SUPPLY: DECENTRALIZED CROWDFUNDING SYSTEM

A Graduation Project Thesis Presented to

School of Information Technology and Computer Science

Nile University

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science

By

Mohab Mohamed

Amir Ibraheem

Abdulrahman Ragheb

Karim Sherif

Under Supervision of

Dr. Heba Aslan

July 2023

TABLE OF CONTENTS

CONTENTS

TABLE OF CONTENTS v
LIST OF FIGURES ix
Abstract xi
CHAPTER ONE Introduction 1
CHAPTER TWO BACKGROUND 3
2.1 Blockchain 3
2.1.1 Crowdfunding Decentralized Apps 3
2.1.2 History 6
2.1.3 Public/Private Key Asymmetric Encryption 7
2.1.4 Digital Signature Encryption 7
2.1.5 SHA-256 Hashing 8
2.1.6 Bitcoin peer-to-peer Network Protocol 9
2.1.7 Bitcoin Consensus 10
2.1.8 Blockchain From Critical Viewpoints 12
2.2 Ethereum 13
2.2.1 Smart Contract 14
2.2.2 Gas Fees 14
2.2.3 ERC-20 Token 15
2.2.4 ERC-1155 Token 15
2.3 Web Server: Model View Controller (MVC) 16
2.3.1 History 16
2.3.2 **Controller**: Web Server 17
2.3.3 **Model**: MongoDB 19
2.3.4 **View**: Single-Page Application (SPA) 19

CHAPTER THREE Methodology 21
3.1 Requirement Analysis 21
3.1.1 Functional Requirements: 21
3.1.2 Non-Functional Requirements: 22
3.2 Development & Deployment Tools 22
3.2.1 Functional Programming 23
3.2.2 Object-Oriented Programming 23
3.2.3 Test-Driven Development 24
3.2.4 Hardhat Framework 25
3.2.5 Git & Git Flow 26
3.2.6 GitHub Actions CI: 28
3.2.7 Docker 28
3.3 User Interface Design 29
3.4 Project Architecture 30
3.4.1 Use Case Diagram 30
3.4.2 Class Diagrams 32
3.4.3 Sequence Diagrams 34
CHAPTER FOUR Results & Discussion 37
4.1 Campaign Structure 38
4.2 Reward Token Structure 42
4.3 Web Page Performance 43
4.4 API Endpoints Performance 45
4.4.1 Rewards API Endpoints Evaluation 46
4.4.2 Campaigns API Endpoints Evaluation 47
4.5 Gas Fees Evaluation 48
4.6 Discussion 50
CHAPTER FIVE Conclusion & Future WORK 53

5.1 Conclusion 53
5.2 Future Work 53
References 55

LIST OF FIGURES

Figure 2-1. Public/Private Key Encryption [15] 8

Figure 2-2. SHA-256 Hash Function [16] 8

Figure 2-1. Bitcoin Block Body [11] 9

Figure 2-2. EVM [27] 15

Figure 2-3. The MVC Pattern [30] 17

Figure 2-4. Communication Between User and Web Server in SPA [40] 20

Figure 3-1. Git Flow [48] 27

Figure 3-2. Number of Mobile Users Over Years [53] 30

Figure 3-3. System's Use Case Diagram 32

Figure 3-4. The Class Diagram of Supply's Smart Contracts 33

Figure 3-5. Sequence Diagram for Creating a New Crowdaunding Campaign Scenario 35

Figure 4-1. Main Page 38

Figure 4-2. App's Responsive Design 38

Figure 4-3. Campaign struct of CrowdCampaigns Smart Contract 39

Figure 4-4. Rest of Campaign Data on Mongoose Schema 40

Figure 4-5. Part of Campaign Creation Form 40

Figure 4-6. Campaign Creation Constraints 41

Figure 4-7. Campaign Details 1 42

Figure 4-8. Campaign Details 2 42

Figure 4-9. Token structure 43

Figure 4-10. Fundraiser Rewards 43

Figure 4-11. Supply's Main Page Chrome Lighthouse Metrics 45

Figure 4-12. Rewards API Endpoints Evaluation by Postman Collection Runner 47

Figure 4-13. Campaigns API Endpoints Evaluation by Postman Collection Runner 48

Figure 4-15. Gas Fees Report of Our Contracts' Operations by Hardhat-gas-reporter Plugin 50

ABSTRACT

Blockchain is a cutting-edge technology that made a lot of success in previous years. Since the Blockchain idea is built on the decentralization of data, it depends only on crowdsourcing as it requires users to crowdfund the miners/validators to keep its network alive. That's why most Blockchain apps use the concept of crowdfunding to make grow up business. Crowdfunding on the Ethereum blockchain has provided fundraisers with security, transparency, and lower fees because all operations that are made on the chain don't require third parties as usual Web servers. We aimed to make a Web3 crowdfunding platform named Supply that provides all Ethereum blockchain advantages plus a great user interface and experience. Users of Supply earn valuable SFT (Semi-Fungible Token) rewards by funding their favorite crowdfunding campaigns. This decentralized app is different as it accepts payment in any ERC-20 (Ethereum Request for Comments 20) tokens by swapping them with DAI stablecoin through integration with Uniswap DEX (Decentralized Exchange) Protocol. Supply is a transparent and safe option for people who want to launch a campaign or fund a campaign.

CHAPTER ONE
INTRODUCTION

Fundraising is the process of soliciting funds from individuals or organizations to support a project. When projects are publicized on platforms such as Kickstarter, they often involve a large amount of money and a promise from the creator that they will continue working on the project until it is successful. While success is not guaranteed, refunds are not guaranteed, and there have been notable cases of fraud. There are many ways to raise money and they often don't reflect the interests of investors very well. The Kickstarter crowdfunding platform uses an all-or-nothing funding model. This means that if a project does not receive the funding it needs, no funds are exchanged, and the project cannot take place. The Indiegogo platform works in a way that reflects the amount of money that is received.

Ethereum blockchain financial systems implement these traditional models on smart contracts technology that make the system code immutable and open source which adds full transparency to the funding process. Ethereum blockchain can be used to implement different crowdfunding types such as donation, reward-based, and equity crowdfunding.

Some companies have success stories using these techniques to help generate funds quickly; for instance, Brave Browser has raised $35 million in less than 30 seconds for Basic Attention Token (BAT) [1]. However, there are a lot of failed and scammed projects that have caused investors to lose their money. These projects are divided into the categories of parody, scam, hack, and deceased [2]. The majority of crowdfunding platforms are created in a centralized manner, necessitating user trust. Kickstarter for example requires creators to show a prototype of what they are making before submitting a fundraising proposal, and the goal fund can be raised only after the project has been delivered. Also, white papers present a prototype that teaches investors how to resolve a hard problem. Demonstrating a prototype is challenging and does not ensure that a finished product can be produced. Regrettably, there is no assurance that funds will be reimbursed if a project fails, is fraudulent, or fails to arrive on schedule on Kickstarter or Indiegogo. For instance, the Zano project on Kickstarter raised almost $3 million, and the

PopSLATE 2 project on Indiegogo raised close to $1 million. A gadget for autonomous, intelligent, and swarming (sic) nano drones to take high-definition pictures and movies was what Zano promised to deliver, but the project was a failure without reimbursements and was of poor quality [3]. With PopSLATE 2, an iPhone smart case with an e-reader, a second screen, and more battery was promised. The most recent project update said that no money was available for reimbursements [4].

Smart Contract technology is not capable of solving all crowdfunding industry problems; however, it can ensure a fully transparent funding process with clear rules and regulations. Also, direct payment between parties can solve the issue of having intermediaries in our system. Usually, in traditional crowdfunding systems, intermediaries of the payment process are banks, credit card companies, and online payment gateways like PayPal or Stripe. Speaking of credit cards, the electronic payments industry is dominated by four companies. Mastercard, Visa, Discover, and American Express, most of the world's card payments are handled by those four [5]. Generally, each third party in any application adds a layer of dependency and requires trust. Especially in payment systems it also increases costs. That is why we decided to use Smart Contracts of the Ethereum blockchain to make a crowdfunding platform with the least layers of dependencies possible to provide transparency and security. Since we use Ethereum, we could make use of its other products like NFTs (non-fungible tokens). We used a similar concept of NFTs named SFTs to introduce reward-based crowdfunding in our platform Supply.

CHAPTER TWO

BACKGROUND

The main functionality of Supply is on the Ethereum blockchain. Blockchain has huge advantages such as audibility, anonymity, persistence, and decentralization. Blockchain like Bitcoin is a distributed ledger that records the addresses and balances of its users [6]. However, there are other blockchains like Ethereum that are more than a distributed ledger. But how did blockchain technology start? How does it provide these advantages? What are the blockchain critical opinions?

2.1 Blockchain

2.1.1 Crowdfunding Decentralized Apps

Blockchain technology has become a popular tool for crowdfunding projects due to its ability to facilitate trust, reduce transaction costs, and enhance the security and transparency of transactions. This part of our literature review will focus on the various works that have tackled the same problem as crowdfunding with blockchain, the various methodologies used to resolve the issue, and the advantages and disadvantages of using blockchain technology in crowdfunding.

Decentralized applications (DApps) utilize blockchain's features to create decentralized platforms, eliminating the need for intermediaries and enhancing trust among participants. Traditional crowdfunding platforms, such as Kickstarter and Indiegogo, have limitations regarding trust, accountability, and fee structures. Blockchain-based crowdfunding DApps address these issues by leveraging smart contracts, tokenization, and decentralized governance mechanisms [7].

Research into blockchain and crowdfunding has been conducted in several areas, including economic, legal, and security implications. For instance, research by Colwell examined the determinants of success and failure in equity crowdfunding using the Internet, while Follain and Mollick explored the legal and economic implications of blockchain-based crowdfunding.

Research by Li explored the security implications of crowdfunding with blockchain, and research by Mavridis explored the use of Smart Contracts for crowdfunding [8].

The various methodologies used to integrate crowdfunding with blockchain vary depending on the particular use case. For example, blockchain technology can be used to reduce transaction fees and provide secure, immutable ledgers for crowdfunding transactions. Smart Contracts can also be used to automate certain aspects of the crowdfunding process, and blockchain technology can help to reduce the amount of bureaucracy associated with traditional crowdfunding platforms. Additionally, blockchain technology can provide additional layers of security by providing a distributed record of transactions, allowing for the identification of fraudulent activity. Overall, blockchain technology has become a powerful tool for crowdfunding activities due to its ability to facilitate trust, reduce transaction costs, and enhance the security and transparency of transactions. However, it is important to consider the advantages and disadvantages, as well as the potential regulatory issues, when deciding whether to use blockchain technology in crowdfunding.

Decentralized applications are applications that run on a decentralized network, most probably blockchain. While they are still relatively new, DApps have been gaining traction in recent years due to their potential to provide secure and transparent services. This literature review will examine the various types of DApps, the advantages and disadvantages of using DApps, and the potential use cases and applications for DApps.

DApps are typically categorized into three types: financial DApps, data DApps, and computing DApps. Financial DApps are applications that enable users to store, exchange, and manage digital assets, such as cryptocurrencies. Data DApps are applications that allow users to store, manage, and share data securely and transparently. Computing DApps are applications that enable users to utilize computing resources, such as storage and computing power, on a decentralized network.

The advantages of using DApps include enhanced security and transparency, reduced costs, and increased trust and efficiency. DApps are typically more secure than traditional applications because they are not centrally managed and rely on distributed consensus protocols to ensure the

accuracy and integrity of data. Additionally, DApps are typically more transparent than regular applications, as the data stored on the blockchain is publicly available. Furthermore, DApps can reduce costs associated with traditional applications by eliminating the need for a third-party intermediary. Finally, DApps can increase trust and efficiency by providing a secure and transparent platform for users to interact with one another.

The potential use cases and applications for DApps are vast, including financial services, data storage, gaming, and healthcare. For instance, financial DApps can be used to facilitate payments, store digital assets, and enable peer-to-peer lending. Data DApps can be used to store and share data securely and transparently, such as medical records or confidential documents. Gaming DApps can be used to create virtual worlds and enable users to play games with one another. Finally, healthcare DApps can be used to store and share medical records securely and transparently, and to enable healthcare providers to communicate with one another.

In addition to the potential applications of DApps, it is important to consider the potential risks associated with using DApps. As with any technology, there are potential security risks such as the risk of data tampering or theft. Additionally, due to the decentralized nature of blockchain networks, it can be difficult to track and trace malicious activity. Furthermore, DApps may not be as reliable as regular applications due to the distributed nature of their infrastructure.

Features of Blockchain technology in Crowdfunding Applications:

- Tokenization, representing assets or rights on the blockchain, enables fractional ownership and liquidity in crowdfunding campaigns. Smart contracts, and programmable self-executing agreements, automate the distribution of funds, establish rules, and ensure transparency. These features reduce fraud, increase efficiency, and empower participants [8].
- Blockchain technology provides transparent and immutable records of transactions, enhancing accountability in crowdfunding campaigns. Contributors can verify fund allocation, project progress, and use of funds, minimizing the risk of fraudulent activities.

Additionally, decentralized governance models enable community-driven decision-making, promoting fairness and inclusivity.

- The decentralized nature of blockchain technology eliminates the reliance on a central authority, reducing the risk of data breaches and hacking attacks. The use of cryptographic techniques ensures secure transactions and protects sensitive information. Trust is enhanced as contributors can track the movement of funds and verify the integrity of the crowdfunding process [9].
- Despite the potential benefits, crowdfunding blockchain DApps face challenges such as scalability, user adoption, regulatory compliance, and the risk of token market volatility. Technical limitations, including network congestion and high transaction fees, hinder the widespread adoption of blockchain-based crowdfunding platforms. Addressing these challenges is crucial for the long-term success of such DApps [8].
- The integration of crowdfunding with blockchain technology holds promising opportunities for innovation. Further research is needed to explore hybrid models that combine the advantages of blockchain with other emerging technologies like artificial intelligence and the Internet of Things (IoT). The evolution of decentralized identity solutions and interoperability among different blockchain networks can also enhance the functionality and usability of crowdfunding DApps [10].

2.1.2 History

In 2008, a research paper titled “Bitcoin: A Peer-to-Peer Electronic Cash System” has been published by anonymous Satoshi Nakamoto [11]. The author had an ambitious project which was to make an electronic global currency that takes place over all traditional currencies and named it Bitcoin. The ambition that others had before in the 1980s and the 1990s. Those were reliant on a primitive cryptographic technique known as Chaumian blinding and they failed because they relied on centralized intermediaries that could manipulate the currency the way they want. In 1998, there was the first proposal to make a more decentralized electronic currency. It was proposed by Wei Dai and called b-money [12]. That proposal suggested creating electronic money through a consensus algorithm which is solving hard mathematical puzzles as they consume tons of resources, hence costing money. The implementation of that proposal never saw the light because of weak agreement on consensus algorithm details. The last try before Bitcoin

was in 2005 by Hal Finny. He introduced a concept named Prow (Reusable Proofs of Work) [13]. This concept used a computationally expensive algorithm named Hashcash puzzle which was made by Adam Back [14]. Along with b-money, Hal created a true cryptocurrency. Unfortunately, this project fell short of its ideals and eventually used centralized backend solutions. Here came Bitcoin to make use of all these attempts and prove an ideal decentralized monetary system. "*A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution*" by these words Satoshi Nakamoto started his paper by pointing out the objective of Bitcoin.

By using public/private key encryption, digital signature encryption, SHA-256 hashing (Secure Hash Algorithm 256 bits), peer-to-peer network, and consensus, the Bitcoin proposal proved its ability to make a pure decentralized digital coin system. But how could Bitcoin do this?

2.1.3 Public/Private Key Asymmetric Encryption

Bitcoin uses public/private key asymmetric encryption to allow its users to create addresses by themselves without a need for a central authority like a bank. The user chooses any word to be their private key then this encryption algorithm uses an elliptic curve to generate the unique public key for this chosen private key. Because there is no way to guarantee that two users choose the same private key it is a must to use a very long unpredictable private key. For example, Bitcoin users make a private key of 256 random characters.

2.1.4 Digital Signature Encryption

Bitcoin uses digital signature encryption to prevent malicious users from spending other users' coins. Meaning, each transaction a user does should be encrypted by his/her private key so other users can only decrypt his/her transaction by his/her public key. This way, only the one with the private key can issue transactions without exposing their private key to others. Asymmetric (public/private key) encryption methods are incorporated by all blockchain protocols due to their high security even if they are much slower than their peers, symmetric encryption methods.

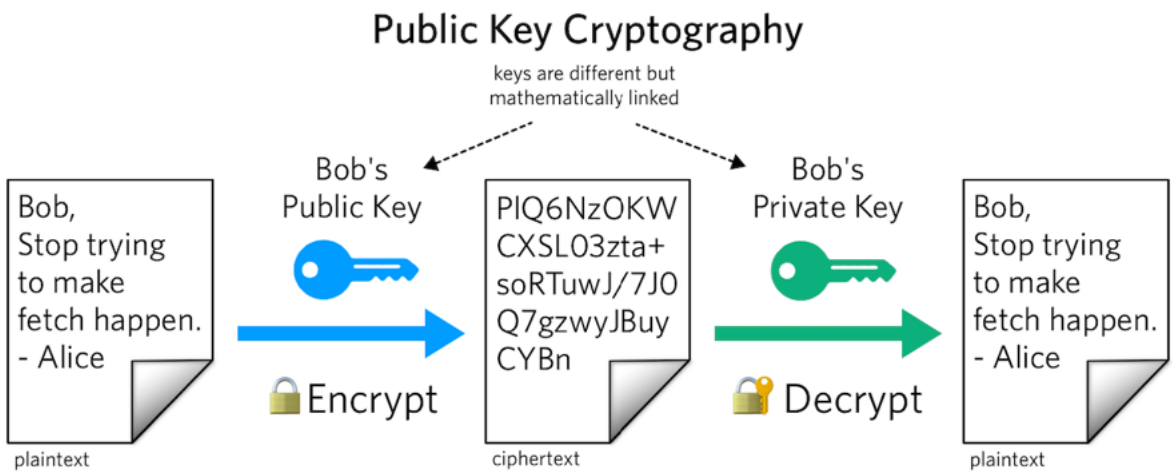

Figure 2-1. Public/Private Key Encryption [15]

2.1.5 SHA-256 Hashing

Bitcoin utilizes SHA-256 one-way hashing function to secure data from malicious modifications. SHA-256 is a hash function that takes any input and always outputs a unique fixed-length hash which can be used as a digital fingerprint. Even a tiny alteration to the input changes the output hash drastically.

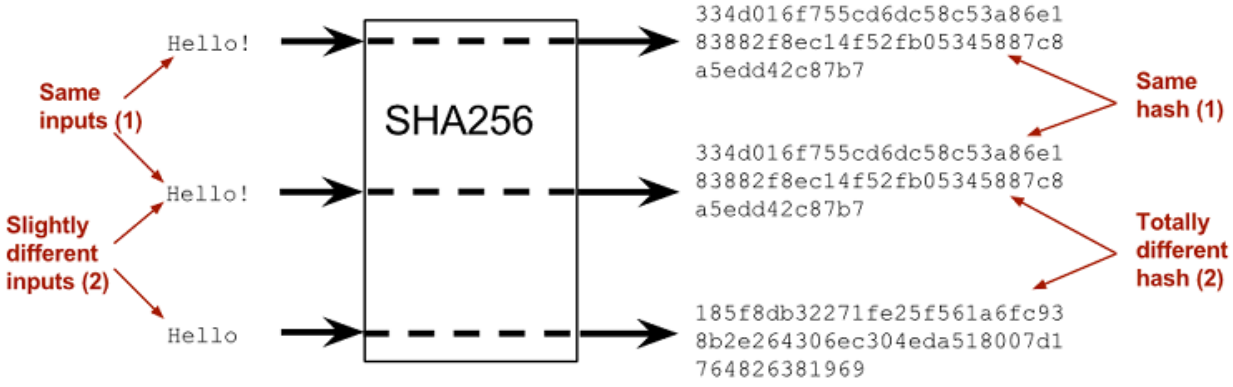

Figure 2-2. SHA-256 Hash Function [16]

Bitcoin hashes each transaction in each block and to save block header disk space, all transaction in each block gets hashed in a Merkle Tree data structure [17]. The reason behind pursuing small block headers is to provide light users with a small version of the blockchain, so, anyone can validate transactions without running a full node which usually requires hundreds of terabytes of disk space. By hashing transactions into a root hash that is kept in the header and

then hashing the header elements we get the block hash which is a fingerprint for each block that changes completely for every subtle modification. Since the blocks in the chain are pointing back to each other, then change in one block would change all nodes after it. Moreover, changing one node would be very costly because it requires remining all blocks after it. Mining is a very special concept for blockchain which we will address when we discuss consensus. Bitcoin utilizes hashing in various ways for many reasons. The huge gain of SHA-256 hashing is to detect any small malicious alternation in any block easily. Leaving no chance for attackers to alternate the data of the blockchain without getting detected easily.

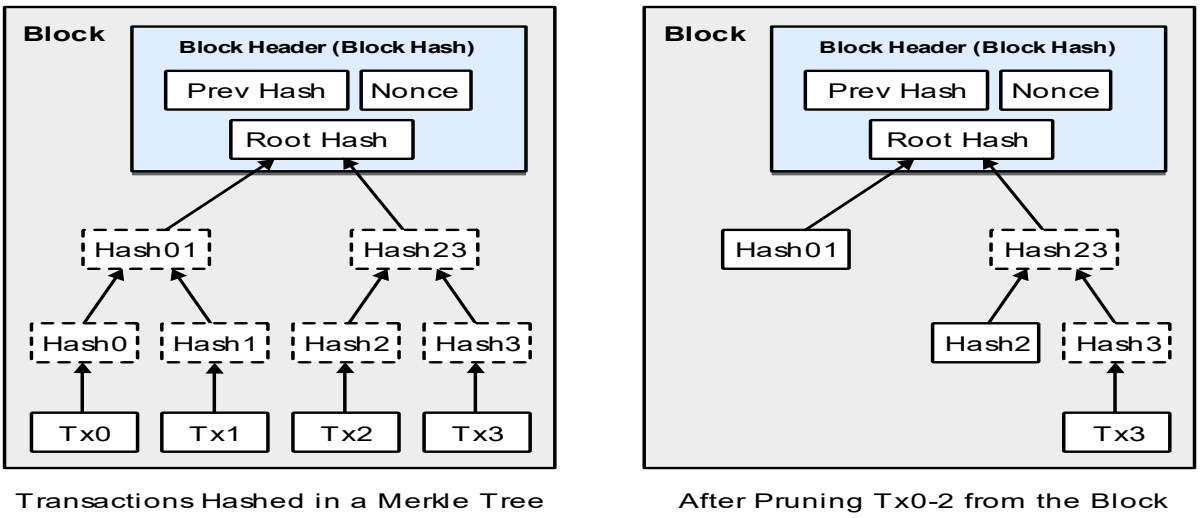

Figure 2-1. Bitcoin Block Body [11]

2.1.6 Bitcoin peer-to-peer Network Protocol

Bitcoin nodes communicate over a protocol named Bitcoin Protocol which is a protocol for a peer-to-peer network. BitTorrent was the most popular peer-to-peer protocol before the Bitcoin protocol. They share some similarities and differences. A node inside a peer-to-peer network starts to find its peers. No node can be connected to all nodes on the network. This network facilitates propagating transactions to something called Memory Pool, so miners include those transactions in new blocks. Miners also propagate new blocks when they get mined to other nodes. Notably, many open-source software implementations connect to the Bitcoin protocol on the default port 8333 to fetch and post transactions and blocks [18]. Each software has its way of finding peers, transactions, and block propagation and syncing. Aside from these community

efforts, blockchain is a distributed database that depends heavily on its users' communication to get updated and verified. Thus, people use peer-to-peer network because it is a decentralized and private network which is harmonious with the blockchain manifest. Satoshi outlined the network role in the following steps."

1) New transactions are broadcast to all nodes.

2) Each node collects new transactions into a block.

3) Each node works on finding a difficult proof-of-work for its block.

4) When a node finds a proof of work, it broadcasts the block to all nodes.

5) Nodes accept the block only if all transactions in it are valid and not already spent.

6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash."

2.1.7 Bitcoin Consensus

Bitcoin uses many encryption algorithms to secure the money transfer process without relying on centralized entities to achieve transparency. Nonetheless, there are yet many rules that can't be enforced without human intervention. For example, we have said that if the transaction message can be decrypted by the accompanied public key, then it has been signed by the private key of this public key. Now we also need all nodes to do this public key decryption check and if the message can't be decrypted then this must be a fraudulent transaction and should not be added to any block. This manual check is a part of Bitcoin consensus which is the process of nodes validating transactions.

One of main Satoshi's concerns in his paper was to solve the issue of double-spending. This means that a user might make two correctly digitally signed transactions in one block but spend money in the second transaction that supposedly has been spent by the first transaction. Only this sort of malicious action can be detected manually by each node. Satoshi said "*Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required*

to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network". Thus, he made it the responsibility of each node to follow all consensus rules that have been announced since day 1. Generally, if some node doesn't follow the rules, it would be obvious that it is completely different than others unless 51% of other nodes follow these not agreed-upon rules. Also, if one node has computational power more than 51% of other nodes it would destroy Bitcoin because there is a consensus rule named longest-chain rule which assumes that the longest chain is always the valid one. He and we believe this 51% attack is very unlikely to happen.

One of the most important consensus rules in Bitcoin is proof-of-work algorithm. The value of Bitcoin is proportional to the amount of computational work done to mine (create) each unit. Satoshi used a very similar algorithm to Adam Back's Hashcash puzzle. Given all next block data, the puzzle is to find a number called the nonce that makes the SHA-256 hash of the next block have leading zeros equal to the block difficulty. This puzzle is very computationally expensive as its cost increases exponentially with every leading zero especially since there is no other choice than iterating the nonce from 0 to infinity. There is a consensus rule that decides the difficulty of a block depending on the block's per-hour rate. By consensus, the first one who mines the new block gives themselves the reward and each transaction fee which is a fee that sometimes gets paid on top of a transaction to get it accepted faster. These rewards have a special transaction which is the first transaction in the block that is called The Coinbase transaction. Simply, if someone gives themselves the rewards without being the competition winner, their blockchain would be different from others which in turn would make it not agreed-upon, then, useless.

To conclude, there are conventional rules for blockchain miners. They are determined at the start of the blockchain in a paper called the White Paper which is the constitution of this blockchain. These rules must be very concise, so no debates happen about them. Also, if a rule is done programmatically then it should be deterministic, so all nodes produce the exact same output. There is a reason why blockchains use very long numbers for their native coins like Satoshi coin and Wei coin. The reason is calculating float numbers is known to be a very problematic topic in computer science [19]. Without diving into details, calculating float

numbers can’t produce deterministic outputs which wouldn’t make any consensus among nodes. As a rule, all consensus rules must be crystal clear and deterministic.

2.1.8 Blockchain From Critical Viewpoints

One common critical viewpoint center around scalability issues in blockchain networks. As blockchain grows, the size of this distributed database also expands, leading to increased storage and processing requirements. This can result in reduced transaction speeds and higher costs, hindering the technology's scalability. But there are solutions to this, such as sharding, off-chain transactions, or layer-two scaling solutions [20].

Another critical viewpoint revolves around the environmental impact of blockchain networks, particularly those that rely on proof-of-work (PoW) consensus mechanisms like Bitcoin. Critics argue that the energy-intensive PoW mining process consumes significant electricity, leading to a substantial carbon footprint. Notably, only one miner wins the block reward and all others’ efforts are in vain. That is too much energy consumed for a lost cause, literally. Research articles addressing this concern propose alternative consensus mechanisms, such as proof-of-stake (PoS) of Ethereum, which require significantly less energy and have a lower environmental impact [21].

Privacy and data protection are other areas of concern within the blockchain space. While blockchain networks offer transparency and immutability, they also raise questions about the confidentiality of sensitive information recorded on the ledger. We believe many businesses do not think of blockchain as a backend service as there is no ability to keep business secrets. So, researchers are working on implementing privacy-preserving mechanisms like zero-knowledge proofs or differential privacy in blockchain systems to address these concerns [22].

For us, we also have some concerns about Blockchain decentralization. Blockchain Whales, Whales, in any market, are the people who control most of this market share. This can be tolerable in centralized markets but in blockchain, this could destroy the whole decentralization cause. Wallets, many users use software digital wallets like Metmask which require trust. Wallets save users’ private keys and sign transactions with them on behalf of their users. That’s why many people buy physical wallets, or they create pair keys for themselves and sign

transactions for themselves which requires technical expertise. Node Providers, storing hundreds of terabytes of blockchain data is not so feasible. So, people use blockchain node cloud providers like Infura as if they have a full blockchain on their computers. Satoshi Nakamoto thought of this and made the idea of light nodes as users can download only blocks' headers and depend on people who have the full blockchain to get the data of blocks' bodies, as we discussed earlier. That is true. But running a light node still needs setup and some technical expertise. Those node providers lead the blockchain to centralization.

Overall, critical viewpoints on research articles about blockchain technology serve as essential tools for advancing the field. They shed light on challenges and limitations that need to be addressed and stimulate further research and development to make blockchain systems more efficient, sustainable, secure, and privacy-preserving. As technology continues to evolve, constructive criticism plays a vital role in steering blockchain toward a more robust and inclusive future.

2.2 Ethereum

Ethereum is a distributed Turing-complete state machine. The Ethereum state is a big data structure named modified Merkle Patricia Trie. Unlike Bitcoin, Ethereum state holds much more data than addresses and balances. This state changes with each new block and these changes abide by the rules that are defined by Ethereum Virtual Machine (EVM). Ethereum has a state transition function Equation (1) that must give a deterministic output. Given the old state S and new transactions T, the Ethereum state transition function gives a new state. Transactions in Ethereum are not necessarily related to transferring and minting money, unlike Bitcoin. Instead, they might be a call to create a smart contract or a call to execute an existing smart contract byte code [23]. Ethereum can be described as an extension of Bitcoin that aims for similar and different goals. It uses a unified virtual machine to do all blockchain operations like validating transactions and other operations. Instead of the Bitcoin way that every node does operations on its machine which could lead to different outputs between nodes, though a minimal chance of error, but possible.

$$Y(S, T) = S' \qquad (1)$$

2.2.1 Smart Contract

A smart contract is a program code that resides at a specific address on the EVM and runs on it to change its state. Smart contracts cannot be modified or deleted and all interactions with them are irreversible. Also, their code is the only way to modify the Ethereum state [24]. However, there are many programming languages to write smart contracts, they all compile down to Ethereum Assembly bytecode which is the only language the Ethereum machine understands. We use a high-level strict-typed language named Solidity. One of the biggest advantages of EVM smart contracts is their composability as they can be thought of as open APIs. Meaning that contracts can use other contract codes and data and even deploy new contracts. Having this advantage, we have used Uniswap contracts to add the functionality of swapping Crypto tokens. Notwithstanding, smart contracts have a big limitation as they cannot communicate with any other code or data sources outside of the EVM. That would require using HTTP protocol which would jeopardize the consensus, decentralization, and security [25].

2.2.2 Gas Fees

One of the most pressing matters of Ethereum is Gas fees. In order to execute smart contracts code and for the network to continue in general, people who execute transactions on the blockchain must pay fees to the validator of the next block who stores their transactions. Gas in Ethereum is a unit that measures the computational effort needed to execute a transaction as one computational step costs 1 Gas. In a supply-and-demand auction format, validators receive fees for processing transactions. Issues happen when the network has a bottleneck, in that case, Gas fees can skyrocket. For Instance, in 2021, average Gas fees on Ethereum have gone from 4 dollars all the way to 44 dollars in 6 months [26]. Gas fees are the main concern of all Ethereum

users and developers alike because their prices are highly volatile and unreliable.

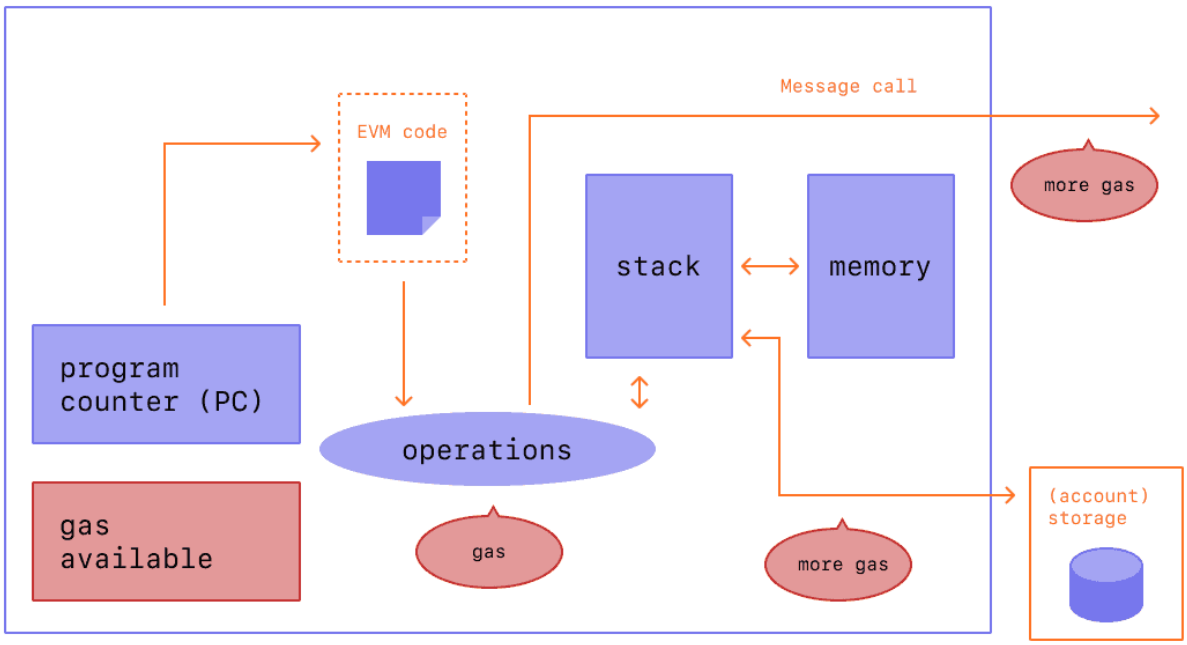

Figure 2-2. EVM [27]

2.2.3 ERC-20 Token

ERC-20 tokens are fungible tokens that work the same way as fiat currency. All Cryptocurrencies that are made on top of Ethereum Blockchain of this token type. In 2015, Ethereum got more attention and success which addressed many issues. People created many currencies that could not be trusted or used properly. A developer named Fabin Vogelstellar wrote the 20th comment on the GitHub page Ethereum Request for Comment (ERC) and proposed a standard model for fungible tokens [28]. People used this standard mainly to implement voting tokens and fait-like coins. For instance, Dai stablecoin, the coin we raise our crowdfunding campaigns in. We also accept other ERC-20 currencies but first, we use the famous decentralized exchange protocol Uniswap to swap these currencies with DAI currency.

2.2.4 ERC-1155 Token

ERC-1155 token is known to be a fungibility-agnostic and gas-efficient standard token. Its distinctive feature is to represent the fungible and non-fungible tokens at once. ERC-721 is a non-fungible token (NFT) that represents unique properties that only one person can hold.

Mixing ERC-20 with ERC-721 tokens is the concept that ERC-1155 tokens are based on [29]. Meaning that many people can hold a unique property which is often exampled by game items. We give ERC-1155 tokens as rewards for people who participate in crowdfunding campaigns not only as incentives, but also as valuable gifts.

2.3 Web Server: Model View Controller (MVC)

Our Blockchain application is not fully decentralized as apps can't achieve full decentralization today considering that adding data to the blockchain is very expensive. That is why we ought to use different data stores like NoSQL databases or interplanetary file system (IPFS) which is a decentralized way to store files but not secure and reliable as much as blockchain. Every data store of those 3 has its own best usage. Connecting with NoSQL database or IPFS unlike blockchain peer-to-peer network can only be done through HTTP or IPFS protocols using IP address which is why we needed to set up a monolithic Web server to communicate with other data stores and to serve our website's front-end static files.

2.3.1 History

Supply uses the Web server that follows the design pattern of Model View Controller which was first envisioned by Trygve Reenskaug at the Xerox Parc in the 1970s. He needed to connect the user's model to the computer model. As he said, "The essential purpose of MVC is to bridge the gap between the human user's mental model and the digital model that exists in the computer" [30]. Later, in 1988, Pope and Krasner had an article in the Journal of Object-Oriented Programming named "A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80". It described the MVC paradigm in the following words "*Isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each specific unit without having to know everything about any other unit*". [31]

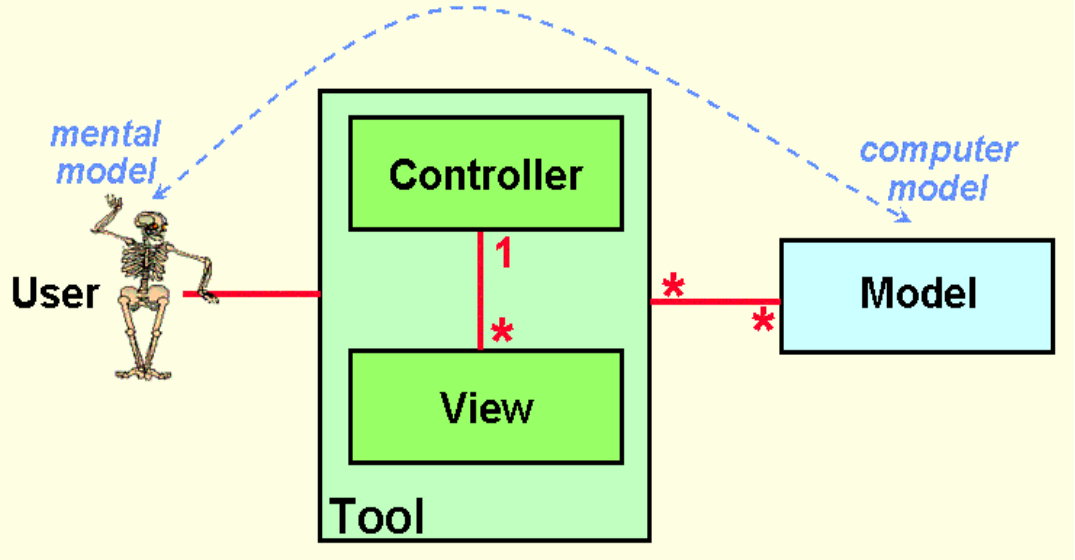

Figure 2-3. The MVC Pattern [30]

The MVC model is suitable for Web application development because usually it is just about integrating different technologies [32]. In Sum, View is a Web browser, Controller is an express.js backend server, and Model is MongoDB NoSQL database.

2.3.2 ***Controller**: Web Server*

A web server usually is a piece of software that runs on hardware that is optimized to accept users' requests via Hypertext Transfer Protocol (HTTP) which is an application layer of Transmission Control Protocol (TCP) in the Open Systems Interconnection model (OSI model). Web servers' role is to accept requests from Web browsers then call some functions and return the output in response. In other words, a Web server gives the ability to a Web user to run some code that is on another machine and then get the result. For example, if a user wants to see a Web page and has its URL; this Web page is not on their machine, so they need to ask another machine that has it to send it to them. This other machine is called the Web server and this URL carries out the name of this server and the name of the requested page. Communicating with other machines programmatically is called API (application programming interface).

API: APIs define the rules that applications need to follow to communicate with other applications. This process is troublesome as developers try to communicate with every

programming language from many different environments. That's why we need more guidelines to make transferring data between applications much more standardized like REST APIs.

REST API: We have used REST APIs to make communications with our Web server. REST APIs come with many benefits such as scalability and independence. REST APIs scalability and independence comes from statelessness which means that every request is completed independently of the previous ones. Also, their independence allows developers to write both client and server applications in different programming languages without affecting the API design.

Node.js: Node.js JavaScript runtime was created by Ryan Dahl in 2009. He wanted to get JavaScript running outside browsers, to run on operating systems. Traditionally, JavaScript is a single-threaded language. As it cannot create new threads because it does not interact with operating systems. However, Web browsers can create threads. JavaScript Node runtime does not also interact with operating systems and uses a library called LibUV which is written in C++. It provides Node with threading, I/O operations, network operations, hashing, and other features. This library is what makes Node Web servers advantageous on other Web servers. Node provides a high-performance, asynchronous event-based server. It is event-based and uses a single thread for its event loop, this allows developers to use asynchronous interfaces to do I/O operations. While other Web servers follow the multiple-thread request model. Using multiple threads could result in many unexpected behaviors which could lead to fatal errors. Web servers and databases are very sensitive to errors because they could break the servers down, lead to a security flaw, or just mess up the data. In return, Node is not very great at handling intensive computational work. These types of operations usually need multi-threading while JavaScript language itself executes on a single thread [33].

Express.js: Creating Web servers with Node.js could make use of its benefits. Express framework provides a thin layer of fundamental Web application features to make server development more straightforward and maintainable [34]. The framework implements HTTP request/response cycle in a stack of functions called Middleware functions. Express.js is a lightweight framework. It does not do much. It is mainly a router for HTTP endpoints and has a rich ecosystem that facilitates using other people's modules [35].

2.3.3 ***Model**: MongoDB*

MongoDB is the most popular NoSQL database. For building data stores, it is a great tool, especially because of its ability to fully utilize so-called “sharding-nothing cluster architecture”. NoSQL databases do not follow the most famous and successful model of Relational databases. Alternatively, they use schema-less design which means collections (tables in the relational model) don’t have the same set of fields or structure for documents (rows in the relational model) and that is great for large data which can often be unpredictable in its structure. In addition, NoSQL databases can scale horizontally which means the Distribution of a single logical database system across a cluster of machines, also known as Sharding. We use MongoDB to store off-chain data that is related to the Website. NoSQL databases are suitable to do operations with blockchain data more than relational databases as blockchain data have different types and structures than usual data. Which could lead to troubles with relational databases because of their rigorous schema design [36].

Mongoose ODM: Since writing MongoDB validation, casting, and business logic boilerplate is tiresome we have used Mongoose. It is an (object document mapping) ODM to simplify business logic in our Node API apps. As Mongoose package creators described it “Mongoose provides a straight-forward, schema-based solution to model your data. It includes built-in validation, type casting, business logic hooks, query building.” [37]

2.3.4 ***View**: Single-Page Application (SPA)*

Early days of Web pages have been suffering from user low connectivity despite being popular [38]. Starting from the early 2000s, new techniques and concepts have been invented to improve the Web functionalities such as AJAX (Asynchronous JavaScript and XML). The whole technique was made up to allow Web pages to update page content without refreshing the page which is later replaced by Single-Page Application [39]. There is no exact definition for the Single-Page Application technique, but we can say A. Mesbah and A. van Deursen’s definition is “*the single-page web interface is composed of individual components which can be updated/replaced independently so that the entire page does not need to be reloaded on each user action*” [38]. Figure 2-6 illustrates the communication of a SPA between a user and a Web

server. This technique is dominant in all JavaScript frameworks today Hence, we have used it using React.js.

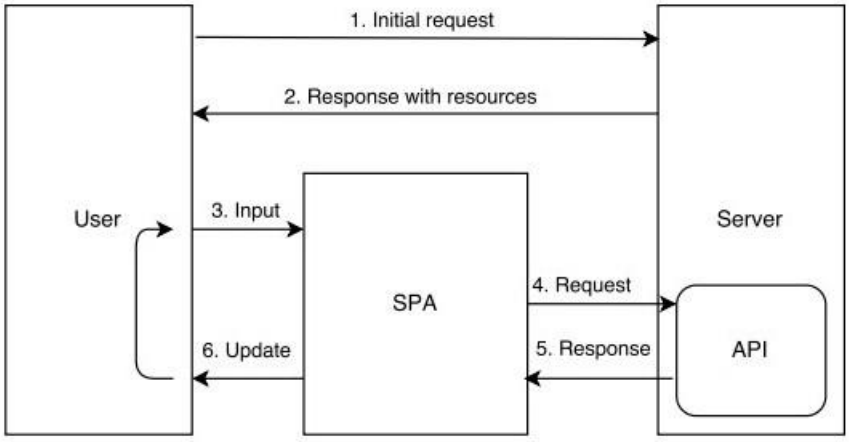

Figure 2-4. Communication Between User and Web Server in SPA [40]

React.js: React is a framework developed by Meta Inc. ReactJS is known for using the concept of reusable components also, for being highly efficient because of the virtual document object model (DOM) feature. ReactJS has a virtual DOM inside the memory. When there is a change to be displayed on the Web page, instead of instantly updating the DOM, it changes first in the virtual DOM. Then an algorithm called diff is used to compare the DOM and the virtual DOM then desired and relevant nodes only of the DOM tree are updated. This provides high interactivity hence, a better user experience [41].

CHAPTER THREE

METHODOLOGY

Supply's main objective is to achieve a high level of decentralization, hence, transparency. After analyzing our project requirements, we have taken some technical and business decisions to ensure the application is as transparent, secure, and efficient as possible. We have outlined the project architecture using UML diagrams. We also have taken some deployment and development approaches such as using an object-oriented paradigm with Solidity programming language, using Functional Programming paradigm with JavaScript, using Test-Driven Development (TDD), using Docker in deployment, using GitHub Actions for creating Continuous Integration (CI) pipelines, and many other tools.

3.1 Requirement Analysis

3.1.1 Functional Requirements:

- **User Registration**: the system should allow both fundraisers and campaign launchers to Login using their Digital Wallet accounts.
- **Campaign Creation**: campaign launchers should be able to create new crowdfunding campaigns by specifying details like campaign title, description, funding goal, and image.
- **Campaign Management**: campaign launchers should have the ability to manage their campaigns, including closing their campaigns and stopping accepting funds.
- **Funds**: users should be able to view campaigns, select a campaign they want to support, and fund campaigns using their wallets.
- **Refund**: The app should have a refunding policy. If a campaign hasn't reached 30% of its funding goal, then fundraisers should be able to refund.
- **Transparency**: the app should ensure transparency by providing a public ledger of all funds and transactions, accessible to both campaign launchers and fundraisers.

- **Smart Contract Integration**: the app should integrate with a blockchain network and utilize smart contracts to securely manage transactions, enforce campaign rules, and withdraw funds.
- **Withdrawal Funds**: If a campaign has reached 30% of its funding goal, then the campaign launcher should be able to withdraw all campaign funds.
- **Payment Options**: Users should be able to fund campaigns in Eth and any ERC-20 token.

3.1.2 Non-Functional Requirements:

- **Security**: the app should ensure secure transactions and protect user data by implementing appropriate encryption mechanisms, and wallet authentication.
- **User-Friendly Interface**: the user interface of the app should be intuitive, easy to navigate, and visually appealing to users.
- **Performance**: the system should provide quick response times for user interactions, such as loading campaign details, and processing funding.
- **Latency**: system's API endpoints should have an average response time of less than 500ms.
- **Cost**: the system should not cost users more than 10 dollars for Gas fees per transaction
- **Compatibility**: the app should be compatible with multiple web browsers and devices, ensuring an excellent user experience across different platforms.
- **Error Handling**: the app should be robust in handling errors and exceptions, providing informative error messages to users.

3.2 Development & Deployment Tools

We have made some technical decisions during the development process. So, we have used various paradigms and tools to keep the development organized and maintainable. Using these tools becomes a necessity for developers when working in teams.

3.2.1 Functional Programming

Functional Programming is a programming paradigm that emphasizes the use of pure functions, immutable data, and declarative programming techniques. In Functional Programming, programs are built by composing and manipulating functions, treating them as first-class citizens. Unlike imperative programming, which focuses on changing the program state, Functional Programming focuses on adding to the program state instead of modifying it. Functional Programming focuses on two concepts: data is immutable, and functions are stateless [42]. We have used this paradigm in JavaScript server code as it provides five main features:

1. **Modularity**: Functional Programming promotes modularity by encouraging the decomposition of a problem into smaller, reusable functions. This modular approach enhances code readability, maintainability, and reusability.
2. **Immutability**: In Functional Programming, data is immutable, meaning it cannot be modified once created. Immutable data structures eliminate the risks associated with unintended side effects and enable easier code debugging.
3. **Pure Functions**: Pure Functions produce the same output for the same input, without causing any side effects. They simplify code understanding and unit testing, improve parallelization capabilities, and cause no unexpected behaviours.
4. **Higher-Order Functions**: It allows functions to take other functions as arguments or return them as values. They are used to reduce code duplication and to write clean code by separating the logic from implementation details.
5. **Concurrency and Parallelism**: The inherent immutability and absence of shared state in functional programming make it well-suited for concurrent and parallel programming. By avoiding mutable state, functional programs can better utilize multi-core architectures and simplify the management of concurrency.

3.2.2 Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that enables the creation of modular and reusable code by organizing data and behavior into objects. Solidity incorporates certain object-oriented concepts, allowing developers to leverage the advantages of OOP in their Ethereum smart contracts.

In Solidity, OOP is achieved through the use of contracts, which serve as the fundamental building blocks of smart contracts. A contract in Solidity encapsulates data and functions, providing a way to define and interact with objects [43]. These are the key features and benefits of using object-oriented programming in Solidity:

1. **Encapsulation**: Solidity contracts encapsulate related data and functions into a single unit, providing abstraction and modularity. This allows for better organization, code reuse, and separation of concerns, leading to more maintainable and extensible smart contracts.
2. **Inheritance**: Solidity supports inheritance, allowing contracts to inherit properties and functions from other contracts. Inheritance promotes code reuse and facilitates the creation of contract hierarchies, where child contracts inherit behaviours from parent contracts. This enables developers to build upon existing contracts and enhance or specialize their functionality.
3. **Polymorphism**: Solidity supports polymorphism through function overriding and function overloading. Polymorphism allows contracts to define functions with the same name but different implementations or input parameters, enabling more flexible and expressive contract design.
4. **Code Reusability**: By leveraging inheritance and polymorphism, object-oriented programming in Solidity promotes code reusability. We can create generic parent contracts that can be inherited and extended by multiple child contracts, reducing code duplication, and increasing development efficiency.
5. **Modifiability and Maintainability**: OOP principles in Solidity enhance the modifiability and maintainability of smart contracts. Encapsulating code into objects with well-defined boundaries makes it easier to understand, modify, and debug. Also, the use of inheritance and polymorphism enables us to make changes in a specific part of the contract hierarchy without affecting other parts.

3.2.3 Test-Driven Development

Test-Driven Development (TDD) is a software development approach that emphasizes writing tests before writing the actual code. It follows a cycle of writing a test, running the test, writing the code to pass the test, and then refactoring if necessary. TDD promotes the creation of reliable, maintainable, and almost bug-free code by ensuring that it is thoroughly tested. We have

used TDD with our Smart Contracts and Express.js API endpoints. For Smart Contracts testing, we have used Mocha for testing Chai for assertion. For API endpoints testing, we have used Jest framework and a library called Supertest.

1. **Jest** is a JavaScript testing framework that provides a simple and intuitive interface for writing unit tests, including tests for API endpoints. It offers a rich set of features, including test assertions, mocking, test runners, and code coverage analysis. Jest is well-suited for testing JavaScript code in various environments, including Node.js and browser [44].
2. **Supertest** is a library that works alongside testing frameworks like Jest and allows for testing HTTP endpoints by making requests and asserting responses. It provides a high-level API for sending HTTP requests and making assertions on the responses received. Supertest simplifies the testing of API endpoints by allowing developers to simulate requests and test the behaviour and correctness of the server responses [45].

3.2.4 Hardhat Framework

Hardhat is a popular development framework specifically designed for Ethereum smart contract development [46]. It provides a comprehensive set of tools and functionalities that streamline the process of compiling, testing, and deploying Solidity contracts. With Hardhat, we could efficiently build and maintain reliable Ethereum applications. Hardhat offers these key features:

1. **Smart Contract Compilation**: Hardhat simplifies the compilation of Solidity smart contracts. It integrates with the Solidity compiler and automatically compiles the contracts within the project. It supports different Solidity versions and offers options for custom configurations.
2. **Testing Environment**: Hardhat provides a testing environment that allows developers to write and execute tests for their smart contracts. It supports various testing frameworks such as Mocha and Chai, enabling the creation of test suites. The testing environment also offers features like test coverage analysis and mocking.
3. **Network Management**: Hardhat facilitates the management of Ethereum networks during development. It allows configuring and deployment local development networks for

testing purposes. Also, it supports integration with popular Ethereum test networks like Sepolia. One of the great features Hardhat offers is the ability to fork the Mainnet. We used this to test interactions of our contracts with existing Uniswap contracts with their current state on the Miannet. Without this Hardhat feature, our integration with Uniswap contracts project couldn't be tested, hence, couldn't be done.

4. **Deployment**: Hardhat enables easy deployment of smart contracts to Ethereum networks. It provides built-in deployment scripts and supports custom deployment scripts for complex deployment scenarios. Hardhat also allows for task automation.
5. **Plugin System**: Hardhat offers a flexible plugin system that extends the framework's functionalities. We have used a plugin called Hardhat-Gas-Reporter. It allows us to check the Gas usage for every Mocha test we have. That can help us make the app very effective in terms of cost and performance.

3.2.5 Git & Git Flow

Git is a distributed version control system widely used in software development to manage source code and track changes over time. It provides a robust and efficient way to collaborate on projects, maintain code history, and manage different versions of the codebase [47]. 4 key features that summarize Git:

1. **Distributed Architecture**: Git follows a distributed architecture, allowing multiple developers to work on a project simultaneously. Each developer has a local copy of the entire repository, enabling them to make changes and commit them independently.
2. **Branching and Merging**: Git offers powerful branching and merging capabilities, allowing developers to create multiple branches for different features or bug fixes. Branches can be easily created, switched between, and merged, enabling concurrent development and the isolation of changes.
3. **Commit and History Tracking**: Git tracks changes to the codebase through commits, which represent a snapshot of the project at a specific point in time. Developers can review commit history, view changes made, and roll back to previous versions if needed.
4. **Collaboration and Remote Repositories**: Git facilitates collaboration by supporting remote repositories. Developers can push their local changes to a shared remote repository, pull updates from others, and resolve conflicts during merging.

Git Flow is a branching model and workflow that provides guidelines for managing branches and releases in a structured manner. It offers a branching strategy designed to optimize collaboration, feature development, and release management. Git Flow includes the following branches:

1. **Main Branch**: Represents the stable and production-ready codebase. Changes in this branch are usually deployed to the live environment.
2. **Development Branch (develop)**: Serves as the integration branch for ongoing development. Feature branches are merged into this branch for testing and integration.
3. **Feature Branches**: Created for developing new features or enhancements. Each feature branch is based on the development branch and is eventually merged back into it.
4. **Release Branches**: Created when preparing for a new release. Bug fixes and final adjustments are made in this branch before merging into the main branch.
5. **Hotfix Branches**: Used to quickly address critical issues in the production codebase. Hotfixes are based on the main branch and merged into both main and development branches.

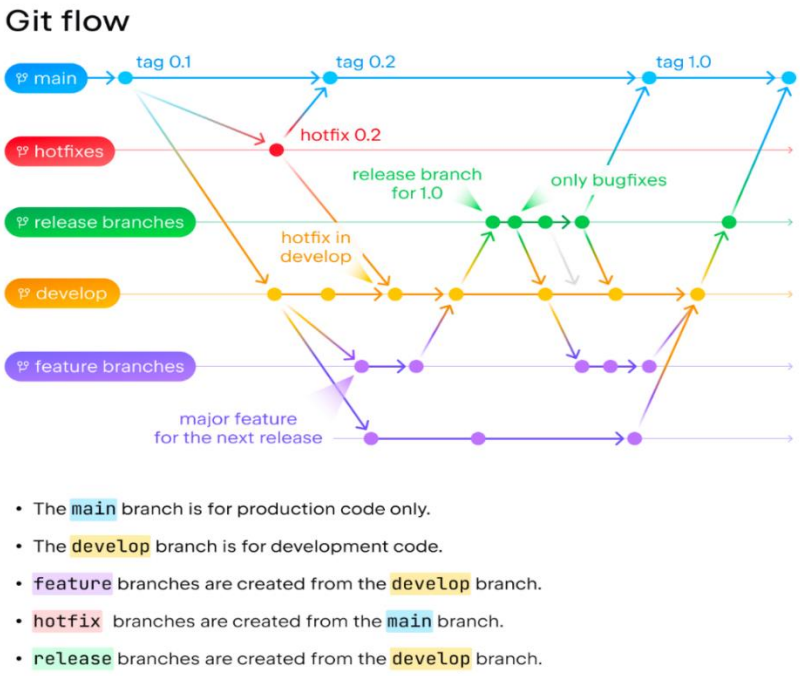

Figure 3-1. Git Flow [48]

3.2.6 GitHub Actions CI:

GitHub Actions is a powerful CI (Continuous Integration) platform provided by GitHub. It enables to automate various aspects of the software development lifecycle, including continuous integration, continuous deployment, and workflow automation. With GitHub Actions, we define workflows that automatically build, test, and deploy the frontend and server code in different environments like different Operating Systems or different versions of programs, increasing development efficiency and ensuring code quality [49].

1. **Workflow Automation**: GitHub Actions allowed us to define custom workflows using YAML syntax. Workflows are composed of one or more jobs, each containing a set of steps to be executed. Workflows can be triggered based on specific events, such as pushes to a repository, and pull requests.
2. **Continuous Integration** (CI): With GitHub Actions, we could set up CI pipelines to automatically build and test. CI workflows can include steps for compiling code, running unit tests, performing code quality checks, and generating artifacts.

3.2.7 Docker

Docker is a popular containerization platform that allows developers to package an application and its dependencies into a container. This container can be easily deployed and run consistently across different environments, making it an excellent choice for website deployment.

For the deployment of our website, we opted to leverage Docker to ensure a seamless and consistent deployment process. Docker allowed us to package the entire web application, including its runtime, libraries, and configurations, into a single container. This container could then be deployed on various target environments without worrying about differences in the underlying infrastructure. The container also provides security to our server through its isolated nature.

3.3 User Interface Design

Mobile-first design is a design method that focuses on giving users better experiences by starting the designing process with the smallest of screens and then moving to bigger screens reaching the desktop screen size. The idea came to light in 2010 when Eric Schmidt, Google CEO back then, said at a conference that Google would focus more on mobile users in their design. He quoted "*What's really important right now is to get the mobile architecture right. Mobile will ultimately be the way you provision most of your services. The way I like to put it is, the answer should always be mobile-first. You should always put your best team and your best app on your mobile app."* [50]. We have 4 reasonings behind using mobile-first design:

1. **User Experience**: since mobile design is more limited, we focus more on adding the most important functionality and when we design bigger screen sizes, we start to add the less important functionality [51].
2. **Scalability**: it ensures we always use flexible CSS features like flexbox and grid layout as they are better for project scaling.
3. **Reachability**: Search Engine Optimization (SEO), as Google stated they started using mobile-friendliness as a ranking factor for mobile searchers. They quoted "*Getting good, relevant answers when you search shouldn't depend on what device you're using. You should get the best answer possible, whether you're on a phone, desktop, or tablet. We started using mobile-friendliness as a ranking signal on mobile searches. We'll start rolling out an update to mobile search results that increases the effect of the ranking signal to help our users find even more pages that are relevant and mobile-friendly*" [52].
4. **Accessibility**: according to Statista, there were 3.5 billion mobile users in 2020. Every year, there are 300 million new mobile users.

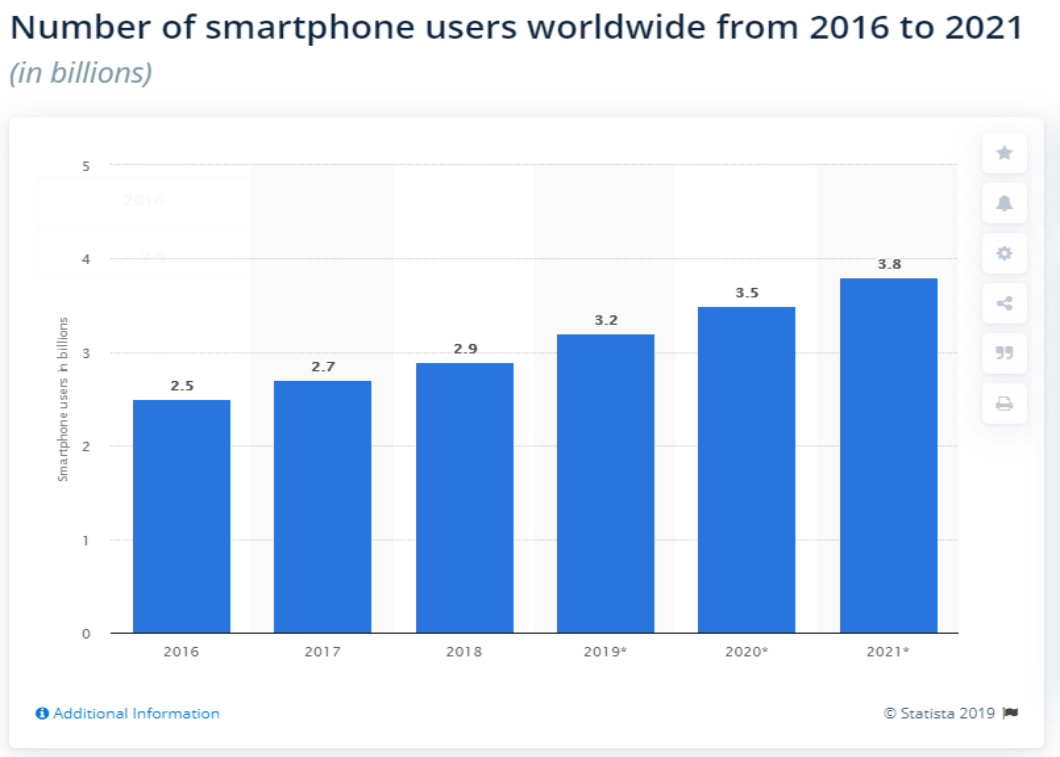

Figure 3-2. Number of Mobile Users Over Years [53]

3.4 Project Architecture

The following diagrams were made to describe how the project is structured and to describe user, and server interactions. They were made in The Unified Modeling Language (UML). UML has been described by some as “the lingua franca of software engineering” [54]. This Language is intended to standardize the way software developers visualize their system designs.

3.4.1 Use Case Diagram

A use case diagram is a fundamental tool in UML (Unified Modeling Language) used to outline the requirements of a new software program during development. It describes the expected behavior of the system without delving into the specific implementation details. Use cases are presented through both textual and visual means, with the use case diagram being one such representation. The main idea behind use case modeling is to design the system from the perspective of end users, allowing for effective communication of the system's behavior in terms understandable to the users. By defining all the externally visible system behavior, use case

diagrams aid in creating a clear and comprehensive view of the software's functionalities. In our system Use Case diagram, we have many actors to interact with.

The first main actor is the campaign launcher. A campaign launcher can primarily do 4 things: launch a campaign, edit or close his/her campaign, and withdraw funds he/she has in his/her campaign. Launching a campaign includes 3 operations: adding the main part of campaign data to the blockchain through CrowdCampaigns Ethereum Smart Contract, adding the secondary part of the campaign data to MongoDB, and uploading the campaign image to IPFS through the IPFS client Web3.storage. As for campaign editing, the campaign launcher can edit only the secondary campaign data which is on MongoDB. Also, the campaign launcher can close their campaign and stop accepting funds using CrowdCampaigns Smart Contact. Finally, the campaign launcher can withdraw funds for his/her campaign if the campaign has collected more than the campaign soft cap. This process is regulated in CrowdCampaigns Smart Contract.

The second main actor is the fundraiser. This actor can view campaigns, and fund, or refund a campaign. Funding campaigns can be done in 3 ways: funding with DAI, which is the only Crypto coin the campaigns accept payments with, funding with Ether, which is the native currency of Ethereum, and funding with other Ethereum Crypto coins. The latest two ways need an extra step to be accepted as a fund for a campaign. Ether and other Crypto coins than DAI must first be swapped with DAI through Uniswap DEX protocol Smart Contract. Finally, the fundraiser can make a refund from a campaign if the campaign has collected less than the campaign soft cap.

There is a secondary actor worth mentioning, that is MetaMask digital wallet. Here are some actions that can only be done if the actor is connected to MetaMask: launching, closing, funding a campaign, and withdrawing, or refunding funds from a campaign.

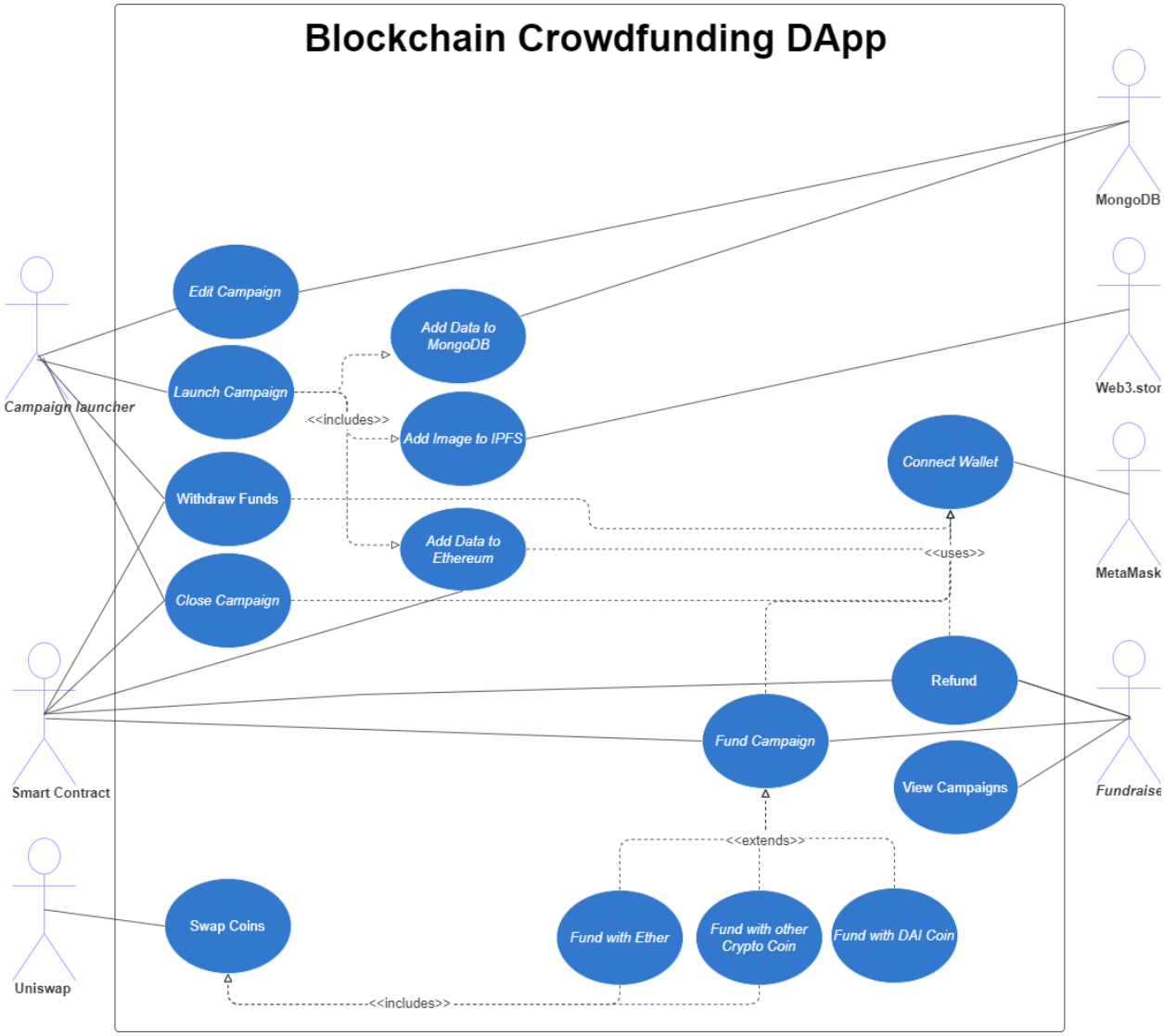

Figure 3-3. System's Use Case Diagram

3.4.2 Class Diagrams

The class diagram is made up of classes and the relationships between them. Each class has its attributes and methods, also, its relations with other classes in types of Aggregation, Inheritance, Composition, and so on [55].

We believe that contracts in Solidity are analogous to C++ classes [56]. Hence, visualizing Solidity Smart contracts in a Class Diagram is doable. In Figure 3-4, the main class is CrowdCampaigns which is the main Contract that has all the logic of crowdfunding campaigns. This Contract uses the interface of SwapRouter to swap funded tokens with DAI stablecoin that

the campaign creator accepts payment in, for the reason of its price stability. SwapRouter Contract already exists on the blockchain and was made by Uniswap DEX protocol. Since all contracts on the blockchain can use each other code that is marked as public, we have made use of the most famous coins decentralized exchange (DEX) platform Uniswap. This platform is not a central authority that owns the tokens and trades with others. Instead, it uses the idea of Liquidity Pools that facilitate Crypto coins trading based on the order book model. The same way the traditional stock exchanges work like Nasdaq but automatically without human interference [57].

CrowdCampaigns Contract uses the interface of ERC20 Contract to make it able to recognize all different Crypto tokens that are available on Ethereum, including DAI tokens. As for the WETH9 interface, this solves the predicament of Solidity treating Ether (the native currency of Ethereum) differently from other Crypto tokens. So, it is to transform Ether to Wrapped Ether which is an ERC20 token like all other tokens to finally get swapped with them normally.

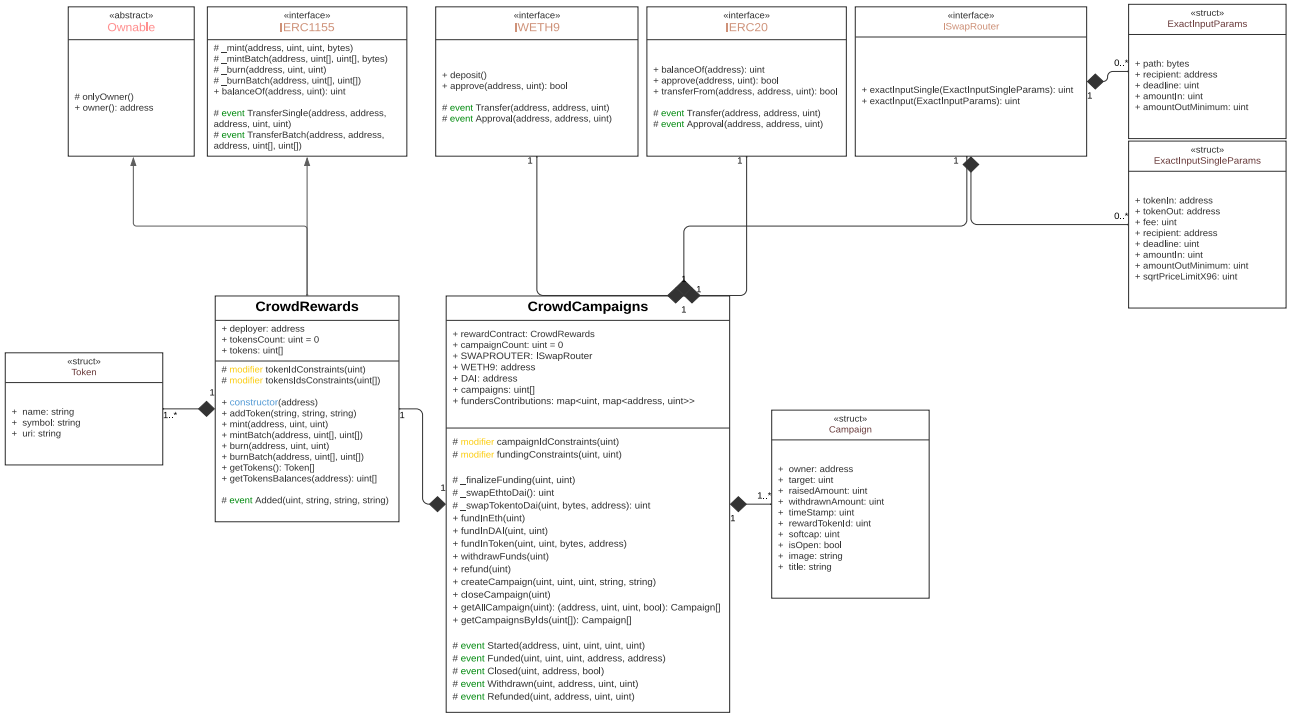

Figure 3-4. The Class Diagram of Supply's Smart Contracts

3.4.3 Sequence Diagrams

UML Sequence Diagrams are behavioral and interaction diagrams. This diagram captures how objects interact with each other in a time frame and an order of events. This kind of diagram represents objects on the vertical axis and their interaction on the horizontal axis. A sequence Diagram might be a perfect way to draw the order of events for some scenarios because the whole diagram is organized according to time. The time progresses as the arrow goes down [58].

In figure 3-5 sequence diagram it explains how the user interacts with the system, and how the system interacts with the blockchain. First, the user sends a message to the website which is the straight line to the website to initiate a campaign, then the website requests a campaign to be created in the blockchain. The blockchain then sends a retune massage which is the dotted line to the website where a campaign has been created, then the website shows a successful massage to the user signaling that the campaign was successfully created. The same happens when a user tries to fund a campaign of their choice, the only difference is that there are multiple conditions for the funds to be accepted, therefore there is an “if” condition in the diagram that if all the conditions are met then a successful massage will be shown to the user that he successfully funded. And, if not all conditions have been met then the user will be shown an error with the reason as to why it was rejected. The same thing happens if the user searched for a campaign or requests to close his campaign.

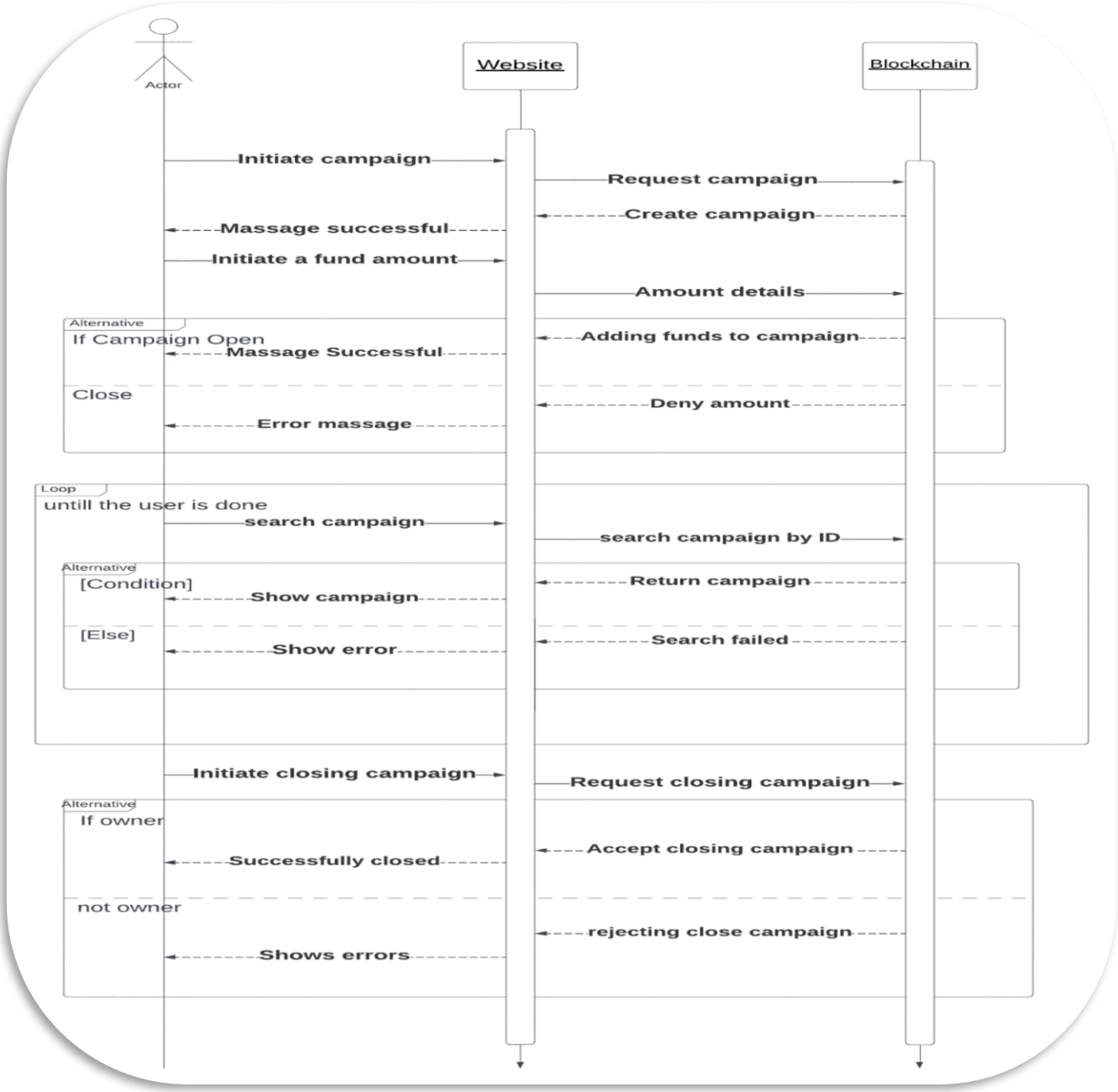

Figure 3-5. Sequence Diagram for Creating a New Crowdaunding Campaign Scenario

CHAPTER FOUR

RESULTS & DISCUSSION

With this design and method, we aimed to create a website that satisfies our requirements. Our objective from the beginning was to make a platform that provides a more transparent, cheaper, easier, and more direct crowdfunding process. To hit our objective, we have used decentralized technologies like Ethereum Smart Contracts for the backend and IPFS for file storage. In addition to other centralized technologies like Express.js for a backend web server and React.js for the website user interface.

Measuring the performance of our blockchain-based website can be done from 3 aspects: web user interface performance, API endpoint performance, and Smart Contract functions' Gas fees. Since Gas fees are calculated by 1 Gas for every computational step, then it is a performance and cost measure.

A user can do many activities in a crowdfunding app. Anyone who browses the platform can see all the campaigns listed and explore more about each campaign. Supply mainly uses JavaScript libraries to create the user interface, handle user inputs and communicate with the Ethereum network. The main page shows a card preview of the campaigns that are fetched from the server that fetches data from CrowdCampaigns Smart Contract. Each card leads to a detailed page for its campaign. Here is the main page:

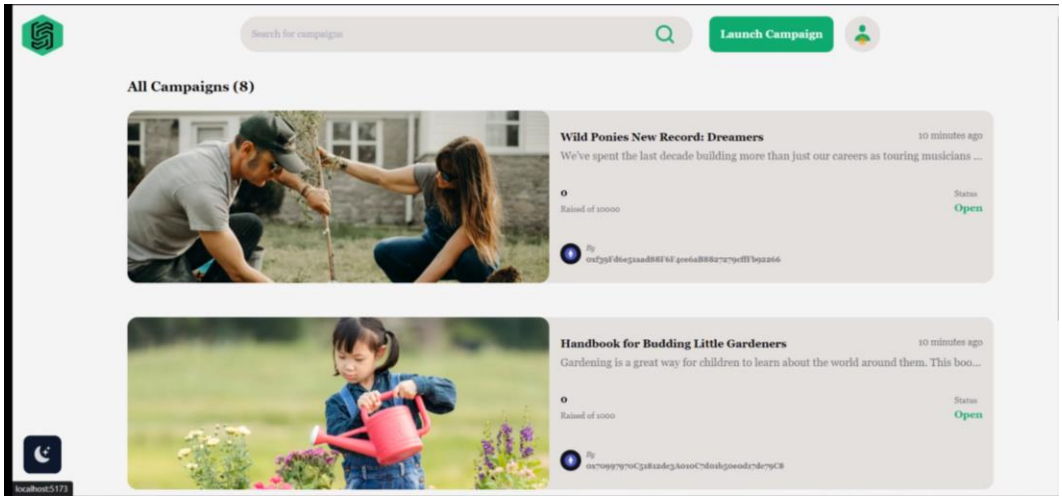

Figure 4-1. Main Page

As shown in figure 4-2 the website is responsive. Meaning the design and the way the website is shown respond to the user's behavior and environment based on screen size and platform. This website (Supply) is made with the mobile-first design to achieve many advantages as we discussed earlier.

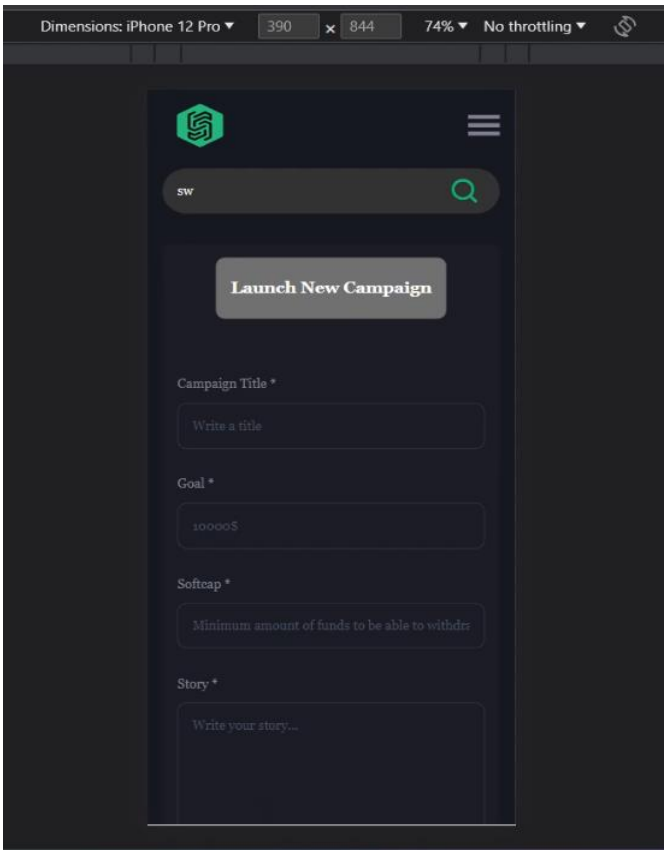

Figure 4-2. App's Responsive Design

4.1 Campaign Structure

Fetching and posting data to three sources to create a campaign is tough. Campaign data is divided into three parts; the main part is stored on the Ethereum blockchain on our CrowdCampaigns Smart Contract, the second part is stored on MongoDB, which is a NoSQL database, and the third part is IPFS file storage called Web3.Storage to store the campaign image. The reason behind that is storing data on blockchain costs money, hence, storing large

amounts of text like the description is neither affordable nor efficient. In the same way, we cannot store images on the blockchain.

The Campaign struct data structure contains multiple data fields as shown in Figure 4-3, here they are:

1. **owner**: the Ethereum address of the campaign launched.
2. **target**: the target amount of Dai the campaign wants to collect.
3. **raisedAmount**: raised amount of the campaign funds.
4. **isOpen**: a Boolean to check if campaign status is open or closed.
5. **image**: image URI.
6. **title**: campaign title.
7. **timeStamp**: the timestamp of the campaign at starting.
8. **rewardTokenId**: which reward token the campaign rewards its funders with.
9. **softcap**: which is the amount of raised amount that fundraisers after it cannot refund, and the owner can withdraw the campaign funds.
10. **withdrawnAmount**: the amount that got withdrawn from the total funds by the campaign owner.

```
struct Campaign {
    address owner;
    uint target;
    uint raisedAmount;
    uint withdrawnAmount;
    uint timeStamp;
    uint rewardTokenId;
    uint softcap;
    bool isOpen;
    string image;
    string title;
}
```

Figure 4-3. Campaign struct of CrowdCampaigns Smart Contract

This is all the data that the campaign has on the Smart Contract. There is also data we store on MongoDB. Here is the second part of the campaign structure along with its constraints on Mongoose Schema:

```
{
  campaignId: {
    type: Number,
    required: [true, 'campaignId is missing'],
    unique: [true, 'campaignId must be unique'],
    immutable: true
  },
  desc: {
    type: String,
    required: [true, 'description is missing']
  },
  category: {
    type: String,
    required: [true, 'category is missing']
  },
  message: {
    type: String
  },
  featured: {
    type: Boolean,
    default: false
  }
},
{ timestamps: true }
```

Figure 4-4. Rest of Campaign Data on Mongoose Schema

All these pieces of data are inputted throw the following form:

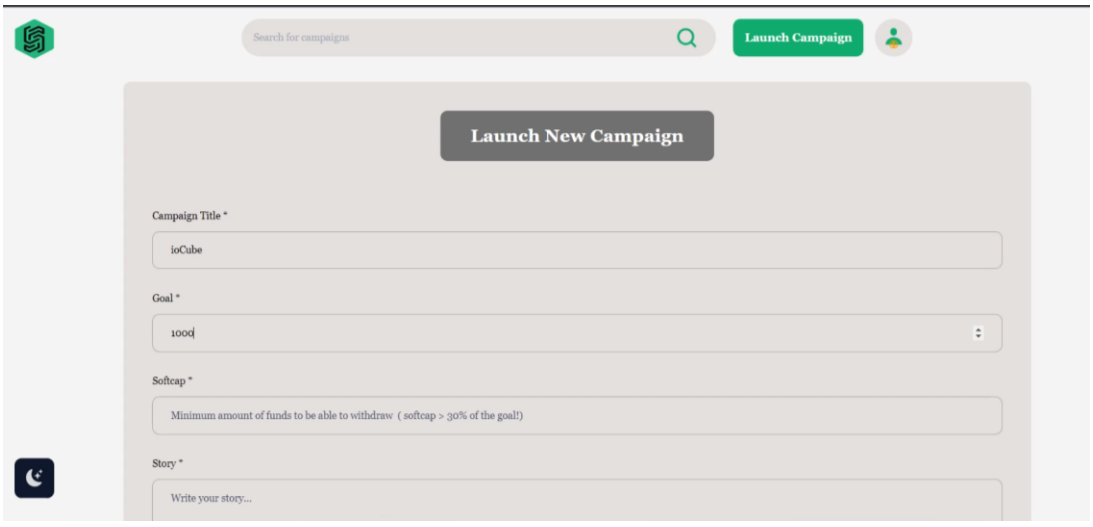

Figure 4-5. Part of Campaign Creation Form

There are some constraints to launching new campaigns. Firstly, every address can launch one campaign every week, considering that launching a campaign is not an everyday action for people to do. Also, the Ethereum blockchain ecosystem is full of bots because it is totally online and allows its users to use public/private key encryption to create their addresses by themselves. Thus, preventing spamming is an essential consideration when creating Ethereum Smart Contracts even if it adds some Gas fees. Secondly, the campaign soft cap amount must be above 30% of the campaign target. So, fundraisers ensure there is enough time to refund their funds. Plus, many other intuitive constraints as follows:

```
        require(_rewardTokenId > 0, 'Reward token id must be greater than 0');
        require(_target > 0, 'Target amount must be greater than 0');
        require(_softcap <= _target, 'Softcap must be less than the target');
        require(
            _softcap >= (_target * 3) / 10,
            'Softcap must be at least 30% of the target'
        );
        require(bytes(_title).length > 0, "Title can't be empty");
        require(bytes(_image).length > 0, "Image can't be empty");
```

Figure 4-6. Campaign Creation Constraints

Every campaign has a page specified for showing its details, alongside all the campaign funds and fundraisers. On this page, there is a fund card so people can fund the campaign through it, the campaign owner can close their campaign, the owner can withdraw their funds, and fundraisers can refund if refunding is available. So, this page displays all campaign data, all related funds, the way for users to fund the campaign, and all management actions that the campaign owner or fundraisers can take. Here it is:

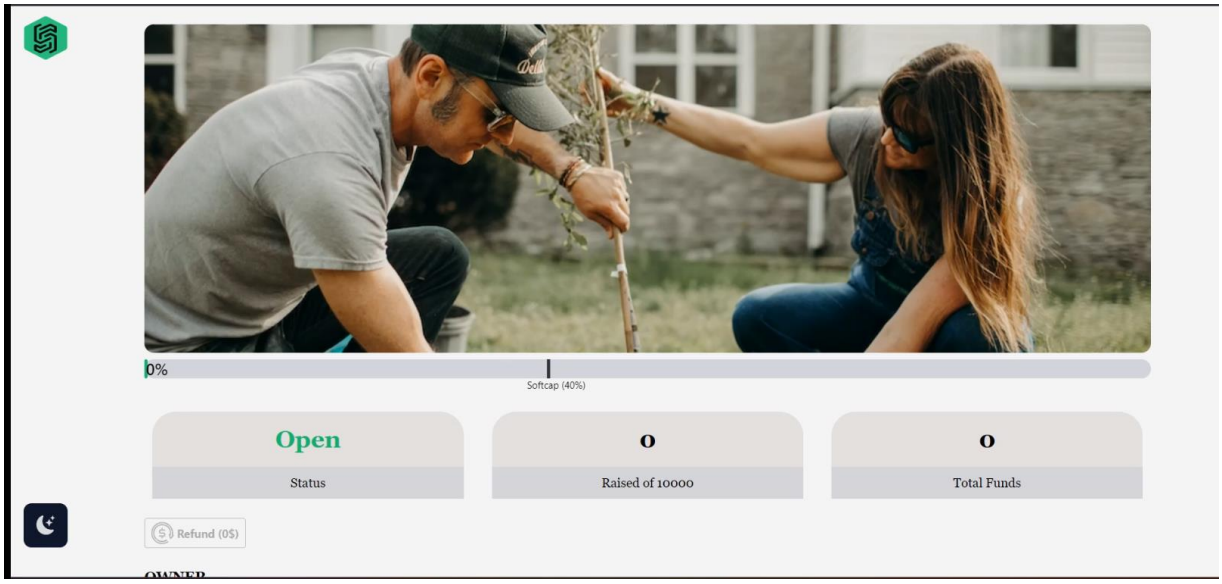

Figure 4-7. Campaign Details 1

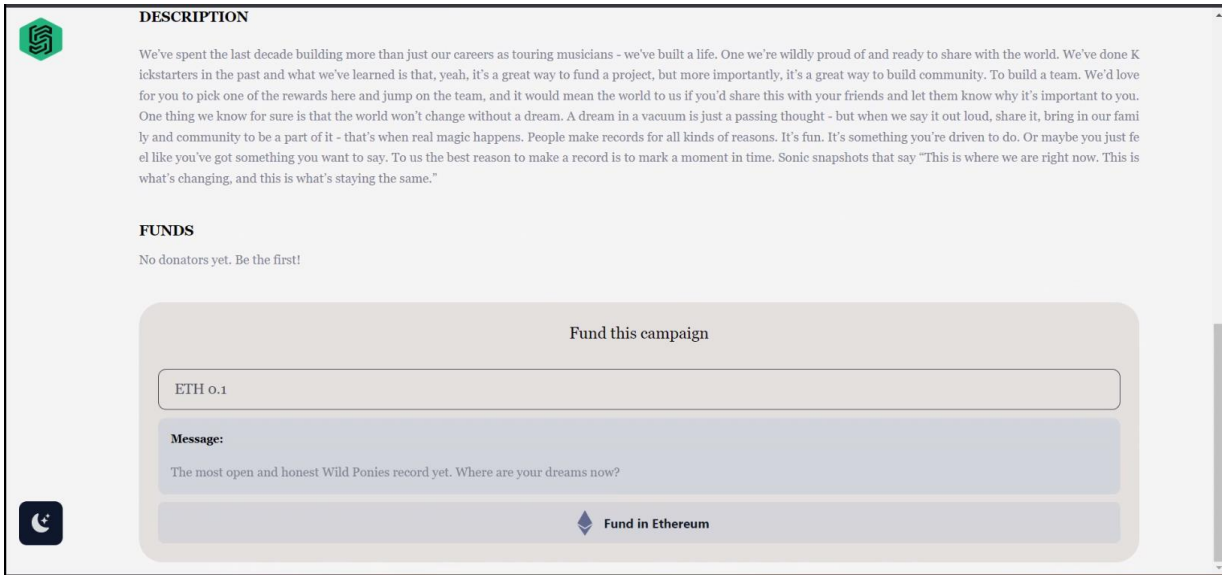

Figure 4-8. Campaign Details 2

4.2 Reward Token Structure

The Campaign reward ERC1155 token struct contains multiple 3 basic data fields as shown in Figure 4-9, they are:

```
struct Token {
    string name;
    string symbol;
    string uri;
}
```

Figure 4-9. Token structure

Whenever the user Ethereum address is shown on our website, this user's all rewards show up as follows:

Figure 4-10. Fundraiser Rewards

4.3 Web Page Performance

Chrome Lighthouse is a powerful tool provided by Google to assess and improve the performance, accessibility, best practices, and search engine optimization (SEO) of web applications and websites. Running a website through Lighthouse generates a comprehensive report, providing developers with valuable insights into their application's performance and user experience. The Lighthouse report is divided into several categories, each focusing on different aspects of web development. These categories include Performance, Accessibility, Best Practices, and SEO [59]. In this section, we will show and discuss Chrome Lighthouse metrics for Supply's main page.

In the Performance category, Chrome Lighthouse evaluates the loading speed and responsiveness of the website. It measures critical performance metrics, such as First Contentful Paint (FCP), Time to Interactive (TTI), and Total Blocking Time (TBT). A high score in this

category indicates that the website loads quickly and provides a smooth user experience, contributing to improved user engagement and retention. The Performance score is not the best we can do especially since using an SPA user interface library like React makes the website very heavy in return for way better user experience, notwithstanding, there are some ways to boost React performance. In addition, we can use simple techniques like lazy-loading. Enhancing the Performance score is one of our priorities in future work.

The Accessibility category assesses how well a web application can be used by individuals with disabilities. Lighthouse checks for elements like proper alt text for images, semantic HTML, keyboard accessibility, and sufficient color contrast. A strong score in this area indicates that the website is inclusive and can be accessed by a broader range of users, complying with accessibility standards and guidelines. We made some efforts at this.

The Best Practices category evaluates the overall code quality and adherence to industry best practices. It checks for aspects such as the use of modern web technologies, secure connections (HTTPS), avoiding deprecated APIs, and other coding standards. A higher score in this section ensures that the website follows current development practices, reducing the risk of compatibility issues and security vulnerabilities. We have better work to do this at this point.

The SEO category focuses on the website's search engine optimization. It assesses factors such as meta tags, heading structure, URL structure, and mobile-friendliness. A favorable score in this category means the website is well-optimized for search engines, leading to better discoverability and visibility in search results. We could have done better by adding more HTML tags to the HTML header, this would make an impact.

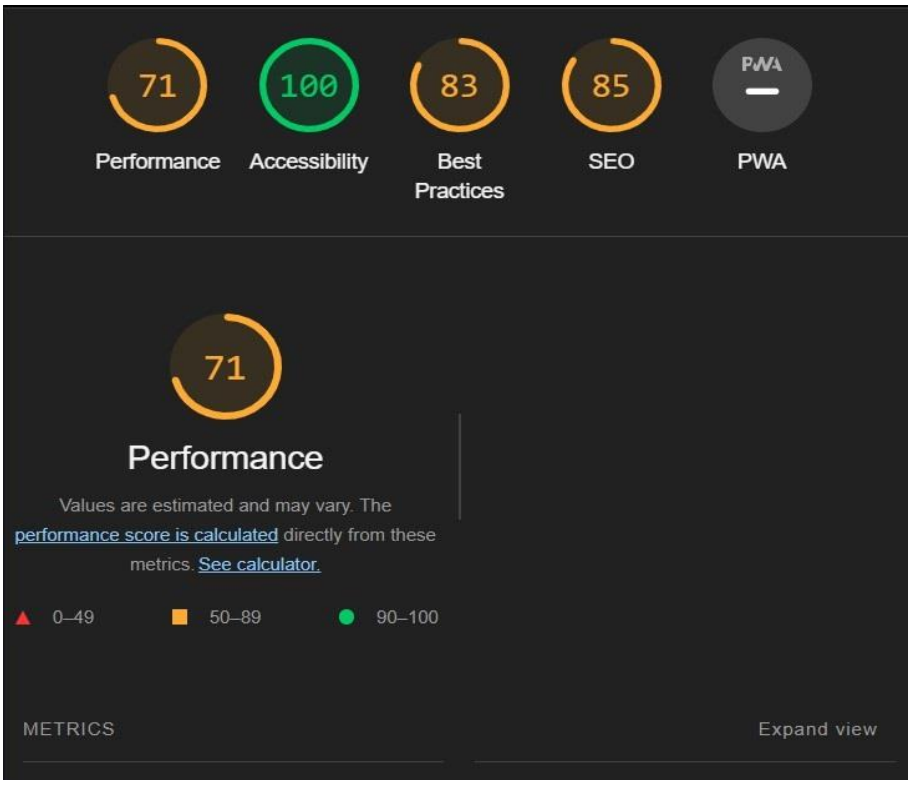

Figure 4-11. Supply's Main Page Chrome Lighthouse Metrics

4.4 API Endpoints Performance

Testing API endpoints' performance is not an easy task. Most API requests have their latency outside of the application. In my observation, if a single request has been done two times consecutively, one might take 300ms and the other 900ms. This happens for different reasons:

1. **Database Query Time**: If the API involves database queries, the time taken to fetch and process data from the database can significantly impact the overall response time. Especially, that database might be hosted on a cloud and uses its server to communicate with our server. In our case, we fetch data from the Ethereum protocol which is a peer-to-peer protocol that leads to more unexpected response time.
2. **Network Latency**: The time taken for data to travel from the client to the server and vice versa over the network. It can be influenced by the physical distance between the client and the server and the quality of the network connection.
3. **Caching**: by serving requested data from the cache instead of executing the request again, we might get the response 10 times faster. Caching is implemented everywhere. Hence,

part of caching functionality is not even controlled by developers. All that leads to highly unexpected response times.

4. **Concurrent Requests**: The number of requests being processed simultaneously by the server. A high number of concurrent requests can lead to increased response times due to resource contention.
5. **Server Load**: The overall load on the server, including CPU, memory, and disk usage. A heavily loaded server may experience slower response times.
6. **Hardware and Infrastructure**: The server's hardware specifications and the efficiency of the underlying infrastructure can affect response times [60].

That being said, there are benefits to API performance testing. We can simulate user traffic by API performance testing, so we can observe how our API endpoints behave under certain loads, through an interval of time. These runs can identify performance bottlenecks and issues [61]. By using Postman Collection Runners, we evaluated campaign and rewards API endpoints. We made two runs for rewards and campaigns GET API Endpoints. Each run has 100 virtual users that keep requesting these endpoints for 2 minutes. Notably, in the production server, there is a request limiter for each IP address to prevent spamming. Spamming might be an attack or any malicious action. We pause this request limiter in development mode.

4.4.1 Rewards API Endpoints Evaluation

Overall, the average response time for every request is 1,9 seconds which is good considering the rate of requests on this run is 31.75 requests per second. In our application type, that much of requests per second is not usual. We could say that the expected time for rewards API endpoints would be from 1000ms to 1500ms.

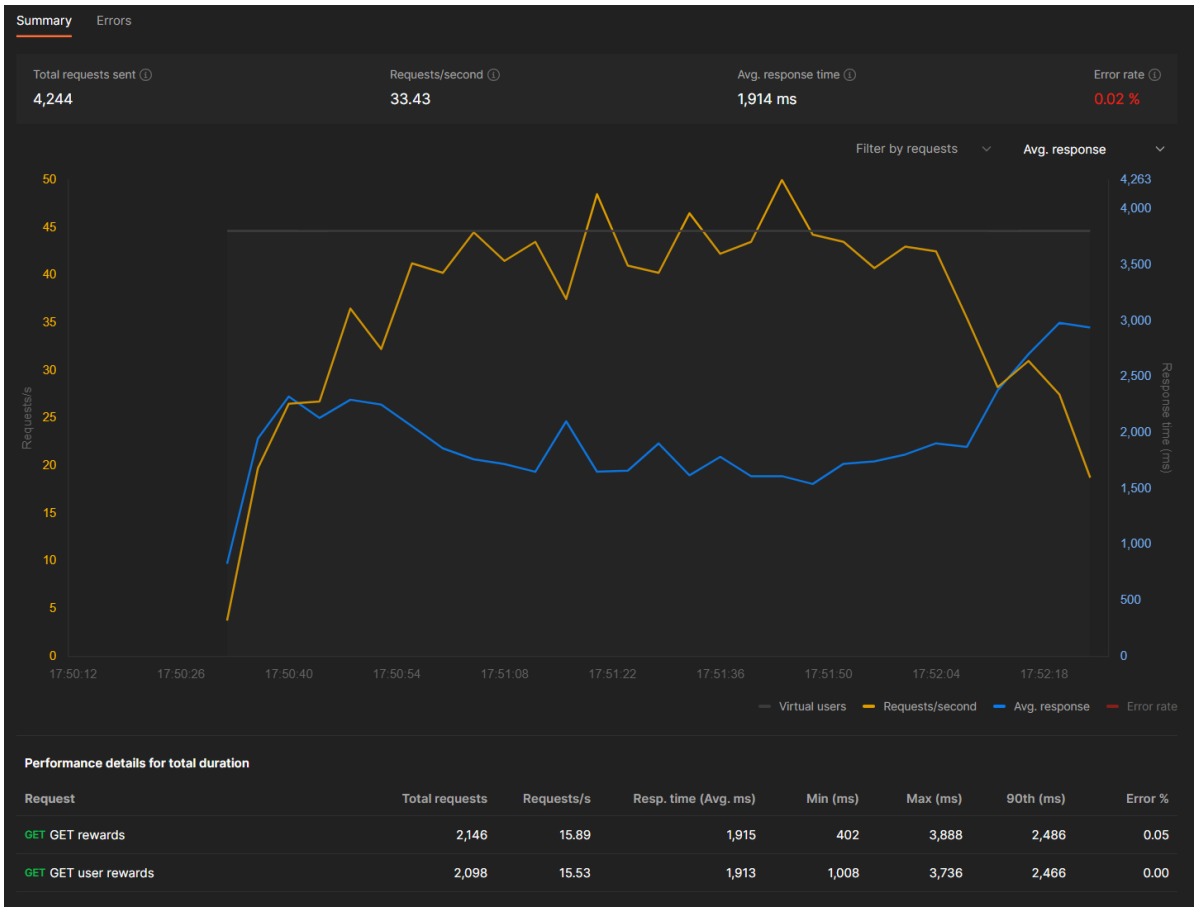

Figure 4-12. Rewards API Endpoints Evaluation by Postman Collection Runner

4.4.2 Campaigns API Endpoints Evaluation

The average response time for GET requests of campaign endpoints is 2.2 seconds. This is not a great performance. The rate of requests on this run is 31.75 requests per second, which is more than the usual rate. Apparently, the requests that filter Solidity Events take too much time Which is something interesting. This motivates researchers to find out how these Solidity Events get queried and how to make it more efficient. Especially, querying Events by their indexed values from sources outside Smart Contracts is the only usage of Events. Overall, the performance of campaign endpoints was expected to be better but, at the same time, it is tolerable. And it can be optimized.

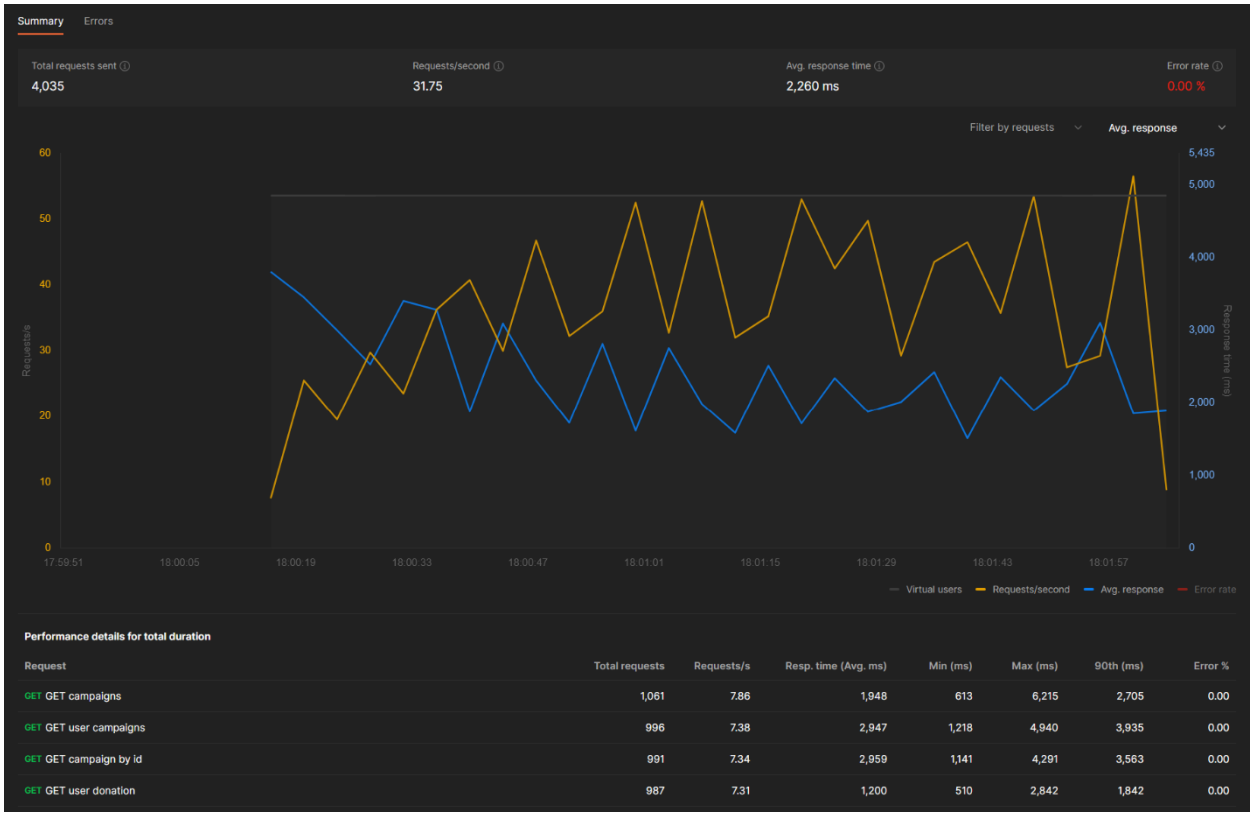

Figure 4-13. Campaigns API Endpoints Evaluation by Postman Collection Runner

4.5 Gas Fees Evaluation

Gas fee evaluation is the most critical measure for Smart Contracts' efficiency. In Smart Contract code, every function call that modifies the blockchain state requires to be mined/validated by miners/validators, so it requires its caller to pay the fees of this operation. If those fees are very high, people simply will not pay. Those Gas fees are determined by a lot of factors. As Smart Contracts developers, we can only control one of them, which is to lower the code execution space and time complexity. That being said, those many other factors that affect the Gas prices could make it very hard for people to pay any fees. In short, Smart Contract developers must write the most efficient code ever.

We have used many techniques to make the Gas costs low. There are such trivial techniques that surprisingly work, such as using Equation (3) instead of Equation (2). In addition to avoiding well-known computer science data structure and algorithms don’ts.

$$for(uint\ i = 0; i < campaignsIdsCount; i++) \quad (2)$$

$$for(uint\ i; i < campaignsIdsCount; ++i) \quad (3)$$

in Figure 4-15, we show a report of some function calls of our Smart Contracts and how much they averagely cost. We are going to calculate how much each operation costs in terms of US Dollars. Using the current Ethereum prices of July 6, 2023. At that moment, 1 Gas equals 50 Gwei and 1 Ether equals $1,915.47. We are going to use the Equation (4) to convert the cost number in the report to US Dollars. The cost number in the report is in terms of Gwei. Gwei is a unit of measurement of Ether as 1 Ether equals 10e9 Gwei. So, we take the price of 1 Gas which is 50 Gwei divide it by 10e9 to get the price of 1 Gas in Ether. Then multiply that Gas price by the Gas fees (the number from the table). Finally, we multiply that result by the price of Ether which is $1,915.47 to get the total Gas cost in terms of US Dollars.

$$Gas\_fees\_USD = ((50 / 10e9) * Gas\_fee) * 1915.47 \quad (4)$$

Starting with Contracts deployment costs, the campaigns contract costs $554,6 and the rewards contract costs $289.7. We have also evaluated some functions. addToken costs $19.2, mint costs $5.2, closeCampaign costs $2.96, createCampaign costs $71.7, fundInEth costs $17.4, refund costs $7.4, and finally withdrawFunds costs $8.3. It seems that some operations cost too much. We are looking for more improvements to balance this up. Just to keep note, these USD prices are not accurate because Ether and Gas prices are very volatile as they change every 10 seconds. Ethereum has scalability issues which are hopefully going to be resolved in its future updates (layer 1 solutions), by creating sidechains, or by layer 2 solutions.

Solc version: 0.8.1	Optimizer enabled: true	Runs: 200	Block limit: 30000000 gas

Methods						
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
CharityRewards	addToken	-	-	201456	2	-
CharityRewards	mint	-	-	54930	1	-
CrowdCharity	closeCampaign	-	-	30953	2	-
CrowdCharity	createCampaign	-	-	435480	11	-
CrowdCharity	fundInEth	134925	233330	181960	4	-
CrowdCharity	refund	-	-	77862	2	-
CrowdCharity	withdrawFunds	-	-	86786	2	-

Deployments			% of limit	
CharityRewards	-	-	3024919	10.1 %	-
CrowdCharity	-	-	5791126	19.3 %	-

Figure 4-15. Gas Fees Report of Our Contracts' Operations by Hardhat-gas-reporter Plugin

4.6 Discussion

Through this study, we had the ambition to make a crowdfunding platform that achieves a high layer of decentralization, hence, a high layer of transparency. We have discussed how blockchain started and some critical opinions about it. We believe the key point in blockchain debates is how efficient this idea is. Unarguably, the idea is hard to think of and to convince others to participate in it. Convincing people to have nodes in this decentralized network and to pay for computational power was the hardest part of its journey. Given how brilliant the idea is and how a technological advance it is. Is it efficient? Will it prevail? The exact answer is unforeseeable because of the many factors that determine it. For us, we have used blockchain primarily because we believe in it as a technological advance. Even with the idea's obscure future, we believe that researching it would help not only overcome its caveats but also innovate new ideas that might make new technological advances. To sum up our opinion about blockchain, we believe it is a phenomenon that is worthy of research and use.

Since we couldn't implement a full platform on the blockchain, we have used other technologies that are not as decentralized as blockchain. This came up with a huge problem other than degrading the platform transparency. Mixing some irreversible operations with reversible

ones opens the door for irreversible fatal errors from unexpected behaviors which could destroy the whole system. Creating part of the campaign on a regular database and the other part on a blockchain was a hard decision. Implementing this flawlessly is the only way to create this system. Thus, we did our best to keep this process safe and efficient.

CHAPTER FIVE
CONCLUSION & FUTURE WORK

5.1 Conclusion

As software systems trend nowadays are moving towards Web 3.0 and decentralized systems to solve some present problems that are caused by heavily depending on centralized systems. It is worth researching and building these decentralized solutions of Web 3. The existing crowdfunding platforms are handled by intermediary corporations that have a say and an action on various parameters of a campaign, the alternative solution based on blockchain peer-to-peer network handling campaign transactions seem more direct, transparent, and cheaper. Supply aims to explore and create ways to remove these intermediaries in a crowdfunding business. This is done with the help of smart contracts, written for crowdfunding applications deployed on the Ethereum blockchain. Without much effort, the campaign creator and its investor can perform their intended activities using this crowdfunding website. Other blockchain networks such as Cardano, Steller, and TRON provide different language choices and platform configuration choices compared to Ethereum, but these platforms haven't proven themselves yet compared to the Ethereum network. In the future, Supply can discover other blockchain networks if they prove themselves to be as competent as Ethereum or not.

5.2 Future Work

Supply is working and fully functional, however, there are ways to enhance its experience. Improving the API endpoints' performance, we have discussed issues we have dealt with in this topic in the Results section.

1. **Web server**: Usually, backend developers aim for 500ms for each request which we couldn't do for some endpoints. That requires more researching to solve.
2. **Frontend**: optimizing image loading and using React Query package instead of using useEffect React Hook for API fetching.

3. **Gas fees**: in our Contracts, we have experienced some very high transaction fees which can be mitigated by more code optimization or solved by using Ethereum layer 2 solution.
4. **Reward tokens**: NFTs and SFTs has many use cases in the Ethereum ecosystem but we left that open to the future.
5. **Digital Wallets**: We have made Metamask digital wallet integration with Supply but there are other popular digital wallets that other people use such as Brave, Opera, or Coinbase
6. **New blockchains**: There are other blockchains that use Smart Contract Technology just like Ethereum. In a future release, Supply could work with cryptocurrencies from other blockchain worlds such as Binance or Solana.

REFERENCES

[1] P. Rob, "Why Floyd Mayweather, Startups, and Practically Everyone Else Are Betting on Digital Currencies," Accessed: Feb. 27, 2020. [Online]. Available: https://www.inc.com/business-insider/cryptocurrency-mainstream-bitcoin-ethereum-ico-floyd-mayweather-times-square.html.

[2] "Deadcoins curated list of projects and ICOs," Accessed: Mar. 10, 2020. [Online]. Available: https://deadcoins.com.

[3] M. Harris, "How Zano Raised Millions On Kickstarter And Left Most Backers With Nothing," Accessed: Feb. 28, 2020. [Online]. Available: https://medium.com/kickstarter/how-zano-raised-millions-on-kickstarter-and-left-backers-with-nearly-nothing-85c0abe4a6cb.

[4] "popSLATE Support Section, popSLATE 2 - Smart Second Screen for iPhone," Accessed: Feb. 24, 2020. [Online]. Available: https://www.indiegogo.com/projects/popslate-2-smart-second-screen-for-iphone#/updates/all.

[5] Insider Intelligence, “Here are the credit card networks and payment networks you need to know,” Insider Intelligence, 15-Apr-2022. [Online]. Available: https://www.insiderintelligence.com/insights/credit-card-networks-payment-list/. [Accessed: 27-Jun-2023].

[6] Z. Zheng, S. Xie, H. N. Dai, X. Chen, and H. Wang, “Blockchain challenges and opportunities: a survey,” Int. J. Web Grid Serv., vol. 14, no. 4, p. 352, 2018.

[7] D. Appelbaum and S. S. Smith, "Blockchain Basics and Hands-On Guidance: Taking the Next Step toward Implementation and Adoption," The CPA Journal, vol. 88, pp. 28-37, 2018. [Online]. Available: https://www.cpajournal.com/category/magazine/june-2018-issue/

[8] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 11, pp. 2266-2277, Nov. 2019, doi: 10.1109/TSMC.2019.2895123.

[9] S. Azzopardi, J. Ellul, and G. J. Pace, "Monitoring smart contracts: Contractlarva and open challenges beyond," in Runtime Verification: 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings 18, pp. 113-137, Springer International Publishing.

[10] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," IEEE Access, vol. 4, pp. 2292-2303, 2016.

[11] Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System." Bitcoin.org, 2008. Available: https://bitcoin.org/bitcoin.pdf. Accessed: June 29, 2023.

[12] W. Dai, "b-money," 1998. [Online]. Available: http://www.weidai.com/bmoney.txt.

[13] H. Finney, "Reusable Proofs of Work (RPoW)," Nakamoto Institute, 2004. [Online]. Available: https://nakamotoinstitute.org/finney/rpow/index.html.

[14] A. Back, "Hashcash - a denial of service counter-measure," [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf, 2002.

[15] IvanOnTech, “What is A blockchain wallet - the complete guide,” Moralis Academy, 14-Jan-2021. [Online]. Available: https://moralismoney.com/blog/what-is-a-blockchain-wallet-the-complete-guide. [Accessed: 03-Jul-2023].

[16] Manning Publications, “Cryptographic hashes and Bitcoin,” Manning, 26-Jun-2017. [Online]. Available: https://freecontent.manning.com/cryptographic-hashes-and-bitcoin/. [Accessed: 03-Jul-2023].

[17] R.C. Merkle, "Protocols for public key cryptosystems," in Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pp. 122-133, Apr. 1980.

[18] Bitcoin Core, "Bitcoin P2P Network," Bitcoin Developer Documentation, [Online]. Available: https://developer.bitcoin.org/devguide/p2p_network.html.

[19] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” ACM Comput. Surv., vol. 23, no. 1, pp. 5–48, 1991.

[20] Tschorsch, F., & Scheuermann, B. (2016). Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. IEEE Communications Surveys & Tutorials, 18(3), 2084-2123.

[21] Murlidharan, S. (2020). A Comprehensive Survey of Blockchain: From Theory to Real-world Applications. IEEE Access, 8, 188709-188734.)

[22] Liu, Y., Xie, S., Zhang, H., Zhang, X., & Hou, Y. T. (2018). A Survey on Applications of Blockchain in Data Management. IEEE Transactions on Big Data, 6(3), 1-48.)

[23] V. Buterin, “Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform,” 2013.

[24] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” 2014.

[25] Zichichi, M., Contu, M., Ferretti, S., D’Angelo, G.: LikeStarter: a Smart-contract based Social DAO for Crowdfunding,” In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 313–318, Paris, France, (2019)

[26] H. Arslanian, “Ethereum,” in The Book of Crypto, Cham: Springer International Publishing, 2022, pp. 91–98.

[27] “Ethereum virtual machine (EVM),” ethereum.org. [Online]. Available: https://ethereum.org/en/developers/docs/evm/. [Accessed: 23-Feb-2023].

[28] Fabian Vogelsteller and Vitalik Buterin, "ERC-20: Token Standard," Ethereum Improvement Proposals, no. 20, November 2015. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20.

[29] W. Radomski, A. Cooke, P. Castonguay, J. Therien, E. Binet, and R. Sandford, "ERC-1155: Multi Token Standard," Ethereum Improvement Proposals, no. 1155, June 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1155.

[30] Trygve Reenska, https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html

[31] Glenn E. Krasner, Stephen T. Pope, “A cookbook for using the model-view controller user interface paradigm in Smalltalk-80”, Journal of Object-Oriented Programming, vol. 1, no. 3, 1988, pp. 26-49.

[32] Mikić, I., & Mitrović, M. (2017). MVC and MVP Architectural Patterns for Client-Side Web Applications. In 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) (pp. 1293-1298). IEEE. doi: 10.23919/MIPRO.2017.7973439.

[33] L. P. Chitra and R. Satapathy, "Performance comparison and evaluation of Node.js and traditional Web server (IIS)," 2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), Chennai, India, 2017, pp. 1-4, doi: 10.1109/ICAMMAET.2017.8186633.

[34] Agarwal, Rachit. 2014. "Why use ExpressJS over NodeJS for Server-Side Development?" Blog, Algoworks.

[35] Agoi, Abel. 2017. "A Simple Explanation of Express Middleware." Medium, December 06.

[36] C. Győrödi, R. Győrödi, G. Pecherle and A. Olah, "A comparative study: MongoDB vs. MySQL," 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 2015, pp. 1-6, doi: 10.1109/EMES.2015.7158433.

[37] “Mongoose,” Mongoosejs.com. [Online]. Available: https://mongoosejs.com/. [Accessed: 16-Feb-2023].

[38] A. Mesbah and A. van Deursen, “Migrating multi-page web applications to single-page AJAX interfaces,” in 11th European Conference on Software Maintenance and Reengineering (CSMR’07), 2007.

[39] S. Murugesan, “Understanding Web 2.0,” IT Prof., vol. 9, no. 4, pp. 34–41, 2007.

[40] E. Molin, ‘Comparison of Single-Page Application Frameworks: A method of how to compare Single-Page Application frameworks written in JavaScript’, Dissertation, 2016.

[41] S. Aggarwal, “Modern Web-Development using ReactJS,” International Journal of Recent Research Aspects ISSN, vol. 5, no. 1, pp. 133–137, Mar. 2018.

[42] Hughes, J. "Why Functional Programming Matters." The Computer Journal, vol. 32, no. 2, 1989, pp. 98-107. doi: 10.1093/comjnl/32.2.98.

[43] “Solidity — solidity 0.8.21 documentation,” Soliditylang.org. [Online]. Available: https://docs.soliditylang.org/en/latest/index.html. [Accessed: 29-Jun-2023].

[44] Facebook. (2021). Jest - JavaScript Testing Framework. [Online]. Available: https://jestjs.io/. [Accessed: June 29, 2023].

[45] Visionmedia. (2021). supertest - Super-agent driven library for testing Node.js HTTP servers using a fluent API. [Online]. Available: https://github.com/visionmedia/supertest. [Accessed: June 29, 2023].

[46] Roelofsen, M., et al. "Hardhat: A Flexible Ethereum Development Environment." GitHub, 2020. [Online]. Available: https://github.com/nomiclabs/hardhat. [Accessed: June 29, 2023].

[47] Chacon, S., & Straub, B. "Pro Git." Apress, 2014. Available: https://git-scm.com/book/en/v2. [Accessed: June 29, 2023].

[48] E. Verbina, “Introducing the Space Git flow,” The JetBrains Blog. [Online]. Available: https://blog.jetbrains.com/space/2023/04/18/space-git-flow/. [Accessed: 29-Jun-2023].

[49] GitHub. "GitHub Actions Documentation." [Online]. Available: https://docs.github.com/en/actions. [Accessed: June 29, 2023].

[50] A. Ha, “Eric Schmidt on Google’s ‘mobile first’ attitude, weaknesses,” VentureBeat, 13-Apr-2010. [Online]. Available:

https://venturebeat.com/mobile/eric-schmidt-mobile-first/. [Accessed: 30-Jun-2023].

[51] T. H. Tran, “Mobile-first design: An easy guide to everything you need to know,” Invisionapp.com. [Online]. Available: https://www.invisionapp.com/inside-design/mobile-first-design/. [Accessed: 30-Jun-2023].

[52] “Continuing to make the web more mobile friendly,” Google for Developers. [Online]. Available: https://developers.google.com/search/blog/2016/03/continuing-to-make-web-more-mobile. [Accessed: 30-Jun-2023].

[53] “Mobile network subscriptions worldwide 2028,” Statista. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 30-Jun-2023].

[54] M. Petre, "UML in practice," 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 2013, pp. 722-731, doi: 10.1109/ICSE.2013.6606618.

[55] “What is Class Diagram?” Visual-paradigm.com. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/. [Accessed: 20-Feb-2023].

[56] “Language influences — solidity 0.8.21 documentation,” Soliditylang.org. [Online]. Available: https://docs.soliditylang.org/en/latest/language-influences.html. [Accessed: 29-Jun-2023].

[57] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," Tech. Rep., Uniswap, 2021.

[58] “What is Sequence Diagram?” Visual-paradigm.com. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/. [Accessed: 20-Feb-2023].

[59] “Lighthouse,” Chrome Developers. [Online]. Available: https://developer.chrome.com/docs/lighthouse/. [Accessed: 30-Jun-2023].

[60] J. Smith and A. Johnson. (2018). "Web API Design: Creating Interfaces that Developers Love," 2nd ed., O'Reilly Media, Sebastopol, CA, USA.

[61] “Testing API performance,” Postman Learning Center. [Online]. Available: https://learning.postman.com/docs/collections/testing-api-performance/. [Accessed: 01-Jul-2023].