POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Non Fungible Tokens in Digital Art: a case study with Ethereum

Supervisor:
Prof. Pierluigi Plebani

Candidate:
Davide Clementi - 919807

Academic Year 2020-2021

Acknowledgements

I would like to thank my supervisor Prof. Pierluigi Plebani for his availability, for having guided me and giving me the opportunity to participate to this project. My thanks also go to Memooria Srl and to Advanced Technology in Health and Wellbeing group of the IRCCS San Raffaele Hospital for supporting me in the realization of the prototype and for the numerous material provided to me. Last but not least, I would also like to thank my loved ones for their support throughout my university career.

Abstract

Nowadays blockchain technology, a cutting-edge sector in fintech, have grown exponentially and also the applications became various and in many fields. The reasons behind this fast evolution can be found in the features of decentralization, transparency, security, reliability, traceability and censorship-resistant that the blockchain ecosystem provides. In the last years, this technology has been taken into consideration for its intrinsic characteristics as a possible solution to the creation of certificate and attestation between two or more parties, as the whole process is able to assure the validity of the contract even without the presence of a legal authority. Recently, blockchain technology has also entered the art world. The introduction of Non Fungible Tokens into the blockchain has made it possible to associate an ownership to works of art and collectibles while maintaining the property of uniqueness. The so called cryptoart is characterized in fact by the possibility to publish digital art work into blockchain in the form of NFTs. This aspect gives the possibility to manage the ownership and transfer of artwork in secure and verifiable manner. The goal of this thesis is to provide a practical running example on how Ethereum can be used to create and manage Non Fungible Tokens in the domain of Digital Art. The blockchain ecosystem is used to integrate an existing system, able to generate images from body-sensor data of an individual, in order to provide a service that makes the resulting personal photo a non fungible token.

Keywords: Blockchain; Off-chain; On-chain; NFT; Smart Contract

Abstract in lingua italiana

Al giorno d'oggi la tecnologia blockchain, un settore all'avanguardia nel fintech, é cresciuta in modo esponenziale e anche le applicazioni sono diventate varie e in molti campi. Le ragioni alla base di questa rapida evoluzione possono essere individuate nelle caratteristiche di decentralizzazione, trasparenza, sicurezza, affidabilitá, tracciabilitá e resistenza alla censura che l'ecosistema blockchain fornisce. Negli ultimi anni questa tecnologia é stata presa in considerazione, per la sue caratteristiche intrinseche, come possibile soluzione alla creazione di certificati e attestazioni tra due o piú parti; tutto il processo é in grado di assicurare la validitá del contratto anche senza la presenza di un'autoritá legale. Di recente, la tecnologia blockchain é entrata anche nel mondo dell'arte. L'introduzione dei Token non fungibili nella blockchain ha permesso di associare una proprietá ad opere d'arte e oggetti da collezione mantenendone l'unicitá. La cosiddetta cryptoart si caratterizza infatti per la possibilitá di pubblicare opere d'arte digitale in blockchain in forma di NFT. Questo aspetto da la possibilitá di gestire la proprietá e il trasferimento delle opere d'arte in modo sicuro e verificabile. L'obiettivo di questa tesi é di fornire un esempio pratico su come Ethereum puó essere utilizzato per creare e gestire token non fungibili nel dominio dell'arte digitale. La piattaforma blockchain viene utilizzato per integrare un sistema esistente, in grado di generare immagini a partire dai dati che provengono da sensori corporei. Il ruolo della blockchain in questo scenario é quello di fornire un servizio che renda la foto risultante un token non fungibile.

Keywords: Blockchain; Off-chain; On-chain; NFT; Smart Contract

Contents

List of Figures

Chapter 1

Introduction

Specific domains such as brands protection and art require to guarantee the scarcity property and certify the ownership of a given source. These properties are better known as intellectual properties and in the physical world are guaranteed by the usage of contracts certified by a central authority (lawyers, notaries). These certificates are useful to verify the ownership and keep track of eventual transfer of ownership.
In this context, the blockchain is more and more emerging as a promising technology able to guarantee these properties without relying on a central authority to guarantee the authenticity of the certificates. In fact, with blockchains, thanks to the mapping of all the transactions, everyone is able to see who is the actual owner of a given source. However there is another problem, if we consider also the context of virtual objects, the traditional web paradigm is based on many copies and replications of single sources (abundancy), with this scenario it is very difficult to guarantee who is the owner of a given asset.

Goal of this thesis is to investigate how blockchain can be used to guarantee the property of a digital assets, that in the specific case analysed is an image. What makes the image important to protect in terms of property concerns the generation process which is based on an artificial intelligence algorithm driven by the sensations felt by the user who is watching a photograph. On this basis, the resulting image is unique as it is the result of the neurological reaction of the viewer measured using specific body sensors.

In particular, the thesis investigate the use of NFTs (Not Fungible Tokens) as a way to achieve the aforementioned objective. In fact, NFT are able to

guarantee the scarcity property. Each token generated will be the unique result of the image generation process mentioned at a precise timestamp represented by the mint of the token itself. In the result obtained is important to consider also the nature of the different data used, otherwise it will be hard to find a feasible solution. The goal of this thesis is to propose a feasible approach to these problem, analysing the processes, the feasibility, the performance and the cost of a possible implementation.

This document, including the introduction as first chapter, is divided in six chapters:

- The second chapter have the purpose to clarify the context of the entire work and create a common background. In this chapter will be provided in particular a in-depth description of the blockchain technologies, problems and eventually possible solutions. In particular with the solution will be also described which are the best practices to minimize the cost and maximize the performance.

- The aim of the third section is to describe the project analyzed which has proposed the image generation process. The chapter gives an overview analysis of the set of data inside the project and a brief description of the algorithm used. This description is important to understand how process works and what the focus of the analysis will be. All these information have been provided by the team associated to this project.

- The fourth chapter contains a brief description of the process we need to map. Inside this section we can also find the motivations behind the choice of some specific tools and technologies for the solution of the problem. It will contain also a documentation to the design and analysis process of the project. This part is very significant since it includes the modelling of the application architecture. Here there is a description of all the main decisions taken for the actual development of the proof of concept.

- The fifth part is a documentation of the project implementation, considering problems, performance cost and solutions.

- The sixth chapter represents what conclusions can we draw about the results we obtained and some final considerations.

Chapter 2

Technology background

2.1 Blockchain paradigms

Blockchain is a distributed, decentralized and cryptographic ledger characterized by blocks of immutable recorded transactions shared among multiple nodes linked in a P2P network. The main characteristics of blockchain are transparency, security, reliability, traceability and censorship-resistant. These characteristics, gives to untrusted partipants the possibility to communicate and send transaction between each other without any need of a trusted third party[11]. The components that are usually included inside a blockchain and integrated inside a single software client are[11]:

- "A P2P network connecting participants and propagating transactions and blocks of verified transactions" [11].
- Messages as transactions.
- A consensus rule used to guarantee the validity of a transaction.
- A State machine that according to consensus rule defined, process the transaction.
- A Chain in which are constantly added and linked all the valid blocks.
- A consensus algorithm used to decentralize the control over the blockchain, by forcing participants to cooperate in the enforcement of the consensus rules
- An incentivization scheme to economically secure the state of the machine in an open environment

2.1.1 Blocks

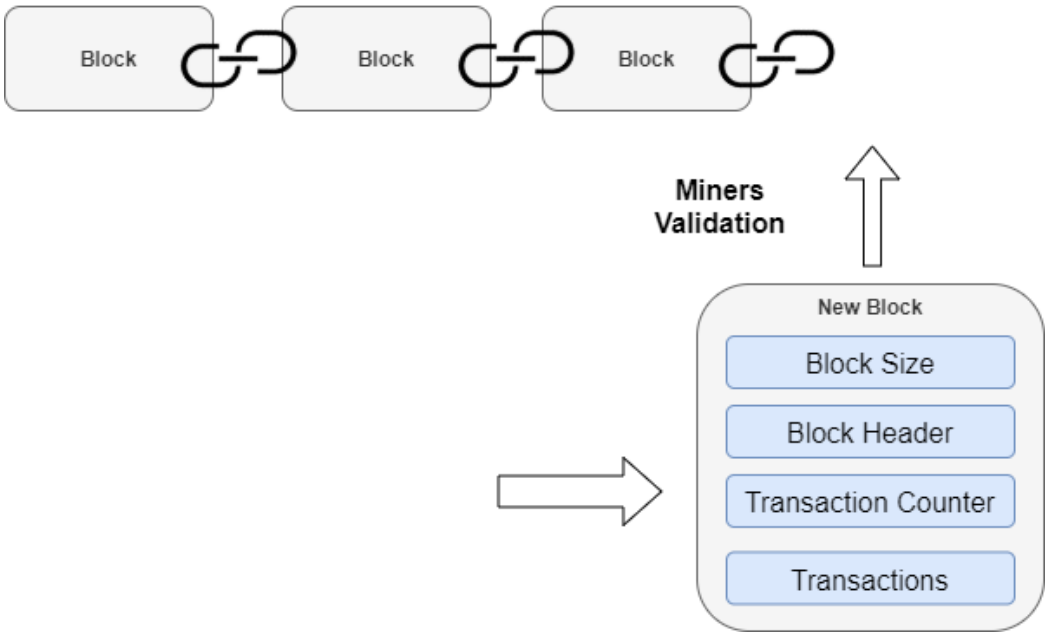

Figure 2.1: Block Structure

The Blockchain technology is made by units of blocks, each one is characterized by[12]:

- The block size, representing the dimension of the block
- The block header which it will be describe with detail later
- Transaction Counter is the number of transactions inside the block
- Transactions is the set of transactions inside the block

Among all these fields the block header assumes a particular relevance, this portion is composed by[12]:

- The version of the protocol used
- The timestamp which report the time in which the block has been mined
- The difficulty is an indicator of how challenging was the problem to solve from miner perspective

- A pure numeric value named nonce
- A reference to the block that comes before the new one inserted. This connection has done including the previous block hash
- the root hash of the Merkle tree, representing the way transactions are organized inside the block [12](Bitcoin and Ethereum). Merkle tree is a tree data structure in which parent nodes have pointers to child nodes as reference and the leaves contain the hash of the transactions[6].

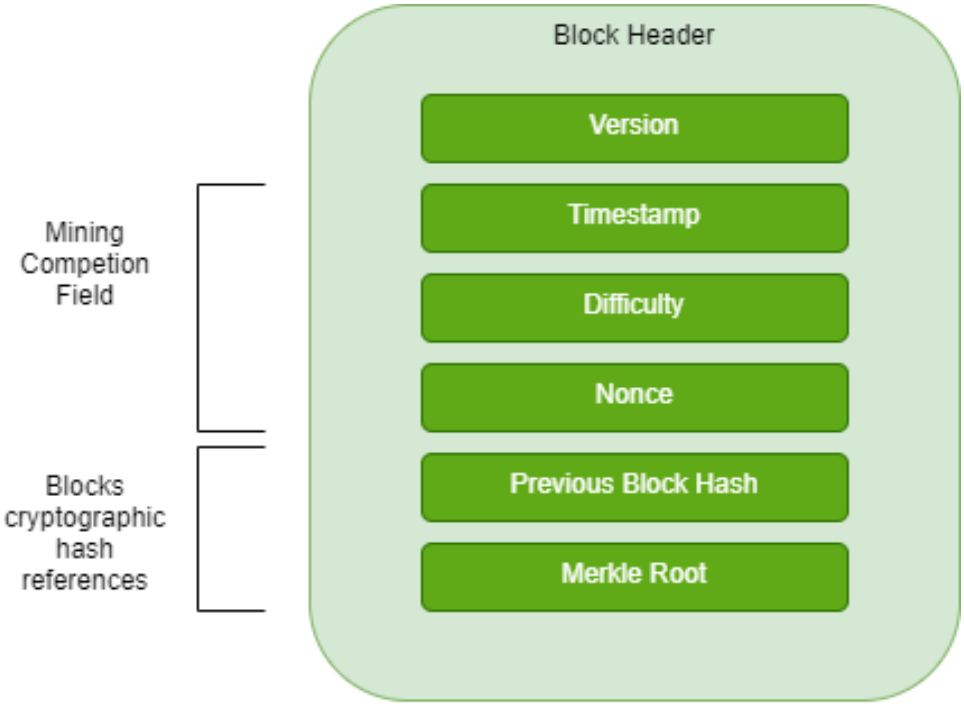

Figure 2.2: Block Header

2.1.2 Consensus Algorithms

Consensus algorithm has a crucial importance inside the blockchain ecosystem. The two kinds of consensus algorithm which are currently widely used are [22]:

- Proof-of-work where the participants in order to mine and validate a block are required to solve a computational problem of increasingly difficulty. Each participants proportionally with its effort is then rewarded with an amout of coins specific to the blockchain. The algo-

rithm can be defined as function of the number of participants and the difficulty of the computational problem to solve.

- Proof-of-stake is intrinsically linked to stake or in better word the currency holding. The higher is the stake the higher is the credibility and assurance that the node will not try to tamper the ledger. The algorithm gives a proportional weight to each node according to its stake.

The main disadvantage of the POW is the significant power consumption required to perform the mining which affects the overall energy consumption of blockchain technology. For this reason miners need to improve their device and need of course power supply to remain profitable and do their job. This algorithm create a potential security problem if miners can secure a 51% plus stake they can potentially tampering the algorithm and make the blockchain insecure. However this may require an amount of sources that is far larger than the benefits it can gives. The assumption is generally that the miners are considered fair (Bitcoin).

2.1.3 Smart contract

Smart contracts represent program that operates inside the blockchain, their development is useful to automatize and customize transactions according to the different needs. Their main characteristics are to mirror the main functionalities of the real contract without the need of a third party certification authority. Smart contract as a real-world contract represents a binding agreement between two or more parties. The smart contract is automatically executed once the terms of an agreement of the specified conditions are met[10]. The absence of a third party is justify by the distributed and verified code execution among the network nodes that are inside the blockchain network. Smart contracts need to consume gas to make the transaction works, this cost of execution belongs to the fact that the transaction is executed on multiple nodes and each one need a reward for the usage of their computational power. Another important characteristic is that as a real world contract smart contract are immutable and not censorable. The range of Smart contract application is very wide, some of the many case of usage can be smart propriety, e-commerce, real-estate, business process and even digital/musical property.

2.1.4 Blockchain platforms

There are many blockchain platforms the most famous and used, that maintain their position on top of market cap, are Bitcoin and Ethereum, but there are also many other emerging ecosystems such as Ripple, Algorand, Cardano. Among all this blockchain technologies there are few of them that gives through the smart contracts the possibility to the developer to implement customize solution. Ethereum is one of the best well documented platform with a wide dynamic community. The scripting language adopted by Ethereum is turing complete and this gives the possibility to realize more complex and customize contracts than some other blockchain solutions.

2.2 Token standards

Tokens can be described as tradable digital units that encapsulate a value that is not only to be considered as a purely economic but it includes also other aspects such as reputation, copyright, utility and voting rights. This kind of digital asset can be minted by any individual or organization that have to define the set of rules that characterize the token itself such as the monetary policy and the value it represents within the ecosystem. An important distinction needs to be done between coin and token. The first is a native "token" that represent the cryptocurrency which runs on a specific blockchain; the second is a digital asset that runs on top of a blockchain and can be used for different application generally in the context of the organization that created it. It is important to remember that tokens have value only in the context of the organization, outside of it they have no tradable and economic value. Crypto currency however rise the problem of double spending, that is to say the problem of spending twice a given cryptocurrency[18]. The blockchain solve the potential double spending-problem of digital currency thanks to the usage of a peer to peer network that validates each transaction with a cryptographic signatures[14]. Token can be divided according to their fungible property in fungible token and non-fungible token. The concept of fungibility can be expressed as the possibility to trade a token with other tokens of the same type (and value)[17].

2.2.1 Fungible Token

The core of Fungible token are three main principles[13]:

- Token of the same type needs to be considered equivalent, since they are indistinguishable and identical in value.
- A token can be traded with another token of the same type (and value).
- A token can be divided in smaller fractions and maintaining the two property described before.

The interface used by Ethereum to represent the fungible token is the ERC20[16].

2.2.2 NFT

NFTs (Non-Fungible Token) differently from the fungible token are not interchangeable since they have to represent unique assets. This kind of token is linked to the need of representing digital, copyright property of collectible object that for their nature must be unique. Another important property is the indivisibility, the fact that this kind of token represent unique and not interchangeable object make it unfeasible and unreasonable to consider the possibility to fractionize it. Another important point is that even if the data to which the NFT refers may have other copies, each single NFT creation is considered as unique. In better word if we got a unique digital asset we might create different NFT from it but each one will be distinguishable from the other one since each one will have a different hash identifier. For example we can say that even if someone create another token from the same art work the identifier associated to it will be different and distinguishable from what was the real owner identifier; consequently we can say who is the real owner of the original asset. For each mint is possible to track all the asset property movement from an owner address to another and everything got a timestamp where all the passages are visible to everyone.

One of the most famous implementation of this interface is in 2017 with "criptokitties" a game created on top of Ethereum, the popularity of this project paved the way for the usage of NFT in also other fields. In particular this new token was really suitable to keep track of what can be named as intellectual property especially the digital one. Recently at the end of 2020 and 2021 NFT start to become very used among digital content creator, artists and brand that want to sold their unique product[17].

The standard used by Ethereum to manage the NFT is ERC721. One of

the main difference between this kind of interface and the ERC20 standard is different management of the address and in particular the link between the addresses and the token. In ERC20 the necessity was to create a link between the address and the token, in the ERC721 there are an association between an address and multiple token since each one may be unique.

2.2.3 Semi-fungible Token

One extension of the NFT model can be the semi-fungible token in which the idea is to group assets that can be either fungible or not fungible. In Ethereum this kind of token can be implemented with the extension of the ERC1155 interface [17]. This solution guarantee the possibility to bind an id address with multiple tokens even if they are of different kinds. With this new interface is possible to save cost in transfering multiple tokens of any kind and in the deployment of the smart contract avoiding the production of useless redundant bytecode[24].

2.3 Oracles

2.3.1 Oracle Problem

The development of applications and smart-contracts on blockchain has allowed the creation of reliable, safe and tamper-proof programs. On the other hand, these guarantees given by blockchain exist because the environment is by its nature closed to everything that does not appear to be on-chain, therefore everything that is outside the ecosystem is considered unreliable.

With the growth in complexity of applications developed in a decentralized way, the need for interaction between on-chain programs and what is off-chain, in other words what is external to the blockchain, gain a great importance. In particular, the exchange of information between the parties became relevant, since the cost of storage or computation on blockchain environment in some case may become really costly and even unfeasible for the block limits.

The interaction with what is external to the blockchain is made available through the existence of some middle party operate between what is trust on the on-chain and what instead is not outside the chain. The name usage to describe these middle parties is oracle. The choice of this name refers to

the capacity of these third parties to answer to questions that are external to the blockchain ecosystem and that can only be assume as valid.

This create a problem, the blockchain paradigm is the decentralization but with this new kind of third party entity, all the information coming from the outside are given by the oracle itself that for its properties can be considered has some sort of centralize authority. The other important problem remaining still is the trust of this bridging party. With this scenario the centralize oracle represents a bottle-neck since smart contracts will rely on these information, even if these data has been manipulated or filtered. Some of the solution adopted by this kind of centralized oracle try to show as a guarantee a certificate that validates and proof the authenticity of the external sources. For example one centralize oracle adopting this solution is Provable that make available, to the smart contract developer, some proofs of validation.

2.3.2 Available Solution

The proposed solution is given by the usage of a decentralized oracles principle to solve partially the problem of the central trustless authority. The decentralized oracles are relying on a network of computers that operate as different nodes each one of this is called miner. The solution is partial since it depends on the formulation of the aggregate function of the decentralized oracles. Generally this kind of aggregation is made according to a certain consensus algorithm that can be different depending on oracle's network policies. In most of the available solution however is not take into account the origin of the sources, for that reason is up to developer to choose only trustworthy entities to retrieve these kind of information.

2.3.3 How Oracles Works

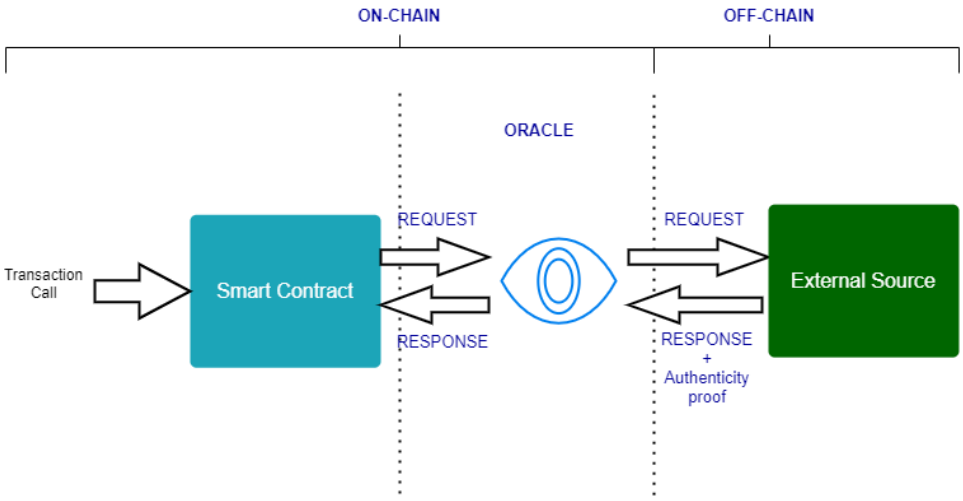

Figure 2.3: How Oracle works

The usage of oracles is very various and for different purposes some of the main common functionality are retrieving information from the off-chain world or delegate the computation to external sources. In order to guarantee the fairness of the transaction on the blockchain environment, some infrastructures have adopted the usage of the decentralized oracle network (DON). In this solution we have a group of independent oracles that provide data external to the blockchain. This will improve the security of the system since everything will work according to a consensus rule and without the presence of a single central entity. In DON the nodes are selected according to the network policy and perform the required jobs, then according to the consensus algorithm that verify the integrity of the data retrieved/computed, the miners can, with the policy of the network, receives a reward for the good work or even a penalty if the algorithm detects fraudsters. Miners need to be considered as computers running a software that receives and executes tasks. Generally the reward that is given to the miner by doing its job correctly is the specific token of the Network. The smart contracts usually interacts with the oracles by the usage of a standardize interface. The API offered can be implemented and be adapted to smart contract according to the purpose, generally the oracle interfaces offer some query function to retrieve/compute the data offchain useful to the program.

2.3.4 Oracles Patterns

Oracles need to be considered as a bridge working between the blockchain ecosystem and the outside world and can be used in both direction either from the blockchain to the off-chain or also from the off-chain to the blockchain. For this reason we can consider to group the oracle in two main categories inbound or outbound based, each one can have a two kinds of approach push-based or pull-based. With inbound it is indicated a flow of information or data coming from the off-chain and directed to the blockchain, whereas with the outbound word we refers to an opposite flow originated from the on-chain ecosystem and incoming into the outside world. The two fundamental approaches pull-based and push-based are similar to two very common patterns in other network protocols known as request-response and publish-subscribe. The first approach start with a request for given data/source and it ends with an answer from the other party that can be either positive or even an error. The push-based approach instead is characterized by two main group of actors. We have the listener (also known in publish-subscribe as subscriber) that is waiting to receive the notification of a certain event that is posted by the publisher.

Combining these typologies we obtain the four standard pattern generally adopted Pull-based inbound Oracle, Push-based inbound Oracle, Pull-based outbound Oracle and Push-based outbound Oracle.

2.3.4.1 Pull-based inbound Oracle

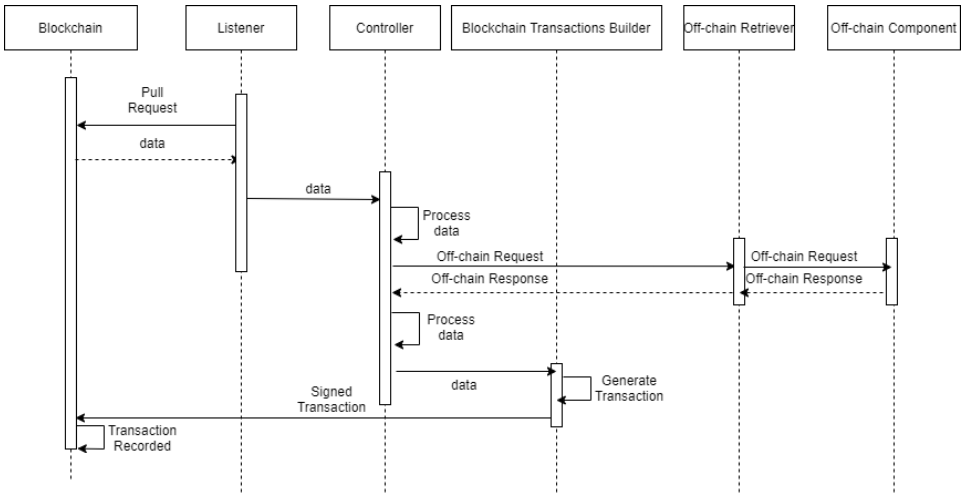

Figure 2.4: Pull-based inbound Oracle, [21]

This solution works in the following way[21]:

- The smart contract send a request state to the oracle
- Once the oracle received the request, it will ask to the off-chain the smart contract desired state
- The oracle forward the state required to the smart contract

The positive aspect of the Pull-based inbound oracle is that the response to the request is always provided no matter if it is either positive or an error answer. The negative aspect is that the arrival time of the answear relies on the speed of the on-chain ecosystem.

2.3.4.2 Push-based inbound Oracle

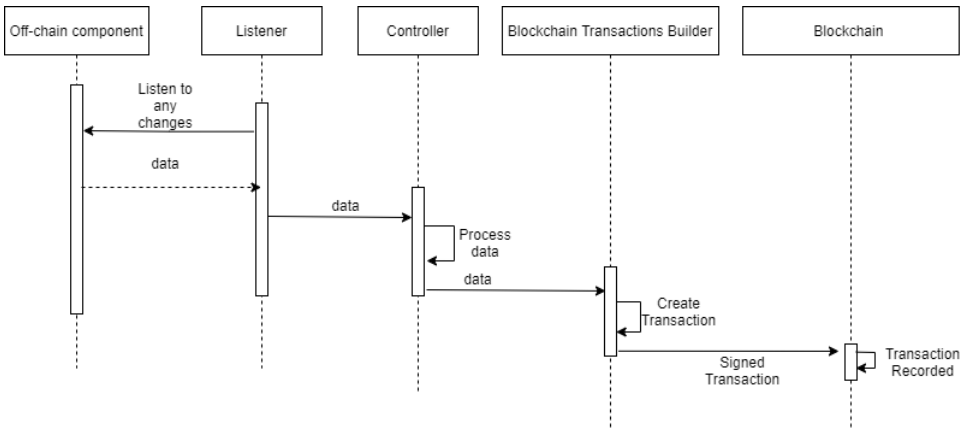

Figure 2.5: Push-based inbound Oracle [21]

This kind of approach works as follows[21]:

- The blockchain is registered to be notify after a certain event occurs
- The event occurs and the oracle transfer the interested data from the outside world to the on-chain environment

2.3.4.3 Pull-based outbound Oracle

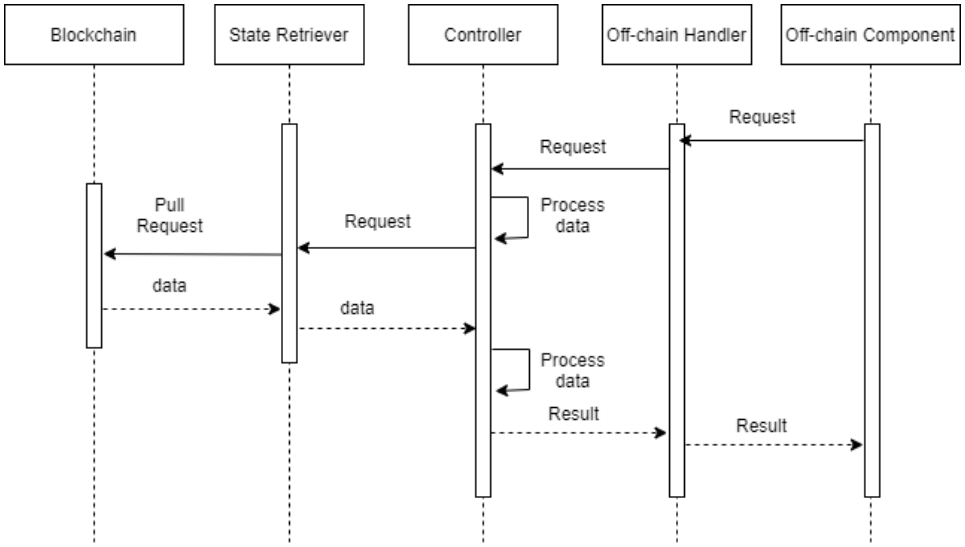

Figure 2.6: Pull-based outbound Oracle [21]

This kind of pattern follows these steps[21]:

- The off-chain applications ask for data inside the blockchain
- The Oracle forward the request to the blockchain
- The Oracle receive the response from the blockchain and send the answer back to the off-chain applications

The positive point is the opportunity to request in a precise way the data on the on-chain. The negative aspect is that this approach might require some time because it depends to the blockchain dimension.

2.3.4.4 Push-based outbound Oracle

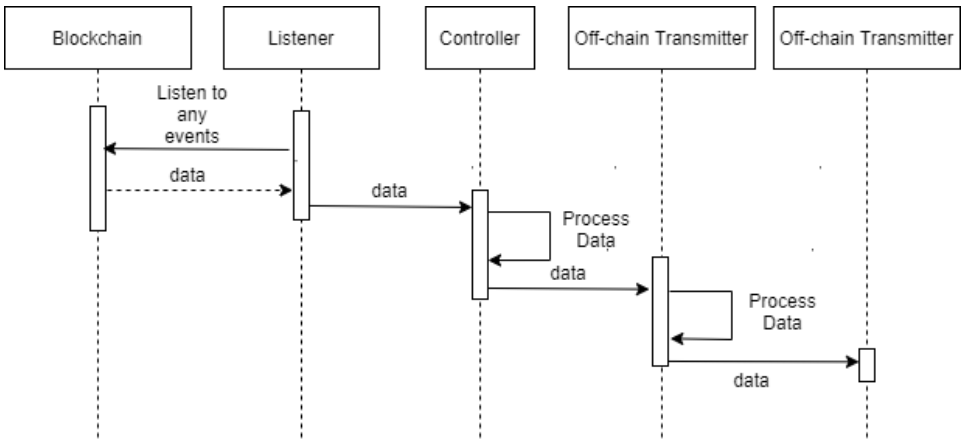

Figure 2.7: Push-based outbound Oracle [21]

The steps required in this standard are[21]:

- The oracle listen to the blockchain and receive sooner or later the data required by the off-chain system
- Once the event is triggered the oracle send the data back to the off-chain

2.3.5 Hardware Oracles

In case there are any data that need to be constantly monitored outside the blockchain, can be took into consideration also hardware oracles. This solution is characterized by the usage of sensors and IoT technologies in order

to observe physical data emitted by the interested process, transform them in way that can be comprehensible to the digital world and then transmit them to the smart contract.

Chapter 3

Case study requirements

In this section we will have a brief description of the use case requirements for the project. Here will be included also all the different sources, provided by Advanced Technology in Health and Wellbeing lab of San Raffaele Hospital, for the analysis of the process.

3.1 Overall Process

The aim of this project is to develop a solution to the problem of monitoring the sensations and the feelings of a person who is watching a given visual artwork. The focus of this process is to produce as output another unique photo characterized by the same input photo, modified in some parts, according to the sensations of the observer. The application can be summarize as a block-box receiving in input a given artwork and user sensations and producing in output the input picture modified. This result need to certify the sensations felt by the user when he/she was watching the picture.
The main process and components can be briefly described as follows:

1. The user is sitting and watching a given photo, the name associated to this first picture is Neffie.

2. The body sensors monitor user body activities while he/she is watching the photo. When the user wants to stop the experiment all the data collected by the sensors and the associated Neffie, will be sent to an AI algorithm.

3. The algorithm with the data provided will do an elaboration that

eventually will end up with the production of a new photo, whose associated name is Coffie.

3.2 Data Analysis

Before starting the design choices for the Dapp development might be a good practice to focus on a classification of what elements will come into play during the process execution.

3.2.1 Data in Input

The first thing to do is to make a clear distinction between the inputs of the process. We need to consider essentially two types of input: primary and secondary.

The primary inputs are associated to the inputs produced by the human body and subsequently perceived by the tools. The secondary inputs can be considered as the union of what are the signals produced by all the sensors and the image (also known as Neffie) observed by the user. The data that we will considered inside the smart contract development are the secondary inputs, because they are fundamental too feed the algorithm and then to produce the Outputs to be included inside the NFT.

All the unfiltered signals are saved as file .csv inside a dropbox folder. The secondary input files produced by each monitoring process are:

- Emotive EEG (sensor parameters that might be changed by the researcher)
- EEG is useful to observe brain activity, the tools associated is a sensor helmet.
- SHIMMER GSR is the signal that represents the variation of skin sweating. It is kept by two electrodes that first inject a costant current and then measure the difference of impedance.
- SHIMMER PPG is characterized by one mono frequency electrode "Transmitter-Receiver", this one acquire the blood pressure variation corresponding to a blood vessel in order to interpret user cardiac activity. This signal and the GSR are kept by a tool including all of the three electrodes and is put on the subject wrist.

- TOBII GAZE is the signal obtained by tracking the eye movements, observing the coordinate of eye position.
- WEBCAM `FACE_EMOTION` differently from the other data, in this file, we found data that are not raw. These inputs are produced by an AI system that exploit the usage of the deep-learning to predict the subject feeling by looking at facial expression. For this reason the data contained in this file are probabilities associated to each possibile feeling the user may have at each period of time.

3.2.2 Data in Outputs

The outputs produced by the process represent the results of the elaboration of the AI algorithm chosen and the secondary inputs given to the algorithm.

Inside the system we have different kinds of output to consider. These results are distinguishable in "static" output, artistic outputs, time laps output and intermediate outputs. The static result corresponds to a report containing user experience information, specifically it has a more technical and scientific description of how the process works specifying the signals involved and five captions:

1. The image (NEFFIE) chosen by the user with a short introduction to the picture.
2. The Heatmap of what are the main points that capture the user attention.
3. The graphic representing the trend of the GSR curve and the portion of the picture corresponding to the peak.
4. An image representing the main emotion captured by the deep learning algorithm.
5. The cognitive load associated to the analysis of the parameters associated to the helmet. These parameters are elaborated from the RAW data coming from the EEG with the usage of a deep learning algorithm that do the computation on a cloud computing system.

The artistic outputs has this name, because the subject indirectly with his signals gives a "personal" interpretation to the photo he has watched. The

new generated photo is also called COFFIE because it represents the cognitive photography related to user feelings. Inside the "output"/"input" folder there are two files, one is the image COFFIE the other one contains the image with a layout formula. This equation includes a synthesis of what the AI algorithm is doing.

The time laps output is a video of the first five second of the experience, with an evaluation of the neurophysiological body response.

The intermedium outputs are the file "SGN Cardinals", "SGN Entire" and "PredAndGSR timestamp"; they are calculated with the raw signals in input. Among these results the file "PredAndGSR timestamp" is useful as input to the AI algorithm that produce the static report (final report). "SGN entire" is useful to the creation of the heatmap in the "static" output. The "SGN cardinals" file contains some useful information about the COFFIE creation. The creation of the COFFIE takes into account:

- The blur of specific points that capture the user focus.
- The portion size of the picture to modify.
- How big it has to be increase that specific portion.

3.2.3 Other files

In addition to the already described files in input and output we got also some other file saved inside the folder. The other files included inside the folder are:

- Playlist is a file representing the image chosen by the subject during the session, which includes also the path that individuate the location of the image.
- Sensor configuration indicates which are the sensors that are active during the process, just in case disabling the sensors does not compromise the system execution. For the moment the system works with the assumption that all the sensor are enabled and the parameters are correctly set.
- `Session-info` includes all the information of the user session, the parameter inside also takes into consideration the reliability percentage of the packages that have been transferred. The fraction of EMOTIV is

not 100% since all the computation is done in cloud and it is difficult to have everything perfectly sinchronized due to the different clock frequency.

These files are not directly useful to our purpose but they might be important as well for the correct functioning of the process.

3.3 Algorithm

Another component of this project is the algorithm, we consider it as something already provided and useful to our purpose. All the rights of this algorithm are reserved and associated to Neffie team, we consider it as it was provided to us with no modifications. The main component of the algorithm is written with the usage of java programming language and its role is to call all the other components including also parts, library and executable written in other languages (Python, MATLAB). the main objective of the algorithm is to receive the input Neffie image and generate, according to the sensor data associated to the user, another image (Coffie).

3.4 First Consideration

The sensations transmitted as input to the application can be tracked as time series of a given amount of time, where in each instant of time is reported the given type of sensation. The sensations take into account are heartbeat, eye movement, sweating pressure and brain waves. The output produced by the system need to be considered unique and associated to a given person, for this reason we consider to represent this aspect as an NFT token.

The necessity to work with big time series create a problem on the storage of all this data. Saving all of these information will create feasibility and cost issues. The solution is to consider the usage of an oracle in order to get only the data required by the smart contract, that are only a small amount of all the data generated by the monitoring process. The image generated according to the good and common practice of NFT, to reduce the cost of blockchain usage, will be stored on an outside service. On the smart contract will be considered only a reference to that file.

3.5 Oracle Analysis

The choice of the oracle(s) to be use inside the project have to take into consideration:

- The possibility of doing computation on data in order to create a synthesis of what is really needed to the smart contract
- The compatibility with the NFT interface
- The cost of usage in terms of gas (cryptocurrency used by the current blockchain)
- The possibility to use even an Hardware Oracle based on sensors in order to monitor the relevant activity of the user
- A possible oracle able to certify that a given action happen at a certain time or using as alternative a proof of validation

Chapter 4

Design implementation

In this chapter, there will be a description of what and how at high level the application will be developed, including all the process and different considerations took into account before the effective code development. The considerations done, can divide the development process in essentially two cases of study.

4.1 Process phases

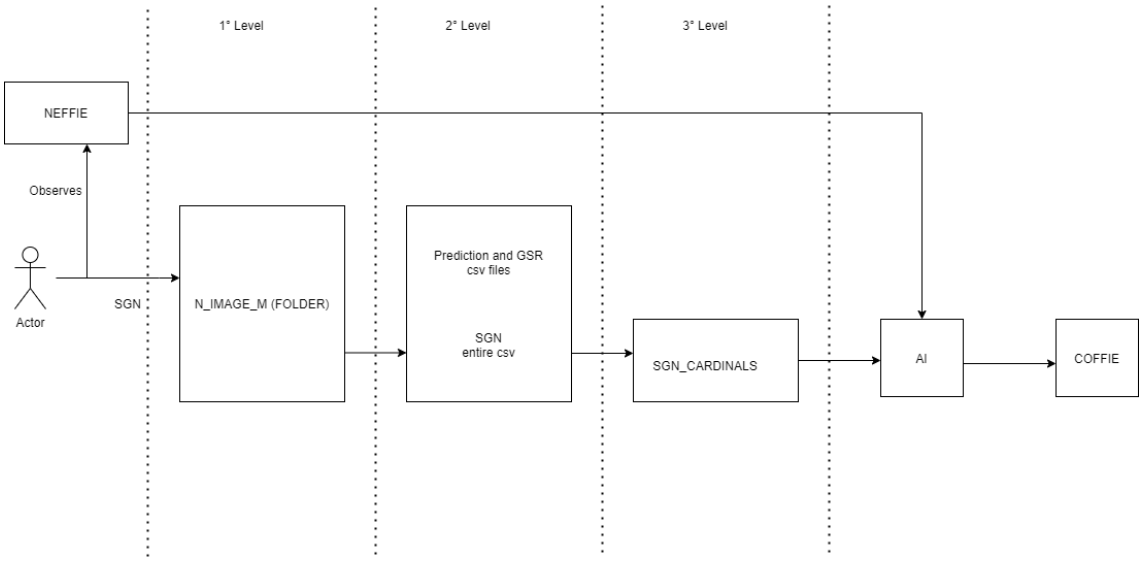

Figure 4.1: Process Workflow for Coffie creation

The input data produced by input signals and the Neffie picture are sent to

an AI algorithm that generates the outputs. The role of the smart contracts is to consider the input and output data, keep track and validate the data generated from the algorithm and manage the ownership of the data. All of these activities need to balance the trust of the blockchain platform and the cost of using it. However it is also possible to focus the attention on other less trust but more cheaper alternatives off-chain.

4.2 Assumptions

All the data saved in inputs, outputs and all the elaboration process is assumed to be perfectly working. The algorithm as hypothesis will always produce a response.

4.3 Blockchain platform

Many available platforms give the possibility to implement smart contract, however our choice was focused on finding a robust and well documented network with the possibility to reduce to the minimum the development costs. For these reasons the platform we have chosen is Ethereum. To minimize the expense we have considered also the implementation of some other models, always with the idea of preserving the main features of the application.

For this reason, in the analysis, we took into account some alternative to the classical public blockchain technology. These other possibilities however even if they minimize the cost they do not guarantee always some of the main properties of the blockchain such as trasparency, immutablility, decentralization, non repudiation and censorship resistant. These solutions include private or permissioned blockchain techonologies.

4.3.1 Ethereum

Ethereum is one of the most uses and well documented open-source platform in particular for smart contract development. There are many supported open-source libraries and utilities that can be used by programmers to integrate their smart contract with the best practices to reduce gas consuming (ether) and improving the security of the contract.

In Ethereum blockchain are available for the development some standard

API interface. These are more or less suitable for most of the use case. These interfaces are very important to create a common knowledge that make the communication between the user, including application, wallet and broker and the smart contract itself as standard as possible. This set of standards is classified by the usage of the term ERC (Ethereum Request for Comment), all the old and new interface are in continuous development by the Ethereum community. EIP (Ethereum Improvement Proposal) are concise designed documents used by the Ethereum community to improve and introduce new technical features generally for development purpose[4].

4.3.2 Private or Permissioned blockchain

With permissioned blockchain it is possible to define only partially the decentralization property. This kind of blockchain is able to perform features that the traditional blockchain is not able (or only partially) to guarantee. One of this new functionality can be for example the real-world identity of the user that have done a certain transaction. For this reason this kind of blockchain seems very suitable in regulated industries. "For example banks are required to establish the real-world identity of transacting parties to satisfy Know Your-Customer (KYC) regulation"[25].

The standard blockchain is instead known as permissionless blockchain. This name is used in fact to indicate a completed distributed system in which everybody can join without any kind of restriction.

In Permissioned blockchain there is one or more central authorities as gateway, their role is to assign a certain level of privilege to each node. These authorities are able to give permission to nodes to do transaction, to mine or in some case even to create assets. It is important to specify the existence of some trade-off approach between permissioned and permissionless blockchain, this in order to include transaction processing rate, cost, censorship-resistance, reversibility [2].

4.4 Oracles adoption and useful pattern

A more deeply analysis of the project lead us to the take some considerations concerning what kind of oracle(s) can better satisfy our needs. The main hardware component to take into account is represented by the helmet with inside the sensor that monitor user's sensation. Since it is a single helmet

we do not consider multiple nodes as oracles we will therefore have only a single central element that is capturing different stream of input. It can be consider also the possibility to adding another oracle in order to certify the produced output of the smart contract at a given time.

A useful pattern that can be taken as reference for the next phases of implementation can be the Push-based inbound Oracle. In our case the blockchain needs to receive from the sensors (off-chain) the streams of input and a possible solution to solve this issue is to apply the Push based, in this way the smart contract keeps the input from the oracle that is listening when the sensors are active. The pull-based possibility might be useless on the inbound since we cannot know in advance when the process starts and ends. For this reason even if it is less expensive than the push based, it has no sense in this specific scenario to use the pull approach.

Other valid options that might be useful are a pull-based inbound oracle to retrieve the time and maybe a pull-based outbound approach to send the result to an off-chain storage.

4.5 Storage

For the storage of the NFT result we decide to use a folder on IPFS. Our decision ended up on this system because unlike centralize storage solution it is single-point failure resistant and the decentralized infrastructure on top of it reflects the same main paradigms of blockchain itself (decentralization, censorship-resistant and immutability). IPFS unlike other storage infrastructures is content base, therefore to access the content of a file we do not use the location but we address the content in order to retrieve the file we are interested in. The content-based solution prevents the possibility to access to malicious content since the reference is an unique hash to a specific content, the location-based addressing refers to an address that is more human friendly than an hash but for this reason the content might be not what we aspect[5].

The data coming from the original project are saved in a dropbox in the following extensions in .pdf, .jpeg, .csv and .mp4.

4.6 Data Ownership and Access

The data in Input and in Output need to be associated to a certain ownership. When the process has finished, the owner has the right to access to all of these data. However in the blockchain environment a reference to the transaction of the output is always available to everyone that wants to verify the ownership.

The input picture need to be associated to who produced the photo, in other words the photographer. The ownership of the inputs signals belongs to the monitored subject. The Output is the product of the photographer, the algorithm that produced it and the subject that has been observed, for this reason in the NFT we need to set these characteristics. With the Dapp developed on top Ethereum blockchain each of these actors need to access their data. The application need to make visible the NEFFIE (input image) and the COFFIE (output image), all of the other data can be accessed through a link that point to storage location but are not directly show by the application itself.

4.7 Architecture of the Prototype

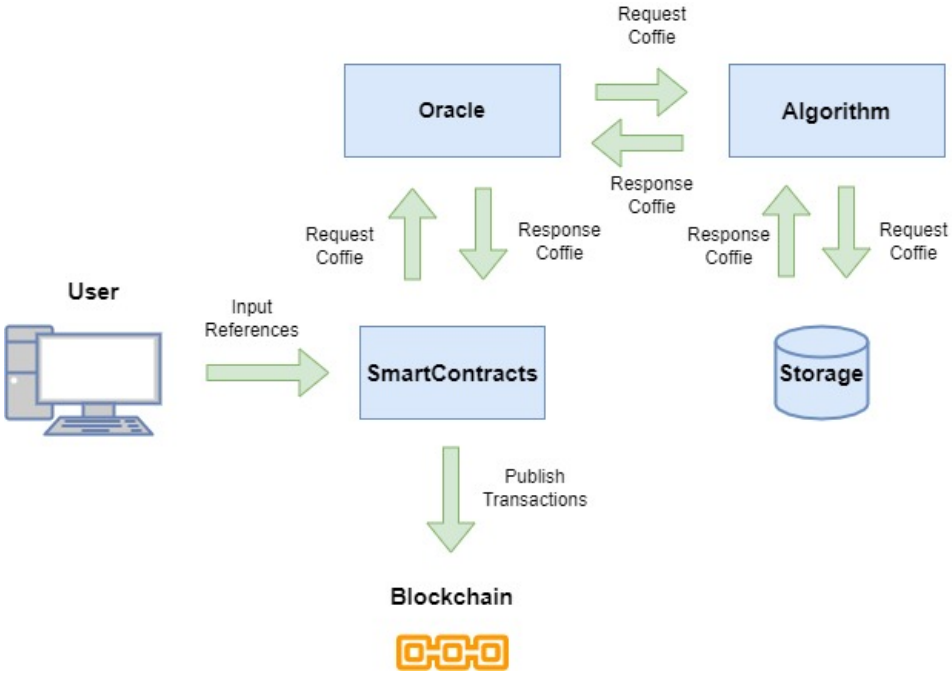

Figure 4.2: Architecture

The figure above shows a possible structure of the architecture for the practical implementation. The components are essentially:

- An interface for the interaction with the smart contracts.
- The oracle(s) to call and get back the result from the algorithm elaboration.
- A storage unit to collect all the data in input and ouput.
- The smart contracts communicate with both oracle and interface. After the algorithm elaboration there will be the generation of the Not Fungible Token associated to the output.
- The blockchain in which all the main smart contracts actions and the resulting transactions are recorded.

The architecture of the application will take into account the possibility of rely on oracle solution, for the main elaboration process of the application.

This solution has a good advantage in terms of gas cost and a simplify smart contracts development. The main problem in this solution is loss of trust, one of the fundamental property of the blockchain. This aspect however need always to be considered when we have to deal with some data stored offchain.

The idea of this solution is to consider essentially two main smart contracts. The first will be associated to the output data, in order to store the result on the correct address on the blockchain. The second smart contract is intended to be devoted to the algorithm that, with the incoming inputs, generates the desired outputs.

The incoming data are stored inside IPFS and their hash will be saved inside the algorithm smart contract in order to associate it with the correct address. In the same way the results produced by the algorithm will be stored as output inside IPFS and then saved inside the output smart contract, the COFFIE will be managed to include the three process to which the image is related to. The intention is also trying to take into account both the expense and the performance from the point of view of the cost and time spent.

4.7.1 Data Managing

The data in inputs for example cannot be consider as internal since they always coming from the external world that is clearly offchain.

This consideration and the possible solution will be a bit in contrast with the trust property of the blockchain. An example of trust problem might be in what the original image will be. One question might be in fact how can I be sure about what image was used as input by the process. The same case can be propagated also to the data coming from the sensors, in this case the problem is even more difficult to approach since we need to consider also other aspects that are connected to multiple time series information. We need to be sure that the biometrics parameters monitored by the sensors belongs to the user who is watching the picture and not to somebody else.

Some of the Output Data need essentially to be stored or at least containing a reference to the algorithm results. This because otherwise, if we consider all the data outside the blockchain, it will be impossible to generate an NFT. Regarding the property of ownership there are different approach useful to consider the share of a property. Here there is a list of possible alternatives:

- Considering a standard interface that provides functions that deals with the realization of share ownership.
- Implementing a smart contract, that manage the ownership problem, through a simple mapping and assignments of the NFT variable.

However since the usage of NFT for mapping the result of the process is almost necessary, the second solution was almost mandatory. The aim of the sequence diagram below is to provide a technical description of how the smart contracts should interact for the NFT creation.

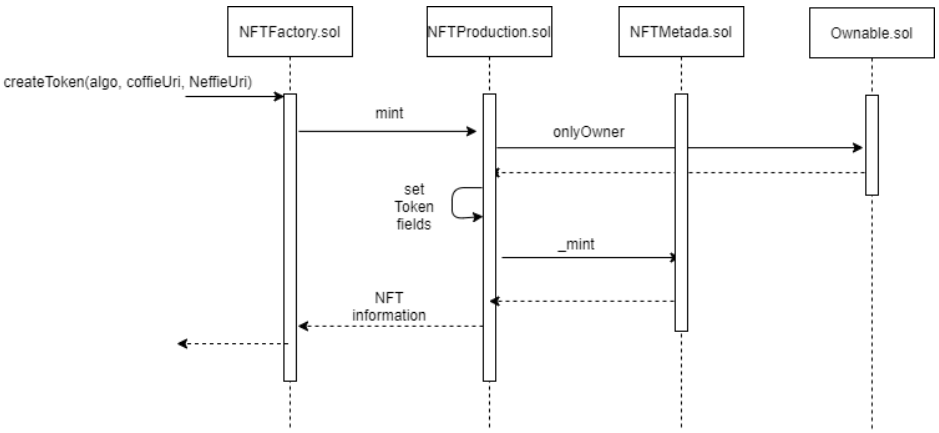

Figure 4.3: NFT creation sequence diagram

4.7.2 Algorithm

In this type of solution the management of the algorithm will be a trade-off between a full development onchain and offchain.

The objective is to consider onchain only some main important characteristics which may be of less impact on the cost but that are able to guarantee a good trust. All the heavy computations that might require a good expense of resources will be left offchain and accessible throught the usage of an oracle.

4.8 Design Considerations

Before working on the code, it might be important to clarify all the central problematics that may rise from the analysis of the workflow.

The storage platform used is dropbox, but for our purpose is more suitable to use as explained before a distributed file sharing system (IPFS). So one of the step is to transfer the information from dropbox to the IPFS platform. This has to be done for all the data of our interest stored in the Dropbox folder (input and output).

Another important consideration to be done is how to launch the algorithm, after the capture and validation of the input Information. A possible solution is to create a user interface that act as a client. The user can interact with the interface and get all the necessary information such as the data in input and output associated to a certain user address.

4.9 Deployment methodologies

The are many options for smart contract development one of them is to use a local node running on pc at a spefic port and the usage of a CLI or GUI to see the publication of the transaction on the local environment. Some other options that works pretty similarly to the blockchain ecosystem are testnets, each one providing a different proof and build on top of different types of nodes. Some of the main testnets are:

- Ropsten a proof-of-work testnet that emulate most likely the behaviour of the blockchain.

- Kovan a proof-of-work testnet working on top of OpenEthereum clients[3].

- Rinkeby build on top of geth client is a proof-of-authority network. This means that new blocks are added by a set of pre-defined trusted nodes, instead of by whichever miner provides a proof-of-work[3].

- Goerli a proof-of-authority network, compatible with both Geth and OpenEthereum clients[3].

4.10 Client technology

In order to find a way to call and post transaction on blockchain systems the developing of the only smart contracts is not enough, we need a way to interact with the contracts through an interface.

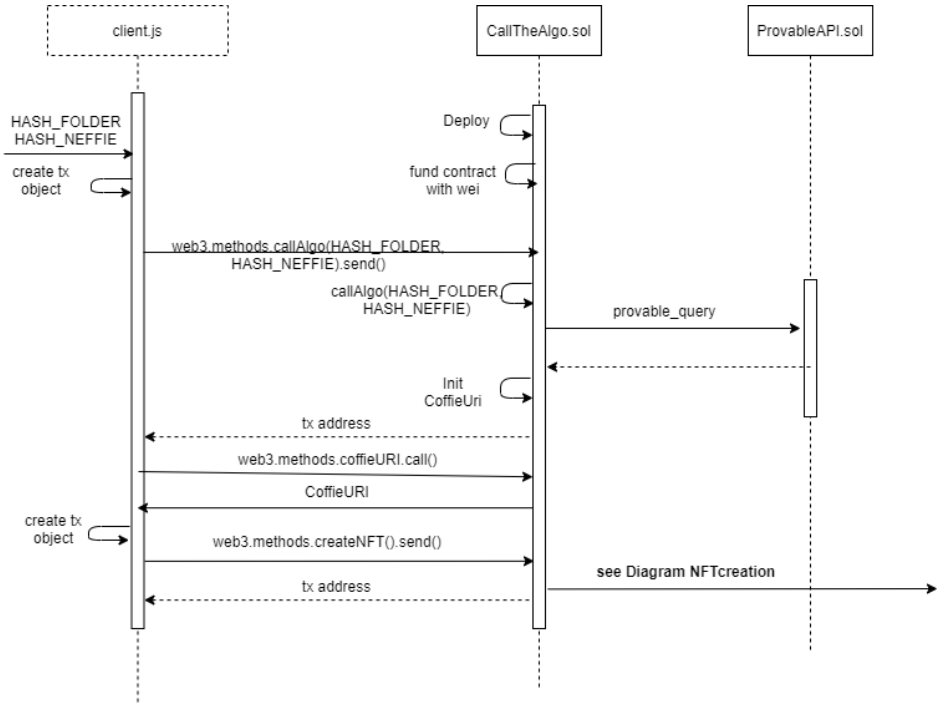

Figure 4.4: UML client interface workflow

This kind of interaction is also useful to have a first approximation of what will be the the expense of interacting with our smart contracts. It is also important to test the performance in terms of time of waiting to receive a confirmation of the transaction publication on the blockchain. A set of libraries useful to manage the communication with the blockchain environment is web3, this is avaible for both Python and Javascript technologies.

For the first realization of the project it will be simply used node.js to write the user interface. In future versions of the tool, it might be included the usage of a specific framework such as angular/react/vue to create a complete frontend and give to the user a good experience and usability of the application.

Chapter 5

Implementation document

In this section will be described the choices done during the implementation of the Dapp. This description will always take in consideration and divides the two main phases described also in the chapter below and will be based on the samples of data and executable files provided by the Neffie team project.

In both cases there is the assumption of the migration of data information from Dropbox to IPFS. This is a key point to make the system work as expected.

5.1 Prototype

The aim of this section is to include all the main characteristics of the first prototype of the application.

5.1.1 Folder Organization

In this subsection will be described the folders organization of the project. This might be useful to make fluently the project navigation and to make it more readable.

5.1.2 Smart Contracts Project

The first project have been developed with the usage of the remix framework for both deploy and development of the smart contract. Here we have essentially the folder contracts that includes all the smart contracts utilized. In this directory there all the contracts needed to our application and also the

interfaces needed to the creation of a provable query and to the production of the NFT. The folder contracts include also another folder called artifacts, in this one is fill with the JSON ABI of the smart contracts that have been compiled.

There are included also by default some other folders inside the project repository[1]:

- Scripts, here there are some basics scripts that are useful for the smart contracts deployment and migration
- Tests is useful to test our smart contracts, and see if they work as expected
- .dep is an additional folder in which is possible to store imported file from external repositories, such as github

5.1.3 Client Folder

This folder includes all the tools and components to make the interface work. It is simply composed by a file client.js which is the core of the interface. In node modules are instead visible all the library and file used to develop the client interface. The two main modules that have been imported are web3 to setup an environment that is able to communicate with the ethereum blockchain and ethereumjs-tx to send firmed transactions[9].

5.1.4 First Phase description

The input is stored on the blockchain thanks to the usage of the hash obtained by the storage IPFS. This is useful as reference for the user of the application that wants to access to the data on the blockchain. For each file in input will be considered an hash unique value associated to the root folder. The smart contract is essentially characterized by simple functions useful to store the various hash.

The script file linked to the smart contract has essentially to follow this steps:

- Give the reference to the CallTheAlgo.sol smart contract
- Generate the NFT and get the transaction id and CoffieURI of the result

The algorithm will be called and executed exclusively offchain. In particular it will have a smart contract dedicated call "CallTheAlgo.sol" and it will include an hash code as a successful result. The execution of the algorithm is totally delegated to an oracle that validate the execution and a make request to the algorithm. The call of the oracle has been done with the usage of a script.

The output has been developed and represented with a similar pattern as respect to the input code. Almost everything with a relevant size has been considered external to the blockchain. In the smart contract it is only possible to find the reference of the files' storage location. The smart contract of the output, such as the input, have the same function whose utility is to save as variable the reference of the file. The hash reference to the COFFIE and the hash of the NEFFIE are saved inside the token during the creation.

The smart contract used to represents this process are three:

- CallTheAlgo.sol
- NFTFactory.sol
- NFTProduction.sol

5.1.5 The oracle

The oracle that has been used inside this solution is provable. In the smart contract "CallTheAlgo.sol" there is a method that got as parameters the HASH folder in which we will found the results and the HASH of the NEFFIE. After the execution of this method we will get the location of the output file (COFFIE). The query used inside the smart contract is a provable_query of computation type.

```
/** Parametro riferito alla cartella contenente NEFFIE
*/
function callAlgo(string memory _hashfolder,string memory _hashNeffie) public payable{
    currentFolder = _hashfolder;
    currentNeffieHash = _hashNeffie;
    if (provable.getPrice("computation") > address(this).balance) {
        emit LogNewProvableQuery("Provable query was NOT sent, please add some ETH to cover for the query fee");
    }else{
        emit LogNewProvableQuery("Provable query was sent, standing by for the answer..");
        //l' ipfs sta a rappresentare l' ipfs dell' archivio
        //FIRST CASE COMPUTATION 3 MINUTE PROCESSING AND HIGH COST
             provable_query("computation",["QmSJK3QzhMvWSaWfV4quK7mBeVMGJQEs92fDwJSSM6pGpU", _hashfolder]);
        //SECOND CASE POST QUICK, CHEAP AND EVEN MORE SCALABLE
        //  provable_query("URL","json(https://us-central1-coffiegen.cloudfunctions.net/app/?var="+ _hashfolder + ").HASH");
    }
}
```

Figure 5.1: CallTheAlgo provable query

For what we can observe from the code, the query has to receive as parameter an IPFS HASH corresponding to container of the computation and the parameters to be passed to the algorithm. In order to execute the algorithm for the project has been utilized a docker container working on his own by launching a python script whose aim is to simulate how the real algorithm should work. The script receive the parameter passed to the smart contract and produce as output the IPFS path to the result COFFIE. In order to be executed by provable we need to follow these steps:

- Create a zip for the root folder with inside both the docker file and the script
- Save the Zip in IPFS
- Copy the IPFS generated and paste it as first parameter of the function provable_query

On the smart contract side once the call has done, we need to wait a minimum amount of time before receiving the __callback with inside the result of the algorithm. Once the result is obtained is stored inside the smart contract variable "CoffieURI".

```
//callback quando il risultato è pronto
function __callback(bytes32 myid, string memory result, bytes memory proof) public override{
    require (msg.sender == provable_cbAddress());
    currentCoffieUri = result;
}

fallback() payable external{}
```

Figure 5.2: CallTheAlgo provable __callback

The __callback of provable only the first time is free, the next times it requires the usage of the smart contract balance. So the smart contract got also a fallback anonymous method that is useful to make possible the transfer of gas from the wallet account to the contract.

To guarantee a proof certification of the result given by the oracle, provable give us difference kinds of proof[7]:

- TLSNotary Proof, works on top of TLS functionality. In particular the TLS master key is split between three entities the server, an auditee and an auditor. This last two actors are provable itself considered as the auditee and a particular designed AWS image will act as the auditor.
- Android Proof it is based on a software remote attestation developed by google, the aim of this kind of certification is to make sure that the application is running on a safe environment that is directly linked to the Provable infrastructure.
- Ledger Proof is useful to attest to a third party that either an application and a device developed by provable are running in a TEE of a ledger device.

5.1.6 NFT development

```
contract NFTProduction is NFTokenMetadata, Ownable {
    string uri_neffie;
    string uri_folder;
    string uri_token;
    bytes32 algo;


    event create_coffie(string);
    event created_coffie(address , uint256, string);

  constructor() {
    nftName = "Coffie";
    nftSymbol = "COF";
  }

  function mint(address _to, uint256 _tokenId, string calldata _uri) external onlyOwner {
    super._mint(_to, _tokenId);
    super._setTokenUri(_tokenId, _uri);
  }

  function setNeffie_uri(string memory _uri) external{
      uri_neffie =_uri;
  }

  function set_Algo(bytes32 _algo) external{
      algo =_algo;
  }

  function setSignals_uri(string memory _uri) external{
      uri_folder = _uri;
  }

}
```

Figure 5.3: ERC721 implementation

The smart contracts that are involved in this process are actually all the contracts in the project. This because the production of the NFT can only happen after the algorithm execution and once the result has been saved inside the smart contract variable "CoffieURI". Inside "CallTheAlgo" we got both the initialization of the CoffieURI fields and then the creation of the real NFT. With the createToken method we will make a call to the NFTFactory smart contract that will create a new NFT.

```
function createNewToken(address _user, bytes32  _algo, string memory _neffie,
string memory _uriFolder, string memory _uriCoffie) public returns(address, address,
bytes32, string memory, uint256){
    NFTProduction nft = new NFTProduction();
    tokenId++;
    //set to associate token information
    nft.setNeffie_uri(_neffie);
    nft.set_Algo(_algo);
    nft.setSignals_uri(_uriFolder);
    nft.mint( _user , tokenId, _uriCoffie);
    nfts.push(address(nft));
    users.push(_user);
    addressToUri[_user].push(_uriCoffie);
    return (address(nft), _user,_algo, _uriCoffie, tokenId) ;
}

function AllUriByOwner(address _owner) public view returns (string[] memory){
      return addressToUri[_owner];
}

function getAllNFTAddress() public view returns(address[] memory){
    return nfts;
}
```

Figure 5.4: NFTFactory smart contract implementation

The information of the NFT minted for the given account will be saved inside a struct with all the different fields related to the Coffie token. After that the data have been saved inside the struct it will be push inside an array in order to keep track of all the token that have been generated.

The use of the intermediary smart contract "NFTFactory" is necessary to develop a solution that takes into consideration both provable and the NFT without overcome the limit size of the block. The NFTFactory contains a simple function called "CreateNewToken" which will:

1. Generates a new NFT by assign to a temporary attribute the new "NFTProduction" object.

2. Set all the main important characteristics to be contained inside the NFT, such as the uri of the Coffie and the reference to the linked Neffie.

3. Call the mint function of the NFT production to create a new Token associated to the account of the user who is doing the call

We got also some other get methods to retrieve some useful information about the token that have been created. The "NFTProduction" smart contract is a simple NFT smart contract derived from the interface ERC721 provided by Ethereum. The only functionality is the mint that generate the

token and associate it with the account specified. It might be considered for a future development also a simple implementation of a "safeTrasfer" function in order to trade the token created from an account to another one.

5.2 Final realized

In these realized since we try to focus our attention on performance and cost of execution, we decide to use another kind of NFT interface the ERC1155 standard. This interface gives us the possibility to save some cost and time in the transfer of the ERC since in this case we are able to move multiple NFT of the same type at the same time from an account to another.

This part presents also another solution for the algorithm elaboration. In order to make the execution and the answer from the algorithm faster and even less gas expensive considering the provable query on smart contract side, the prototype will use a web service to execute and receive the result of the algorithm.

5.2.1 Analysis of the two standards

All the Ethereum standard interfaces are useful for a personalize token representation, they are characterized by some functions that are fundamental for both creation and transfer of tokens from an account to another one. Generally we can see in all the representation at least four functions [16][15][24]:

- Constructor useful to initialize some of the characteristics of the token itself such as for example the name and the symbol, these specific features might be important especially for the user to better distinguish the new token among the other possible one.
- The mint is used by the smart contract itself to generate the token, this function will always require an address where to mint and some token features.
- The burn is instead used to delete the token that have been generated by sending them to an address where the token are not more retrievable and reachable.
- The transfer that in some way it must check the validity of the transaction to send, for example the amount of token of the owner must

be sufficient compared to the quantity to be transferred. After all the validations required, the token and eventually its given amount can be transferred to the address that has been specified.

The main differences between the two interfaces can be explained by the analysis of the functions they provide for the token NFT management and creation. Both ERC721 and ERC1155 standards in order to guarantee their intrinsic functionalities, they require to satisfy the properties of the interface ERC165[15][24]. This standard is relevant and essential in both cases since its role is to publish and detects what interface a smart contract implements. This give the possibility to the smart contract to adapt its way to interact with another one[23].

The ERC1155 is able to overcome the limitation of the previous standards ERC20 (fungible) and ERC721(not fungible) in token generation and transfer. In both old fungible and not fungible standard proposed by the ethereum community, each time we need to create a token we need also to deploy a new contract. With ERC721 we are able in fact to mint each time a unique token of the same type, instead with the new standard ERC1155 within the same contract is possible to generate multiple tokens which will contain its own meta-data and attribute. These features of the new standard give us the possibility to delete some useless redundant bytecode for each contract generation.

The functions offered by the interface express in a clear way the new features that the standard ERC1155 introduces. In particular we can see the standard functions of transfer and balance split in two types[24]:

- Single type refers to a variable amount of tokens of the same kind fungible or not fungible.
- Batch type is instead a set of a variable amount of token of different kind that can be either fungible and not fungible.

Another important difference is in the choice of the meta-data to be used for a given token. In the ERC721 standard we got both name and symbol in ERC1155 we do not have these references. These choices of development was taken since the symbol is not useful to identify a virtual asset and is prone to collision, the name instead was not used since the idea was to use the Metadata JSON as the definitive asset name and reduce duplication of

data. The JSON solution allows the developer to localize the name without storing each string on-chain otherwise there will be a lot of expense.

5.2.2 Implementation of ERC1155

```
//multiple different token
function transferFromBatch(address from, address to, uint256[] calldata ids, uint256[] calldata amounts) external returns (bool){
    safeBatchTransferFrom(from, to , ids, amounts, "");
}

//transfer of a total quantity of a given token id
function transferFrom(address from, address to, uint256 id, uint256 amounts) external returns (bool){
    safeTransferFrom(from, to, id, amounts,"");

}

function createToken(string memory _neffie, string memory coffie, uint quantity) external{
    uint256 tempId = 0;
    //string comparison anche così totCoffieSameNeffie[_neffie]
    //se già presente tra le neffie note
    if(neffieCollection[_neffie] == true){
        tempId = neffieToId[_neffie];
        totCoffieSameNeffie[_neffie] = totCoffieSameNeffie[_neffie] + 1;
        //mint token e aggiungilo a quelli presenti
        _mint(msg.sender, tempId , quantity, "");
    }else{
        incrementId ++;
        neffieToId[_neffie] = incrementId;
        tempId =  incrementId;
        neffieCollection[_neffie] = true;
        totCoffieSameNeffie[_neffie] = 0;
        //mint token nuovo
        _mint(msg.sender, tempId , quantity, "");
    }
    if(tempId == 0)
        emit Error("impossible id to mint");
}
```

Figure 5.5: ERC1155 implementation

In this new version of the smart contract, we need to manage some different features and characteristics that in ERC721 are managed in a different way. In the contract developed we can find the usage of the function Create Token to mint the new NFT. The parameters feed "createToken" in similar way to what we have in the ERC721 implementation. We have in both implementation the same two parameters "HashNeffie" and "HashCoffie" but in this case we got also as additional feature the quantity of the token to be minted. This function however is not only limited to the token creation but it has to create a new one according to certain condition, before the mint we need also to check if that type of token already exist. We need then to modify the total quantity for that specific token. In case the new token to be generated is an NFT that does not already exists we need to generate a new

id for that precise token and passing to _mint function an amount of token according to the quantity value specified as parameter of "CreateToken". The implementation of this check require the usage of a mapping variable between the HASH Neffie string and a boolean value which will indicate if that given string has already been used to produce a given Coffie or not. A different array implementation might be too expensive if we have many Hash Neffie. The best practice is always too avoid the usage of arrays when we have to deal with many elements.

Another functionality is the possibility to create multiple token at the same time. The parameters of the function are three:

- A set of string for each HASH of the NEFFIE
- A set of string for each hash of the folder
- An amount for each token to be created

This operation can be done essentially in two possible way. The first one is to passing directly by hand all the information as parameters to the function "createBatch". The other one is to store the value in a temporary structure and then once the user thinks they are ready to be mint he(she) will call "createBatch". This second approach is more easy to manage from the user point of view. It requires however a good analysis on the type of structure to use. We need to remember that store and modify the state of a contract has in fact a cost more or less feasible.

5.2.3 Rest Service Algorithm

The Rest architecture is the most suitable solution for our objective. This not only in pure term of computation, time and expense performance the best solution, in this way we are also able to better track all the process. The solution created in the prototype works well but the computation it is totally delegated to the docker image working on AWS. Since the usage of docker is only a possible approach to the problem is better to have a more scalable and maintainable solution for a more agile development of the next versions of the tool.

This will require a host in which we can store the algorithm, a new deployment environment for the computation, a small change on the blockchain development side and on the client interface. In the contract "CallTheAlgo"

we need to change the query since this time we do not need the oracle to perform a computation but only to return the correct result. The smart contract implementation defers in the provable query used, this time is in fact enough to have an oracle that certify and gives a proof of the result that is coming from the offchain world. This can be done by passing to the oracle the hash parameter with a POST/GET HTTP request. For the REST API implementation it was more suitable to move the algorithm implementation from python script to node.js. This environment gives the possibility to use express.js, a framework that offers to the developer many utility to manage HTTP request.

5.2.4 Rest API implementation

For the deployment of the application and the query of the result, differently from the previous AWS embedded environment of Provable, we decide to use firebase cloud technology. The motivation of this choice belongs to the necessity to find an environment that is not only able to perform the algorithm computation, but it also has to make available to the smart contract a public ip. This last requirement is fundamental to make the provable query feasible, since without it we will not be possible to perform the HTTP method invocation. Firebase allows the developer to use a servless framework that is able to running the javascript or typescript code inside a managed environment [2]. This process works as follows [2]:

- Deploy the project, after that the google service will be able to deal with the functions immediatly.
- The function will be fired as soon as the HTTP request will be received

For the development might be consider an implementation of a simple file YAML with inside the required HTTP methods in order to manage the "backend" mapping interaction with the result provided by the algorithm. However since our use case require only a GET method, we can possibly avoiding the usage of the API gateway, that instead might be useful for future implementation or when we have multiple sources to map. The file YAML is useful in backend API implementation since it represents the simplest way to give a documentation about the request and response of the application. With some specific editors such as swagger it is possible to

```yaml
# openapi2-functions.yaml
swagger: '2.0'
info:
  title: COFFIE Gen API
  description: API on firebase Cloud Functions backend serveless framework
  version: 1.0.0
host: us-central1-coffiegen.cloudfunctions.net
schemes:
  - https
produces:
  - application/json
paths:
  /app:
    get:
      summary: Generate the Coffie
      #operationId: hash_neffie
      parameters:
        - in: query
          name: var
          type: string
          required: true
          description: HASH folder
      x-google-backend:
        address: https://us-central1-coffiegen.cloudfunctions.net/app
      responses:
        '200':
          description: A successful response
          schema:
            type: string
        '400':
          description: Bad Request
          schema:
            type: string
        '500':
          description: Server Error
          schema:
            type: string
```

Figure 5.6: Yaml API Documentation

generate from the file yaml a document that describe the API. The user can read the documentation but also test the API if the project has already been deployed.

The Algorithm implementation has remain essentially the same of the previous one the only difference is that this time instead of returning the value, it might be more suitable for the new implementation to return a JSON as a string. This because it will be more manageable to get with provable the result we need for the NFT Coffie Creation. The query API provided by provable provides to the user a way to extrapolate the required information from a string JSON

5.3 User Experience

The application requires a simple interface representation useful to interact directly with the smart contract already deployed on a rinkeby testnet. In order to connect the interface to the testnet it was necessary to setup a provider that was able to link the local folder with the testnet. The provider used for this scope was INFURA.

The interface in order to work need to have some references and addresses:

- The MetaMask account address in order to refund the transaction with gas. This is fundamental also to provide the account private key necessary for automatic confirmation of each transaction.
- The address of the main contract already deployed.
- The project key of INFURA API provider in order to obtain the JSON RPC (in the project over HTTPS) to be linked to the node used.
- The ABI JSON of the contract developed, compiled and deployed.

In order to perform the calls to the smart contract connected to Metamask it is important to make a distinction between the transactions that require wei to be executed and the transactions that comes free after the main smart contract deployment. From the perspective of the interface all the calls are done to some functions. There is not a clear distinction between attributes and methods since with web3 they are called in the same way.

All the attributes that are public and view methods has no cost in term of gas and can be called any time it is required without signing explicitly the call to the transaction. This also works for methods assigned as view, since they do not require a change of state of the smart contract they also do not need any ether to be called. This kind of operation can be done by the simple usage of the call method on the function needed.

All the other methods instead needs transactions to be signed by the private key of the account to be executed automatically, without an explicit owner confirmation, on smart contract side. This because calling methods that change the state of the smart contract is a cost in terms of gas, since it requires an elaboration and some miners are needed in order to full fill the job.

To invoke this methods we needs first to create the transaction object.

```
const txObject = {
    from: account,
    nonce:    web3.utils.toHex(txCount),
    chainId: 4,
    gasLimit: web3.utils.toHex(10000000),
    gasPrice: web3.utils.toHex(web3.utils.toWei('10', 'gwei')),
    to: address,
    data: callTheAlgo.methods.readyTo().encodeABI()
}
const tx = new Tx(txObject,{'chain':'rinkeby'});
```

Figure 5.7: Transaction Object

In this entity we need to specify each of its components showed in the picture above. What define the transactions objects are:

- "Nonce" The number of transactions made by the sender prior to this one[9].
- "From" is the address of the account which execute the transaction.
- "Id" refers to the testnet used, in our case we have developed on Rinkeby so the id is 4.
- "To" is a field containing the address of the account that have been developed.
- "gasLimit" here we can setup a limit to the cost of the transaction, when the gas cost is too expensive the transaction will be reverted.
- "gasPrice" this is the gas fee for the transaction.
- "value" represents how many wei to transfer in the address specified by the field "to" of the transaction.
- "data" is the object to which we need to associate the encoded ABI of the function to be called.

After doing this we need to explicitly signed the txObject and use the function send, to launch the execution of the method on the smartcontract and the consequent publication of the transaction.

Between each transaction to let the tool work as expected it might be good

to set anyways a timeout between each transaction. The mining of the block is not immediate and might require few seconds.

In this interface we consider simply essentially two calls to the methods of the smart contract. The first guarantee the execution of the algorithm and the returned of the "CoffieUri". The second one gives a proof to the creation and publication of the NFT on the blockchain thanks to the transaction id. The transaction related to the NFT creation got a timeout set to a fix amount of time that represents an estimation of the maximum time required for the callback transaction and for the computation of the result to be executed and mined on smart contract side.

The Id of all the transactions can be used to see on Testnet the success of the transaction. On the rinkeby Tesnet are also available all the information concerning the name, the symbol and the id of the NFT that have been mined, including all the passages of account.

Chapter 6

Conclusions

This part represent a summary of the results obtained, with a description of the ratio between the cost in terms of gas and the trust.

The first simple proof-of concept of this project has been useful to provide a first solution that maps all the main characteristics of the processes, this fulfil the goal of creating a first application with all the components needed to a future deep and complex analysis. The aim of this first approach was only to create a work solution and the skeleton of the real application.

The performance in terms of gas, the time and trust performance are not take precisely into account. For the development on blockchain these features are very important since they they are fundamental to decide if it is really convenient to create a blockchain architecture over a given project. In terms of cost of gas there are many thing that can be taken into consideration such as the reduce of public variable only when required and also the reducing of the view function will decrease significantly the cost of the smart contract deployment. The performances are also dependent to the oracle and to the service the smart contract require, the choice of using a specific technology or another one can have a different impact on the final solution. The usage in the final realised of the ERC1155 API has given the opportunity to reduce significantly the gas cost of the smart contract. A significant result in time performance can be observed if we compare the initial implementation with the final one. Both solution has been developed with the usage of the provable oracle. The first one delegates to the oracle the entire computation of the algorithm, the second instead guarantees the passage of parameters to an external algorithm. In the first case the

time required can be approximated in a range between 2 and 3 minutes, the second one requires few seconds.

Bibliography

[1] https://remix.ethereum.org/.

[2] Cloud functions for firebase. https://firebase.google.com/docs. 30-10-2021.

[3] Connecting to public test networks. https://docs.openzeppelin.com/learn/connecting-to-public-test-networks.

[4] Eips. https://eips.ethereum.org/.

[5] Ipfs docs. https://docs.ipfs.io/concepts/. Last Updated: 22/06/2021.

[6] Merkle tree. https://en.wikipedia.org/wiki/Merkle_tree. last edited on 5 November 2021.

[7] Provable. http://docs.provable.xyz/. 28-08-2019.

[8] Provable ethereum. http://docs.provable.xyz/# ethereum. 28-08-2019.

[9] web3.js - ethereum javascript api. https://web3js.readthedocs.io/. 2016.

[10] Maher Alharby and Aad van Moorsel. Blockchain-based smart contracts : A systematic mapping study. pages 125–140, 2017.

[11] Andreas Antonopoulos and Gavin Wood. *Mastering Ethereum*. O'Reilly Media, 2019.

[12] Andreas M. Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, January 2015.

[13] Imran Bashir. *Mastering Blockchain*. Packt Publishing Ltd, August 2020.

[14] Jacob Eberhardt and Stefan Tai. On or off the blockchain? insights on off-chaining computation and data. pages 3–15, 2017.

[15] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. Eip-721: Non-fungible token standard. `https://eips.ethereum.org/EIPS/eip-721`, Ethereum Improvement Proposals, no. 721, January 2018. [Online serial]. Created 24 January 2018.

[16] Vitalik Buterin Fabian Vogelsteller. Eip-20: Token standard, thereum Improvement Proposals, no. 20, November 2015. [Online serial].

[17] Devin Finzer. The non-fungible token bible: Everything you need to know about nfts. https://opensea.io/blog/guides/non-fungible-tokens/. 10 January 2020.

[18] Jake Frankenfield. Double-spending. https://www.investopedia.com/terms/d/doublespending.asp. 30 June 2020.

[19] Pierluigi Freni, Enrico Ferro, and Enrico Ferro. Tokenization and blockchain tokens classification: a morphological framework. 2020.

[20] Gregory McCubbin. Intro to web3.js Â· ethereum blockchain developer crash course, 2021.

[21] Roman Mühlberger1, Stefan Bachhofner, Eduardo Castelló Ferrer, Claudio Di Ciccio, Ingo Weber, Maximilian Wöhrer5, and Uwe Zdun. Foundational oracle patterns: Connecting blockchain to the off-chain world. pages 35–51, 2020.

[22] P. Rajitha Nair and Dr. D. Ramya Dorai. Evaluation of performance and security of proof of work and proof of stake using blockchain. 2021.

[23] Christian Reitwießner, Nick Johnson, Fabian Vogelsteller, Konrad Feldmeier Jordi Baylina, and William Entriken. Eip-165: Standard interface detection. `https://eips.ethereum.org/EIPS/eip-165`, Ethereum Improvement Proposals, no. 165, January 2018. Created 23 January 2018.

[24] Witek Radomski, Andrew Cooke, Philippe Castonguay, and Ronan Sandford James Therien, Eric Binet. Eip-1155: Multi token stan-

dard. https://eips.ethereum.org/EIPS/eip-1155, Ethereum Improvement Proposals, no. 1155, June 2018. Created 17 June 2018.

[25] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. A taxonomy of blockchain-based systems for architecture design. 04 2017.

[26] Gavin Zheng, Longxiang Gao, Liqun Huang, and Jian Gua. *Ethereum Smart Contract Development in Solidity.* Springer Nature Singapore Pte Ltd, 2021.