SCHOOL OF INFORMATION, COMPUTER AND COMMUNICATION TECHNOLOGY (ICT), SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY, THAMMASAT UNIVERSITY

PROJECT REPORT

CSS486 Blockchain Development

Charity Crowdfunding

By

Mr. Theerapat Wiwattanapornpanit 6222772525

Mr. Chanatip Phuwiphadawat 6222770792

Mr. Veerawich Kaoaien 6222782219

Section 1

Abstract—This technical report presents the development of a decentralized crowdfunding platform utilizing blockchain technology and smart contracts. The project aims to leverage blockchain's decentralized ledger and smart contract automation to enhance transparency, efficiency, and security in crowdfunding. The report outlines the design and implementation of the platform, highlighting key features including project creation, funding, and tracking. It also explores secure user authentication mechanisms. By demonstrating the potential of blockchain, this project showcases its ability to revolutionize traditional crowdfunding and foster trust among participants. The findings contribute to the existing knowledge on blockchain applications in crowdfunding and provide a foundation for further research and development in this field.

Keywords—Decentralized crowdfunding platform, Blockchain technology, Smart contracts, Transparency, Efficiency, Security, Project creation, Funding, Tracking, User authentication mechanisms, Trust, Revolutionize, Traditional crowdfunding, Findings, Insights, Blockchain applications, Crowdfunding

Introduction

Crowdfunding has emerged as a popular alternative for fundraising, offering individuals and organizations access to financial support from a wide pool of contributors. However, traditional crowdfunding platforms often face challenges related to transparency, intermediaries, and security. The advent of blockchain technology provides an opportunity to address these issues and revolutionize the crowdfunding landscape.

This report presents a comprehensive study on the design and implementation of a decentralized crowdfunding platform utilizing blockchain technology and smart contracts. By harnessing the inherent advantages of blockchain, such as transparency, immutability, and decentralization, the proposed platform aims to enhance the crowdfunding experience for both project creators and funders.

The report explores fundamental concepts and technologies involved in developing the crowdfunding platform. It examines the utilization of smart contracts, which are self-executing agreements stored on the blockchain, to automate various aspects of the crowdfunding process, including project creation, fund collection, and distribution. The transparency and security provided by blockchain technology play a pivotal role in establishing trust among participants and mitigating risks of fraud or fund misappropriation.

Moreover, the report delves into implementation details, including the selection of programming languages and development frameworks. It also discusses the integration of user authentication mechanisms to ensure secure participation within the platform.

Furthermore, the report highlights potential advantages of the proposed decentralized crowdfunding platform, such as increased accessibility for project creators, wider participation from funders, reduced reliance on intermediaries, and enhanced accountability through transparent transaction records. Additionally, it acknowledges the limitations and challenges associated with blockchain-based crowdfunding, such as scalability issues and regulatory considerations.

This research aims to shed light on the possibilities and implications of leveraging blockchain technology in the context of crowdfunding. By examining the design and implementation of a decentralized crowdfunding platform, we contribute to the growing body of knowledge on blockchain applications in the field of finance and explore the potential of this innovative technology to reshape traditional fundraising practices.

Technological Background

Crowdfunding has emerged as a popular method for raising funds, offering individuals and organizations access to financial support from a large pool of contributors. However, traditional crowdfunding platforms often face challenges related to transparency, intermediaries, and security. The advent of blockchain technology presents an opportunity to address these challenges and revolutionize the crowdfunding landscape.

Blockchain Technology:
Blockchain technology is a distributed and decentralized ledger system that enables the secure and transparent recording of transactions across multiple computers or nodes. It operates on a consensus mechanism, where transactions are verified and added to the blockchain through a network of participants, eliminating the need for a central authority. Each transaction is cryptographically linked to the previous one, forming a chain of blocks that are immutable and tamper-resistant. The use of blockchain technology provides several benefits such as transparency, security, immutability, and trustworthiness.

Smart Contracts:
Smart contracts are self-executing agreements encoded on the blockchain that automatically enforce predefined conditions and actions. They enable trustless and automated interactions between parties, as the contract's terms and conditions are directly written into the code. Smart contracts are integral to blockchain-based applications, allowing for the execution of predefined functions, data storage, and interactions with other contracts. In the proposed decentralized donation platform, smart contracts will be utilized to handle the creation and management of donation projects, track fund transfers, and maintain an immutable record of transactions.

Cryptocurrency:
Cryptocurrencies are digital or virtual currencies that utilize cryptographic techniques for secure transactions and control the creation of new units. They operate on decentralized networks, typically blockchain-based, and provide a means of transferring value without the need for intermediaries like banks. Cryptocurrencies offer advantages such as fast and low-cost transactions, global accessibility, and the potential for pseudonymous or anonymous transactions. In the context of the decentralized donation platform, accepting donations in cryptocurrency provides users with the option of making contributions anonymously and facilitates seamless cross-border transactions.

Web-based Interface:
A web-based interface serves as the user-facing component of the decentralized donation platform. It provides a user-friendly and accessible means for individuals to interact with the platform, create accounts, browse donation projects, make contributions, and track their donations. The web-based interface will utilize modern web technologies such as HTML, CSS, and JavaScript to create an intuitive and responsive user experience. It will integrate with the underlying blockchain infrastructure and smart contracts to enable seamless interaction with the decentralized donation platform.

Ganache Blockchain:
Ganache is a local development blockchain that is commonly used for testing and development purposes. It provides a simulated Ethereum environment where developers can deploy and test their smart contracts without interacting with the main Ethereum network. Ganache allows for faster and more efficient development by providing a local blockchain network with predefined accounts and test Ether. It also offers features such as transaction inspection, contract debugging, and network customization. In the proposed decentralized donation platform, Ganache blockchain can be utilized during the development and testing phases to ensure the smooth functioning of smart contracts and interactions with the web-based interface.

Solidity:
Solidity is a programming language specifically designed for writing smart contracts on the Ethereum blockchain. It is a statically-typed language that resembles JavaScript and is used to define the behavior of smart contracts in the decentralized ecosystem. Solidity enables developers to specify the functions, data structures, and interactions within the smart contract. It also supports various features like inheritance, libraries, and interfaces. Solidity code is compiled into bytecode that can be executed on the Ethereum Virtual Machine (EVM). In the context of the decentralized donation platform, Solidity will be used to write and deploy the smart contracts that handle the creation, funding, and management of donation projects.

Integration of Ganache, Solidity, and Web Interface:
The integration of Ganache, Solidity, and the web-based interface is crucial for the seamless functioning of the decentralized donation platform. Ganache provides a local blockchain environment for testing and development, allowing developers to deploy and interact with smart contracts written in Solidity. The web-based interface serves as the user-facing component, enabling users to access and interact with the platform's features. The interface communicates with the Ganache blockchain and interacts with the deployed smart contracts using web3

libraries or similar tools. This integration ensures that the web interface can display real-time project information, facilitate donation transactions, and retrieve data from the blockchain in a secure and reliable manner.

Additionally, it's important to mention the potential benefits of integrating blockchain technology and smart contracts in the context of crowdfunding. These benefits include increased transparency, reduced transaction costs, enhanced security, elimination of intermediaries, and automation of crowdfunding processes. By leveraging these technologies, the decentralized donation platform can provide a more efficient, transparent, and inclusive fundraising experience for both project creators and contributors.

Our Proposed System

The proposed system is a decentralized donation platform that leverages blockchain technology, smart contracts, and a web-based interface to facilitate transparent and secure donation processes. The system aims to provide users with an efficient and trustworthy platform to create and announce donation projects, accept contributions in cryptocurrency, and maintain immutable records of transactions. Additionally, it offers user authentication and registration functionality through a web-based interface.

The key components and functionalities of the proposed system include:

Decentralized Donation Platform:

The system utilizes blockchain technology to create a decentralized environment where donation projects and transactions are recorded on a public ledger. This ensures transparency, immutability, and tamper-resistance, providing users with a trustworthy and auditable platform. Each donation project is assigned a unique identifier and includes information such as the topic, description, funding goal, start and end dates, and the address of the project owner.

Smart Contracts:

Smart contracts written in Solidity are deployed on the blockchain to handle the creation, funding, and management of donation projects. These contracts define the rules and conditions for donation processes, including verifying project status, accepting contributions, and updating project fund status. By automating interactions between donors, project owners, and the platform, smart contracts ensure secure and reliable execution of transactions.

Cryptocurrency Donations:

The platform accepts donations in cryptocurrency, enabling users to contribute funds using popular cryptocurrencies like Bitcoin or Ethereum. Cryptocurrency donations offer benefits such as fast and low-cost transactions, global accessibility, and the potential for anonymous contributions. Users can make donations by sending cryptocurrency to the designated project address, triggering the execution of smart contracts and updating the project's funding status.

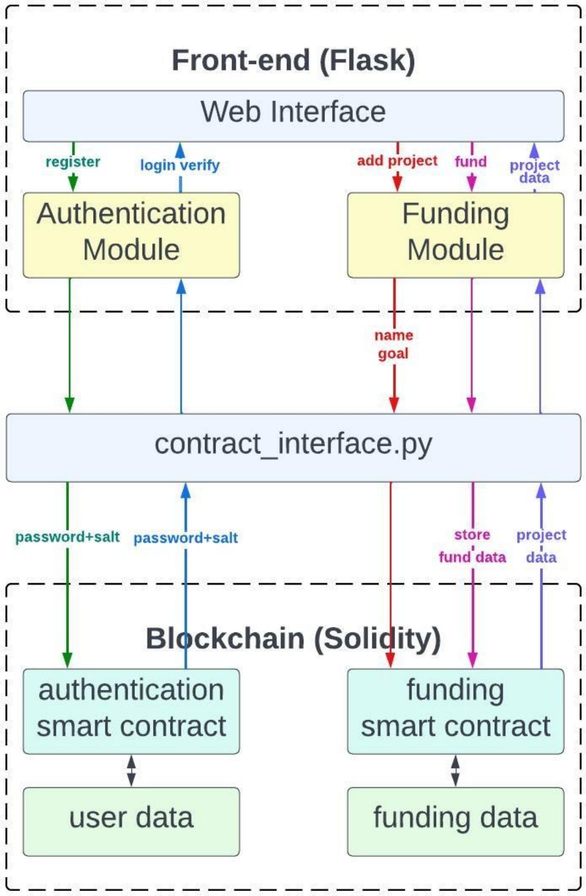
Web-based User Interface:

The system provides a web-based interface accessible to users, allowing them to browse and select donation projects, create accounts, and make contributions. The interface offers features for project creation, viewing project details, tracking donation progress, and managing user profiles. It integrates with the underlying blockchain and smart contracts through web3 libraries or similar tools, enabling real-time updates and secure interactions. The web-based interface utilizes modern web technologies such as HTML, CSS, and JavaScript to create an intuitive and responsive user experience.

User Authentication and Registration:

The system offers user authentication and registration functionality to ensure secure access and protect user information. Users can create accounts by providing necessary details such as username, password, and personal information. User authentication ensures that only authorized individuals can access and interact with the platform's features, maintaining the integrity and security of the donation processes. This authentication mechanism safeguards user privacy and ensures that donation transactions are performed by genuine users.

By combining these components and functionalities, the proposed system creates a decentralized donation platform that addresses the challenges of transparency, security, and intermediaries often associated with traditional crowdfunding methods. It empowers individuals and organizations to engage in trustworthy and efficient fundraising while providing contributors with a seamless and secure donation experience.

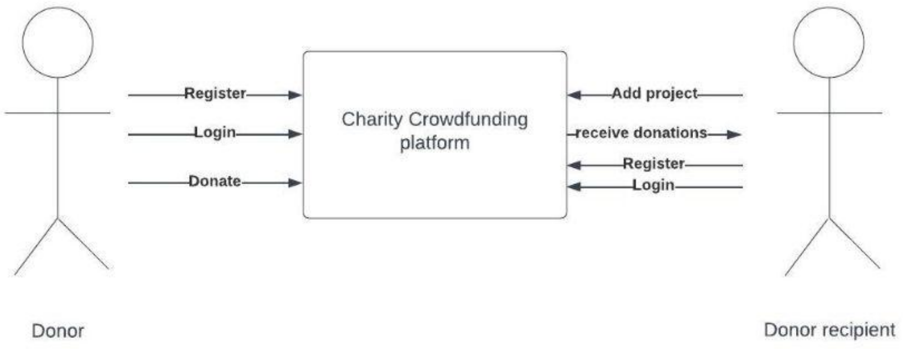

Front-end (Flask)
Web Interface
register
login verify
add project
fund
project data
Authentication Module
Funding Module
name goal
contract_interface.py
password+salt
password+salt
store fund data
project data
Blockchain (Solidity)
authentication smart contract
funding smart contract
user data
funding data

Experiment Detail

The development or experiment for the proposed decentralized donation platform involves several stages and processes to ensure the successful implementation of the system. The following development or experiment details outline the key steps involved:

1. **Requirements Gathering:**

- Conduct a thorough analysis to gather the functional and non-functional requirements of the decentralized donation platform.
- Identify the desired features, user roles, donation project details, authentication and registration requirements, and integration with the Ganache blockchain.



2. **Smart Contract Development:**

- Write the smart contracts using the Solidity programming language to handle the creation, funding, and management of donation projects.
- Define the data structures, functions, and conditions necessary for the smooth execution of donation processes.
- Deploy the smart contracts on local Ganache blockchain

In contract.sol, There are 2 main functions

1) Create project function:
   4 parameters:
   - topic: A string that represents the topic or title of the project.
   - description: A string that provides a description of the project.
   - goal: An unsigned integer (uint256) that indicates the funding goal for the project.
   - deadline: An unsigned integer (uint256) representing the deadline (in some units like seconds) by which the project should achieve its funding goal.

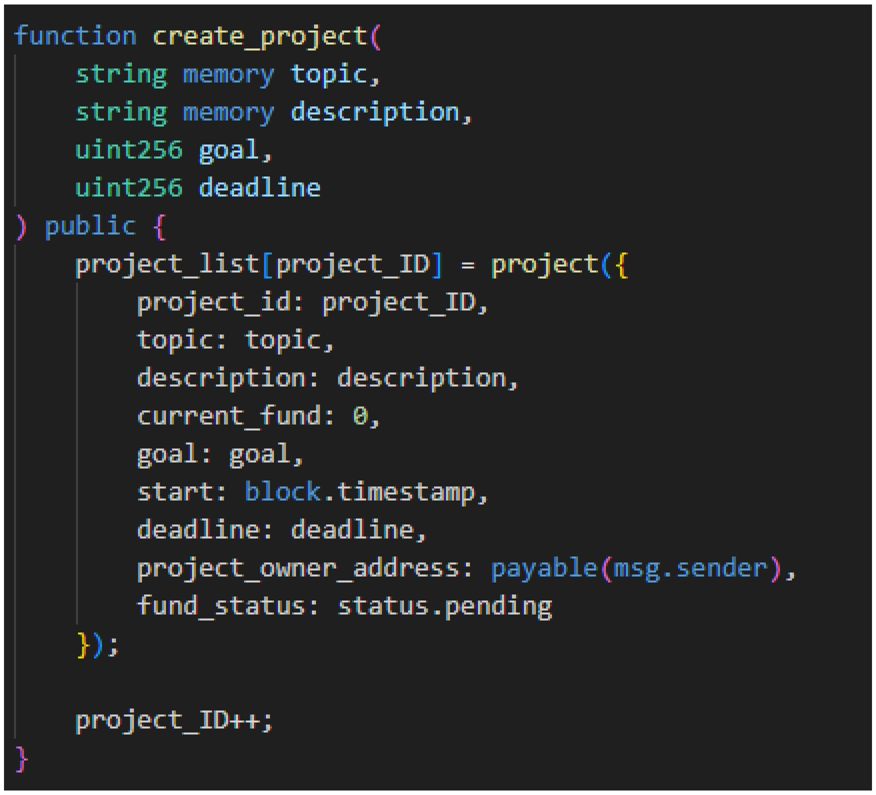
The function create_project is defined with public visibility, meaning it can be called from outside the contract by anyone. It takes four parameters: topic, description, goal, and deadline. The parameters are passed as memory references, indicating that they will be temporarily stored in memory during the function execution.

Inside the function, a new project is created and stored in a mapping called project_list. The mapping associates a unique project ID (project_ID) with a struct called project, which contains various project details.

The project struct has the following properties:

- project_id: Represents the ID of the project.
- topic: Stores the topic or title of the project.
- description: Stores the description of the project.
- current_fund: Represents the current amount of funds raised for the project. Initially set to 0.
- goal: Indicates the funding goal for the project.
- start: Stores the timestamp of when the project was created. It is set to the current block's timestamp using block.timestamp.
- deadline: Indicates the deadline for achieving the funding goal. It is specified by the caller of the function.
- project_owner_address: Stores the Ethereum address of the user who created the project. It is set to msg.sender, which represents the address of the caller of the function.
- fund_status: Represents the status of the project's funding. It is initially set to status.pending.
-

Finally, after creating and storing the project, the project_ID is incremented by 1 to ensure that the next project created will have a unique ID.



```
function create_project(
    string memory topic,
    string memory description,
    uint256 goal,
    uint256 deadline
) public {
    project_list[project_ID] = project({
        project_id: project_ID,
        topic: topic,
        description: description,
        current_fund: 0,
        goal: goal,
        start: block.timestamp,
        deadline: deadline,
        project_owner_address: payable(msg.sender),
        fund_status: status.pending
    });

    project_ID++;
}
```

2) Fund project function:

The function fund_project is defined as external, meaning it can be called from outside the contract by anyone. It takes a single parameter ID of type uint256, which represents the ID of the project to fund.

The function begins with several required statements that serve as preconditions before funding the project. These conditions ensure that:

- The funding status of the project with the given ID is status.pending. If it is not pending, i.e., the funding is already terminated, the function reverts with an error message.
- The current timestamp (block.timestamp) is less than the deadline specified for the project. If the deadline has already passed, the function reverts with an error message.
- The balance of the caller (msg.sender) is greater than the amount being contributed (msg.value). If the caller's balance is insufficient, the function reverts with an error message.

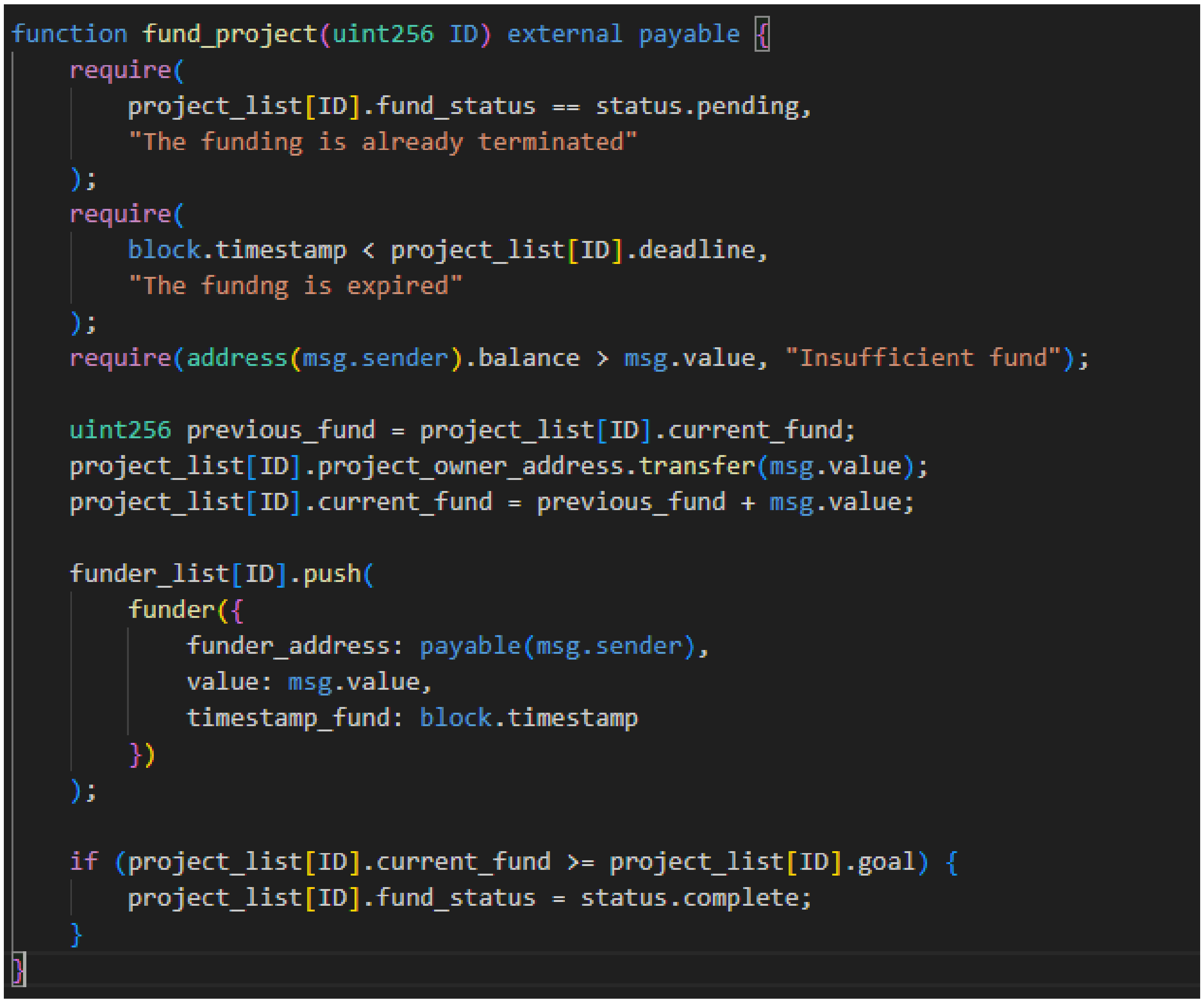
After the precondition checks, the function proceeds to fund the project. It first stores the current value of project_list[ID].current_fund in a variable called previous_fund.

Then, the amount of ether sent with the function call (msg.value) is transferred to the project owner's address (project_list[ID].project_owner_address) using the transfer function. This action effectively sends the contributed funds to the project owner.

Next, the current_fund property of the project with the given ID is updated by adding the contributed amount to the previous_fund value.

The code then appends the funder's information to the funder_list mapping for the corresponding project ID. It creates a new funder struct and populates it with the funder's Ethereum address (msg.sender), the contributed value (msg.value), and the timestamp of the fund transaction (block.timestamp). The struct is then pushed into the funder_list array for the specified project ID.

Lastly, the function checks if the current fund amount (project_list[ID].current_fund) for the project with the given ID is greater than or equal to the funding goal (project_list[ID].goal). If it is, the fund_status of the project is updated to status.complete, indicating that the funding goal has been reached.



```
function fund_project(uint256 ID) external payable {
    require(
        project_list[ID].fund_status == status.pending,
        "The funding is already terminated"
    );
    require(
        block.timestamp < project_list[ID].deadline,
        "The fundng is expired"
    );
    require(address(msg.sender).balance > msg.value, "Insufficient fund");

    uint256 previous_fund = project_list[ID].current_fund;
    project_list[ID].project_owner_address.transfer(msg.value);
    project_list[ID].current_fund = previous_fund + msg.value;

    funder_list[ID].push(
        funder({
            funder_address: payable(msg.sender),
            value: msg.value,
            timestamp_fund: block.timestamp
        })
    );

    if (project_list[ID].current_fund >= project_list[ID].goal) {
        project_list[ID].fund_status = status.complete;
    }
}
```

In simple_authen.sol, There is a main function

1) Fund project function:

The function register is defined with the public visibility, meaning it can be called from outside the contract by anyone. It takes six parameters: username, password, salt, account_address, name, and surname. The parameters are passed as memory references, indicating that they will be temporarily stored in memory during the function execution.

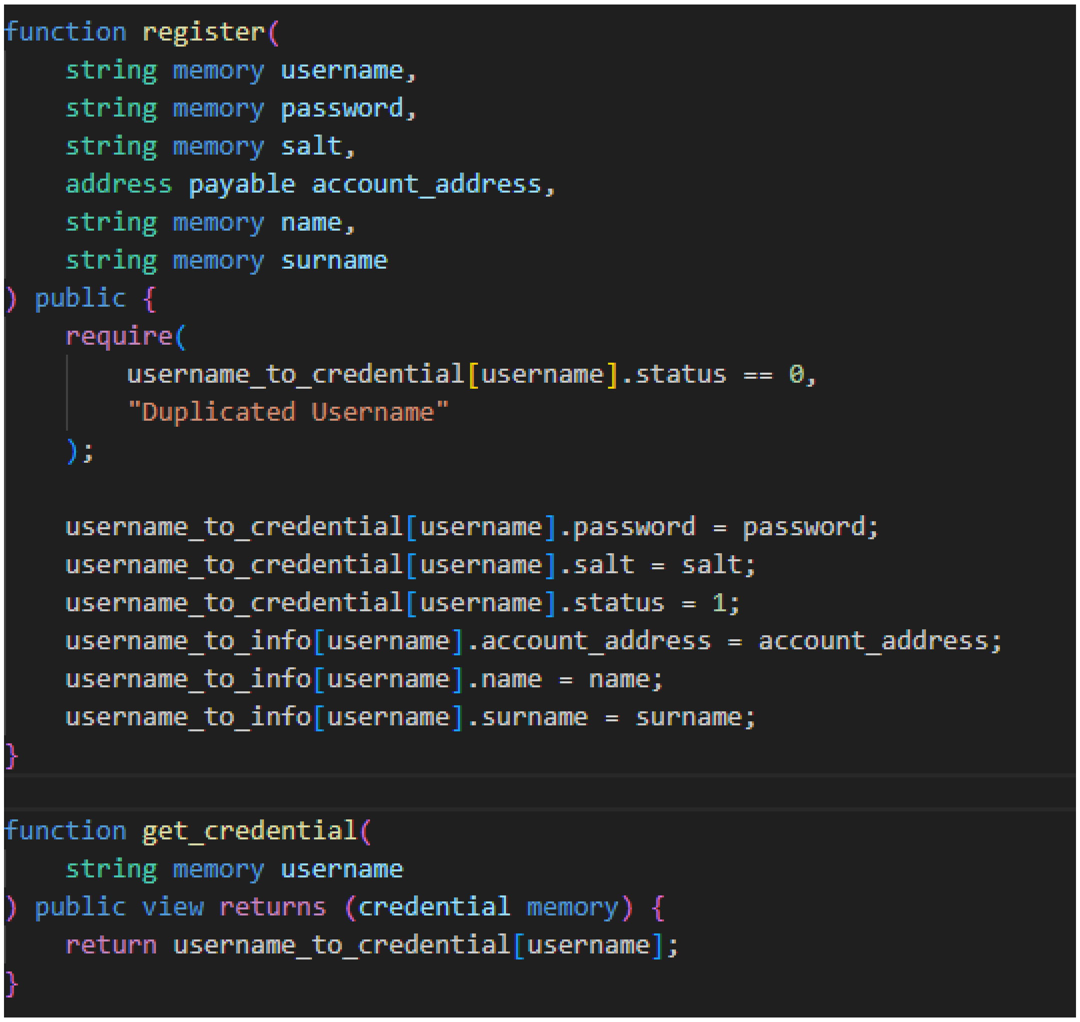
The function starts with a require statement that checks if the status of the username in the username_to_credential mapping is 0, indicating that the username is not already registered. If the status is not 0, meaning the username already exists, the function reverts with an error message stating "Duplicated Username."

After the check, the function proceeds to register the user by storing their credentials and information in two different mappings: username_to_credential and username_to_info.

In the username_to_credential mapping, the password and salt associated with the username are set to the provided values. Additionally, the status is updated to 1, indicating that the username is now registered.

In the username_to_info mapping, the user's account address, name, and surname are stored based on the username.

Overall, the register function allows users to register by providing their username, password, salt, account address, name, and surname. It ensures that the username is not already registered before storing the user's information in the contract's mappings.



```
function register(
    string memory username,
    string memory password,
    string memory salt,
    address payable account_address,
    string memory name,
    string memory surname
) public {
    require(
        username_to_credential[username].status == 0,
        "Duplicated Username"
    );

    username_to_credential[username].password = password;
    username_to_credential[username].salt = salt;
    username_to_credential[username].status = 1;
    username_to_info[username].account_address = account_address;
    username_to_info[username].name = name;
    username_to_info[username].surname = surname;
}

function get_credential(
    string memory username
) public view returns (credential memory) {
    return username_to_credential[username];
}
```

For Deploying smart contract, we deploy with contract_deploy.py:

Solidity Compiler Setup:

- The code begins by importing necessary libraries and modules, such as solcx for Solidity compilation and web3 for interacting with the blockchain network.
- The script changes the current working directory to the "smart_contract" folder using the chdir function.
- It installs the Solidity compiler version 0.8.5 using the install_solc function from solcx.

Blockchain Network and Web3 Setup:

- The script reads the blockchain network information from the "network_deploy.txt" file.
- It creates an instance of the Web3 library, connecting to the specified blockchain network using the HTTP provider.

Reading Smart Contract Source Code:

- The script reads the source code of the main contract from the "contract.sol" file and the authentication contract from the "simple_authen.sol" file.

Compilation of Smart Contracts:

- The script compiles the main contract using the compile_standard function from solcx. It specifies the Solidity version, the source code content, and the desired output selection, including ABI, bytecode, and metadata.
- The authentication contract is also compiled similarly.

Retrieving Bytecode and ABI:

- The compiled contracts' bytecode and ABI are extracted from the compiled output for further use.

Saving Compilation Output:

- The compiled contract and authentication contract information are saved to JSON files using the dump function from the json module.

Deployment of Smart Contracts:

- The script initializes the deployment address from the first account available in the Web3 provider.
- It creates instances of the main contract and authentication contract using the ABI and bytecode.

- The contracts are deployed using the constructor() function, and the deployment transactions are sent from the deployment address.
- The script waits for the deployment transactions to be mined and retrieves the contract addresses from the transaction receipts.
- The contract addresses are saved to separate text files.

Saving Account Addresses:

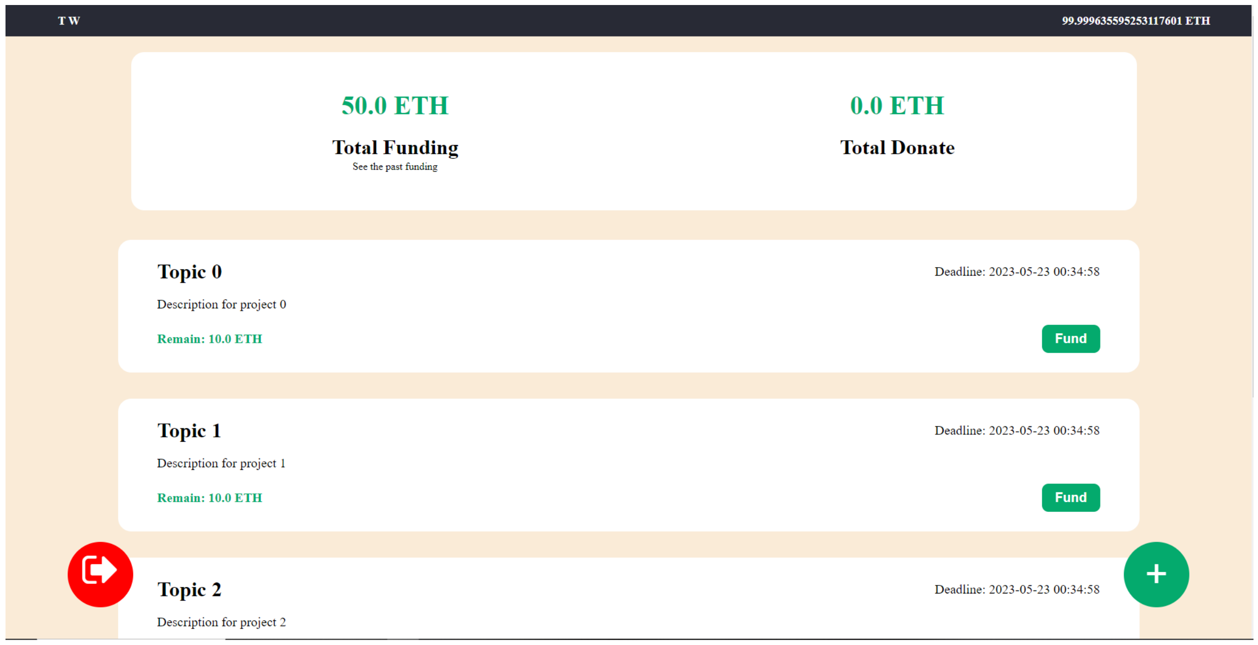
- The addresses of all available accounts in the Web3 provider are saved to the "account_address_list.txt" file.

In summary, this code snippet compiles and deploys smart contracts using the Solidity compiler and Web3 library. It saves the compiled contract information, contract addresses, and account addresses to relevant files for further use in interacting with the deployed contracts on the blockchain network.

3. Web-Based Interface Development:

- Develop a user-friendly web-based interface using modern web technologies such as HTML, CSS, and JavaScript.
- Design and implement the interface to support features such as project creation, project browsing, user registration and login, and donation tracking.
- Integrate the interface with the smart contracts and Ganache blockchain to enable seamless interaction and real-time updates.

Home page: You can fund many projects. Create a project for your charity, and see overview of funding detail on this homepage.


T W

99.999635595253117601 ETH

50.0 ETH

Total Funding

See the past funding

0.0 ETH

Total Donate

Topic 0

Deadline: 2023-05-23 00:34:58

Description for project 0

Remain: 10.0 ETH

Fund

Topic 1

Deadline: 2023-05-23 00:34:58

Description for project 1

Remain: 10.0 ETH

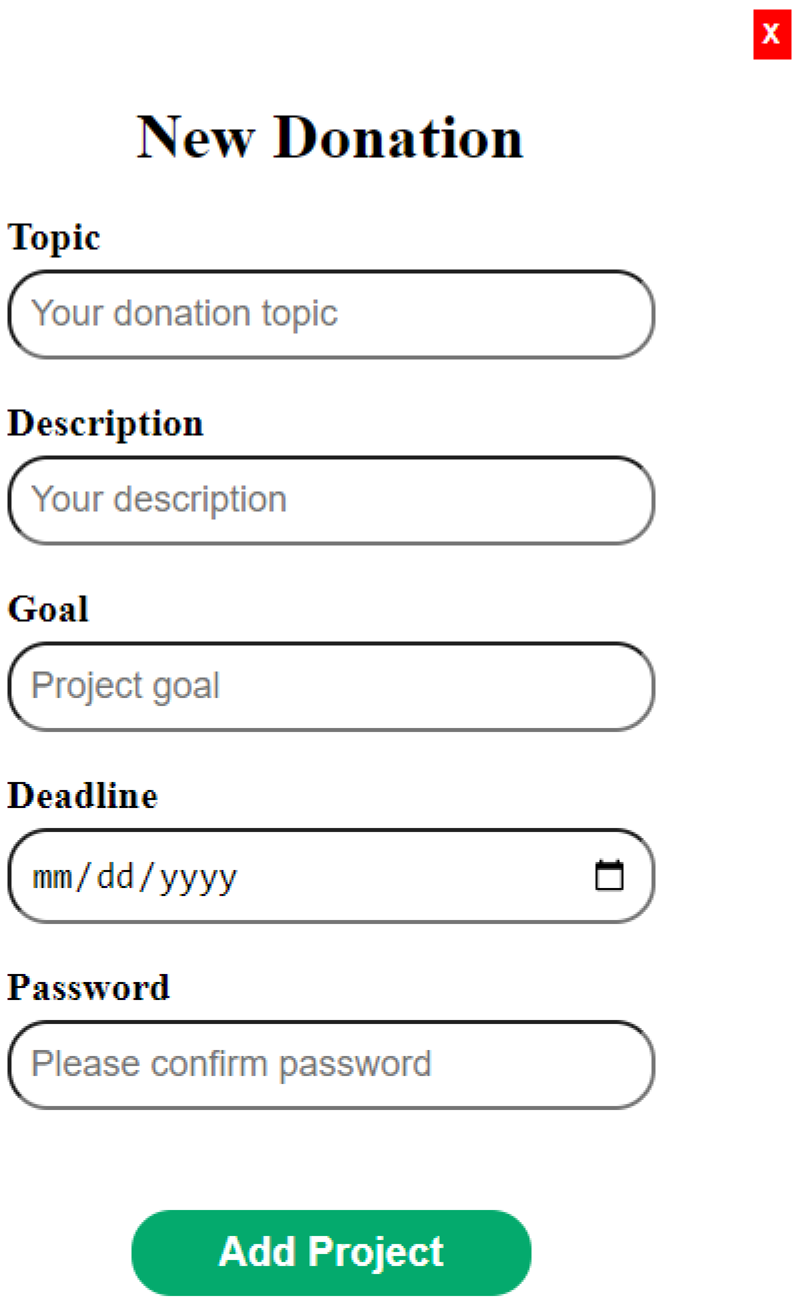
Fund

Topic 2

Deadline: 2023-05-23 00:34:58

Description for project 2

Create a project: You can put your topic, description, goal, deadline and create your charity project. We also check your password for high security performance on your account.



New Donation

Topic

Your donation topic

Description

Your description

Goal

Project goal

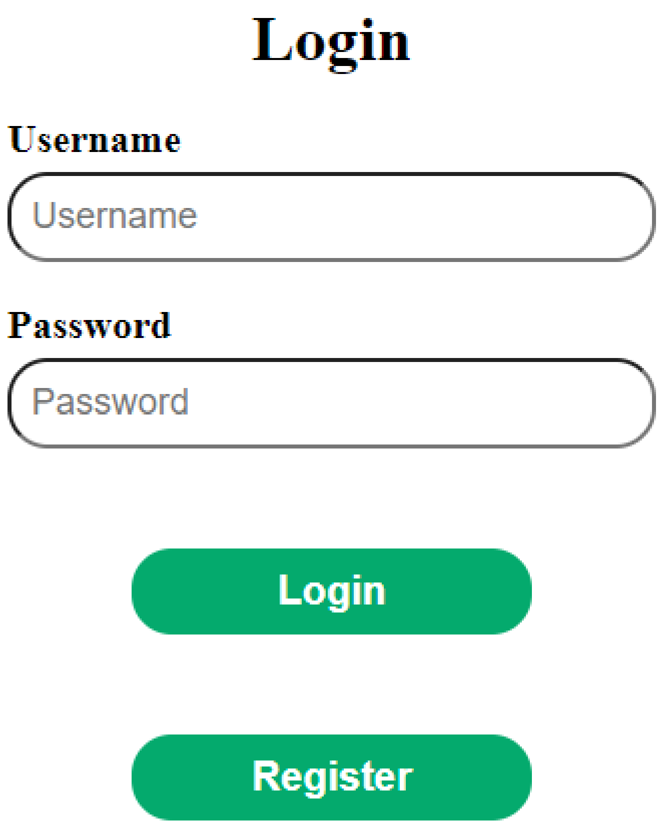
Deadline

mm/dd/yyyy

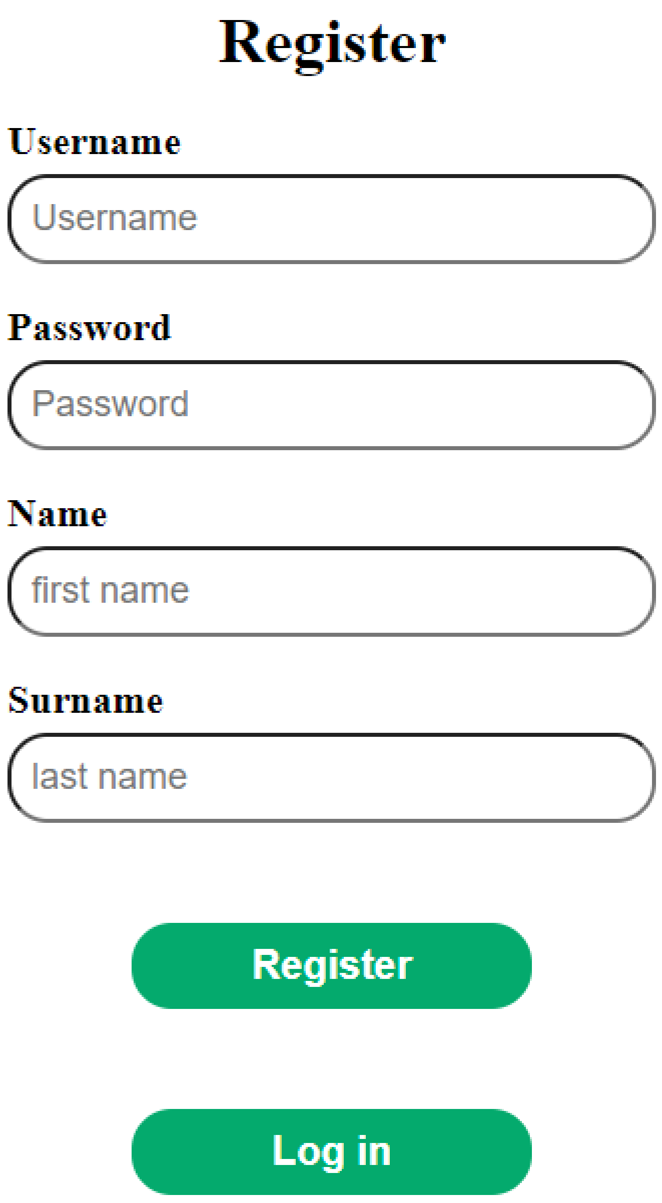
Password

Please confirm password

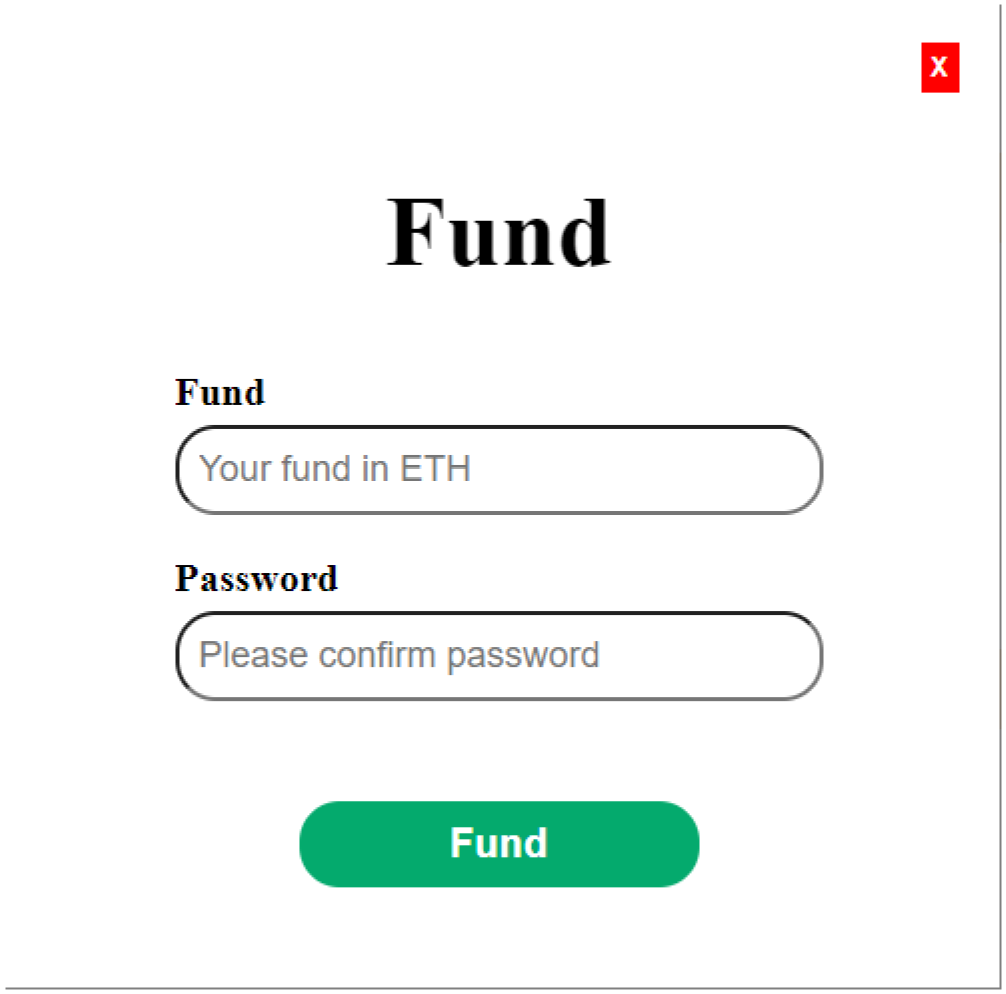
Login page: Before you want to go to our website. You have to login with your own account. Our website will check username and password that are correct or incorrect to your authentication.



Register page: You can store your credentials and information within the contract. To improve your money in this account. And we will protect your account by checking your username that is duplicated when creating your account.



Fund page: It shows when you clicked the fund button. You can put your money to fund this project that you are interested in. We also check your password for high security performance on your account.



For web interface.py worked with Flask:

Flask Setup:

- The code starts by importing the necessary modules from Flask, such as Flask for creating the application, render_template for rendering HTML templates, request for handling HTTP requests, and others.
- An instance of the Flask application is created with the name app, and a secret key is set to enable session management.

Route Definitions:

- The code defines several routes using the @app.route decorator to handle different URLs requested by the user.

The main routes include:

- / and /landing: Renders the landing page template.
- /login: Handles the login form submission and performs credential validation using the contract.credential_validation function from the contract_interface module.
- /register: Handles the registration form submission and calls the contract.register function to register a new user.
- /add_project: Handles the project creation form submission, validates the credentials, and calls the contract.create_project function to create a new project in the smart contract.
- /see_past: Renders the template to display past projects and related information. It retrieves project data and funder data using functions from the contract_interface module.
- /fund_project: Handles the project funding form submission, validates the credentials, and calls the contract.fund_project function to fund a project in the smart contract.
- /logout: Clears the session data and redirects to the landing page.
- /homepage: Renders the homepage template and retrieves user, project, and funder information to display using functions from the contract_interface module.

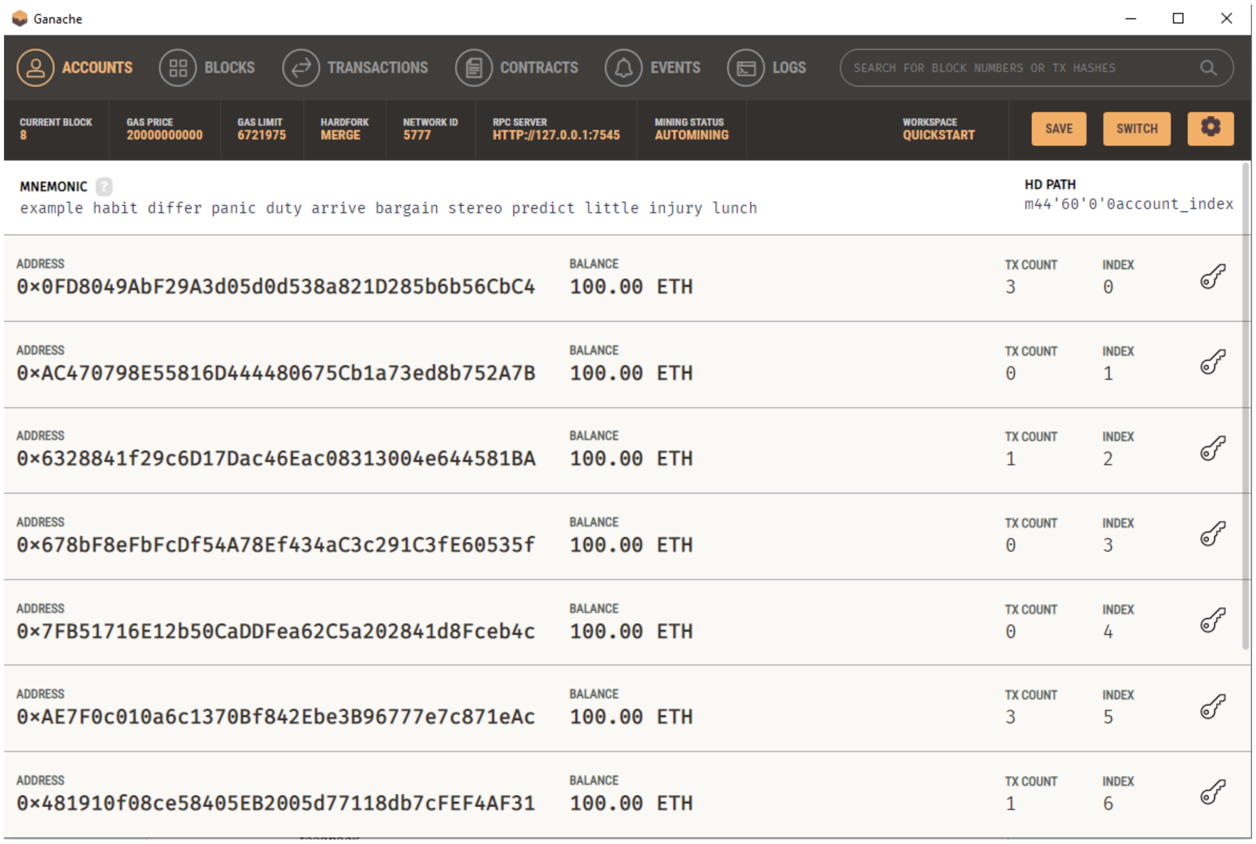
Flask Application Execution:

- The code runs the Flask application on the localhost server at port 5000 using app.run("localhost", 5000).

In summary, this code snippet sets up a Flask application to handle user interactions with the smart contract. It defines routes to handle login, registration, project creation, project funding, and displaying project-related information. The application communicates with the smart contract through functions from the contract_interface module.

4. **Ganache Blockchain Integration:**

- Set up a local Ganache blockchain environment for development and testing purposes.
- Deploy the smart contracts onto the Ganache blockchain network and configure the necessary accounts and test Ether.
- Establish the connection between the web-based interface and the Ganache blockchain using web3 libraries or similar tools.



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
8	20000000000	6721975	MERGE	5777	HTTP://127.0.0.1:7545	AUTOMINING	QUICKSTART

MNEMONIC
example habit differ panic duty arrive bargain stereo predict little injury lunch

HD PATH
m44'60'0'0account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×0FD8049AbF29A3d05d0d538a821D285b6b56CbC4	100.00 ETH	3	0
0×AC470798E55816D444480675Cb1a73ed8b752A7B	100.00 ETH	0	1
0×6328841f29c6D17Dac46Eac08313004e644581BA	100.00 ETH	1	2
0×678bF8eFbFcDf54A78Ef434aC3c291C3fE60535f	100.00 ETH	0	3
0×7FB51716E12b50CaDDFea62C5a202841d8Fceb4c	100.00 ETH	0	4
0×AE7F0c010a6c1370Bf842Ebe3B96777e7c871eAc	100.00 ETH	3	5
0×481910f08ce58405EB2005d77118db7cFEF4AF31	100.00 ETH	1	6

5. **Testing and Quality Assurance:**

- Perform comprehensive testing of the entire system to ensure its functionality, reliability, and security.
- Conduct unit tests for the smart contracts, validating their behavior under various scenarios.
- Execute integration tests to verify the interaction between the web-based interface, smart contracts, and the Ganache blockchain.
- Implement security measures such as input validation, authentication, and encryption to protect user data and transactions.

To test our website, we created add sample data.py :

Importing Modules:

- The code imports the necessary modules, including contract_interface for interacting with the smart contract, random for generating random values, datetime for working with dates and timestamps, and timedelta for manipulating time intervals.

Loading Account Addresses:

- The code reads the account addresses from the "account_address_list.txt" file located in the "./smart_contract" directory. The file contains a list of Ethereum account addresses, with each address on a separate line.
- The account addresses are stored in the account_list variable for later use.

Generating Project Data and Creating Projects:

- The code generates project data using a loop. In this case, it creates 5 projects.
- For each iteration, a random account address is selected from the account_list using random.choice(account_list).
- The contract.create_project function is called with the generated project details, including a topic, description, goal, deadline, and the randomly selected account address.
- The project details include a unique topic and description for each project, a goal of 10 (presumably representing the fundraising goal), and a new timestamp for the deadline.

Retrieving Project Information:

- After creating the projects, the code retrieves project information using the contract.get_project function.
- It iterates over a range of 5 (presumably the number of projects created) and calls contract.get_project with the project ID to fetch the project details.
- The project details are printed, presumably for verification purposes.

In summary, this code snippet interacts with the smart contract by creating multiple projects with random project details and retrieves the information of these projects using the contract_interface module. The account addresses used for project creation are read from a file, and project details such as topic, description, goal, and deadline are generated randomly.

Conclusion

In conclusion, this technical report presented the development of a decentralized crowdfunding platform utilizing blockchain technology and smart contracts. The platform aims to revolutionize traditional crowdfunding by leveraging blockchain's transparency, immutability, and decentralized nature. By providing a secure and efficient environment for project creators and funders, the platform enhances transparency, eliminates intermediaries, and fosters trust among participants.

The report discussed the fundamental concepts and technologies involved in developing the platform, including blockchain technology, smart contracts, cryptocurrency donations, and a web-based interface. It highlighted the advantages of blockchain technology in crowdfunding, such as increased accessibility, wider participation, reduced reliance on intermediaries, and transparent transaction records.

The integration of Ganache, Solidity, and the web-based interface was emphasized as a crucial aspect of the proposed system, ensuring seamless functionality and real-time updates. The report also acknowledged potential challenges, such as scalability issues and regulatory considerations, associated with blockchain-based crowdfunding.

By demonstrating the potential of blockchain technology in crowdfunding, this project contributes to the existing knowledge on blockchain applications in the field of finance. It provides a foundation for further research and development in this area, opening up new possibilities for secure and transparent fundraising practices.

In conclusion, the proposed decentralized crowdfunding platform offers a promising solution to the challenges faced by traditional crowdfunding methods. By harnessing the power of blockchain technology and smart contracts, it enables a transparent, efficient, and secure fundraising experience for project creators and contributors alike.