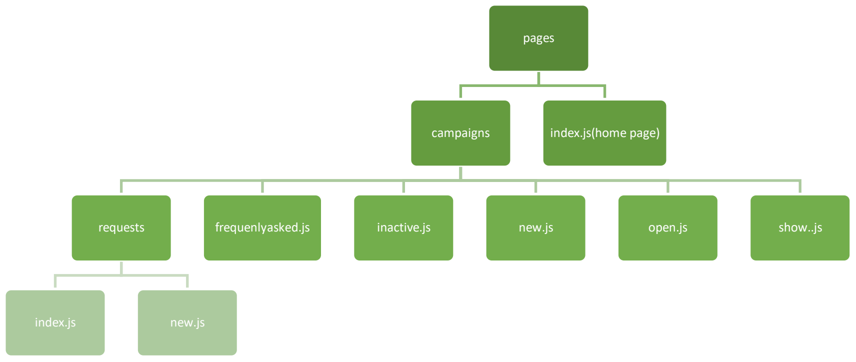
```
const { createServer } = require('http');
const next = require('next');

const app = next({
    dev: process.env.NODE_ENV !== 'production'
});

const routes = require('./routes');
const handler = routes.getRequestHandler(app);

app.prepare().then(() => {
    createServer(handler).listen(8080, err => {
        if (err) throw err;
        console.log('Ready on localhost:8080');
    });
});
```

The ‘pages’ directory breaks down the various individual pages of the application utilizing the Next.JS pages directory and routing architecture. The home page is configured at the top of the directory, with the subfolder ‘campaigns’ housing specific campaign pages such as individual campaign metrics (show.js), list of all open campaigns (open.js), and creation of new campaigns (new.js). The subfolder ‘requests’ in the ‘campaigns’ folder further breaks down the funding request information of individual campaigns such as request metrics (index.js) and creation of new requests (new.js):



The ‘components’ directory aids the pages directory with specific React and web3 functionality. For example, the header and layout components encompass the application’s pages container, navigation bar, and styling utilizing the Semantic UI React library and CSS in-line styling. These components are applied to all pages. Below are example code snippets from the home page, which includes the layout import. The layout also contains the header import, which includes the navigation bar. Together, the header and layout create styling and the container for each application page:

```
import Layout from '../components/Layout';
…(home page code)………

return (
            <Layout>
                <div>

             …(home page code)………

                 </div>
            </Layout>
```

Frontend and Smart Contract Interaction

The pages and components directories utilize the Campaign and Campaign Factory JavaScript interface files (discussed in The Smart Contract Interface section) to communicate with the contracts deployed on the Ethereum blockchain.

For example, the home page top banner displays the total number of open campaigns, the total number of funded campaigns, and the total amount of Wei contributed across the entire application. The banner can fetch these details by calling on the getSummary function of the Campaign contract, from the Campaign JavaScript interface file. The React code then uses individual campaign data points housed on the blockchain such as target deadline, sum contribution, and minimum balance to form conditions and return the proper data on the banner. Below are code snippets from the home page(index.js) that describe how the number of funded campaigns are pulled:

Import Interface: Imports the Campaign JavaScript interface file which contains web3.js library set-up, and the ABI of the Campaign contract housed on the Ethereum blockchain.

```
import Campaign from '../ethereum/campaign';
```

Fetch Data: Calls on the Campaign smart contract getSummary function and return the needed data points (housed in the getSummary array) from the Ethereum blockchain.

```
async fetchCampaignDetails(address) {
       const campaign = Campaign(address);
        const summary = await campaign.methods.getSummary().call();
        const sumContribution = summary[5];
        const minimumBalance = summary[6];
        const targetDeadline = Number(summary[7]);
        return {,minimumBalance,sumContribution, targetDeadline };
    }
```

Define Variable Condition: Defines the condition that concludes a Campaign is fully funded using the getSummary data points. Filters on only those campaigns in the campaignDetails array that meet the condition: sumContribution >= minimumBalance.

```
const fundedCampaigns = this.state.campaignDetails.filter(({ sumContribution,
minimumBalance }) => {
            return (sumContribution >= minimumBalance);
        });
```

Render Data on Browser: Pulls the length of the fundedCampaigns array, which displays the number of Campaigns that meet the condition for being fully funded to the frontend home page (see highlighted).

```
<Statistic >
<Statistic.Value>{fundedCampaigns.length}</Statistic.Value>
Statistic.Label>Funded Campaigns</Statistic.Label>
</Statistic>
```

In addition, the front-end design mimics certain conditions defined in the smart contract. For example, the smart contract code dictates that a campaign is *active* when the deadline is not passed OR the sum contribution is greater than or equal to the minimum balance; and is *inactive* once the deadline is passed AND the sum contribution is less than the minimum balance. There are several functions the are restricted to active campaigns only, such as contributing to a campaign (see highlighted):

Smart Contract Modifiers for Active and Inactive Campaigns:

```
modifier activeCampaign() {
      require (now <= targetDeadline || sumContribution >= minimumBalance);
        _;
    }

modifier inactiveCampaign() {
        require(now > targetDeadline && sumContribution < minimumBalance);
        _;
    }
```

Smart Contract Contribute Function – Restricted to Active Campaigns Only:

```
function contribute() public payable activeCampaign {
        require (msg.value >= minimumContribution);

        sumContribution = sumContribution + msg.value;

        contributions[msg.sender] += msg.value;

       if (!approvers[msg.sender]) {
        approvers[msg.sender] = true;
        approversCount++;
        approversAddressList.push(msg.sender);
        }
    }
```

These modifiers are not only applied in several of the smart contract's functions – per the above example that a user can only contribute to an active/open campaign – but also applied to the frontend design (*Please note a potential future update for CrowdCoin is to handle scenarios like this exhaustively on the frontend. There are still scenarios in which the smart contract code defines conditions that are not adequately represented on the frontend. This is discussed further in the 'Limitations and Future Updates" section*). For example, once a campaign is considered inactive/closed, the contribution form - which allows users to contribute to a campaign - disappears on the individual campaign metrics page (show.js):

Define Variable Condition: Defines the condition that a campaign is not fully funded, which matches the logic on the smart contract. Data points (target deadline, sum contribution) are fetched directly from the campaign on the Ethereum blockchain.

```
const campaignNotFunded = currentTimestamp > targetDeadline && sumContribution < 
minimumBalance;
```

Render Condition on Browser: When the campaign meets the 'campaign not funded condition', a warning message appears and the contribution form disappears, disallowing users from viewing or *attempting* to use the form (the form submission would error out per the smart contract code regardless). When the campaign does not meet the condition, the contribution form remains.

```
{campaignNotFunded ? (
    <Message warning>
        <Message.Header>
         The campaign has not met its funding goals.
        </Message.Header>
        <p>
         The manager of the campaign can now return the remaining balance back to
         the contributors.
        </p>
    </Message>
    ) : (
        <>
        <ContributeForm address={this.props.address} />
        </>
    );
}
```

Frontend External Libraries and UI Frameworks

The below external libraries and UI frameworks were used to aide CrowdCoin's architecture:

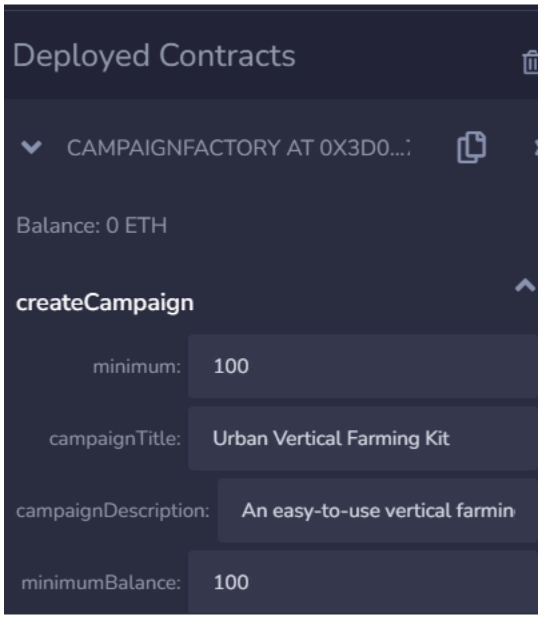
Library/Framework	Description	Usage in CrowdCoin
Web3.JS	A JavaScript library for interacting with the Ethereum Blockchain	Used to connect the front end to the Ethereum blockchain, enabling interaction with smart contracts
Next.JS	A React framework for server-rendered applications	Utilized for page routing and rendering
Node.JS	A JavaScript runtime for server-side applications	Employed to set up the server for the application, handling requests and serving Next.JS pages architecture
Semantic UI React	A UI framework that provides pre-styled components	Utilized to style the frontend components, ensuring a consistent and visually appealing UI.

Testing and Quality Assurance

Remix Testing

Before deploying smart contracts to the live Ethereum blockchain test network: Infura, the Remix testing tool was used to debug, compile, and deploy CrowdCoin's smart contract code within the web browser. Any changes to the smart contract code were re-run through Remix for debugging and deployment testing.

Remix also aided the use of MetaMask, transaction execution, and generation of key data points. The main functions of CrowdCoin's smart contract code were thoroughly tested using test MetaMask addresses, and test Wei amounts - using Sepolia test Ether. Below is an example of a deployed CampaignFactory contract, and the 'create campaign' function using Remix:



Balance: 0 ETH

minimum: 100

campaignTitle: Urban Vertical Farming Kit

campaignDescription: An easy-to-use vertical farmin

minimumBalance: 100

Mocha Testing

Mocha.JS testing framework was utilized to automate the testing of key functions of CrowdCoin's smart contract code.

The code begins by importing necessary libraries such as 'assert', 'ganache'(test network), and 'web3' to facilitate interactions with the Ethereum network:

```
const assert = require('assert');
const ganache = require('ganache');
const { Web3 } = require('web3');
const web3 = new Web3(ganache.provider());
```

It also pulls the compiled smart contracts and deploys both the CampaignFactory and Campaign contracts onto the local Ganache test network:

```
const compiledFactory = require('../ethereum/build/CampaignFactory.json');
const compiledCampaign = require('../ethereum/build/Campaign.json');


beforeEach(async () => {
    accounts = await web3.eth.getAccounts();


    factory = await new web3.eth.Contract(JSON.parse(compiledFactory.interface))
        .deploy({ data: compiledFactory.bytecode })
        .send({ from: accounts[0], gas: '1000000' });

    await factory.methods.createCampaign('100').send({
        from: accounts[0],
        gas: '1000000'
    });

    [campaignAddress] = await factory.methods.getDeployedCampaigns().call()
    campaign = await new web3.eth.Contract(
        JSON.parse(compiledCampaign.interface),
        campaignAddress
    );
});
```

The test suite includes various test cases to verify different aspects of the smart contract, such as deployment, manager assignment, contribution approval, payment request creation, and request processing. Each test case uses assertions to compare expected outcomes with actual outcomes, ensuring that the smart contract functions as intended. See example below for testing of deployment – the test ensures that both the CampaignFactory and Campaign Contracts can be successfully deployed to the Ethereum blockchain by ensuring that contract addresses are generated:

```
describe('Campaigns', () => {


    it('deploys a factory and a campaign', () => {
        assert.ok(factory.options.address);
        assert.ok(campaign.options.address);
    });
```

This suite of tests helps ensure that the smart contract's core functionalities are secure and functioning correctly on the blockchain. (*Please note the Mocha.JS test script is not exhaustive and should be further updated. This is discussed further in the 'Limitations and Future Updates" section*)

UI Testing

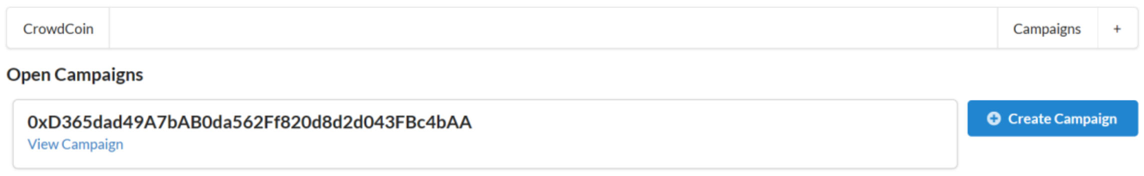
UI testing was also executed throughout the development and after the development phase. Both smart contract and frontend functionality was thoroughly tested on each component of the UI, in multiple different browsers (Chrome, Microsoft Edge, Internet Explorer). Browsers with and without the MetaMask extension were tested, to ensure that non MetaMask users could still access a read-only application with data dynamically updating from the Ethereum blockchain. Role-based security was tested (e.g. Manager and non-Manager roles) to ensure various functionality was only permitted for the intended roles (*Please note that while scaling, styling, responsiveness, and accessibility testing were considered in the creating of the application, much work is left to be done. This is discussed further in the 'Limitations and Future Updates" section*).

Beyond the Bootcamp: Limitations and Future Updates

Post Bootcamp: CrowdCoinV2

CrowdCoinV1 was created with the assistance of an online Udemy bootcamp: Ethereum and Solidity: The Complete Developer's Guide taught by Stephen Grider. After gaining a baseline understanding of frontend languages like React, JavaScript, HTML, and CSS, this bootcamp was a great segway to building an initial dApp like CrowdCoin. The bootcamp was highly informative on blockchain fundamentals, the Ethereum/Web3 world, and Solidity programming. CrowdCoinV1 provides a good initial prototype for a crowdfunding dApp and some key functions, but does lack complexity and user-friendly design. See screenshots below:

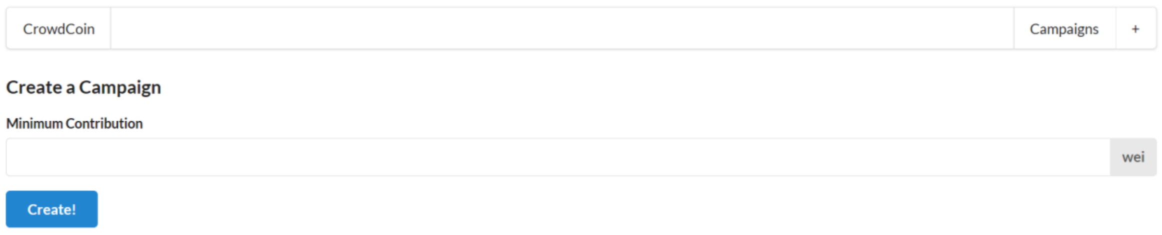
Home Page:



Open Campaigns

0xD365dad49A7bAB0da562Ff820d8d2d043FBc4bAA

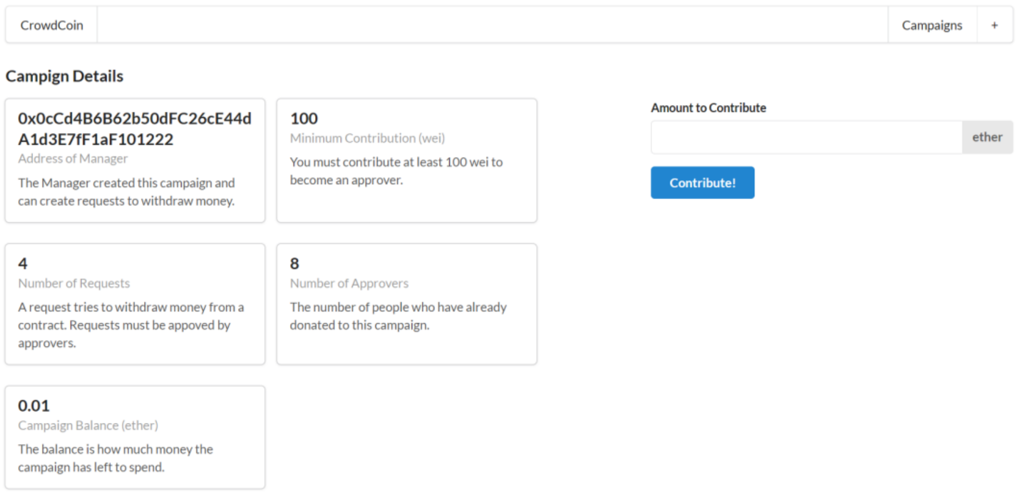
Create Campaign Page:



Create a Campaign

Minimum Contribution

Campaign Display Page:



Campign Details

0x0cCd4B6B62b50dFC26cE44dA1d3E7fF1aF101222
Address of Manager
The Manager created this campaign and can create requests to withdraw money.

100
Minimum Contribution (wei)
You must contribute at least 100 wei to become an approver.

4
Number of Requests
A request tries to withdraw money from a contract. Requests must be appoved by approvers.

8
Number of Approvers
The number of people who have already donated to this campaign.

0.01
Campaign Balance (ether)
The balance is how much money the campaign has left to spend.

Amount to Contribute

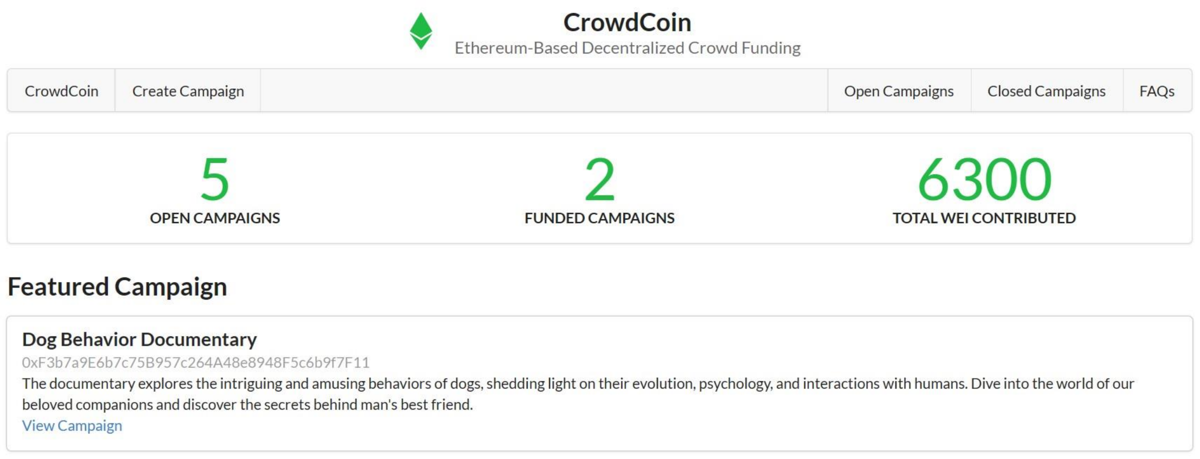
CrowdCoinV2 – the exact application which is discussed in this technical documentation – addresses many gaps and issues from CrowdCoinV1. The below table contains all updates made from V1 to V2:

Update	High-Level Technical Changes
Expand create campaign function with additional fields	Updated smart contract code to accommodate campaign-specific details beyond the minimum contribution. New variables such as campaign title, description, minimum balance, and deadline were incorporated. For instance, the function createCampaign(uint minimum, string campaignTitle, string campaignDescription, uint minimumBalance, uint targetDeadline) was updated.
Fix approvers count	Revised the smart contract code to prevent multiple contributions from the same address from incrementing the approvers count. Logic was adjusted to ensure that the count only increases for unique contributors.
Manager and recipient approval restrictions	Enhanced the smart contract's logic for approving funding requests to prevent the manager and recipient of a request from approving their own proposals. This was achieved by adding require conditions to the approval process.
Add active and inactive modifiers	The smart contract was augmented with modifiers to restrict certain functions based on whether a campaign is active or inactive. For example, the activeCampaign modifier checks if the current time is within the campaign's deadline or if the minimum balance has been reached.
Proportional contribution return	Introduced a function in the smart contract that enables the manager to return contributions proportionally to contributors based on their original contribution percentages. The code includes calculations using mapping variables to ensure fairness.
Progress bar for campaign metrics	Integrated a percentage progress bar into the individual campaign metrics page to visually represent the total contributions relative to the minimum balance required for success. Custom messages were added to communicate progress status to users.

Update (cont.)	High-Level Technical Changes (cont.)
Validation checks for create campaign function	Modified the smart contract's createCampaign function to include conditions ensuring that the minimum balance is greater than or equal to the minimum contribution and that the campaign deadline is in the future. These checks were added using require statements.
Notes column for funding requests	The frontend was updated to include a notes column in the individual campaign's funding requests index. The notes provide users with the current status of each request, including whether it's complete, ready for finalization, or requires more approvals.
Branding and styling enhancements	Applied style and branding adjustments to the frontend, including the addition of a header banner displaying key metrics for the entire application. The banner fetches and dynamically renders data from the Ethereum blockchain to keep users informed.
Conversion to wei units	Altered the frontend to display all ether units as wei units due to limited test funds, ensuring accurate representation of contributions, funding requests, and metrics.
FAQs page addition	Introduced a new FAQs page to address user queries and enhance user experience by providing readily accessible information on key topics.

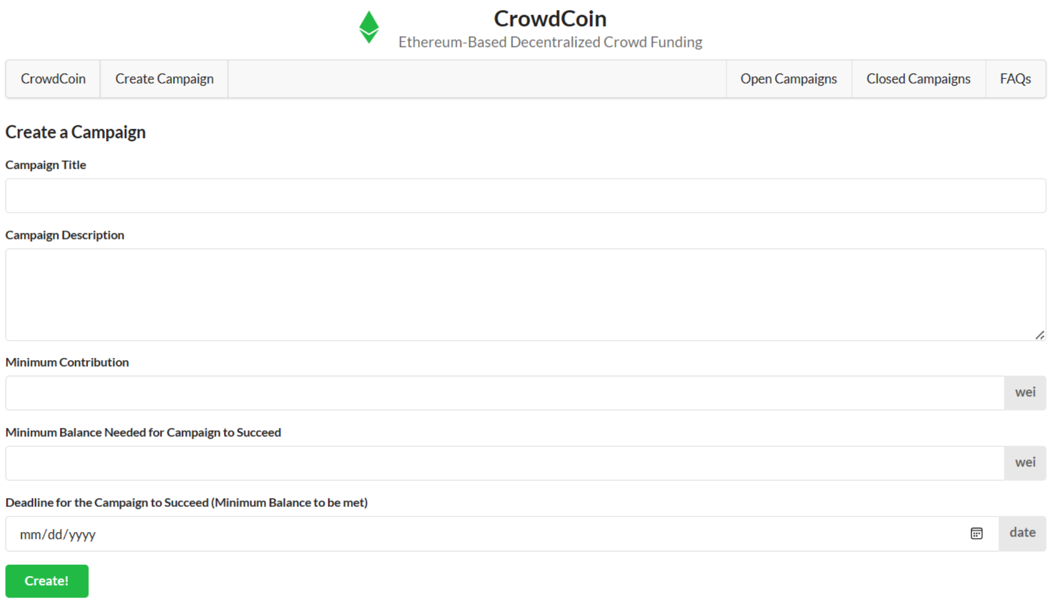
Below are screenshots of CrowdCoinV2 UI:

Home Page:



Dog Behavior Documentary
0xF3b7a9E6b7c75B957c264A48e8948F5c6b9f7F11
The documentary explores the intriguing and amusing behaviors of dogs, shedding light on their evolution, psychology, and interactions with humans. Dive into the world of our beloved companions and discover the secrets behind man's best friend.

Create Campaign Page:



CrowdCoin
Ethereum-Based Decentralized Crowd Funding

CrowdCoin | Create Campaign | Open Campaigns | Closed Campaigns | FAQs

Create a Campaign

Campaign Title

Campaign Description

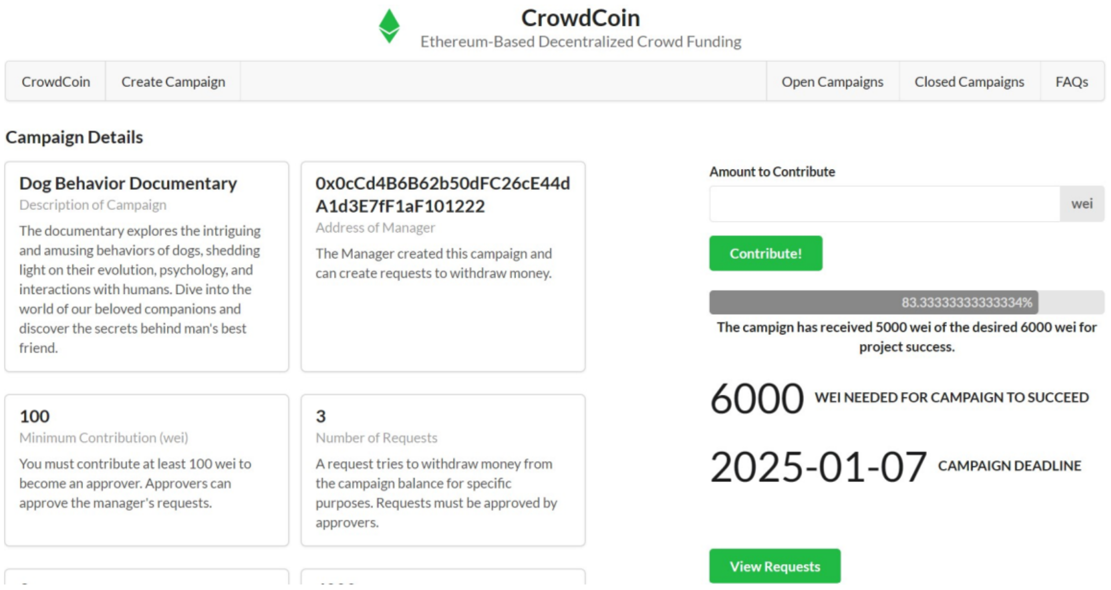
Minimum Contribution — wei

Minimum Balance Needed for Campaign to Succeed — wei

Deadline for the Campaign to Succeed (Minimum Balance to be met) — mm/dd/yyyy — date

Create!

Campaign Display Page:



CrowdCoin
Ethereum-Based Decentralized Crowd Funding

CrowdCoin | Create Campaign | Open Campaigns | Closed Campaigns | FAQs

Campaign Details

Dog Behavior Documentary
Description of Campaign
The documentary explores the intriguing and amusing behaviors of dogs, shedding light on their evolution, psychology, and interactions with humans. Dive into the world of our beloved companions and discover the secrets behind man's best friend.

0x0cCd4B6B62b50dFC26cE44dA1d3E7fF1aF101222
Address of Manager
The Manager created this campaign and can create requests to withdraw money.

100
Minimum Contribution (wei)
You must contribute at least 100 wei to become an approver. Approvers can approve the manager's requests.

3
Number of Requests
A request tries to withdraw money from the campaign balance for specific purposes. Requests must be approved by approvers.

Amount to Contribute — wei

Contribute!

83.33333333333334%

The campign has received 5000 wei of the desired 6000 wei for project success.

6000 WEI NEEDED FOR CAMPAIGN TO SUCCEED

2025-01-07 CAMPAIGN DEADLINE

View Requests

Limitations & Future Updates

Even with CrowdCoin2 boasting several exciting new features, the application – and its V2 rollout - still has limitations related to security, accessibility, frontend design, testing, etc. Please see the below table for CrowdCoin's current limitations and future updates to address these issues:

Limitation	Future Update – Recommended Action
Outdated Solidity version (0.4.17)	Update to the latest Solidity version and refactor the codebase to leverage the latest language features and optimizations.
Incomplete Test Cases with Mocha.JS	Enhance the Mocha.JS test suite to encompass all aspects of the application's functionality, considering alternative testing frameworks as well.
Frontend Inconsistencies with Smart Contract Logic	Align the frontend behavior closely with smart contract logic, ensuring consistent user experiences. Implement role-based security and separate login accounts to address this issue.
Scalability and Zoom-In Limitation	Optimize CSS and Semantic UI React components to ensure seamless scaling and accommodate browser zoom-in beyond 200%.
Accessibility & Responsiveness Gaps	Make the application responsive across various platforms, including mobile devices. Update the app to adhere to accessibility best practices and legal requirements.
Lack of Detailed Campaign Information	Enhance campaign profiles to include creator bios, photos, videos, and more media options, providing users with comprehensive campaign information.
Smart Contract Security Audit	Conduct a thorough security audit of the smart contract code and implement recommended security measures in alignment with the latest best practices.
Contributor Information and Incentives	Showcase a feed displaying contributors' unique addresses and contribution amounts. Introduce VIP tiers or digital awards to incentivize contributions to campaign funding.
Navigation Challenges with Numerous Campaigns	Introduce tag categories for individual campaigns, allowing managers to specify tags during campaign creation. Implement a search bar and category filtering for users to easily find campaigns based on tags and keywords.
Lacking User Account Management	Develop user profiles linked to Ethereum account addresses, enabling users to log in, track their supported campaigns, and manage their contributions effectively.

Code Repository

Git Repository

The full GitHub repository is located here.

Instructions for Running CrowdCoin Locally

Please see the below instructions for launching CrowdCoin locally.

Getting Started

Follow these steps to set up and run CrowdCoin on your local machine:

Prerequisites

Before you begin, ensure you have the following software installed on your computer:

- Node.js: Download and install Node.JS
- MetaMask: Download and install MetaMask (Note that CrowdCoin can still operate in read-only mode without MetaMask. The application links to an Ethereum blockchain node endpoint when MetaMask is not in use)

Installation

Download the project repository by either downloading the zip file or using Git to clone the repository (green "<>Code" button > Clone with HTTPS, SSH or GitHub CLI)

Navigate to the project directory in your terminal: **cd crowdcoin-main**

Install the project dependencies: **npm install**

Running the App

After installing the dependencies, start the app: **npm start**

Open your web browser and go to: http://localhost:8080/

Usage

Explore the CrowdCoin app and its features. Using Ether from your MetaMask wallet, you can create a new campaign, contribute to existing campaigns, and more!

Contributing

If you'd like to contribute to this project, feel free to fork the repository, make your changes, and submit a pull request.

Happy crowdfunding with CrowdCoin!