Faculty of
Computer & Informatics Eng.
Department of Software &
Information Systems Eng.



Crowdfunding Platform Based on Smart Contract

A report submitted in partial fulfillment of the requirements for the degree of Bachelor's Degree in Software and Information System Engineering for the undergraduate students

Prepared by

Ahmad Yones

Fandi Alsous

Ibrahim Hafez

Supervised by

Prof. Wassim Juneidi

Academic Year

2022 – 2023

SUPERVISION CERTIFICATION

I certify that the preparation of this project entitled **Crowdfunding Platform Based on Smart Contract** prepared by **Ahmad Yones**, **Fandi Alsous**, and **Ibrahim Hafez** was made under my supervision at *Department of Software & Information System Eng. – Faculty of Informatics Engineering* in partial fulfillment of the Requirements for the Degree of *Bachelors of Software & Information System Engineering*.

X

Prof. Wassim Juneidi
Supervisor
Saturday, January 21, 2023

ABSTRACT

Crowdfunding applications have become very popular in many ways across the global economy. They are used to source for charities, cover legal fees, fund non-profits, and invest in new products and services. The primary issue with many crowdfunding applications is lack of transparency or decentralization, making it an ideal blockchain candidate.

In this project, we have highlighted the distinction between the traditional crowdfunding platform as well as blockchain network-based crowdfunding platform. This project highlights the issues and challenges faced by the industries, as mentioned earlier, by using the traditional methods as well as the solutions to the problems provided by the blockchain network-based systems to those industries. This project helps the people to understand the benefits of blockchain network-based systems in their respective industries as well as execute it to improve the transparency, efficiency, and security of the system altogether.

Keywords – Crowdfunding, **Blockchain**, **Campaign**, **Smart Contract**, **Request-Approval**, **Consensus**.

ملخص

أصبحت تطبيقات التمويل الجماعي شائعة جدًا من نواح كثيرة عبر الاقتصاد العالمي. يتم استخدامها كمصدر للجمعيات الخيرية، وتغطية الرسوم القانونية، وتمويل المؤسسات غير الربحية، والاستثمار في منتجات وخدمات جديدة. المشكلة الأساسية في العديد من تطبيقات التمويل الجماعي هي الافتقار إلى الشفافية أو اللامركزية، مما يجعلها مرشحًا مثاليًا لتنفيذها بتقنية سلسلة الكتل.

في هذا المشروع، قمنا بتسليط الضوء على الفرق بين منصة التمويل الجماعي التقليدية ومنصة التمويل الجماعي القائمة على شبكة سلسلة الكتل. كما يسلط هذا المشروع الضوء على المشكلات والتحديات التي تواجهها الصناعات، كما ذكرنا سابقًا، باستخدام الأساليب التقليدية بالإضافة إلى حلول المشكلات التي توفرها الأنظمة القائمة على شبكة سلسلة الكتل لتلك الصناعات. يساعد هذا المشروع الأشخاص على فهم فوائد الأنظمة المستندة إلى شبكة سلسلة الكتل في الصناعات الخاصة بهم وكذلك تنفيذها لتحسين شفافية وكفاءة وأمن النظام تمامًا.

Dedication

I dedicate this graduation project to all the people who have been part of my journey. To my professors and mentors, for their guidance and support throughout my years in college; to my family, my friends, and my classmates, for their encouragement, friendship, and understanding; and to myself, for persisting through the toughest of times and never giving up.

This project is a culmination of years of hard work, dedication, and determination to reach the finish line. It is a testament to the importance of setting lofty goals and working hard to achieve them, regardless of the obstacles. It is an acknowledgement of the journey I have taken to get to where I am today, and a reminder that hard work pays off in the end.

This project is dedicated to those who have believed in me and guided me along the way. It is dedicated to the times I have pushed myself to the limit and gone beyond what I thought was possible. It is dedicated to the moments of success and the moments of failure, for they have taught me more than any victory ever could.

It is a celebration of my accomplishments, and a reminder to never give up on my dreams. I am so grateful for the support and encouragement I have received throughout my journey, and I thank you all from the bottom of my heart.

Acknowledgments

I would like to express my sincere gratitude to my academic advisor *Prof. Wassim Juneidi*, and my academic mentor *Prof. Mouhib Alnoukari* for their invaluable guidance, encouragement and support throughout this project.

I would also like to thank all the faculty members of *Faculty of Informatics Engineering* in *Syrian Private University*, who have provided me with timely advice and support in the development of this project.

I am also thankful to my friends and family for offering assistance and encouragement whenever I needed it most.

I would like to thank my classmates and lab mates for their unwavering support throughout the project.

Last but not least, I would like to thank all those anonymous people who have offered valuable feedback and insight into the project. Their assistance has made this project possible.

Once again, I would like to express my deepest appreciation and gratitude towards all the people mentioned above for their support and assistance throughout the project.

Table of Contents

ABSTRACT III

Dedication V

Acknowledgments VII

List of Tables X

List of Figures XI

List of Abbreviations XII

Chapter 1 Introduction 1

1.1. Introduction 2

1.2. Problem Definition 4

.1.3 Referral Study and Review 5

1.4. Project Objectives 8

1.5. Stakeholders 10

1.6. Concepts 11

Chapter 2 Project Analysis and Design 13

2.1. Methodology 14

2.2. Requirements Analysis 18

2.2.1. Functional Requirements 18

2.2.2. Non-Functional Requirements 18

2.2.3. Usecase Diagram 19

2.2.4. Sitemap 20

2.3. Project Design 21

Chapter 3 Implementation 23

3.1. Architecture 24

3.2. Technologies and Tools 36

3.3. Front-end Project 37

3.3.1. Running Project 37

.3.3.2 Project Structure 37

3.3.3. Work Samples 40

3.4. Campaign Project 46

3.4.1. About The Project 46

3.4.2. Project Structure 47

.3.4.3 Code Sample 48
Chapter 4 Conclusion, and Future Improvements 55
4.1. Results 56
.4.2 Conclusion 57
4.3. Future Work and Improvements 58
References 59

List of Tables

Table 1.1 Stakeholders..... 10

List of Figures

Figure 2.1 Proof of Work 16
Figure 2.2 Hashed Block 16
Figure 2.3 Usecase Diagram 19
Figure 2.4 Sitemap 20
Figure 2.5 Campaign Design 21
Figure 2.6 Campaign UML Diagram 22
Figure 3.1 Web2.0 Architecture 25
Figure 3.2 Web3.0 Architecture 27
Figure 3.3 JSON RPC Web3.0 Architecture 30
Figure 3.4 Metamask Web3.0 Architecture 32
Figure 3.5 Front-end Project Structure 37
Figure 3.6 Front-end Pages Folder 38
Figure 3.7 Front-end State Context Code 39
Figure 3.8 Registration Page 40
Figure 3.9 Login Page 41
Figure 3.10 Campaign Creation Page 42
Figure 3.11 My Campaign Page 42
Figure 3.12 My Contributions Page 43
Figure 3.13 Pending Campaigns Page 44
Figure 3.14 Request Creation Page 44
Figure 3.15 Approving Page 45
Figure 3.16 Contract Design 46
Figure 3.17 Contract Project Structure 48
Figure 3.18 Testing (1) 49
Figure 3.19 Testing (2) 50
Figure 3.20 Testing (3) 50
Figure 3.21 Testing (4) 51
Figure 3.22 Testing (5) 52
Figure 3.23 Testing (6) 53
Figure 3.24 Testing (7) 54
Figure 3.25 Testing (8) 54

List of Abbreviations

Abbreviation	Definition
ABI	Application Binary Interface
ACH	Automated Clearing House
B2B	Business to Business
B2C	Business to Consumer
CA	Certificate Authority
CRDT	Conflict-free Replicated Data Type
CVN	Card Verification Number
DA	Decentralized Apps
DB	Database
EDI	Electronic Data Interchange
ETH	Ethereum
EVM	Ethereum Virtual Machine
ICO	Initial Coin Offering
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
JWT	JSON Web Token
P2P	Peer to Peer
PKI	Public Key Infrastructure
RFC	Request for Comment
RPC	Remote Procedure Call
SDK	Software Development Kit
TLS	Transport Layer Security
UI	User Interface
VM	Virtual Machine
WWW	World Wide Web

Chapter 1

Introduction

1.1. Introduction

The blockchain is an incorruptible digital ledger that records every transaction. It is a distributed system thus all the records are stored in every node in the decentralized network. Ethereum allows running applications in the blockchain called Smart Contracts. All the Smart contracts are run on the Ethereum Virtual Machine.

Crowdfunding provides an easy way to find cash for innovative project ideas. The problem with the current crowdfunding companies charging high fees and sometimes there were scams happened. Implementing a crowdfunding strategy in blockchain will help to avoid these types of problems. By incorporating Peer to Peer smart contract for crowdfunding remove the traditional transaction fees and platforms fees normally associated with other crowdfunding platforms, such as *Kickstarter*.

The objective of our project is to create a reliable application so that every new idea gets life. We have designed a crowdfunding site which is a blockchain based website. We provide an easy-to-use interface for everyone to create and post their ideas on this application. These ideas then become public to everyone. Anyone who wishes to support their ideas can contribute. All these processes are done in an interactive manner.

Crowdfunding has become an increasingly popular way for individuals and organizations to raise funds for their projects and ventures. However, traditional crowdfunding platforms can be limited in terms of transparency, security, and flexibility. Blockchain technology has the potential to revolutionize the crowdfunding industry by providing a decentralized, transparent, and secure platform for raising funds.

A blockchain-based crowdfunding platform operates on a decentralized network, which allows for direct interactions between project creators and backers without the need for intermediaries. This can provide more transparency in the funding process, allowing backers to easily verify the authenticity and progress of a project. Additionally, the use of smart contracts and digital tokens can provide a more secure way to handle financial transactions and to prevent fraud and mismanagement.

Blockchain-based crowdfunding platforms can also offer more flexibility in terms of the types of projects that can be funded. Traditional platforms are mainly designed for creative

projects such as films, music albums, and art projects, while blockchain-based platforms can support a wide range of projects, including those that involve smart contracts and tokens.

Another important aspect of blockchain-based crowdfunding is the tokenization of assets, which allows for the easy transfer of ownership or access rights. This can provide more ongoing participation and engagement between project creators and backers and can create a more engaged and invested community of backers who can continue to support the project and its creator in the future.

In addition, blockchain-based crowdfunding platforms can provide a way to comply with legal and regulatory requirements and can be highly scalable, which allows them to handle a large number of transactions and users. This can make the process of raising funds more cost-effective for project creators, who can save on intermediaries such as payment processors and legal fees.

Overall, blockchain-based crowdfunding has the potential to provide a more transparent, secure, and flexible platform for raising funds, which can benefit both project creators and backers. However, it's important to keep in mind that like any new technology, blockchain-based crowdfunding is still in its early stages and there are some challenges that need to be addressed, such as the lack of regulation, security risks and the need for more user-friendly interfaces.

1.2. Problem Definition

The problem of developing a crowdfunding application is the design and implementation of a platform that allows individuals or organizations to raise funds for a specific project or venture through the collective effort of a large number of people. This platform must be user-friendly, secure, and transparent, allowing project creators to present their ideas and proposals in an effective manner, and allowing backers to make informed decisions about which projects to support. Additionally, the platform must have the capacity to process and manage financial transactions in a safe and efficient manner, and to ensure the proper disbursement of funds to project creators.

In order to be successful, the crowdfunding application must also address potential issues such as fraud and mismanagement, lack of transparency and accountability, and difficulty in verifying the authenticity and progress of a project. It must also provide a variety of features and tools that can help project creators to promote their projects and engage with their backers.

Moreover, the platform should be able to cope with legal and regulatory requirements, which can vary greatly depending on the jurisdiction, should be able to handle a large number of transactions and users and should be able to provide an easy and accessible way to invest and track the progress of the projects.

1.3. Referral Study and Review

Kickstarter is a well-known crowdfunding platform that has been in operation since 2009. It is a centralized platform that allows creators to present their ideas and proposals to a wide audience, and to solicit funding from backers in exchange for rewards or equity.

Kickstarter could be used as a referral study for blockchain-based crowdfunding in several ways. Firstly, it highlights the importance of having a user-friendly and visually appealing platform that can effectively showcase projects to potential backers. Secondly, it illustrates the need for a secure and transparent system for handling financial transactions. Finally, it shows the importance of having a system in place to prevent fraud and mismanagement.

However, Kickstarter has a few limitations that could be addressed by blockchain-based crowdfunding. One of these limitations is the lack of transparency and accountability in the funding process, as well as the difficulty of verifying the authenticity and progress of a project. Additionally, it operates as a centralized platform, which can make it vulnerable to hacking or other forms of data breaches.

On the other hand, a blockchain-based crowdfunding platform would enable the use of smart contracts, which can provide more transparency and accountability in the funding process, while also providing a tamper-proof record of all transactions. This can increase trust between project creators and backers, and make it easier to verify the authenticity and progress of a project. Additionally, the use of digital tokens and decentralized platform can provide a more secure and efficient way to handle financial transactions and to prevent fraud and mismanagement.

Another key difference between Kickstarter and blockchain-based crowdfunding platforms is the level of flexibility and scalability. Kickstarter is mainly designed for creative projects such as films, music albums, and art projects. On the other hand, blockchain-based crowdfunding platforms can support a wide range of projects, including those that involve smart contracts and tokens, which allows for more diversity in the types of projects that can be funded.

Additionally, Kickstarter operates on a "all or nothing" model, meaning that if a project does not reach its funding goal, backers will not be charged and the project creator will not receive any funds. This can be a disadvantage for some projects that may not be able to reach their funding goal. On the other hand, blockchain-based crowdfunding platforms can offer different funding models, such as the "keep what you raise" model, where project creators can keep any funds they raise, even if they don't reach their funding goal.

Lastly, Kickstarter operates on a per-project basis, meaning that once a project is funded and completed, backers do not have any further involvement or rights. On the other hand, blockchain-based crowdfunding platforms can allow for ongoing participation and engagement between project creators and backers, through the use of digital tokens or other mechanisms. This can create a more engaged and invested community of backers who can continue to support the project and its creator in the future.

❖ **How Kickstarter implement the payment?**

Kickstarter uses a third-party payment processor, such as Stripe or Amazon Payments, to handle financial transactions on the platform. When a backer decides to support a project, they will enter their credit card information and the payment will be processed through the chosen payment processor. The funds are then held by Kickstarter until the project reaches its funding goal or the funding period ends. If the project is successfully funded, the funds are released to the project creator, minus a 5% fee taken by Kickstarter and a small fee for the payment processor.

The payment process on Kickstarter is relatively simple and straightforward, but it does rely on centralized payment processors. This can make the platform vulnerable to hacking or other forms of data breaches, and can also make it difficult for backers to verify the authenticity and progress of a project.

On the other hand, blockchain-based crowdfunding platforms can use digital currencies, such as Bitcoin or Ethereum, to handle financial transactions. This allows for a more secure and transparent way to handle funds, as all transactions are recorded on a tamper-proof ledger that is visible to all participants on the network. Additionally, blockchain-based platforms can use smart contracts to automate the disbursement of funds, which can provide more security and transparency in the funding process.

In the existing system, the problem is that the companies charge heavily to both the donor and the user. There is no track of the records of the money, transparency, communication between the investor and the user is developing the project. The trust is the main problem when it comes to the crowdfunding with the existing companies. None of these companies provide the donor guarantee policy.

- Not Transparent
- High Charges
- Donor guarantee policy not available
- No track of Records

In the proposed system, the campaign creators will post their project ideas in the campaign and the interested people will donate the fund to the project idea. Where it defers from the old crowdfunding is that all the money is now digital currencies like ether. All ether coin will be recorded and keep tracks in the blockchain. Where the blockchain is an immutable ledger. The Donor has control over the funded money. With the Request approval module, the donor has full control over the money they invested. Only if one by two of the investors need to approve the request made by the creators. By giving control on invested money the Trust is built.

- Trust
- Control over money
- No Charges
- Donor Guarantee Policy
- All transactions are recorded
- Money is Stored Securely

1.4. Project Objectives

- **Decentralization** – A key objective of blockchain-based crowdfunding is to create a decentralized platform that allows for direct interactions between project creators and backers, without the need for intermediaries.
- **Transparency** – Blockchain-based crowdfunding platforms aim to provide transparency in the funding process, allowing backers to easily verify the authenticity and progress of a project.
- **Security** – The use of blockchain technology and smart contracts can provide a more secure way to handle financial transactions and to prevent fraud and mismanagement.
- **Flexibility** – Blockchain-based crowdfunding platforms can support a wide range of projects, including those that involve smart contracts and tokens, which allows for more diversity in the types of projects that can be funded.
- **Scalability** – Blockchain-based crowdfunding platforms are designed to be highly scalable, which allows them to handle a large number of transactions and users.
- **Legal and regulatory compliance** – Blockchain-based crowdfunding platforms should provide a way to comply with legal and regulatory requirements which can vary greatly depending on the jurisdiction.
- **Tokenization** – Tokenization enables the conversion of rights to an asset into a digital token, allowing for the easy transfer of ownership or access rights.
- **Community engagement** – Ongoing participation and engagement between project creators and backers can be enabled through the use of digital tokens or other mechanisms.
- **Diversification of Investment options** – Blockchain-based crowdfunding platforms can offer different funding models, such as the "keep what you raise" model, where project creators can keep any funds they raise, even if they don't reach their funding goal, which can diversify the investment options.
- **Cost-effective** – Blockchain-based crowdfunding platforms can reduce the costs associated with intermediaries, such as payment processors and legal fees, making it more cost-

effective for project creators to raise funds.

1.5. Stakeholders

The stakeholders of a crowdfunding application include the project creators, the backers or investors, consultant who is responsible for providing feasibility study about the campaign, and the platform or company operating the crowdfunding application. Additionally, there may be intermediaries such as payment processors, and the wider community or society in which the project is located.

Table 1.1 Stakeholders

Stakeholder	Description
Campaign Creators (Owner / Manager)	These are the individuals or organizations who initiate crowdfunding campaigns in order to raise funds for a specific project or venture. They are responsible for creating and managing the campaign, including setting funding goals and deadlines, and promoting the campaign to potential investors
Investors (Donor / Contributor)	These are the individuals or organizations that contribute funds to the campaign in exchange for rewards or equity. They typically use the platform to discover and invest in new projects, and they may also have the ability to vote or govern the project's development
Platform Administrators	These are the individuals or organizations that operate and manage the blockchain-based crowdfunding platform. They are responsible for maintaining the platform's infrastructure, enforcing platform rules and regulations, and providing support to campaign creators and investors
Escrow Agents	These are entities or individuals that hold and manage the funds raised during the campaign, ensuring that they are only released to the campaign creator when certain conditions are met, such as reaching the funding goal or the campaign deadline
Financial Advisors (Consultant)	These are entities or individuals that provide legal advice to campaign creators and investors regarding compliance with laws and regulations related to crowdfunding and blockchain technology

1.6. Concepts

Crowdfunding based on blockchain technology typically involves the use of smart contracts and digital tokens. These concepts are essential for the creation of decentralized and trustless crowdfunding platforms.

- **Crowdfunding** – Crowdfunding is a method of raising funds for a project or venture by soliciting small contributions from a large number of people, typically via the internet. Crowdfunding platforms allow project creators to present their ideas and proposals to a wide audience and to solicit funding from backers in exchange for rewards or equity.
- **Blockchain** – Blockchain is a decentralized, digital ledger that records transactions across a network of computers. Each block in the chain contains a record of multiple transactions and is linked to the previous block, creating a chain of blocks that cannot be altered. This creates a tamper-proof and transparent record of all transactions on the network, enabling the secure transfer of digital assets and the creation of decentralized applications.
- **Smart Contracts** – Smart contracts are self-executing contracts with the terms of the agreement written directly into lines of code. These contracts can be programmed to automatically execute actions when certain conditions are met. For example, smart contracts can be used to automatically release funds to project creators once a certain funding goal is reached, or to automatically distribute tokens to backers.
- **Digital Tokens** – Digital tokens are digital assets that can be used to represent ownership or access rights in a crowdfunding campaign. These tokens can be issued on a blockchain network and can be traded or exchanged like traditional assets. For example, a project creator may issue tokens that represent a share of ownership in the project, or tokens that can be exchanged for access to a product or service.
- **Decentralized Platform** – A decentralized platform is a platform that operates on a blockchain network, rather than a centralized server. This allows for a more transparent and secure environment for crowdfunding, as all transactions and data are recorded on a tamper-proof ledger that is visible to all participants on the network.

- **Trustless** – Trustless refers to the ability for parties to engage in transactions without the need for a trusted intermediary. In the context of crowdfunding, this means that project creators and backers can interact directly with each other, without the need for a centralized platform to act as an intermediary.
- **Tokenization** – Tokenization is the process of converting rights to an asset into a digital token. These tokens can be traded on a blockchain network, allowing for the easy transfer of ownership or access rights. In crowdfunding, tokenization can be used to represent equity or rewards in a project.

These concepts provide the backbone of a crowdfunding platform based on blockchain, enabling the creation of a trustless, decentralized and transparent ecosystem for raising funds and investing in projects.

In a blockchain-based crowdfunding platform, a finalize request typically refers to the process of completing a funding campaign and releasing funds to the campaign creator.

When a crowdfunding campaign is created on a blockchain-based platform, the funds are typically held in escrow by a smart contract until the campaign reaches its funding goal or the campaign deadline is reached. Once the campaign is successfully funded or the deadline has passed, the campaign creator can initiate a finalize request to release the funds from escrow to their designated wallet.

The finalize request process often includes a review and approval process by the platform's administrators to ensure that the campaign met all of the necessary conditions, such as reaching its funding goal or fulfilling any other requirements specified in the smart contract.

It's important to note that the finalize request process can vary depending on the platform, as different platforms may have different rules and processes in place for releasing funds. Additionally, in some platforms, the finalize request may also include the distribution of tokens or other assets to investors as part of the campaign.

Chapter 2

Project Analysis and Design

2.1. Methodology

A. Overview

Both Crowdfunding and Cryptocurrency is a trend on the Internet and they match perfectly. Blockchain technology is one solution that can be used to reduce the problems that occur in crowdfunding. The contract is written in a way that all money will be added to the pool. When the request meets the specified condition then all the money will be transferred to the recipient.

Ethereum is an open-source, public, blockchain based distributed platform and operating to featuring smart contract functionality. It is the modified version of Bitcoin via transaction-based state transitions. Ether is a cryptocurrency which is generated and used by the Ethereum platform. Ethereum provides a decentralized operating, the Ethereum Virtual Machine (EVM), which can execute an application on the public nodes.

B. Blockchain

The blockchain is originally originated from the Bitcoin, invented by unknown people. The Blockchain is a list of continuously growing records called blocks. Each Block is linked to each other and they were secured using cryptography. Blockchain has the characteristics of integrity, decentralization, Immutability, Security, Anonymity. Blockchains can be divided into three types: 1) public blockchain (Bitcoin and Ethereum); 2) consortium blockchain (Hyperledger and Ripple) and 3) private blockchain.

C. Peer to Peer

The very important part of how blockchain works are based on Peer to Peer (P2P) system. The whole blockchain is connected to all the node in the network. This means information stored on blockchain cannot be lost or destroyed, to do so have to destroy every single node on the network and that is impossible.

D. Consensus Protocol

Consensus protocol is the most important one in the blockchain technology. The Blockchain consensus protocol is what which keeps the blocks on all the node to synchronize with each other. The term 'Consensus' means that the nodes have to agree

with the same state of the blockchain. Consensus protocol allows blockchain to be updated every minute (depends on the network) and ensures that every block in the chain is true. The aim of the consensus protocol is to guarantee a single chain is used and followed by all the nodes.

E. Proof of Work

Proof of work (abbreviated to PoW) is a consensus protocol used widely by many cryptocurrencies. This process is known as mining and the node on the network is called as miners. The Proof of Work is a mathematical problem one that requires considerable work to achieve the solution. The only way to solve the problem is through the node in the network have to run the process based on trial-and-error basis. A miner will continue testing different unique values until a suitable hash is produced. The miner who manages to solve will add next block, adding the block to the chain and validates all the transactions within it, and receiving the reward associated with the block. Fig. 4 illustrates the block.

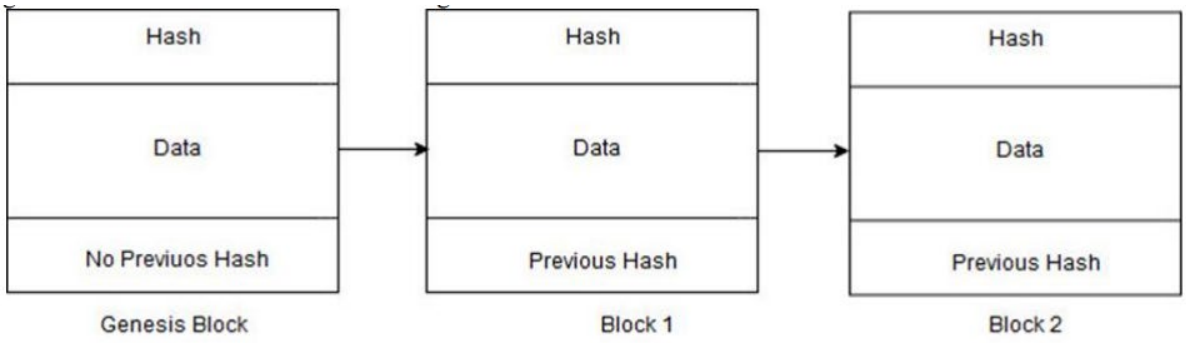


Figure 2.1 Proof of Work

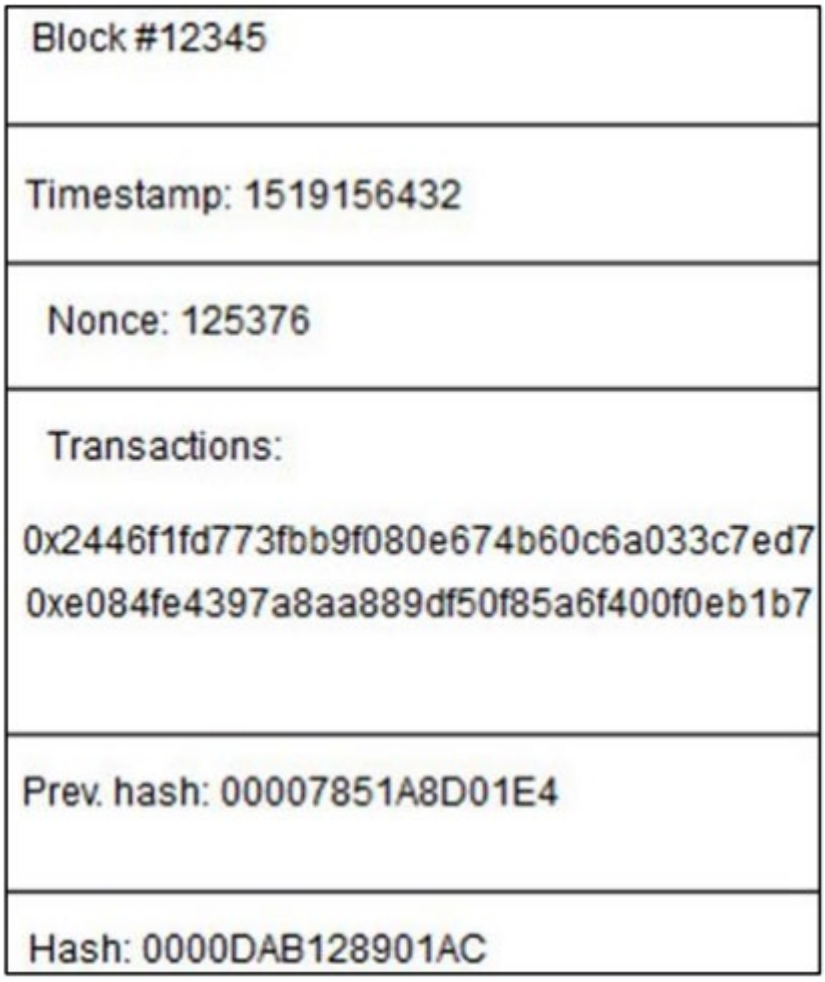


Figure 2.2 Hashed Block

F. Main Modules

Campaign Creation – Create Request Module – Request Approval Module – Finalize Module.

2.2. Requirements Analysis

2.2.1. Functional Requirements

- ✓ As a user I want to register a new account.
- ✓ As a user I want to sign in/out to/from the application.
- ✓ As a user I want to browse all deployed campaigns.
- ✓ As a user I want to create a new campaign.
- ✓ As the owner of campaign, I should be able to create new campaign's requests.
- ✓ As the owner of campaign, I should be able to finalize campaign's request.
- ✓ As a contributor, I should be able to contribute money to a campaign.
- ✓ As a contributor, I should be able to refund money from a campaign.
- ✓ As a contributor, I should be able to browse campaign's requests.
- ✓ As a contributor, I should be able to approve campaign's request.
- ✓ As a consultant, I should be able to browse pending campaigns.
- ✓ As a consultant, I should be able to approve or reject a pending campaign.

2.2.2. Non-Functional Requirements

- ✓ Security: The application must ensure that all transactions are secure and protected against any malicious attacks.
- ✓ Scalability: The application must be able to handle increasing user base and transaction volume with minimal latency.
- ✓ Availability: The application should always be available and accessible to users, even during peak usage times or when the network is congested.
- ✓ User Experience: The user interface should be intuitive and easy to use for all users.

2.2.3. Usecase Diagram

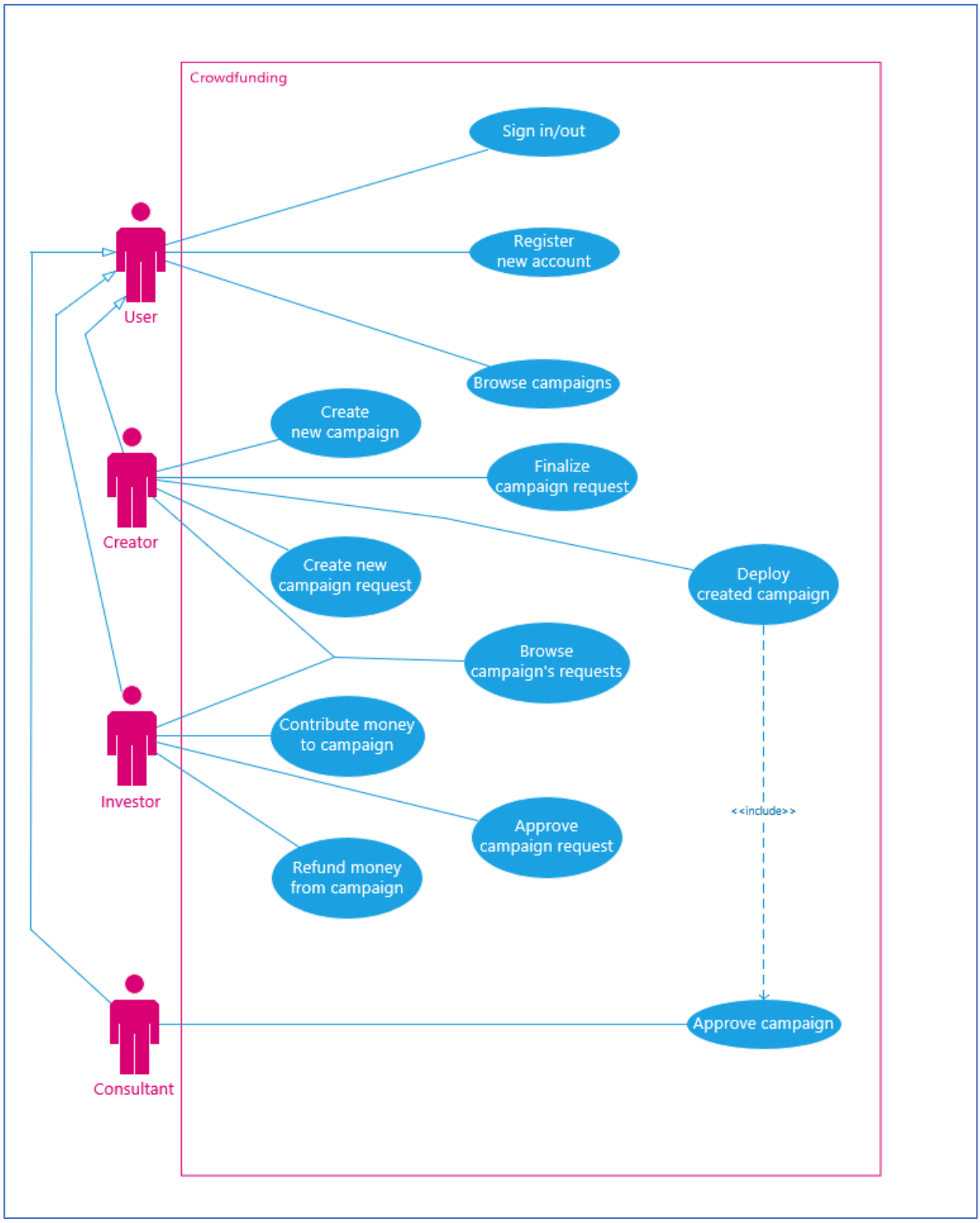


Figure 2.3 Usecase Diagram

2.2.4. Sitemap

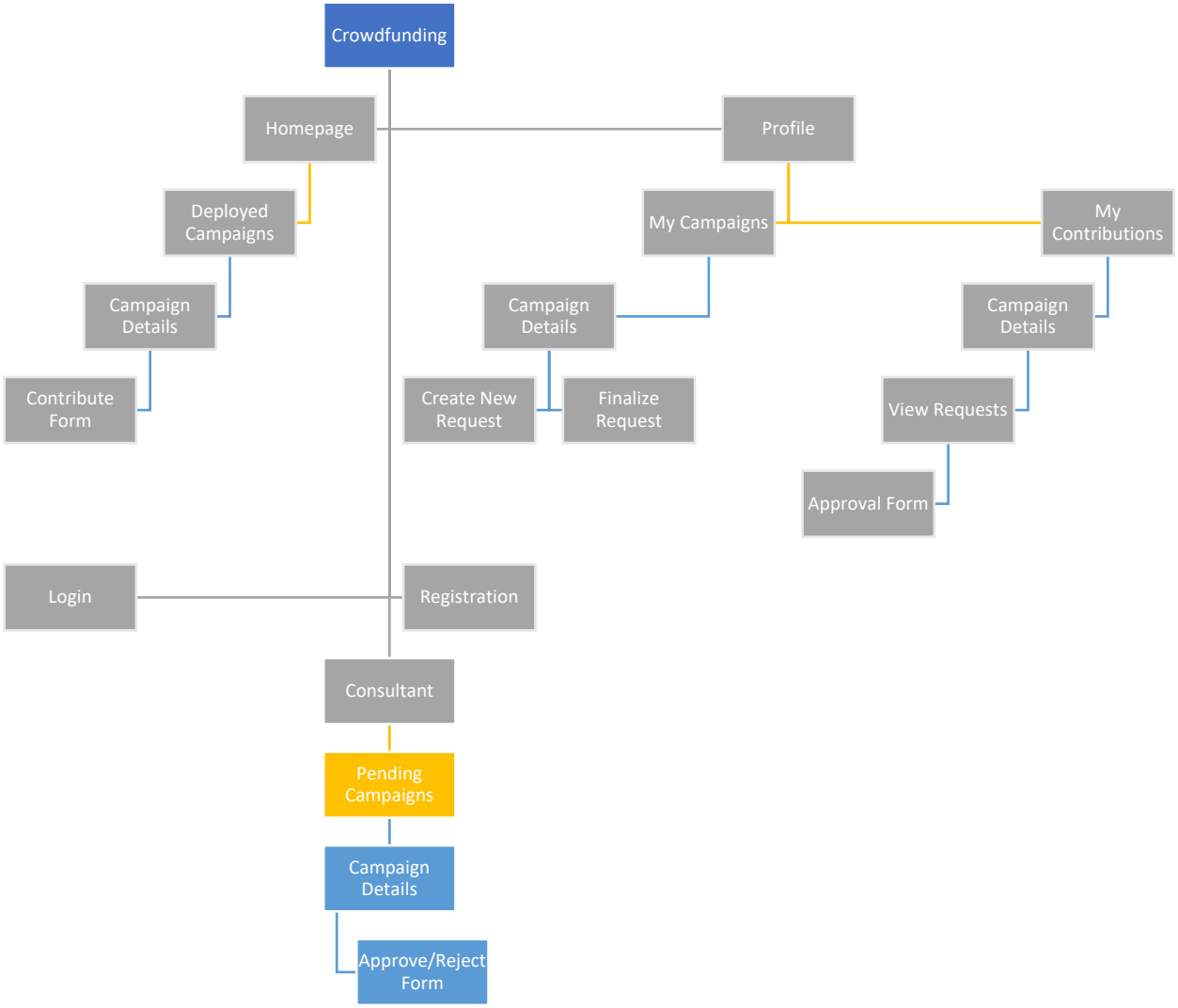


Figure 2.4 Sitemap

2.3. Project Design

All the contract code is written in solidity which is used to deploy contract in blockchain platform. The Campaign Factory is built which contains all the source code to deploy new campaign in the network. With the use of campaign factory, the new campaigns can be created. Whenever a campaign factory is deployed a one-time gas fee is needed and it is a very small amount. Initially, the new Campaign is created by giving the Idea of the project, Minimum Contribution to the project and detailed description of the project. When a new campaign is created a block will be created and added to the blockchain. Fig.1 shows the architecture of the system.

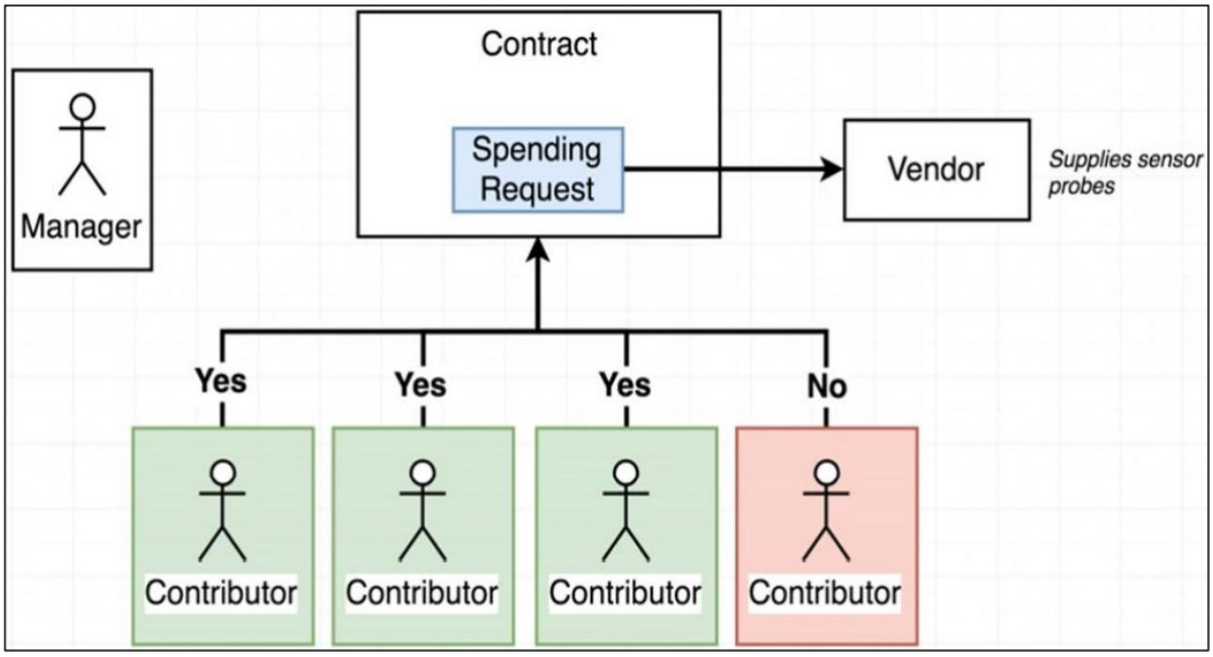


Figure 2.5 Campaign Design

The proposed system will be implemented using the solidity programming language. Solc is the solidity compiler used to compile the Campaign Factory and Campaign file into bytecode and ABI. The Bytecode will be deployed in the blockchain where the ABI is in JSON format and is used to interact with the front-end. The front-end is designed using the React Js, Next Js, and Semantic-UI. The user interactive form will be used to contribute easily. The creator or manager of the project is to request money for buying some accessories. He will create a request using the request form. This will be recorded and stored in the blockchain. All the investors

need to approve the request if it is necessary. If not, they can reject the request. Once all the investor all voted then the request will be finalized, there should be a minimum of 1/2th of the investor should be approved the request. If it meets the requirement then the money will be transferred to the vendor.

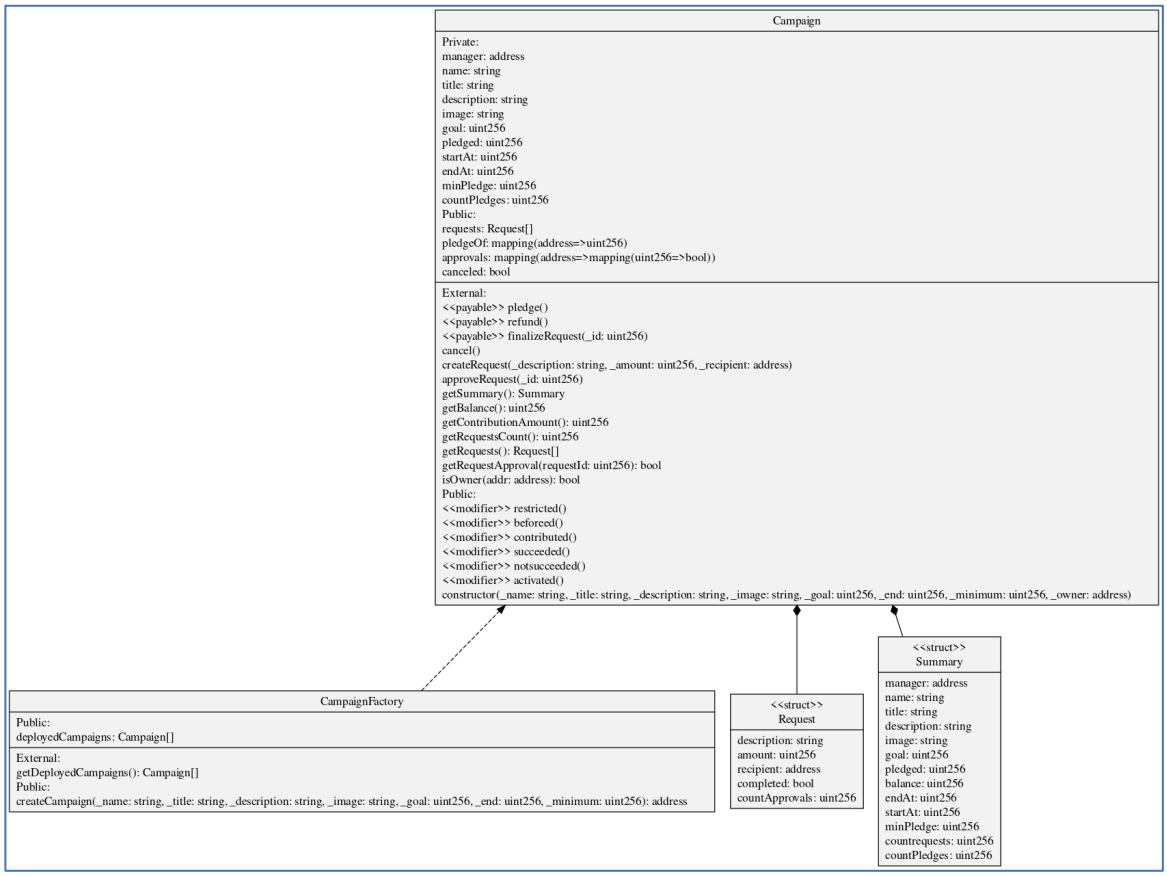


Figure 2.6 Campaign UML Diagram

Chapter 3

Implementation

3.1. Architecture

The architecture of Web 3.0 applications (or "DApps") is completely different from Web 2.0 applications.

Take Medium, for example, a simple blogging site that lets users publish their own content and interact with content from others. As a web 2.0 application, it may sound simple, but there's a lot that goes into Medium's architecture to make it all possible:

First, there must be a place to store essential data, such as users, posts, tags, comments, likes, and so on. This requires a constantly updated database.

Second, backend code (written in a language like Node.js, Java, or Python) must define Medium's business logic. For example, what happens when a new user signs up, publishes a new blog, or comments on someone else's blog?

Third, frontend code (typically written in JavaScript, HTML, and CSS) must define Medium's UI logic. For instance, what does the site look like, and what happens when a user interacts with each element on the page?

Putting it all together, when you write a blog post on Medium, you interact with its frontend, which talks to its backend, which talks to its database. All of this code is hosted on centralized servers and sent to users through an internet browser. This is a good high-level summary of how most Web 2.0 applications work today.

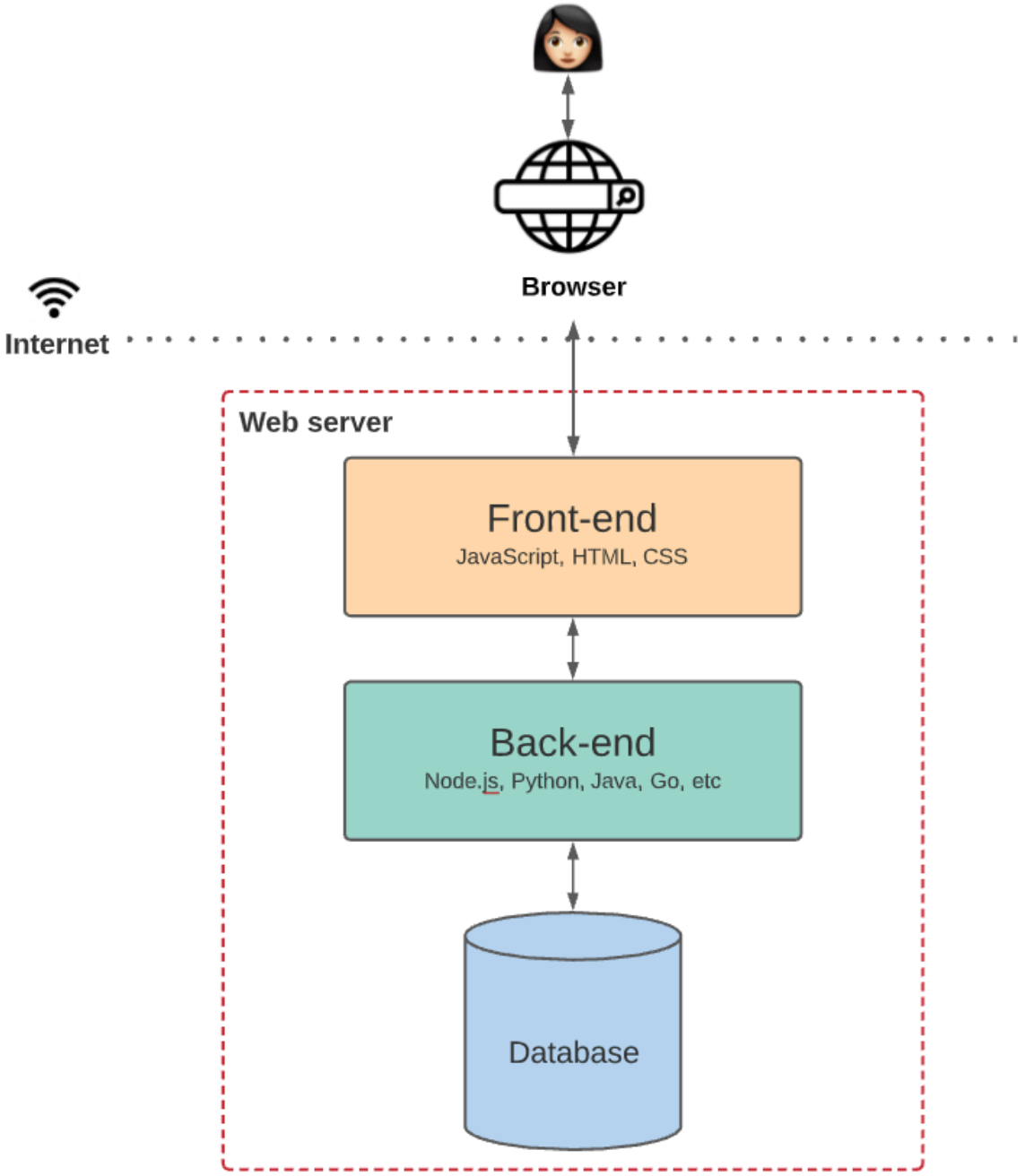


Figure 3.1 Web2.0 Architecture

But all of that's changing. Blockchain technology has unlocked an exciting new direction for Web 3.0 applications. What makes Web 3.0 different?

Unlike Web 2.0 applications like Medium, Web 3.0 eliminates the middle man. There's

no centralized database that stores the application state, and there's no centralized web server where the backend logic resides.

Instead, you can leverage blockchain to build apps on a decentralized state machine that's maintained by anonymous nodes on the internet.

By "state machine," I mean a machine that maintains some given program state and future states allowed on that machine. Blockchains are state machines that are instantiated with some genesis state and have very strict rules (i.e., consensus) that define how that state can transition.

Better yet, no single entity controls this decentralized state machine — it is collectively maintained by everyone in the network.

And what about a backend server? Instead of how Medium's backend was controlled, in Web 3.0 you can write smart contracts that define the logic of your applications and deploy them onto the decentralized state machine. This means that every person who wants to build a blockchain application deploys their code on this shared state machine.

And the front end? It pretty much stays the same, with some exceptions, which we will cover later.

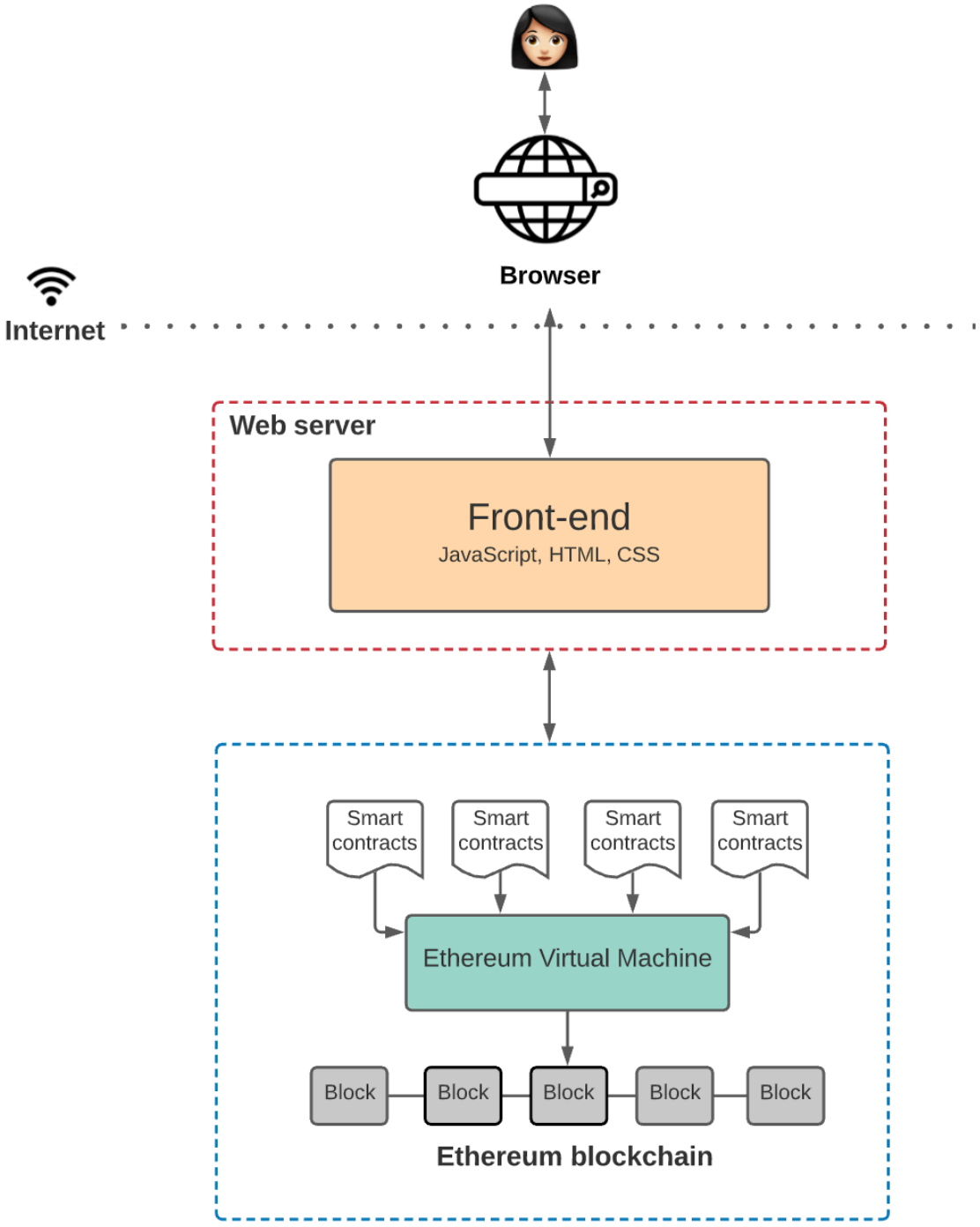


Figure 3.2 Web3.0 Architecture

A Closer Look:

1. **Blockchain**

The Ethereum blockchain is often touted as a "world computer." That's because it's a globally accessible, deterministic state machine maintained by a peer-to-peer network of nodes. State changes on this state machine are governed by the rules of consensus that the peers in the network follow.

So, in other words, it's literally designed to be a state machine that anyone in the world can access and write to. As a result, this machine isn't owned by any single entity — but collectively by everyone in the network.

One more thing to know: data can only be written to the Ethereum blockchain — you can never update existing data.

2. **Smart Contract**

A smart contract is a program that runs on the Ethereum blockchain and defines the logic behind the state changes happening on the blockchain. Smart contracts are written in high-level languages, such as Solidity or Vyper.

Because smart contract code is stored on the Ethereum blockchain, anyone can inspect the application logic of all smart contracts on the network.

3. **Ethereum Virtual Machine (EVM)**

Up next, you have the Ethereum Virtual Machine, which executes the logic defined in the smart contracts and processes the state changes that happen on this globally accessible state machine.

The EVM doesn't understand high-level languages like Solidity and Vyper, which are used to write smart contracts. Instead, you have to compile the high-level language down into bytecode, which the EVM can then execute.

4. **Front-end**

Finally, we have the frontend. As we mentioned before, it defines the UI logic, but the frontend also communicates with the application logic defined in smart contracts.

The communication between the frontend and smart contracts is a little more complicated than it appears in the diagram above.

How Does the Frontend Code Communicate with Smart Contracts on Ethereum? We want our frontend to communicate with our smart contracts so that they can invoke functions, but recall that Ethereum is a decentralized network. Every node in the Ethereum network keeps a copy of all states on the Ethereum state machine, including the code and data associated with every smart contract.

When we want to interact with the data and code on a blockchain, we need to interact with one of these nodes. This is because any node can broadcast a request for a transaction to be executed on the EVM. A miner will then execute the transaction and propagate the resulting state change to the rest of the network.

There are two ways to broadcast a new transaction:

- **Set up your own node which runs the Ethereum blockchain software**
- **Use nodes provided by third-party services like Infura, Alchemy, and Quicknode**

If you use a third-party service, you don't have to deal with all the headaches of running a full node yourself. After all, setting up a new Ethereum node on your own server can take days. (There's a lot of data to sync — It can even take up more bandwidth and storage than a typical laptop can handle).

Moreover, the cost of storing the full Ethereum blockchain goes up as your DApp scales, and you need to add more nodes to expand your infrastructure. That's why, as your infrastructure becomes more complex, you'll need full-time DevOps engineers. They'll help you maintain the infrastructure to ensure reliable uptime and fast response times.

All that to say, avoiding these headaches is why many DApps choose to use services like Infura or Alchemy to manage their node infrastructure for them. Of course, there's a trade-off since this creates a centralized chokepoint.

The nodes that you connect with when you need to interact with the blockchain (whether you set them up yourself or use existing ones from third-party services) are often called "providers".

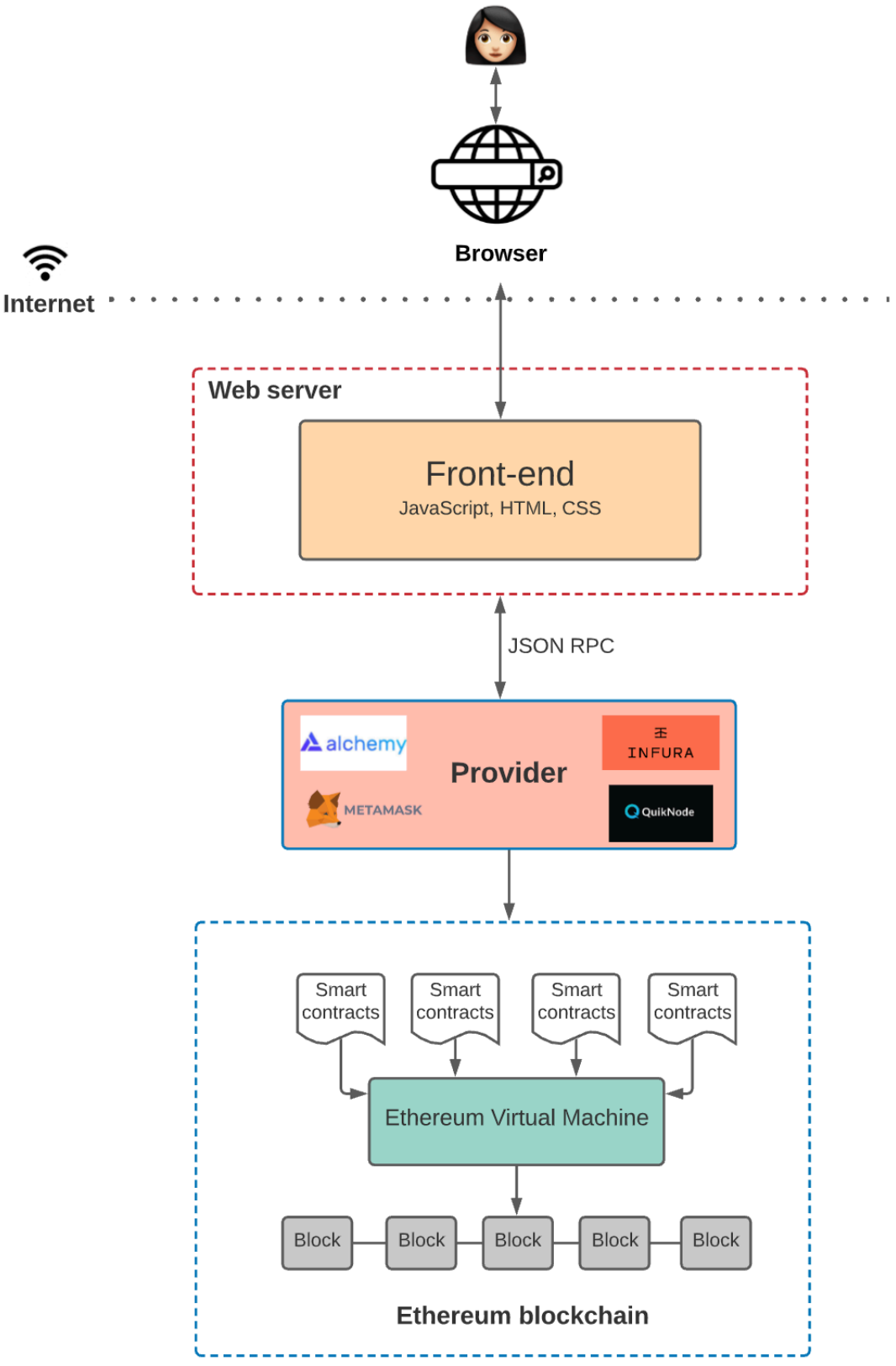


Figure 3.3 JSON RPC Web3.0 Architecture

Every Ethereum client (i.e., provider) implements a JSON-RPC specification. This ensures that there's a uniform set of methods when frontend applications want to interact with the blockchain. If you need a primer on JSON-RPC, it's a stateless, lightweight remote procedure call (RPC) protocol that defines several data structures and the rules for their processing. It's transport-agnostic, so the concepts can be used within the same process, over sockets, over HTTP, or in many various message-passing environments. It uses JSON (RFC 4627) as a data format.

Once you connect to the blockchain through a provider, you can read the state stored on the blockchain. But if you want to write to the state, there's still one more thing you need to do before you can submit the transaction to the blockchain— "sign" the transaction using your private key.

For instance, imagine we have a DApp that lets users read or publish blog posts to the blockchain. You might have a button on the frontend that allows anyone to query for the blog posts written by a particular user. (Recall that reading from the blockchain does not require a user to sign a transaction).

However, when a user wants to publish a new post onto the chain, our DApp would ask the user to "sign" the transaction using their private key — only then would the DApp relay the transaction to the blockchain. Otherwise, the nodes wouldn't accept the transaction.

This "signing" of transactions is where Metamask typically comes in

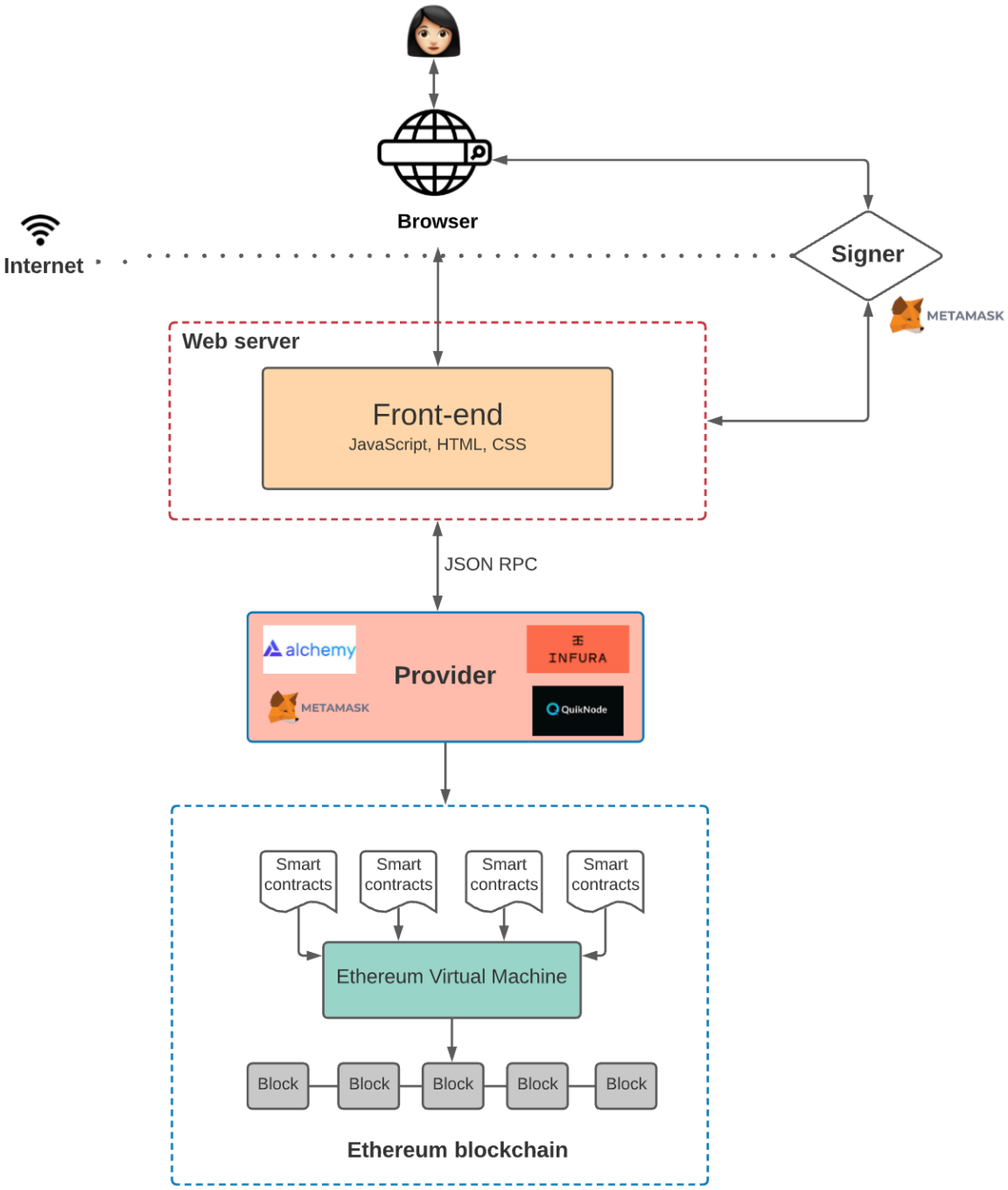


Figure 3.4 Metamask Web3.0 Architecture

Metamask is a tool that makes it easy for applications to handle key management and transaction signing. It's pretty simple: Metamask stores a user's private keys in the browser, and whenever the frontend needs the user to sign a transaction, it calls on Metamask.

Metamask also provides a connection to the blockchain (as a "provider") since it already has a connection to the nodes provided by Infura since it needs it to sign transactions. In this way, Metamask is both a provider and a signer.

- **What is Solidity?**

Proposed in 2014, Solidity is an object-oriented, high-level programming language for implementing smart contracts that run on the Ethereum Virtual Machine (EVM). Solidity is inspired by well-known programming languages like C++ and JavaScript. As a web3 developer, working with Solidity is beneficial to learning web3 development.

- **How Solidity used?**

Specific to its nature as a blockchain coding language, Solidity was created to expressly be compiled on the Ethereum Virtual Machine (EVM). As such, Solidity is able to write secure Ethereum-compatible smart contracts, for any EVM-compatible blockchain.

- **How is Solidity work?**

Solidity developers write smart contracts, which are programs running on the EVM. Similar to classes in other languages, smart contracts can have multiple functions, internal states and public declarations.

Solidity contracts are compiled into bytecode to be read by the EVM. After compilation, the Solidity contracts will look no different from those written in other EVM smart contract languages like Vyper or Huff.

Like any other developer tool, Solidity goes through regular version updates. Note that because smart contracts are deployed once and are irreversible after joining the blockchain, Solidity version constraints will only matter at the time of creation.

While Solidity developers can theoretically write Ethereum smart contracts directly in bytecode, high-level languages like Solidity allow engineers to write contracts more efficiently without any low-level manipulation.

❖ **How does Solidity interact with the EVM?**

The following is a summary of how Solidity works to change the state of the EVM:

- Write smart contracts in Solidity.
- Solidity smart contracts are compiled to bytecode.
- Bytecode is converted into EVM opcodes.
- EVM opcodes alter the state of the EVM.

The EVM is a virtual state machine and record keeper for Ethereum's "global computer." In the simplest terms, the EVM maintains the global state of the network, which includes addresses, account balances and token holdings.

Solidity smart contracts are written to make changes to the EVM's global state. The EVM's global state is changed by actions such as minting tokens, sending and receiving tokens, or releasing assets.

A host of EVM operations, encoded into EVM opcodes, prompt these changes to the global state. Each EVM opcode is designed to prompt the EVM to perform a specific kind of state change, such as sending tokens from one address to another, minting tokens, or even destroying smart contracts.

In the last step, the EVM translates the smart contract's bytecode into EVM opcodes, which make the necessary changes to the blockchain.

Each opcode comes with a certain gas fee, charged in ETH to mitigate the risk of a malicious actor clogging up the network with an inordinate amount of requests. The total gas cost of a transaction is thus calculated by the price per unit of gas times the total gas cost of a transaction's opcodes.

❖ **What is Thirdweb?**

Thirdweb is a web3 app framework that makes it easy to develop web3 apps by providing a dashboard, SDKs, and contracts to help you create web3 applications. The dashboard makes it easy to deploy and manage permissions on smart contracts.

It provides SDKs in popular languages such as JavaScript, Python, Go, Unity, and C#

that you can use to interact with your contracts. Thirdweb also offers a variety of pre-built contracts (tokens, marketplaces, treasury, and governance) to integrate into your app or import your smart contracts using thirdweb deploy.

Benefits of using thirdweb:

- **Thirdweb is fully on-chain** – Thirdweb does not store any data and does not have access to your contracts which means that you don't need to trust thirdweb's infrastructure or servers.
- **Thirdweb provides SDKs in multiple popular languages and frameworks** – You don't need to learn any other web3 tool or framework.
- **You own the smart contract you deploy** – There is no dependency on thirdweb, and you own all admin rights to the code deployed on-chain. If thirdweb disappears tomorrow, you can still access your smart contract.
- **Thirdweb's smart contracts are audited** – With the immutability nature of blockchain, it's imperative to mitigate the risk of smart-contract exploitation. With blockchain, you are often handling people's investments in large sums. You want to ensure you and any user interacting with your contract are protected.
- **Thirdweb is free to use** – Thirdweb does not charge for any of the features it provides. The only fees you will be paying are transaction fees to deploy on the blockchain

The full suite of tools from the dashboard, SDKs, and contracts allows you to build web3 apps quickly and without much friction. In the following guides, you will learn how to use SDKs and dashboard to deploy and manage our contracts and build the frontend for your app using Next.js.

3.2. Technologies and Tools

The technology and tools needed to build a typical crowdfunding platform based on smart contracts would include:

1. **Ethereum Blockchain:** Ethereum is the most popular platform for developing smart contracts and creating decentralized applications (dApps).
2. **Ethereum Virtual Machine:** The Ethereum Virtual Machine (EVM) is used to execute the smart contracts.
3. **Solidity:** Solidity is the programming language used to write the smart contracts on Ethereum.
4. **Truffle:** Truffle is a development environment that helps developers compile, debug and deploy their contracts.
5. **Web3.js:** Web3.js is a JavaScript library that can be used to interact with decentralized applications (dApps) and smart contracts on Ethereum.
6. **Metamask:** Metamask is an Ethereum wallet that allows users to store ETH and other tokens. It also allows users to interact with dApps and sign transactions on the Ethereum blockchain.
7. **IPFS:** InterPlanetary File System (IPFS) is a distributed file storage protocol that can be used to store documents, images, and other types of files.
8. **Authentication and Authorization:** Authentication and authorization methods such as OAuth, JSON Web Tokens (JWTs), and OpenID Connect can be used to create secure access to the platform and its components.

3.3. Front-end Project

3.3.1. Running Project

The full documented guide is supported in *crowdfund-frontend* repository with ***README.md*** file.

3.3.2. Project Structure

The next project structure:

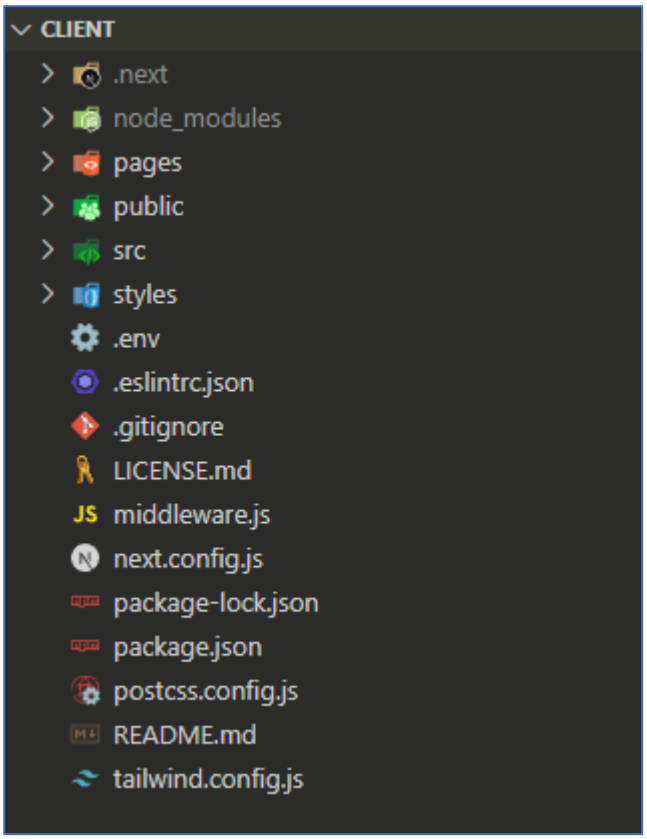


Figure 3.5 Front-end Project Structure

Pages – provides the routing mechanism

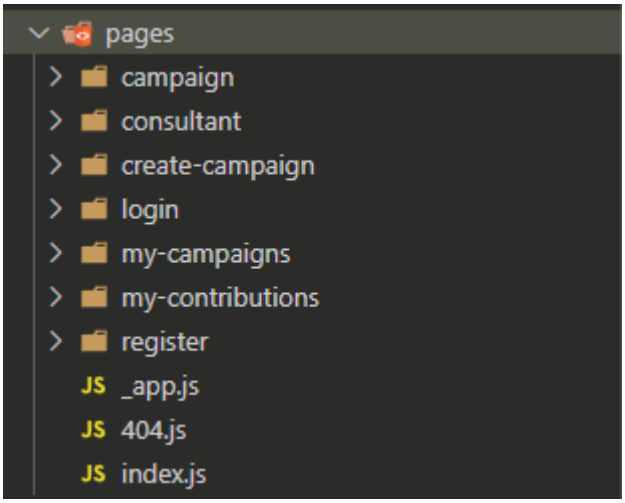


Figure 3.6 Front-end Pages Folder

The whole magic relies in the State Context of our project:

```
import React, { useContext, createContext, useState } from "react";
import {
  useAddress,
  useContract,
  useMetamask,
  useContractWrite,
} from "@thirdweb-dev/react";
const StateContext = createContext();

export const StateContextProvider = ({ children }) => {
  const [token, setToken] = useState(null);
  const [role, setRole] = useState(false);

  const address = useAddress();
  const connect = useMetamask();
  //related to Factory Smart Contract

  //getting the address(contract) of the FactoryCampaign contract
  const { contract } = useContract(
    "0x944ac44717a2c9722C3A4A525A735884B0C04753"
  );

  //getting the createCampaign function
  const { mutateAsync: createCampaign } = useContractWrite(
    contract,
    "createCampaign"
  );

  //creating the functionality of the creation of a campaign
  const publishCampaign = async (form) => {
    try {
      const data = await createCampaign([
        form.name,
        form.title,
        form.description,
        form.image,
        form.target,
        form.deadline.toString(), //unix based timestamp
        form.min,
      ]);
      return data;
    } catch (err) {
      console.error(err);
    }
  };

  return (
    <StateContext.Provider
      value={{
        address,
        publishCampaign,
        contract,
        connect,
        token,
        setToken,
        role,
        setRole,
      }}
    >
      {children}
    </StateContext.Provider>
  );
};

export const useStateContext = () => useContext(StateContext);
```

Figure 3.7 Front-end State Context Code

First we are getting the address from the *useAddress()* Hook, it is the address of the user's wallet that is connected to the browser, then we get the connect method that is provided by *useMetamask()* which help us connect to our wallet in the browser, then we get our *CampaignFactory* contract and feed it to the *useContract()* that is provided through thirdweb library, *useContract()* spits one variables, the contract ABI which gives the ability to interact with the contract methods, then we provide this contract ABI to another hook called *useContractWrite()* and the function name in string format, which returns the contract method that we need such as *createCampaign()*.

This Context Provider hold the whole functionalities, data, and states, and provide it to every component that needs it.

3.3.3. Work Samples

A registration page is a webpage that allows users to create an account on a website or application. A typical registration page will consist of a form that requests certain information from the user, such as a username, email, password and confirm password.

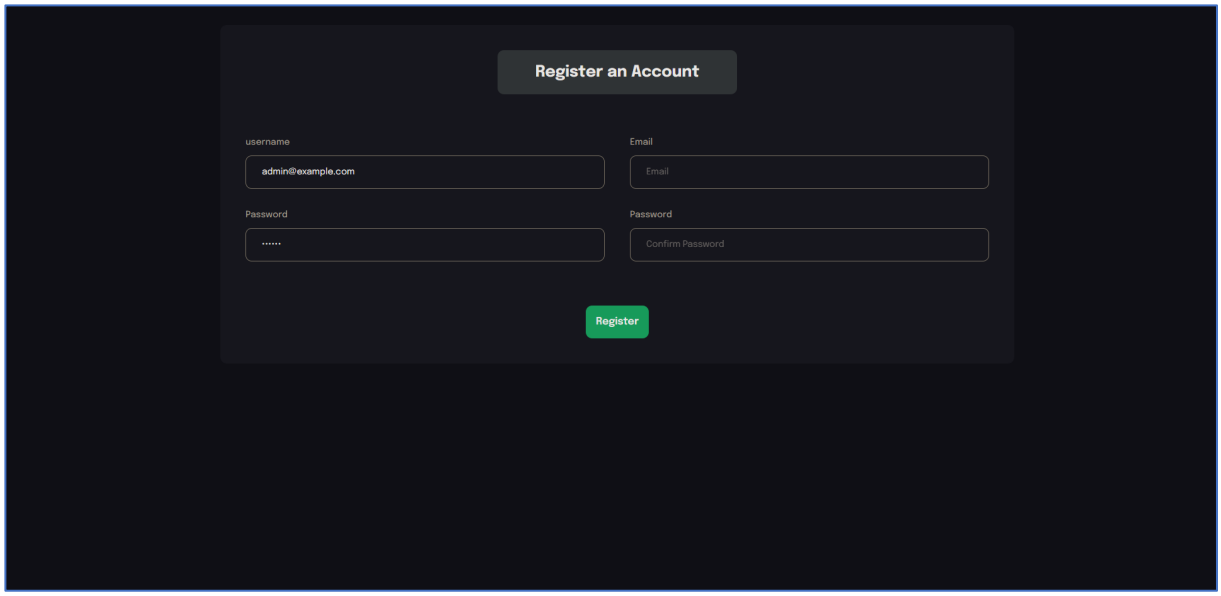


Figure 3.8 Registration Page

A login page is a webpage that allows users to access their account on a website or application. A typical login page will consist of a form that requests certain information from

the user, such as an email address and a password.

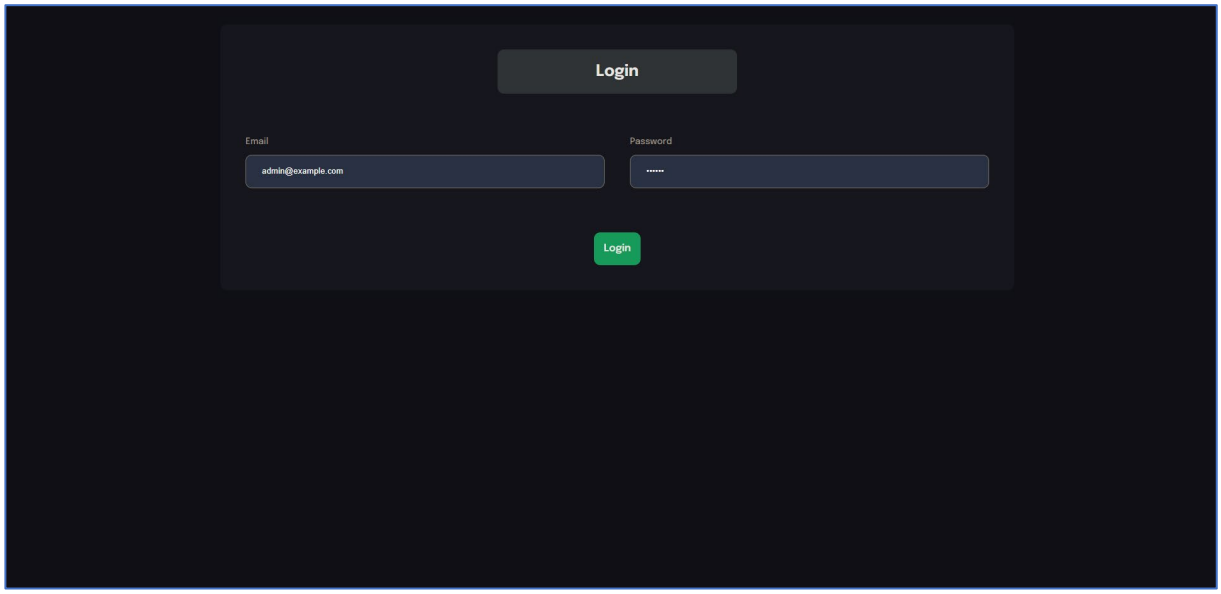


Figure 3.9 Login Page

When a campaign creator wants to raise funds for a specific project or venture, they can initiate a create request on the platform. This typically involves filling out a form with details about the campaign, such as the funding goal, deadline, rewards offered to investors, and other information relevant to the campaign.

Once the create request is submitted, it may go through a review and approval process by the platform's administrators. This process can ensure that the campaign meets all of the necessary conditions and regulations, such as the campaign's Whitepaper, legal documents and smart contract auditing.

Once the create request is approved, the campaign will be live on the platform and available for investors to discover and invest in.

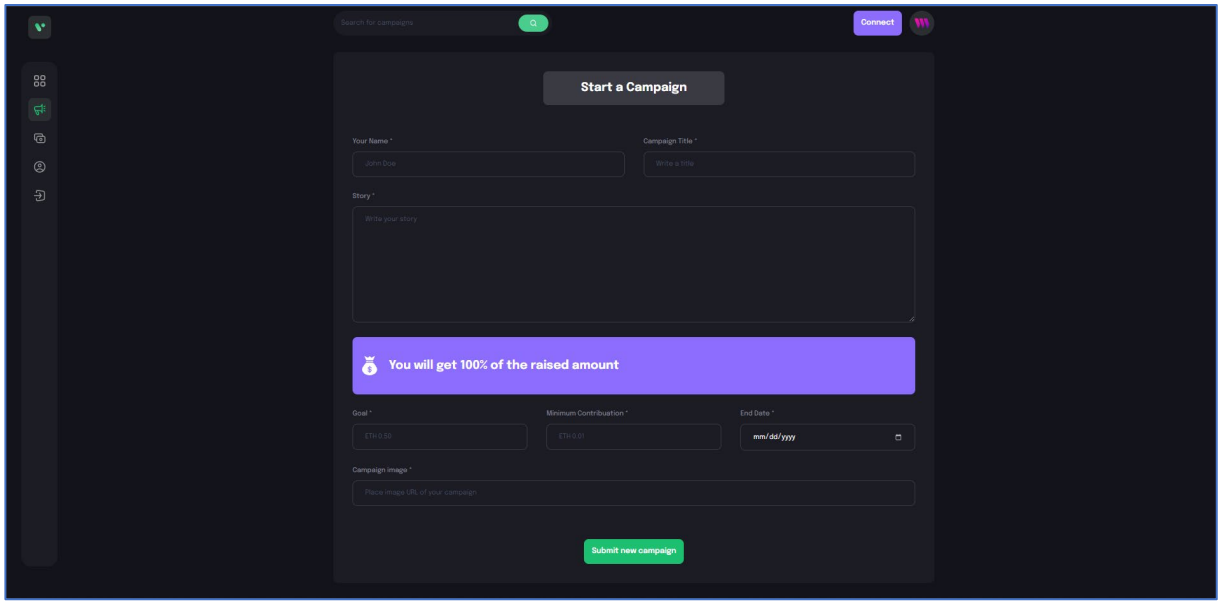


Figure 3.10 Campaign Creation Page

It would typically have a list of all the campaigns that the user has created, or pending, rejected, or approved and needed to be deployed, with each campaign represented by a card or a brief summary. The summary would include key details such as the campaign title, funding goal, amount raised, deadline, and the status of the campaign (e.g., active, funded, expired, etc.).

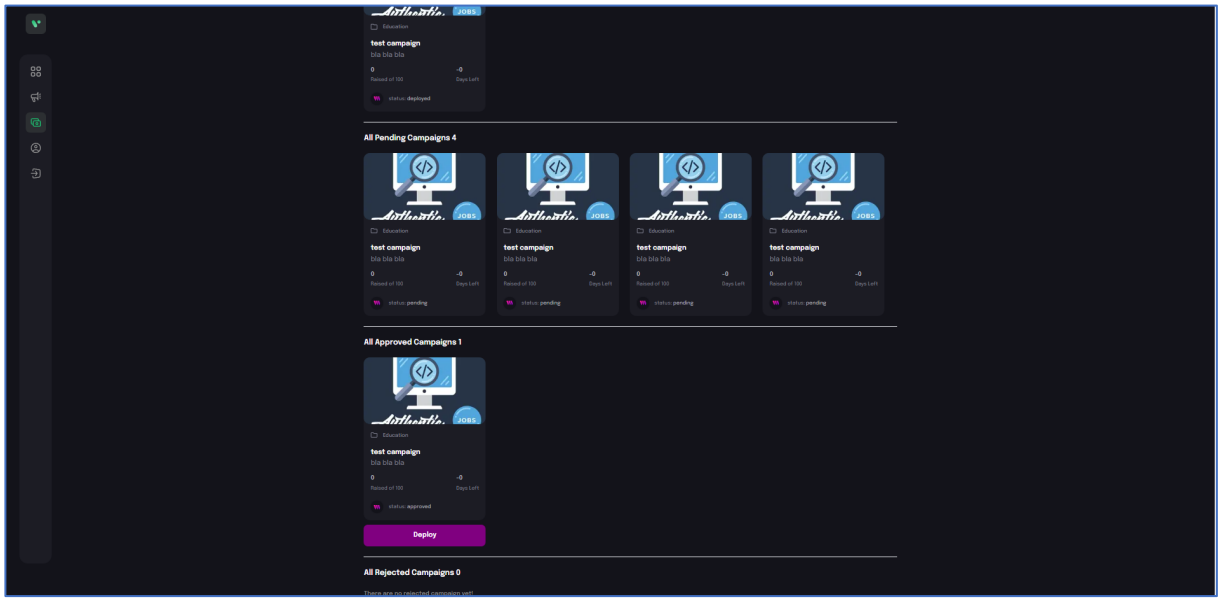


Figure 3.11 My Campaign Page

It would typically have a list of all the campaigns that the user has invested in, with each campaign represented by a card or a brief summary. The summary would include key details

such as the campaign title, funding goal, amount raised, deadline, and the status of the campaign (e.g., active, funded, expired, etc.).

The contributions page would display the amount of money or tokens the user has contributed to each campaign, as well as the rewards or equity the user is entitled to receive.

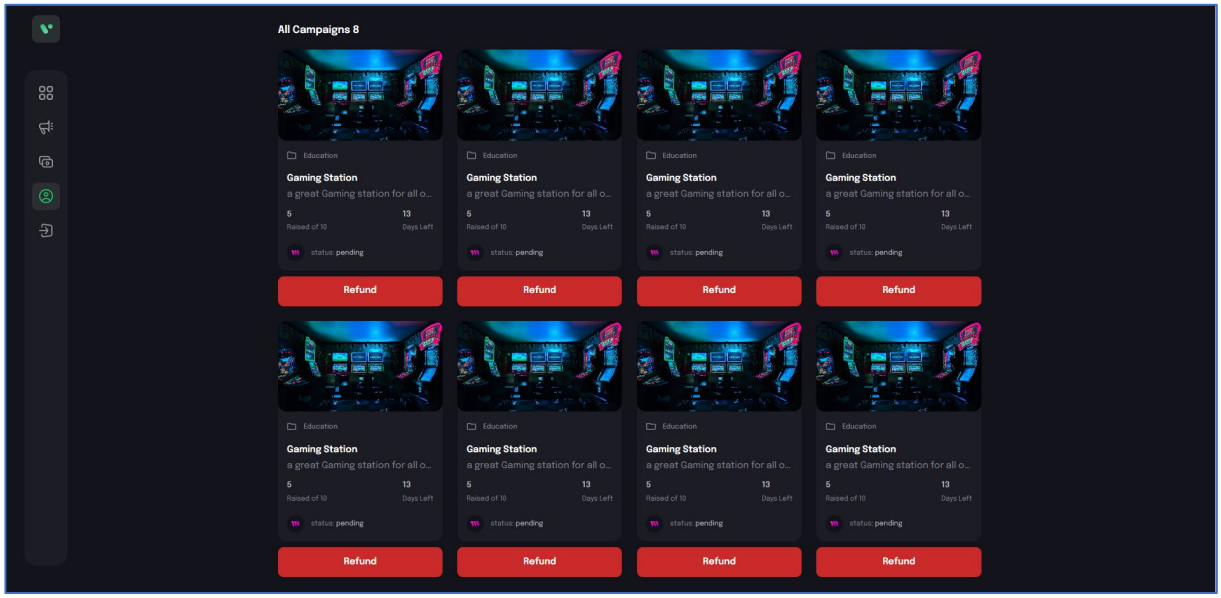


Figure 3.12 My Contributions Page

Consultant can view the pending campaign and either accept the campaign which put the campaign in the approved state, or reject it.

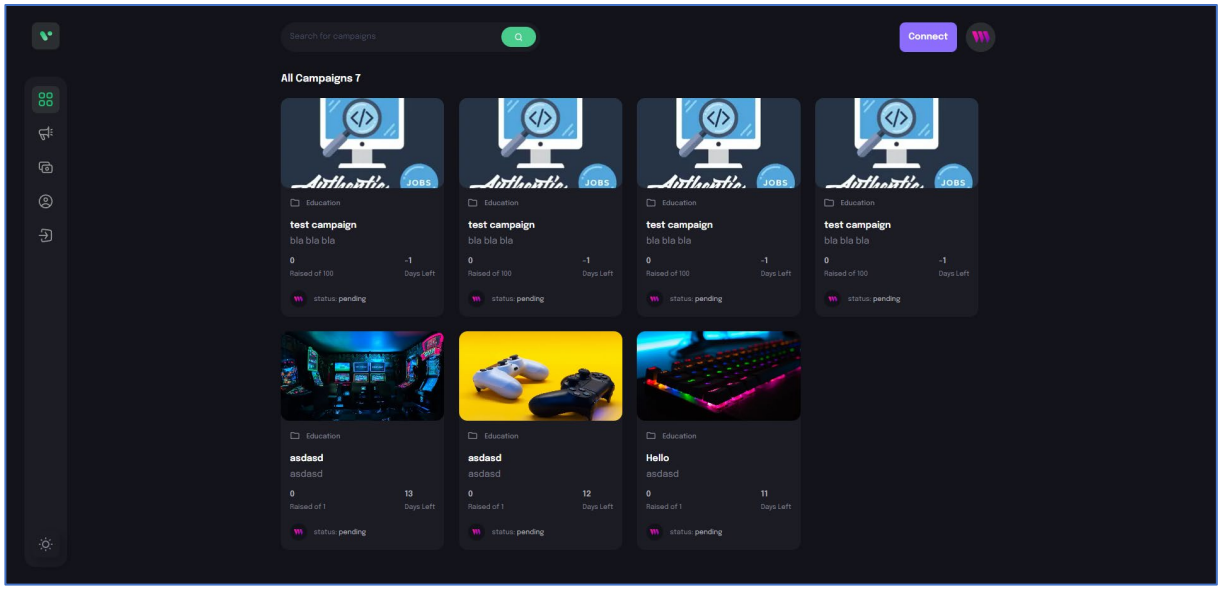


Figure 3.13 Pending Campaigns Page

This allows the campaign owner to create a new request by providing the amount of ethers that the request should need and the recipient address and the reason behind creating this request.

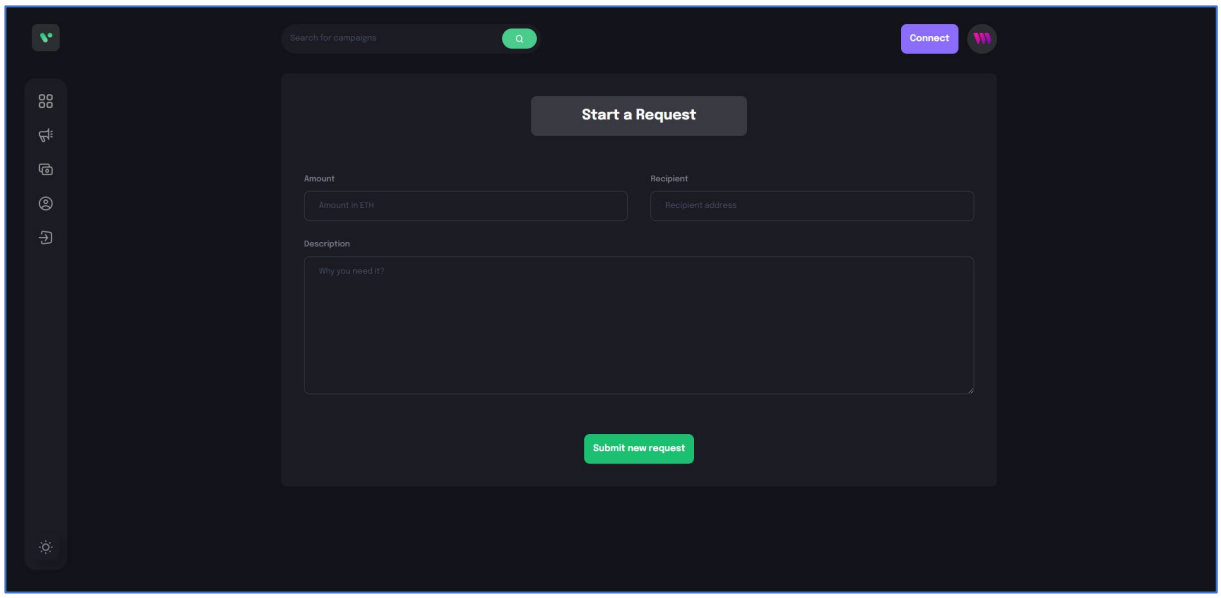


Figure 3.14 Request Creation Page

A page where a contributor can see the list of requests on a specific campaign and can approve on it.

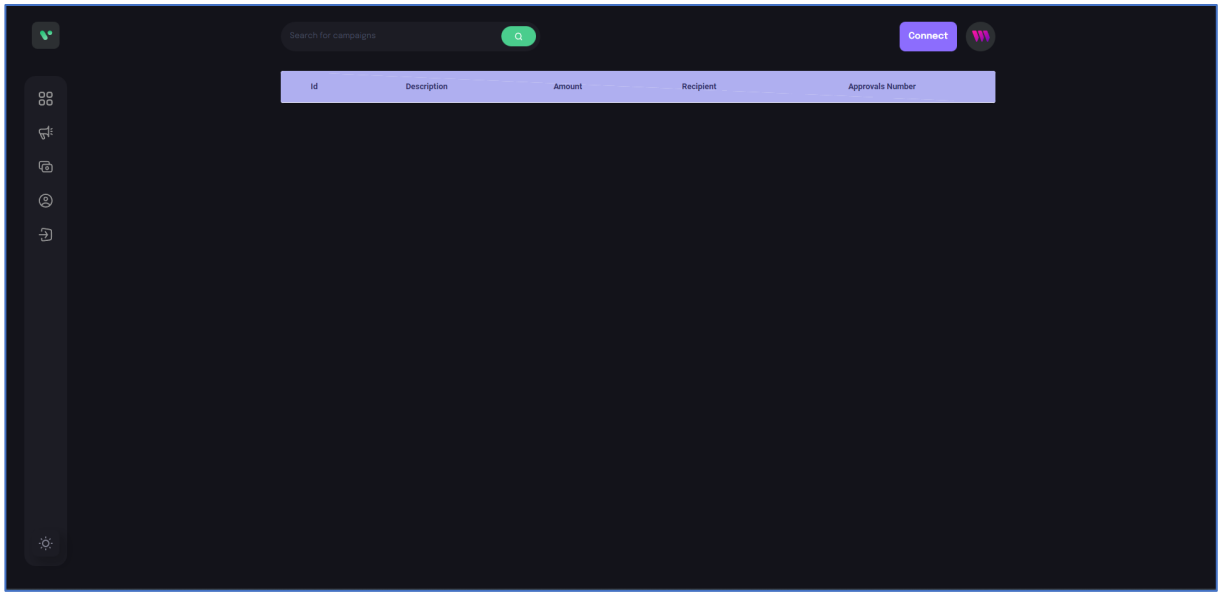


Figure 3.15 Approving Page

3.4. Campaign Project

3.4.1. About The Project

Smart contracts are resilient and highly secured applications deployed on the blockchain. We are going to build a smart campaign contract.

First, take a look at this diagram of what we built:

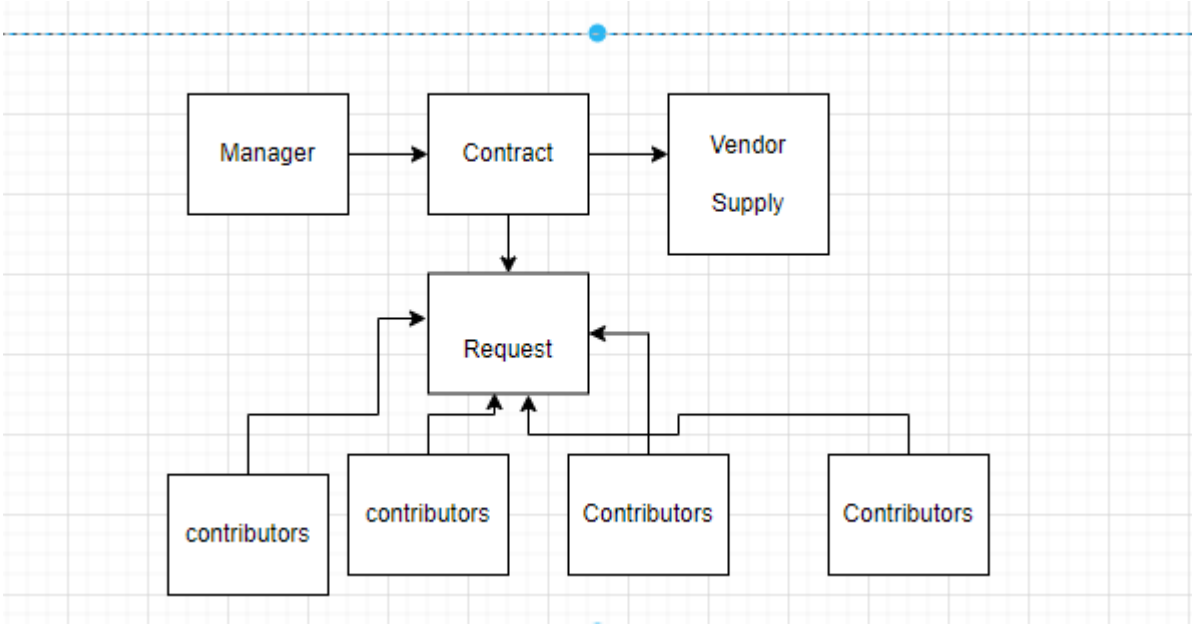


Figure 3.16 Contract Design

- The business owner calls the contract to launch a campaign using several justifications, including the number of money that must be raised for the campaign, its start timestamp, and its end timestamp.
- As long as the campaign has not already begun, the business owner may cancel it at any time.
- By using the "pledge" function and entering the "id" of the campaign along with the number of tokens to be pledged, users can donate their tokens to a particular campaign.
- As long as the campaign is still active, they can also withdraw their previously pledged money.

- After the campaign is over, one of the two possible results is:
    - A campaign is considered successful when it raises the required number of money and meets the business owner's minimum requirements. In this scenario, the business owner can call the "finalize" function to withdraw all the amount.
    - If not, enough amount is pledged, which is the other scenario where a campaign fails, pledgers can withdraw their contributions from the contract by using the "withdraw" function.
- The vendor account is filled in the struct when requesting a spending request and, once approved, sent to that vendor and not the manager.
- The accountability and security of contributors' funds are guaranteed.

3.4.2. Project Structure

The solidity project typically contains a contracts folder, which includes all the source code for the smart contracts, a build folder, which contains all the compiled contracts, and a test folder, which contains various test scripts for the project.

The web3.js project folder should include all the files necessary to deploy web3.js on your server. This will typically include a *package.json* file, JavaScript source files, and configuration files for connecting to an Ethereum node. The Solidity project folder should include all the files necessary to compile and deploy Solidity smart contracts to your Ethereum node. This will typically include Solidity source files, compilation configuration files, and deployment scripts.

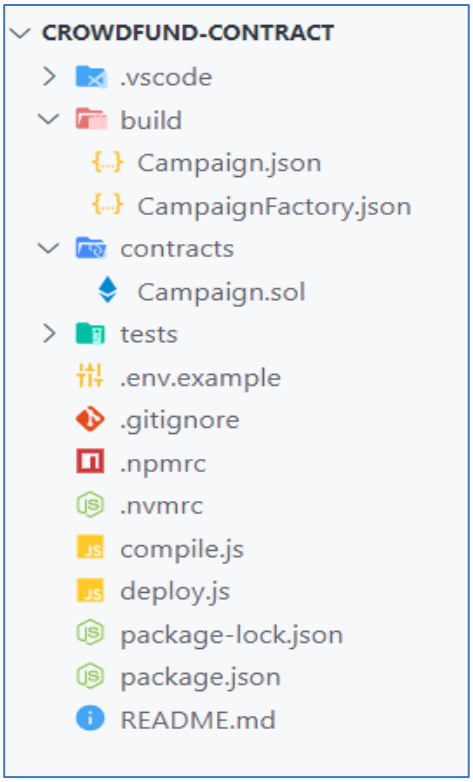


Figure 3.17 Contract Project Structure

3.4.3. Code Sample

The constructor function

The *Constructor* or *function* campaign is the first function to run as we want to make sure only the manager can call the contract. Note that we made it public to see the caller of the contract. *msg.sender* is a universal variable that we assign to the manager, *manager = msg.sender;* whenever the contract is called so, whoever calls the contract is the manager. We *called (uint minimum)public* which is an unsigned integer for the amount of money required to become a contributor, we can have a different minimum or create a static amount. Then we can call in the function *MinimumContribution = minimum;* so whoever contributes to the contract is added here *uint public MinimumContribution;*

The approval process

The approvals' function is called when a contributor pays *payable* a minimum amount to the contract. The require function kicks out anyone that has not added value to the

minimumContribution and pushes a contributor to the approvals list. Next, we introduce the payable function and the approval variable, and we will see why they play a significant role in the smart contract.

Requesting funds

Contributors can send funds into our contract, but we cannot pull money out of the request function, leading us to create a struct. Struct is a collection of key-value pairs that have different types. We create a struct as a request that the contributors approve before sending it to the vendor, which is also a struct type. Struct is not an instance but a definition of a type we can use multiple times across the project.

Testing the contract

We'll test the contract right in the IDE.

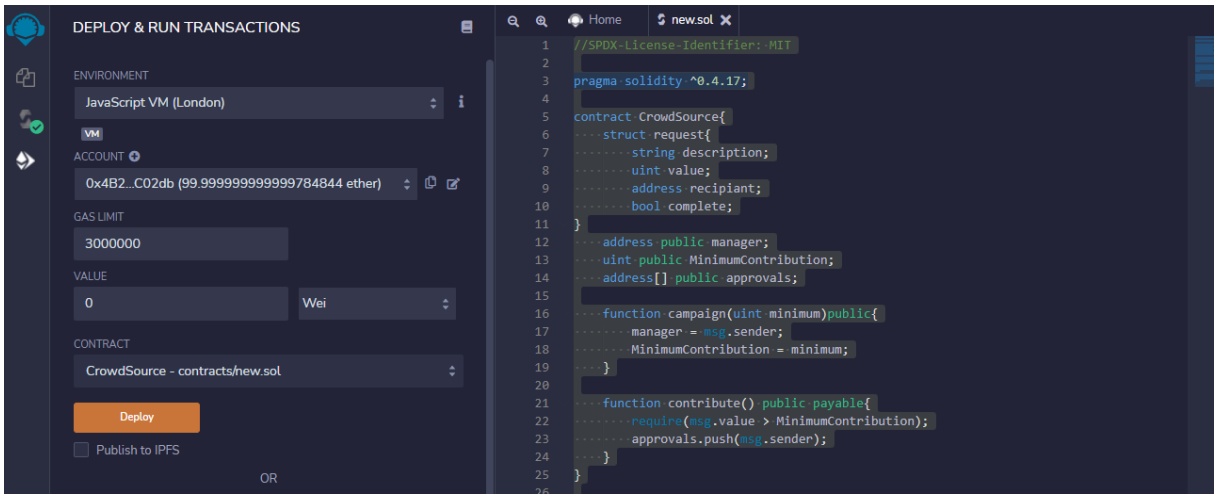


Figure 3.18 Testing (1)

Running a test on Remix is straightforward. We can see the DEPLOY AND RUN TRANSACTIONS on the left-hand side.

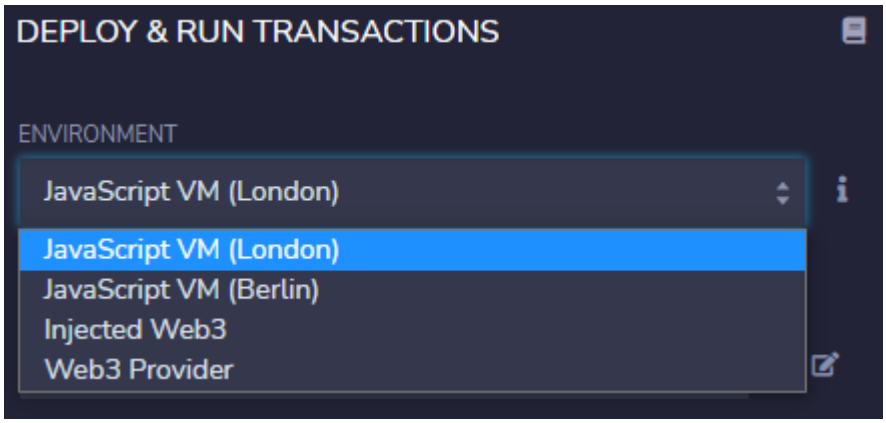


Figure 3.19 Testing (2)

Environment – There are about four different environments, but most times will use just two JavaScript VM (London) and Injected Web3.

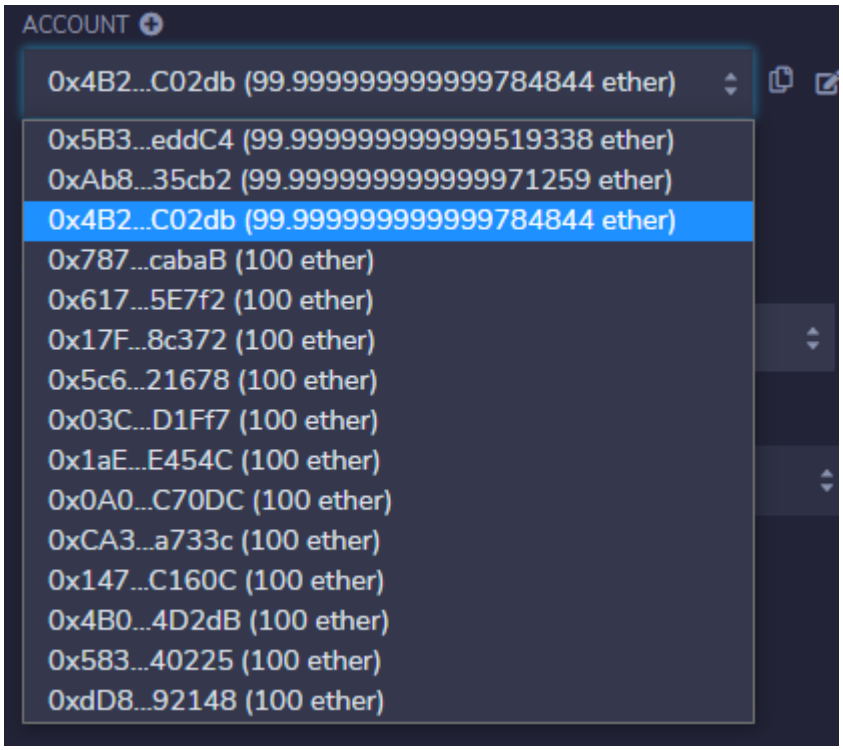


Figure 3.20 Testing (3)

Account – Remix gives 15 free accounts that we can use to deploy contracts and test our deployment.



Figure 3.21 Testing (4)

Gas Limit Gas limit replicates the actual gas fees when deploying a contract. Each deployment or change in the contract will cost us a gas fee, and we will review the gas fee later on.

Value The value is the fund we put into the contract when deploying it. Wei is the smallest unit, while ether is the highest unit of the currency used on Ethereum.

Contract The name of the contract, which in this case is new.sol. .sol is a file format for solidity contracts.

Deploy Deploys the contract into the blockchain network so we can interact with the network.

Compile the contract and click on deploy, a contract will be created below.

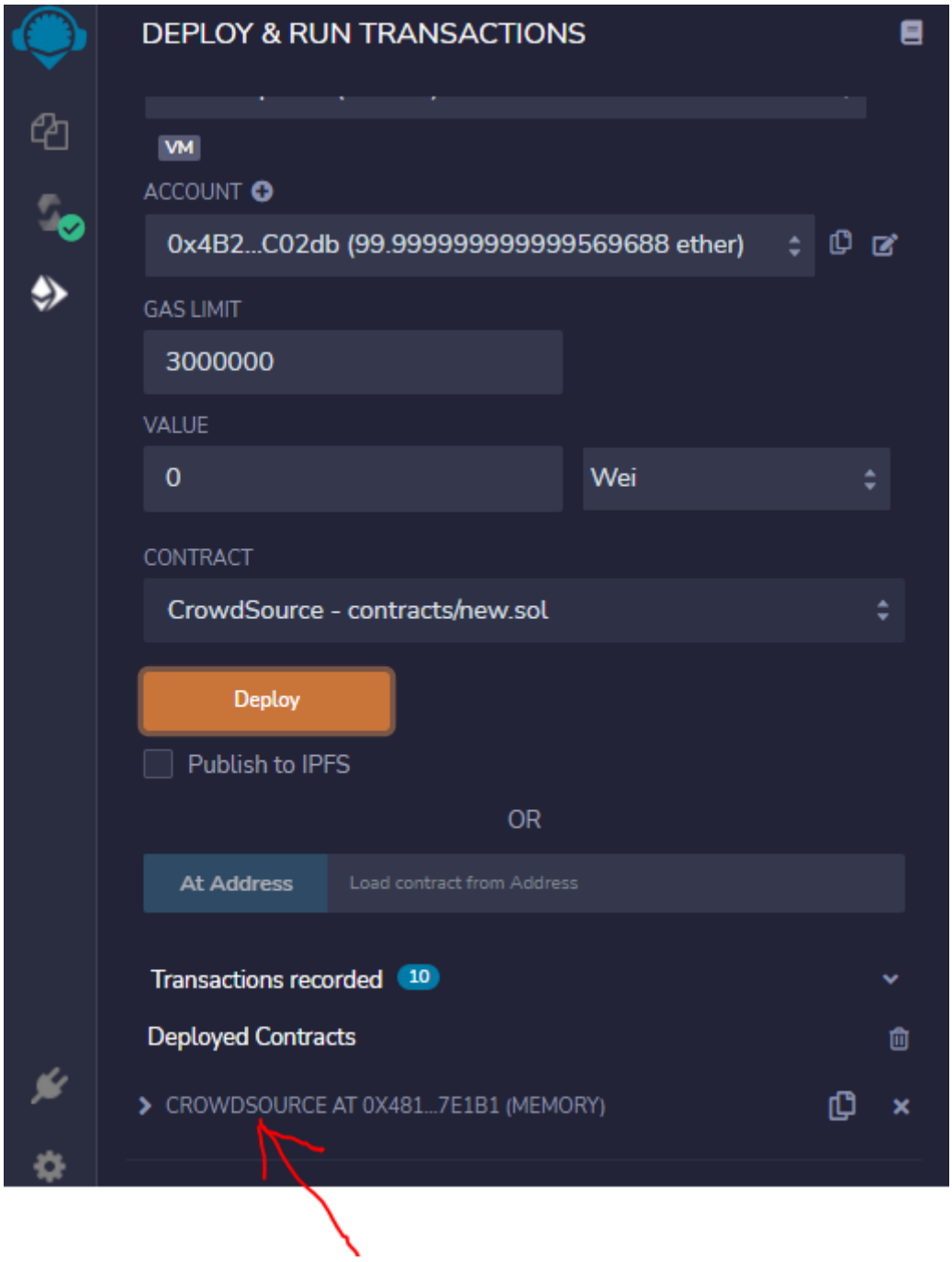


Figure 3.22 Testing (5)

Click on the drop-down button by the left, and see the full deployment.

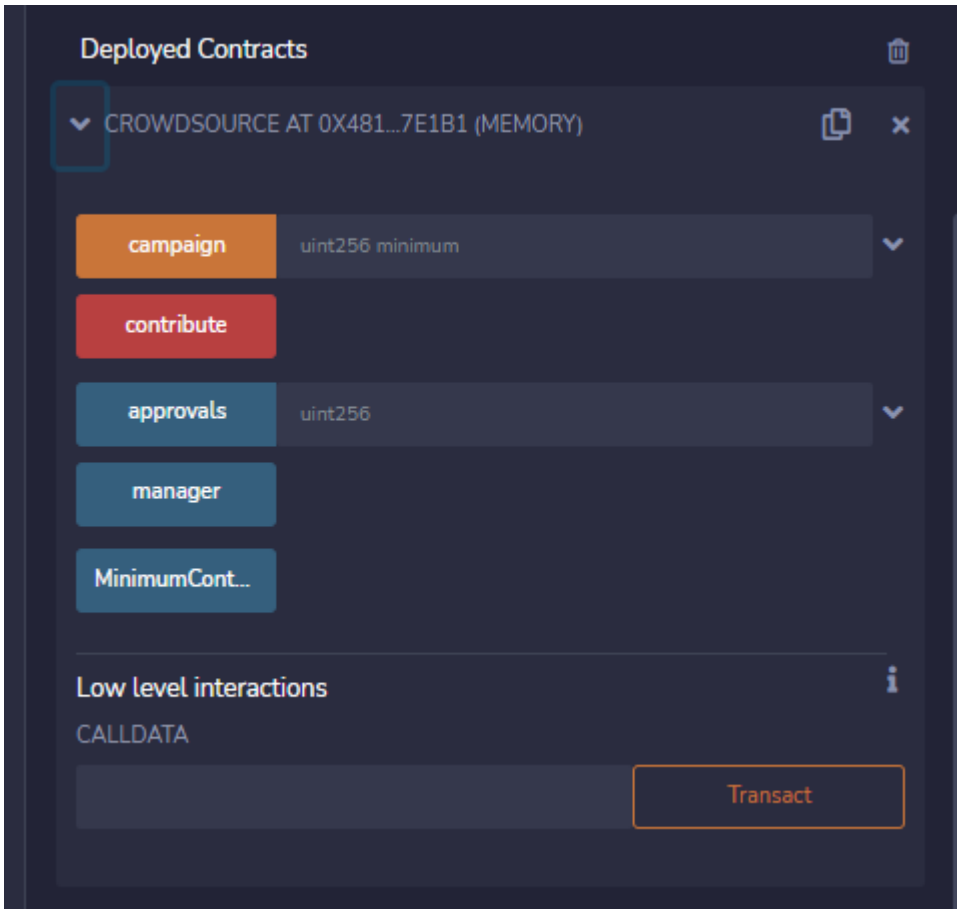


Figure 3.23 Testing (6)

In our contract, we have the campaign which reads uint256, which is an unsigned integer; this is the part we input our contribution into the contract, so whoever puts any funds is added to the contract.

Let's test it. We will put a 100wei into the campaign input space but remember to change the account above, so we do not make the contract manager a contributor.

Use another account and input 100wei.

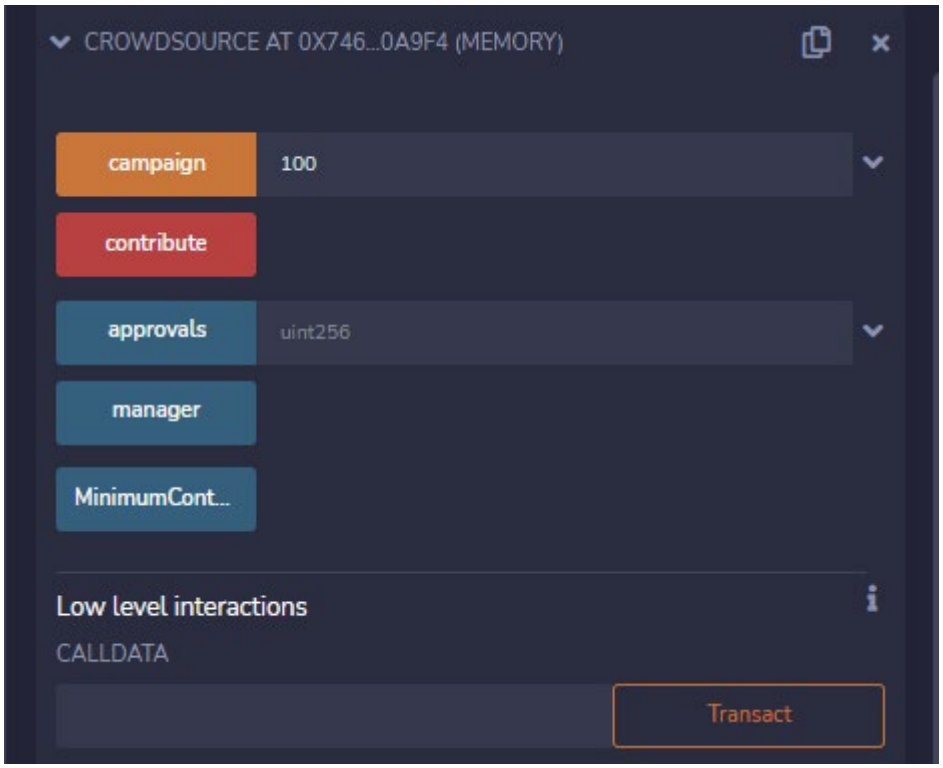


Figure 3.24 Testing (7)

Click on the campaign, and notice a panel on the right that turns green, it means funds has been sent to the contract.

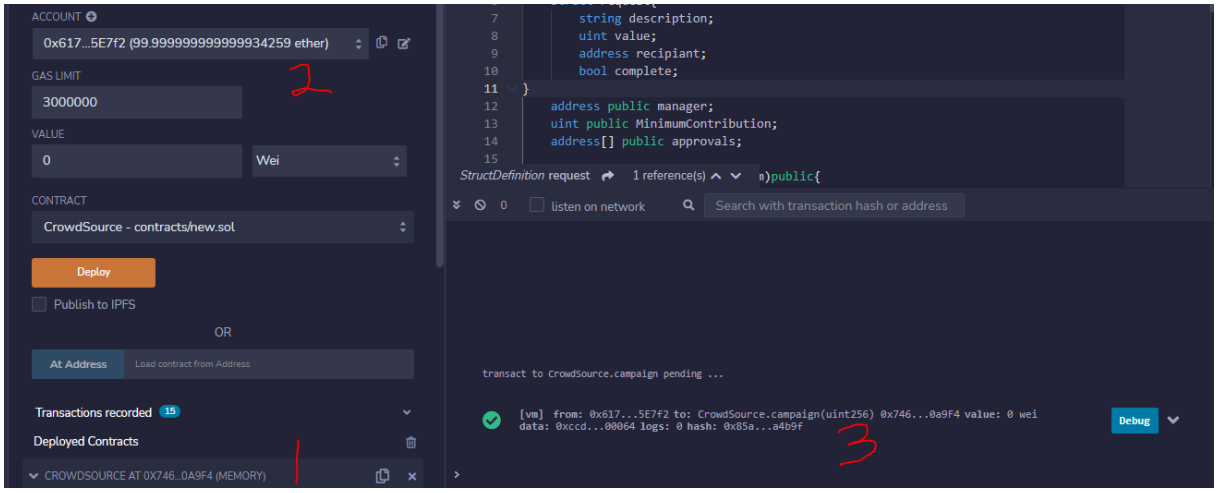


Figure 3.25 Testing (8)

Number 1 is the deployed contract by the manager; notice the address begins with 0X746.. while number 2 is the new contributor to the contract.

Number 3 tells us that a contributor joined the contract

Chapter 4

Conclusion, and Future Improvements

4.1. Results

- ✓ Offering a secure and transparent way to raise funds for a project
- ✓ Help to process donations and manage investments in an immutable, decentralized ledger
- ✓ Offering enhanced security, as there is no centralized authority that intermediates the transactions
- ✓ Allowing for a much more secure crowdfunding process without the risk of censorship or fraud
- ✓ Raise funds with the assurance that all donations are secure and guaranteed to reach their intended destination

4.2. Conclusion

Blockchain in crowdfunding is a relatively new concept to the community. We have taken that into consideration and designed this app so that even a common man can use it with ease. But this is not the end. With the evolution of Blockchain and introducing of ICOs, our application has a bright future and a large scope for improvement and evolution. The world is still adjusting to Blockchain and Cryptocurrencies and it'll take a couple of years more for Ethereum based Dapps to become popular and to be recognized by the community. In such a situation Blockchain based crowdfunding application is a tough concept to be understood by everyone. We have taken that into consideration and designed this app so that even a common man can use it with ease. But this is not the end. With the evolution of Blockchain and introducing of ICOs, our application has a bright future and a large scope for improvement and evolution. In the future, we wish to provide an even easier and safer way for all ideas to get life through our crowdfunding application.

4.3. Future Work and Improvements

1) **Use IPFS as Decentralized Database** – OrbitDB is a serverless, distributed, peer-to-peer database. OrbitDB uses IPFS as its data storage and IPFS Pubsub to automatically sync databases with peers. It's an eventually consistent database that uses CRDTs for conflict-free database merges, making OrbitDB an excellent choice for decentralized apps (dApps), blockchain applications, and offline-first web applications. There are Go and Javascript implementations available.

   We are currently using MongoDB, you might be more comfortable with ThreadDB or AvionDB, which are unstructured databases built on top of IPFS.

2) **Implement rewards-based crowdfunding** – expectation of receiving a non-financial reward in return, such as goods or services at a later stage for donors and investors.

3) **Voting Algorithm** – for use in fundraising that increases fairness for all investors in approving requests funding for a project. A voting system is a method for people to choose one from two or more possibilities. For electronic voting, it is vital to secure the vote against fraud, so selecting a robust security protocol is critical. For online projects, there are issues of requiring performance for continued payment, and fairness to all investors regardless of the money they invest. To avoid fiascos where projects request vast sums of money and then do not deliver, we can define a smart contract supporting projects with requests and a sequence of opportunities for the investors to vote on whether to approve. For a project with multiple requests, a new algorithm Anwar gives each investor an equal say and equal risk until their money runs out. This protocol works in a smart contract which requires no central authority, parties, or private channel.

4) **Feasibility Consultation** – add more powerful features for consultants to be able to create a full detailed study about pending campaigns.

5) **Admin Dashboard** – a necessary part of any app that, while mainly working behind the scenes, are crucial features to provide users with stellar service and smooth-running apps.

References

[1] Shuai Wang, Liwei Ouyang, Yong Yuan, “Blockchain-Enabled Smart Contracts: and Future Trends”, IEEE Trans. on Sys. Man, and Cybernetics, Feb 2019, 2168-2232.

[2] Yong Yuan , Fei-Yue Wang, “Blockchain and Cryptocurrencies: Model, Techniques, and Applications, IEEE Trans.on Sys. Man and Cybernetics, Sep 2018, 2168-2232.

[3] Bhabendu Kumar Mohanta, Soumyashree S Panda, Debasish Jena, “An Overview of Smart Contract and Use Cases in Blockchain Technology” , 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Oct 2018, 978-1-5386-4430-0.

[4] Yi-Hui Chen, Shih-Hsin Chen, Iuon-Chang Lin, “Blockchain based smart contract for bidding system”, 2018 IEEE International Conference on Applied System Invention (ICASI), Apr 2018, 978-1-5386-4342-6.

[5] X. Wang, X. H. Zheng, X. Z. Zhang, K. Zeng, F.-Y. Wang, "Analysis of cyber interactive behaviors using artificial community and computational experiments", IEEE Trans. Syst. Man Cybern. Syst., Jun. 2017, 995-1006.

[6] F.-Y. Wang, Y. Yuan, X. Wang, R. Qin, "Societies 5.0: A new paradigm for computational social systems research", IEEE Trans. Comput. Social Syst., Mar. 2018, 2168-2232.

[7] VitalikButerin,“A Next-Generation Smart Contract and Decentralized Application Platform” [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[8] S. Nakamoto, “Bitcoin: a peer-to-peer electronic cash system”, [Online]. Available: https://bitcoin.org/bitcoin.pdf.