TRUST BASED DECENTRALIZED CROWDFUNDING USING ETHEREUM BLOCKCHAIN

An Major Project Report Submitted
In partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering (Cyber Security)

by

KASAM ROHIT REDDY
(Regd No: 21N31A6237)
JANGAM SWATHI
(Regd No: 21N31A6228)
THUMMANAPALI MANIKANTA
(Regd No: 21N31A6262)

Under the Guidance of

Mrs. P. LAVANYA REDDY
Assistant professor
School of Emerging Technologies
MRCET (Autonomous Institution, UGC Govt. of India)

MRCET CAMPUS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-CYBER SECURITY
MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTU, Hyderabad, Approved by AICTE, Accredited by NBA & NAAC – 'A' Grade, ISO 9001:2015 Certified)
Maisammaguda (v), Near Dullapally, Via: Kompally, Hyderabad – 500 100, Telangana State, India.

2024-2025

TRUST BASED DECENTRALIZED CROWDFUNDING USING ETHEREUM BLOCKCHAIN

An Major Project Report Submitted
In partial fulfillment of the requirement for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering (Cyber Security)

by

KASAM ROHIT REDDY
(Regd No: 21N31A6237)
JANGAM SWATHI
(Regd No: 21N31A6228)
THUMMANAPALI MANIKANTA
(Regd No: 21N31A6262)

Under the Guidance of

Mrs. P. LAVANYA REDDY
Assistant professor
School of Emerging Technologies
MRCET (Autonomous Institution, UGC Govt. of India)

MRCET CAMPUS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-CYBER SECURITY
MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTU, Hyderabad, Approved by AICTE, Accredited by NBA & NAAC – 'A' Grade, ISO 9001:2015 Certified)
Maisammaguda (v), Near Dullapally, Via: Kompally, Hyderabad – 500 100, Telangana State, India.

2024-2025

DECLARATION

We hereby declare that the project entitled "**Trust based Decentralized Crowdfunding using Ethereum Blockchain**" submitted to **Malla Reddy College of Engineering and Technology,** affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH) as part of IV Year B.Tech – II Semester and for the partial fulfillment of the requirement for the award of **Bachelor of Technology** in **Computer Science and Engineering (Cyber Security)** is a result of original research work done by us.

It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree.

Kasam Rohit Reddy (21N31A6237)
Jangam Swathi (21N31A6228)
Thummanapalli Manikanta (21N31A6262)

CERTIFICATE

This is to certify that this is the Bonafide record of the project titled **Trust based Decentralized Crowdfunding using Ethereum Blockchain**, submitted by **Kasam Rohit Reddy (21N31A6237) , Jangam Swathi (21N31A6228)** and **Thummanapali Manikanta (21N31A6262)** of **B.Tech IV Year – II Semester** in the partial fulfillment of the requirements for the degree of **Bachelor of Technology** in **Computer Science and Engineering (Cyber Security)** during the year 2024-2025. The results embodied in this project report have not been submitted to any other university or institute for the award of any degree.

Project Guide
Department of CSE (ET)

Project Coordinator
Department of CSE (ET)

EXTERNAL EXAMINER

HEAD
OF THE DEPARTMENT

Date of Viva-Voce Examination held on:_____

ACKNOWLEDGEMENTS

We feel ourselves honored and privileged to place our warm salutation to ourcollege "Malla Reddy College of Engineering and Technology (Autonomous Institution – UGC Govt. of India) and our Principal **Dr. S Srinivasa Rao,** Professor who gave us the opportunity to do the Project during our IV Year B.Tech II Semester and profound the technical skills.

We express our heartiest thanks to our Director **Dr. V S K Reddy,** Professor for encouraging us in every aspect of our project and helping us realizeour full potential.

We also thankful to our Head of the Department **Dr. M V Kamal,** Professor for providing training and guidance, excellent infrastructure and a nice atmosphere for completing this project successfully.

We would like to express our sincere gratitude and indebtedness to our project supervisor, **Mrs. P. Lavanya Reddy,** Assistant Professor for his valuable suggestions and interest throughout the course of this project.

We convey my heartfelt thanks to our overall Project Coordinator **Mrs. P Satyavathi and Dr. M Narendra,** Professor for allowing for their regular guidance and constant encouragement during my dissertation work.

We would like to thank all our supporting **staff** of the Department of CSE (Emerging Technologies) and even all other department who have been helpful directly and in-directly in making our project a success.

Finally, we would like to take this opportunity to thank our **families** for their support and blessings for completion of our project that gave us the strengthto do our project.

Kasam Rohit Reddy **21N31A6237**
Jangam Swathi **21N31A6228**
Thummanapalli Manikanta **21N31A6262**

ABSTRACT

Our project is about building a decentralized crowdfunding application using blockchain technology. Traditional crowdfunding platforms are controlled by companies, which means there can be risks like fraud, high fees, or lack of trust.

This system uses smart contracts on the Ethereum blockchain to create a safe, transparent, and secure way to raise and manage funds. With this app, project creators can easily launch campaigns, and backers can send money while being able to track where and how the money is used.

The application is built using React for the front end and Web3.js to connect to the blockchain. All project data is stored on the blockchain, which means it cannot be changed or hidden. This helps build trust between users.

The platform removes the need for middlemen and gives full control to the users. In the future, features like identity checks, automatic refunds, and rating systems for project creators can be added. This project shows how blockchain can be used in real life to make funding safer and more open for everyone. It also helps promote the idea of decentralized apps (dApps) for finance and fundraising.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

List of Abbreviations:

ETH – Ether (Ethereum’s native cryptocurrency)

UI – User Interface

UML – Unified Modeling Language

SRS – Software Requirement Specification

dApp – Decentralized Application

EVM – Ethereum Virtual Machine

NFT – Non-Fungible Token

ABI – Application Binary Interface

CLI – Command Line Interface

NPM – Node Package Manager

IPFS – InterPlanetary File System

API – Application Programming Interfaceting

CHAPTER 1

INTRODUCTION

1.1 Introduction:

In today's digital world, trust, transparency, and security have become essential—especially in areas involving financial transactions and fundraising. Traditional crowdfunding platforms often involve centralized control, high service fees, and potential risks such as fraud or misuse of funds. To address these challenges, this project presents a Blockchain-Based Crowdfunding Application which is a decentralized solution built on Ethereum that ensures secure, transparent, and direct interactions between project creators and funders.

The primary goal of this application is to leverage smart contracts to automate the management of funds, enforce rules without third-party intervention, and provide real-time tracking of financial flows. Unlike traditional platforms that store and process data on centralized servers, this app uses blockchain to record all activities, ensuring that no data can be changed or hidden once stored. This level of immutability and transparency significantly enhances user trust.

Furthermore, the platform enables peer-to-peer transactions, eliminating the need for intermediaries and reducing associated costs. With built-in support for Web3 wallets and decentralized logic, users have full control over their contributions and projects. By combining blockchain's core principles with modern web technologies, this project aims to deliver a secure, user-driven, and cost-effective crowdfunding system, well-suited for the future of fundraising in the digital era.

1.2 Motivation:

The motivation behind developing a Blockchain-Based Crowdfunding Application stems from the growing need for trustworthy, transparent, and decentralized fundraising platforms. Traditional crowdfunding systems—such as Kickstarter and GoFundMe—often depend on centralized authorities to manage user data, control financial transactions, and enforce platform rules. This centralized structure can lead to issues such as lack of transparency, delayed fund disbursement, hidden fees, and even mismanagement of resources.

Blockchain technology offers a promising alternative by enabling peer-to-peer transactions, immutability, and trustless execution of smart contracts. By using Ethereum as the backbone for the project, smart contracts can automate processes like fund release, refunding, and goal verification—removing the need for intermediaries and reducing the risk of fraud.

Additionally, as blockchain adoption grows across sectors, there's a clear gap in user-friendly crowdfunding applications that leverage these benefits while maintaining an intuitive interface. This project aims to bridge that gap and empower individuals, startups, and communities to raise and manage funds efficiently and securely.

Ultimately, this project is driven by the vision of creating a democratic, secure, and transparent fundraising ecosystem that gives full control to users while ensuring accountability and efficiency through smart technology.

1.3 Literature Review

i) Giudici, G., Nava, R., & Rossi Lamastra, C. et.al, Crowdfunding: The Role of Blockchain Technology in Trust Disintermediation.

Published In: Journal of Industrial and Business Economics, 2018

Abstract: This study investigates how blockchain can enhance the transparency and security of crowdfunding by removing reliance on centralized platforms. Traditional crowdfunding models depend heavily on intermediaries, which can lead to issues such as lack of trust, delayed payments, and misuse of funds. The paper argues that blockchain provides a distributed trust mechanism that can significantly improve user confidence.

Methodologies Used: Case study analysis, blockchain platform evaluation (Ethereum), smart contract logic modeling.

Conclusion: The researchers concluded that blockchain-based crowdfunding systems can eliminate third-party interference, reduce operational costs, and increase campaign accountability. By enabling peer-to-peer interactions and storing immutable transaction records, the system ensures that funds are only released under clearly defined conditions, thus enhancing fairness and donor trust.

ii) Molavi, R., & McDonald, C. et.al, *Blockchain-Based Crowdfunding: An Overview and Research Agenda*.

Published In: Journal of Theoretical and Applied Electronic Commerce Research,2021.

Abstract: This paper provides a comprehensive overview of blockchain's application in crowdfunding and identifies gaps in the current research. It highlights the technological, legal, and social factors that impact blockchain-based crowdfunding adoption. The authors emphasize that while

blockchain provides transparency, there is still a need to address scalability, legal frameworks, and user experience.

Methodologies Used: Systematic literature review, comparative analysis of blockchain-based and traditional crowdfunding platforms, smart contract structure analysis.

Conclusion: The paper concludes that blockchain crowdfunding holds great potential in addressing key issues such as fraud and fund misuse. However, successful implementation depends on balancing decentralization with proper governance models, as well as developing interfaces that make these platforms accessible to non-technical users.

iii) Md. Nazmus Saadat et al., Blockchain Based Crowdfunding Systems

Published In: Indonesian Journal of Electrical Engineering and Computer Science, Vol. 15, No. 1, July 2019

Abstract: This paper discusses the implementation of blockchain in crowdfunding platforms to address major issues such as fraud, lack of transparency, and unregulated campaigns. It proposes a system that utilizes Ethereum smart contracts to automate fund handling, improving campaign legitimacy and execution timelines. The authors emphasize how smart contracts can prevent fraud and ensure project deliverables by only releasing funds when predefined conditions are met.

Methodologies Used: The authors developed a working prototype using ReactJS for the frontend, Node.js for the backend, and Solidity for writing smart contracts. The contracts were deployed on the Ethereum Rinkeby test network using Infura. Additionally, the system used MetaMask for user authentication and IPFS for decentralized file storage. Contributors could view transactions via Etherscan API, and a voting mechanism ensured

funds were released only when contributors approved expense requests.

Conclusion: The study concludes that blockchain significantly enhances trust and transparency in crowdfunding systems. By automating fund release and maintaining immutable transaction records, smart contracts eliminate the need for centralized control. The researchers also propose future integration of ERC-223 tokens for gas efficiency and better transaction handling. Although still in the prototype phase, the system demonstrates the viability of decentralized, trustless crowdfunding applications.

iv) D. L. Falak et al. , Crowd-Funding Using Blockchain Technology

Published In: International Journal of Research Publication and Reviews, Vol. 3, No. 11, November 2022

Abstract: This study introduces a blockchain-based crowdfunding system that overcomes limitations of traditional platforms, such as lack of transparency, high transaction fees, and the absence of donor control. It proposes a decentralized platform where donors can monitor their contributions and campaign owners are accountable for their spending. All activities are secured and recorded via smart contracts on Ethereum.

Methodologies Used: The platform was designed as a decentralized web app with a frontend allowing users to create projects, contribute funds, and approve or reject spending requests. Smart contracts, compiled using Solidity, were deployed on the Ethereum blockchain. The system's architecture ensures that every interaction between investor and campaign creator is mediated by code, reducing the role of human error or misuse.

Conclusion: The paper demonstrates that integrating blockchain into crowdfunding systems enhances operational efficiency and user confidence. The immutability of blockchain ensures secure storage of campaign data, and the use of smart contracts eliminates dependency on third-party

platforms. The authors also highlight future improvements, such as expanding regulatory support and legal framework integration.

v) Mollick, E. et al, *The Dynamics of Crowdfunding: An Exploratory Study.*

Published In: Journal of Business Venturing, Vol. 29, Issue 1, 2014

Abstract: Although this paper focuses on traditional crowdfunding, it provides a strong foundation for understanding crowdfunding dynamics, including success factors and challenges. It addresses key risks such as campaign delays and failure to deliver promised outcomes—problems that blockchain aims to mitigate.

Methodologies Used : A large-scale data analysis of Kickstarter campaigns was conducted to identify patterns in success rates, delivery timelines, and investor behavior. It also examined the role of project updates and community interaction.

Conclusion: The study emphasizes the need for transparency and accountability in crowdfunding. While it does not address blockchain directly, the issues it outlines—such as delays, fraud, and limited donor control—are directly addressed by decentralized technologies.

1.4 Problem Definition:

Traditional crowdfunding platforms face several challenges related to transparency, security, and centralization. Users are often required to place their trust in a centralized authority to manage funds, verify campaign legitimacy, and handle financial transactions. This centralized model creates multiple points of failure, such as fraudulent campaigns, misuse of funds, hidden transaction fees, delays in fund release, and lack of accountability.

Moreover, contributors typically have limited visibility into how their funds are being utilized, and once a campaign ends, there is no reliable mechanism to ensure that funds are spent as promised. In regions with less-developed financial infrastructure, even accessing these platforms can be a barrier due to banking requirements or currency limitations.

There is a clear need for a decentralized solution that enables trustless interactions between campaign creators and contributors, enforces transparency through immutable records, and automates the fund management process without relying on third-party intermediaries.

This project addresses these challenges by developing a Blockchain-Based Crowdfunding Application using Ethereum smart contracts. The platform ensures secure, transparent, and tamper-proof campaign creation and funding processes, empowering both donors and project creators with full control and visibility over transactions.

1.5 Objective of the Project:

The main objective of this project is to design and develop a decentralized crowdfunding platform using blockchain technology that ensures transparency, security, and trust between campaign creators and contributors. Unlike traditional platforms that rely on centralized intermediaries, this project aims to build a trustless system where all transactions and activities are recorded immutably on the blockchain.

The objectives of this project are as follows:

1. To implement a blockchain-based crowdfunding platform where users can create fundraising campaigns and receive contributions directly through smart contracts.
2. To automate the fund management process using Ethereum smart contracts that handle contributions, approvals, and fund disbursements .
3. To ensure transparency and trust by storing all transactions and campaign-related data on a public blockchain, allowing users to track funds in real-time.
4. To empower contributors with voting rights for approving fund withdrawal requests, giving them control over how their funds are used.
5. To develop an intuitive user interface that allows seamless interaction with the blockchain through wallet integration (e.g., MetaMask).
6. To reduce platform dependency and transaction fees by removing intermediaries and allowing peer-to-peer interactions.

CHAPTER 2

SYSTEM ANALYSIS

2.1 Existing System & Proposed System:

Existing System :

There are several popular crowdfunding platforms in use today, such as Kickstarter, GoFundMe, Indiegogo, and Patreon. These platforms have enabled individuals, entrepreneurs, and organizations to raise funds for a variety of projects by reaching out to a wide online audience. However, these systems operate on centralized architectures, which come with several limitations and risks.

While these platforms provide basic functionality for hosting campaigns and accepting contributions, they lack transparency, real-time control over funds, and full protection against fraud or misuse of resources. Additionally, contributors must rely entirely on the platform and campaign creator for fund utilization, which may not always result in the promised outcome.

Disadvantages of Current Existing System:

- Centralized control: Most existing crowdfunding platforms are owned and managed by centralized entities, which makes them a single point of failure and raises concerns about manipulation or platform downtime.
- High platform fees: Platforms typically charge significant service fees, ranging from 5% to 10% of the total funds raised, reducing the actual amount received by project creators.
- Lack of transparency: Donors often have no way to verify how their money is used after it is contributed, as platforms do not provide real-time fund

usage tracking.

- Risk of fraud or misuse: In many cases, campaign creators may fail to deliver on their promises, with minimal legal or technical safeguards in place to recover the funds.
- No control for contributors: Contributors cannot intervene or stop fund release if a campaign fails to deliver. They have no voting or approval rights in how funds are managed.
- Limited global access: Many platforms restrict usage based on region, currency, or banking systems, making it harder for users in developing countries to participate.
- Slow fund disbursement: Funds raised on centralized platforms often take days or even weeks to be processed and released to the campaign creators.
- Poor accountability mechanisms: There are limited tools for campaign evaluation or enforcement of milestones. Contributors must trust the creator without verification mechanisms.
- Regulatory dependence: Centralized platforms must comply with varying regional regulations that may limit innovation and increase operational costs for creators.
- Censorship and content restrictions: Centralized platforms often moderate or reject campaigns based on internal policies, political biases, or regional sensitivities, which can stifle innovation or activism.
- Lack of interoperability: Funds raised are typically bound to a single currency or platform, preventing easy integration with other services or wallets, especially in the blockchain ecosystem.
- Inadequate reward management: Traditional platforms lack automated mechanisms to ensure that promised rewards (e.g., merchandise or digital perks) are fulfilled or distributed to backers.

Proposed System:

Our proposed system is a web-based decentralized crowdfunding platform built on Ethereum blockchain technology using React. It eliminates the need for centralized intermediaries by using smart contracts to manage campaign creation, fund contributions, and withdrawal requests. This system ensures transparency and security by making all campaign data and transactions verifiable on the blockchain.

When a user initiates a campaign, the system automatically deploys a smart contract that manages the contributions, milestones, and fund release conditions. Contributors have the ability to approve or reject fund withdrawal requests, ensuring their investments are only used as agreed. The system is integrated with MetaMask for secure wallet-based authentication and interaction with the blockchain network.

Each campaign is isolated and managed through its own contract, allowing for separate tracking, contribution management, and milestone validation. The front-end interface is designed using modern web technologies, making the platform intuitive and accessible to both technical and non-technical users.

Designed to be open and accessible, the application can accommodate anyone with an internet connection, expanding the pool of project creators and backers.

Advantages of Proposed System:

- **Decentralized Fund Management:** Smart contracts handle all contributions and fund disbursements, removing the need for a central authority and reducing the risk of manipulation or fraud.

- **Real-Time Transparency**: All transactions and funding activities are recorded on the blockchain, making them publicly verifiable and

impossible to tamper with.

- **Contributor Empowerment**: Contributors can vote on withdrawal requests, giving them control over how and when their funds are used, ensuring better accountability from campaign creators.

- **Reduced Transaction Fees**: Unlike traditional platforms that charge high service fees, blockchain minimizes costs through peer-to-peer transactions and automatic contract execution.

- **No Central Point of Failure**: As a decentralized application (DApp), the system continues to function securely even if individual components or servers are compromised.

- **Global Accessibility**: The use of cryptocurrency removes barriers posed by regional banking systems, allowing anyone with internet access to launch or fund a campaign.

- **Automated Milestone Enforcement**: Funds can be locked and released in stages, depending on the completion of specific project goals, increasing trust between creators and backers.

- **Enhanced Security**: MetaMask integration ensures that all interactions require secure wallet verification, eliminating risks associated with passwords or server-based logins.

2.2 Functional Requirements:

i) MetaMask-Based User Authentication:

- Users should be able to log in through MetaMask, which verifies their Ethereum wallet address as a unique identity.
- The system must verify wallet connection and display appropriate errors if MetaMask is not installed or connected.
- Upon successful authentication, users should be able to access the platform's features securely without needing traditional credentials.

ii) Project Creation:

- Authenticated users should be able to create crowdfunding campaigns by providing a title, description, target amount, and deadline.
- The system must deploy a new smart contract on the Ethereum blockchain for each campaign.
- A confirmation message should be displayed upon successful project creation and contract deployment.

iii) Project Listing and Browsing:

- Users must be able to view a list of all active and completed crowdfunding campaigns on the homepage.
- Each project card should display key information such as creator address, amount raised, target goal, and deadline.
- The system should fetch real-time data from the blockchain to ensure up-to-date funding status.

iv) Project Funding:

- Users should be able to fund any listed project by entering the amount of Ether they wish to contribute.
- The transaction must be confirmed via MetaMask and recorded on the Ethereum blockchain.

- The system should update the campaign's funding status immediately after the transaction is mined.

v) Dashboard for Personal Campaigns:

- Each user should have access to a dashboard that displays the projects they have created or funded.
- The dashboard must show project status, funding progress, and campaign end dates for easy tracking.
- Information must be retrieved in real-time from the blockchain to reflect accurate funding data.

2.2.1 Software Requirements

1. Frontend Framework

- **React.js:**

  React is used to develop the user interface of the web application. It allows for dynamic rendering of campaign data and provides a responsive and efficient user experience. React also supports component-based architecture, making the app modular and maintainable.

2. Blockchain Wallet Integration

- **MetaMask:**

  MetaMask is a browser extension that serves as a cryptocurrency wallet and allows users to interact with the Ethereum blockchain directly from their browsers. It is used for user authentication, account access, and signing transactions.

3. Blockchain Development Environment

- **Truffle Suite:**

  Truffle is a development environment, testing framework, and asset pipeline for Ethereum. It simplifies the process of compiling,

deploying, and testing smart contracts.

- **Ganache:**
  Ganache is a local in-memory Ethereum blockchain used for testing and development purposes. It provides instant mining, logging, and account simulation to test smart contracts without using real cryptocurrency.

4. Backend and Package Management

- **Node.js:**
  Node.js is a JavaScript runtime used for server-side scripting and managing project dependencies through npm (Node Package Manager). It is essential for running scripts, compiling contracts, and building the React app.

5. Smart Contract Language

- **Solidity:**
  Solidity is the primary programming language for writing smart contracts on the Ethereum blockchain. It is used to define contract logic for managing crowdfunding campaigns, fund contributions, approvals, and disbursements.

6. Blockchain Platform

- **Ethereum:**
  Ethereum is the underlying decentralized platform used to deploy and execute smart contracts. The application interacts with the Ethereum blockchain to store data, execute transactions, and provide a secure and tamper-proof environment.

2.2.2 Hardware Requirements

- Ram: 4GB or more.

- Processor: intel i3 dual core processor.
- Processor Speed: 1GHz to 3GHz.
- Hard disk space: 500mb.

CHAPTER 3

SOFTWARE REQUIREMENTS

3.1 Software

1. **React**: React is a JavaScript library used for building fast and dynamic user interfaces. It allows developers to build reusable UI components and efficiently update and render the right components when data changes. In this project, React is used to develop the front-end of the application, providing an interactive interface for users to create and view crowdfunding campaigns.

   Key Features of React:

   - **Component-Based Architecture:** Enables modular and reusable UI components.
   - **Virtual DOM:** Improves performance by minimizing real DOM manipulation.
   - **Declarative Syntax:** Makes the UI code more predictable and easier to debug.
   - **Strong Community Support:** Widely adopted and supported with extensive libraries.
   - **Integration with Web3:** Easily integrates with Ethereum blockchain through Web3 and MetaMask.

2. **MetaMask**: MetaMask is a browser-based cryptocurrency wallet and gateway to blockchain applications. It allows users to interact with the Ethereum blockchain directly from their browser by signing and sending transactions securely.

   Key Features of MetaMask:

- **Ethereum Wallet:** Stores Stores ETH and other Ethereum-based tokens, and provides account management tools for users and developers.
- **Transaction Signing:** Securely signs transactions and smart contract calls, protecting users' private keys during blockchain interactions.
- **Browser Integration:** Works directly as a browser extension, allowing dApps to connect with the user's wallet in real time.
- **Account Switching:** Supports multiple wallet accounts, letting users manage different project or identity profiles conveniently.
- **Security and Privacy:** Keeps private keys locally encrypted and never shares them with the application, reducing the risk of exposure.

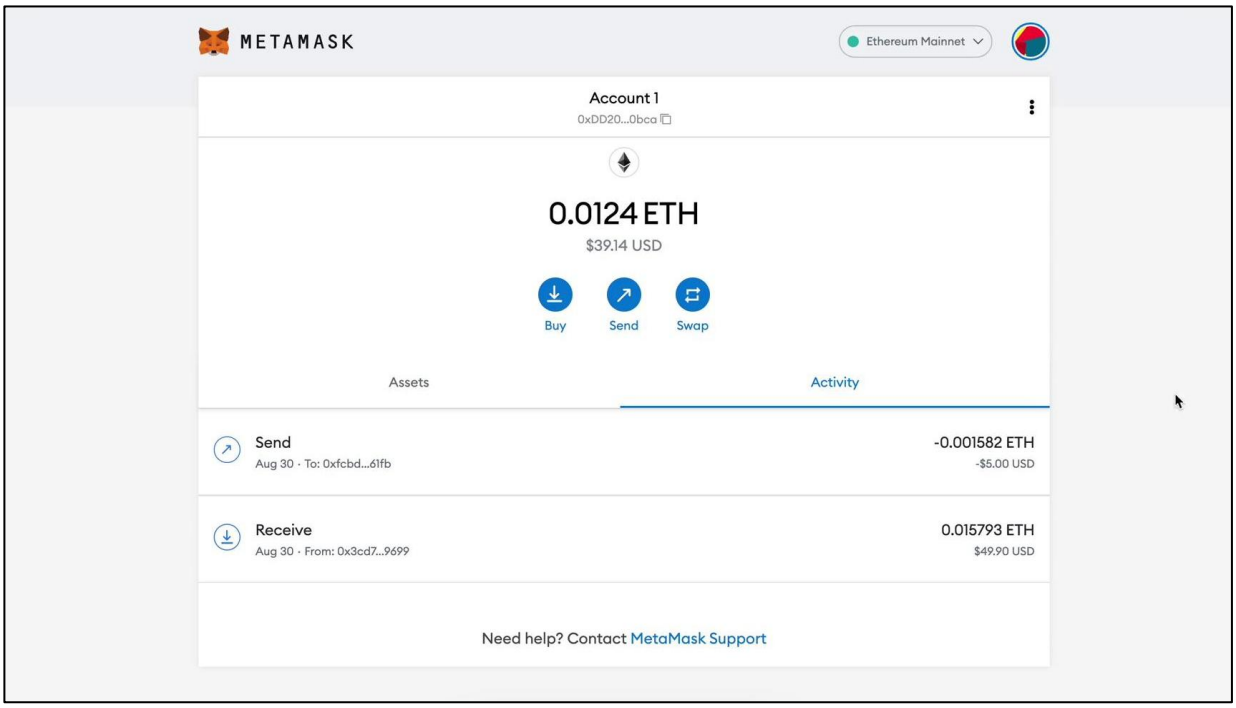

Fig 3.1.1 MetaMask Wallet

3. **Truffle :** Truffle is a popular development framework for Ethereum that simplifies the lifecycle of smart contract development, including compiling, deploying, and testing.

Key Features of Truffle:

- **Smart Contract Compilation:** Automatically compiles Solidity smart contracts, producing ABI and bytecode for blockchain deployment.
- **Migrations Management:** Uses structured migration scripts to handle smart contract deployment in a consistent and repeatable way.
- **Testing Framework:** Provides built-in support for writing unit and integration tests in JavaScript and Solidity to verify contract behavior.
- **Scriptable Deployments:** Developers can create scripts for custom logic during deployment, such as initializing states or seeding data.
- **Network Configuration:** Allows easy switching between local (Ganache) and public test networks like Rinkeby or Goerli.

4. **Ganache:** Ganache is a widely used blockchain emulator developed by Truffle Suite for Ethereum development. It provides a local, personal Ethereum blockchain environment that allows developers to simulate and test smart contracts without needing to interact with a real blockchain. Ganache is particularly useful for developers who want to quickly test their smart contracts, experiment with different scenarios, and debug issues in a controlled and safe environment.

Key Features of Ganache:

- **Local Blockchain Simulation:** Offers an instant, customizable blockchain for testing smart contracts without real currency or network latency.
- **Pre-Funded Test Accounts:** Provides accounts with fake Ether for free, allowing unlimited testing of transactions and contract logic.
- **Real-Time Logging:** Displays live updates of blockchain events, making it easier to track what's happening during development.
- **Automatic Block Mining:** Simulates Ethereum mining instantly,

removing delays and making testing faster and smoother.

- **Desktop and CLI Versions:** Available as a GUI for beginners and a command-line tool for advanced users and CI integration.

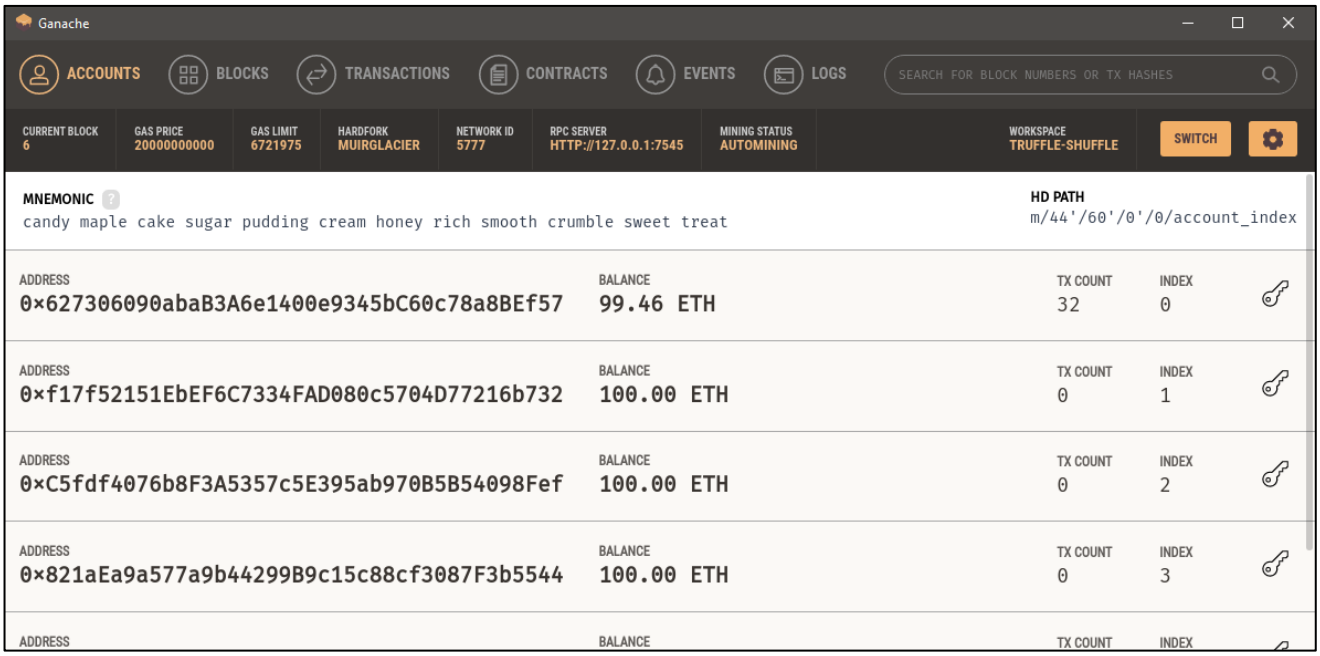

Fig 3.1.2 Ganache Home Page

5. **Node.js:** Node.js is a JavaScript runtime built on Chrome's V8 engine, widely used for server-side scripting and managing backend logic and development tools.

   Key Features of Node.js:

   - **Asynchronous I/O:** Uses non-blocking input/output to handle multiple operations simultaneously, ideal for high-performance applications.
   - **NPM Support:** Includes Node Package Manager (npm) to manage libraries and dependencies efficiently, such as Web3, Truffle, and more.
   - **Cross-Platform Compatibility:** Works across Windows, macOS, and Linux, ensuring portability for different developer environments.
   - **Real-Time Capability:** Supports event-driven architecture for building

real-time systems like wallets and data dashboards.

- **Scalable Infrastructure:** Easily integrates with microservices, APIs, and modern DevOps workflows, supporting scalable deployment.

6. **Solidity :** Solidity is the most widely-used programming language for writing smart contracts on the Ethereum blockchain. It is a statically typed, contract-oriented language.

   Key Features of Solidity:

   - **Smart Contract Development:** Enables developers to build decentralized applications by defining custom rules and automated logic.
   - **Support for Complex Data Types:** Offers arrays, mappings, structs, and more, supporting rich business logic and workflows.
   - **Event Logging Mechanism:** Emits events that help frontends and backends track state changes on the blockchain efficiently.
   - **Inheritance and Modifiers:** Supports object-oriented principles such as inheritance and modifiers for code reusability and security.
   - **Widespread Adoption:** Backed by a large developer ecosystem, tutorials, and tools, making it the go-to choice for Ethereum developers.

7. **Ethereum:** Ethereum is an open-source blockchain platform that enables developers to build and deploy decentralized applications (dApps) and smart contracts.

   Key Features of Ethereum:

   - **Smart Contract Execution:** Uses Ethereum Virtual Machine (EVM) to run contract logic in a secure and decentralized manner.
   - **Decentralized Ledger:** Ensures tamper-proof recording of all transactions, increasing transparency and data security.

- **Token Standards Support:** Allows creation of custom assets using standards like ERC-20 and ERC-721 for tokens and NFTs.
- **Global Network:** Provides access to a wide, decentralized network of nodes, ensuring high reliability and uptime.
- **Integration with Web3 Ecosystem:** Easily connects with wallets like MetaMask, tools like Truffle, and libraries like Web3.js for full-stack dApp development.

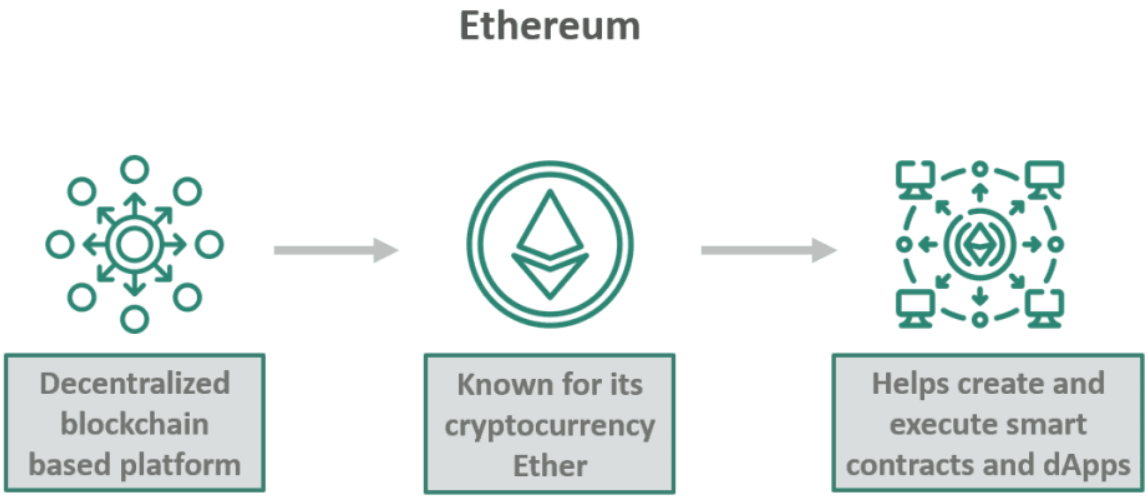

Fig 3.1.3 Ethereum Blockchain

3.2 Modules

Making of this project consisted of different modules such as,

1. Meta Mask Login

Functionality:

This module handles secure user authentication by integrating the MetaMask browser extension. Users must have the MetaMask wallet installed and connected to access the application. Upon login, the system fetches and verifies the user's Ethereum wallet address.

Key Features:

- Wallet-based authentication via MetaMask plugin for added blockchain security.

- Automatic detection of connected Ethereum account during login.
- Secure transaction signing using the MetaMask interface.
- Prevents unauthorized access by verifying wallet ownership on each session.

2. Projects Creation Module

Functionality:

This module allows authenticated users to create new crowdfunding projects on the blockchain. Users input details like project name, description, funding goal, and deadline via a user-friendly form. Upon submission, these inputs are packaged into a smart contract and deployed to the Ethereum network.

Key Features:

- Form-based UI for entering project name, description, deadline and funding goal.
- Smart contract interaction to register new projects on the Ethereum network.
- Automatic assignment of project creator address to track ownership.
- Validations to ensure no field is left incomplete during project creation.
- Support for deletion of messages from both sender and receiver devices.

3. Projects Home Page

Functionality:

Displays a central dashboard of all active and past crowdfunding projects. Users can browse project details, funding status, and view available campaigns to support. It also includes search and filter options to help users find specific campaigns or categories.

Key Features:

- Lists all deployed project smart contracts with summary cards.
- Displays live funding stats using real-time blockchain queries.
- Filters and sorting options for viewing specific types of campaigns.
- Option to view individual project details for funding or insight.

4. Funding Project and MetaMask Integration

Functionality:

Allows users to contribute Ether to a selected project directly through MetaMask. This module handles real-time blockchain interaction to process the funding transaction. The system handles errors like insufficient balance or failed transactions gracefully. This module ensures that contributions are recorded on-chain and reach the intended smart contract directly.

Key Features:

- Simple and intuitive UI for entering the amount to fund a project.
- Integration with MetaMask to prompt transaction signing and broadcasting.
- Confirmation alert once the transaction is recorded on the Ethereum chain.
- Updates project status and funding total after successful contributions.

CHAPTER 4

SYSTEM DESIGN

4.1 DATAFLOW DIAGRAM:

A Data Flow Diagram (DFD) is a graphical representation used to visualize the flow of data within a system. It illustrates how information moves between external entities, processes, and data stores. DFDs help identify how data is input, processed, and output by the system. They are commonly used in system analysis to understand and document the structure and flow of information.

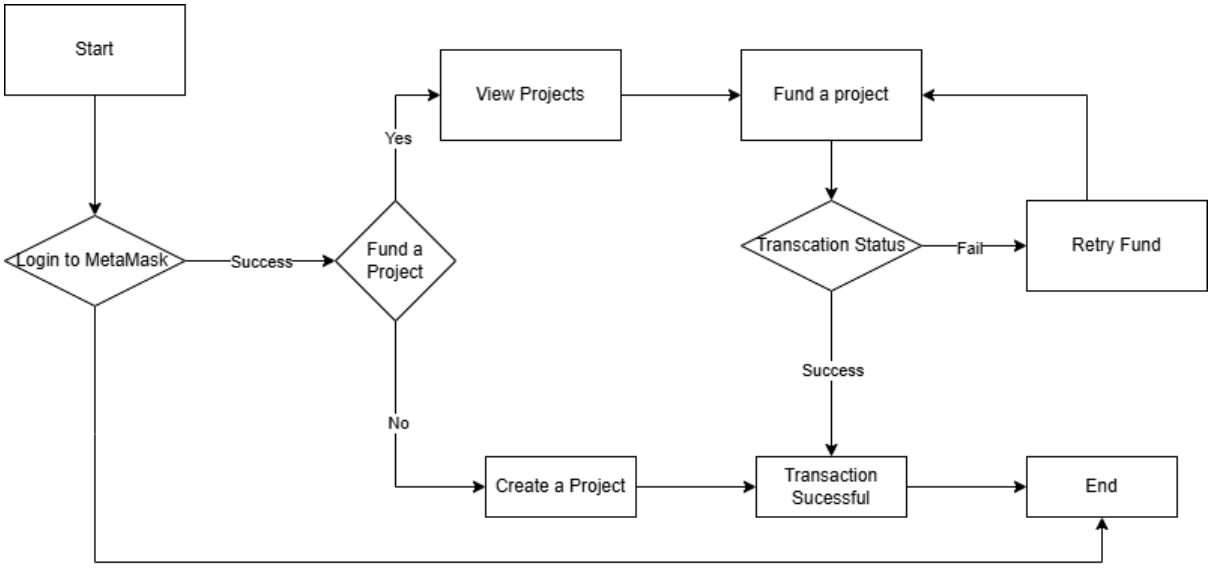

Fig 4.1.1 DATA FLOW DIAGRAM

4.2 ARCHITECTURE DIAGRAM:

An Architectural diagram is a visual representation of the structure, components, relationships, and interactions within a system or application. It shows cases how different elements works either a simple element or diagram or even complex ones.

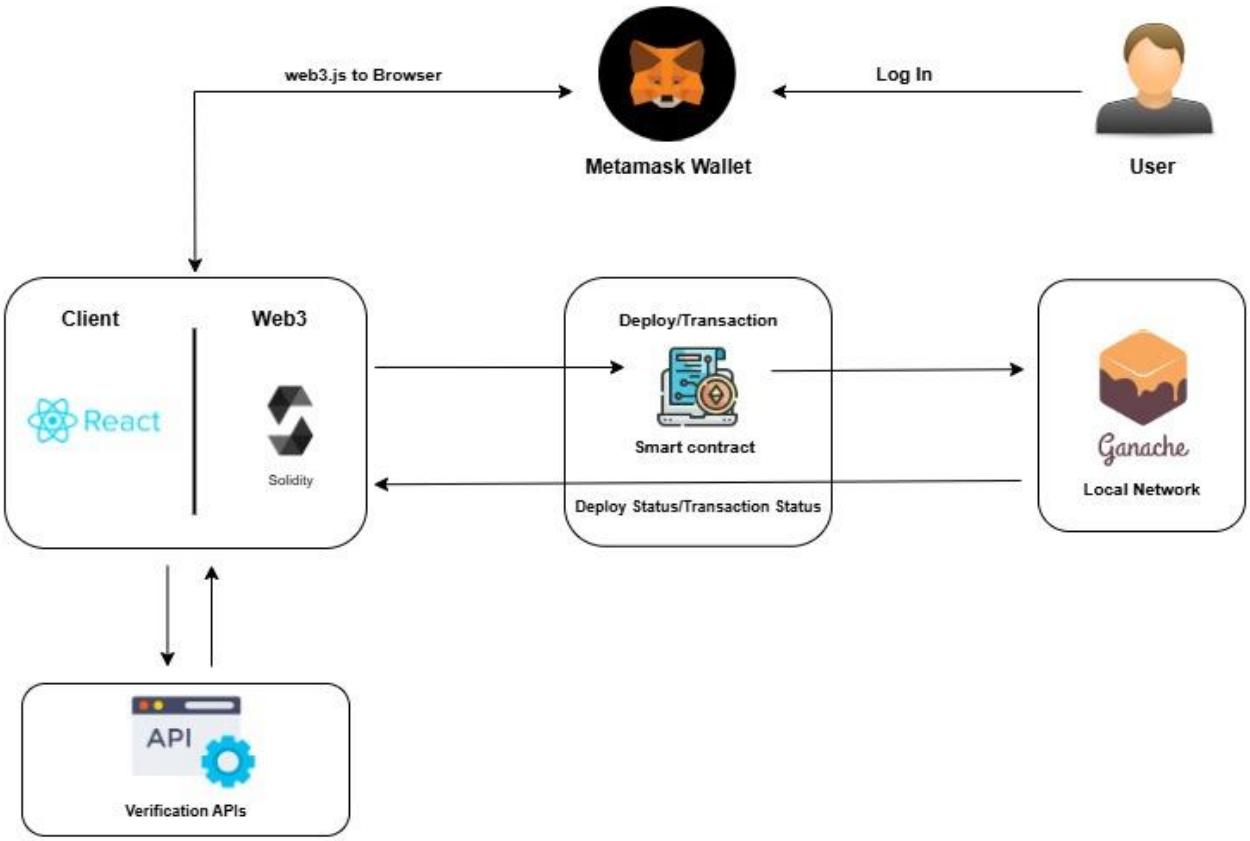

Fig 4.2.1 ARCHITECTURE DIAGRAM

4.3 UML DIAGRAMS

The Unified Modeling Language (UML) is a standardized visual modeling language used to depict and design the structure and behavior of systems, primarily in software engineering. UML diagrams provide a way to visualize system architecture, workflows, and data interactions, making complex designs easier to understand and communicate.

Types of UML Diagrams

UML diagrams are categorized into two main types:

1. **Structural Diagrams:** Illustrate the static aspects of a system, such as classes, components, and their relationships.
2. **Behavioral Diagrams:** Depict the dynamic aspects of a system, including interactions and processes.

Structural UML Diagrams

1. Class Diagram:

Purpose: Describes the structure of a system by showing its classes, attributes, methods, and the relationships among them.

Elements:

- Class: Represented by a rectangle divided into three sections (name, attributes,
- methods).
- Attributes and Methods: Specify properties and behaviors of a class.
- Relationships: Including associations, generalizations, and dependencies.
- Use Cases: Used during the design phase to model and plan system architecture.

2. Object Diagram:

Purpose: Provides a snapshot of the system at a particular time, showing instances of classes and their relationships.

Elements: Similar to class diagrams, but shows specific object instances rather than generic classes.

Use Cases: Useful for understanding real-time data and states within a system.

3. Component Diagram:

Purpose: Models the components of a system and how they interact through interfaces.

Elements:

- Component: Represents a modular part of a system (e.g., libraries, executables).
- Interface: Defines the services a component provides.
- Connector: Shows dependencies between components.
- Use Cases: Useful for understanding system deployment, organization.

4. Deployment Diagram:

Purpose: Shows the physical deployment of software artifacts on nodes (hardware).

Elements:

- Nodes: Represent physical devices or servers.
- Artifacts: Represent software components or files deployed on nodes.
- Communication Links: Show connections between nodes.
- Use Cases: Often used to plan system infrastructure, deployment, and performance optimization.

5. Package Diagram:

Purpose: Organizes the structure of a system into groups of related elements, making the overall structure easier to manage.

Elements:

- Packages: Represent a grouping of related classes, interfaces, or other packages.
- Dependencies: Show relationships between packages.
- Use Cases: Useful for managing large projects by breaking down the system into

  smaller, related parts.

Behavioral UML Diagrams

1. Use Case Diagram:

Purpose: Models the interactions between users (actors) and the system, showing system functionality.

Elements:

- Actors: Represent users or other systems that interact with the system.
- Use Cases: Represent functionalities or services the system provides.
- Relationships: Includes associations between actors and use cases, as well as
- include, extend, and generalization relationships between use cases.
- Use Cases: Useful during requirements analysis to capture system requirements from the user's perspective.

2. Sequence Diagram:

Purpose: Depicts the order of interactions (messages) between system

components or objects over time.

Elements:

- Lifeline: Represents an object's life span within a sequence.
- Activation Bar: Indicates when an object is active (performing an action).
- Messages: Show communication between objects.
- Use Cases: Valuable for detailing the flow of logic in use cases, identifying object interactions, and visualizing execution order.

3. Activity Diagram

Purpose: Models workflows or processes within the system, representing both sequential and concurrent activities.

Elements:

- Actions/Activities: Represent individual tasks or operations.
- Decision Points: Represent conditional branching within the workflow.
- Flow: Arrows indicate the direction of workflow.
- Use Cases: Useful for representing complex workflows, business processes, and decision logic.

4. State Machine Diagram

Purpose: Depicts the states of an object and the transitions that cause changes in state.

Elements:

- States: Represent different conditions of an object.
- Transitions: Show movement from one state to another based on events or conditions.
- Initial/Final States: Represent the start and end of an object's lifecycle.

- Use Cases: Useful for designing and visualizing the behavior of objects that have distinct states, such as in lifecycle management.

5. Communication Diagram (also known as Collaboration Diagram)

Purpose: Emphasizes the relationships and interactions between objects.

Elements:

- Objects: Represent system components.
- Messages: Represent interactions between objects.
- Use Cases: Useful for showing how different parts of a system communicate to achieve functionality.

6.Interaction Overview Diagram

Purpose: Combines features of activity diagrams and sequence diagrams, summarizing interactions within a system.

Elements:

- Interaction Nodes: Represent steps that involve complex interactions.
- Control Flow: Shows the sequence of interactions.
- Use Cases: Useful for visualizing large, complex workflows.

7. Timing Diagram

Purpose: Depicts the behavior of objects over a specific time period, showing changes in object states and interactions.

Elements:

- Lifelines: Represent individual objects or actors.
- State Changes: Represent the transitions in an object's state over time.
- Use Cases: Useful in real-time systems to understand how timing impacts system functionality and performance.

Importance of UML Diagrams

1. **Enhanced Communication**: UML diagrams create a shared understanding of system components and interactions, aiding communication among stakeholders, including developers, designers, and non-technical members.

2. **Efficient System Design and Development**: UML diagrams provide a blueprint for system architecture and workflows, ensuring clarity in requirements and design phases, and reducing rework.

3. **Documentation**: UML diagrams serve as visual documentation, offering a clear record of system structure and behavior throughout its lifecycle. This is valuable for maintenance and future upgrades.

4. **Error Identification and Troubleshooting**: By visualizing the system structure and workflows, UML diagrams help in identifying design issues early, leading to more robust system architecture.

5. **Scalability and Reusability**: UML diagrams allow for modular design and reusable components, enabling easier scaling and future expansions.

Practical Applications of UML Diagrams

- **Software Development**: UML diagrams guide the software development lifecycle, from requirement gathering to testing and deployment.
- **Business Process Modeling**: UML diagrams help in modeling business processes, capturing workflows and identifying optimization opportunities.
- **System Engineering**: UML diagrams assist in the engineering of complex systems, especially when coordinating different subsystems and ensuring reliable interaction.
- **Database Modeling**: Class and object diagrams are valuable for designing database schemas, defining relationships, and visualizing

database structure.

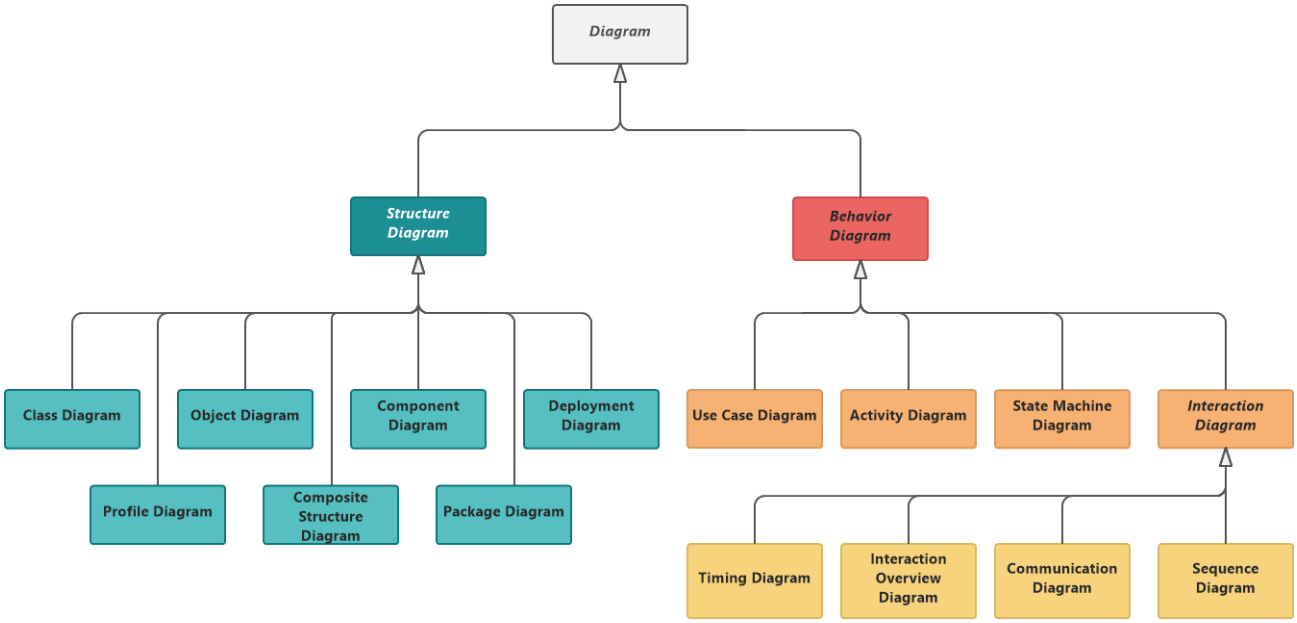

Fig 4.3.1 Types of UML Diagrams

SEQUENTIAL DIAGRAM

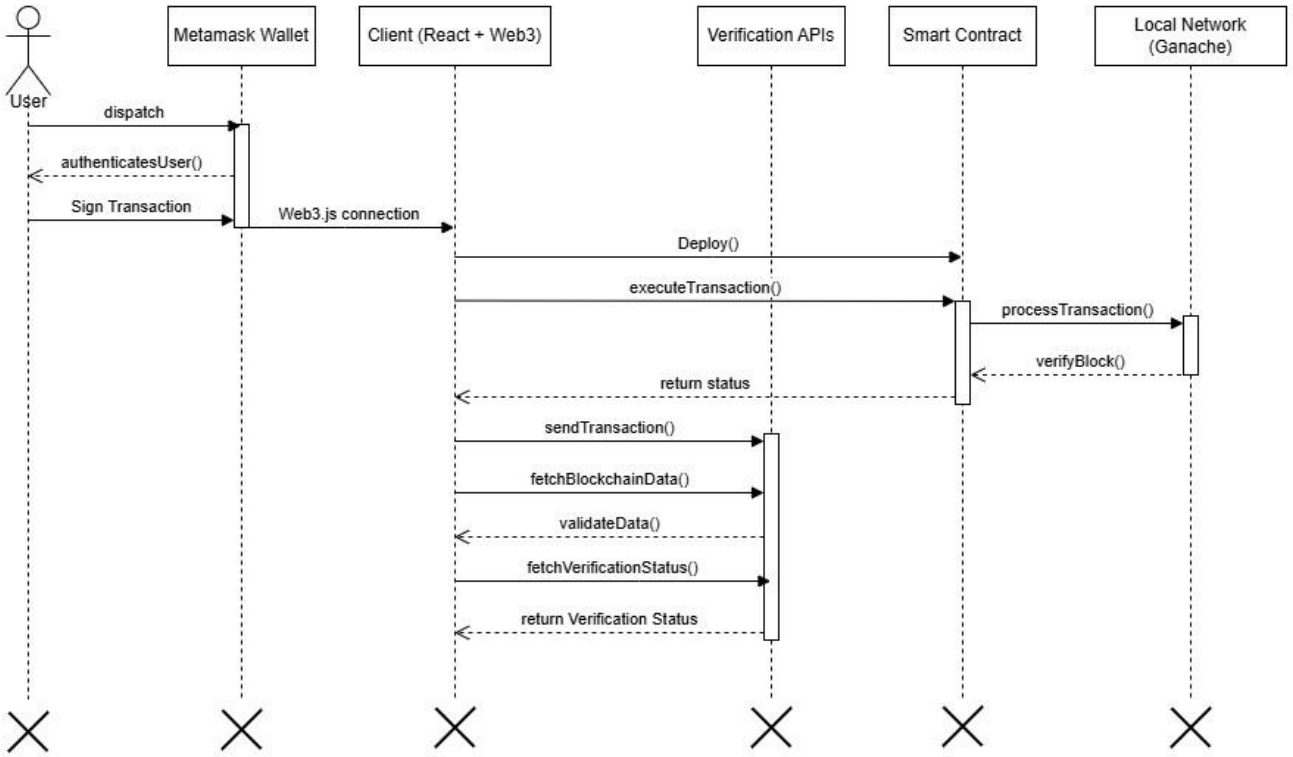

Fig 4.3.2 SEQUENTIAL DIAGRAM

USE-CASE DIAGRAM

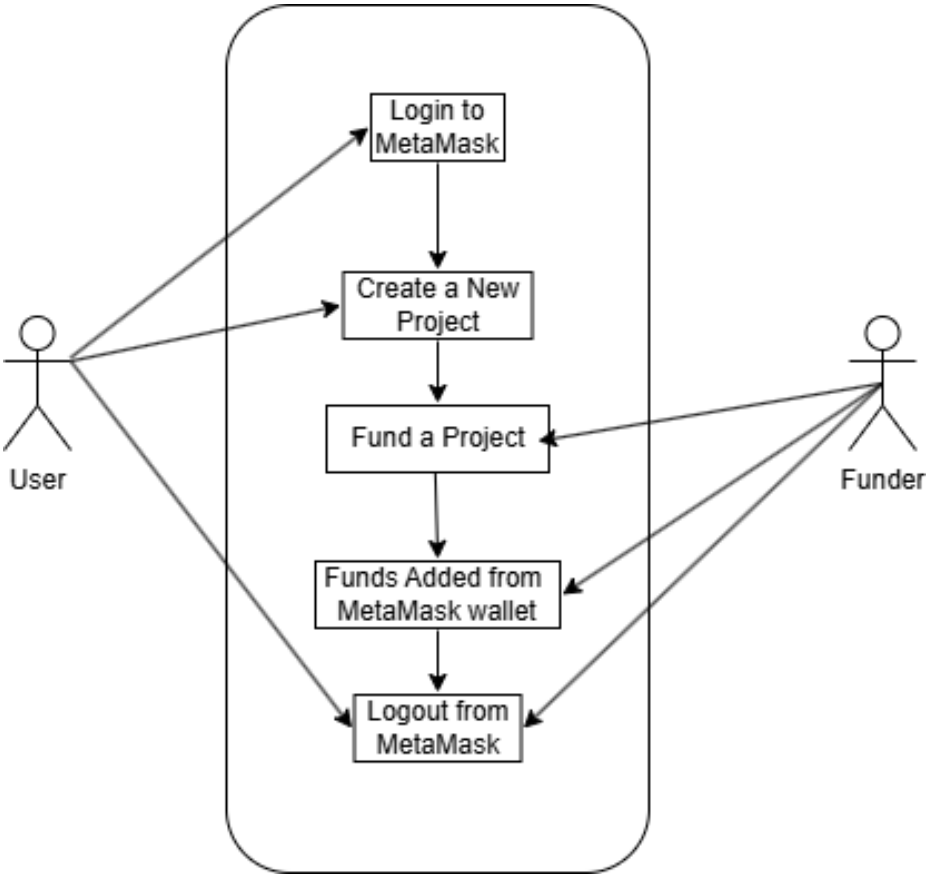

Fig 4.3.3 USECASE DIAGRAM

CLASS DIAGRAM

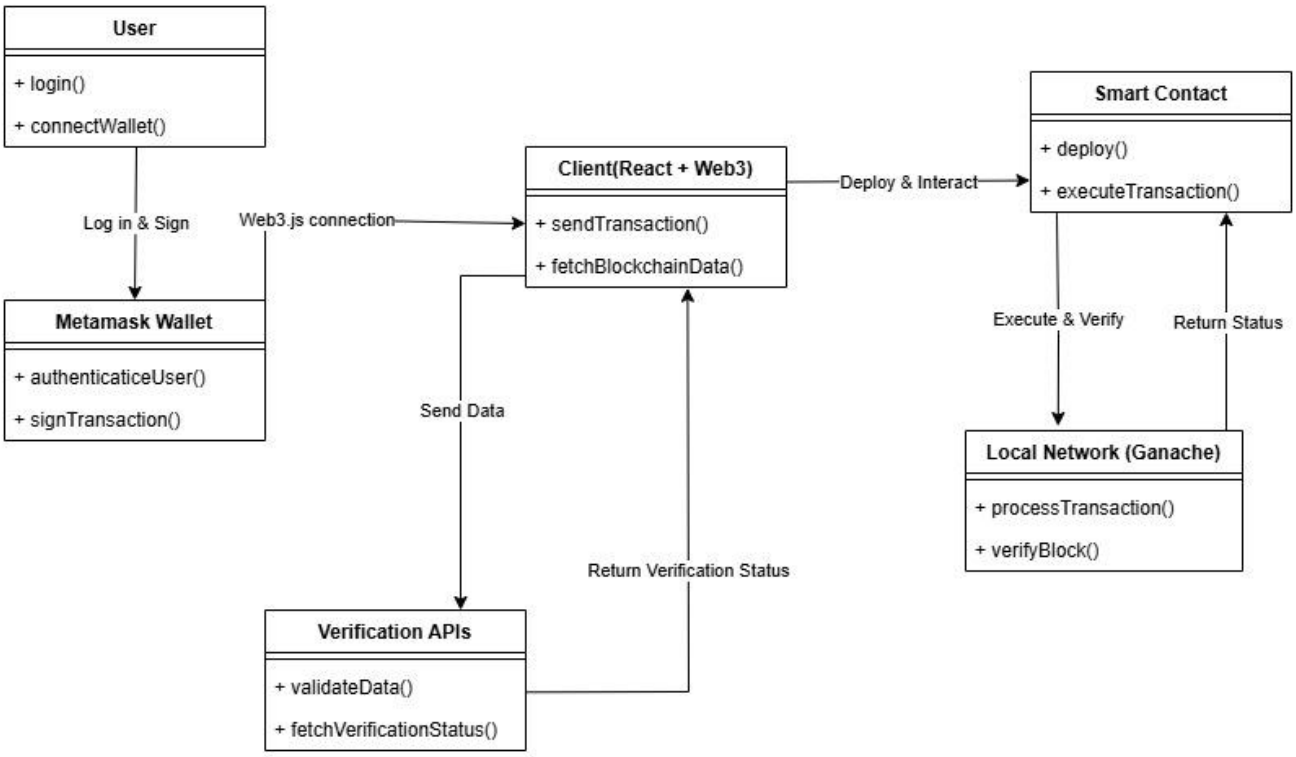

Fig 4.3.4 CLASS DIAGRAM

Chapter 5

Software Development Life Cycle

5.1 Phases of SDLC

The Software Development Life Cycle (SDLC) is a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time. The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands.

The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:

- Requirement analysis
- Planning
- Software design such as architectural design
- Software development
- Testing
- Deployment

Planning Stage:

Before we even begin with the planning stage, the best tip we can give you is to take time and acquire proper understanding of app development life cycle. The planning stage (also called the feasibility stage) is exactly what it sounds like: the phase in which developers will plan for the upcoming project. It helps to define the problem and scope of any existing systems, as well as determine the objectives for their new systems. By developing an effective outline for the upcoming development cycle, they'll theoretically catch problems before they affect development and help to secure the funding and resources they need to

make their plan happen.

Analysis Stage:

The analysis stage includes gathering all the specific details required for a new system as well as determining the first ideas for prototypes.

Developers may:

- Define any prototype system requirements
- Evaluate alternatives to existing prototypes
- Perform research and analysis to determine the needs of end-users

Furthermore, developers will often create a software requirement specification or SRS document. This includes all the specifications for software, hardware, and network requirements for the system they plan to build. This will prevent them from overdrawing funding or resources when working at the same place as other development teams.

Design Stage:

The design stage is a necessary precursor to the main developer stage.

Developers will first outline the details for the overall application, alongside specific aspects , such as its:

- User interfaces
- System interfaces
- Network and network requirements
- Databases

They'll typically turn the SRS document they created into a more logical structure that can later be implemented in a programming language.

Operation, training, and maintenance plans will all be drawn up so that developers know what they need to do throughout every stage of the cycle moving

forward.

Once complete, development managers will prepare a design document to be referenced throughout the next phases of the SDLC.

Development Stage:

The development stage is the part where developers actually write code and build the application according to the earlier design documents and outlined specifications.

This is where Static Application Security Testing or SAST tools come into play. Product program code is built per the design document specifications.

In theory, all of the prior planning and outlined should make the actual development phase relatively straight forward. Developers will follow any coding guidelines as defined by the organization and utilize different tools such as compilers, debuggers, and interpreters. Programming languages can include staples such as C++, PHP, and more. Developers will choose the right programming code to use based on the project specifications and requirements.

Testing Stage:

Building software is not the end. Now it must be tested to make sure that there aren't any bugs and that the end-user experience will not negatively be affected at any point. During the testing stage, developers will go over their software with a fine-tooth comb, noting any bugs or defects that need to be tracked, fixed, and later retested. It's important that the software overall ends up meeting the quality standards that were previously defined in the SRS document.

Depending on the skill of the developers, the complexity of the software, and the requirement for the end-user, testing can either be an extremely short phase or take a very long time. Take a look at our top 10 best practices for software testing projects for more information.

Implementation and Integration Stage:

After testing, the overall design for the software will come together. Different modules Or designs will be integrated into the primary source code through developer efforts, usually by leveraging training environments to detect further errors or defects.

The information system will be integrated into its environment and eventually installed. After passing this stage, the software is theoretically ready for market and may be provided to any end users.

Maintenance Stage:

The SDLC doesn't end when software reaches the market. Developers must now move to a maintenance mode and begin practicing any activities required to handle issues reported by end-users. Furthermore, developers are responsible for implementing any changes that the software might need after deployment.

This can include handling residual bugs that were not able to be patched before launch or resolving new issues that crop up due to user reports. Larger systems may require longer maintenance stage compared to smaller systems.

CHAPTER 6

IMPLEMENTATION

Core Smart Contract Snippet:

```
import "@openzeppelin/contracts/math/SafeMath.sol";
contract CrowdFunding {
    using SafeMath for uint256;
    Project[] private projects;
    event ProjectStarted(
        address contractAddress,
        address projectStarter,
        string projectTitle,
        string projectDesc,
        uint256 deadline,
        uint256 goalAmount
    );
    function startProject(
        string calldata title,
        string calldata description,
        uint256 durationInDays,
        uint256 amountToRaise
    ) external {
        uint256 raiseUntil = now.add(durationInDays.mul(1 days));
        Project newProject = new Project(
            msg.sender,
            title,
            description,
            raiseUntil,
            amountToRaise
```

```
        );
contract Project {
    using SafeMath for uint256;
    enum State {
        Fundraising,
        Expired,
        Successful
    }

    address payable public creator;
    uint256 public amountGoal;
    uint256 public completeAt;
    uint256 public currentBalance;
    uint256 public raiseBy;
    string public title;
    string public description;
    State public state = State.Fundraising;
    mapping(address => uint256) public contributions;

    event FundReceived(
        address contributor,
        uint256 amount,
        uint256 currentTotal
    );
    constructor(
        address payable projectStarter,
        string memory projectTitle,
        string memory projectDesc,
        uint256 fundRaisingDeadline,
        uint256 goalAmount
```

```
) public {
    creator = projectStarter;
    title = projectTitle;
    description = projectDesc;
    amountGoal = goalAmount;
    raiseBy = fundRaisingDeadline;
    currentBalance = 0;
}
function contribute() external payable inState(State.Fundraising) {
    require(msg.sender != creator);
    contributions[msg.sender] = contributions[msg.sender].add(msg.value);
    currentBalance = currentBalance.add(msg.value);
    emit FundReceived(msg.sender, msg.value, currentBalance);
    checkIfFundingCompleteOrExpired();
}
function checkIfFundingCompleteOrExpired() public {
    if (currentBalance >= amountGoal) {
        state = State.Successful;
        payOut();
    } else if (now > raiseBy) {
        state = State.Expired;
    }
    completeAt = now;
}
function payOut() internal inState(State.Successful) returns (bool) {
    uint256 totalRaised = currentBalance;
    currentBalance = 0;
    if (creator.send(totalRaised)) {
        emit CreatorPaid(creator);
        return true;
```

```
        } else {
            currentBalance = totalRaised;
            state = State.Successful;
        }
        return false;
    }
    function getInfo()
        public
        returns (
            address payable projectStarter,
            string memory projectTitle,
            string memory projectDesc,
            uint256 deadline,
            State currentState,
            uint256 currentAmount,
            uint256 goalAmount
        )
    {
        projectStarter = creator;
        projectTitle = title;
        projectDesc = description;
        deadline = raiseBy;
        currentState = state;
        currentAmount = currentBalance;
        goalAmount = amountGoal;
    }
}
```

Main UI Javascript Snippet :

```
import React, { useState, useEffect } from "react";
import Web3 from "web3";
```

```
import CrowdFunding from "./contracts/CrowdFunding.json";
import Project from "./contracts/Project.json"
import { BrowserRouter as Router, Redirect, Route, Switch } from 'react-router-dom'

const App = () => {
  const [web3, setWeb3] = useState(new Web3(window.ethereum))
  const [accounts, setAccounts] = useState([])
  const [contract, setContract] = useState(null)
  const [projects, setProjects] = useState([])
  const [loading, setLoading] = useState(true)

  console.log(projects)
const connect = async () => {
      const pk = localStorage.getItem('cacheKey')
      const nId = localStorage.getItem('cacheNID')
      try {
        if (pk && nId) {
          const web3 = new Web3(window.ethereum)
          console.log(pk)
          const accounts = !pk || pk === undefined ? [] : [pk];
          const networkId = nId;
          console.log(networkId)

          const deployedNetwork = CrowdFunding.networks[networkId];
          const instance = new web3.eth.Contract(
            CrowdFunding.abi,
            deployedNetwork && deployedNetwork.address,
          )

          localStorage.setItem("cacheKey", accounts[0]);

          setWeb3(web3)
          setAccounts(accounts)
          setContract(instance)
          setLoading(false)
        }
        else {
          setLoading(false)
        }
      }
      catch (error) {
        setLoading(false)
        console.error(error);
      }
    }
    connect()
  }, [])
```

```
const crowdfundProject = (address) => {
   const instance = new web3.eth.Contract(Project.abi, address);
   return instance
 }

const getAllProjects = () => {

   contract.methods.returnAllProjects().call().then((pr) => {
     pr.forEach(async (projectAddress) => {
       const projectInst = crowdfundProject(projectAddress);
       await projectInst.methods.getInfo().call()
         .then((projectData) => {
           const projectInfo = projectData;
           projectInfo.isLoading = false;
           projectInfo.contract = projectInst;
           if (!isExpired(projectInfo) && !isComplete(projectInfo)) {
             setProjects(p => [...p, projectInfo])
           }

         })
     });
   })

<AppContext.Provider value={values}>
       <div className="App">
         <Router>
           <Switch>
             {accounts.length < 1 && <Redirect from="/all" to="/login" />}
             {accounts.length < 1 && <Redirect from="/create" to="/login" />}
             {accounts.length < 1 && <Redirect from="/projects/my" to="/login" />}

             <Route path="/" exact component={Page} />
             <Route path="/all" component={AllProjects} />
             <Route path="/create" component={CreateProject} />
             <Route path="/projects/my" component={MyProjects} />
             <Route path="/projects/funded" component={FundedProjects} />
             <Route path="/login" exact component={Login} />
           </Switch>
         </Router>
       </div>
     </AppContext.Provider>
   );
```

New Project / Campaign Snippet:

```
import React, { useContext, useState } from 'react'
import "./style.css"
import { AppContext } from "../../utils/AppContext"
import { Link } from 'react-router-dom'

const CreateProject = (props) => {

    const { web3, accounts, contract, crowdfundProject } = useContext(AppContext)

    const [project, setProject] = useState({
        title: '',
        description: '',
        duration: null,
        amountGoal: null
    })

const create = async (e) => {
        e.preventDefault()
        if (project.title !== '' &&
            project.description !== '' &&
            project.duration &&
            project.amountGoal) {


            contract.methods.startProject(
                project.title,
                project.description,
                project.duration,
                web3.utils.toWei(project.amountGoal, 'ether')
            ).send({ from: accounts[0] })
                .then(res => {
                    const projectInfo = res.events.ProjectStarted.returnValues;
                    projectInfo.isLoading = false;
                    projectInfo.currentState = 0;
                    projectInfo.contract = crowdfundProject(projectInfo.contractAddress);
                    window.location.href = "/projects/my"
                })
        }
        else {
            alert("Fill all the fields.")
        }
    }
```

```
<form onSubmit={create}>
    <h2 className="heading">Create Project</h2>
    <div className="ip-fields">
    <input placeholder="Title"
     value={project.title}
     onChange={(e) => setProject({ ...project, title: e.target.value })} />

     <input placeholder="Duration (days)"
     value={project.duration}
     onChange={(e) => setProject({ ...project, duration: e.target.value })} />
    </div>

    <div className="ip-fields">
    <input placeholder="Goal Amount (ETH)"
           value={project.amountGoal}
            onChange={(e) => setProject({ ...project, amountGoal: e.target.value })} />

      <textarea placeholder="Description"
       value={project.description}
       onChange={(e) => setProject({ ...project, description: e.target.value })} />
     </div>
     <button>Create</button>
     </form>

export default CreateProject.
```

CHAPTER 7
TESTING

Introduction

7.1 Software Testing:

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

White Box Testing:

It is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

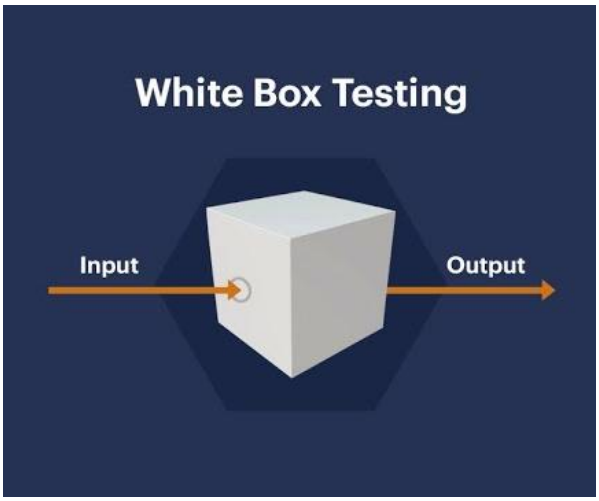

Fig 7.1.1 White box testing

Black Box Testing:

It is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

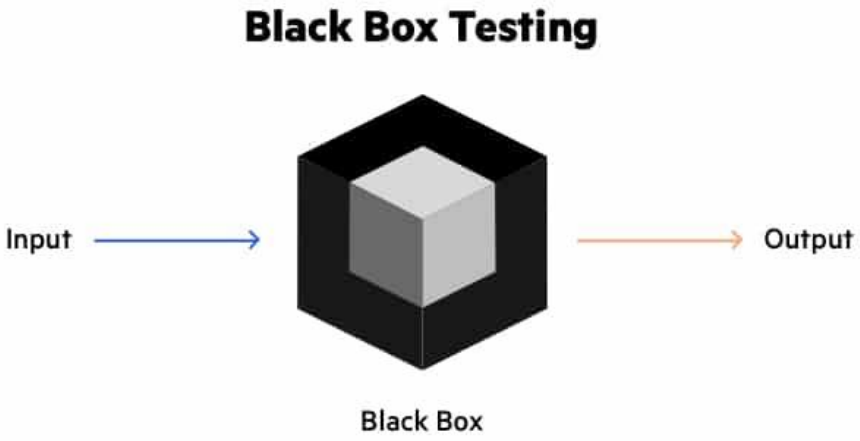

Fig.7.1.2 Black box testing

Unit Testing:

A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system. In most programming languages, that is a function, a subroutine, a method or property.

The isolated part of the definition is important. In his book "Working Effectively with Legacy Code", author Michael Feathers states that such tests are not unit tests when they rely on external systems: “If it talks to the database, it talks across Network, it touches the file system, it requires system configuration, or it can't be run at the same time as any other test.

A unit can be almost anything you want it to be -- a line of code, a method, or a class. Generally though, smaller is better. Smaller tests give you a much more Granular view of how your code is performing. There is also the practical aspect that when you test very small units, your tests can be run fast; like a thousand tests in a second fast.

During this testing, each module is tested individually and the module interfaces are verified for consistency with design specifications. All important processing paths are tested for the expected results. All error-handling paths are also tested.

In unit testing different modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goal to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself.

Integration Testing:

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group.

The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units. Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as integration testing.

The following are the types of Integration Testing:

i) Top-Down Integration:

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth-first or breadth-first manner.

In this method, the software is tested from the main module, and individual stubs are replaced when the test proceeds downwards.

ii) Bottom-up Integration:

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated.

The bottom-up approach tests each module individually and then each module is module is integrated with a main module and tested for functionality.

7.2 Test Outputs

Table 7.2.1 - Sample Test Cases

Project Title	Trust based Decentralized Crowdfunding using Ethereum Blockchain		
Module	Entire dApp		
Date of Review		Review/Version No.	1
Tester Name	Kasam Rohit Reddy	Reviewer Name:	Jangam Swathi

Test No	Test Objective	Test Steps	Expected Results	System Generated Results	Test Impact (High/Medium)	Remarks
TC_1	Meta Login	Log into the metamask account.	Logs into the relevant account and show the projects created.	Logged into the MetaMask user and showed all projects.	Pass	_____
TC_2	Create New Projects	Creating new projects to fund.	Successfully created projects with fund option.	Successfully created projects with required funds.	Pass	_____
TC_3	Delete Project	Delete a created project.	Deletes a project from the list of projects.	Didn't show the option to delete.	Fail	Could have been caused due to unstable module integration.
TC_4	Fund a project	Connect to the metamask wallet and ask for verification.	Connects to the metamask wallet and funds the project	Connected to the metamask wallet & Transferred the funds.	Pass	_____
TC_5	Switch User	Log out of the account and connect to the other users.	Logs out of the account and open another user page.	Logs out of the account and opens another user page.	Pass	_____

CHAPTER 8
OUTPUT SCREEN

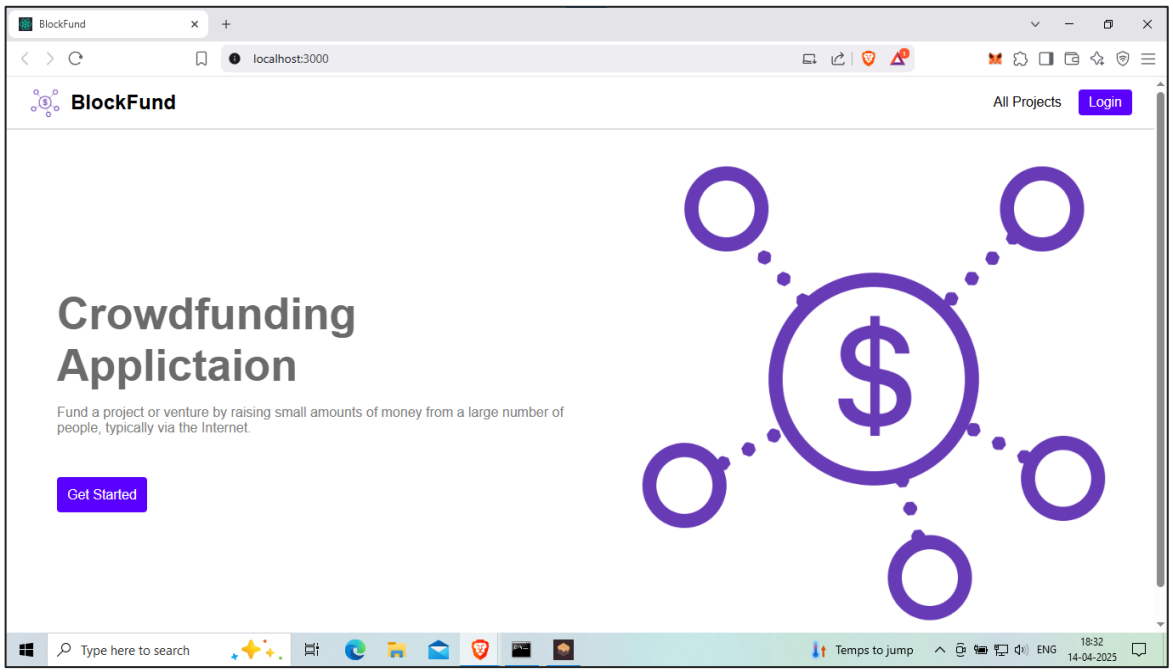

Fig 8.1 Home Page of CrowdFund

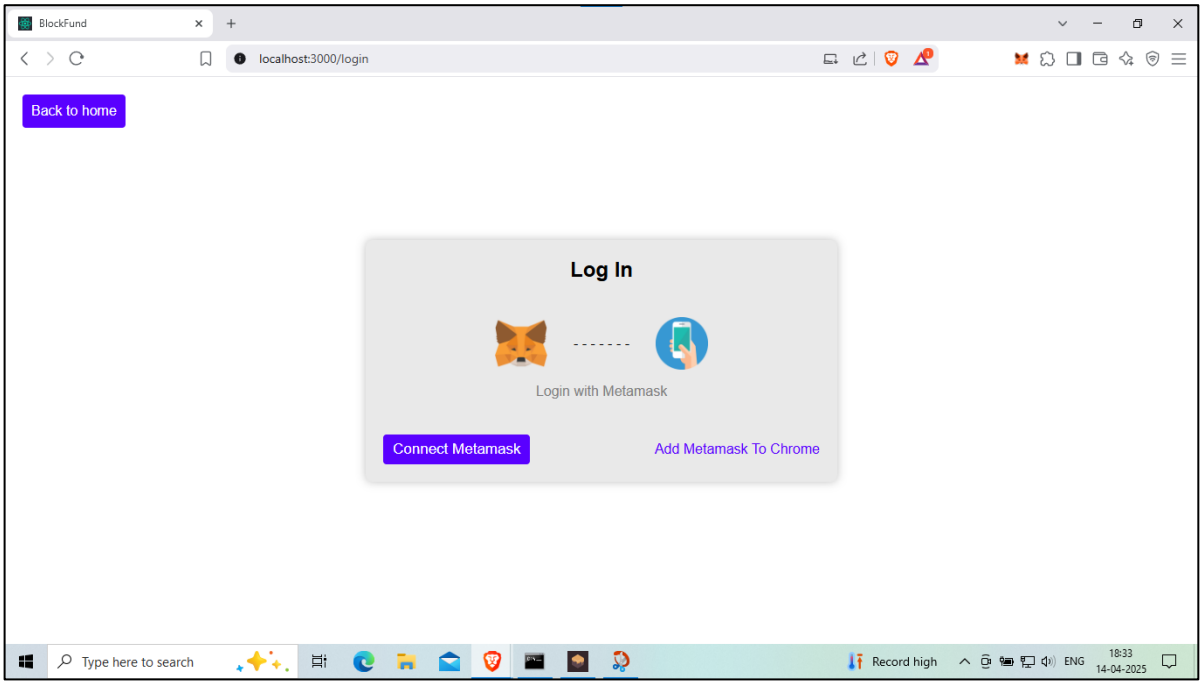

Fig 8.2 Login page of CrowdFund

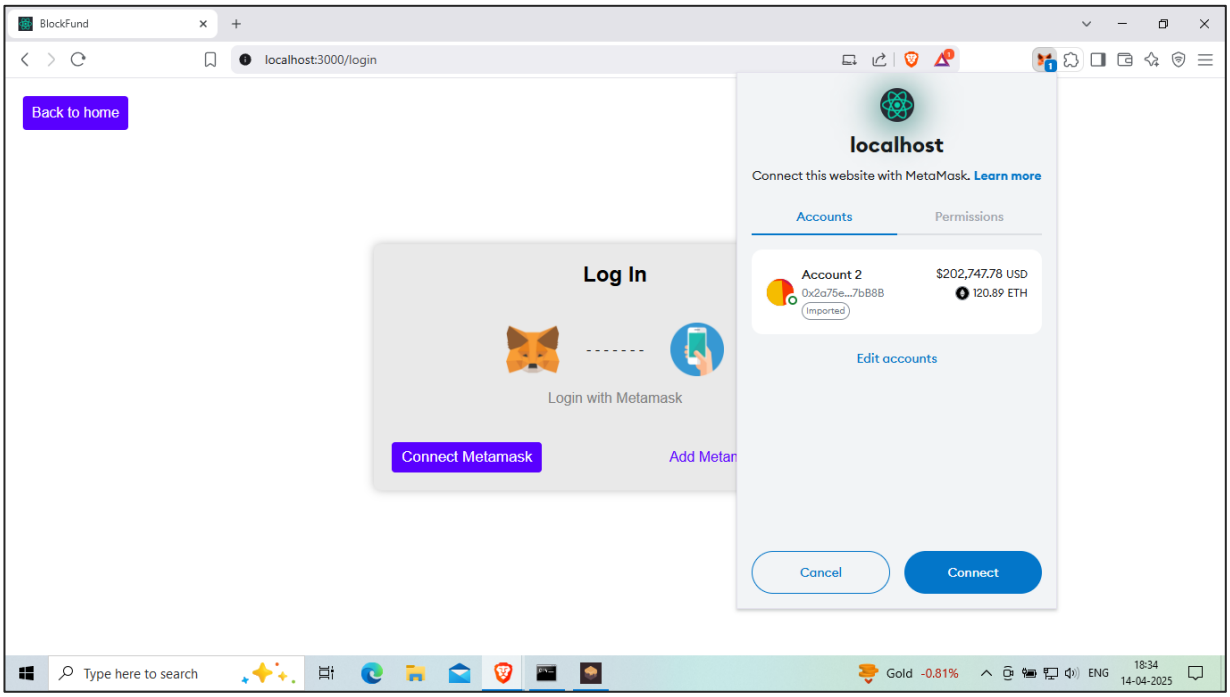

Fig 8.3 MetaMask Login Page

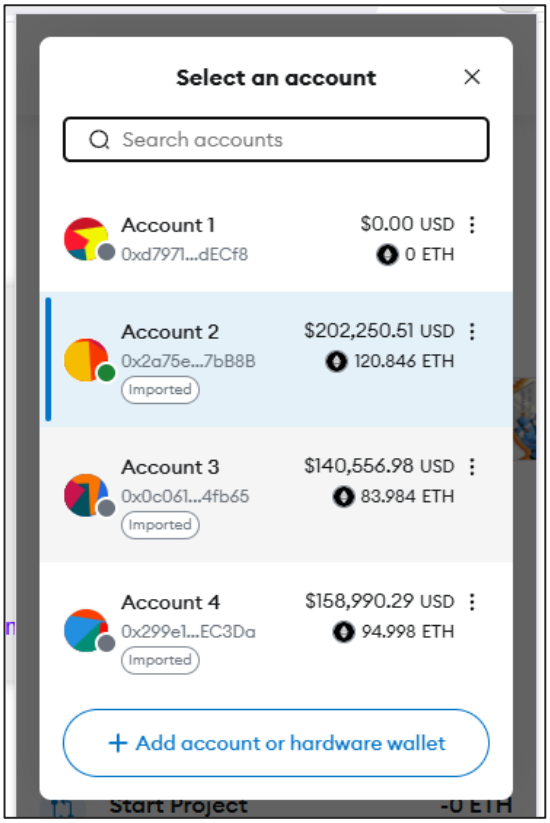

Fig 8.4 Diff. Accounts in MetaMask

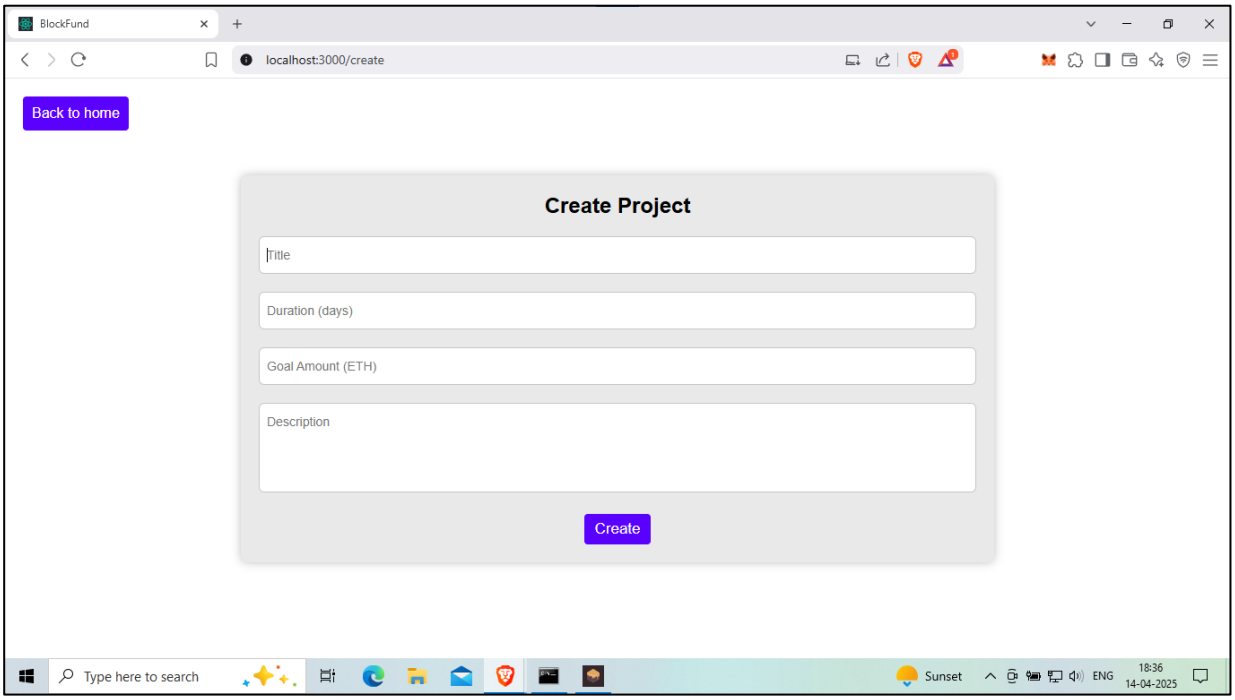

Fig 8.5 Create Project Page

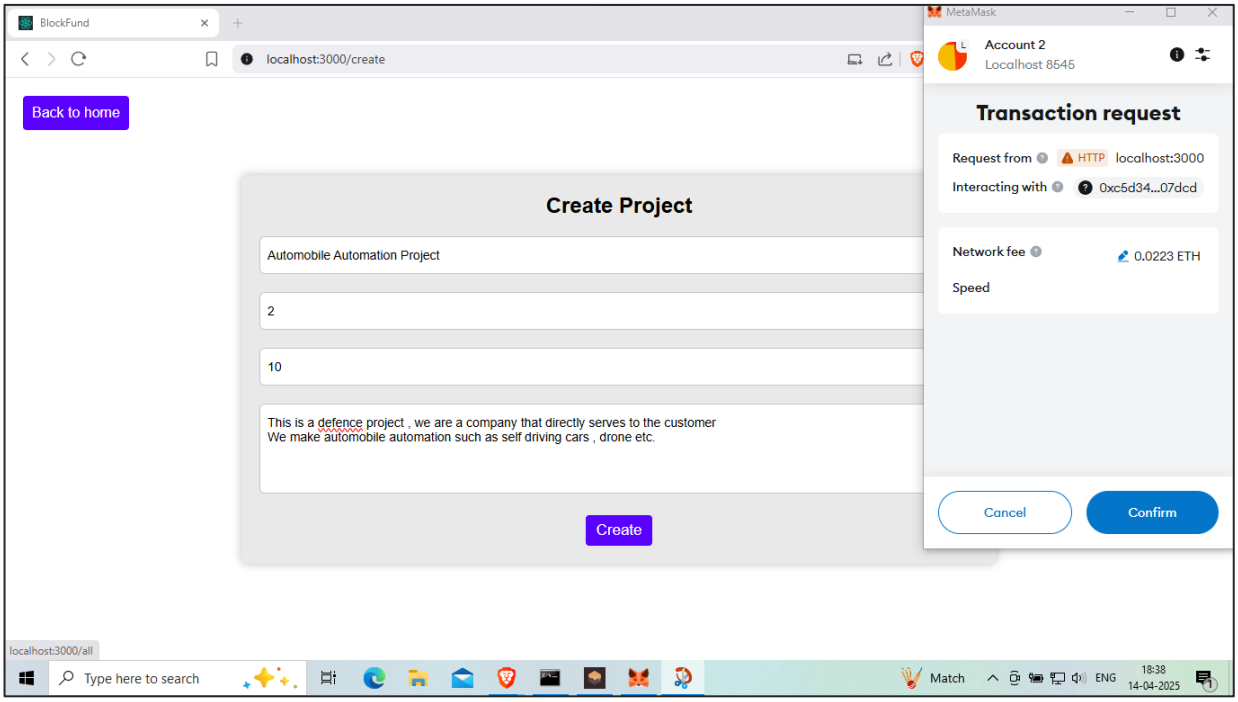

Fig 8.6 Create Project - Confirmation

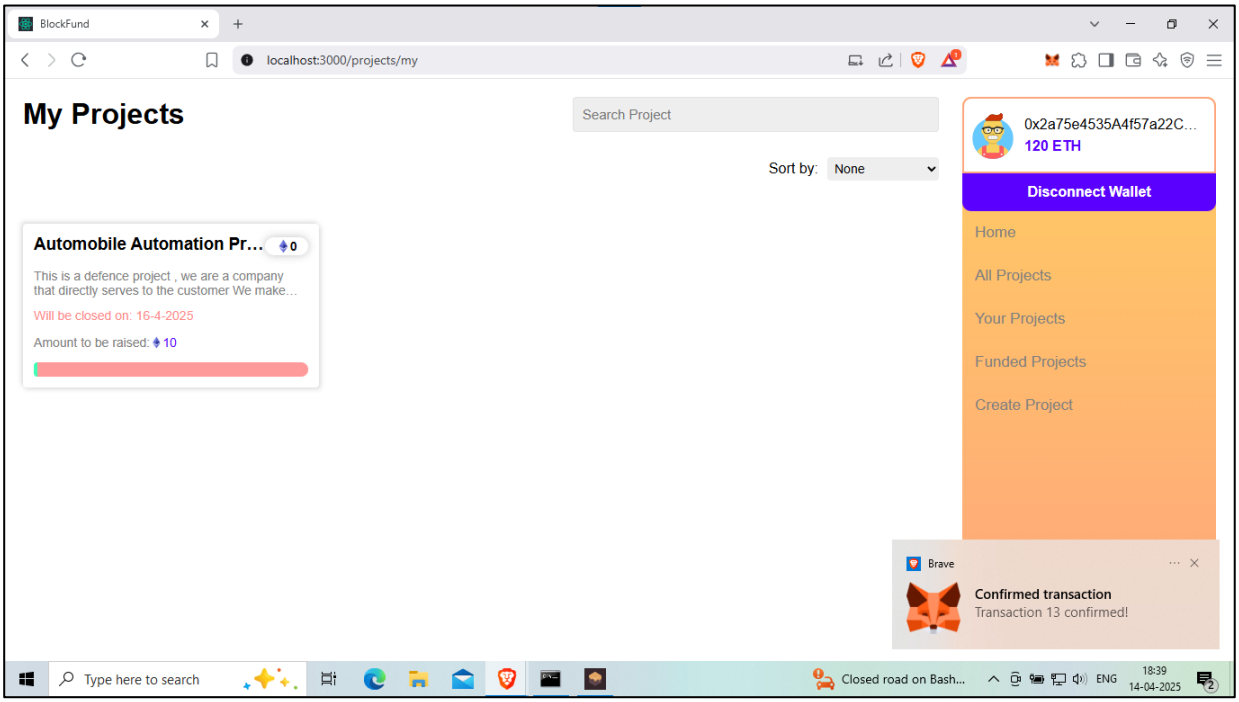

Fig 8.7 Project Created Successfuly

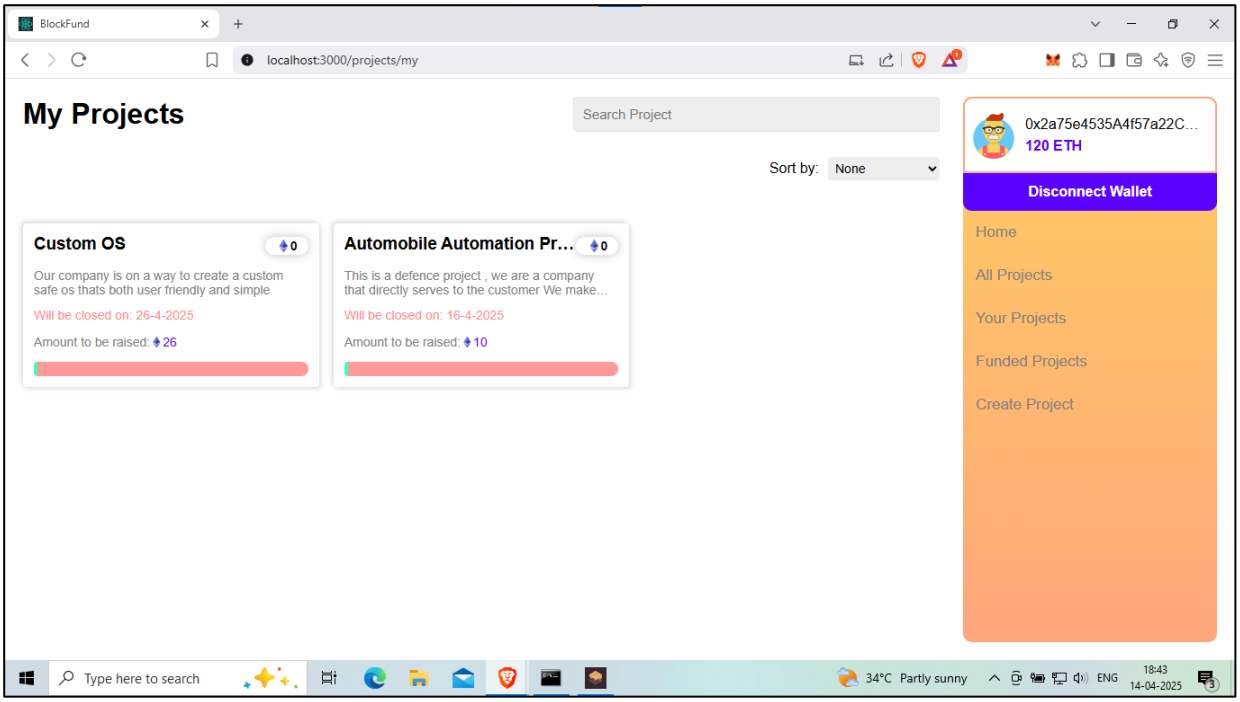

Fig 8.8 2nd Project Created

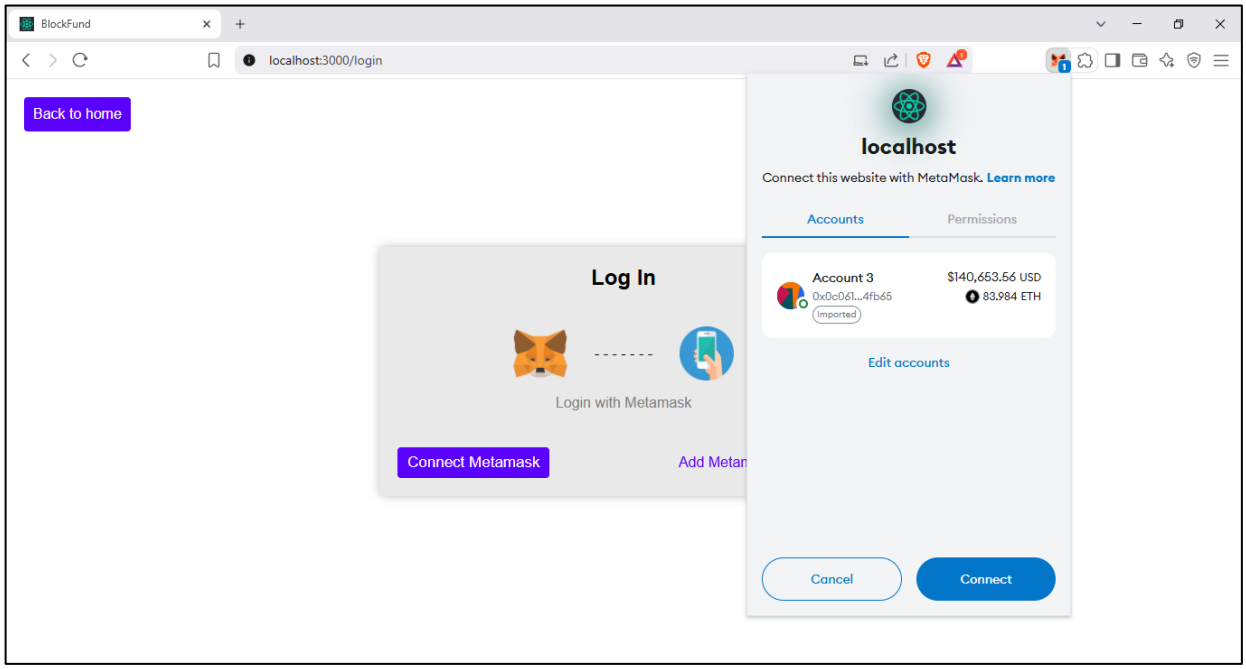

Fig 8.9 Switching to Another Account

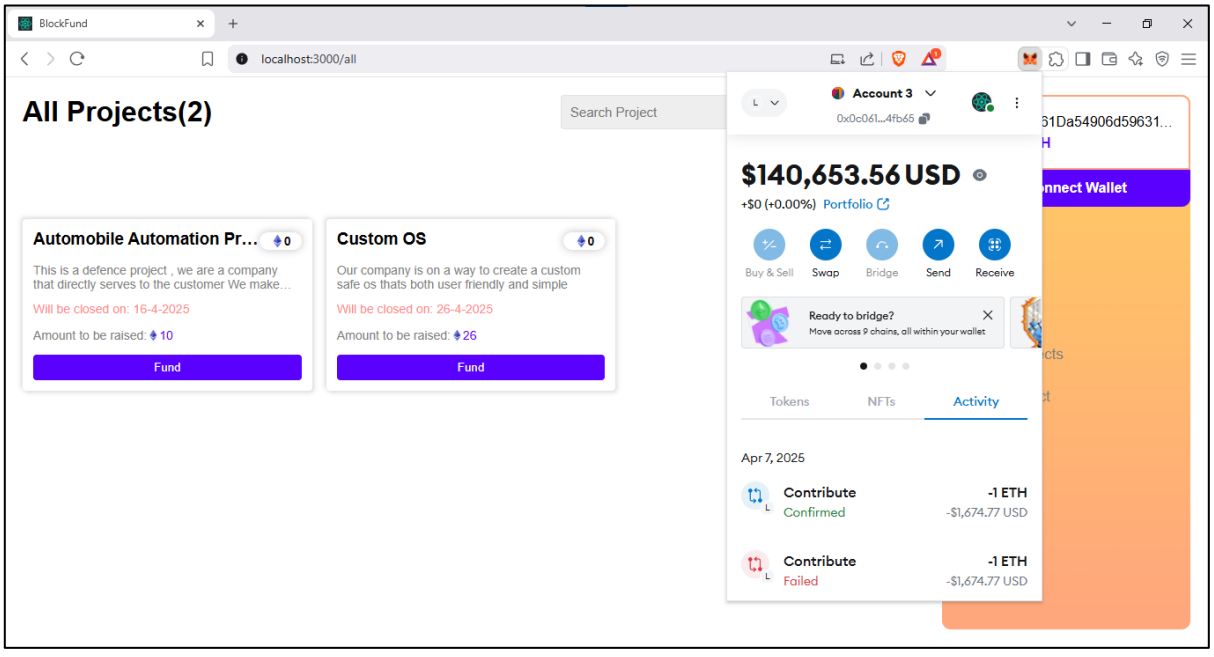

Fig 8.10 All Projects Page

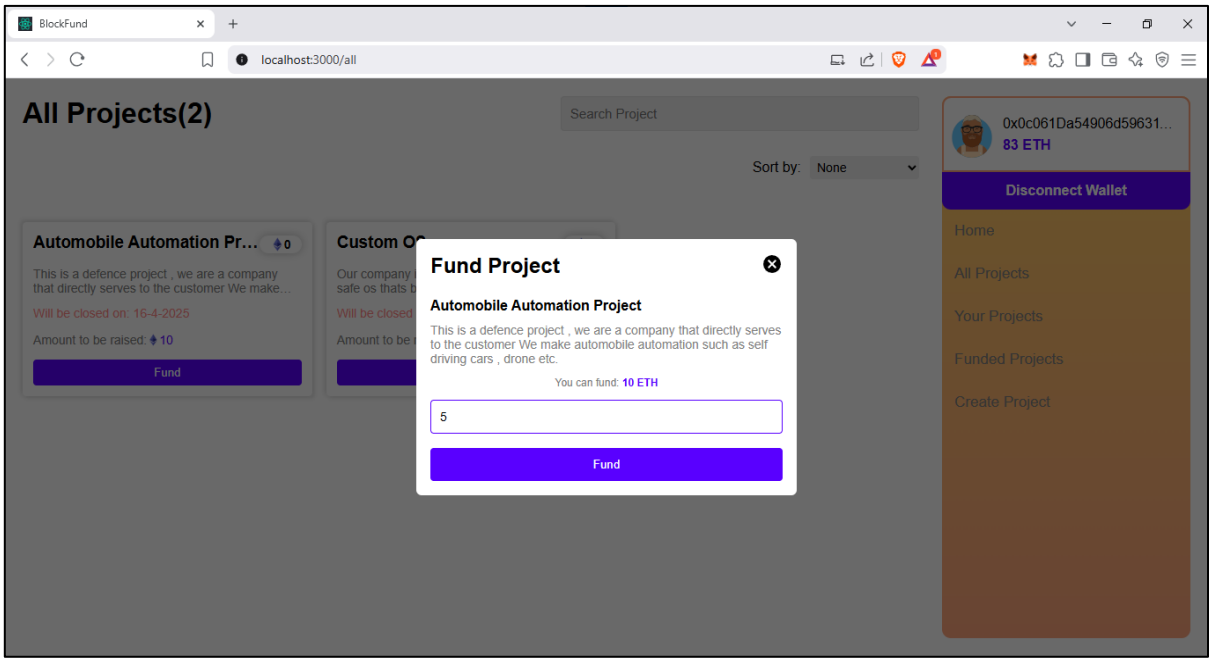

Fig 8.11 Fund Project Dialog

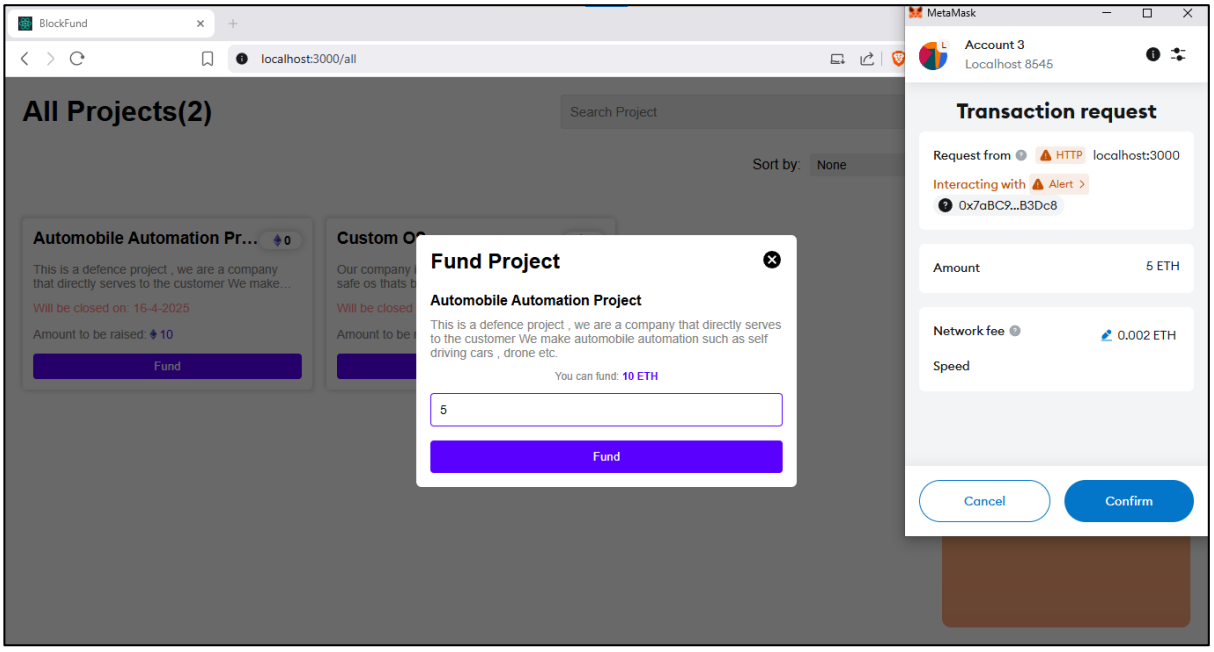

Fig 8.12 Funds Transfer Request

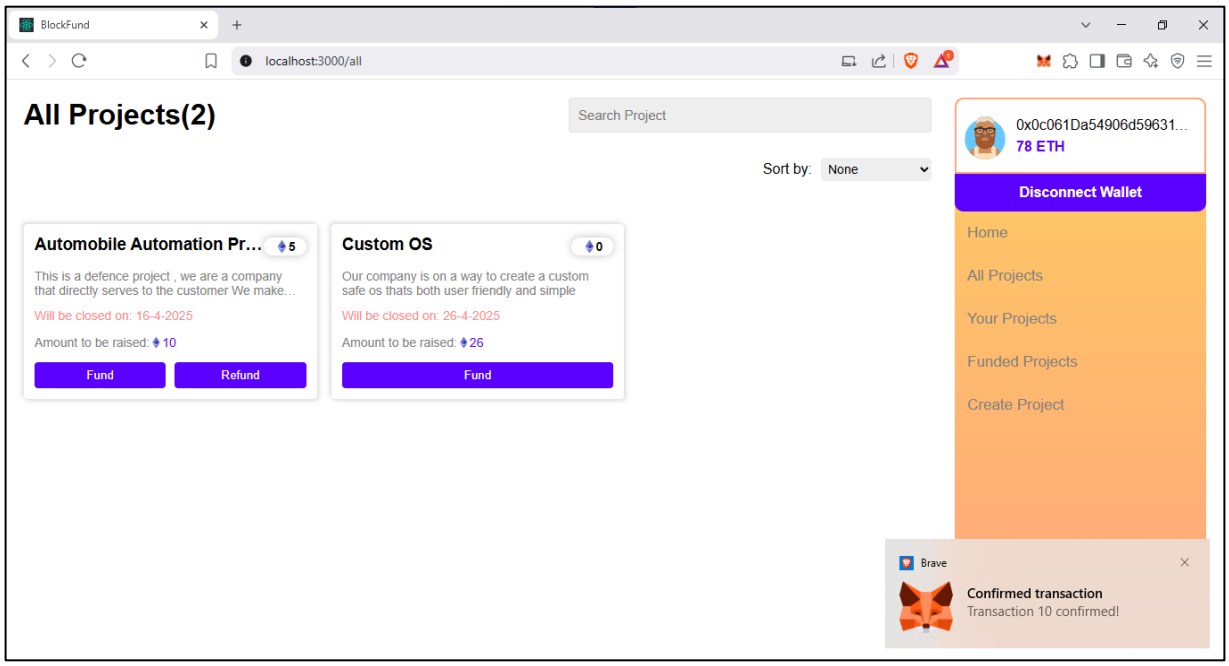

Fig 8.13 Funds Tranfer - Confirmed

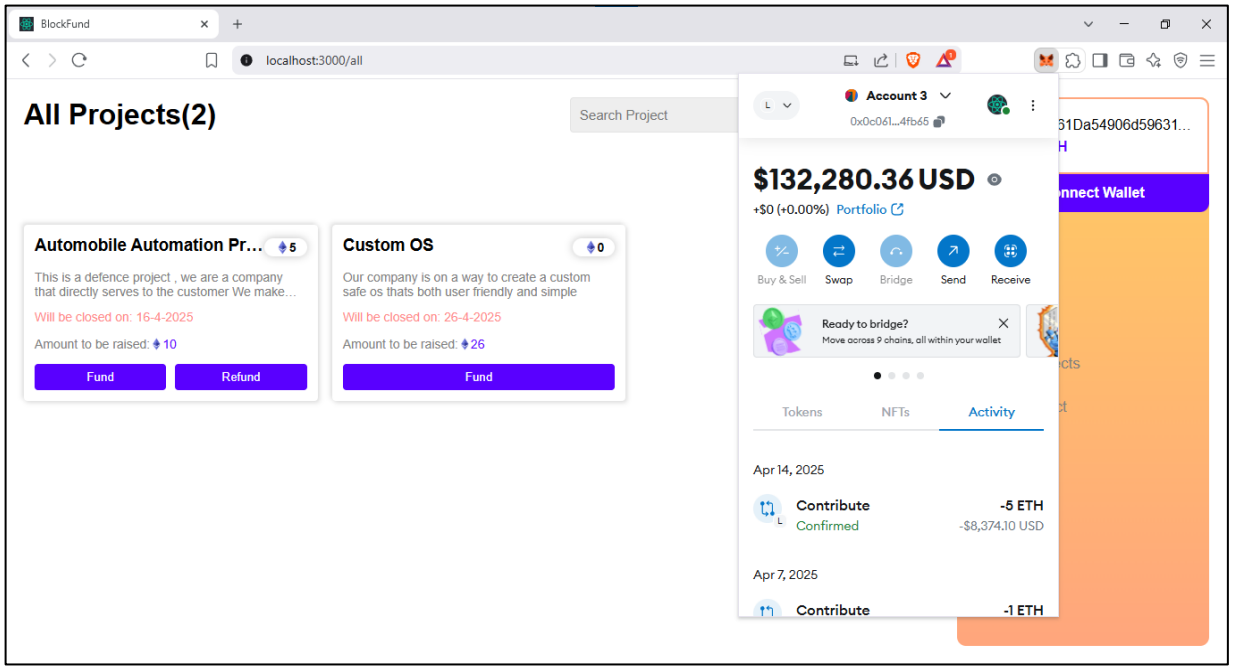

Fig 8.14 Funds Transfer - History

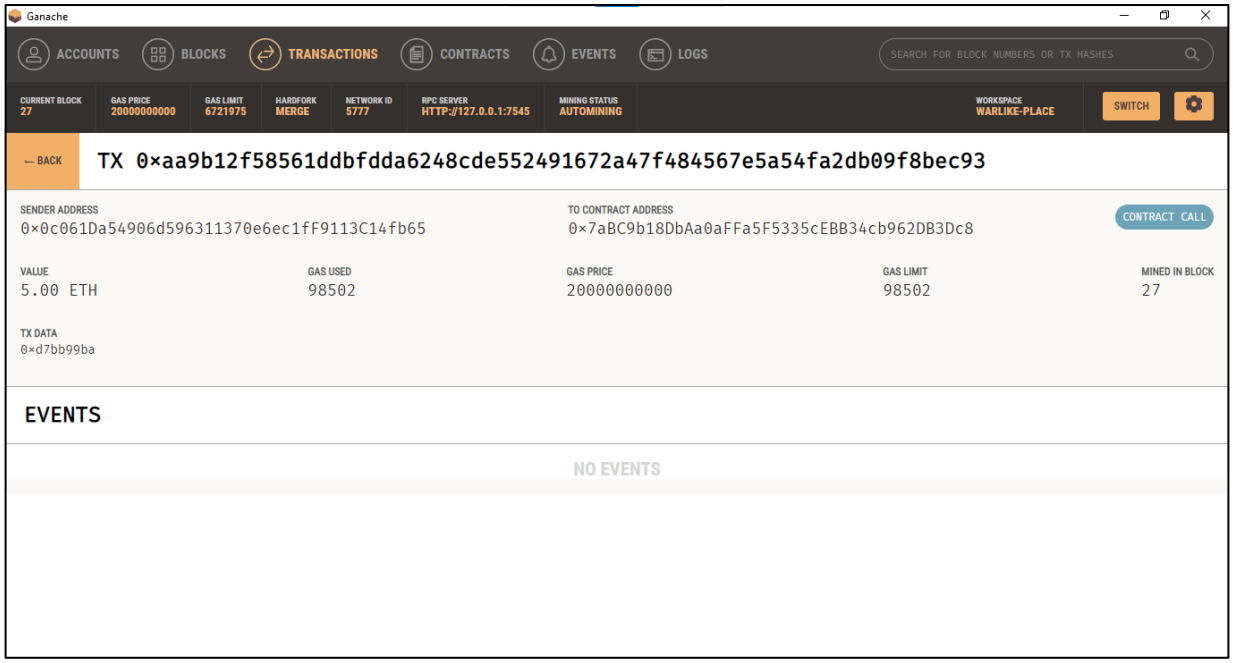

Fig 8.15 Ganache - Transaction History

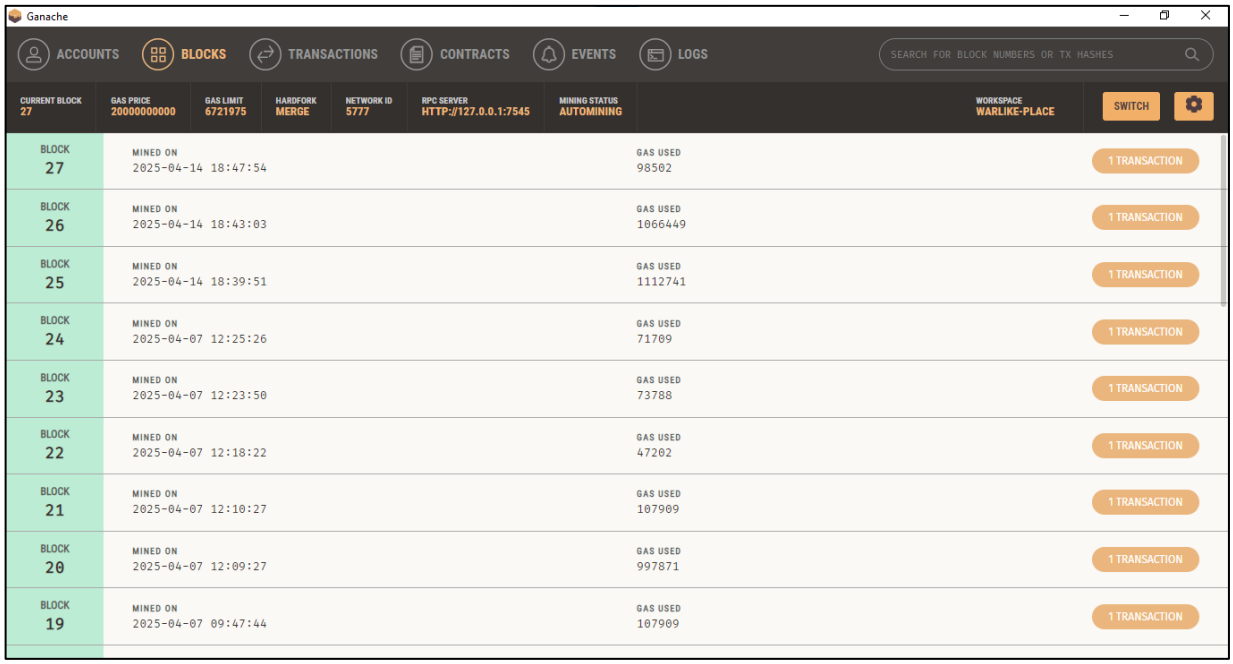

Fig 8.16 Ganache - Transaction Added to Block

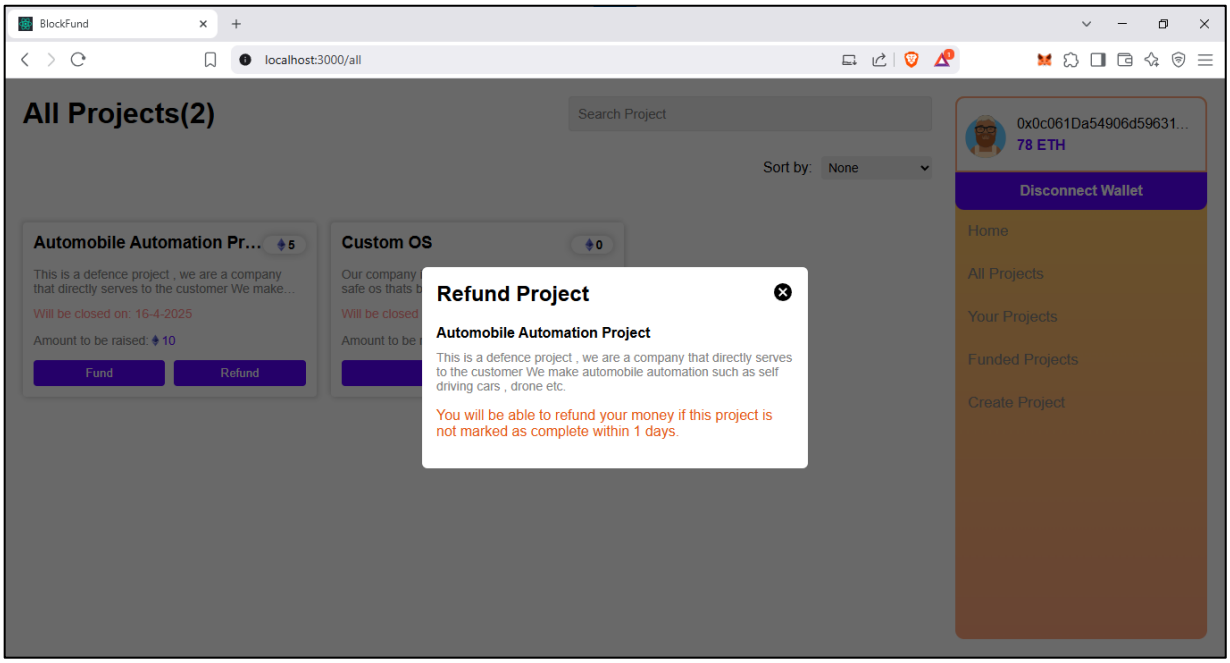

Fig 8.17 Refund Page

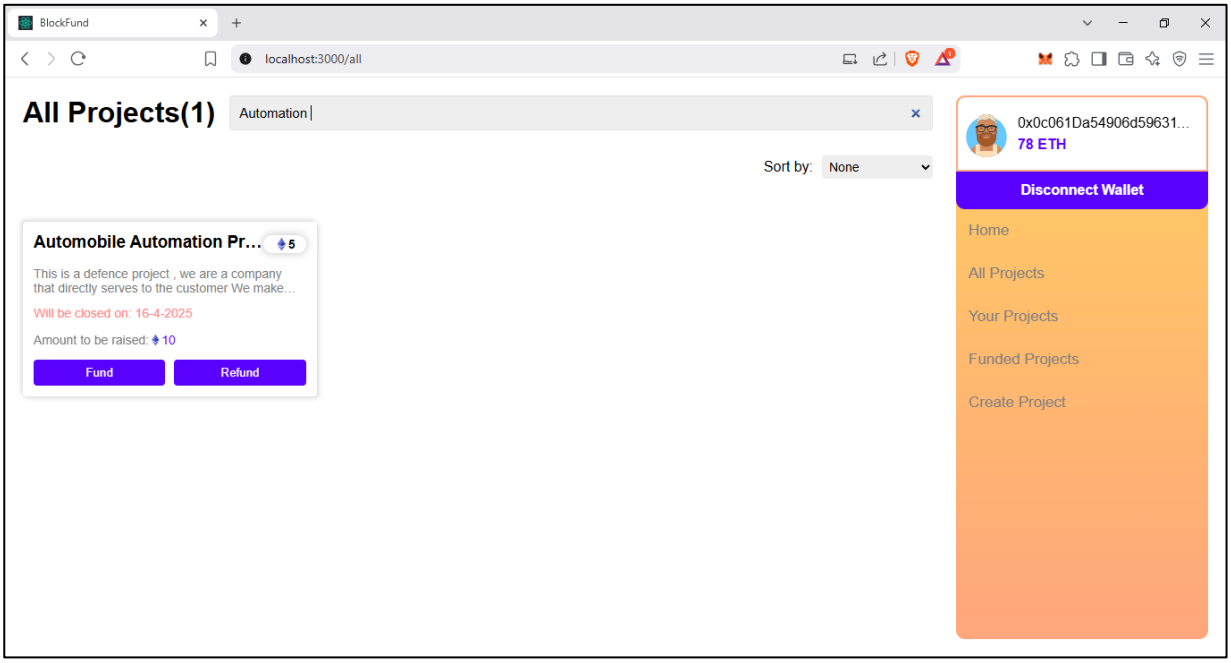

Fig 8.18 Search Function

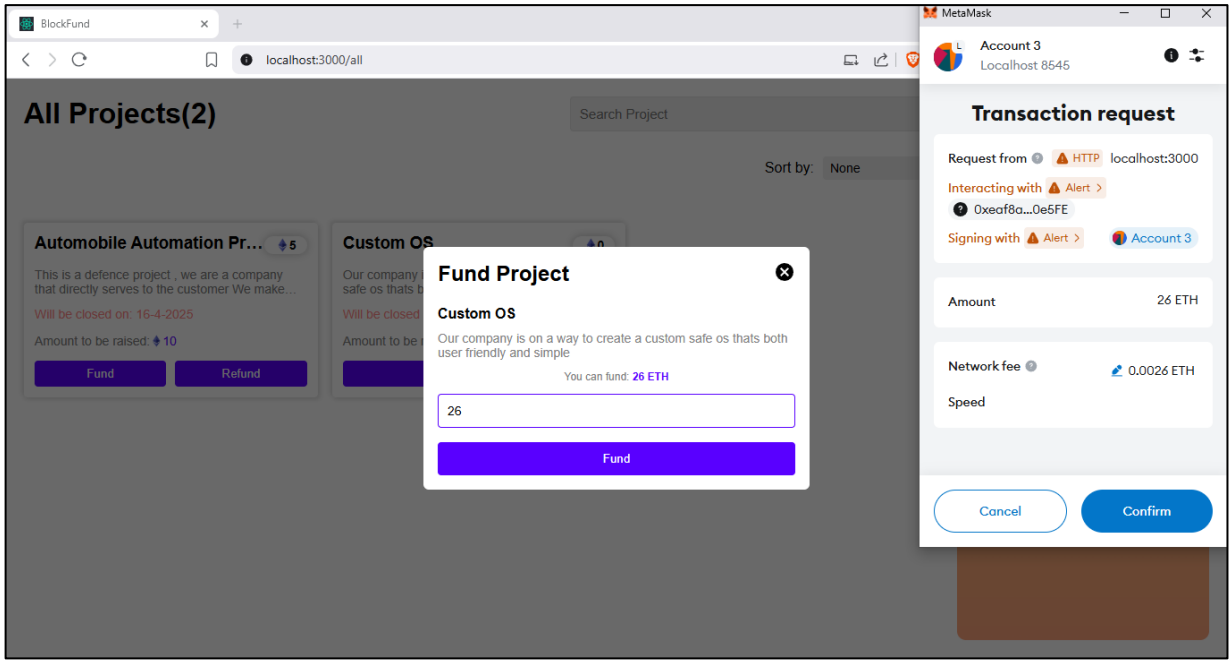

Fig 8.19 Fund Project

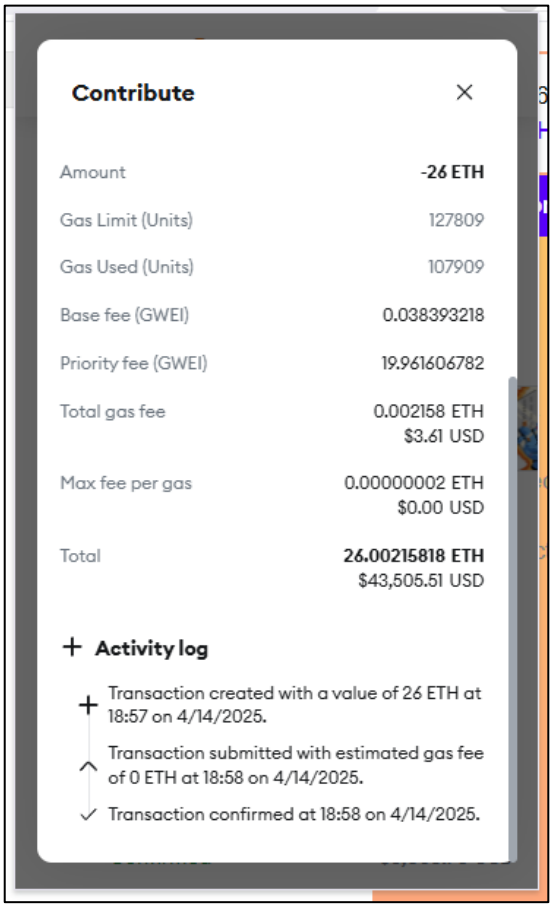

Fig 8.20 Cost of Contribution

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSION

This project successfully demonstrates the design and implementation of a decentralized crowdfunding platform using Ethereum blockchain technology. Traditional crowdfunding systems often suffer from centralization, lack of transparency, and the risk of fund misuse. By leveraging blockchain and smart contracts, this application eliminates the need for intermediaries and enables trustless interactions between campaign creators and contributors.

The application allows users to securely connect via MetaMask, create projects with defined goals and deadlines, and contribute Ether directly through their wallets. All campaign data and contributions are recorded immutably on the blockchain, ensuring complete transparency and security. The use of Solidity for smart contracts, Truffle for deployment, Ganache for local testing, and React.js for the frontend collectively contribute to a robust and scalable solution.

Through smart contract automation, contributors are assured that campaigns receive funds directly and securely. Real-time blockchain integration via Web3.js allows the user interface to reflect live campaign statuses, further increasing user trust and platform usability.

In conclusion, this decentralized crowdfunding application demonstrates how blockchain can transform financial interactions by promoting openness, reducing fraud, and empowering users. Future enhancements could include mobile support, KYC integration, analytics dashboards, and deployment on public Ethereum testnets or the mainnet for broader accessibility and impact.

9.2 Future Scope

While the current implementation successfully demonstrates the core functionality of a decentralized crowdfunding platform, there is significant potential to enhance and expand the system in future iterations.

One major area of development is the integration of Know Your Customer (KYC) and identity verification mechanisms to comply with regulatory standards and improve platform credibility. This would help filter out fraudulent campaigns and build trust among contributors, especially in large-scale fundraising scenarios.

Another opportunity lies in deploying the platform on Ethereum testnets like Goerli or Sepolia, or even the Ethereum mainnet, enabling real-world transactions with actual tokens. Support for Layer-2 solutions such as Polygon or Arbitrum could also be integrated to reduce gas fees and increase scalability.

The user experience can be improved by developing a mobile-responsive version or a full mobile application to increase accessibility and usability across devices. In addition, implementing data analytics and funding trends dashboards can help users and creators make informed decisions based on platform-wide insights.

Support for multi-currency wallets and token-based rewards (e.g., ERC-20/721 tokens for backers) can also be explored. These features can incentivize participation and introduce gamification into the funding ecosystem.

Finally, adopting DAO (Decentralized Autonomous Organization) governance could allow the platform to evolve under community control, further decentralizing decision-making processes.

CHAPTER 10

REFERENCES

10.1 Websites

Websites referred for Secure Chat Applications and serverless app designs:

1. https://www.tpointtech.com/how-does-blockchain-support-crowdfunding
2. https://www.e3s-conferences.org/articles/e3sconf/pdf/2024/21/e3sconf_icecs2024_02021
3. https://www.developcoins.com/blockchain-crowdfunding-platform
4. https://www2.fundsforngos.org/articles-searching-grants-and-donors/the-role-of-cryptocurrency-in-the-future-of-crowdfunding/

10.2 Reference Books

1. "**Mastering Ethereum: Building Smart Contracts and DApps.**" by Antonopoulos, A. M., & Wood, G.
   - This comprehensive guide to Ethereum explains smart contracts, the Ethereum Virtual Machine (EVM), and how to build decentralized applications (dApps). It is an essential resource for blockchain developers.

2. "**Blockchain Basics: A Non-Technical Introduction in 25 Steps.**" by Drescher, D.
   - Offers a beginner-friendly yet thorough explanation of blockchain principles, ideal for understanding the foundation on which Ethereum and crowdfunding platforms are built.

3. "**Bitcoin and Cryptocurrency Technologies.**" by Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S.

   - Although focused on Bitcoin, this book provides crucial insights into the broader blockchain ecosystem, including decentralization, consensus mechanisms, and security—all of which apply to Ethereum.

4. "**Ethereum Projects for Beginners.**" by Zheng, Q., & Li, X.

   - This book offers step-by-step guides on developing smart contracts and deploying them on Ethereum, including practical projects similar to crowdfunding applications.

5. "**Designing Data-Intensive Applications**" by Martin Kleppmann

   - Provides insights on data storage, distributed systems, and data synchronization—relevant for P2P and secure communication.

6. "**Network Security Essentials: Applications and Standards**" by William Stallings

   - Focuses on practical network security standards and applications, including firewalls, VPNs, and secure communication.

7. "**Token Economy: How the Web3 Reinvents the Internet.**" by Harris, J.

   - A strategic overview of how tokens, smart contracts, and decentralized platforms are reshaping industries. Includes insights into how crowdfunding and financing can evolve using Ethereum and blockchain.

10.3 References:

1. Buterin, V. (2014). **A Next-Generation Smart Contract and Decentralized Application Platform**. Ethereum White Paper.

2. Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). **An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends**. IEEE International Congress on Big Data.

3. Molavi, R., & McDonald, C. (2021). **Blockchain-Based Crowdfunding: An Overview and Research Agenda**. Journal of Theoretical and Applied Electronic Commerce Research, 16(6), 2664–2682.

4. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., et al. (2016). **Formal Verification of Smart Contracts: Short Paper**. ACM Workshop on Programming Languages and Analysis for Security.

5. Giudici, G., Nava, R., & Rossi Lamastra, C. (2018). **Crowdfunding: The Role of Blockchain Technology in Trust Disintermediation**. Journal of Industrial and Business Economics, 45, 65–84.

6. Alharby, M., & van Moorsel, A. (2017). **Smart Contract Development: Challenges and Opportunities**. arXiv preprint**.**

7. Hen, C., & Bellavitis, C. (2020). **Blockchain Disruption and Decentralized Finance: The Rise of Decentralized Business Models**. Journal of Business Venturing Insights, 13, e00151.

RESEARCH PAPERS

Indonesian Journal of Electrical Engineering and Computer Science
Vol. 15, No. 1, July 2019, pp. 409~413
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v15.i1.pp409-413

Blockchain based crowdfunding systems

Md Nazmus Saadat, Syed Abdul Halim, Husna Osman, Rasheed Mohammad Nassr, Megat F. Zuhairi
Malaysian Institute of Information Technology, Universiti Kuala Lumpur, Malaysia

Article Info

Article history:

Received Dec 04, 2018
Revised Jan 21, 2019
Accepted Mar 10, 2019

Keywords:

Blockchain
Crowdfunding
Malaysia
Smart contracts

ABSTRACT

Initially, blockchain is only used as a foundation of cryptocurrency, but today, we can see the rise of this new emerging technology are being implemented in many industries. In the future, most technologies around the world are expected to use blockchain as an efficient way to make online transactions. One of the areas that blockchain technologies can be applied is crowdfunding platforms. The most common problem with current crowdfunding scene in around the world including is that the campaigns are not regulated and some of the crowd-funding campaign turned out to be fraud. Besides, the completion of some projects also was significantly delayed. This project aims to solve these problems by applying Ethereum smart contracts to the crowdfunding site to that the contracts will be fully automatically executed, thus preventing frauds and ensuring that the projects can be delivered within duration given.

Corresponding Author:

Md. Nazmus Saadat,
Systems & Networking,
Malaysian Institute of Information Technology (MIIT),
Kuala Lumpur, Malaysia.
Email: mdnazmus@unikl.edu.m

1. INTRODUCTION

In simple terms, crowdfunding can be defined as raising funds for a project or a campaign by a group of people instead of using established entities such as banks or loan provider. Freedman and Nutting [1] defined crowdfunding as a method of collecting many small contributions, by means of an online funding platform, to finance or capitalize a popular enterprise. The crowdfunding action mainly involved three parties, which are the contributors, crowdfunding platform and project managers. Some popular crowdfunding plat-forms include kickstarter.com, Indiegogo.com and mystartr.com.

The main benefit of crowdfunding is that it can raise the amount of money needed in short amount of time. This is due to that many people today use Internet and social media which means that through these channels the project founder can reach out the public within a short amount of time [2]. Besides, many project founders have chosen crowdfunding to raise money for their projects as it is harder to gain loans from bank or other investors [3]. This happen because that most loans take a long time to be processed. Some studies also stated that there are benefits of crowdfunding in non-financial term. For instance, crowd funders can provide value-added involvement and feedback to the project, while also creating publicity and public awareness of the business [4]. Schlueter [5] believes that there are two main advantages of crowdfunding. The first bene-fit is crowdfunding provide better matches between the inventors and the funders from all around the world. The second advantage is, investors also have access to more information in the initial phase of the project. This information is very valuable to investors and might boost their eagerness to invest on such crowdfunding projects.

However, despite having several advantages, crowdfunding platforms still have many flaws that needed to be improved. One of the main issues that that have been in traditional crowdfunding platform is fraud cases [6], stated that online crowdfunding leaves contributors susceptible to fraud because traditional legal and reputation security measures may not work. This is further stressed in [7] who stated as no

credentials are needed to post a project and, once the project has been posted, and there are few legal obligations to deliver what the project promised. Some other crowdfunding problems highlighted by other researchers includes 1) the rewards are significantly delayed 2) campaign initiators cease communicating with their backers for more than six months after an unmet delivery date, or 3) the promised product is never delivered and the backers are not fully refunded [8]. Another study made in [9], shows that there were over 75% crowdfunding projects deliver products later than expected.

By implementing smart contracts in crowdfunding system, we can create a contract that will hold a contributor's money until any given date or goal is reached. Depending on the outcome, the funds will either be given to the project owners or safely returned to the contributors.

Blockchain can be defined as a distributed database of records of all transactions that have been executed and shared among participated parties. The characteristics of blockchain includes decentralization of data, persistency, anonymity and auditability [10]. There are two main components in blockchain system, which are transaction and block. Transaction represents the action triggered by the participant, while the block is a collection of data recording the transaction and other associated details such as the correct sequence, timestamp of creation, etc [11]. The transaction-records, or blocks, in a blockchain are linked together cryptographically, rendering them tamper proof [12]. This means that each block that have been inserted cannot be modified or deleted. To achieve reliability, blockchain uses consensus algorithms. This is to ensure that all nodes in the system have same exact copy, thus eliminating the need of trust between each node in the network. The process to establish consensus in the blockchain network is called mining. Mining is important as it ensures that each data that added into the ledger is immutable and secure [13]. Among popular consensus algorithms include Proof of Work (PoW), Proof of Stake (PoS) and Byzantine Fault Tolerance (BFT). As the blockchain has many advantages over traditional system, various industries around the world are being disrupted by its implementation. This includes real estate activities, finance and insurance, arts, entertainment & recreations, etc [14]. Among the advantages of using blockchain technologies include ability to operate in with high efficiency and low cost. Besides, the integrity of blockchain data also can be trusted as once it entered the ledger, it cannot be tampered by anymore [15]. The only concern about using blockchain technologies is the high usage of power and computing resources by the distributed system [16]. Due to its anonymous nature, blockchain technology also may lead to cybercrimes which includes money laundering, online gambling, get-rich-quick scams, extortion, illegal trading and terrorist financing [17].

One of the appealing features in the blockchain is smart contract. Smart contract can be defined as executable code that runs on blockchain to facilitate, execute and enforce the terms of an agreement. It is used to automatically executed the agreements once the specified conditions are met [20]. Smart contracts can be deployed on different platforms such as Bitcoin, NXT and Ethereum. It consists of program code, a storage file, and account balance. Users in blockchain network can create a contract by posting transactions on blockchain network [18].

One of the areas that blockchain can be implemented is crowdfunding platform. Nowadays, most of the existing crowdfunding plat-form today are centralized-based platform which can leads to some problems. As stated in [6], online crowdfunding leaves contributors susceptible to fraud because traditional legal and reputation security measures may not work. Among example of popular crowdfunding platform nowadays include kickstarter.com, Indiegogo.com, mystartr.com, and fundly.com. The applications of blockchain in crowdfunding platform have been discussed in [19]. The author stated that the blockchain can solve crowdfunding issues such as fraud, money laundering, and information asymmetry. As the data in blockchain is transparent, investors can scrutinize the data on the block to correspond the project's authenticity. There are also some companies that have started implementing blockchain in crowdfunding platform. At this moment this research is written, crowfunding platform that we found are WeiFund, Acorn and Cybit. WeiFund is operated by allowing user to create campaigns which used smart contracts. Besides, it also allows user to create their own smart contracts. At this moment, WeiFund is still in alpha phase. Meanwhile, Cybit and Acorn are still in main ICO crowdsale phase. In Malaysia context however, there is no undergoing development for blockchain based crowdfunding yet. Table 1 shows the current crowdfunding platforms in Malaysia.

In Malaysia, usage of cryptocurrencies is still studied by Central Bank Malaysia (CBM). As for now, usage of Bitcoin or other cryptocurrencies is not illegal in Malaysia, however, it is not recognized as legal tenders [17].

Table 1. List of Existing Crowdfunding Platform in Malaysia

No	Name	Niche	Use blockchain?
1	Mystatr	Social clause	No
2	PitchIn	Equity crowdfunding	No
3	SkolaFund	Education	No
4	Give.my	Education	No
5	100percentproject	Education	No

2. RESEARCH METHOD

The system is built by using ReactJs as front-end and use NodeJS as its backend. For contracts development, Solidity language is used. The contract is compiled into ABI code in JSON format using solc npm package. The ABI interface then is parsed to instance of Web3 provider for contract deployment. Instead of using local node, we used Infura which acts as remote node to connect to Ethereum network. To start using the system, we need to set up cryptocurrency wallet called Metamask. Metamask is a browser extension which allows users to interact with any decentralized application(dApps). After user create account in Metamask, he can transfer the Ethereum to his account. Once user have some Eth in his account, he can start interacting with the system as Metamask has injected a Web3 instance in the web browser.

Then, campaign can start creating campaign and other users can contribute to the campaign. Apart from that, campaign manager also can create requests to show how the money collected will be used. The contributors decide whether the expenses are appropriated or not, and if it is approved by majority of the backers, then only the Ether will be sent to the vendors. The system is connected to the Ethereum network by using Infura infrastructure. As this system only acts as prototype, we do not use main Ethereum network, instead, we used testnet which behaves similarly to main Ethereum network. For this project, we use a proof-of-authority blockchain called Rinkeby network to stimulate transactions that are performed by users in this system. As we are using Rinkeby network, Ether cannot be mined, instead, we have to request from Rinkeby Test Faucet at https://faucet.rinkeby.io/. The details of transactions performed by users, either fail or success, can be viewed by using Etherscan API. Figure 1 shows the diagram of system architecture and Figure 2 shows the transactional flow of Ethereum in the system.

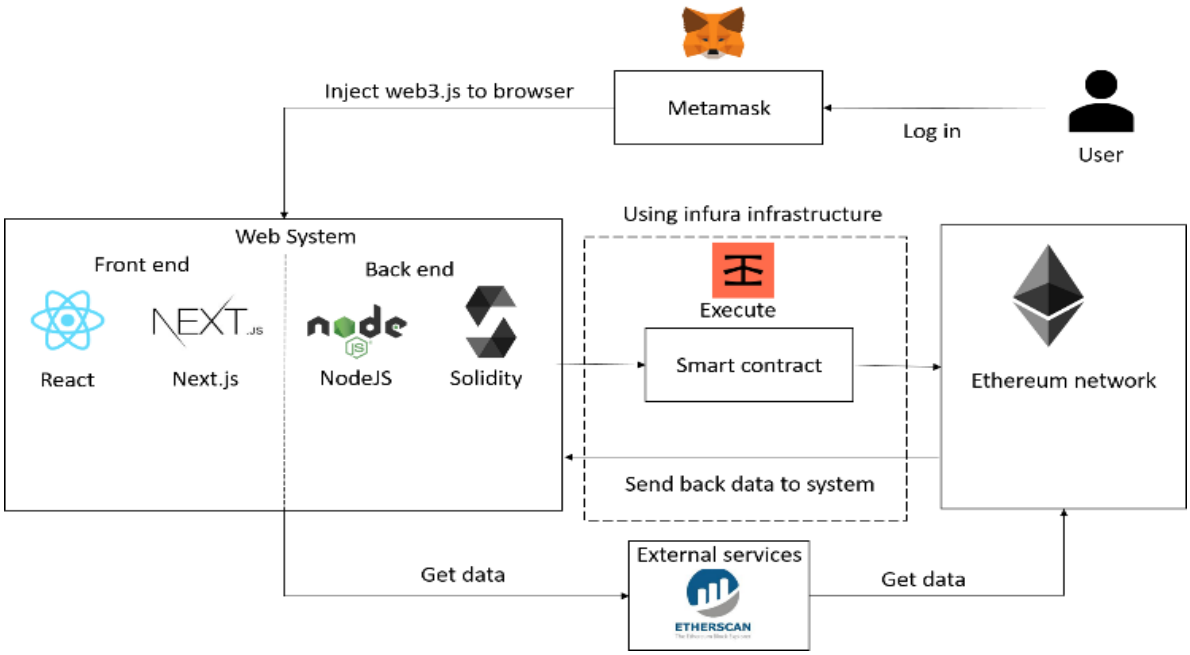

Figure 1. System architecture

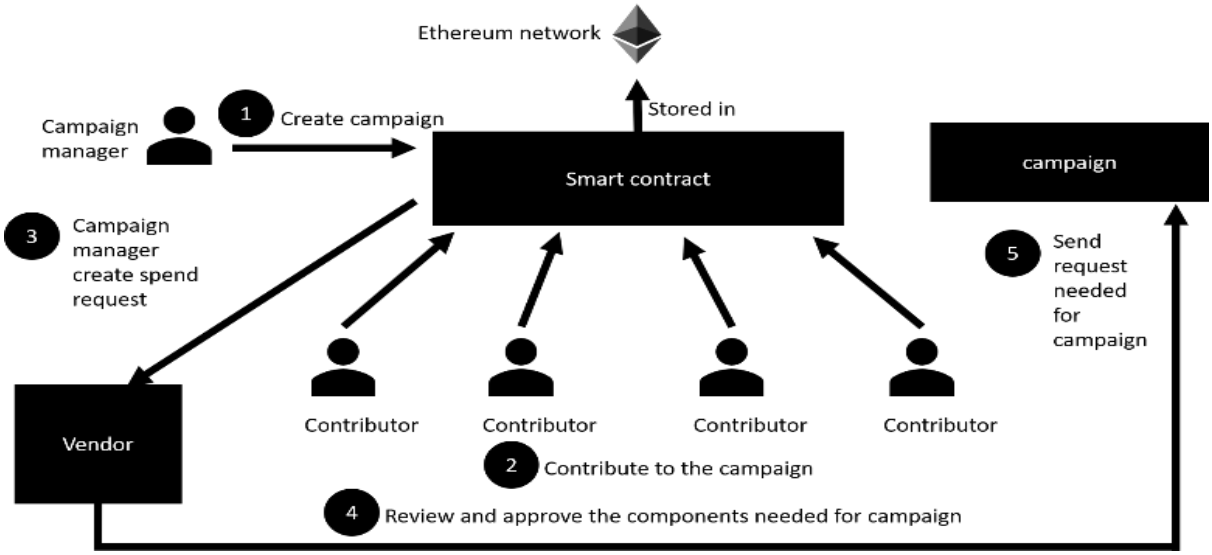

Figure 2. Flow of ether in proposed blockchain model

Flow:

a. Campaign manager create campaign. To create the campaign, he needs to fill campaign details. To convince contributors, he also may upload the proposal in PDF format. The proposal will be stored in InterPlanetary File System (IPFS), a peer-to-peer file sharing system.
b. The campaign then will appear in campaign page. If contributors are interested with the campaign, he may contribute to the campaign.
c. Campaign manager create expense request, which consists of list the items needed to execute the campaign.
d. Contributors received notifications new expense request has been added.
e. Contributors review either the item proposed by campaign manager is appropriate or not, if it is appropriate, then contributors can vote to agree with the item listed.
f. If majority of contributors agree, smart contract will send collected funds to the respective vendors.
g. The respective vendor then sends the item agreed to the campaign manager.

Each transaction is recorded in blockchain and can be seen by all users in Etherscan website as shown in Figure 3.

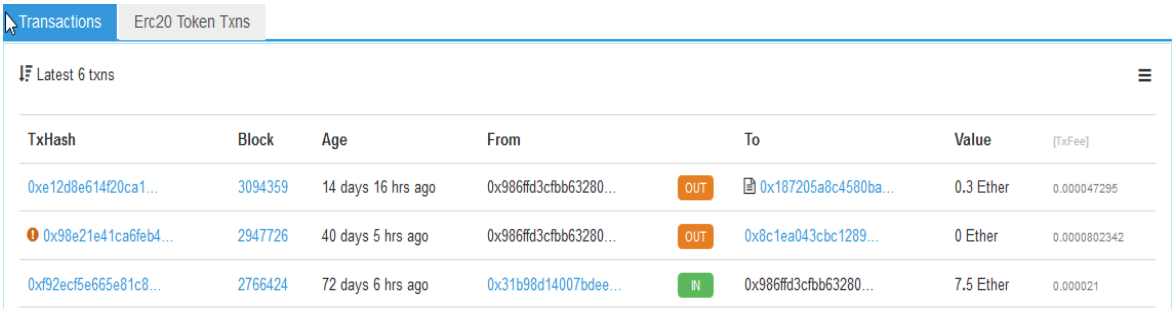

Figure 3. Records of Ethereum transaction

3. RESULTS AND ANALYSIS

Implementation of blockchain technology in crowdfunding platform increase contributor's confidentiality when contributing to campaign. This is due to the nature of blockchain transactions which is transparent. All users can view the records of each transaction which can be seen by using Etherscan API. Besides, the implementation of smart contracts also eliminates the need of trust of each stakeholder for the campaign as the contract is automatically executed once the conditions are met. We are in midst of finalizing

the implementation of the system and data of the results would be available soon which is not yet available in our hand at this moment. There would be some acceptance study as well which is also in progress.

Implementation of blockchain technology in crowdfunding platform increase contributor's confidentiality when contributing to campaign. This is due to the nature of blockchain transactions which is transparent. All users can view the records of each transaction which can be seen by using Etherscan API. Besides, the implementation of smart contracts also eliminates the need of trust of each stakeholder for the campaign as the contract is automatically executed once the conditions are met.

4. CONCLUSION

Provide a statement that what is expected, as stated in the "Introduction" chapter can ultimately In this paper, blockchain based crowdfunding platform is proposed to provide more transparent transactions in decentralize structure. For future works, we plan to implement ERC-223 tokens into the smart contracts as it provides more benefits which are: Consume less gas, allows developers to handle incoming token transactions and eliminates the problems of lost tokens [21].

REFERENCES

[1] Freedman et al. "The Foundations of Online Crowdfunding", *Equity Crowdfunding for Investors (eds D. M. Freedman and M. R. Nutting), 2015,* doi:10.1002/9781118864876.ch1.

[2] T. Dannberg, "Advantages and Disadvantages with Crowdfunding : - and Who are the Users?", *Dissertation,* 2017.

[3] Schwienbacher et al, "Crowdfunding of Small Entrepreneurial Ventures", *Handbook Of Entrepreneurial Finance, Oxford University Press*. http://dx.doi.org/10.2139/ssrn.1699183.

[4] Macht et al, " The Benefits of Online Crowdfunding for Fund-Seeking Business Ventures", *Strategic Change,* 2014, 23 (1-2). pp. 1-14. ISSN 10861718.

[5] Schlueter et al, "Underlying Benefits and Drawbacks of Crowdfunding from the Perspective of Enterepeneurs in Germany", *5th IBA Bachelor Thesis Conference,* University of Twente. Available at: http://essay.utwente.nl/67409/1/Schlueter_BA_MB.pdf [Accessed 15 Aug 2018].

[6] Gabison et al, "Understanding Crowdfunding and its Regulation", 2015, doi:10.2791/562757.

[7] Ramos et al, S. (2014). "Crowdfunding and the Role of Managers in Ensuring the Sustainability of Crowdfunding Platforms (Rep.)", *Publications Office of the European Union*. doi:10.2791/76003.

[8] Cumming et al, "Crowdfunding from Fraudfunding", *Max Planck Institute for Innovation & Competition* Research Paper No. 16-09. Available at SSRN: https://ssrn.com/abstract=2828919 or http://dx.doi.org/10.2139/ssrn.2828919.

[9] Mollick et al., "The Dynamics of Crowdfunding: An Exploratory Study (June 26, 2013)", *Journal of Business Venturing*, Volume 29, Issue 1, January 2014, Pages 1–16.

[10] Z. Zheng et al, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends", *IEEE International Congress on Big Data (BigData Congress), Honolulu*, HI, 2017, pp. 557-564.doi: 10.1109/BigDataCongress.2017.85.

[11] Miraz et al., "Applications of Blockchain Technology beyond Cryptocurrency", *Annals of Emerging Technologies in Computing* (AETiC), 2018. 2. 1-6.

[12] Chen et al., "Exploring Blockchain Technology and its Potential Applications for Education. Smart Learning Environments", 5. 10.1186/s40561-017-0050-x.

[13] A. Angrish et al., "A Case Study for Blockchain in Manufacturing: "FabRec": A Prototype for Peer-to-Peer Network of Manufacturing Nodes", *Procedia Manufacturing*, vol. 26, pp. 1180-1192, 2018.

[14] Friedlmaier et al., "Disrupting Industries With Blockchain: The Industry, Venture Capital Funding, and Regional Distribution of Blockchain Ventures", *Proceedings of the 51st Annual Hawaii International Conference on System Sciences* (HICSS), January 2018. Available at SSRN: https://ssrn.com/abstract=2854756 or http://dx.doi.org/10.2139/ssrn.2854756.

[15] Alharby et al., "Blockchain Based Smart Contracts: A Systematic Mapping Study", 2017, 125-140. 10.5121/csit.2017.71011.

[16] Delmolino et al., "Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab", 2016, 9604. 79-94. 10.1007/978-3-662-53357-4_6m.

[17] Gebert et al., "Application Of Blockchain Technology In Crowdfunding", 2017, New European.

[18] Ming Li et. al. (2017). CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing. IACR Cryptology ePrint Archive.

[19] [plasma whitepaper]

[20] Tsankov, P., Dan, A.M., Cohen, D.D., Gervais, A., Buenzli, F., & Vechev, M.T. (2018). Securify: Practical Security Analysis of Smart Contracts. CoRR, abs/1806.01143.

[21] A. Kosba, A. Miller, E. Shi, Z. Wen and C. Papamanthou, "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts," 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, 2016, pp. 839-858.

BLOCKCHAIN-BASED SMART CONTRACTS : A SYSTEMATIC MAPPING STUDY

Maher Alharby[1,2] and Aad van Moorsel[1]

[1]School of Computing Science, Newcastle University, Newcastle, UK
[2]College of Computer Science and Engineering, Taibah University, Medina, KSA

ABSTRACT

An appealing feature of blockchain technology is smart contracts. A smart contract is executable code that runs on top of the blockchain to facilitate, execute and enforce an agreement between untrusted parties without the involvement of a trusted third party. In this paper, we conduct a systematic mapping study to collect all research that is relevant to smart contracts from a technical perspective. The aim of doing so is to identify current research topics and open challenges for future studies in smart contract research. We extract 24 papers from different scientific databases. The results show that about two thirds of the papers focus on identifying and tackling smart contract issues. Four key issues are identified, namely, codifying, security, privacy and performance issues. The rest of the papers focuses on smart contract applications or other smart contract related topics. Research gaps that need to be addressed in future studies are provided.

KEYWORDS

Blockchain, Smart contracts, Systematic mapping study, Survey

1. INTRODUCTION

Transactions between parties in current systems are usually conducted in a centralised form, which requires the involvement of a trusted third party (e.g., a bank). However, this could result in security issues (e.g., single point of failure) and high transaction fees. Blockchain technology has emerged to tackle these issues by allowing untrusted entities to interact with each other in a distributed manner without the involvement of a trusted third party. Blockchain is a distributed database that records all transactions that have ever occurred in a network. Blockchain was originally introduced for Bitcoin (a peer-to-peer digital payment system), but then evolved to be used for developing a wide range of decentralised applications. An appealing application that can be deployed on top of blockchain is smart contracts.

A smart contract is executable code that runs on the blockchain to facilitate, execute and enforce the terms of an agreement between untrusted parties. It can be thought of as a system that releases digital assets to all or some of the involved parties once the pre-defined rules have been met [1]. Compared to traditional contracts, smart contracts do not rely on a trusted third party to operate, resulting in low transaction costs. There are different blockchain platforms that can be utilised to develop smart contracts, but Ethereum is the most common one. This is because Ethereum's language supports Turing-completeness feature that allows creating more advanced and

Dhinaharan Nagamalai et al. (Eds) : AIS, CSIT, IPPR, IPDCA - 2017
pp. 125– 140, 2017. © CS & IT-CSCP 2017 DOI : 10.5121/csit.2017.71011

customised contracts. Smart contracts can be applied to different applications (e.g., smart properties, e-commerce and music rights management).

The main aim of this study is to identify the research topics that have been carried out about blockchain-based smart contracts and current challenges that need to be addressed in future studies. To achieve this aim, we selected a systematic mapping study as the methodology for our study. We followed the systematic mapping process presented in [2] to search for relevant papers in scientific databases and to produce a map of current smart contract research. The produced map could help researchers identify gaps for future studies. The focus of our study is to only explore smart contract studies from a technical point of view.

The structure of this paper is as follows. Section 2 discusses background information about blockchain and smart contracts technologies. It also discusses several smart contract platforms and potential applications. Section 3 describes the research methodology adopted for our study. Section 4 presents the results of searching and screening for relevant papers and the results of classifying smart contract topics. Section 5 discusses the results and answers the research questions of the study. Section 6 concludes the paper.

2. BACKGROUND

This section presents general background information about blockchain and smart contracts technologies. It also discusses some blockchain platforms that support the development of smart contracts. Finally, it provides some potential use cases for smart contracts.

2.1. Blockchain Technology

A blockchain is a distributed database that records all transactions that have ever occurred in the blockchain network. This database is replicated and shared among the network's participants. The main feature of blockchain is that it allows untrusted participants to communicate and send transactions between each other in a secure way without the need of a trusted third party. Blockchain is an ordered list of blocks, where each block is identified by its cryptographic hash. Each block references the block that came before it, resulting in a chain of blocks. Each block consists of a set of transactions. Once a block is created and appended to the blockchain, the transactions in that block cannot be changed or reverted. This is to ensure the integrity of the transactions and to prevent double-spending problem.

Cryptocurrencies have emerged as the first generation of blockchain technology. Cryptocurrencies are basically digital currencies that are based on cryptographic techniques and peer-to-peer network. The first and most popular example of cryptocurrencies is Bitcoin. Bitcoin [3] is an electronic payment system that allows two untrusted parties to transact digital money with each other in a secure manner without going through a middleman (e.g., a bank). Transactions that occurred in the network are verified by special nodes (called miners). Verifying a transaction means checking the sender and the content of the transaction. Miners generate a new block of transactions after solving a mathematical puzzle (called Proof of Work) and then propagate that block to the network. Other nodes in the network can validate the correctness of the generated block and only build upon it if it was generated correctly. However, Bitcoin has limited programming capabilities to support complex transactions. Bitcoin, thus, does not support the creation of complex distributed applications on top of it.

Other blockchains such as Ethereum have emerged as the second generation of blockchain to allow building complex distributed applications beyond the cryptocurrencies. Smart contracts, which will be discussed in the following section, are considered as the main element of this generation [4]. Ethereum blockchain is the most popular blockchain for developing smart contracts. Ethereum is a public blockchain with a built-in Turing-complete language to allow writing any smart contract and decentralised application.

There are two types of blockchain, namely, public and private blockchain [5]. In a public blockchain, any anonymous user can join the network, read the content of the blockchain, send a new transaction or verify the correctness of the blocks. Examples of public blockchains are Bitcoin, NXT and Ethereum. In a private blockchain, only users with permissions can join the network, write or send transactions to the blockchain. A company or a group of companies are usually responsible for giving users such permissions prior to joining the network. Examples of private blockchains are Everledger, Ripple and Eris.

2.2. Smart Contracts

A smart contract is executable code that runs on the blockchain to facilitate, execute and enforce the terms of an agreement. The main aim of a smart contract is to automatically execute the terms of an agreement once the specified conditions are met. Thus, smart contracts promise low transaction fees compared to traditional systems that require a trusted third party to enforce and execute the terms of an agreement. The idea of smart contracts came from Szabo in 1994 [6]. However, the idea did not see the light till the emergence of blockchain technology. A smart contract can be thought of as a system that releases digital assets to all or some of the involved parties once arbitrary pre-defined rules have been met [1]. For instance, Alice sends X currency units to Bob, if she receives Y currency units from Carl.

Many different definitions of a smart contract have been discussed in the literature. In [7], the author classified all definitions into two categories, namely, smart contract code and smart legal contract. Smart contract code means "code that is stored, verified and executed on a blockchain" [7]. The capability of this smart contract depends entirely on the programming language used to express the contract and the features of the blockchain. Smart legal contract means code to complete or substitute legal contracts. The capability of this smart contract does not depend on the technology, but instead on legal, political and business institutions. The focus of this study will be on the first definition, which is smart contract code.

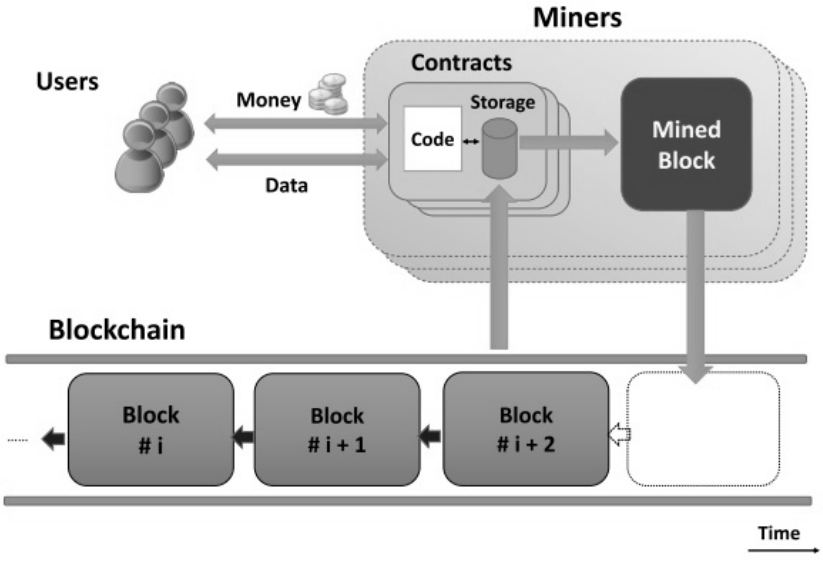

Figure 1. Smart contract system [8].

A smart contract has an account balance, a private storage and executable code. The contract's state comprises the storage and the balance of the contract. The state is stored on the blockchain and it is updated each time the contract is invoked. Figure 1 depicts the smart contract system.

Each contract will be assigned to a unique address of 20 bytes. Once the contract is deployed into the blockchain, the contract code cannot be changed. To run a contract, users can simply send a transaction to the contract's address. This transaction will then be executed by every consensus node (called miners) in the network to reach a consensus on its output. The contract's state will then be updated accordingly. The contract can, based on the transaction it receives, read/write to its private storage, store money into its account balance, send/receive messages or money from users/other contracts or even create new contracts.

There are two types of smart contracts, namely, deterministic and non-deterministic smart contracts [9]. A deterministic smart contract is a smart contract that when it is run, it does not require any information from an external party (from outside the blockchain). A non-deterministic smart contract is a contract that depends on information (called oracles or data feeds) from an external party. For example, a contract that requires the current weather information to be run, which is not available on the blockchain.

2.3. Platforms for Smart Contracts

Smart contracts can be developed and deployed in different blockchain platforms (e.g., Ethereum, Bitcoin and NXT). Different platforms offer distinctive features for developing smart contracts. Some platforms support high-level programming languages to develop smart contracts. We will only focus on three public platforms in this section.

- Bitcoin [3] is a public blockchain platform that can be used to process cryptocurrency transactions, but with a very limited compute capability. Bitcoin uses a stack-based bytecode scripting language. The ability of creating a smart contract with rich logic using Bitcoin scripting language is very limited [10]. In Bitcoin, a simple logic that requires multiple signatures to sign a single transaction before confirming the payment is possible. However, writing contracts with complex logic is not possible due to the limitations of Bitcoin scripting language. Bitcoin scripting language, for example, neither supports loops nor withdrawal limits [1]. To implement a loop, the only possible way is by repeating the code many times, which is inefficient.
- NXT is a public blockchain platform that includes built-in smart contracts as templates [10]. NXT only allows developing smart contracts using those templates. It does not, however, allow customized smart contracts due to the lack of Turing-completeness in its scripting language.
- Ethereum [1,11] is a public blockchain platform that can support advanced and customized smart contracts with the help of Turing-complete programming language. Ethereum platform can support withdrawal limits, loops, financial contracts and gambling markets. The code of Ethereum smart contracts is written in a stack-based bytecode language and executed in Ethereum Virtual Machine (EVM). Several high-level languages (e.g., Solidity, Serpent and LLL) can be used to write Ethereum smart contracts. The code of those languages can then be compiled into EVM bytecodes to be run. Ethereum currently is the most common platform for developing smart contracts.

2.4. Smart Contract Applications

There are various possible applications where smart contracts can be applied to. Some of these applications are as follows:

- Internet of Thing and smart property [12]: there are billions of nodes that are sharing data between each other through the Internet. A potential use case of blockchain-based smart contracts is to allow those nodes to share or access different digital properties without a trusted third party. There are various companies that investigate this use case. For example, Slock.it is a German company that utilises Ethereum-based smart contracts for

renting, selling or sharing anything (e.g, selling a car) without the involvement of a trusted third party.

- Music rights management [13]: a potential use case is to record the ownership rights of a music in the blockchain. A smart contract can enforce the payment for music owners once a music is used for commercial purposes. It also ensures the payment is being distributed between the music's owners. Ujo is a company that investigates the use of blockchain-based smart contracts in the music industry.
- E-commerce: a potential use case is to facilitate the trade between untrusted parties (e.g., seller and buyer) without a trusted third party. This would result in reduction of trading costs. Smart contracts can only release the payment to the seller once the buyer is satisfied with the product or service they received [14].

There are other possible applications such as e-voting, mortgage payment, digital right management, motor insurance, distributed file storage, identity management and supply chain.

3. RESEARCH METHODOLOGY

We selected the systematic mapping study presented in [2] as the research methodology for our study to explore studies related to smart contracts. The results of this systematic mapping study would allow us to identify and map research areas related to smart contracts. In addition, it would allow us to identify research gaps that need to be considered for future studies. The process for the systematic mapping study falls into five steps as depicted in Figure 2.

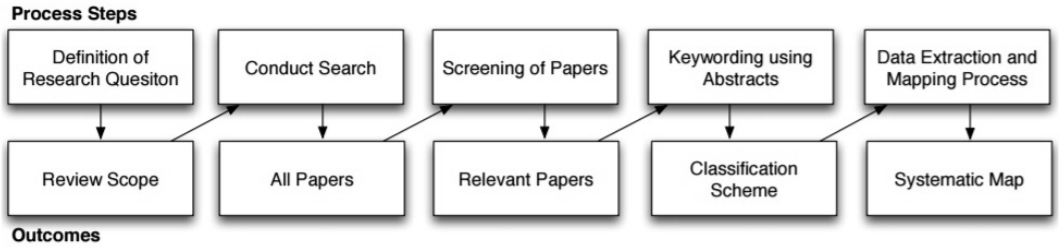

Figure 2. Steps of the systematic mapping study [2].

Definition of research questions:

This step is to identify the research questions the study is aiming to answer. For our study, we defined the following research questions:

RQ1. What are the current research topics on smart contracts?

RQ2. What are the current smart contract applications?

RQ3. What are the research gaps that need to be addressed in future studies?

Conducting the search:

This step is to search and to find all scientific papers that are related to the research topic, which is smart contracts. For our study, we decided to select the term ‘smart contract’ as the main keyword to search for papers. We selected this term because we wanted to narrow down the focus of our study to only cover smart contract related works. After identifying the keyword for the searching process, we selected the scientific databases to conduct our search. We selected IEEE Explore, ACM Digital Library, ScienceDirect, Springer, Ebsco and Scopus. Our focus was to only include high quality papers published in conferences, journals, workshops, symposiums and books.

Screening for relevant papers:

This step is to search for papers that are relevant to our research questions. We followed the same approach as in [15] to look for relevant papers. We first tried to exclude papers that were irrelevant to our study based on their titles. If we were unable to decide on a paper, we would go a step further by examining its abstract. We also used exclusion criteria to screen each paper. We excluded: (1) non-English papers, (2) papers without full text available, (3) papers that utilised smart contracts in fields other than computer science, (4) redundant papers and (5) articles, newsletters and grey literature.

Key-wording using abstracts:

This step is to classify all relevant papers using the key-wording technique described in [15]. We first read the abstract of each paper to identify the most important keywords and the main contribution. Those keywords were then used to classify papers into various categories. After classifying all papers, we read the papers and made changes to the classification when necessary.

Data extraction and mapping process:

This process is to gather all the required information to address the research questions of this study. We gathered different data items from each paper. These data items embrace the main aims and contributions of papers.

4. STUDY RESULTS

This section discusses the results of the systematic mapping study that we conducted on smart contracts. We first discuss the results of searching and screening for relevant papers. Then, we discuss the results of the classification process.

4.1 Searching and Screening Results

Searching and screening for relevant papers are two steps of the systematic mapping study that we discussed in Section 3. The results of these steps are as follows. In the searching phase, we looked for all papers using the term ‘smart contract’ in different scientific databases. We gathered 154 papers in total (as on 5 May 2017). In the screening phase, we first excluded irrelevant papers based on their titles and/or their abstracts (we excluded 109 irrelevant papers). There are two reasons why we had a high number of excluded papers. First, many papers were irrelevant to our study, since our focus was to explore smart contracts from a technical perspective. For instance, many papers discussed the topic from an economic or legal point of view. Another reason is that some excluded papers were about cryptocurrencies or blockchain in general, which do not contribute to our research questions. After that, 17 papers were removed as they were duplicates, resulting in 28 papers. Among the 28 papers, four papers were excluded as they only discuss general information about smart contract and how it works, without providing any useful contribution. Thus, we only selected 24 papers to conduct our systematic mapping study. Figure 3 summaries the results of searching and screening for relevant papers.

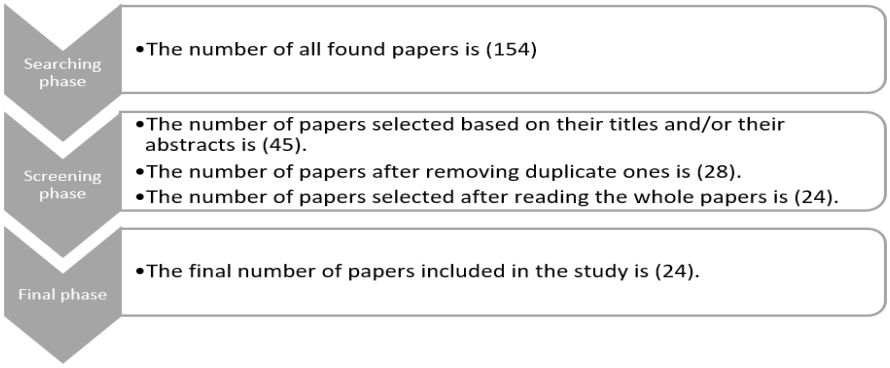

Figure 3. Searching and screening results.

4.2 Classification Results

By applying the Key-wording technique that we discussed in Section 3, we classified the papers into two categories, namely, smart contract issues and other smart contract related topics. We found about two thirds of the papers fall into smart contract issues category. We classified those issues into four categories, namely, codifying, security, privacy and performance issues. *Codifying issues* mean challenges that are related to the development of smart contracts. *Security issues* mean bugs or vulnerabilities that an adversary might utilise to launch an attack. *Privacy issues* mean issues related to disclosing contracts information to the public. *Performance issues* mean issues that affect the ability of blockchain systems to scale. Table 1 summaries the identified issues and the proposed solutions. For other smart contract related topics category, there are nine papers that developed smart contract applications or reported about other topics (e.g., the combination of smart contract and The Internet of Thing).

Table 1. Smart contract issues and the proposed solutions.

Smart contract issues		**Proposed solutions**
Codifying issues	Difficulty of writing correct smart contracts [8,16,17,18].	• Semi-automation of smart contracts creation [18]. • Use of formal verification methods [16,17]. • Education (e.g., online tutorials) [8].
	Inability to modify or terminate smart contracts [19].	• A set of standards for modifying/terminating smart contracts [19].
	Lack of support to identify under-optimised smart contracts [20].	• Use of 'GASPER' tool [20].
	Complexity of programming languages [21].	• Use of logic-based languages [21].
Security issues	Transaction-ordering dependency vulnerability [22,23].	• Use of 'SendIfReceived' function [22]. • Use of a guard condition [23]. • Use of 'OYENTE' tool [23].
	Timestamp dependency vulnerability [23].	• Use block number as a random seed instead of using timestamp [23]. • Use of 'OYENTE' tool [23].
	Mishandled exception vulnerability [23].	• Check the returned value [23]. • Use of 'OYENTE' tool [23].
	Re-entrancy vulnerability [23].	• Use of 'OYENTE' tool [23].

	Criminal smart contract activities [24].	• NA.
	Lack of trustworthy data feeds 'Oracles' [25].	• Use of 'Town Crier (TC)' tool [25].
Privacy issues	Lack of transactional privacy [26].	• Use of 'Hawk' tool [26]. • Use of encryption techniques [27].
	Lack of data feeds privacy [25].	• Use of 'Town Crier (TC)' tool [25]. • Use of encryption techniques [25].
Performance issues	Sequential execution of smart contracts [28].	• Parallel execution of smart contracts [28].

Codifying issues

From the literature, we found four issues that might face developers during writing smart contracts, namely, the difficulty of writing correct contracts, the inability to modify or terminate contracts, the lack of support to identify under-optimised contracts and the complexity of programming languages.

The first one is the difficulty of writing correct smart contracts [8,16,1718]. Correctness of smart contracts in this context means contracts that are functioning as intended by their developers. The reason why it is important to have correct smart contracts is because those contracts have valuable currency units [8,16]. Thus, if a smart contract was not executed as intended, some of its currency units would disappear. An example that illustrates this is the Distributed Autonomous Organisation (DAO) attack, which led to over 60 million US dollars being moved into an adversary account [23].

In an attempt to tackle this issue, three solutions were identified from the literature. The first solution is to semi-automate the creation of smart contracts [18] to ease the process of writing smart contracts. Semi-automation means the translation of human-readable contract representations to smart contract rules. The second solution is to provide developers with guidelines to aid them write correct contracts. Delmolino et al. [8], released online materials (e.g., a tutorial) to help developers write correct smart contracts. The last solution is the adoption of formal verification techniques to detect unintended behaviours of smart contracts [16,17]. This can help developers recognise those behaviours before posting their contracts to the blockchain. Bhargavan et al. [16] utilised formal methods to analyse and verify the correctness of smart contracts, while Bigi et al. [17] went a step further by combining formal methods with game theory techniques to validate smart contracts.

The second issue is the inability to modify or terminate smart contracts [19]. Due to the immutability feature of blockchain, smart contracts cannot be changed or terminated after deploying it into the blockchain. This is different from legal law which allows the rules to be modified or terminated. In an attempt to tackle this issue, Marino et al. [19] presented a set of standards to allow smart contracts to be changed or terminated. Such standards are taken from legal contracts and then defined to fit in the context of smart contracts. Those standards were then applied to Ethereum-based smart contracts to prove their success. For details about those standards and how can be applied to Ethereum-based smart contracts, we refer the reader to [19].

The third one is the lack of support to identify under-optimised smart contracts [20]. To run a smart contract, each computational or storage operation in the contract costs some money. An under-optimised smart contract is a contract that contains unnecessary or expensive operations. Such operations result in a high cost at the user's side. In an attempt to tackle this issue, Chen et al. [20] identified seven programming patterns (e.g., unnecessary and expensive operations in a loop) in smart contracts which lead to unnecessary extra costs. They also proposed ways to

enhance the optimisation of those patterns to reduce the overall cost of executing smart contracts. They proposed and developed a tool called 'GASPER' to detect contracts that suffer from those patterns. They used the tool to examine current Ethereum smart contracts and found most of them suffer from such patterns.

The last issue is the complexity of smart contract programming languages [21]. Current smart contracts are based on procedural languages such as Solidity. In a procedural language, the code is executed as a sequence of steps. Thus, programmers must specify what should be done and how to do it. This makes the task of writing smart contracts in those languages cumbersome and error prone [21]. In an attempt to tackle this issue, Idelberger et al. [21] proposed to utilise logic-based languages instead of procedural languages. In logic-based languages, programmers do not necessarily have to specify the sequence of steps for a contract. This will ease the complexity of writing smart contracts. However, algorithms for logic-based languages are expensive and inefficient.

Security issues

From the literature, we found six security issues, namely, transaction-ordering dependency, timestamp dependency, mishandled exception, criminal activities, re-entrancy and untrustworthy data feeds. In addition to these issues, Atzei et al. [29] surveyed several vulnerabilities in Ethereum smart contracts.

The first issue is transaction-ordering dependency [22,23]. This problem occurs when two dependent transactions that invoke the same contract are included in one block. The order of executing transactions relies on the miner. However, an adversary can successfully launch an attack if those transitions were not executed in the right order. For example, assume there is a puzzle contract that incentives the user who solves the puzzle. A malicious owner is listening to the solutions provided by the users. Once a user submitted a correct solution to the puzzle (Tu), the malicious owner sends a transaction (To) to update the contract's reward (e.g., reduce the reward) right away. Those two transactions (To and Tu) might be included in the same block by chance. If the miner executed To before Tu, the user would get a lower reward and the malicious owner would succeed in his attack [23]. To tackle this issue, Natoli et al.[22] suggested the use of Ethereum-based functions (e.g., SendIfReceived) to enforce the order of transactions. Similarly, Luu et al.[23] suggested using a guard condition such that "a contract code either returns the expected output or fails". A tool called 'OYENTE' developed by [23] can be used to detect contracts that are vulnerable to transaction-ordering dependency.

The second issue is timestamp dependency [23]. This problem occurs when a contract uses the block timestamp as a condition to trigger and execute transactions (e.g., sending money). For instance, a game-based contract that takes the block timestamp as a random seed to select the winner. The block timestamp is usually set as the current local time by the miner who generated the block. However, an issue with the timestamp is that a dishonest miner could vary its value by about 15 minutes from the current time, while the block is still accepted by the blockchain system. As the timestamp of a block is not guaranteed to be accurate, contracts that rely on timestamp value are vulnerable to threats by dishonest miners. To tackle this issue, Luu et al.[23] suggested using the block number as a random seed for contracts instead of using the block timestamp. This is because the value of the block number is fixed (miners cannot vary the block number value). To detect contracts that are vulnerable to timestamp dependency, 'OYENTE' tool presented in [23] can be used.

The third issue is mishandled exception vulnerability [23]. This problem occurs when a contract (caller) calls another contract (callee) without checking the value returned by the callee. When calling another contract, an exception (e.g., run out of gas) sometimes raised in the callee

contract. This exception, however, might/might not be reported to the caller depending on the construction of the call function. Having not reported an exception might lead to threats as in the KingOfTheEther (KoET) contract [23]. In KoET, an adversary might send a transaction that results in an exception in order to buy the throne from the current king for free. To tackle this issue, Luu et al.[23] highlighted the importance of checking the value returned by the callee. In the KoET example, the code can be improved to not release the throne till the payment from the adversary is completed successfully without any exception. The 'OYENTE' tool proposed by [23] can be used to detect mishandled exception vulnerability in smart contracts.

The fourth issue is re-entrancy vulnerability [23]. This problem occurs when an attacker utilises a recursive call function to conduct multiple repetitive withdrawals, while their balances are only deduced once. In June 2016, an attacker utilised the re-entrancy vulnerability in the Decentralised Autonomous Organisation (DAO) to steal over 60 million US dollars [23]. Luu et al. [23] developed a tool called 'OYENTE' to detect this vulnerability.

The fifth issue is criminal activities. Jules et al. [24] highlighted the feasibility of constructing three different types of criminal activities in smart contract systems, namely, "leakage/sale of secret documents, theft of private keys and calling-card crimes, a broad class of physical-world crimes (murder, arson, etc.)" [24]. These crimes can be implemented efficiently in the Ethereum blockchain by utilising cryptographic techniques as follows. Leakage of secret documents can be achieved with the support of Serphent (an Ethereum scripting language). Theft of private keys can be achieved using Succinct Non-interactive ARgument of Knowledge (SNARKs) cryptographic primitives. Authenticated data feeds, which is data from an external party, can facilitate the calling-card crimes. The authors of [24], however, did not attempt to tackle those crime activities, but instead, they highlighted the importance of constructing safeguards against such activities.

The last issue is the lack of trustworthy data feeds (oracles) [25]. As we explained in Section 2.2, some smart contracts require information (data feeds) from outside the blockchain. The problem is that there is no guarantee that the information provided by an external source is trustworthy. In an attempt to tackle this issue, Zhang et al. [25] built a Town Crier (TC) solution that acts as a trusted third party between external sources and smart contracts to provide authenticated data feeds for smart contracts. Figure 5 explains the architecture of TC solution. The TC solution consists of a TC contract that resides on the blockchain and a TC server that resides outside the blockchain. To send a data feeds request, a user contract can send a request to the TC contract, which will then be forwarded to the TC server. The server then communicates with external data sources via HTTPS to get the data feeds. Upon getting the required data feeds, the server will forward those feeds to the TC contract, which will then be forwarded to the user contract.

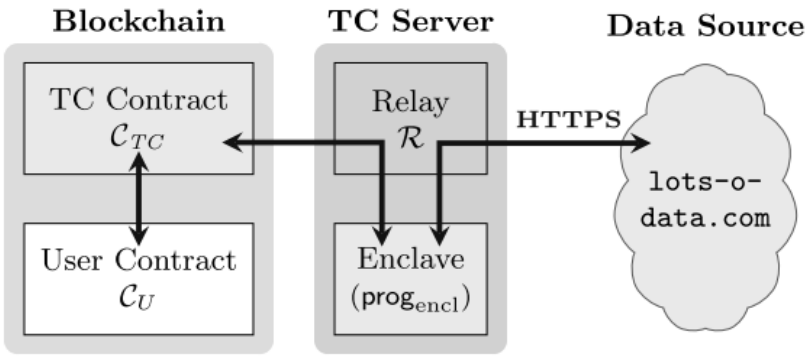

Figure 4. Architecture of TC solution [25].

Privacy issues

From the literature, we found two privacy issues, namely, the lack of transactional privacy and the lack of data feeds privacy.

The first issue is the lack of transactional privacy [26,27]. In blockchain systems, all transactions and users' balances are publicly available to be viewed. This lack of privacy could limit the adoption of smart contracts as many people consider financial transactions (e.g., stock trading) as confidential information [26]. To tackle this issue, Kosba et al.[26] built a tool called 'Hawk' that allows developers to write privacy-preserving smart contracts without the need of implementing any cryptography. The tool is responsible for compiling smart contract code to privacy-preserving one. Watanabe et al.[27] proposed to encrypt smart contracts before deploying them to the blockchain. Only participants, who are involved in a contract, can access the contract's content by using their decryption keys.

The second issue is the lack of data feeds privacy [25]. When a contract requires data feeds to operate, it sends a request to the party that provides those feeds. However, this request is exposed to the public as anyone in the blockchain can see it. To tackle this issue, Zhang et al. [25] extend their Town Crier (TC) tool to support private requests. A contract can encrypt the request using the TC's public key, before sending the request. Upon receiving the encrypted request, the TC can decrypt it using its private key. Thus, this would guarantee that the content of the request is kept secret from other users/contracts in the blockchain.

Performance issues

From the literature, we only found one performance issue, which is the sequential execution of smart contracts [28]. In blockchain systems, smart contracts are executed sequentially (e.g., one contract at a time). However, this would affect the performance of the blockchain systems negatively as the number of smart contracts that can be executed per second will be limited. With the growing number of smart contracts in the future, the blockchain systems will not be able to scale. Vukolić [28] suggested to execute smart contracts in parallel as long as they are independent (e.g., "do not update the same variables" [28]). By doing so, the performance of blockchain systems would be improved as more contracts can be executed per second.

Other topics

Apart from smart contract issues, we found nine papers from the literature that propose smart contract applications or discuss other smart contract related topics.

There are four smart contract applications proposed in the literature, namely, trading and fair exchange, identity management, Internet of Thing and agreements establishment applications. For trading and fair exchange, Bogner et al. [30] developed a smart contract application on top of the Ethereum blockchain to allow untrusted participants to share everyday objects (e.g., rent devices). For identity management, Al-Bassam et al. [31] built a system called `SCPKI' on top of the Ethereum blockchain to overcome the limitations (e.g, centralisation and lack of transparency) of the Public Key Infrastructure. This system allows entities to manage their identities in a transparent way without the involvement of a trusted third party such as central authorities. For the Internet of Thing, Huh et al. [32] used Ethereum smart contracts to define and manage the behaviours of a few devices under specified conditions. For example, an air conditioner that switches to energy saving mode when the usage of electricity reaches 170 KW. For agreements establishment, Carrillo et al. [33] developed an application that allows two untrusted parties (e.g., consumer and provider) to negotiate and then establish an agreement as a contract.

In addition to smart contract applications, there are different topics that were discussed in the literature. In [12], the authors discussed how the combination of blockchain-based smart contracts with the Internet of Thing could be powerful in terms of facilitating the sharing of services. In [9],

the authors discussed the possibility of applying blockchain-based smart contracts for licensing management. For example, the use of smart contracts to control the license of software products. In [14], the authors investigated the possibility of creating complex smart contracts without relying on scripts. In [34], the authors proposed a new consensus method called 'credibility' for contracts management (e.g., digital right management) to avoid the limitations of existing consensus methods. In [35], the authors proposed a semantic index approach to search for information in the Ethereum blockchain.

5. DISCUSSION

This section discusses the study results and answers the research questions that we defined in Section 3.

RQ1: What are the current research topics on smart contracts?

The results of this systematic mapping study showed that most of the current research on smart contracts is about identifying and tackling smart contract issues. Four different issues were identified, namely, codifying, security, privacy and performance issues. Codifying and security issues were among the most discussed issues. This is because smart contracts store valuable currency units and any security breach or coding error could result in losing money. The identified codifying issues are the difficulty of writing correct codes, the inability to modify or terminate contracts, the lack of support to identify under-optimised contracts and the complexity of programming languages. The identified security issues are transaction-ordering dependency, timestamp dependency, mishandled exception, re-entrancy, untrustworthy data feeds and criminal activities. The identified privacy issues are the lack of transactional privacy and the lack of data feeds privacy. The identified performance issue is the sequential execution of smart contracts. Although there are some proposed solutions to tackle these issues, some of them are only abstract ideas without including any concrete evaluation. A few others are still not tackled yet. For example, the solution proposed by [21] is only a suggestion to use alternative programming languages without any implementation. Criminal activities identified by [24] are still not overcome yet.

Other research proposed smart contract applications or studied other smart contract related topics. The proposed applications are trading and fair exchange, identity management, Internet of Thing and agreements establishment. The studied topics are combining smart contracts with the Internet of Thing and licensing management, studying scripting languages for smart contracts, proposing new consensus methods and proposing an indexing approach to search for useful information in blockchain systems.

RQ2: What are the current smart contract applications?

Smart contract applications are solutions that have been developed on top of blockchain technology. We identified some smart contract applications developed on top of the Ethereum blockchain. Those applications are to allow untrusted participants to share everyday objects, establish an agreement as a contract, manage their identities and control the behaviours of the Internet of Thing devices. Furthermore, we identified other applications that were built as a smart contract tool on top of the blockchain to detect or tackle codifying, security and privacy issues. Some of these tools are 'GASPER', 'OYENTE', 'HAWK' and 'Town Crier'.

RQ3: What are the research gaps that need to be addressed in future studies?

From this systematic mapping study, we identified a number of research gaps in smart contract research that can be studied by future research. The methodologies used to identify those gaps are as follows. First, observing issues or limitations from the papers included in this study (e.g., gaps

number 2, 3 and 5). Second, recognising issues that were highlighted by the papers included in this study, but still are not solved yet (e.g., gaps number 1 and 4).

The first one is the lack of studies on scalability and performance issues. The sequential execution of smart contracts affects the ability of blockchain systems to scale as we discussed in Section 4.2. With the growing number of smart contracts in the future, this issue will increase further. The author of [28] described a very high-level solution, which is parallel execution of contracts, without any concrete evaluation. Parallel execution of contracts faces a challenge in how to execute contracts that depend on each other at the same time. It is, therefore, essential to conduct research on identifying and tackling performance issues to ensure the ability of blockchain to scale.

The second gap is that almost all current research is discussing smart contracts on the Ethereum blockchain, although there are some other blockchains (e.g., NXT and Eris) that can support the creation of smart contracts. Different blockchains have distinctive features and advantages. Thus, future research might investigate different implementations of blockchain to deploy and run smart contracts.

The third gap is the small number of smart contract applications. Although the concept of smart contract has gained a lot of attention, there are only a few applications developed by the literature. This is because smart contract concept is still in its infancy stage. Banasik et al.[14] claimed that smart contracts are not widely common in practice. For future research, therefore, researchers could consider studying various potential applications such as e-commerce and cloud storage.

The fourth gap is the lack of research on tackling criminal activities in smart contracts. The author of [24] only identified three types of criminal activities that can be conducted on smart contracts without proposing any solution to them. Thus, future research could focus on identifying more types of criminal activities and proposing solutions to overcome them.

The last gap is the lack of high quality peer-reviewed research on smart contracts. Most of the research is conducted as blog articles or grey literature without providing great contributions. There is, therefore, a need for high quality publications on smart contracts.

6. CONCLUSION

Blockchain technology is a distributed database that records all transactions that have ever occurred in the network. The main feature of blockchain is that it allows untrusted parties to communicate between each other without the need of a trusted third party. Different distributed applications beyond cryptocurrencies can be deployed on top of blockchain. One of these applications is smart contracts, which are executable codes that facilitate, execute and enforce an agreement between untrusted parties. Ethereum is currently the most common blockchain platform for developing smart contracts, although there are some other available platforms.

To understand current topics on smart contracts, we decided to conduct a systematic mapping study. The main aim of this systematic mapping study was to identify and map research areas related to smart contracts. By doing so, we were able identify research gaps that need to be addressed in future studies. The focus of this study was on smart contracts from a technical point of view. Thus, we excluded studies with different perspectives (e.g., papers with an economic perspective). We extracted 24 papers from different databases. We found that most papers identifying and tackling issues on smart contracts. We grouped these issues into four categories, namely, codifying, security, privacy and performance issues. The rest of the papers focuses on proposing smart contract applications or discussing other smart contract related topics.

In this paper, we presented a few research gaps in smart contract research that need to be addressed in future studies. The identified gaps are the lack of studies on scalability and performance issues, the lack of studies on deploying smart contracts on different blockchain platforms other than Ethereum, the small number of the proposed smart contract applications, the

lack of studies on criminal activities in smart contracts and the lack of high quality research on smart contracts. These identified gaps could be studied by researchers as future works.

REFERENCES

[1] V. Buterin, “A next-generation smart contract and decentralized application platform.," Available online at: https://github.com/ethereum/wiki/wiki/White-Paper/ [Accessed 19/02/2017].

[2] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engineering," in Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, pp. 68-77, BCS Learning & Development Ltd., 2008.

[3] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system," 2008.

[4] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, “The blockchain as a software connector," in 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 182-191, IEEE, 2016.

[5] V. Buterin, “On public and private blockchains," Available online at: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ [Accessed 01/03/2017].

[6] N.Szabo, “Formalizing and securing relationships on public networks.," Available online at: http:// rstmonday.org/ojs/index.php/fm/article/view/548/4691 [Accessed 15/02/2017].

[7] J. Stark, “Making sense of blockchain smart contracts," Available online at: http://www.coindesk.com/making-sense-smart-contracts/ [Ac-cessed 06/03/2017].

[8] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, “Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in International Conference on Financial Cryptography and Data Security, pp. 79-94, Springer, 2016.

[9] V. Morabito, “Smart contracts and licensing," in Business Innovation Through Blockchain, pp. 101-124, Springer, 2017.

[10] A. Lewis, ”A gentle introduction to smart contracts," Available online at: https://bitsonblocks.net/2016/02/01/a-gentle-introduction-to-smart-contracts/ [Accessed 25/02/2017].

[11] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, 2014.

[12] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things," IEEE Access, vol. 4, pp. 2292-2303, 2016.

[13] W. Egbertsen, G. Hardeman, M. van den Hoven, G. van der Kolk, and A. van Rijsewijk, “Replacing paper contracts with ethereum smart contracts," 2016.

[14] W. Banasik, S. Dziembowski, and D. Malinowski, “Efficient zero-knowledge contingent payments in cryptocurrencies without scripts," in European Symposium on Research in Computer Security, pp. 261-280, Springer, 2016.

[15] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, “Where is current research on blockchain technology?la systematic review," PloS one, vol. 11, no. 10, p. e0163477, 2016.

[16] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, et al., “Formal verification of smart contracts: Short paper," in Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, pp. 91-96, ACM, 2016.

[17] G. Bigi, A. Bracciali, G. Meacci, and E. Tuosto, “Validation of decentralised smart contracts through game theory and formal methods," in Programming Languages with Applications to Biology and Security, pp. 142-161, Springer, 2015.

[18] C. K. Frantz and M. Nowostawski, “From institutions to code: Towards automated generation of smart contracts," in 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W), pp. 210-215, IEEE, 2016.

[19] B. Marino and A. Juels, “Setting standards for altering and undoing smart contracts," in International Symposium on Rules and Rule Markup Languages for the Semantic Web, pp. 151-166, Springer, 2016.

[20] T. Chen, X. Li, X. Luo, and X. Zhang, “Under-optimized smart contracts devour your money," in 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 442-446, IEEE, 2017.

[21] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, “Evaluation of logic-based smart contracts for blockchain systems," in International Symposium on Rules and Rule Markup Languages for the Semantic Web,167-183, Springer, 2016.

[22] C. Natoli and V. Gramoli, “The blockchain anomaly," in 15th International Symposium on Network Computing and Applications (NCA), 310-317, IEEE, 2016.

[23] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pp. 254-269, ACM, 2016.

[24] A. Juels, A. Kosba, and E. Shi, “The ring of gyges: Investigating the future of criminal smart contracts," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pp. 283-295, ACM, 2016.

[25] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, “Town crier: An authenticated data feed for smart contracts," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pp. 270-282, ACM, 2016.

[26] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in 2016 IEEE Symposium on Security and Privacy (SP),839-858, IEEE, 2016.

[27] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. J. Kishigami, “Blockchain contract: A complete consensus using blockchain," in 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE), pp. 577-578, IEEE, 2015.

[28] M. Vukolić, “Rethinking permissioned blockchains," in Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, BCC '17, pp. 3-7, ACM, 2017.

[29] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on ethereum smart contracts (sok)," in International Conference on Principles of Security and Trust, pp. 164-186, Springer, 2017.

[30] A. Bogner, M. Chanson, and A. Meeuw, “A decentralised sharing app running a smart contract on the ethereum blockchain," in Proceedings of the 6th International Conference on the Internet of Things, pp. 177-178, ACM, 2016.

[31] M. Al-Bassam, “Scpki: A smart contract-based pki and identity system," in Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, BCC '17, pp. 35-40, ACM, 2017.

[32] S. Huh, S. Cho, and S. Kim, “Managing IoT devices using blockchain platform," in 2017 19th International Conference on Advanced Communication Technology (ICACT), pp. 464-467, IEEE, 2017.

[33] P. N. Carrillo, C. I. Peña, and J. L. d. L. Rosa, “Eurakos next: a cryptocurrency based on smart contracts," in Ebook: Artificial Intelligence Research and Development, vol. 288 of Frontiers in Artificial Intelligence and Applications, pp. 221-226, 2016.

[34] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. Kishigami, “Blockchain contract: Securing a blockchain applied to smart contracts," in 2016 IEEE International Conference on Consumer Electronics (ICCE), pp. 467-468, IEEE, 2016.

[35] A. Third and J. Domingue, “Linked data indexing of distributed ledgers," in Proceedings of the 26th International Conference on World Wide Web Companion, WWW '17 Companion, pp. 1431-1436, 2017.