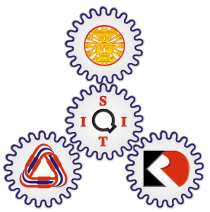

Crowdfunding Platforms

By

Group 9

Phutthachat Kanjanamek	6322770767
Natsareeya Kitikhungumjon	6322771708
Nuttakarn Chaiyabud	6322772383

Lecturer: Dr.Watthanasak Jeamwatthanachai
Blockchain Development
(DES484: Section 1)

Sirindhorn International Institute of Technology
Thammasat University

18 December 2023

Contents

Understand the Problem Domain

The use case for a Crowdfunding dApp revolves around providing a decentralized and transparent platform for fundraising initiatives. Project creators can propose their ideas or ventures, and individuals (backers) can contribute funds to support these projects. The smart contract governing the crowdfunding process automates fund collection, allocation, and distribution based on predefined rules.

Key aspects of the use case:

- **Project Creation**: Project creators use the dApp to create campaigns, outlining their goals, timelines, and funding targets. They can include details such as project descriptions, milestones, and potential rewards for backers.
- **Fundraising: Backers**, interested in supporting specific projects, use the dApp to contribute funds. Contributions are recorded on the blockchain, providing transparency and traceability.
- **Smart Contract Execution**: The smart contract facilitates the automated execution of the crowdfunding campaign. It ensures that funds are only released to the project creator when the funding goal is met, reducing the risk of misuse or misallocation of funds.
- **Tokenization**: Some Crowdfunding dApps may leverage blockchain tokens, representing backers' contributions. These tokens can have utility within the platform or be tradable on external exchanges.
- **Decentralized Governance**: In certain implementations, the dApp may incorporate decentralized governance features, allowing stakeholders to participate in decision-making processes related to the platform's development and policies.

Relevant Regulations, Standards, and Best Practices:

- **Smart Contract Security**: Implementing best practices for smart contract development is essential to prevent vulnerabilities and exploits. Auditing smart contracts for security flaws and following secure coding standards (e.g., ConsenSys' Best Practices) are common approaches.
- **Privacy and Data Protection**: Compliance with data protection laws, such as GDPR, is essential. Crowdfunding dApps must prioritize user privacy and provide transparent policies regarding the collection and use of personal information.
- **Scalability**: Ensuring that the dApp can scale to accommodate a growing user base and increasing transaction volumes is critical. Scalability solutions, such as layer 2 solutions or sharding, may be considered.
- **User Experience (UX) Design**: Best practices in UX design should be applied to create an intuitive and user-friendly interface. Clear instructions, seamless onboarding processes, and responsive design contribute to a positive user experience.
- **Community Engagement**: Building a community around the Crowdfunding dApp is essential. Engaging with users, collecting feedback, and fostering an active and supportive community contribute to the long-term success of the platform.

By addressing these considerations, a Crowdfunding dApp can provide a secure, transparent, and legally compliant environment for both project creators and backers, fostering innovation and collaboration within the decentralized ecosystem.

Identify the Key Requirements

In embarking on the creation of a crowdfunding decentralized application (dApp), it is crucial to define the key requirements for the smart contract and the dApp itself. These requirements, outlined in a Specific, Measurable, Achievable, Relevant, and Time-bound (SMART) manner, form the essential building blocks to ensure the success and effectiveness of the developed solution.

1. **Transparent Smart Contract Execution**
    - Specific: Create a smart contract with clear, auditable logic for crowdfunding processes.
    - Measurable: Integrate blockchain features for transparent fund tracking.
    - Achievable: Ensure the smart contract is easily comprehensible for third-party auditors.
    - Relevant: Enhance user trust by providing transparency in fund allocation and usage.
    - Time-bound: Complete smart contract development and auditing within the specified timeframe.

2. **Regulatory Compliance**
    - Specific: Integrate identity verification features to comply with regulations.
    - Measurable: Regularly update the dApp to align with changes in financial and privacy regulations.
    - Achievable: Collaborate with legal experts to ensure adherence to regional crowdfunding laws.
    - Relevant: Prevent legal complications by addressing compliance requirements proactively.
    - Time-bound: Implement necessary compliance updates before regulatory changes take effect.

3. **User Experience and Accessibility**
    - Specific: Design an intuitive and user-friendly interface for the crowdfunding dApp.
    - Measurable: Conduct user testing to evaluate platform accessibility and usability.
    - Achievable: Ensure compatibility with various devices.
    - Relevant: Cater to users with varying technical proficiency to encourage widespread adoption.
    - Time-bound: Incorporate user feedback and make necessary improvements before the official launch.

Define the Architecture

In designing the architecture for the Crowdfunding dApp, we leverage the Ethereum blockchain along with Next.js as the frontend framework, Hardhat as the development environment, and Solidity for smart contract development. This architecture ensures a decentralized, secure, and scalable platform that aligns with the specific requirements of the Crowdfunding dApp project.

Components of the System:

1. **Smart Contract**
    - Manages the crowdfunding logic, fund disbursement, and governance.
    - Developed in Solidity.
    - Interfaces with the decentralized wallets, handles user authentication, and communicates with the frontend to provide real-time campaign updates.
2. **Decentralized Wallet (MetaMask)**:
    - Serves as the user's decentralized wallet, enabling secure storage of cryptographic keys and facilitating transactions on the Ethereum blockchain.
    - Integrates MetaMask, a popular browser extension providing a user-friendly interface for managing Ethereum assets.
    - Users authenticate and authorize transactions through MetaMask, ensuring a secure and seamless experience for contributing funds to campaigns.
3. **Frontend (Next.js)**
    - Provides a user-friendly interface for project creators and backers to interact with the Crowdfunding dApp.
    - Developed using Next.js, a React framework for server-side rendering, ensuring efficient and responsive UI.
    - Communicates with the smart contract through Ethereum's Ethers.js library to facilitate user interactions, such as contributing to campaigns and viewing project details.

Interaction Between Components:

1. **Frontend and Smart Contract**
   The frontend communicates with the smart contract through Ethers.js, allowing users to interact with campaigns, contribute funds, and view real-time updates.
2. **Smart Contract and MetaMask**
   MetaMask is integrated for user authentication and transaction signing, enhancing security and providing a seamless experience for users interacting with the smart contract.

Relevant Technologies, Frameworks, and Standards:

1. **Ethereum**: Utilized as the underlying blockchain to enable decentralized and transparent transactions.
2. **Next.js**: Chosen for frontend development to provide server-side rendering, improving performance and user experience.
3. **Hardhat**: Used as the development environment for Ethereum smart contracts, offering testing, compilation, and deployment capabilities.
4. **Solidity**: The programming language for smart contract development, ensuring compatibility with the Ethereum Virtual Machine (EVM).
5. **Ether.js**: Facilitates communication between the frontend and Ethereum blockchain, enabling seamless interaction with smart contracts.
6. **MetaMask**: Integrated as the decentralized wallet for user authentication and transaction signing, enhancing the security and usability of the Crowdfunding dApp.

This architecture combines the strengths of Ethereum, Next.js, Hardhat, and Solidity to create a Crowdfunding dApp that is secure, transparent, and user-friendly. By utilizing decentralized technologies and adhering to industry standards, the platform aims to deliver a seamless and trustworthy experience for both project creators and backers within the crowdfunding ecosystem.

Develop the Smart Contract

In developing the smart contract for our crowdfunding decentralized application (dApp), we've employed Solidity, a programming language specifically designed for smart contracts on the Ethereum blockchain. The smart contract, named *CrowdFunding*, serves as the backbone of our dApp, encapsulating the necessary data structures, functions, and logic to facilitate crowdfunding campaigns in a decentralized manner.

The primary data structure is the *Campaign*, representing individual crowdfunding initiatives. Each campaign includes essential details such as the owner's address, campaign title, description, fundraising target, deadline, amount collected, and arrays to track contributors' addresses and donation amounts. These elements collectively define the parameters and status of each campaign.

The smart contract features a function named *createCampaign*, allowing users to initiate new crowdfunding campaigns by providing relevant details. A key requirement is that the campaign's deadline must be set in the future to ensure validity. Upon successful campaign creation, the contract increments the *numberOfCampaigns* variable, signifying the total number of active campaigns.

To support contributions, the *donateToCampaign* function enables users to contribute funds to a specific campaign. The contributed amount is associated with the donor's address and added to the campaign's donation arrays. If the contribution is successful (i.e., the funds are sent to the campaign owner), the total amount collected for the campaign is updated accordingly.

For transparency, the smart contract includes functions to retrieve information about campaigns and their contributors. The *getDonators* function allows users to access the addresses and donation amounts of contributors to a specific campaign. Additionally, the *getCampaigns* function provides an array of all campaigns, allowing users to explore ongoing and completed initiatives.

To enhance the flexibility of the dApp, implementing the *updateCampaign* function, enabling campaign owners to modify campaign details such as the description, target amount, and deadline. The *closeCampaign* function governs the closure of campaigns, considering both successful and unsuccessful outcomes. In the event of a successful campaign, the surplus funds are transferred back to the campaign owner, while unsuccessful campaigns close gracefully.

In summary, our smart contract establishes a robust framework for conducting decentralized crowdfunding campaigns, ensuring transparency, security, and adherence to predefined rules. The logic implemented aligns with the objectives of creating a functional and user-friendly crowdfunding dApp, and further testing and refinement will be conducted to enhance its capabilities and address potential edge cases.

Build the Decentralized App

Building a decentralized application (dApp) around a smart contract marks a transformative step in leveraging blockchain technology's potential. This intricate process amalgamates the robustness of a smart contract with an intuitive and user-friendly front-end interface. To embark on this journey, several key considerations and steps need to be meticulously planned and executed.

Set Up Development Environment:

1. **Install Node.js and NPM:**
    - **Node.js Installation:** Visit the Node.js official website and download the version that fits your operating system. Follow the installation instructions provided on the website to install Node.js.
    - **Verify Installation:** After installation, open your terminal or command prompt and type *node -v* and *npm -v* to verify if Node.js and NPM (Node Package Manager) are installed. These commands should display the installed versions of Node.js and NPM respectively.
2. **Install Hardhat:**
    - **Install Globally:** Once Node.js and NPM are successfully installed, open your terminal or command prompt and execute the command *npm install -g hardhat*. This command globally installs Hardhat, a development environment tailored for Ethereum smart contract development.
    - **Verify Installation:** To ensure Hardhat has been installed properly, type *hardhat* in your terminal. It should display information about the available commands and options provided by Hardhat.

Create a new Next.js project:

- Using Next.js CLI: Run `*npx create-next-app <project-name>*` in your terminal. This command initializes a new Next.js project based on the provided code snippet for setting up a Next.js project.

Configure Hardhat:

1. **Create and Configure Hardhat Project:**
    - **Initialize a New Hardhat Project:** Open your terminal or command prompt and execute *npx hardhat*. This command initializes a new Hardhat project. Follow the prompts to set up the basic structure for your Ethereum development environment.
    - **Update hardhat.config.js:** Once the project is initialized, navigate to the hardhat.config.js file in your project directory. This file holds the configuration settings for your Hardhat project.

- **Specify Ethereum Network Settings:** Within this file, you'll configure various settings related to your Ethereum network, such as specifying the network name, network URLs, API keys if necessary, and other network-specific details required to connect to an Ethereum network.

2. **Add Solidity Compiler Plugin:**
    - **Install Required Packages:** In your terminal or command prompt, execute the command *npm install @nomiclabs/hardhat-ethers ethers @nomiclabs/hardhat-waffle ethereum-waffle chai @nomiclabs/hardhat-etherscan*. This installs several necessary packages and plugins, including the Solidity compiler plugin for Hardhat.
    - **Update hardhat.config.js:** After installing the required packages, update your hardhat.config.js file to include the Solidity compiler plugin. Modify the configuration to include the newly installed plugins and adjust settings as needed for your specific project.

Write the Solidity Smart Contracts:

- Solidity Development: Use the provided contract code snippets to write Solidity smart contracts. Define the crowdfunding contract and other essential contracts required for the dApp.

Implement the Front End:

- Develop User Interface with Next.js and React: Use the provided React components (`Card`, `Hero`, `Footer`, etc.) to create the user interface for the crowdfunding marketplace. Utilize Next.js for routing and React for UI components.

Connect Front-End to Smart Contracts:

- Integrate ethers: Incorporate ethers to establish a connection between the front-end and Ethereum blockchain. Use the provided code snippets within the React components to interact with Ethereum smart contracts.
- Implement Interaction Functions: Utilize functions like `createNewCampaign`, `closeCampaign1`, `handleConnectWallet`, `handleDisconnectWallet`, `createDonation`, etc., to interact with Ethereum smart contracts. These functions handle actions like campaign creation, donation, closing campaigns, and wallet connectivity.

Test the Application:

- Smart Contract Testing: Use Hardhat for testing smart contracts. Write test cases to validate contract functionalities such as campaign creation, donation handling, etc.
- User Interface Testing: Test the user interface thoroughly to ensure proper functionality, including campaign creation, contribution, and management.

Deployment:

- Deploy to Test Network: Deploy the dApp to a test network like Rinkeby or Ropsten using Hardhat for testing in a live environment.
- Deployment to Mainnet: After successful testing and satisfaction with the dApp's performance, deploy it to the main Ethereum network for public use.

These specific steps align with the given code snippets, enabling developers to build a decentralized crowdfunding application using Next.js, Hardhat, and Solidity.

Test and Debug

In the process of developing the smart contract for our crowdfunding decentralized application (dApp), we've diligently followed best practices, including extensive testing and debugging procedures. Unit tests have been meticulously crafted to validate each critical aspect of the CrowdFunding smart contract. These tests cover fundamental functionalities such as campaign creation, donation processing, retrieval of campaign information, and the ability to update campaign details.

The initial test, "Should create a campaign," ensures that campaigns are successfully initiated with the specified parameters, validating the owner's address, campaign details, fundraising target, deadline, initial amount collected, and the open/closed status. Subsequently, the "Should donate to a campaign" test rigorously examines the donation process, confirming that donors are appropriately recorded, and their contributions accurately reflect the expected values.

Furthermore, the "Should retrieve a list of campaigns" test verifies that the function responsible for fetching a list of campaigns functions as intended, with a focus on the returned data structure and the existence of at least one campaign in the list. Additionally, the "Should update a campaign" test ensures that the contract allows for modifications to existing campaigns, validating that the updated campaign details align with the specified changes.

Throughout testing, we faced challenges prompting us to enhance the dApp. Peer reviews played a vital role in identifying areas for improvement, particularly in optimizing wallet connections for a better user experience. We recognize the importance of addressing these challenges and are actively working on solutions.

```
const { expect } = require("chai");  64.8k (gzipped: 16.6k)
const { ethers } = require("hardhat");

Complexity is 6 It's time to do something...
describe("CrowdFunding", function () {
  let crowdFunding;
  let owner;
  let donor;

  before(async () => {
    // Deploy the CrowdFunding contract before running the tests
    const CrowdFunding = await ethers.getContractFactory("CrowdFunding");
    crowdFunding = await CrowdFunding.deploy();
    await crowdFunding.deployed();

    // Get accounts from Hardhat
    [owner, donor] = await ethers.getSigners();
  });

  it("Should create a campaign", async function () {
    const title = "Campaign Title";
    const description = "Campaign Description";
    const target = ethers.utils.parseEther("1");
    const deadline = Math.floor(Date.now() / 1000) + 3600; // One hour from now

    const createTx = await crowdFunding.createCampaign(owner.address, title, description, target, deadline);
    await createTx.wait();

    const campaign = await crowdFunding.campaigns(0);
    expect(campaign.owner).to.equal(owner.address);
    expect(campaign.title).to.equal(title);
    expect(campaign.description).to.equal(description);
    expect(campaign.target).to.equal(target);
    expect(campaign.deadline).to.equal(deadline);
    expect(campaign.amountCollected).to.equal(ethers.utils.parseEther("0"));
    expect(campaign.isClosed).to.equal(false);
  });
```

Figure 1: Test and Debug Example Code I

```
  it("Should donate to a campaign", async function () {
    const donationAmount = ethers.utils.parseEther("0.5");

    const donateTx = await crowdFunding.connect(donor).donateToCampaign(0, { value: donationAmount });
    await donateTx.wait();

    const [donators, donations] = await crowdFunding.getDonators(0);

    // Check if the donor address is included
    expect(donators).to.include(donor.address);

    // Check if the donation amount is close to the expected value
    const expectedDonationAmount = ethers.utils.parseEther("0.5");
    const donationIndex = donators.indexOf(donor.address);
    expect(donations[donationIndex]).to.be.closeTo(expectedDonationAmount, ethers.utils.parseEther("0.0001"));
  });

  it("Should retrieve a list of campaigns", async function () {
    const campaignsList = await crowdFunding.getCampaigns();
    expect(campaignsList).to.be.an("array");
    expect(campaignsList).to.have.lengthOf.at.least(1);
  });

  it("Should update a campaign", async function () {
    const newDescription = "Updated Campaign Description";
    const newTarget = ethers.utils.parseEther("2");
    const newDeadline = Math.floor(Date.now() / 1000) + 7200; // Two hours from now

    await crowdFunding.connect(owner).updateCampaign(0, newDescription, newTarget, newDeadline);

    const updatedCampaign = await crowdFunding.campaigns(0);
    expect(updatedCampaign.description).to.equal(newDescription);
    expect(updatedCampaign.target).to.equal(newTarget);
    expect(updatedCampaign.deadline).to.equal(newDeadline);
  });

});
```

Figure 2: Test and Debug Example Code II

Deploy and Maintain

With the development and testing phase complete, the deployment of our crowdfunding dApp is the next critical step. Although the initial testing occurred on a local Ethereum network for simulation, the transition to a public blockchain network is imminent. Real Ethereum coins are required for deployment on live networks, and while the process is in progress, deployment on the live network is temporarily delayed due to the unavailability of sufficient Ethereum coins for gas fees.

The deployment process will involve ensuring that the smart contract is well-configured for the target network, including adjusting parameters like gas limits and confirming compatibility with the Ethereum Virtual Machine (EVM). Once deployed, ongoing maintenance becomes paramount. Continuous monitoring of the system will be essential to identify and address any bugs, performance issues, or potential security vulnerabilities that may arise in a live environment.

User feedback will be actively sought and addressed to enhance the user experience and address any unforeseen challenges. Regular updates and improvements to the dApp will be implemented based on this feedback, ensuring that the system remains responsive to user needs and expectations.

Security measures will be a top priority, with a commitment to staying informed about emerging threats and vulnerabilities. Any necessary updates or patches will be promptly applied to maintain the integrity and security of the crowdfunding dApp.

Conclusion and Future Works

In the pursuit of creating a decentralized crowdfunding platform, the development and implementation of the Crowdfunding dApp project have been guided by a commitment to transparency, security, and user empowerment within the evolving landscape of blockchain technology. The combination of Ethereum, Next.js, Hardhat, Solidity, and the integration of MetaMask as a decentralized wallet has resulted in a robust and user-friendly platform designed to revolutionize the crowdfunding experience.

Throughout the project, our team has adhered to the fundamental principles of decentralization, ensuring that users have full control over their funds, project creators benefit from a fair and transparent fundraising environment, and governance processes are inclusive and community-driven.

The development journey has not been without its challenges. Ensuring security in smart contract execution, addressing scalability concerns, and navigating the intricacies of decentralized governance presented hurdles that required meticulous planning and innovative solutions. Through collaborative efforts and continuous iteration, our team successfully navigated these challenges to deliver a Crowdfunding dApp that aligns with industry best practices.

As we conclude this phase of the Crowdfunding dApp project, we recognize that innovation is an ongoing process. Future enhancements may include further scalability measures, integration with emerging blockchain standards, and continuous improvement of user interfaces based on community feedback. Additionally, ongoing collaboration with regulatory bodies will be a priority to adapt to evolving compliance standards.

References

[1] L. Besançon, C. F. Da Silva, P. Ghodous and J. -P. Gelas, "A Blockchain Ontology for DApps Development," in IEEE Access, vol. 10, pp. 49905-49933, 2022, doi: 10.1109/ACCESS.2022.3173313.

[2] R. A. Mishra, A. Kalla, N. A. Singh and M. Liyanage, "Implementation and Analysis of Blockchain Based DApp for Secure Sharing of Students' Credentials," 2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2020, pp. 1-2, doi: 10.1109/CCNC46108.2020.9045196.

[3] R. Sujeetha and C. A. S. Deiva Preetha, "A Literature Survey on Smart Contract Testing and Analysis for Smart Contract Based Blockchain Application Development," 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2021, pp. 378-385, doi: 10.1109/ICOSEC51865.2021.9591750.

[4] A. Abuhashim and C. C. Tan, "Smart Contract Designs on Blockchain Applications," 2020 IEEE Symposium on Computers and Communications (ISCC), Rennes, France, 2020, pp. 1-4, doi: 10.1109/ISCC50000.2020.9219622.

[5] K. N. Pankov, "Testing, Verification and Validation of Distributed Ledger Systems," 2020 Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russia, 2020, pp. 1-9, doi: 10.1109/IEEECONF48371.2020.9078541.

[6] M. Muneeb, Z. Raza, I. U. Haq and O. Shafiq, "SmartCon: A Blockchain-Based Framework for Smart Contracts and Transaction Management," in IEEE Access, vol. 10, pp. 23687-23699, 2022, doi: 10.1109/ACCESS.2021.3135562.

[7] B. K. Mohanta, S. S. Panda and D. Jena, "An Overview of Smart Contract and Use Cases in Blockchain Technology," 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bengaluru, India, 2018, pp. 1-4, doi: 10.1109/ICCCNT.2018.8494045.

[8] Tiwari, Varun & Singh, & Vadi, Vikas. (2023). Smart Contract Using Solidity (Remix -Ethereum IDE). 12. 243-249. 10.17148/IJARCCE.2023.12253.

[9] M. Wohrer and U. Zdun, "Smart contracts: security patterns in the ethereum ecosystem and solidity," 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Campobasso, Italy, 2018, pp. 2-8, doi: 10.1109/IWBOSE.2018.8327565.

[10] H. V. Vhatkar, H. G. Singh, A. S. Sonavane, S. Singh and N. Pulgam, "Crowdfunding using Blockchain," 2023 11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP), Nagpur, India, 2023, pp. 1-6, doi: 10.1109/ICETET-SIP58143.2023.10151618.

[11] V. Patil, V. Gupta and R. Sarode, "Blockchain-Based Crowdfunding Application," 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2021, pp. 1546-1553, doi: 10.1109/I-SMAC52330.2021.9640888.