SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

DApp: Crowdfunding Reinvented

BLOCKCHAIN AND DISTRIBUTED LEDGER TECHNOLOGIES

Professor:

Claudio Di Ciccio

Students:

Rui Filipe Mendes Pinto, 2053995

Davide António Ferreira Castro, 2053976

Andrei Sugeac, 2052034

Academic Year 2022/2023

Contents

1 Preface

Crowdfunding has created countless opportunities for entrepreneurs to make their projects come to life with the community's help. It provides a forum for every individual to pitch their ideas in front of investors, but it is also an efficient way for charities to reach many people willing to help. That said, there are still many drawbacks experienced in the current crowdfunding environment that make people skeptical about how legitimate crowdfunding is.
DApp: Crowdfunding Reinvented is a decentralized crowdfunding platform aiming to improve the conventional crowdfunding model by integrating it with the blockchain.

The team behind the project is called the CR7s, and the team members and their responsibilities on the project are:

- Rui Filipe Mendes Pinto: Solidity, Javascript.
- Davide Antonio Ferreira Castro: Backend.
- Andrei Sugeac: Frontend.

In the following chapters, we will look at what the blockchain is and why we intend to use it in our project. After, we will cover the application's architecture, followed by the implementation and finishing with the limitations and improvements for the following versions.

2 Background

2.1 What is the blockchain?

First of all, let's start with an overview of the blockchain. Blockchain can best be described as a digital platform or database for securely storing information and recording transactions. It is distributed across a network of computers (i.e., a decentralized peer-to-peer network) and contains an increasing number of data records informally known as blocks. These blocks are protected by robust cryptography and, therefore, are almost entirely safeguarded against human error, tampering, removal, and revision. Each block contains transaction details and is time-stamped and associated with specific data that links it to the previous block in the chain. When a new transaction takes place, it is authenticated across the network by miners before being recorded as a new block and made available to view by other members of the blockchain.

There exist numerous types of blockchain networks:

- Public Blockchain;
- Private Blockchain;

Public and private blockchains are about accessibility and the ability to transact, meaning there may be a selection of the nodes that can view and transact in the blockchain in the case of private blockchains and no restrictions regarding public ones. Furthermore, a blockchain can be either permissionless or permissioned. This concerns who decides whether a new block should enter the blockchain (ability to participate in consensus). Permissionless blockchains allow their nodes to participate in these decisions, contrary to permissioned blockchains, where only a set of nodes has this decision power. Every blockchain type has advantages and disadvantages, so the type of blockchain to be used depends on many aspects of a project.

Smart contracts are other essential and handy features of the blockchain. A Smart Contract is a computer program that directly and automatically controls the transfer of digital assets between the parties under certain conditions. A smart contract works the same way as a traditional contract while also automatically enforcing the contract. Smart contracts are programs that execute exactly as their creators program them.

2.2 Why decentralization matters

When discussing the blockchain, a characteristic that is often mentioned is that it is decentralized. Decentralization in blockchain refers to transferring authority and decision-making from a centralized entity (person, organization, or group thereof) to a distributed network. Decentralized networks are designed to reduce the amount of trust that members must have in one another and to prevent members from abusing

power or authority in ways that undermine the network's functionality.
Let's list the main advantages that come from decentralization:

- Transparency;
- Provides a peer-to-peer environment;
- Points of vulnerability are reduced;
- Distributes resources more efficiently;
- Immutable;
- Secure.

2.3 Why Crowdfunding?

Our DApp focuses on improving the crowdfunding model. But the choice of crowdfunding was made since it has so much potential to promote and develop new ideas by people. Also, it has the unique characteristic of being remotely online, which facilitates the exposure of a campaign to people from all around the world. While massive exposure to the outside world can result in receiving many donations, it can also result in receiving feedback regarding a project, improving communication between consumers and businesses.

3 Context

3.1 Downside of Crowdfunding

Before going over the aim of our DApp, let's first discuss the big problem in the conventional crowdfunding model that we are trying to solve:

- **Crowdfunding platforms face high risks of fraud** since they don't allow the users to track how their contributions are being used;
- **Money is sent directly to the creators of a campaign**, which oftentimes has resulted in people running away with the money.

Many wide-known crowdfunding platforms have struggled with the risk of fraud, such as Kickstarter, GoFundMe, etc. Apart from fraud, there are some other disadvantages, such as the fact that intellectual property isn't protected on crowdfunding platforms. Basically, any big company can notice the idea of a small startup and, by using their resources, implement it faster and even in better quality.

3.2 Blockchain as a solution

The many benefits of blockchain technology can be applied to the crowdfunding environment in order to solve its underlying issues. The use of smart contracts will guarantee safety and transparency for the users in terms of how their contributions are being used. The contributions in Ethereum are stored in the smart contract and cannot be accessed until specific withdrawal requests are issued by the campaign's creator and approved by the majority of the campaign's contributors, giving them back the control.
For this project, the type of blockchain used will be a Public Permissionless Blockchain, decided by taking into consideration the following aspects:

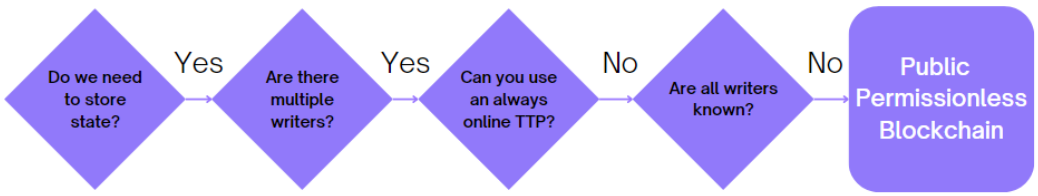


Figure 1: Blockchain type to use in production

4 Architecture

Decentralized applications have a very different architecture from traditional applications, aiming for high security, reliability, and privacy. Contrary to these conventional applications, creating a centralized server that stores the entire application data does not need to be done. Instead, all application data is stored directly in the blockchain. Typically, the following components hold in our DApp architecture:

1. Ethereum blockchain.
2. Smart Contracts.
3. Ethereum Virtual Machine (EVM).
4. Frontend.
5. Backend (to handle part of the communication between the client and the blockchain).
6. Database (used to store data that is too expensive to be stored in the Smart Contract's storage).

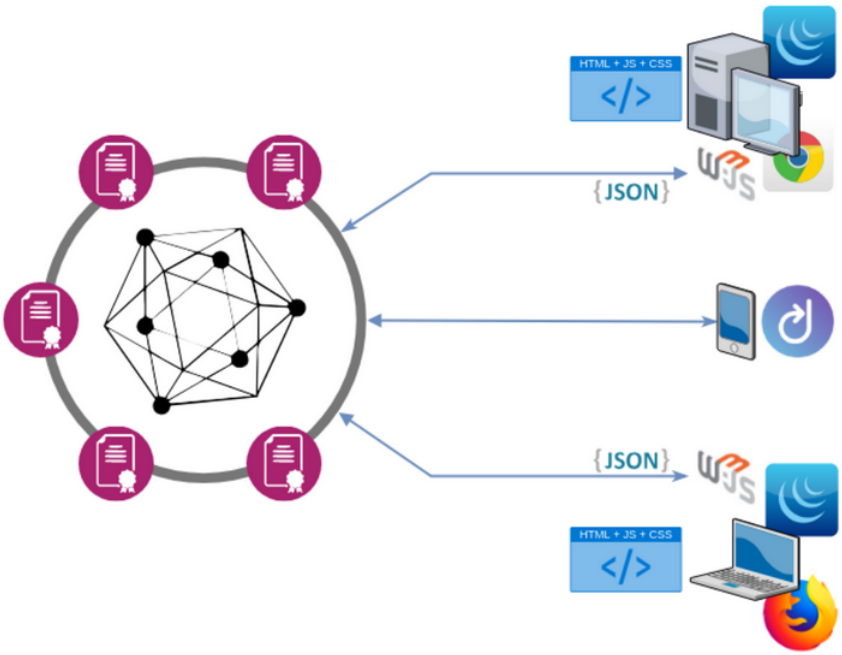


Figure 2: DApp: Software Architecture [1]

4.1 Ethereum Blockchain

Ethereum is a decentralized open-source blockchain platform that provides security, transparency, and traceability to transactions executed inside this environment. Furthermore, it establishes a peer-to-peer network that securely runs and verifies application code, known as Smart Contracts. Typically, the Ethereum blockchain is used for data storage in DApps.

4.2 Smart Contracts

Smart Contracts consist of code with the logic associated with state changes happening in the blockchain. In the case of our DApp, the collection of code was written using Solidity [2].

4.3 Ethereum Virtual Machine (EVM)

The EVM is the collection of all the nodes that are part of the Ethereum blockchain and form a global virtual computer that runs the logic defined in the Smart Contracts. Notice that many of these lines of code result in state changes. All of this is stored again on each node that is part of the EVM.

4.4 Frontend

The frontend development technology was ReactJS, a Javascript library for building web applications. Moreover, we used Material UI (MUI), an open-source React component library that implements Google's Material Design and provides gorgeous components that were very useful during the development [3].

4.5 Backend

The technology used in the backend of our DApp was NodeJS, a JavaScript runtime environment. On it, an API was developed, providing critical endpoints used by the client application. It is essential to mention that this backend by no means increases the centralization of our DApp. It is necessary to handle data that is simply too expensive to store in the blockchain, i.e., Strings.

4.6 Database

The selected technology for the database was MongoDB. As mentioned above, the database stores information that is too expensive to be held in the blockchain nodes' storage.

4.7 Middleware

In order to interact with the Ethereum blockchain through remote Ethereum nodes both in the frontend and backend, the `Web3.js` library was used [4]. Along with this library, we opted to choose two different `HTTP` providers: MetaMask [5] and Alchemy [6]. MetaMask regarding the frontend activity because it enables us to connect the user's account to the DApp and perform actions on our website. Alchemy is on the backend for reasons that will be explained later on. `Web3.js` provides a way to call Smart Contract methods and send transactions asynchronously. In the case of *PURE* or *VIEW* functions that don't modify the internal state of the Smart Contract, a return value is obtained using the `CALL` method from `Web3.js`. In cases where the method isn't either *PURE* or *VIEW*, meaning it is *PUBLIC* or *EXTERNAL*, indeed modifies the internal storage of the contract, we use the `SEND` method from `Web3.js` and then, if needed, listen for *EVENTS* triggered by the Smart Contract's code execution and later on act according to them.

4.8 Why use a Backend?

The main reason we opted to use a backend was to lift some heavy-weight work from the client to a backend, essentially delegating tasks to the backend. A simple example where this applies is when visiting the main page of the DApp and loading the list of all the campaigns. To fetch the campaigns' details, we first need to fetch their contract addresses and then fetch the values of their internal state. To do this, we need to connect to an Ethereum provider that implements a JSON-RPC specification. By delegating part of this task to a backend, we turn the frontend into more lightweight. Besides this, the backend needs to provide an endpoint so that the String details of the campaign can be delivered to the frontend. This last part concerns the communication between the backend and MongoDB. So why not do all this work in the backend? The next question is: why use two different providers for accessing the blockchain? Well, we use MetaMask for the frontend for the reasons presented above. For the backend, as MetaMask is simply a browser plugin that serves as an Ethereum Wallet, we opted for using Alchemy which provides a free-tier subscription and enables us to communicate with the Ethereum blockchain by providing a specific API related to it.

The last reason we need a backend is to store the images awarded as NFTs for the first contributors that donate to a Campaign. As it will be later explained, these images are stored on InterPlanetary File System (IPFS) and follow a specific metadata structure. The problem is that IPFS takes much time to load these images due to traffic restrictions on IPFS gateways. For this reason, having a certain degree of redundancy in the backend is handy, allowing us to provide low-latency access to these images.

4.9 UML Diagrams

This section presents several UML diagrams: concept, an analogous UML component model diagram, collaboration, and use case.

4.9.1 Concept Diagram

A UML Concept diagram is used to describe the structure and content of the Smart Contracts.

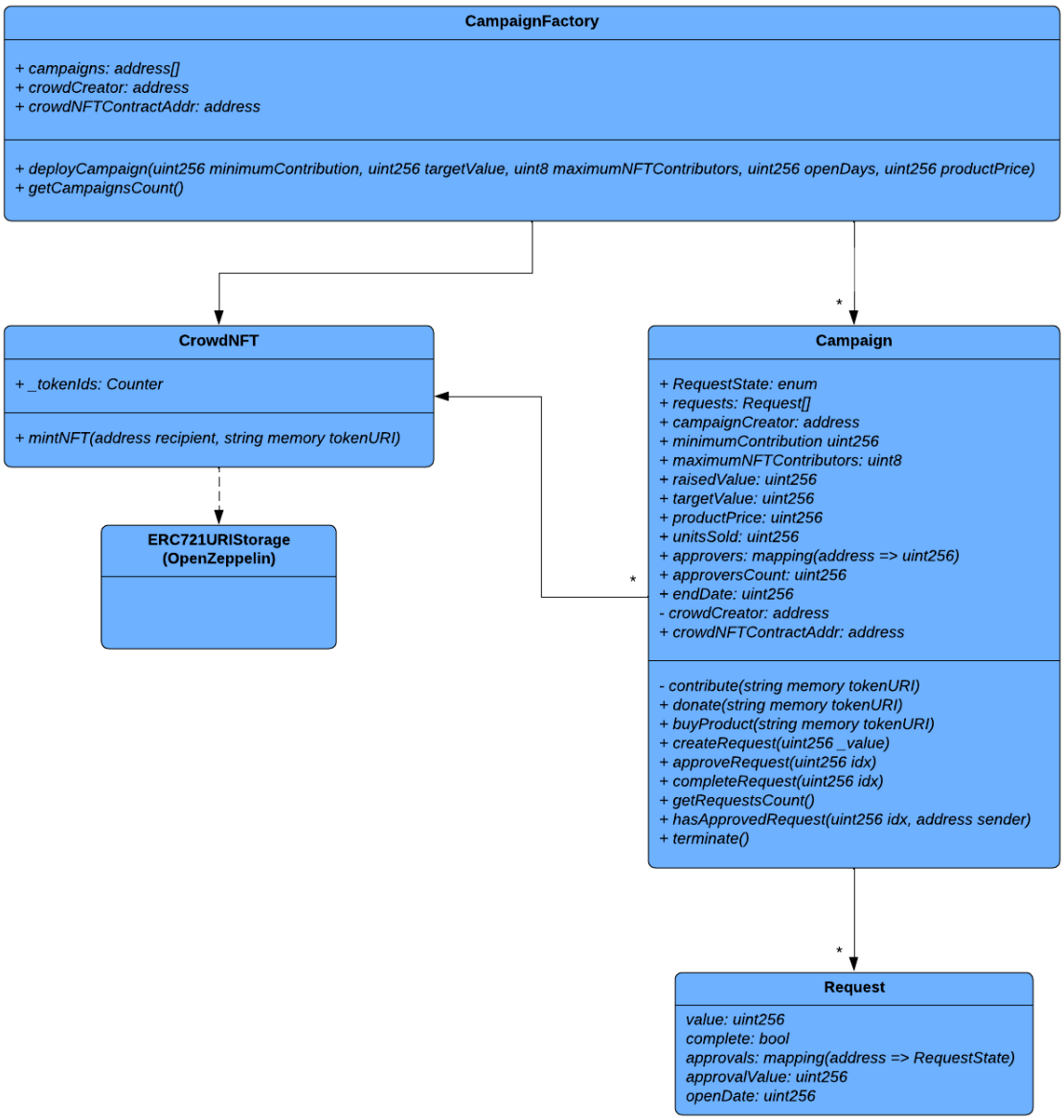


Figure 3: UML Concept diagram

As observed, we have three Smart Contracts: `CampaignFactory`, `Campaign`, and `CrowdNFT`. The `CampaignFactory` is the main contract and uses a slightly modified Factory Design Pattern. It is responsible for dealing with the creation of new campaigns.

Each campaign is associated with an independent contract, the `Campaign` contract. Besides this, the `CampaignFactory` contract also creates an instance of the `CrowdNFT` contract that is responsible for the minting of NFTs and implements `ERC721` pattern [7] by extending the `ERC721URIStorage` contract, as part of the OpenZeppelin library [8]. This library helps to minimize risk by using battle-tested libraries of Smart Contracts. It includes the most used implementations of ERC standards. Whenever a campaign receives a new contributor, the first N contributors may receive an NFT, and that is precisely where the `CrowdNFT` contract comes in. Going back to the `Campaign` contract, it holds the following attributes:

1. **requests**: a list of fund withdrawal requests that is voted among the campaign contributors according to the donated amount to the campaign. Each request holds the following attributes:
    - **value**: the requested amount.
    - **complete**: if the request was closed or not.
    - **approvals**: a mapping that links the contributor address to a *RequestState* enum that states if the person has already approved the request.
    - **openDate**: the date when the request was opened.
2. **campaignCreator**: the campaign creator's address.
3. **minimumContribution**: the minimum value a person can donate to a campaign.
4. **maximumNFTContributors**: the number of available NFTs for the first contributors. This value is decreased as new NFTs are issued.
5. **raisedValue**: the total raised value of the campaign. Incremented each time there's a new contribution.
6. **targetValue**: the target amount the campaign should raise.
7. **productPrice**: the campaign's product price, as a campaign must have a product associated.
8. **unitsSold**: the number of products sold.
9. **approvers**: a mapping between the address of the contributor and the amount in Wei donated to the campaign.
10. **approversCount**: number of different contributors that helped the campaign.
11. **endDate**: the timestamp that marks the end of a campaign.
12. **crowdCreator**: the Crowdfunding platform creator's address.

13. **crowdNFTContractAddr**: the address of the `CrowdNFT` contract used when minting NFTs.

As for the `Campaign` contract methods, we have:

1. **contribute** - Private method that performs the steps to contribute to a campaign and is called by both **donate** and **buyProduct** methods. Apart from transferring a particular value to the Smart Contract, it also awards an NFT to a contributor if the necessary conditions are met by calling the **mintNFT** method of the `CrowdNFT` contract.

2. **donate** - External method only invocable in an open campaign by a person that donates money to the campaign.

3. **buyProduct** - Public method only invocable in an open campaign by a person that buys a campaign's product.

4. **createRequest** - External method only invocable by the campaign's creator that creates a withdrawal request ready to be voted by the campaign's contributors.

5. **approveRequest** - External method only invocable in open campaigns and by campaign contributors that haven't approved the request before. It allows a contributor to approve a specific request.

6. **completeRequest** - External method only invocable by the campaign's creator that finalizes the request if the amount gathered by the contributors that approved the campaign is greater than half of the total raised value by the campaign.

7. **getRequestsCount** - External method that gets the number of requests created.

8. **hasApprovedRequest** - External method that tells if a certain person has already approved a particular request.

9. **terminate** - External method only invocable by the Crowdfunding platform's owner that destroys a campaign. Only used in extreme cases.

The rest of the contracts have a smaller state, meaning the diagram is self-explanatory in these cases.

4.9.2 Analogous Component Diagram

This diagram intends to indicate the backend and frontend modules, as well as the respective dependencies of the project.

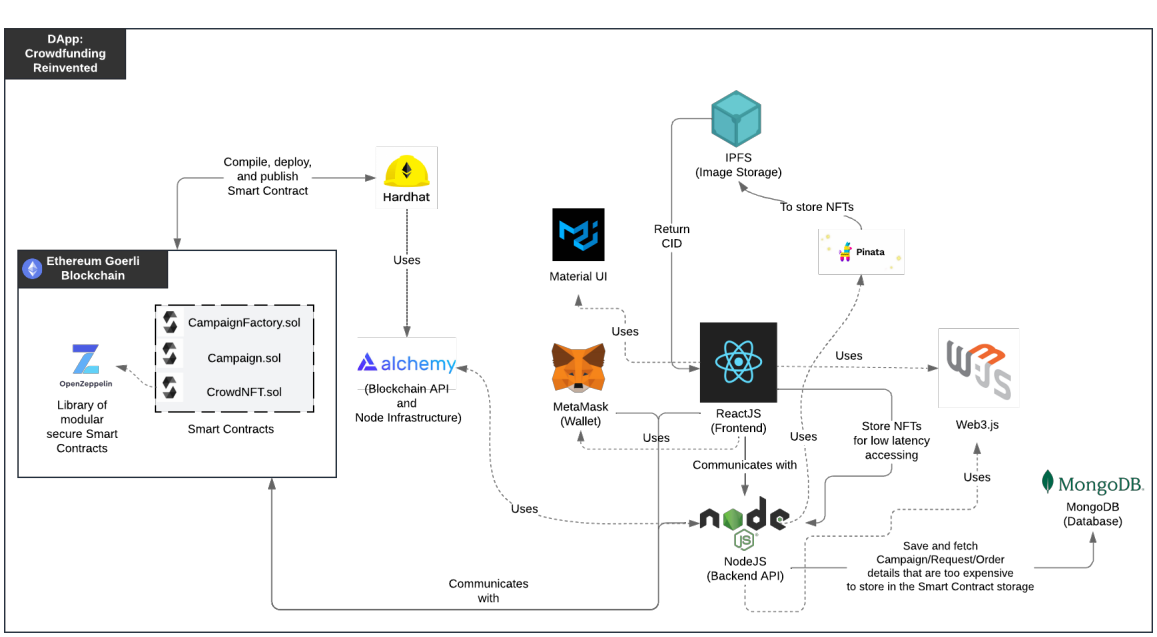


Figure 4: Analogous UML Component diagram

As mentioned, we developed our frontend using ReactJS and MUI. React uses `Web3.js` to communicate with the Smart Contracts on the Ethereum Blockchain. The frontend issues an API HTTP request to the backend that uses Pinata to upload the NFT images to IPFS [9]. MetaMask is the user's wallet that connects to the DApp frontend and allows the user to issue transactions to contribute to a campaign, approve and finalize requests, and create a campaign or request by handling transactions and requesting authorization whenever needed. The backend uses Alchemy to access the Ethereum blockchain. It is also responsible for storing and fetching details related to a campaign or request that are too expensive to be stored in the blockchain in a MongoDB database.

Besides all this, the diagram shows the flow of communication with the Ethereum Goerli Network. This test network enables us to test our Smart Contracts with Ether provided by Goerli Faucet from Alchemy [10]. The use of this test network is fascinating when minting NFTs. As the metadata is stored off-chain on IPFS and follows the standards of OpenSea [11], the world's first and largest Web3 marketplace for NFTs and crypto collectibles, it is possible to use this marketplace to view the minted NFTs, opening new opportunities for users that want to take further actions over their NFTs, such as: selling and transferring them.

Moreover, we introduce a new tool called Hardhat [12]. It is used to compile, deploy, publish and test the Smart Contracts. With a single command, we used it for the first three features, turning our Smart Contracts available in the Ethereum blockchain. Notice that publishing the Smart Contract turned out to be beneficial for debugging them, as the code gets available for all the nodes on the blockchain.

Lastly, we list all the available API endpoints:

Endpoint	Description
POST /campaigns/:campaignAddress	Stores the campaign details in MongoDB.
GET /campaigns/:campaignAddress	Fetches the campaign details.
PUT /campaigns/:campaignAddress	Updated the campaign document with the delivery details.
GET /campaigns/	Fetches campaign details from blockchain and MongoDB.
GET /campaigns/contribute/:personalAddress	Fetches campaign details from the campaigns the user has contributed both blockchain and from MongoDB.
GET /campaigns/personal/:personalAddress	Fetches campaign details from the campaigns the user has created both blockchain and MongoDB.
POST /images/:campaignAddress	Stores the image associated with the campaign and the NFTs it might award to the first contributors if any.
GET /images/nft/:campaignAddress	Used for storing the NFT image to IPFS using Pinata returning the token URI associated with the NFT.
POST /images/nft/:campaignAddress/:imgIndex	Move the NFT image from the campaign folder to the NFT. Invoqued after minting NFT.
POST /requests/:campaignAddress	Store request details into MongoDB, namely its description.
GET /requests/:campaignAddress	Get a request's details both from blockchain and MongoDB.
GET /nfts/:personalAddress	Get the NFTs a user currently owns.

As mentioned, they are used by the frontend and intend to move some of the heavy work from the client to the backend.

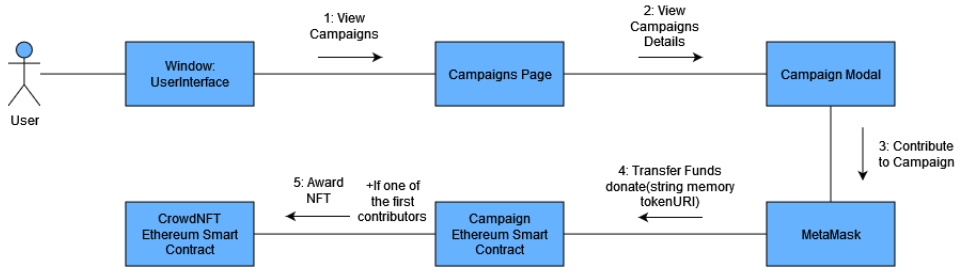
4.9.3 Collaboration Diagram

UML collaboration diagrams are used to show how objects interact to perform the behavior of a particular use case. In our case, we use it to illustrate the interactions among the DApp's components.

The image below shows the steps a user has to perform: contribute to a campaign, complete a withdrawal request, create a campaign, and create a withdrawal request. Starting from the frontend page the user is accessing, going through the interaction with MetaMask until a transaction is issued and a method of the Smart Contract is invoked, and finally, information is stored on the database, this diagram tries to depict the flow of the most important interactions of our DApp.

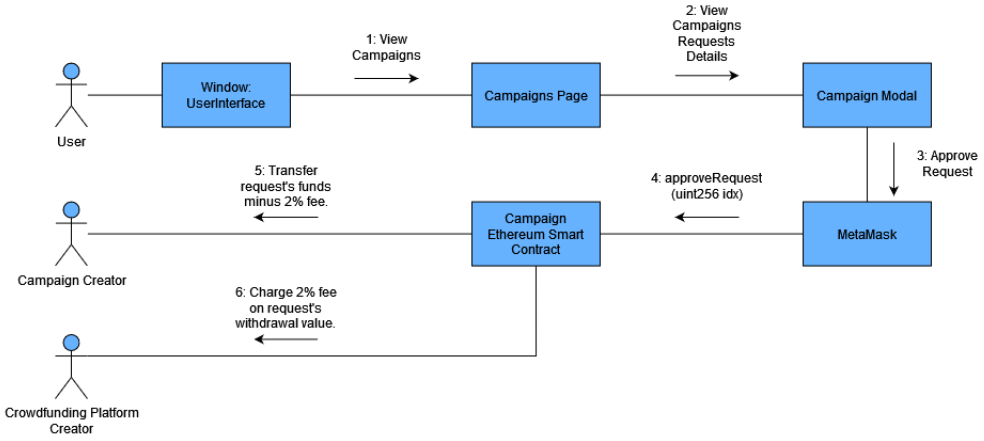
One question that stood out and was hardly enforced during our learning over the Blockchain and Distributed Ledger Technologies course was: "if we are blockchain developers and we work to ease people's lives, how can we also get paid at the end of the day"? This led us to introduce a small 2% fee for each completed request. The rest of the value goes to the campaign creator.

Contribute to Campaign:



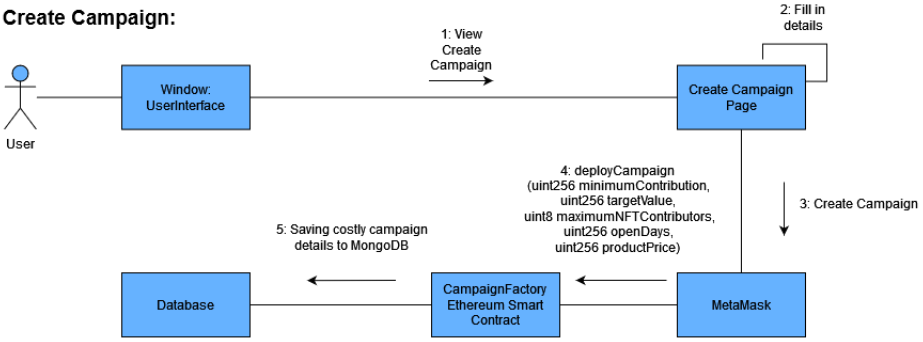
*The procedures to buy a campaign product are the same except that the invoqued Smart Contract function is buyProduct(string memory tokenURI).

Complete Withdrawal Request:



*The procedures to complete a request are the same except that the invoqued Smart Contract function is approveRequest(uint256 idx) and the funds transfer is unexisting.

Create Campaign:



Create Withdrawal Request:

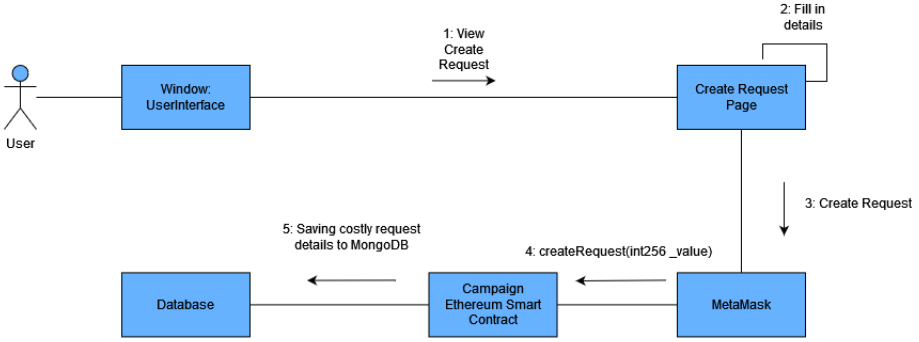


Figure 5: UML Collaboration diagram

4.9.4 Use Case Diagram

UML Use Case diagrams model a system's behavior and help capture the system's requirements. In this case, to goal is to illustrate the main usage scenarios of the DApp.

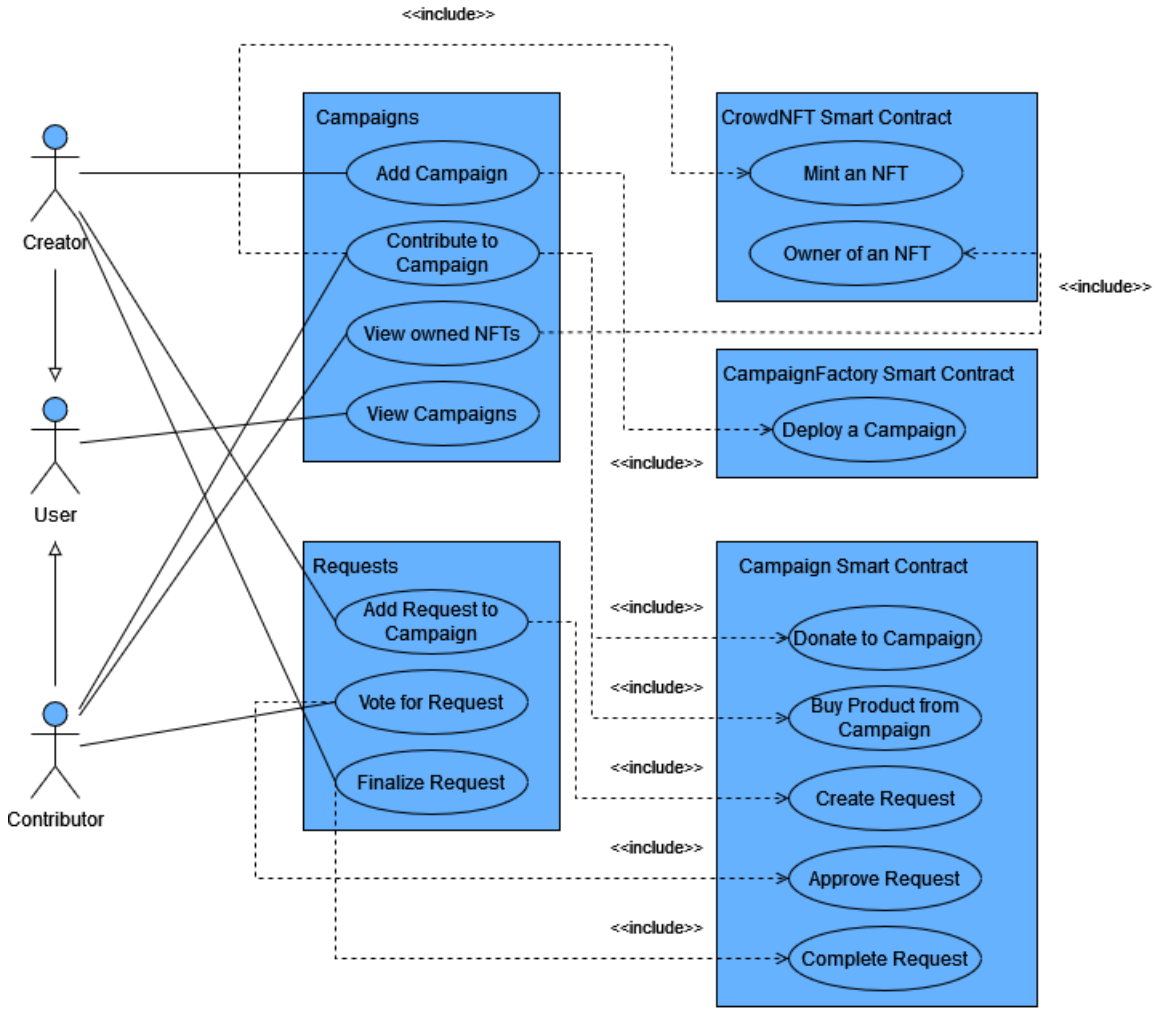


Figure 6: UML Use Case diagram

The user can be seen as actively having two roles: the creator and the contributor. The creator can create a campaign, add requests, and finalize them whenever allowed. This, of course, uses Smart Contracts, as shown in the diagram. The contributor can approve requests and contribute to a campaign by donating money or buying a related product. A regular user can at any time browse through the campaigns.

5 Implementation

For consulting information and interacting with the platform, a web interface was developed that gives the users the option to perform actions on four different main pages: the **All Campaigns**, **Create Campaign**, **Create Request**, and **My Profile** pages. The user can navigate between these using the side menu, illustrated in Figure 7.

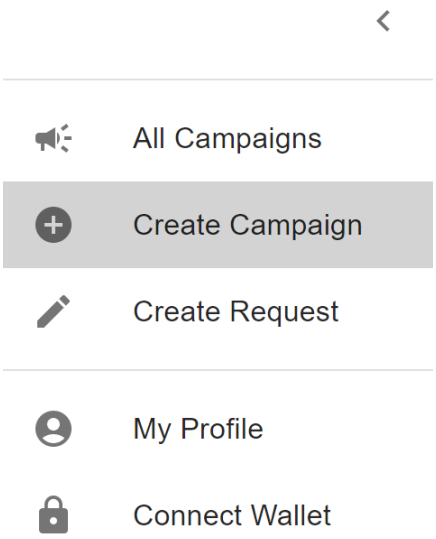


Figure 7: DApp: Side menu

To perform actions in the platform, such as contributing, buying products, creating campaigns, and creating requests, the user must connect their crypto wallet by using the **Connect Wallet** option in the side menu.

5.1 All Campaigns page

The web platform's home page is the All Campaigns page, which lists all campaigns created, displaying details such as the title, description, total funds raised, progress towards the goal, and time left to contribute.

Suppose the user wishes to see more details about a specific campaign. In that case, it is possible to select each campaign and an overlay is displayed containing three different tabs that have information about the campaign: the details, contribution and requests tabs. It is also possible to search a campaign based on its title and description.

5.1.1 Campaign Details

From this tab, one can consult the following information about the campaign:

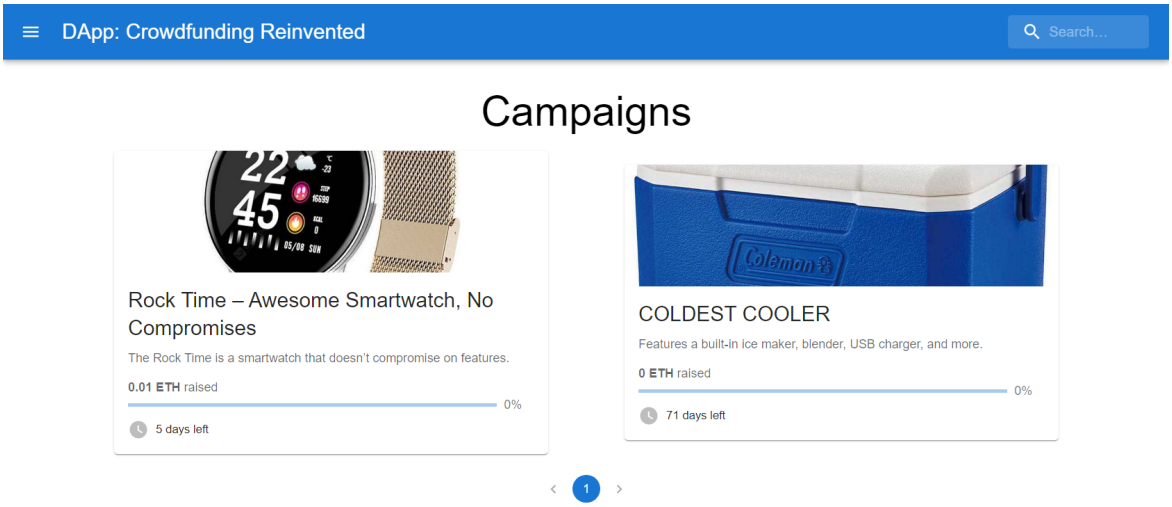


Figure 8: DApp: Home page - All Campaigns

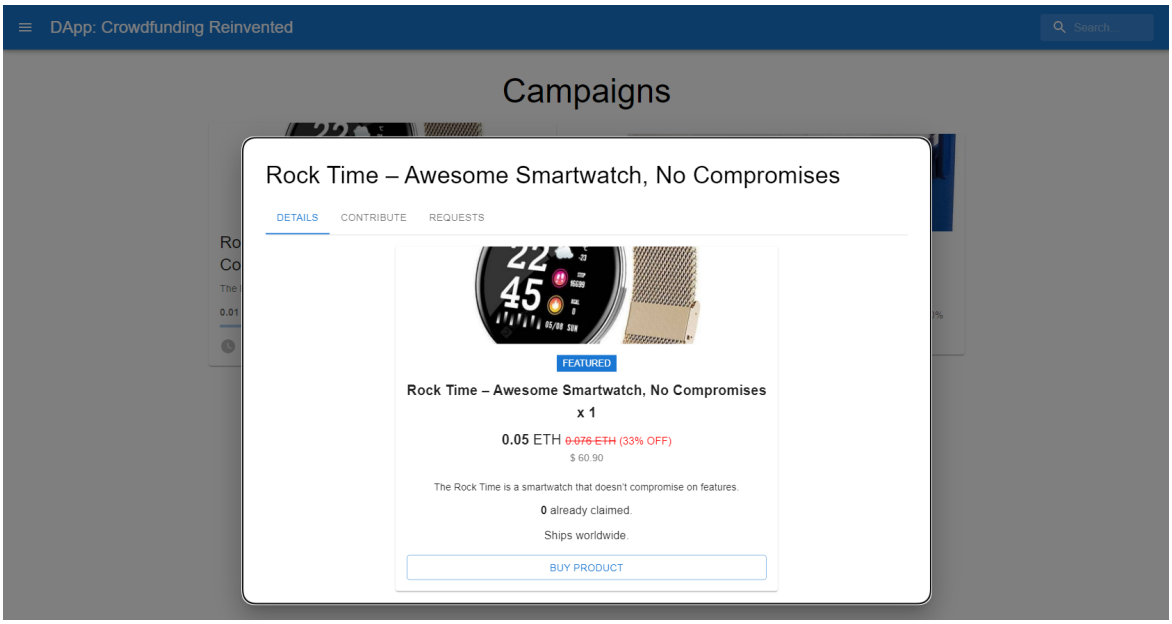


Figure 9: DApp: Campaign - Details tab

- Title of the campaign
- Full description of the campaign
- Price of the product provided by the campaign
- Equivalent value of the product's price in US Dollars
- Number of purchases already claimed

Additionally, the user can buy the product by pressing the **Buy Product** button.

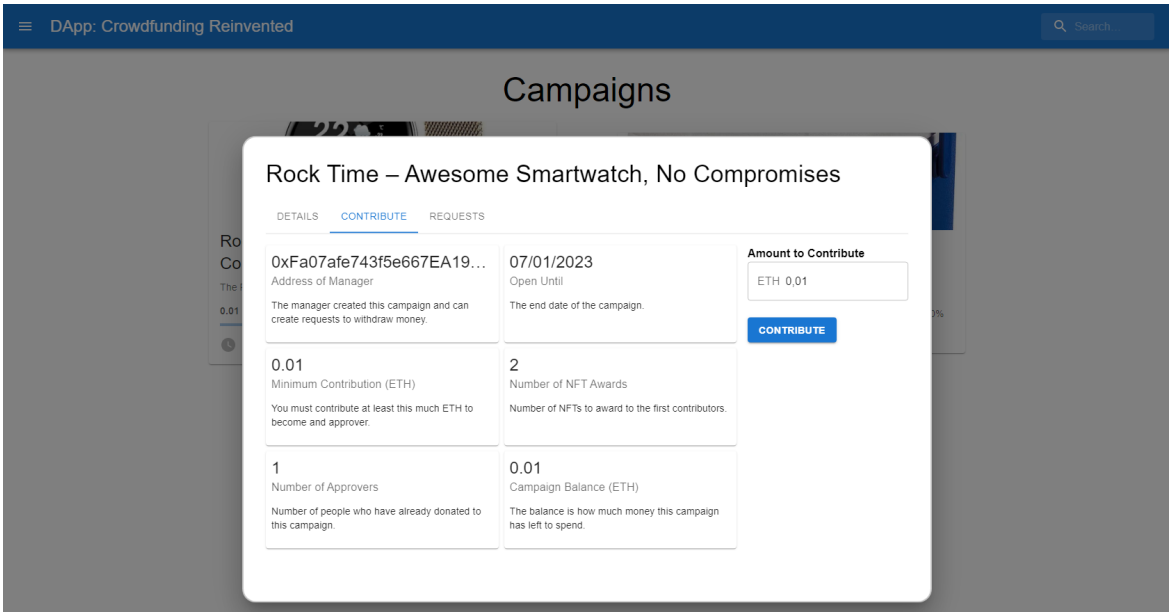


Figure 10: DApp: Campaign - Contribute tab

5.1.2 Contribute to campaign

From the **Contribute** tab, the user is given the option to donate an arbitrary value to the campaign, provided it is larger or equal to the minimum contribution value set by the campaign's owner. For this, the user can insert the desired value in the numeric box and press the button to proceed with the transaction.

The following details are also provided in this tab:

- Address of the campaign's owner/manager
- The end date of the campaign
- Minimum value of each contribution
- Number of NFT awards that are awarded to the first contributors
- Number of contributors (approvers)
- Remaining balance of the campaign

5.1.3 Campaign Requests

When the campaign's manager creates a request to spend some part of the funds raised in the campaign, this information must be available to all the contributors, so there can be voting to determine whether the request will be approved or declined. In this tab, the user can see every request created for the campaign and, if the request is active, can approve it by pressing the **Approve** button.

Each row in the list of requests contains the following:

- Description of the request
- Value representing the part of the raised funds to be spent
- Approval count in the format of V/R:
  - V represents the sum of contributions of the contributors that already approved the request
  - R is the total raised funds in the campaign
- Date of creation of the request
- Option to approve the request
- Option to finalise the request when there are enough approvals (only available for the campaign's manager)

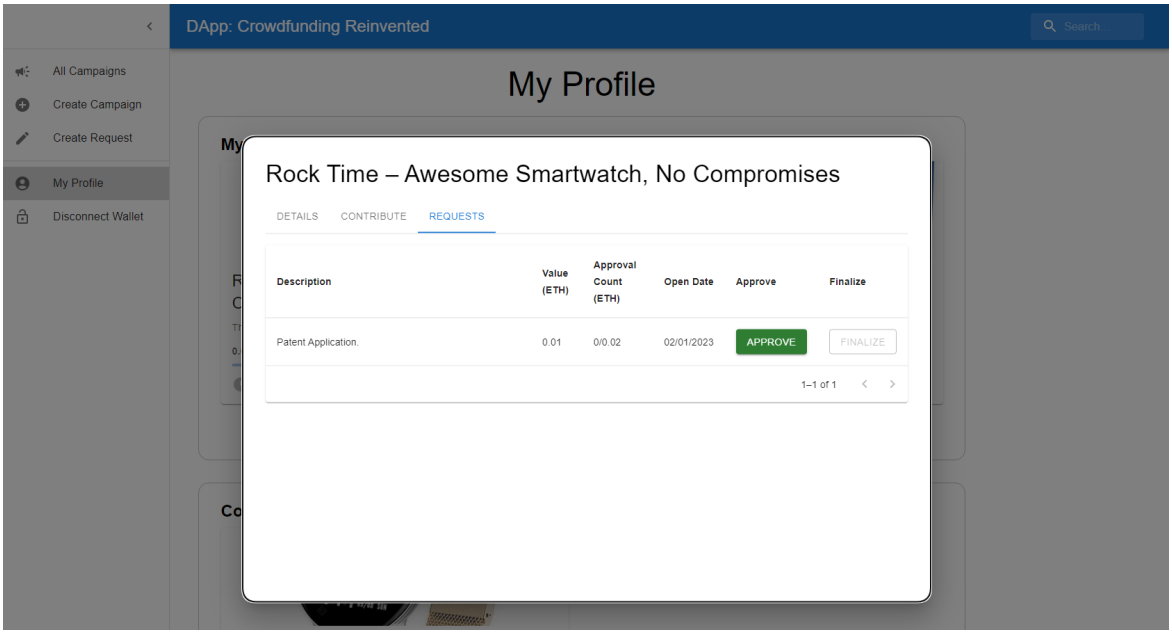


Figure 11: DApp: Campaign - Requests tab

5.2 Create Campaign

To create a new campaign, users can access the **Create Campaign** page. It consists of a two-step form that allows inserting all the needed details to start a campaign, such as:

- Title for the campaign
- More in-detail description of the campaign

- Minimum value for the contributions/donations
- Pretended target of funds to be raised
- Price of the campaign's product
- End date of the campaign
- Representative image of the product
- Set of images for the NFTs to be awarded to the first contributors

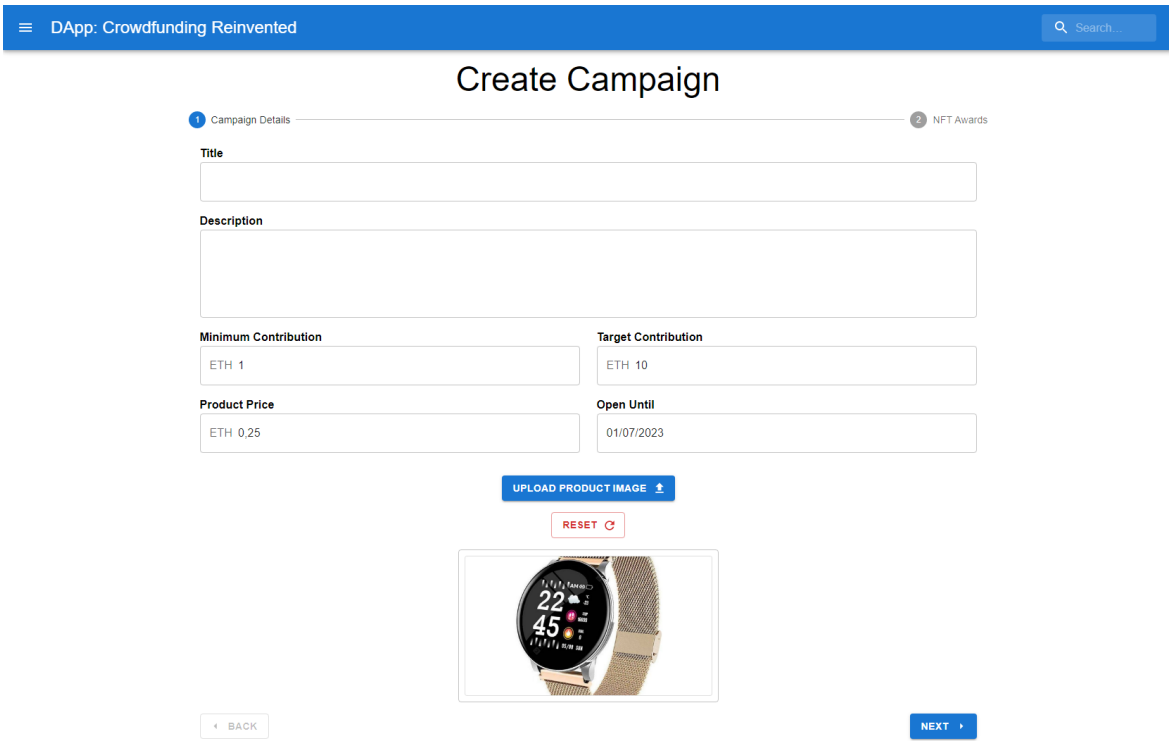


Figure 12: DApp: Create Campaign - First Step

5.3 Create Request

As the owner/manager of a campaign, it is possible to create a new request to spend funds on this page by completing a form with the required information:

- Detailed description or reason for the request
- Value of funds to be spent for the request
- Campaign for which to create the request for

The **Campaign** field presents a drop-down with every campaign associated with the currently connected user.

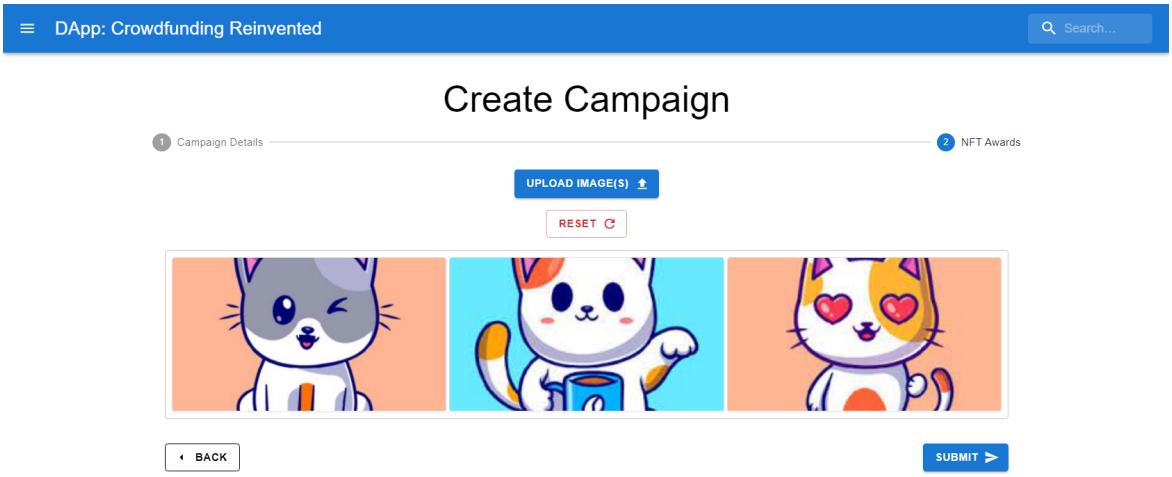


Figure 13: DApp: Create Campaign page - Second Step

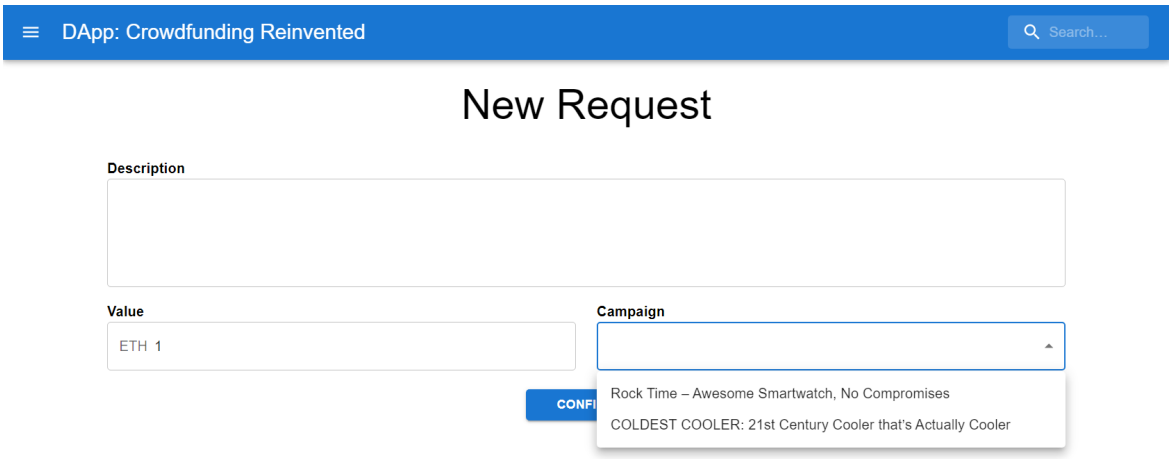


Figure 14: DApp: Create Request Page

5.4 User Profile

When a user's wallet is connected, they can access the **My Profile** page, which contains three different sections with user-related information:

- **My Campaigns** - List of all the campaigns the user has previously created
- **Contributed Campaigns** - List of all the campaigns the user has donated to or bought a product from
- **Awarded NFTs** - Gallery of the NFTs the user was awarded from the campaigns they contributed to. It is possible to select each image to view the NFT and its details on the corresponding page on OpenSea.

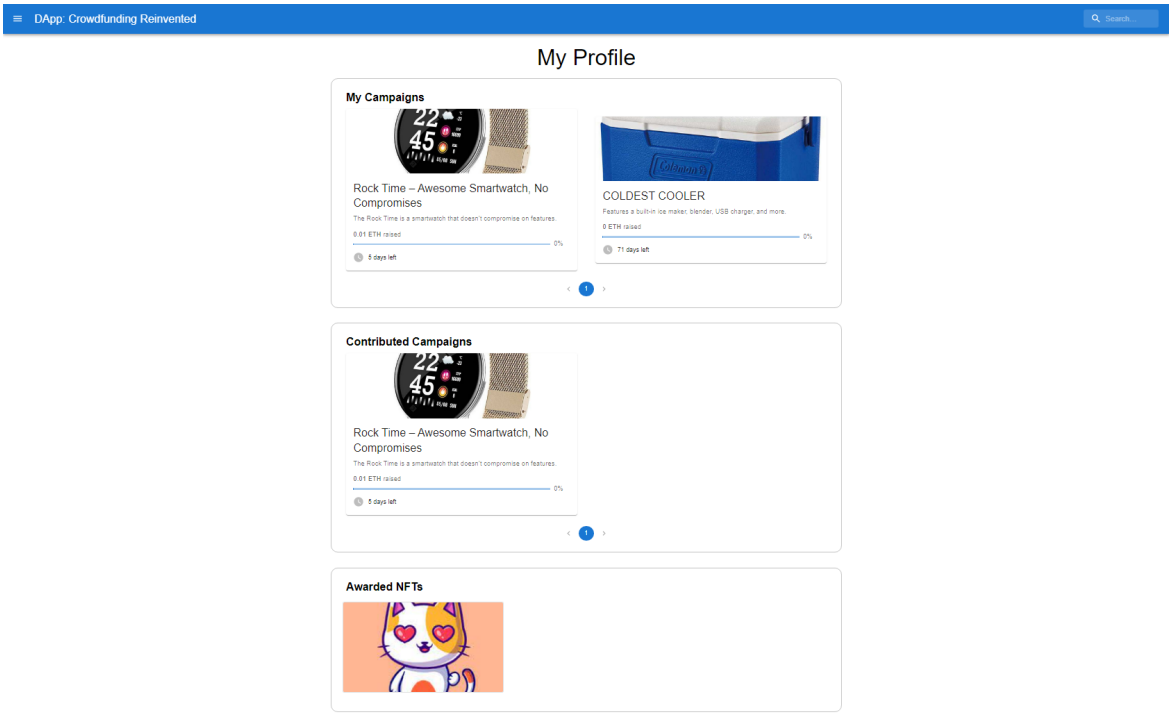


Figure 15: DApp: My Profile page

6 Known Issues and Limitations

During the conceptualisation phase, we have identified some limitations in our work and potential problems that may arise upon deployment and utilisation of the platform developed.

6.1 Cryptocurrency popularity

The use of cryptocurrencies for transactions such as donations to campaigns and acquisition of its corresponding products, which are the core of the crowdfunding platform created, is still a concept that is not broadly used or known, especially when compared to normal money transactions through bank transfer, credit card, or other popular methods.

This represents a limitation to the platform's efficiency in providing a large set of investors to its campaigns and products due to the possibility that most potential investors interested in the campaigns do not use cryptocurrencies.

6.2 Unverified campaigns and products

Every user who signs up to the platform can both donate and create campaigns. However, there is not any verification step in place to check if the products offered by the campaigns are real and will be delivered to the investors and if the campaign owner is committed to do what is described in the campaign. This could only be solved by some kind of Trusted Third Party that would verify the legitimacy of every campaign and corresponding products before allowing owners to raise funds.

7 Future Work

There is always room for future improvements. Our project is no exception. The possible future developments we foresight are:

1. Display the delivery information filled up when buying a campaign's project.
2. Allowing a campaign to have more associated products.
3. Creating an internal marketplace for transferring and selling NFTs awarded due to campaign contributions.

8 Conclusion

The development of this project led to a deeper understanding of the principles of the blockchain technology and how to use it to create a platform that could benefit from it. Beyond the acquired skills on Solidity development and knowledge on the Web3.js, including smart contracts, transactions and NFTs, it was also an opportunity to develop a more critical thinking on the potential of the blockchain to enhance existing applications and platforms.

References

[1] 08 - First Steps with DApp programming (slide 8). https://classroom.google.com/u/1/w/NTQ5NjM1MTQ0MzUz/t/all. Accessed 30 December 2022.

[2] Solidity Documentation. https://docs.soliditylang.org/en/v0.8.17/index.html. Accessed 30 December 2022.

[3] Material UI - overview. https://mui.com/material-ui/getting-started/overview/. Accessed 30 December 2022.

[4] Web3.js - Ethereum JavaScript API. https://web3js.readthedocs.io/en/v1.8.1/. Accessed 30 December 2022.

[5] MetaMask - Introduction. https://docs.metamask.io/guide/#why-metamask. Accessed 30 December 2022.

[6] Alchemy. https://www.alchemy.com/. Accessed 30 December 2022.

[7] ERC-721 Non-Fungible Token Standard. https://ethereum.org/en/developers/docs/standards/tokens/erc-721/. Accessed 30 December 2022.

[8] OpenZeppeling - ERC721. https://docs.openzeppelin.com/contracts/4.x/erc721. Accessed 30 December 2022.

[9] Pinata - Pinning. https://docs.pinata.cloud/pinata-api/pinning. Accessed 30 December 2022.

[10] Goerli Faucet. https://goerlifaucet.com/. Accessed 30 December 2022.

[11] Metadata Standards. https://docs.opensea.io/docs/metadata-standards. Accessed 30 December 2022.

[12] Hardhat - Documentation. https://hardhat.org/hardhat-runner/docs/getting-started. Accessed 30 December 2022.

[13] Architecture of a DApp. https://www.geeksforgeeks.org/architecture-of-a-dapp/. Accessed 30 December 2022.

[14] How to Create an NFT on Ethereum Tutorial. https://docs.alchemy.com/docs/how-to-create-an-nft#step-3-add-goerlieth-from-a-faucet. Accessed 30 December 2022.

[15] How to Mint an NFT from Code. https://docs.alchemy.com/docs/how-to-mint-an-nft-from-code. Accessed 30 December 2022.

[16] NatSpec Format. https://docs.soliditylang.org/en/develop/natspec-format.html. Accessed 30 December 2022.

[17] Mongoose - Documentation. https://mongoosejs.com/docs/guide.html. Accessed 30 December 2022.