DArt: Blockchain applied to Cultural Heritage

Claudio Di Ciccio[1], Giordano Dionisi[1] & Michele Spina[1]

[1] *La Sapienza University of Rome, Department of Computer Science, Faculty of Information Engineering, Computer Science and Statistics, Italy*

eMail / di.ciccio@di.uniroma1.it, dionisi.1834919@studenti.uniroma1.it & spina.1711821@studenti.uniroma1.it

Abstract / Blockchain is a modern technology, it can be described as an open distributed ledger, defined by a chain of blocks composing a connected DAG. DApp (Decentralized APplication) is an application operating on Blockchain system, through smart-contract's use, that run on blockchain using decentralized computing. DArt is a DApp System built on Ethereum Blockchain. DArt is an application with the aim to do the management everything related to real assets in the world of arts and cultural heritage. The purpose of DArt is to allow this according to the guidelines of the management of modern cultural heritage, and to help the structures respect the dictates of the international definition of museum. Every real work of art can be managed in DArt, tracing its possession, transfers, concessions, exhibitions and any type of update regarding restoration and other forms of activity for the maintenance of the asset. Different types of actors have different roles: museums, restoration laboratories, private collectors or common users. In order to make more attractive the participation on the system, a crowdfunding function has been introduced, thus helping the owners of works to find the funds to keep them.

Keywords / Blockchain, Distributed Ledger, Distributed Systems, Ethereum, Cultural Heritage

1. Preface

DArt is **Decentralised Application** to **Manage** any type of physical **Artwork.**
DArt manages **Restoration, Exhibition** and **Crowdfunding** of cultural heritage.
With dart **Everyone** can **Interact:** normal users and verified users: cultural organization and artist.

1.1. Main Responsabilities of Members

Responsabilities		
	Giordano	*Michele*
Context Analysis		X
Research/Definition Needs to Solve		X
Goal		X
Entities' Tokenization		X
Need of Blockchain	X	
Currency Exchange	X	
General Architecture Diagram	X	
Concept Diagram	X	
Component Diagram	X	
Collaboration Diagrams	X	
Use-Case Diagram	X	
Activity Diagrams		X
Implementation		X
Solidity Code Development		X
Graphic Design: HTML, CSS, BS	X	
Javascript Code Development		X
Background	X	
PPTX Presentation	X	X
Aesthetic Part: Logos	X	
Limitations and Issues	X	X
Future Developments		X
Conclusions	X	X

1.2. Report's Outline

The chapters of this report and their brief description are shown below:

- **Background of Blockchain:**
  An excursus of the topics covered during the course
- **Context:**
  An analysis of the context of the world of cultural heritage to understand the role and the importance of DArt in this
- **Goal:**
  What are the goal set during the development of the project and what are the applications of DArt in the real world.
- **Why Blockchain?:**
  Analysis of the decision-making process that led to the choice to use blockchain technology
- **Currency Exchange:**
  Formal definition of the token exchange rate idea
- **Software Architecture:**
  General Architecture and UML Diagrams (Concept Diagram, Component Diagram, Collaboration Diagram, Use Cases and Activity Diagrams)
- **Implementation:**
  Description of the development of smartcontracts and interfaces for interacting
- **Aesthetic Part:**
  Short presentation of DArt's logos
- **Limitations and Issues:**
  Analysis of the limits of the project with a look at future developments
- **Conclusions:**
  Final chapter

2. Background

This section would introduce some useful knowledge and notion to understand the paper and how DArt works. These knowledges are starting points of DArt Project.

2.1. Needed Knowledges

2.1.1. *Distributed Systems*

A distributed system is a computing environment of a collection of autonomous computer systems, physically separated, that communicate and coordinate actions in order to appear as a single coherent system to end-user.[7] Distributed System must be resistant to delays, crashes, failures and byzantine faults (when one or more participants act against the interest of the system)[21].

2.1.2. *CAP Theorem*

Cap Theorem: "Any distributed system can provide only two of the following three guarantees:"

- *Consistency:* Everyone reads the most recent data or receive an error;
- *Availability:* Every request receives a non-error response, without the guarantee that it contains the most recent write;
- *Partition tolerance:* System continues to operate despite an arbitrary number of messages being dropped or delayed by the network between nodes.

Usually in distributed systems we adopt availability and partition tolerance, discarding consistency.The same is valid for the blockchain.[16]

2.1.3. *Hashing*

Hash function is a function used to map a data (of arbitrary size) to fixed-size value. The input values are called keys and the outputs are called hash values (hashes). Two fundamental properties of an effective hashing are: **speed of calculation** and **low probability of collisions** (finding two different keys that generate the same output). Thanks to these properties, hashing can be used to generate pseudo-random numbers and we use hash functions for this purpose.

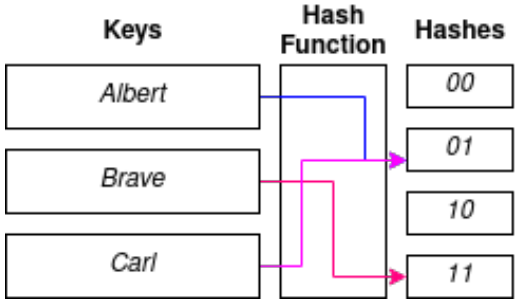


Figure 1: Hash Function

Given one key k_1 and one hash function h, it'ss possible to compute $v_{k_1} = h(k_1)$ with low computational cost, but given h and v_{k_1}, it isn't possible to reconstruct h^{-1}: Calculating k_1 has an high computational cost, because you need to hash every possible key k_i, until k_1 or colliding key is found. Hash Value v_{k_1} is a pseudo-random value, because it's generated by a deterministic function applied to a fixed key, but its value is hardly predictable.

2.2. Blockchain

Blockchain is an open and distributed ledger. It's a chain of blocks composing a connected and acyclic graph with only one leaf. Each block can contains many transaction that represents the information in the ledger. The transaction and the blockchain structure are immutable.

2.2.1. *Bitcoin*

Previously we have told what is a blockchain, that it is basically a mathematical structure.

On this simple concept many protocols have been developed, like as Bitcoin, to manage money, firstly, but then to manage transactions that need to be done and need to be untouched. The bitcoin is the most diffused application of blockchain technology. In fact it is the first most famous blockchain technologies.

Briefly we have that blocks are linked together using hash functions and we have that we pay using transactions. When a user has a transaction to publish, then he has to give it to a miner. It creates the block accumulating all these transactions and it has to hash all the entire block. The resulting hash needs to have with a certain amount of zeros at the end. If this doesn't happen, then the block isn't correct, otherwise it is exactly correct and it can be published. So we use the hash function only to avoid that all miners publish a lot of blocks together, but to have a temporization, practically.

Specifically Bitcoin is a protocol and to see how it works we see the following thing:

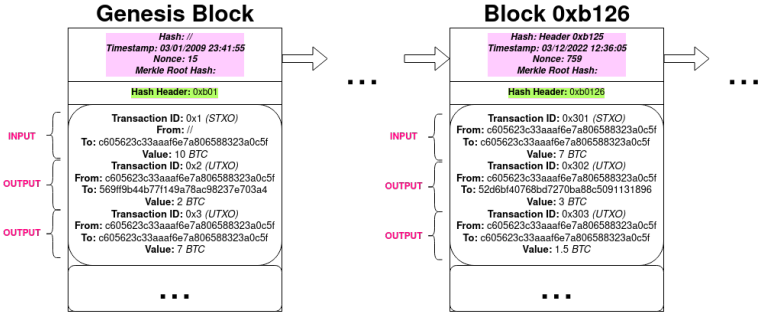


Figure 2: Costruttive Example

We can see that we have the following structure, where we have transactions that we use to spend money or not, namely each block has:

- *Header Part:* structured as following:
  - *Hash of Previous Block* to maintain the chain;
  - *Timestamp* of block: The timestamp for miner to understand when it's mined that specific block. Not really useful, because we are in a distributed environment and a global clock doesn't exist, but it's needed for the other nodes to understand if a block is really valid or not;
  - *Nonce:* It is the only parameter that the Miner can change to obtain the desired hash. It $\in \mathbb{N}$;
  - *Merkle Root Hash:* It's an hash of the list of transactions of the current block;
- *Hash Header:* We hash the header of the block;

- *Body of Block:*
  - For each transaction we have:
    - *Transaction ID:* Global identifier of it;
    - *From Field:* The sender of each transaction;
    - *To Field:* The receiver of each transaction;
    - *Value:* The value in *BTC* to transfer.

    These fields are for the one input and the two outputs of each transaction:

    (i) *Input* is the transaction that we spend (*STXO*), e.g.: banknote;
    (ii) *First Output* is the transaction to transfer to the receiver: e.g.: the banknote to give to a specific user;
    (iii) *Second Output* is the "rest": amount of money that they remain from the Input and First Output, it's a transaction that needs to be spent for the user (we split transactions);
    (iv) If we have:
    Second output > Input - Firts output
    Remaining part is for miner: commission/tip to execute our transaction.

All this is the basic principle of Bitcoin and the basic concept of the Proof of Work that it's based on proof by means of a computational effort.

To have faster computations, we work with ASICs: Circuits without memory, but only with logical ports that are well-designed to solve hashes functions.

To speed up the PoW (Proof-of-Work) we can distribute the work among all the nodes, namely we need to divide the nonce tried for all the nodes in a certain way to split the work to find the correct solution.

Every 2 weeks the system re-see the difficulty of the proof-of-work (if we have an hash target too simple to solve). This is done seeing the time elapsed between 2016 blocks and the system checks if this amount of time is really equal to two weeks, otherwise it changes the difficulty. In fact we have that we need to apply a nonce and to find a value X that: $X \leq N$, where N is the target value. For example:

$X \leq 0000011$ means that the found result needs to be smaller than this amount, so → Higher the number of zeros at left and higher the difficulty of the proof-of-work*. To make harder the proof-of-work we need to increase the amount of left zeros, otherwise we need to decrease the amount of left zeros, so.

2.2.2. *Ethereum 1.0 (Proof of Work)*

In this case we haven't anymore that the transactions can be spent like as Bitcoin, because we have a different concept: Account Model (or Balance Model) and not Transaction Model as Bitcoin.

We don't exchange transactions, so we don't spend transactions for each entity, but we have a real amount: so we exchange values.

In every node we have installed Ethereum Virtual Machine (EVM) to run the smart contracts (that we will see later). Each transaction is executed by each specific node and using the EVM we have a standardization of execution having always same environment for all executions, so we have a middle-layer to avoid compatibility's problems between different CPUs, Systems and so on. So EVM is useful to do operations in the Blockchain for Ethereum.

Each account is represented by means of a table: In each row we have stored the balance (in wei, minimum representation) of corresponding Ethereum account.

In Ethereum we associate a cost for each basic instruction that we can perform: in fact in Ethereum we deploy Smart Contracts, so pieces of code to execute things; this means that these Smart Contracts are codes, so instructions, and these instructions are translated, with the EVM, in a EVM byte-code → A cost is associated for each of these instructions and it is expressed in *gas*. This association never changes in time, respect to Ether, that it fluctuates according to trading. The gas price is needed, in fact every smart-contract needs to be executed by any node in the Ethereum context and we want pay the computation made by all these nodes. Let's suppose the following situation consisting in a Bob's transaction with:

(i) *Gas Price = 200 Wei* → Each gas unit corresponds to 200 Wei. If this amount is too low and so not convenient for the miner, then it can decide to ignore and to not consider it**
(ii) *Gas Limit = 12.000 Gas Units* → It is the maximum amount of gas that our code can consume for the specific transaction, so to avoid that we do a code that does an infinite loop, for e.g., and we lose all the balance for this reason. So it is a threshold over that the execution of smart contract will end, so we know the maximum amount of Ether that we need to spend, in worst case.

We can also have the following:

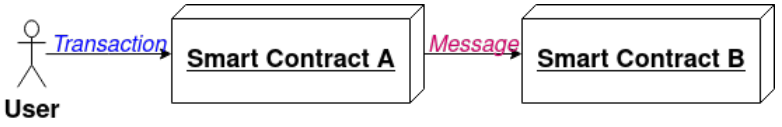


Figure 3: An Example for Gas Price

We can understand that the User calls the Smart contract A (doing a *Transaction*) and then this last one calls the Smart Contract B (with a *Message*): the costs to call the Smart Contract B are paid by the User at same way, not by the Smart Contract A. Then *Transaction* will be recorded in the ledger, instead the *Message* will not be recorded, because we can compute the result after the execution of it, because, theoretically, any node knows its code and so everything regarding of it.

Miner earns the gas regarding of its made execution and the remaining part (so the gas not consumed by the execution) comes back to user, in Ether (because user anticipates the whole amount), even if there were errors. It is the first node that executes our transaction (and verify it) and then he is the responsible to insert it in a block. For this we send the transaction to the miner.

*In fact it's simple to find a value smaller than 0011111 compared to find a value smaller than 0000011.

**For this first of all any transaction will be executed firstly in mining nodes, because they will include the transactions.

In Ethereum we have states: we pass from one state to another one each time that a transaction will be executed. If we have μ transactions in a block, then we have that each block has μ changes between transactions: for each transaction we pass from one state to another one, according to how many instructions will execute. In case of errors we roll back and the miner doesn't get the reward (only the specific fees).

The difficulty of the Proof-of-Work in Ethereum 1.0 increases every 100.000 blocks and this means that we will arrive that the difficulty to forge a block is too high that it will be needed a lot of time to do this forging: the validation of the blocks for miners becomes really too high, in time, and this will bring all the system to die. It will not more convenient to take the reward forging a block for the computational effort needed.

In Ethereum we have these rewards for miners:

(i) *System Reward:* It is given by the system for the work done to forge a new block;
(ii) *Fees:* They are given by the users: each transaction has it. They acquire priority for transactions;
(iii) *Ommers Reward:* This is a particular case:

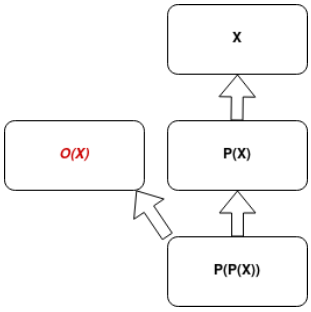


Figure 4: Ommers Reward

There is a fork in a certain moment (this can happen even in Bitcoin, when two different miners win the game more or less simultaneously). The fork that will die, because the majority will continue the other one, is rewarded. Higher reward if we are an Ommer of first generations, otherwise a lower reward. Only one time we can take the *Ommer Reward* for a specific block.

Types of nodes:

(i) *Full / Archival:* These nodes store everything of the blockchain: transactions, states, etc. With them we can see the actual state of the BlockChain, avoiding recomputations;
(ii) *Simple Nodes:* They do some checks and similars.

Types of chain:

(i) *Chain Data:* It contains all blocks (with regarded transactions) that entirely constitute the chain. The Chain Data is needed to see the correctness of hashes (to avoid changes);
(ii) *State Data:* It's a smaller version of Blockchain. It contains only all the results of each state transition of each transaction.

We don't treat with Halting Problem, because we pay to execute all transactions, so we will terminate to execute the code, because at certain moment we will finish the credit of the responsible of that code.

We have states for each account and we need to indicate, at least, the balance for each one.

In Ethereum it's also possible to do the pruning operation, namely we can cut the blockchain to erase some old data (old blocks) not more useful, because they cannot be more changed due to the time in the blockchain.

Differently than proof-of-work of Bitcoin, in this case we cannot use ASICs, because we have the Estash Algorithm. We need to store some DAGs (Directed Acyclic Graphs) to have heaviness and to make the game more fair, so to avoid to have that only ASICs with a big computational power can win, but we put a lot of effort also on the memory of the nodes, so that more nodes can compete to win the Proof-Of-Work challenge. ASICs are difficult to use.

The maximum amount of fees is given by:
gasLimit * gasPrice
For example: 21 000 units of gasLimit (Minimum amount of gas for each transaction)* 200 Gwei (gasPrice) to obtain the maximum amount of fees.

For Ethereum the Proof-of-Work difficulty increases always a lot and this brings the entire system to collpse until a certain moment, because it is not so feasible to manage all this → So we need to leave this concept, mainly in Ethereum where the difficulty increases exponentially with time. Proof-of-Work is bad because burns a lot of resources: Time, Energy and Computation.

2.2.3. *Ethereum 2.0 (Proof of Stake)*

It was proposed in 2021 with the London Upgrade. This new standard was inserted in the EIP (Ethereum Imrpovement Proposal). Either Proof-of-Work and either Proof-of-Stake work in same context. First difference than before: computation of fees to give to miner/validator/proposer, namely:

$$\tau = gasLimit * (baseFee + priorityFee) \quad (1)$$

- $\tau \rightarrow$ Total maximum amount of gas fees.
- *baseFee* → Computed automatically by the protocol according to the inflaction in the system: We burn the amount τ*baseFee to avoid inflaction. In fact we haven't a maximum amount of circulating Ether (as BitCoin), so burn operation is the only solution. But how does system compute the baseFee? Each block contains at most 30 millions of gasUnits (Units of Gas used for computational reasons), because we want avoid that only high-computational nodes can verify and use the Blockchain. Perfect amount of gasUnits is 15 millions, because the gasUnits represent the network utilization: If we have more than 50% (15 millions of GasUnits) then we increase the baseFee, because the network utilization is too high (many requests); otherwise we decrease it. The increase/decrease ratio, each time, is in [-12.5% - +12,5%], to avoid too high oscillations.
- *priorityFee* (informally "tip") times used gasUnits is the amount that the miner/validator earns: higher means that we solicitate more it to insert and validate/mine our specific transaction: higher value,

higher possibility that transaction will be inserted in Blockchain. It's given by the proposer of the block.

In Ethereum 2.0 we have:

(i) *Proposers* (also called Forgers): they propose a new block each time (so many transactions) and they check and propagate it in broadcast. Higher amount of reached nodes and higher possibility that it will be chosen like as the next one in Blockchain. The proposers are chosen according a pool and they are chosen time by time by the system in a pseudo-randomic way.

(ii) *Validators:* To be validator we need to put at stake 32 ETH (like BTC, it's called crypto-asset); if this amount becomes less than 16 ETH, we will exit from the possibility to stake to avoid high loses. The validator bet on which is the future block to insert in the blockchain: The block with the highest bet will be accepted, but each vote depends on the amount of Ether that each validator has in the stake. If we bet on the right block, then we take a reward***, otherwise nothing changes (usually validator bets on most chosen block). Validator is also responsible to check and to propagate the correct blocks.

We have many slots to bet for the following block: Time is divided in epochs: In each epoch we have 32 slots. The first slot of an epoch is called Checkpoint and the corresponding block is called EBB (Epoch Boundary Block). Each slot contains at most one chosen block. It's possible not reach consensus for a certain slot: We skip the round. We have 6 seconds to propose a new block**** and 6 seconds to have a new Commitee Set to decide the following block in the Blockchain (we will see later), so we have a fixed tempo between one block and the following.

For each slot we vote a pair: (Ghost Vote, FFG Vote), where Ghost Vote is the head, justified block, and FFG Vote (with finalized block) is a pair: (checkpoint node of our epoch, checkpoint node of last epoch), needed to maintain consistency).

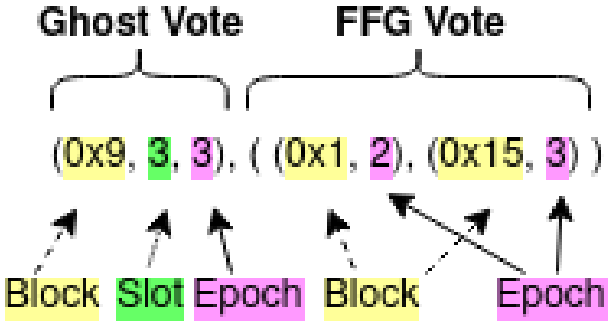


Figure 5: Ghost / FFG Vote

Validator has voted for the block 0x9 in third slot, third epoch. FFG Vote consists in last EBB and the current EBB for Checkpoints.

The exceding part of Ethereum in stake goes directly on the wallet of the validator; The types of balance are:

(a) *Effective Balance* χ*:* What we put at stake, always integer value;

(b) *Real Balance* σ*:* What we really have at stake, even real value.

Theoretically $\chi = \sigma$, but we have a gap:

(a) If $\sigma \in [X, X+1]$, $(X \in \mathbb{N})$ then $\chi = X$. Ex: $\sigma = 25.7$ ETH (we will omit "ETH"), $\chi = 25$;

(b) If $\sigma = \chi = X$, to have $\chi = X+1$, we need to have $\sigma = X + 1.25$. Ex: $\sigma = \chi = 25$: When $\sigma \geq 26.25$ then $\chi = 26$;

(c) If $\chi = X$, then $\chi = X - 1$ when $\sigma \leq X - 0.25$. Ex: $\chi = 25$; when $\sigma <= 24.75$ then $\chi = 24$.

Hysteresis Zones are zones where σ can change without affecting χ.

For these three properties we have a stateful structure (last state is important). We have:

$$Validator\_Effectiveness = \frac{\chi}{\sigma} \tag{2}$$

More higher than 1 is better; LMD-Ghost Algorithm is regarded of management of slots / epochs: so it is used to understand which block we need to select according votes.

We can have forks on the chain but we have in the future the straight chain. Differing that PoW, where we take the longest chain (more trusted), in PoS we take the most voted chain.

The validator could get an huge amount of money, but it needs a good hardware, run PCs all time, lot of bandwidth, etc, because if we are AFAIK, then we have many slashings and penalties.

(iii) *Normal Nodes:* Normal users that want use the services of Blockchain;

(iv) *Committee Set:* It is composed by 127 members + proposer for that specific block and it is established for each slot to decide the new block; A specific Validator can do this role only for one slot for each epoch, so we need at least 4096 validators in the system → So we can propose only one block for each epoch, to avoid to be validators more than one time in the same epoch.

(v) *Sync Committee:* they validate blocks done until certain moment forever, so to set truth. They are grouped to understand which is the final node to understand total correctness of blocks inserted. It is composed by 512 members randomly. Every 256 epochs we have this set and in this case these nodes need to sign (precisely add informations, because we know who are them) the last checkpoint block and they finalize all blocks inside each epoch. Two concepts to understand: justification and finalization given by validator:

- *Justification:* A block is Justified if the Committee Set has accepted it. To do it we need a supermajority, more than 2/3 of votes of validators.
- *Finalization:* A Block is Finalized if the Committee Set has Justified it and if it is linked to a following justified block. Between Epochs we need finalization: we need to finalize the EBB (they permanently tell truth).

*** *A block with a lot of bets has high possibility to be chosen, choosing it means a lower reward.*

**** *Even to check all in a geographic distributed network.*

All this is managed by the Gasper Algorithm.

It's possible to cheat in Ethereum, but it is expensive → In this case the system does slashing (slowly all the amount in the stake will be eroded, higher the times of cheating and higher the erosion, it cuts the total amount of stake) and penalties. We have slashing/penalties if:

(i) *Validator Do Nothing* for some time;
(ii) *Equivocating:* A proposer proposes two different blocks in the same slot;
(iii) *Double Vote:* We have many possibilities:

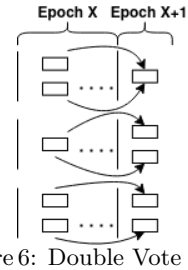


Figure 6: Double Vote Situation

It occours when we don't vote for only one source and one different target;
(iv) *Itself Vote:* The source and target are the same element;
(v) *Sandwich Behaviour* (also *Sorround Vote*);

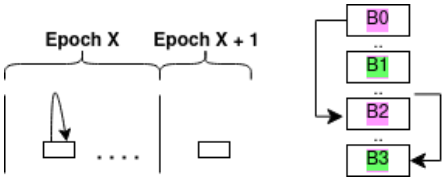


Figure 7: Itself vs Sandwich Situation

We need to have the same last Checkpoint, in this case we have slashing.
(vi) *2/3 Validators Vote Twice:* All their stake is eliminated, so an hard slashing; They do it to cause problems to network, but it's too expensive (system will burn around 4 millions of Ether to these validators).

For Ethereum 2.0 we cannot have more than 1/3 of byzantine nodes, otherwise the system is not more safe.
(vii) *More Than 4 Rounds* (e.g. epochs) without reaching a decision, then we slash money from all validators to come back to a legal situation.

The Casper Algorithm manages penalties, slashings, propagations, verifications, validators, etc.

For Ethereum Protocol we can have:

- *Soft Fork:* We create a fork with suggestions to adapt the protocol → we haven't a total split between the two forks;
- *Hard Fork:* We create a fork with big changes with the other one → Total different behaviour: a split between them.

In Ethereum 2.0 we have another type of Reward, the *Whistleblowing Reward:* The Whisleblowings are spies, they see bad behaviours of users and they take a reward if they spot some problems. They are not mandatorily validators (everything is public, even the messages on each committee set: they spot not correct behaviours and the other nodes can verify their assertion easily).

Paris Upgrade happened in 15-th September 2022. With it we have totally merged PoW and PoS, so we haven't anymore PoW and everything remains only on the PoS concept, replacing the PoW chain. We haven't anymore miners, but validators and proposers.

In the future we will not have only one main chain like as Ethereum (called Beacon Chain), we will want have roll-up: lot of shards managed in all the world.

2.3. Smart Contract

It was introduced with Ethereum: it is an application released with a transaction and it'ss needed to do very useful operation, like as an application, but in decentralized way using Blockchain.

The written code is translated in EVM (Ethereum Virtual Machine) Bytecode: we associate each specific cost to each specific instruction to have an associated payment in fees. Higher lines of code and higher costs: we have a deployment cost and also execution cost.

It manages tokens, that they can be:

- *Fungible:* They can be exchanged between them: each one has a corresponding value;
- *Semi-Fungible:* Middle-way between fungibility and non-fungibility: it considers also time (Ex: Stadium Tickets, same value before a football game, after the match they lose/acquire value);
- *Non Fungible:* No duplicates, by default: we cannot cut them or exchange with others.

Functions of Tokens: **Currency:** Tokens used as exchange; **Commodity:** tokens used as tradeable and linked with many things; **Utility:** Tokens are guarantees; **Security:** Tokens as financial instruments;

When we deploy a smart-contract we put the compiled code, not the source one, automatically, to run it. We can also decide to put the source one: Decision of the author.

The nonce for a smart contract is the number of its instances (first call = zero one); nonce for EOA ((Externally Owned Account: user / program) is the number of transactions run by it.

Smart-Contracts aren't signed and they can be invoked: **Locally** (simulating it without costs) or **With Transaction** (in global blockchain with costs and without modifications)

2.4. Web 3.0

Blockchain and Internet have an high correlation: Blockchain Technology is interesting for Internet:

(i) *Web 1.0:* Internet had clients that do requests to servers (client-server model) → Computational effort for nodes was low compared to computational effort for servers;
(ii) *Web 2.0:* Higher computational effort for local machines (high power for nodes);
(iii) *Web 3.0:* Merge between Blockchain topic and Internet topic → In blockchain we store only important things, out-of-chain we have big files. Blockchains aren't strictly needed, but they are useful.

3. Context

This chapter describes the context of application of DArt, the developed dapp and the subject of this paper, i.e. the world of cultural heritage management and its digitization, in order to better understand the needs and potentials.

3.1. Evolution of management of Cultural Heritage

The idea to take care of a state's artistic heritage (that it's a completely contemporary goal) it is often very widespread, but this does not correspond to the historical truth. Humanity has manifested interest on management of cultural heritage since ancient age, and since the XV century the pope has designated a specific cardinal for the only purpose of managing the registers of works of art belonging to the Papal State to verify their movements and status, with the title of cardinal chamberlain.[34] Our approach to cultural assets has totally changed in the last centuries and still today academics and art critics discuss about it. An Example: only in 2019 many academics have raised criticisms about the over-stress of Vitruvian Man, involves by transportation and continuous exhibitions in different museums and galleries that compromise irreversibly its ink.

The idea of how an artwork should be made available and managed is constantly evolving and presents differences from culture to culture even today in which we live in a highly globalized world, despite being much closer to an international standard. The collaboration from different countries and the cultural Exchange have permitted management of cultural heritage to involve, fueling the debate, and in many case saving very important cultural Heritage from irreversible damage or even destruction.

Also the restoration techniques are continuously evolving, using new chemical products and observing their effects in the time. keeping track of which compounds have been applied to a work, how long and in which conditions it has been preserved, such as humidity, temperature and lighting, are fundamental elements for understanding the behavior of these products and progressing in development of better restoration techniques.

3.2. Third parts controllers

The system that regulates and controls the management of cultural heritage Is developed on three levels. UNESCO - United Nations Educational, Scientific and Cultural Organization, established in Paris on 4 November 1946 - acts as an international guarantor and organizer of conferences through which the member States stipulate pacts and conventions. The World Heritage Convention, dating back to 1972, is in fact the first international instrument that defines the notions of protection and preservation of cultural heritage.[40]

Although UNESCO's domain is very extensive, its politicized nature means that today even some large countries do not partecipate, like the US and Israel withdrew in 2019.[4]

At the second level we see, in Europe's case, the European Union which protects and preserves the cultural heritage of the member states through initiatives and incentives. Indeed, the Council of the European Union adopts measures and recommendations to encourage the action of the Member States, which are responsible for their own cultural policy matters. It is in fact the single State, at the third and last level, which must enact legislation, in compliance with international treaties.

We than see external bodies which, due to their international prestige, act as controllers and promoters of the protection and enhancement of cultural heritage. Among the most important we see ICCROM, an intergovernmental organization that works at the service of its member states to promote the conservation of all forms of cultural heritage, in every region of the world and ICOM, the International Council of Museums, or the main international non-governmental organization representing museums and their professionals. The organization assists the museum community in preserving, conserving and sharing present and future cultural heritage, tangible and intangible.[2]

3.3. Digitization in the World of Art

In the last decade, digitization in the world of culture has become an important subject of study and research, developing fields such as digital museology and digital literacy to make cultural heritage more accessible to anyone. In addition to giving us the ability to remotely access entire libraries or the ability to digitally view an art gallery, technology is also revolutionizing the way cultural heritage is archived by digitizing what were once huge physical paper ledger around the world, subjected to errors, wear and tear and difficulty in consulting. In particular, in recent years there have been questions about how the blockchain can impact on the management of the registers of cultural heritage, so much so that already today it is possible to find several papers on this topic online.

4. Goal

The goal of the project is to build a public ledger using the blockchain to take trace about the management of each cultural heritage. Contrary to many modern projects, DArt does not propose itself as a digitization of artistic material, but as a tool for managing it in the real world. The goal of DArt is not to impose a digital point of view on art, but to put modern technologies at the service of humanistic and artistic culture for the management of cultural heritage according to what is modern ethic. The intent of DArt is to constitute a support element for actors in the art world to migrate towards the international modern definition of museum:

> A museum is a not-for-profit, permanent institution in the service of society that researches, collects, conserves, interprets and exhibits tangible and intangible heritage. Open to the public,

accessible and inclusive, museums foster diversity and sustainability. They operate and communicate ethically, professionally and with the participation of communities, offering varied experiences for education, enjoyment, reflection and knowledge sharing - *ICOM, August 2022*[27][18]

4.1. Actors

The DArt's goal is to be a verified source for any person to analyze and study the management of art in world, looking at museum's practice, The information in the ledger are written only by verified source like museum, art gallery, private collectors and restoration groups. Each person can read these information and look at the source. On the ledger will be possible to analyze information about any artwork in the chain, and also look at any action of the actors. Each user will also have the opportunity to be a patron of any museum through donations bound by smart contracts.

4.2. Applications and uses

The applications of DArt in the management of cultural heritage can be many, in this section some of them are proposed in order to understand the possible impact of DArt on cultural heritage world.

- **Protection**: It consists of any activity aimed at preventing, protecting, maintaining and restoring the assets that make up our cultural heritage with the aim of being able to be publicly enjoyed by the community.[31] DArt will allow to keep track on the blockchain of all the protection activities carried out on an artwork and their key information.
  - **Preventing**: Any action to remove a possibility of risk.
  - **Protecting**: Limitation of risk situations connected to the cultural property in its context
  - **Maintaining**: Coherent, coordinated and planned study, prevention and maintenance activities. Contrary to restoration, maintenance is a preventive action, and tends not to affect the object. Part of this process is monitoring stress level on a work of art.
  - **Restoring**: Dart allows you to keep track of all the restoration works carried out over time, allowing you to verify the good work of the managers and in the case of future restorations, to know which chemicals have already been applied on the work, in order to avoid unwanted reactions.
- **Exhibitions**: Using DArt it will be possible to keep track of where a work has been exhibited, but at the same time know the history of museums and art galleries, thus giving a tool for a critical analysis of them and to measure the overexposure and underexposure of the artists, for example knowing which galleries host the most exhibitions that give more space to female or non-Western artists.
- **Forgery**: Having a trace of the status over time of a work of art can help make its falsification more and more complicated.
- **Donations and Patrons:** Users have the possibility to make restricted donations to the verified actors, tying their use to the protection of a specific artwork. Donor users will be able to be certified through an NFT and obtain the advantages as patrons, established by the institution receiving the donation.

5. Why Blockchain?

5.1. Do We Need Really a Blockchain or We Can Avoid It?

We need to understand how we can implement and develop our project. In the specific we need to understand if it is needed a Blockchain or not for our project and to see this aspect we can use the following diagram:

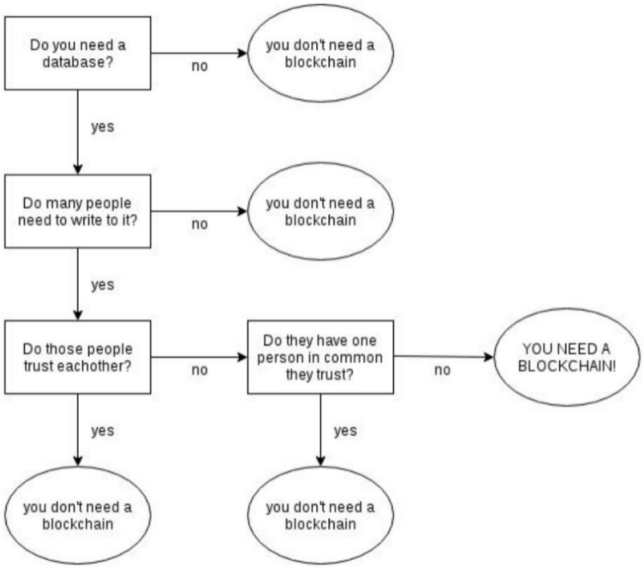


Figure 8: Blockchain or Not?

We need a blockchain because we have the need to store a lot of data (for artworks, museums, users and so on) and also because we have a lot of different entities that write on it, for example the museum that wants to insert an artwork in an exhibition, but also the restorer that want insert a new activity of restoration. For sure there isn't a third-part controller that does the role of guarantee (entity for trustability) and we haven't either that ALL entities can trust together, mainly for the private collectors or private galleries that can tell that they have some artworks that this isn't true → So for sure we need a blockchain, this is the first step.

5.2. But Which Type of Blockchain?

After the previous diagram we can see also the following one to understand, in the specific, which type of blockchain we really need:

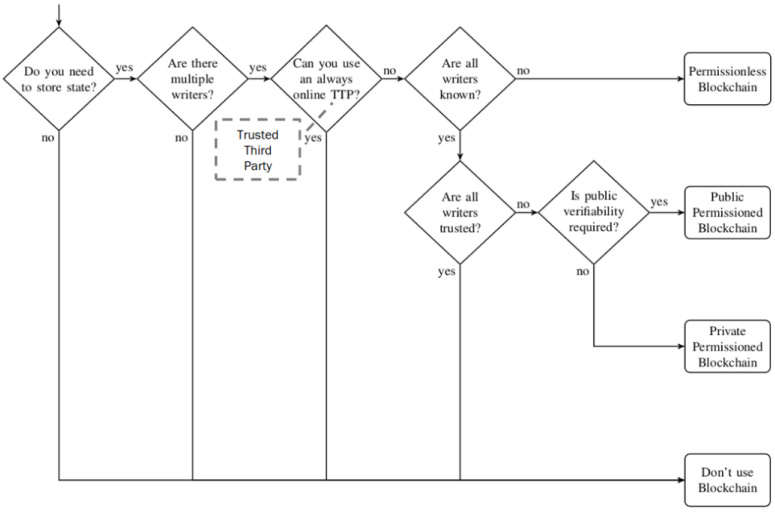


Figure 9: Which Type of Blockchain?

We need a Public Permisioned Blockchain, because, other than what has been said before, we have also that all the writers on the blockchain are known, because only the registered entities can write (museums, private galleries, artists, private collectors and restorers) and the manager of this system accepts or not these requests, so obviously only known writers we have, but not all trusted entities, because we have also small entities like as private collectors that could not be really a lot trustable. Obviously it's required a public veriafiability and all the entities in the entire world can check that everything is going well and everyone can really check the correctness of all the things.

This bring us an huge problem, in fact we want to work with a public permissionless blockchain, namely we want deploy a system (made with smart-contracts) in the Ethereum Context. To do all this we have thought about a very interesting thing, namely we can emulate the permissions needed by our application by means of the use of limitations given by smart-contracts, so using their controls, practically.

6. Currency Exchange

We need to understand how much we need to spend to have a certain amount of DCoins. This isn't a simple question and it needs an hard talk:

First of all we want avoid inflaction in our system → namely we want that higher the amount of DCoins in the system and higher the cost to buy each DCoin, so this totally excludes a constant change ratio between Ether and DCoin.

The ideal ratio, that we want, can be represented by means of the inverse of logarithm, namely:

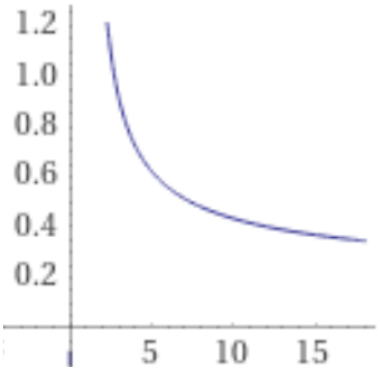


Figure 10: Mathematical Function - Inverse of Logarithm

For Exchange Rate we want have for the exchange rate is something like this:

$$\ln(\frac{x}{+1}) \tag{3}$$

If $\phi^{\dagger} = 0$ then: 1 Wei $= \frac{1}{\ln(\frac{6}{5})}$ DCoins $\rightarrow$ 5.48 DCoins
If $\phi = 1$ then: 1 Wei $= \frac{1}{\ln(\frac{3}{2})}$ Dcoins $\rightarrow$ 2.46 DCoins
If $\phi = 2$ then: 1 Wei $= \frac{1}{\ln(2)}$ Dcoins $\rightarrow$ 1.44 DCoins
If $\phi = 3$ then: 1 Wei $= \frac{1}{\ln(3)}$ Dcoin $\rightarrow 0.91 DCoin$
If $\phi = 4$ then: 1 Wei $= \frac{1}{\ln(4)}$ Dcoin $\rightarrow 0.72 DCoin$

This exchange ratio is very useful because with the increasing of ϕ, then DCoin acquires value, in fact 1 Wei takes less DCoins with this increasing.

This solution has two problems:

(i) The value of DCoins can be real values, not only integer ones → This problematic can be managed;
(ii) We need to do one transaction per time or to do a lot of computations if the user wants to buy an huge amount of DCoins, because we need to compute the amount of DCoin with the constant updating of ϕ and obviously this is really a lot expensive. There isn't any known result to sum up logarithms in a mathematical way, so we cannot do anything better.

We can fix the exchange ratio with a constant value or we can assume a simple value at the end, the future exchange ratio, but we are subjected to scams or cheats or, simply, problems. In fact if we assume to buy at the initial value small value $\upsilon \rightarrow$ Then we have that the user can sell one DCoin per time and he can earn more money than what it has spent, so an huge problem for us. Example to clarify:

Bob spends 2 Wei and he will obtain 5.48*2 = 11.96 DCoins, where 5.48 is the Exchange Ratio of Wei-DCoin when $\phi = 0$; Then if he 11 DCoins selling one per time, he will obtain:

$$\sum_{i=1}^{11} ln(i) = 17,502307846\ Wei$$

But we still have 0.96 DCoin and we have obtained more than 17 Wei, spending only 2 Wei !!

To simplify the exchange system we have decided to replace the formula for calculating the price of the nth

† *ϕ is the total amount of circulating DCoins in the system.*

token in $1/n$. So what we have thought is that we need to exploit a known summatory: a solution is the Gaussian Sum, namely:

$$\sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$$

The use of this summation allows a change of the cost of the computation function from $o(n)$ to $o(1)$.
We want that the ratio $\frac{Wei}{Dcoin}$ decreases with the increasing of ϕ, in an inverse linear way (this means that the ratio $\frac{DCoin}{Wei}$ needs to be crescent in a linear way).

Many possible questions:

(i) *If there are in the system k DCoins and we want buy j DCoins, then which is the amount of Wei that we need to spend?*

Simple:

$$\sum_{i=k+1}^{n} i =$$

$$\sum_{i=1}^{k+j} i - \sum_{i=1}^{k} i = \frac{(k+j)*(k+j+1)}{2} - \frac{k*(k+1)}{2}$$

This is the total amount of Wei to spend and we have that all possible changes from the current ϕ to the future ϕ' [‡]

EX: If we have in the system 5 DCoins and we want buy 4 DCoins, then we need spend:

$$\sum_{i=1}^{9} i - \sum_{i=1}^{5} i = \frac{9*10}{2} - \frac{5*6}{2} = 45 - 15 = 30\ Wei$$

That it's higher compared to buy the same 4 DCoins at the begin, when $\phi = 0 \rightarrow$ In fact at the begin (with $\phi = 0$) the cost of 4 DCoins was: 1+2+3+4 = 10 Wei.

(ii) *If i pay 30 wei and in the system there are 5 DCoins, then how many DCoins i will obtain?*

To answer to this question we need to use the inverse of: $\frac{n*(n+1)}{2}$ that it is:

$$\frac{n*(n+1)}{2}^{-1} = \frac{-1+\sqrt{(1+8DC)}}{2}$$

More in detail we need to have precisely:

$$\frac{1+\sqrt{1+8*(\Omega+\frac{\phi}{\phi+1})}}{2} - k$$

Where Ω is the amount of Wei that we want spend.

EX: If we want spend 30 Wei and in the system there are 5 circulating DCoins, then we have:

$$\frac{1+\sqrt{1+8*(30+\frac{5*(5+1}{2})}}{2} - 5 =$$

$$\frac{1+\sqrt{1+8*(30+15)}}{2} - 5 = \frac{1+\sqrt{1+8*(45)}}{2} - 5$$

$$\frac{1+\sqrt{1+360}}{2} - 5 = \frac{1+\sqrt{361}}{2} - 5 = \frac{1+19}{2} - 5 =$$

$$10 - 5 = 5$$

This final amount is the total amount of DCoin that we will obtain. We always obtain an amount that it is integer, at most the lower part of a real part.

(iii) All this is very important, because now we can do also the opposite, namely we can answer to the following question: *If there are in the system n DCoins and we want sell j DCoins, then how many Wei we need to have?*

$$\sum_{i=n-j+1}^{n} i =$$

$$\sum_{i=1}^{n} i - \sum_{i=1}^{n-j} i = \frac{n*(n+1)}{2} - \frac{(n-j)*(n-j+1)}{2}$$

This is the amount of Wei to have.

EX: If there are in the system 9 DCoins and we want sell 4 DCoins, then we need to have:

$$\sum_{i=1}^{9} i - \sum_{i=1}^{5} i = \frac{9*10}{2} - \frac{5*6}{2}$$

7. Software Architecture

In the **Software Architecture Part** we have the need to understand *How Design* and *How See* the *Environment* and the *System* for an *Abstract Point of View:* It's really a lot important because if we well understand the whole system in this way, then the implementation of it becomes really a lot easier and, then, everything becomes more simple to understand, for all possible point of views.

7.1. General Architecture

First of all to describe our system we seriously need to describe the general architecture:

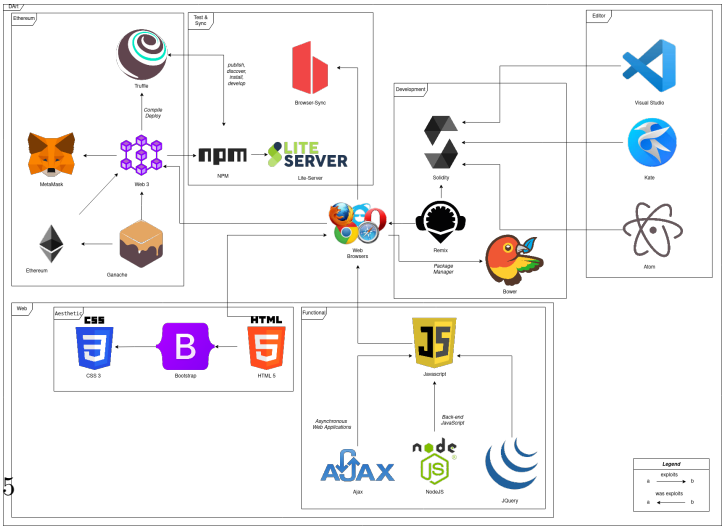


Figure 11: General System Architecture

With this diagram our purpose is to describe all the tools used for the development of our application. You can see that there is an huge amount of things to explain.

[‡] ϕ' *is the future state of circulating DCoins.*

Starting from the external part We have 5 macro-areas to analyze:

(i) **Web Area:** This area concerns all the things regarding of the tools used to do and to realize our web-pages and the entire web-site for:
   (a) The *Aesthetic Part:* how write things, composition of web-pages, style, positions of elements and everything that concerns only the aspect of it;
   (b) The more *Functional Part:* how forms communicate with the smart-contracts, how store the informations inserted by the users, all the part regarding of the asynchronous things that we have, the dynamic appearance of the web-pages according to the inputs that we give them, how can change the states of the smart-contracts and, so, all the things regarding of only the functionalities of all the system that we want provide.

(ii) **Editor Area:** This is the part regarding of all the tools used by us to write whatever possible code present in DArt: Solidity code, HTML code, Javascript development, but also all the implementations with Bootstrap, CSS, JQuery and, so, all the possible languages to make real the abstract functions of our system, in a well-designed web-site;

(iii) **Development Area:** This is the area regarding of the development of the code for smart-contracts: In fact we have Solidity (the programming language to describe everything about smart-contracts in Ethereum), Remix that uses Solidity to compile and also to run and deploy, virtually, the smart-contracts to have a good testing for them, so obviously Remix is a more complex tool than Kate, Visual Studio or Atom, because it is also needed to retrieve some informations (like the ABi) and to do some operations that the other tools cannot provide in the Blockchain context;

(iv) **Test & Sync Area:** This is a very important area: The tools in this area are important because with them we can publish, deploy, compile and synchronize all our web-pages when modifications occur, but with them it's also possible to test all our entire system in a "safe-environment" to spot possible problems to understand before the final releasing of the entire system in the Ethereum Context (so in a private permissionless Blockchain);

(v) **Ethereum Area:** This is the last real field that we have and it is needed to interact with a real or fake Blockchain with all the possible services that it can provide: This first part is made by means of the use of Truffle. Instead the Private Permissionless Blockchain is provided another too much used tools that it is Ganache, where we have some fake users and an entire well-designed Blockchain, to see all the possible functionalities and how really everything works well in a serious environment. Another very used tool in this Web3 environment is MetaMask that it is extremely needed because with it we can have a Wallet, we can have more naturality using the Blockchain tools and to test everything and with it we have a continously control of all the possible transactions that we are doing and how we are doing them, so it's practically mandatory. Finally we have the Ethereum Concept, where everything works well, so thanks to it we have all the stuff and all the possible concepts to have a very good use of smart-contracts for our system (so it is the general context, the super-context, where everything really works).

All these entities are linked by means of relationships regarding of who uses ("exploits") another one or not, like you can see with the legend of the diagram. It's important understand the relationships to know how they interact with them and which tool exploits the other ones, so for this reason we need to exploit a lot them

7.2. Concept Diagram

in Fig. 12 you can see a very huge diagram. This is very very important because it describes all the four contracts of our system and specifically the relationships between them, in fact we can see that DCoin and DArt uses all the other smart-contracts, instead the Patron one uses functions and utilities of DCoin and DArt. the only one that communicate with only one contract is the Verification one that exploits only the DCoin smart-contract.

Another important thing that you can see from this diagram is what each smart-contract contains, in terms of functions and attributes (even complex attributes like as structs or enumerations): for this reason we have chosen to adopt a very huge representation of it, because otherwise is really too difficult to focus our attention on the many things that they are present.

We have to understand that there is the use of only simple structures to memorize informations, to avoid to save in Blockchain things really too difficult structures or really too expensive, like as the strings that we need seriously to avoid because otherwise we fill a lot a block to be stored in the Blockchain and then we need to spend a lot of money to run everything in this field. We need also understand that we have done a very huge use of hash functions, to reduce a lot the dimension of the structures and of the inputs that we have, mainly for the strings that the user can provide like as input, for example the name of an artwork, name of an exhibition, their name and everything that consists to provide strings to be saved in the Blockchain structure.

The diagram is really intuitive to be read, because each smart-contract is represented by means of a class with a part reserved for the attributes and another part reserved for the methods. You know that each possible attribute / method has 4 possible modifiers, namely: public, private, internal (that it is the default one) and external. Now the thing is that we also have a lot of other modifiers: payable, view, pure and so on. With this Concept Diagram we have resumed all these modifiers at the begin of each attribute / method, showing also which are the types of the attributes and which are the returned values of the methods and the formal parameter for each of them (so the entire signature) to easily spot what we need to call and how to use them.

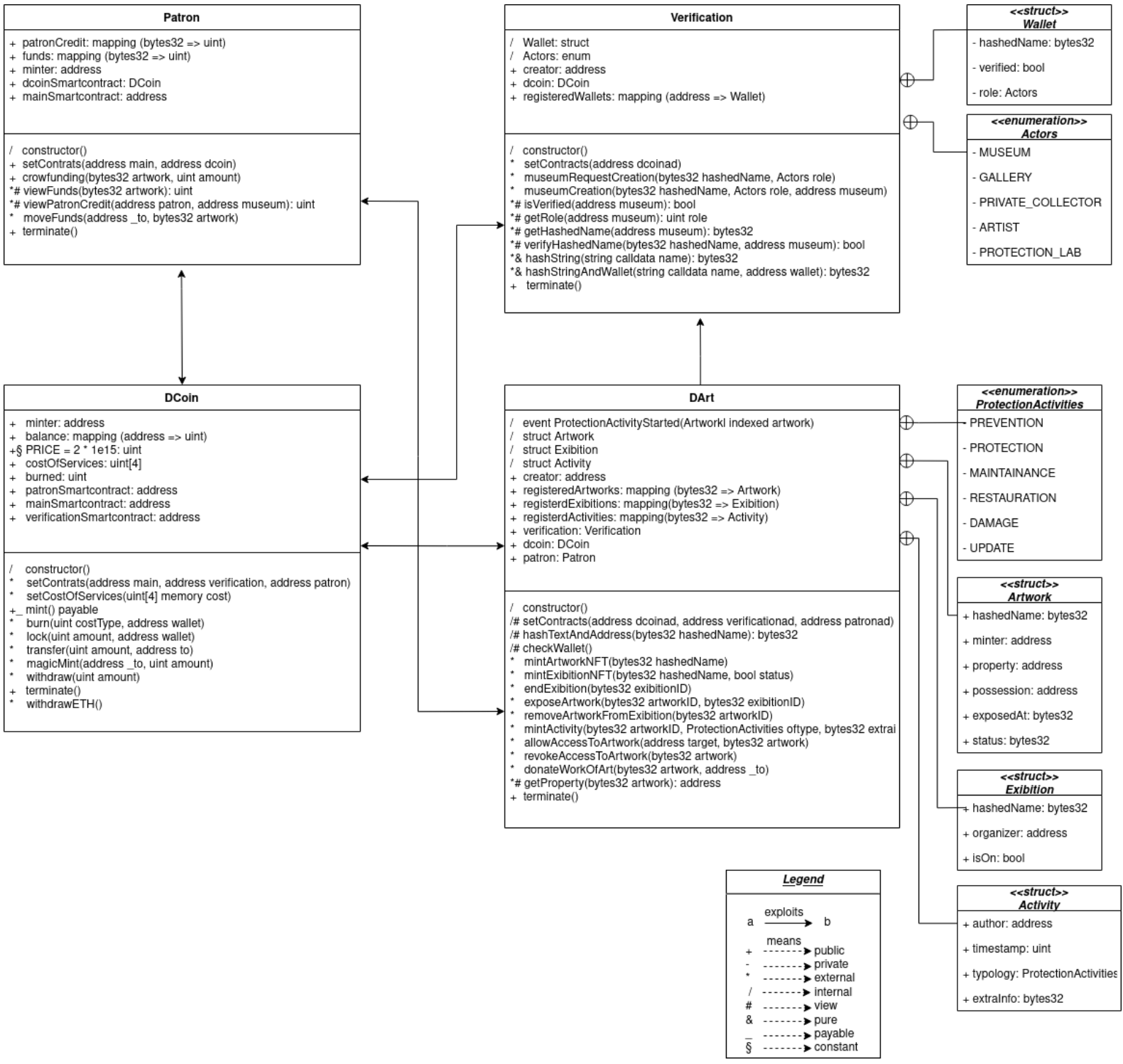


Figure 12: Concept Diagram

The last important thing to focus for this diagram is the use of complex attributes: structs and enumerations. In fact both of them are linked to the appropriate smart-contract's attribute field with a special arrow; for both of them we precisely indicate which are the contained fields, the possible types and all the many possibilities to understand their usage.

Just to conclude with this diagram our purpose is to resume with a figure all the possible functionalities that a smart-contract can do and all the possible interactions that are possible, so one of the most complete diagram that we can have is exactly this one.

7.3. *Component Diagram*

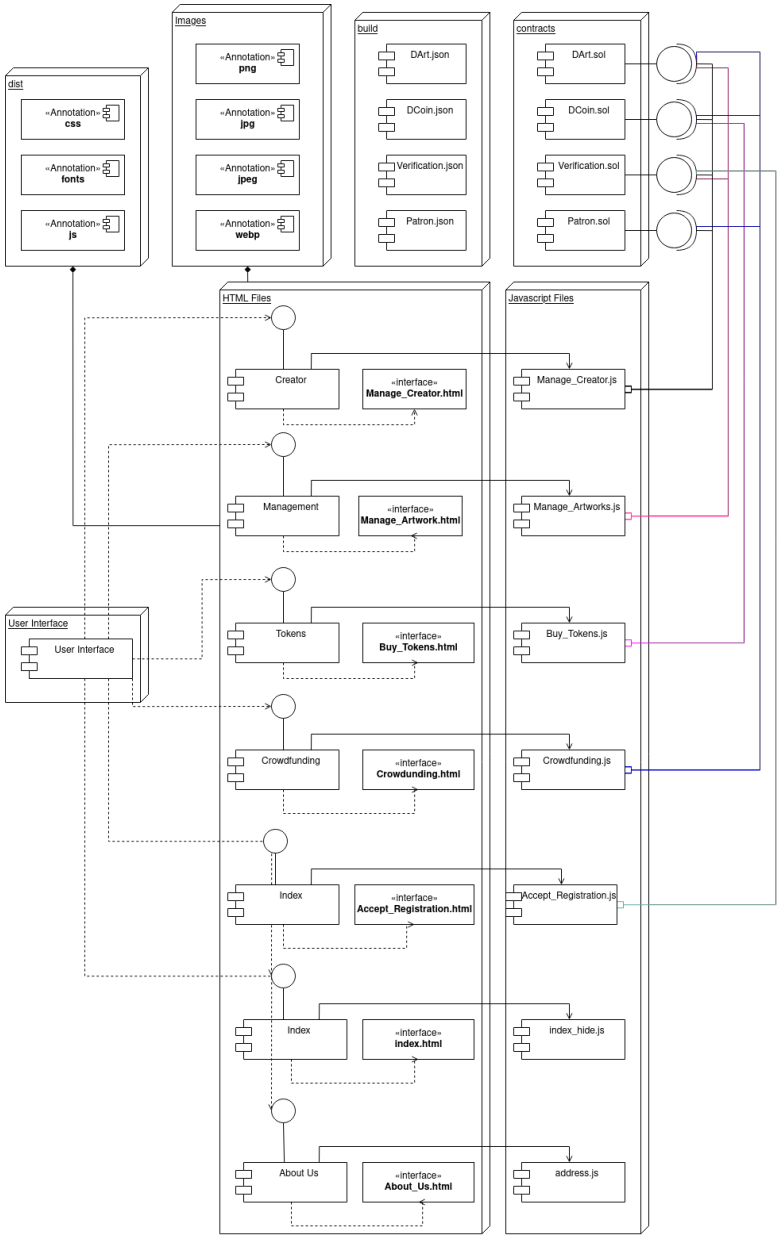


Figure 13: Component Diagram

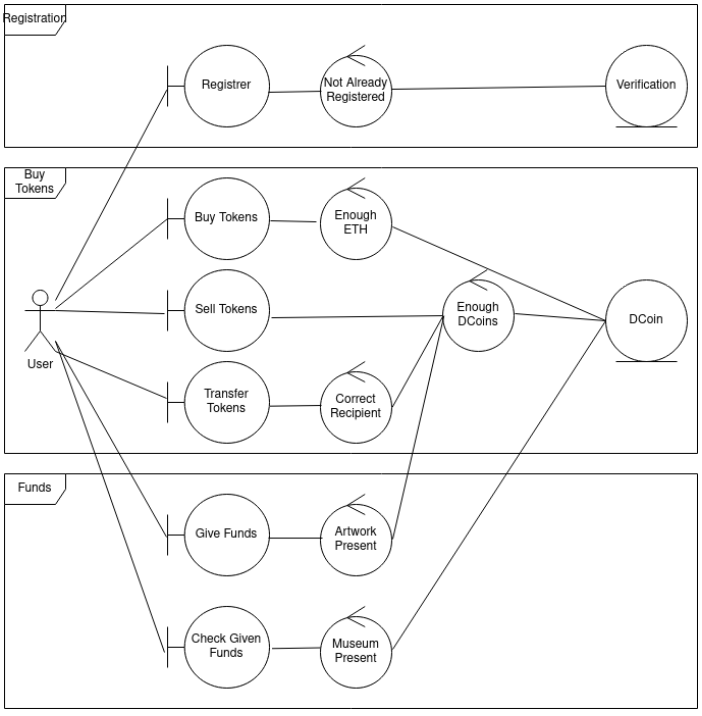
This is another very important diagram because with it we can see all the possible interfaces and all the possible components (modules) of our system. So we can see for each principal part of our system (Creator, Management, ...) which are the relative web-pages of our web-site to access; for each web-site we can spot which are the possible Images files and which are the possible fonts, CSS files and JS files used with the directory "dist". Then we can also spot for each HTML file which are Javascript files that they use and for which they interact to do, then, modifications on the Blockchain with the smart-contracts. Finally, Like you know, each smart-contract is linked to the corresponding JSON file, that represents the ABI of it.

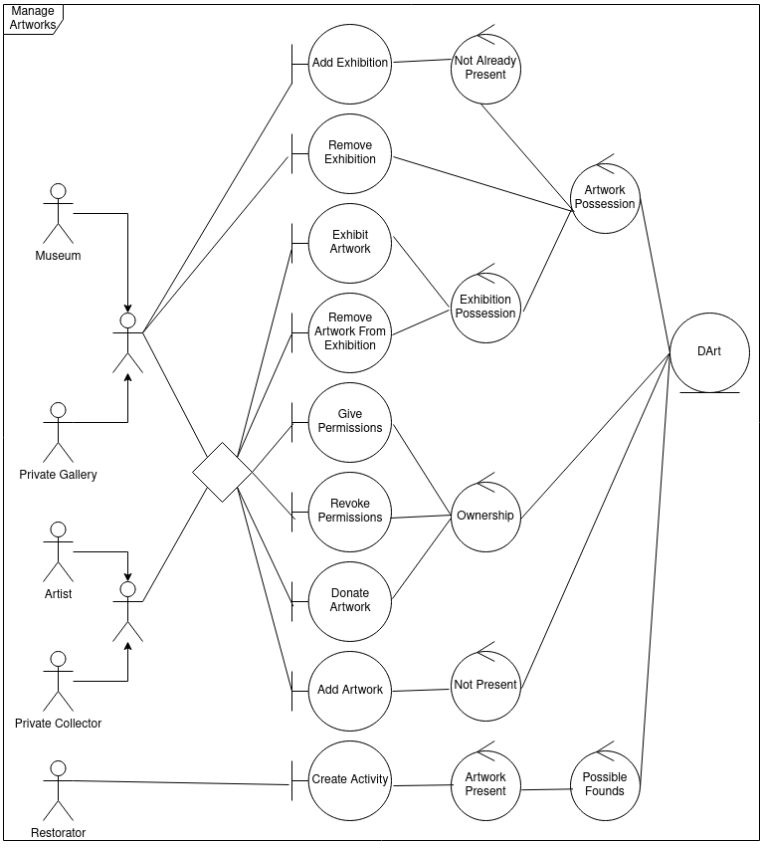
So with this diagram we have another perspective of the system, seeing which are the relationships, interfaces and with which files in which directories each file interacts. For this reason we have a lot of utility by means of it, because it's really a lot representative for the basic interactions for our system.

The last thing to understand is the User Interface component, that it's an abstract module that it's needed because each actor that we have in our system accesses to it with the user interface, then it accesses (considering the right permissions) to the right part of the web-site. We don't see the permissions of the actors, because for it we have the Collaboration Diagram.

7.4. Collaboration Diagrams

The collaboration diagrams (also called communication diagrams) have the focus on messages passed between objects, so we need to represent the communication between actors to interfaces and from interfaces to controls and, finally, to the corresponding involved entity (the smart-contract)





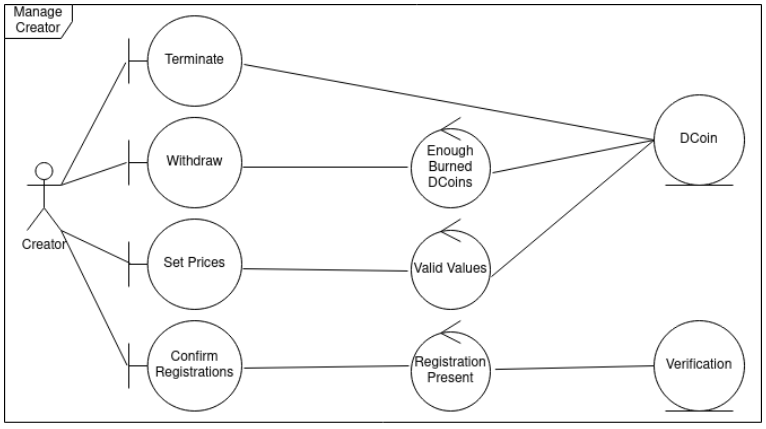


Figure 14: Collaboration Diagram

In this diagram we can see, for each macro-task (called "frame"), the opened interfaces for the user, which are the requirements and which smart-contract will be used. Obviously according to the actor we have different possible actions that can be performed.

So we have that the normal user (that can be registered or not, it doesn't matter) can do a lot of operations, namely the same that we can see later on with the Use-Case Diagram. The important thing is that in this case we see which are the interfaces to communicate, so with which forms / links or tools we communicate our needs (for example to buy tokens), then for each possible interface we need to do some controls: An example is really simple, in fact if we want to buy DCoins, but we haven't enough DCoins, then obviously this action will be rejected. We can also have concatenated controls or that more controls / interfaces use the same control, so we can have even many inputs for a single control, but always we have only one single output for each possible interface / control, because we cannot have a splitting (like as a function works).

Another important thing to fix is when we have an actor that group more distinct actors (for example the anonymous actor that group Museum and Private Gallery): We have this situation because abstractly these two actors are different, butin practise they perfect do the same pattern of actions in our system, so for a readable aim it's not so useful to fill the diagram with many many arrows, but it's better if we group them. Another similar thing to see in the diagram is the use of rhombus, that it has the same aim of the anonymous actor: in fact it is used to group the functionalities that different actors can do, so it is, even in this case, only an element to make easier the read for you for the diagram.

Very last thing to tell is that we always have the smart-contract (that it is the entity) at the end of the diagram and it is really a lot important because with it we have a communication from the interfaces / controls to it. In fact always the actor wants to communicate with the smart-contract, to do this it has the need to use the interfaces and to avoid problems of coherence and integrity we often need to use some controls for these operations.

7.5. Use Cases

The **Use Cases** are one of the most important part of our project, because they explicit which are the *Possible Actions* and *Functions* for our *Users.*

In this way we can see that we have many actors that use our system, namely(see 15):

(i) Normal User (that it needs to be registered into MetaMask): It is the normal user that constitutes our system and it can do a lot of possible operations:
  (a) It can See simply some informations regarding of the artworks or regarding of DArt;
  (b) It can Buy or Sell or Transfer DCoins, so to do financial movements to invest in our coin or to use it in our environment;
  (c) It can Donate DCoins to museums to help the restoration activities for the artworks and to receive some incentivates, given by Patron Credit (you will see later);
  (d) It can Request to be accepted if it is a museum, private gallery, restorer, artist or private collector: keep attention, everyone can request to be registered, but obviously the creator decides who will be registered or not, using an external method of verification.

(ii) Creator: He is who creates the entire system and we can recognize it because we have stored his address → He has a lot of responsibilities:
  (a) He accepts the requests of museum, private galleries, private collectors, artists and restorers:

according to the documents exchanged off-chain, he knows if he can accepts or not some requests, seeing the addresses requested (of MetaMask account), names and role and then he compares everything with the off-chain documents;

(b) He also has the responsibility to link together the smart-contracts, passing to each one the addresses of the other ones that are requested, respect to the necessity to have a good communication between them;

(c) He can withdraw ETH (in fact you need to remember that the Smart-Contract gives DCoins to the user, but it takes Ethereum, that it is stored in its account) and it can terminate the whole system, terminating all the smart-contracts.

(iii) Verified User: He can be strickly subdivided in 5 possible sub-actors:

(a) Restorer: He is the actor that physically repairs the damages of the artworks and that he can fix some problems. Specifically he can create activities:

(1) Restauration Activity: He can fix damages of the artworks;
(2) Maintainance Activity: He can do periodic maintainance, checks or supervisions;
(3) Damage Activity: He can spot damages on the things;
(4) Protection Activity: He can avoid damage of some artworks, e.g. putting a glass over an artwork;
(5) Prevention Activity: He can do some prevention, in the sense that he can prevent some problems of the damages on the artworks;
(6) Update Activity: He can add some plus informations that they cannot be classified in the previous cases;
(7) P.S. For Some Activities: He can take DCoins from the donations to do an activity, e.g.: restoration activity or to perform some interventions.

(b) Artist / Private Collector[§]: They can only create artworks, so the basic concept of the whole system;

(c) Museum / Private Gallery: They are the places (public and private ones) where the artworks are contained. They can do many operations on the artworks:

(1) They can show artworks by means of exhibitions (only if they have the possession of the artworks);
(2) They can remove artworks from exhibitions (if they are the owners of the exhibitions);
(3) They can donate artworks to other museums / private galleries;
(4) They can grant / revoke permissions for the possession for specific artworks.

[§] *They are put together because both of them can perform the same actions.*

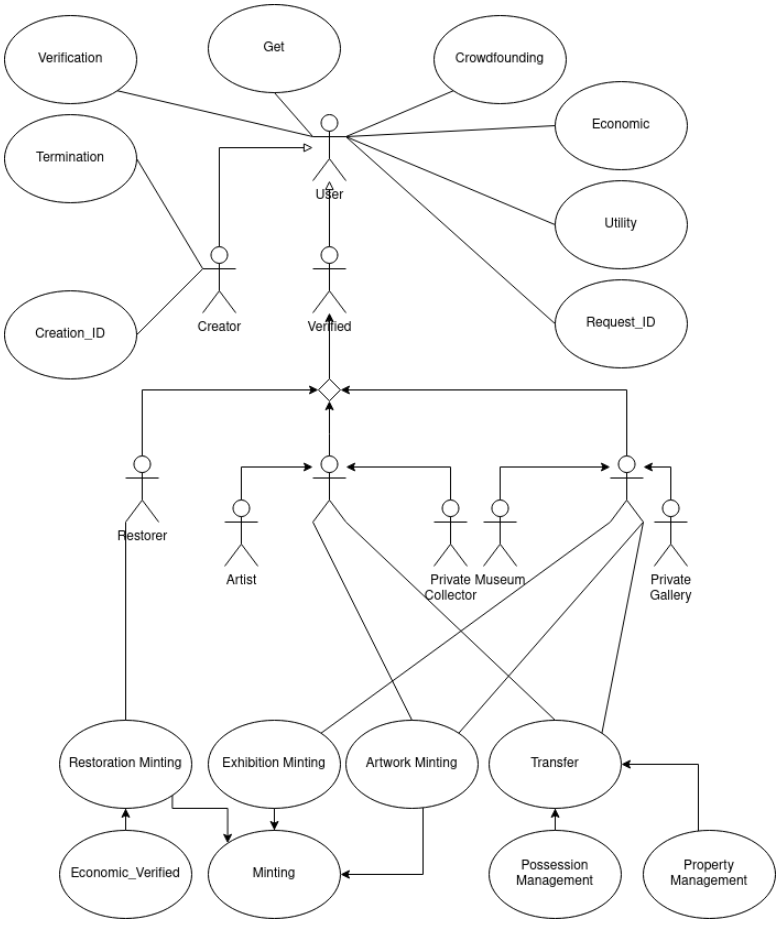


Figure 15: Use-Case Diagram

7.6. Activity Diagrams

7.6.1. *DCoin*

In the figure is present the activity diagram about the minting of DCoins and the withdraw of them.

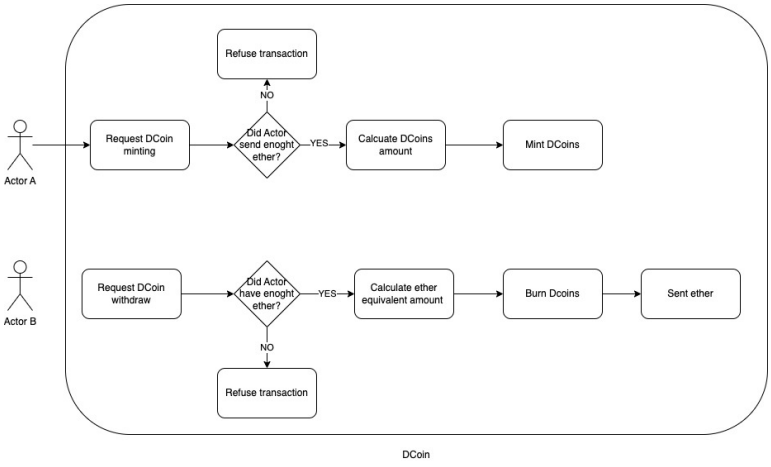


Figure 16: DCoin token activity diagram

7.6.2. *VerificationID*

In the figure is present the activity diagram about the minting of a VerificationID. The Actor A is a actor opening to a verification request, the Actor B is trying to accept a pending request.

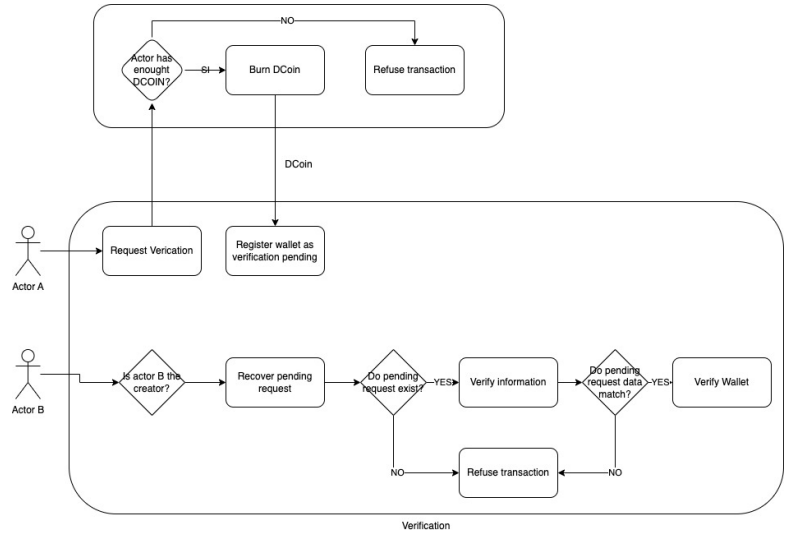


Figure 17: VerificationID token activity diagram

7.6.3. *Patron Credit*

In the figure is present the activity diagram about the minting of a PatronCredid. It is important to remember that each user has an amount of patron credit for each museum on the platform, therefore the amount of patron credit generated with the interaction in the diagram relates to the museum that owns the crowdfunded artwork.

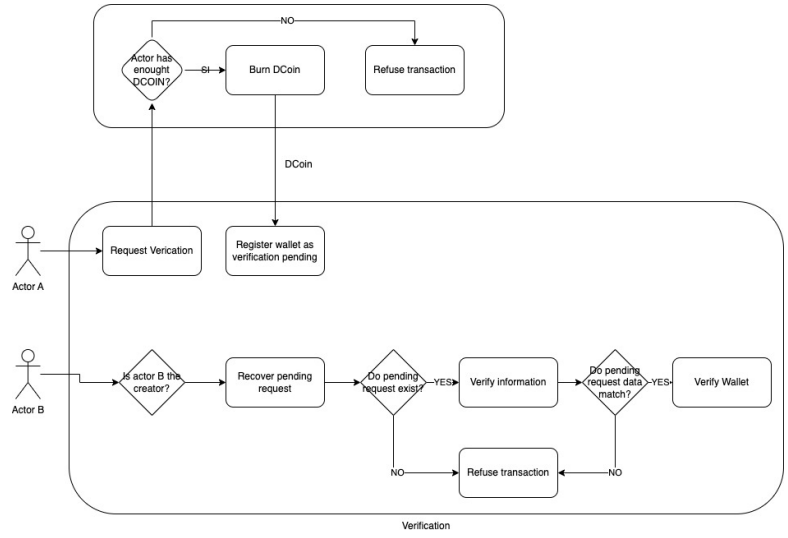


Figure 18: VerificationID token activity diagram

7.6.4. *ArtworkNFT*

In the figure is present the activity diagram about the minting of a Artwork NFT.

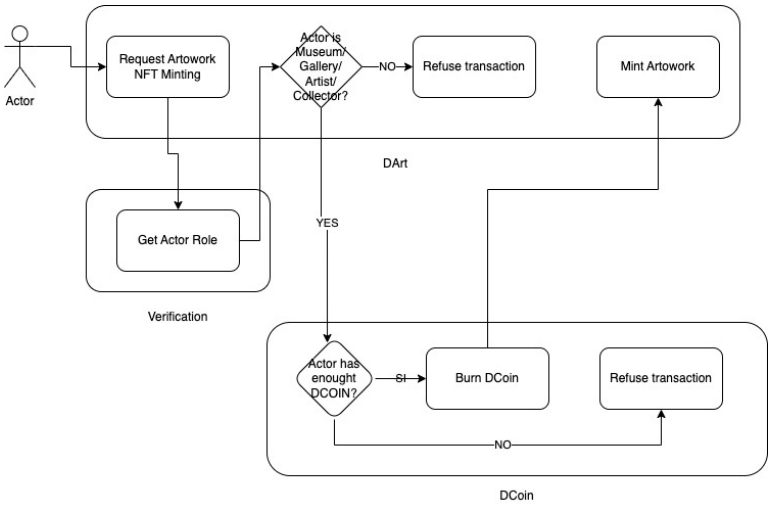


Figure 19: ArtWork NFT activity diagram

7.6.5. *ExhibitionNFT*

In the figure is present the activity diagram about the minting of a Exhibition NFT.

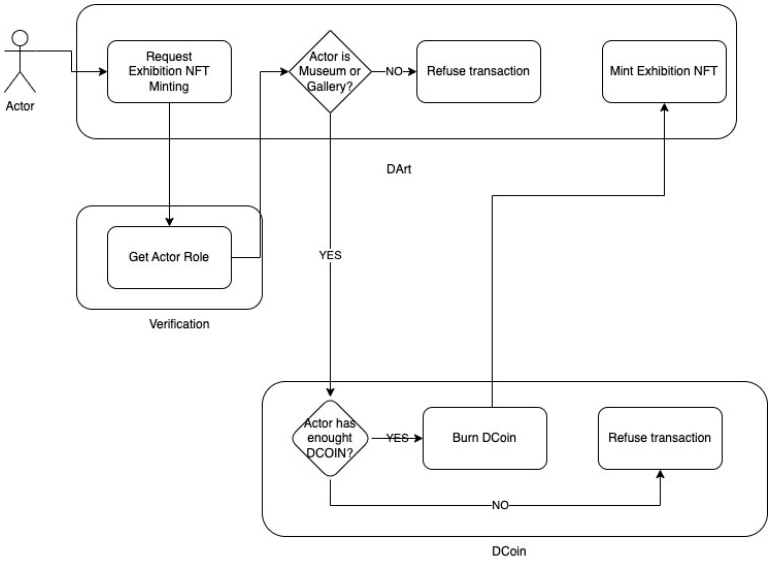


Figure 20: Exhibition NFT activity diagram

7.6.6. *ActivityNFT*

In the figure it is present the activity diagram about the minting of a Activity NFT. It can be seen that this is the only type of interaction with the smart contract that involves all four defined contracts.

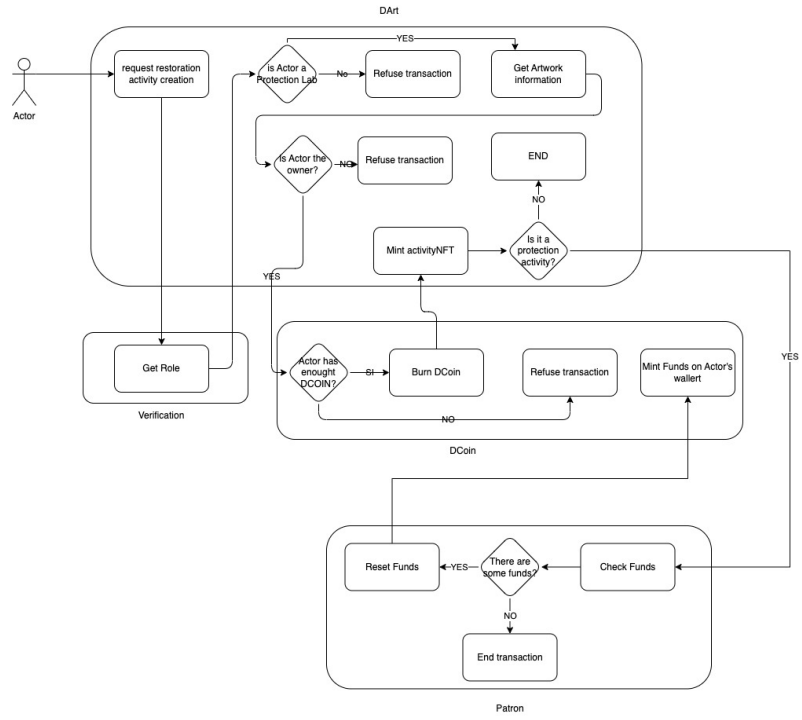


Figure 21: Activity NFT activity diagram

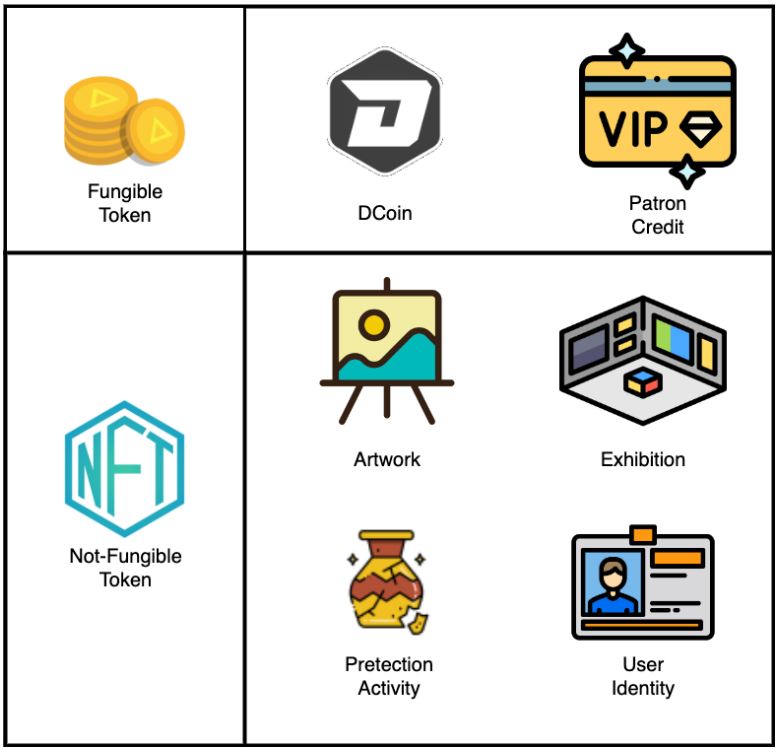


Figure 22: Fungible and not-fungible tokens

8. Implementation

In this charapter is described the developing phase of the smartcontracts, their interaction with website and their role in the dapp. The web interfaces of the DApp are also illustrated. If you want see a brief demo, then you can see here.

8.1. Tokenization

The first step of the developing design is to define the entities to be represented by a token: fungible or not. Obviously the DCoin, the application token, needs to be defined as a fungible token. The same goes for patron credit, although this is not exchangeable or sold in ether. The key elements of DArt, i.e. information on the Artworks, exhibitions and protection activity information, are managed through the non-fungible token. Non-fungible tokens have also been used to represent the identity of registered users.

8.2. Smartcontract interaction

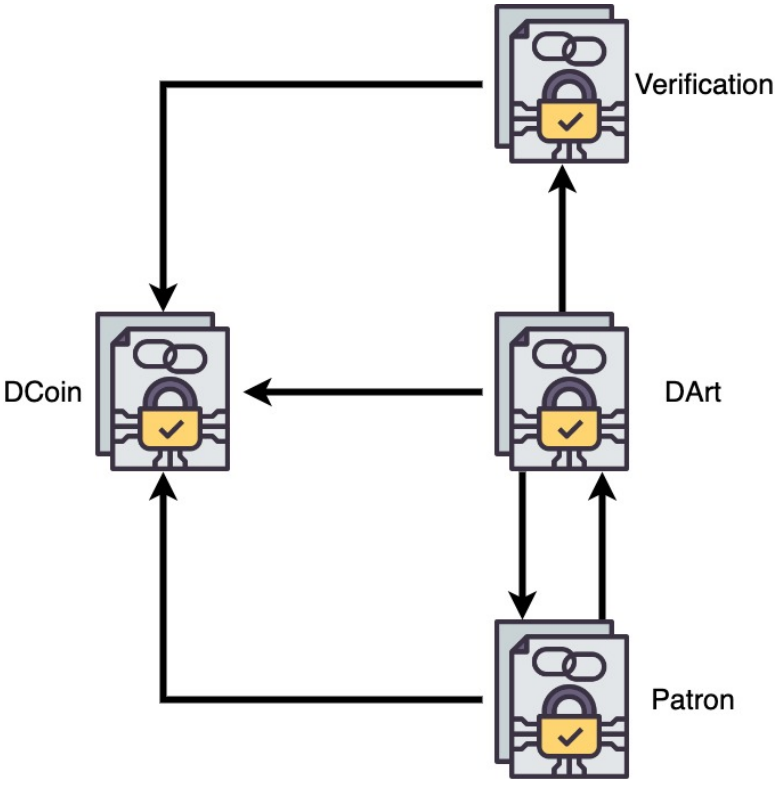


Figure 23: Smartcontract interaction schema

During development designing, four different contracts were designated, which are detailed in the following sections, like you can see in fig. 23.

Given this interaction hierarchy, to reduce deployment costs, the role of deploying all contracts was initially assigned to the *DArt.sol*'s constructor, but subsequently to facilitate interaction with Ganache in the development and debugging phase , it was decided to carry out all four deployments separately and then assign to each contract the addresses of the contracts that interact or must interact with it, at a later time.

```
function setContracts(address dcoinad, address verificationad, address patronad) external{
    require(msg.sender == creator, "Only the creator can set the contracts");
    dcoin = DCoin(dcoinad);
    verification = Verification(verificationad);
    patron = Patron(patronad);
}
```

Figure 24: setContracts function in DArt.sol

8.3. DCoin

DCoin smartcontract contains essential information on the DCoin balance of all addresses, the amount of dcoin burned from the users to accesses to DArt services on information about their cost. There are the classic functions for minting the token, for selling or transferring it, as well as functions accessible to the constructor for terminating the contract or withdrawing the ether equivalent of the DCoins burned by users to access DArt's services. There are also functions accessible only by the Patron smart contract to block and unblock a user's funds. Additionally, DArt and Verification smartcontracts can access to DCoin smartcontract to manage the payment of its services, burning an amount of DCoin in the user's balance equal to the cost of the service set by the creator using the setCostOfServices() function. The cost of the services is stored in the smart contract storage as a vector of uint, with a fixed size of 4.

```
function  setCostOfServices(uint[4] memory cost) external {
    assert(minter == msg.sender);
    costOfServices = cost;
}

function mint() public payable {
    require(msg.value >= PRICE, "Not enough value for a token!");
    balance[msg.sender] += msg.value / PRICE;
    // Guess guess, where does the remainder of the msg.value end?
}

function burn(uint costType, address wallet) external {
    assert(msg.sender == mainSmartcontract || msg.sender == verificationSmartcontract);
    uint amount = costOfServices[costType];
    require(balance[wallet] >= amount, "Not enough DCoins!");
    // Take the amount of HelloToken from the sender and give back the amount of ether
    balance[wallet] -= amount;
    burned += amount;
    // Send the amount of ether to the sender
}
```

Figure 25: setCostOfServices, mint and burn functions in DCoin.sol

The smart contract emits an event whenever a user receives funds through another user's transfer.

```
function transfer(uint amount, address to) external {
    require(balance[msg.sender] >= amount, "Not enough DCoins!");
    require(amount > 0, "Transfer amount canno be less than 1 Dcoin");
    balance[msg.sender] -= amount;
    balance[to] += amount;
    emit Transfer(msg.sender, to, amount);
}
```

Figure 26: transfer function in DCoin smartcontract

8.4. Verification

Verification smartcontract has the role to manage all the information about the identity of verified wallets. Any user can open a verification request, giving its information and minting an Verification NFT, but only the creator che mark it as verified, making it usable in DArt dapp. A user's id is represented by his wallet. In order to ensure the absolute truthfulness of the information, to carry out a verification the creator must input both the id of the user who wants to verify and the information associated with him, such as name and role.

```
//called by a museum that wants to enter in the blockchain
/**
    @notice the sender create a request of verification, minting an IDNFT wi
    @param hashedName the hashed name of the requesting actor
    @param role the type of the actor
 */
function museumRequestCreation(bytes32 hashedName, Actors role) external {
    dcoin.burn(3, msg.sender);
    if (registeredWallets[msg.sender].verified == false){
        registeredWallets[msg.sender] = Wallet(hashedName, false, role);
    }
}

function museumCreation(bytes32 hashedName, Actors role, address museum) ext
    // veryf the sender is the creator/owner of the smart contract and that
    assert(msg.sender == creator);
    require(registeredWallets[museum].hashedName == hashedName && registered
    if (msg.sender == creator && registeredWallets[museum].hashedName != 0){
        // change the status of the museum to verified
        registeredWallets[museum].verified = true;
    }
     else {
        revert();
    }
    emit walletVerified(museum);
}
```

Figure 27: Verification functions example

This smartcontract does not have functions with exclusive access to other smartcontracts, but makes a call to DCoin for payment of the verification request service. In the storage of this contract there are only the addresses of the creator and DCoin.sol, as well as a dictionary that maps each address to a struct Wallet, which is the non-fungible token that represents the identity card of each verified user (or awaiting verification).

```
contract Verification {

    event walletVerified(address indexed wallet);

    /// @notice A collection of informations about a wallet that operate with the contact
    struct Wallet{
        //we have to hash also the address?
        bytes32 hashedName;
        bool verified;
        Actors role;
    }

    // This enum indicates the type of actor associated to a registered wallet
    enum Actors {
        MUSEUM,
        GALLERY,
        PRIVATE_COLLECTOR,
        ARTIST,
        PROTECTION_LAB
    }

    //this is the address of the creator, namely the owner
    //of the smart contract that it can do some special actions
    address public creator;

    DCoin public dcoin;

    mapping (address => Wallet) public registeredWallets;
```

Figure 28: Verification.sol's storage

Unverified user information is considered invalid in interaction with other DArt contracts.

```
// Get the role of a wallet if it is registered in the blockchain as an verified actor
function getRole(address museum) external view returns(uint role) {
    if (registeredWallets[museum].verified == false){
        revert();
    }
    else {

        return uint(registeredWallets[museum].role);
    }
}
```

Figure 29: Verification.sol's function getRole

8.5. Patron

Patron smartcontract has the task of managing all the information and functions related to crowdfunding. Participation in a crowdfunding is open to all users, therefore the contract never needs to verify that an address belongs to a verified wallet or its role, except for the functions dedicated to interacting with the other smart contracts of the dapp. To make the donation you only need the id of the artwork concerned and an amount of DCoin present in your wallet. The collected funds are saved in a mapping that associates an amount of DCoin to each ArtworkNFT id. Each dcoin donated is converted into PatronCredit, which is a token associated with each Museum or Private Gallery. This information is stored in the smartcontract storage with a dictionary that maps the user's address to which the patroncredits belong and the address of the associated museum or gallery, to the amount of tokens in the user's wallet.

```
function crowfunding(bytes32 artwork, uint amount) external {
    address museum  = mainSmartcontract.getProperty(artwork);
    require(museum != address(0x0), "Artwork does not exist");
    dcoinSmartcontract.lock(amount, msg.sender);
    funds[artwork] += amount;
    patronCredit[hashAddressAndAddress(msg.sender, museum)] += amount;
    emit Donation(msg.sender, museum, artwork);
}
```

Figure 30: crowdfunding function code in Patron

Upon the occurrence of appropriate conditions, the blocked funds are moved to an address given in input. This feature is only accessible from the DArt.sol smart contract.

```
function moveFunds(address _to, bytes32 artwork) external {
    assert(msg.sender == address(mainSmartcontract));
    uint fund = funds[artwork];
    if (fund != 0) {
        dcoinSmartcontract.magicMint(_to, fund);
        funds[artwork] = 0;
        emit UnlockedFunds(_to, artwork);
    }
}
```

Figure 31: moveFunds function in Patron

In the two previous examples it is possible to notice that in case of movement (unlock or unlock funds) of DCoin two events are emitted

```
contract Patron {

    event Donation(address sender, address indexed receiver, bytes32 artwork, string message);
    event UnlockedFunds(address indexed receiver, bytes32 artwork, string message);

    mapping (bytes32 => uint) public patronCredit;
    mapping (bytes32 => uint) public funds;
```

Figure 32: Patron's events

8.6. Main Contract (DArt)

This contract has the role of managing all the information and operations for the main elements of DArt, i.e. the artwork, the exhibitions and the protection activities (like restoration). Interaction with this smart contract is limited to the creator and verified users, with the exception of view functions.

```
//we have a struct to store the data about artworks
struct Artwork {
    bytes32 hashedName;
    address minter; //the address of the museum that minted the artwork
    address property;
    address possession;
    bytes32 exposedAt; // Da vedere come gestire lo storico delle esposizioni
    bytes32 status; // Da vedere come gestire lo storico delle operazioni
}

// A struct to collect information about exibitions
struct Exibition {
    bytes32 hashedName;
    address organizer;
    bool isOn;
}

struct Activity{
    // bytes32 artworkId;
    address author;
    uint timestamp;
    ProtectionActivities typology;
    bytes32 extraInfo;
}
```

Figure 33: Dart.sol's structs

Any ArtworkNFT contains information about the artwork name, who minted it, the property and the possesion of the artwork in the real world. Each artwork also contains information about the latest information relating to the protection activity and possibly where the work is currently exhibited, thanks to the references to the registered exhibitions and activities. Each protection activity has a typology field that represents the nature of the activity performed, represented by an enumeration. Each NFT is saved in the storage using a mapping structure.

```
enum ProtectionActivities {
    PREVENTION,
    PROTECTIOIN,
    MAINTAINANCE,
    RESTAURATION,
    DAMAGE,
    UPDATE
}

//this is the address of the creator MAYBE PRIVATE, namely the owner
//of the smart contract that it can do some special actions
address public creator;

mapping (bytes32 => Artwork) public registeredArtworks;
mapping (bytes32 => Exibition) public registerdExibitions;
mapping (bytes32 => Activity) public registerdActivities;
```

Figure 34: Dart.sol code example

Using mintArtWorkNFT and mintExhibitionNFT, users with the necessary permissions and enough dcoins can mint nfts representing an artwork or exhibit. The identifier of these NFTs is hashed by hashing their name and their minter address, allowing different users to create NFTs with the same name.

```
//called by a museum, to add an artwork in blockchain (MAYBE TO DO
/**
    @notice mint an Artwork NFT. The id of the NFT is the hash of its name and the minter address.
    @param hashedName name of the artwork hashed using keccak256
 */
function mintArtworkNFT(bytes32 hashedName) external {
    uint role = verification.getRole(msg.sender);
    assert(role < 4);
    bytes32 kek = hashTextAndAddress(hashedName);
    require(registeredArtworks[kek].minter == address(0x0), "A collision during hashing occurred");
    dcoin.burn(0, msg.sender);
    registeredArtworks[kek] = Artwork(hashedName, msg.sender, msg.sender, msg.sender, 0, 0);
}

/**
    @notice mint an Exibition NFT. The id of the NFT is the hash of its name and the minter address.
    @param hashedName the Exibition's name hashed using keccak256
    @param status indicates the status of the creare exibition, if it's on or not
 */
function mintExibitionNFT(bytes32 hashedName, bool status) external {
    uint role = verification.getRole(msg.sender);
    assert(role < 2);
    bytes32 kek = hashTextAndAddress(hashedName);
    assert(registeredArtworks[kek].minter == address(0x0));
    dcoin.burn(1, msg.sender);
    registerdExibitions[kek] = Exibition(hashedName, msg.sender, status);
}
```

Figure 35: mintArtworkNFT and mintExhibitionNFT functions

Similarly, a protection activity can be minted by calling the mintActivity function. In this case the NFT id is given by the address of the minter and the timestamp of the block. This function can only be called by a PROTECTION LAB that owns the artwork whose restoration it intends to record. To ensure that you have possession of the artwork, its owner must have called the allowAccessToArtwork function, thus granting the permissions.

If the activity is of the PREVENTION, PROTECTION, MAINTAINANCE or RESTAURATION type, the smart contract calls the function seen in the previous section moveFunds, directing the funds to the address of the protection activity minter.

```
function mintActivity(bytes32 artworkID, ProtectionActivities oftype, bytes32 extrainfo) external{
    uint role = verification.getRole(msg.sender);
    assert(role == 4);
    require(registeredArtworks[artworkID].possession == msg.sender,  "You do not have the necessary pe
    bytes32 kek = keccak256(abi.encodePacked(msg.sender, block.timestamp));
    registeredArtworks[artworkID].status =   kek;
    registerdActivities[kek] = Activity(msg.sender, block.timestamp, oftype, extrainfo);
    dcoin.burn(2,msg.sender);
    if (oftype != ProtectionActivities.UPDATE && oftype != ProtectionActivities.DAMAGE){
        patron.moveFunds(msg.sender, artworkID);
    }
}

function allowAccessToArtwork(address target, bytes32 artwork) external {
    //assert(museums[target].verified);
    assert(registeredArtworks[artwork].property == msg.sender);
    registeredArtworks[artwork].possession = target;
    emit allowedAccessToArtwork(artwork, target);
}

function revokeAccessToArtwork(bytes32 artwork) external {
    assert(registeredArtworks[artwork].property == msg.sender);
    emit deniedAccessToArtwork(artwork, registeredArtworks[artwork].possession);
    registeredArtworks[artwork].possession = msg.sender;
}
```

Figure 36: mintActivity functions

All the minting functions interact with the Verification contract to verify the role of the sender and with DCoin for the payment of service costs.

Finally, there are the functions for managing the exhibitions, which allow you to conclude one, exhibit an artwork in your possession or leave it. Only the minter of an exhibition can finish it or exhibit an artwork.

```
function endExibition(bytes32 exibitionID) external {
    require(registerdExibitions[exibitionID].organizer == msg.sender, "You do not have the necessary permiss
    registerdExibitions[exibitionID].isOn = false;
}

function exposeArtwork(bytes32 artworkID, bytes32 exibitionID) external {
    require(registeredArtworks[artworkID].property == msg.sender, "You do not have the necessary permissions
    require(registerdExibitions[exibitionID].isOn, "The exibition is not on");
    require(registerdExibitions[exibitionID].organizer == msg.sender, "The exibition is not on");
    registeredArtworks[artworkID].exposedAt = exibitionID;
}

function removeArtworkFromExibition(bytes32 artworkID) external {
    require(registeredArtworks[artworkID].possession == msg.sender, "You do not have the necessary permissio
    registeredArtworks[artworkID].exposedAt = bytes32(0x0);
}
```

Figure 37: functions to maneae Exhibitions

An event is triggered when possession or ownership of an artwork changes.

```
event allowedAccessToArtwork(bytes32 indexed artwork, address indexed user);
event deniedAccessToArtwork(bytes32 indexed artwork, address indexed user);
event donatedArtwork(bytes32 indexed artwork, address indexed user);
```

Figure 38: Events in DArt.sol

8.7. DCoin Wallet

By connecting your Wallet using Metamask, the user has the ability to control their balance and manage their DCoins, selling them, buying more, or sending them to other wallets.

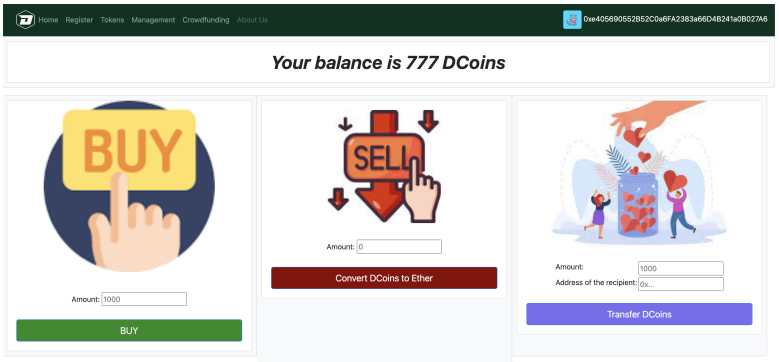


Figure 39: Registration UI

8.8. Registration

On the DArt website there is a special page for the verification request, which interacts with the Verification contract.

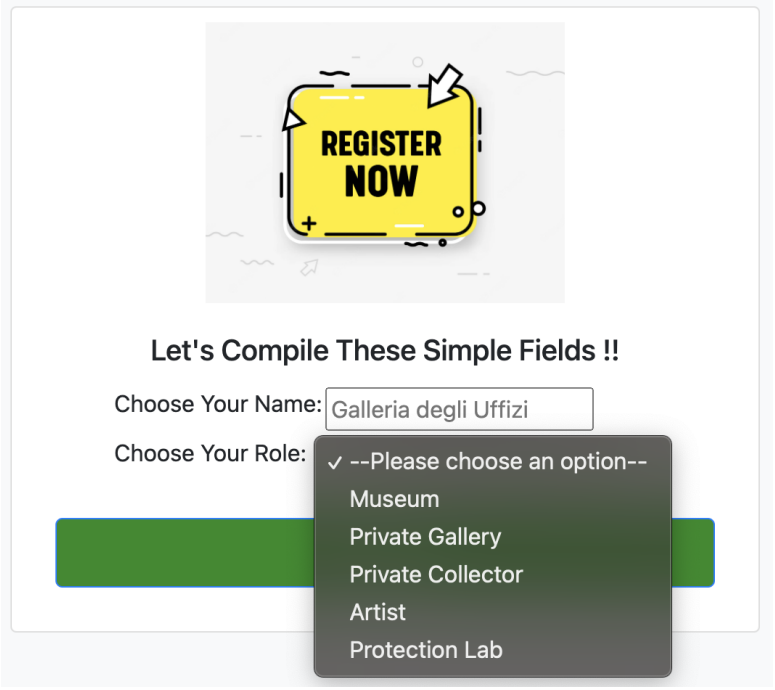


Figure 40: Registration UI

8.9. Creator

A page accessible only to the creator has been implemented. It allows you to accept verification requests, change the price of services, terminate contracts, withdraw burned DCoins and finally carry out the initial setting of communication between smart contracts.

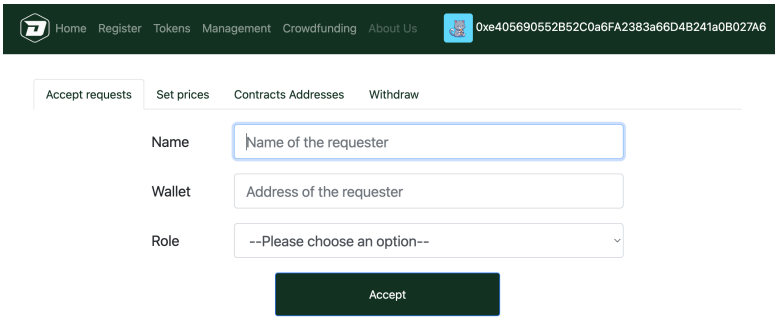


Figure 41: Creator tab UI - 1

8.10. Manager

In the section dedicated to the management and minting of one's NFTs. There are dedicated and different interfaces depending on the role of connected user.
Museums and art galleries can use this interface to mint works of art and exhibitions, exhibit works, donate them, give them possession and conclude their exhibitions.

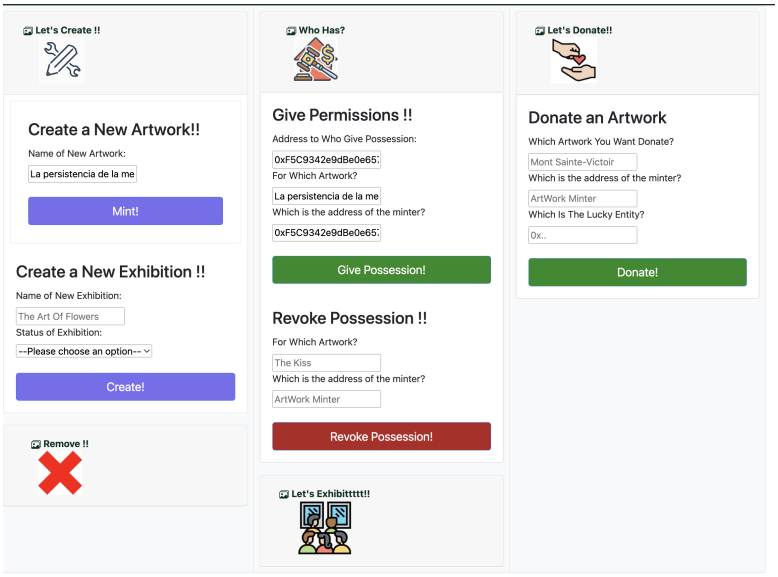


Figure 42: Manager cards UI for Museum and Private Galleries - 2

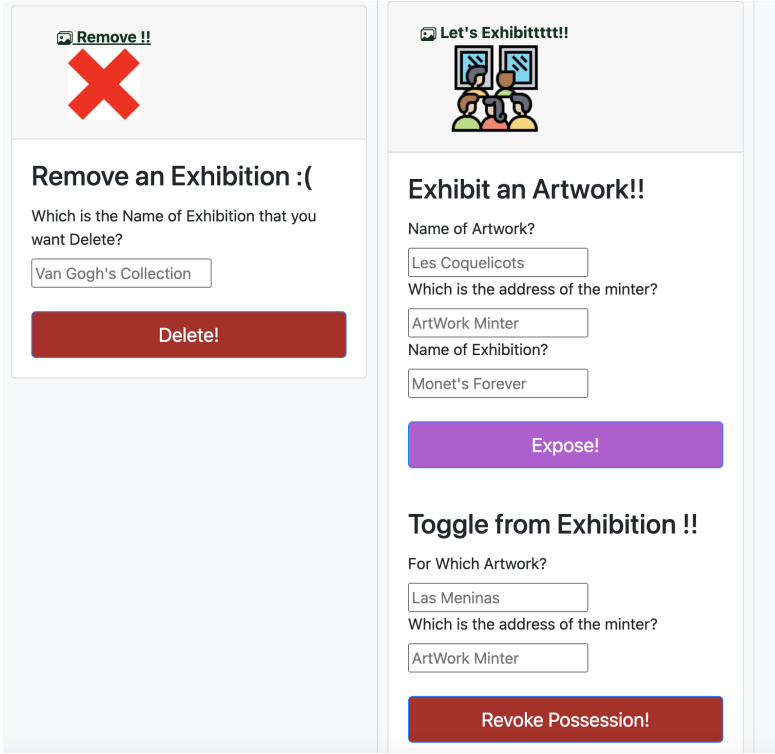


Figure 43: Manager cards UI for Museum and Private Galleries - 2

Users logged in as protection laboratory can interact with the UI to mint protection activities, providing information on the artwork (name and minter) and on the nature of the activity (type and extra information).

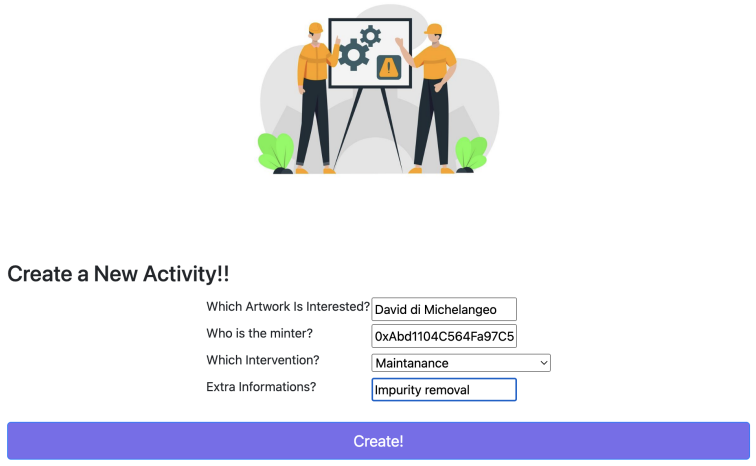


Figure 44: Manager cards UI for Protection laboratories

Artists and private collectors can use the dedicated interface to mint the NFTs relating to their works, as well as manage their possession or donate them.

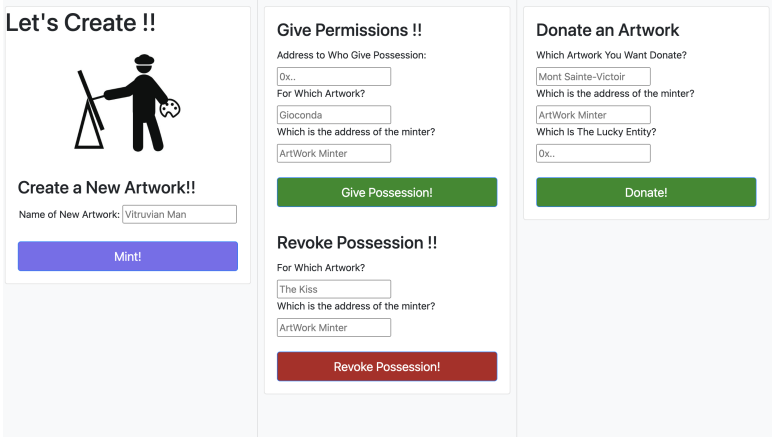


Figure 45: Manager cards UI for Private collectors and Artist

8.11. Crowdfunding

On the page dedicated to crowdfunding there is a card with which it is possible to search for information about a work and participate in the fundraising through a donation in DCoin. The information relating to a work that can be viewed are the minter, possession, ownership, if the work is currently on display and possibly the associated exposure, the last protection activity relating to the asset and infinite the amount of funds raised for a new restoration activity through user donations.

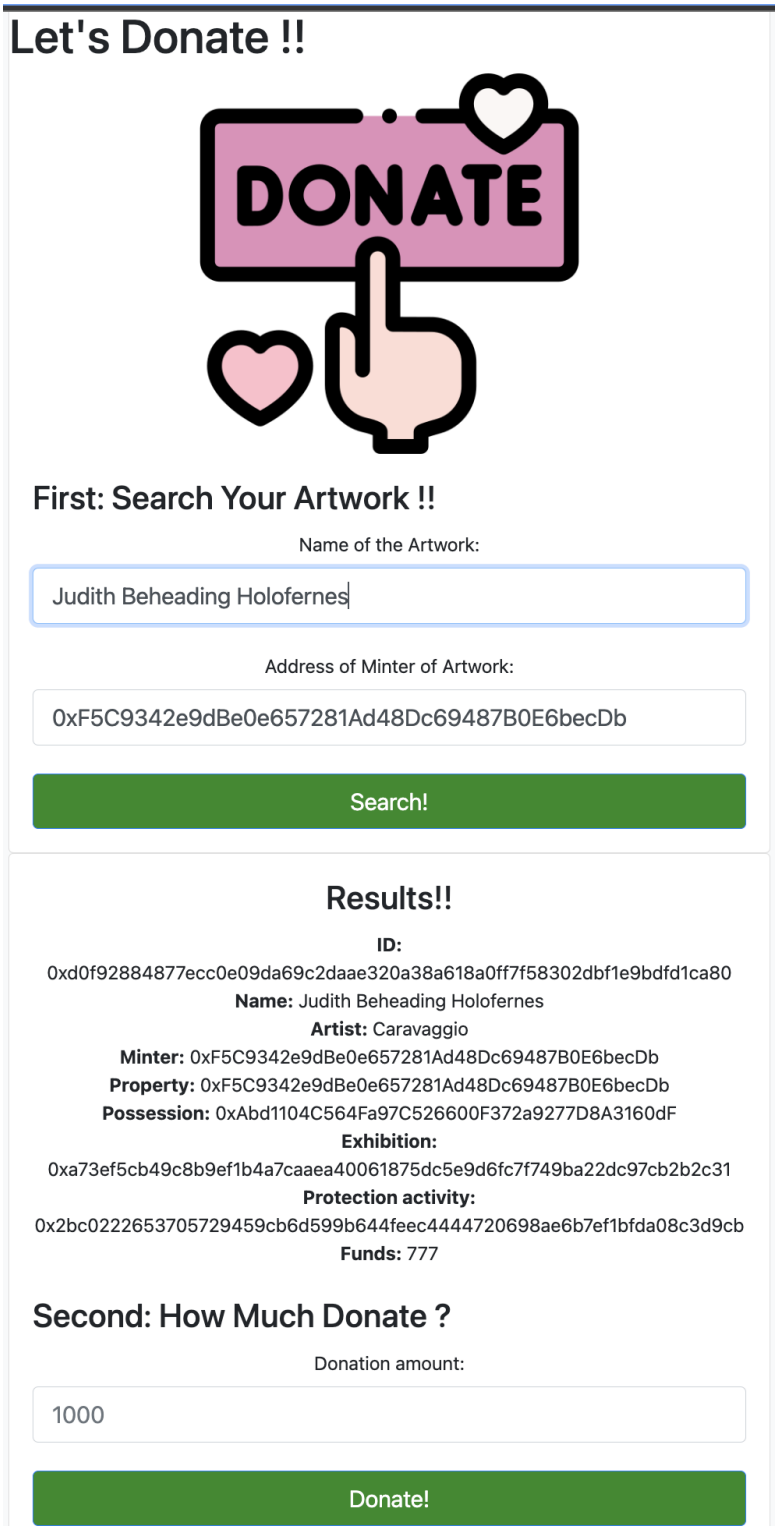


Figure 46: Crowdfunding card UI

There is a second card through which the user can check their patron credits by entering the address of the museum wallet.

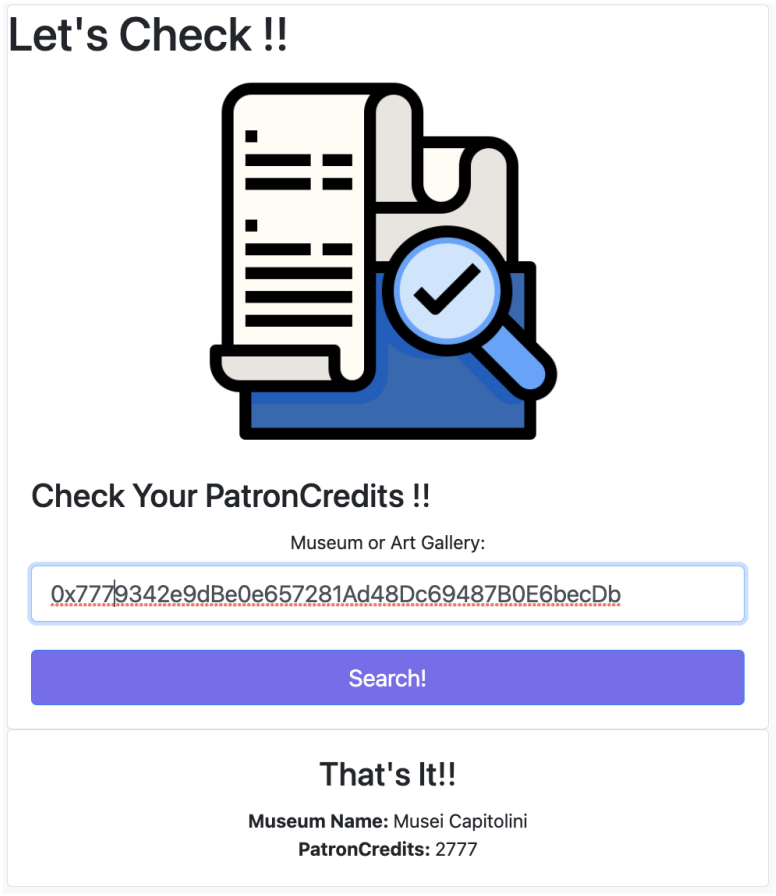


Figure 47: Patron card UI

9. Aesthetic Part

For our application we have worked also on the graphical part that regards of the logo design and application design.

More in detail we have that the logo that we have created for our application is the following one:

Figure 48: Application Logo

Instead the logo of our coin (so the DCoin) is the following one:

Figure 49: DCoin

10. Limitations and Issues

Interaction with the world of culture can present many obstacles. In fact, Dart's proposal presupposes a major change in the management of museums and how they interact with each other. The world of cultural heritage is a difficult market to enter, due to the historical dimension and its players, among which there is a strong presence of public entities. Currently, especially in Europe, museums are mostly public bodies with a long history deep traditions and complex bureaucracy. These two factors mean that any type of change has to deal with a reality that is by its very nature static and vast. In fact, research and innovations in this field take long periods to take place both due to the quantity of stakeholders to take into account and the difficulty of updating such a large and scattered apparatus throughout the territory.

Although museology - the study of the museum as exhibition and collecting space - is making enormous strides in trying to deal with the technologies available at the moment, some countries are struggling to distance themselves from tradition. Convincing an entire sector to change therefore becomes a major undertaking. In fact, the confrontation with change often generates fear, fear of losing one's traditions and of relying on something that could fail. This is often due to the lack of knowledge found above all in countries with a longer history of museums, such as Italy, which is only recently opening up to the use of technology within museums. It can also be noted that on the Italian and international scene it is usually contemporary art museums that are more welcoming towards technology. This creates a paradox because perhaps ancient art museums would need technology the most both to create a more lively relationship with the public and to make their objects - often the result of a primordial collecting that paid little attention to the retrieval of data - more certified and accessible. Many efforts are currently underway from this point of view as the accessibility of the collections as well as the technological advancement of museums are international goals in the field of museology and, although there are national systems for tracking the works, these are single efforts which are most of the times incomplete. In our specific case, we must take into account the fact that in the USA only 25% of people know what blockchain is, although United States is one of the most advanced countries in tech world (Ref.)[¶]. We can speculate that in Italy this percentage is much lower.

It is therefore necessary today more than ever to create a system that has an international level through which interaction between all the actors in the world of culture is possible.

Today the interaction with the system is cumbersome, being necessary to access the information of an artwork the knowledge of its exact name and the address of the minter. The use of the minter as a key is strictly necessary since the name of a work and other information such as the year and the artist do not necessarily constitute a unique key for the work, furthermore to prevent a user can definitively blocking the name of an artwork not really in his possession. To solve this issue it would be useful to interact with a database that allows greater navigability, giving the user the possibility to carry out searches with greater usability

Finally, since it's a complex market there's the need to store much more information relating to activities carried out, in particular the integration of information in different formats, such as images and videos, is essential. To do this, integration with the InterPlanetary File System (IPFS), a protocol, hypermedia and file sharing peer-to-peer network for storing and sharing data in distributed file system, is foreseen as future development.

11. Conclusions

For the development of DArt, the characteristics of the blockchain technology and of a DApp were analyzed, verifying that it was suitable for the use case, following an analysis of the case study and its context. Having viewed the state of the art in the world of cultural heritage and the effective utility of the blockchain application, the DArt objectives were defined in order to be able to continue with the development of the context in a manner consistent with the latter.

Five different types of verified actors have been defined, for each of these special permissions have been defined for interacting with the contract, in addition to the common users and the creator.

The tokenization of the main elements present in the DApp has been defined, such as the artwork, the exhibitions, the protection activities and the identity of the users, for the non-fungible tokens, and the Dcoins, the main expendable token of DArt, and the patron credit, obtained by participating in crowdfunding campaigns.

[¶] *We know that it isn't a paper or something similar to an academic research, but there isn't statistical material on it and we thing that this is "enough" reliable :)*

Four different smartcontracts have been developed that are able to interact and recognize each other: one for monetary management, one for verifying the identity of the actors, one for the management of cultural heritage and finally one for the management of crowdfunding campaigns.

An architecture suitable for deploying the project has been defined, containing both the tools for creating smart contracts and for interacting with them through a user interface on a web3.0 application, interacting with smart contracts through the use of html pages and javascript functions.

The entire system has been represented and designed through the appropriate UML diagrams in the *Software Architecture* section.

Consistent with the applicability analysis in the context of DArt, an analysis of limitations and future developments was made in conclusion, as well as a proposal for a more complex exchange system.

References

[1] *UNESCO culture for development indicators: methodology manual*. UNESCO Publishing, 2014.

[2] *Museums, Ethics and Cultural Heritage*. Taylor & Francis, 2016.

[3] Aynur Abdurazik and Jeff Offutt. Using uml collaboration diagrams for static checking and test generation. In *International conference on the unified modeling language*, pages 383–395. Springer, 2000.

[4] Thomas Adamson. U.s. and israel officially withdraw from unesco, Jan 2019.

[5] Rana M Amir Latif, Khalid Hussain, NZ Jhanjhi, Anand Nayyar, and Osama Rizwan. A remix ide: smart contract-based framework for the healthcare sector by using blockchain technology. *Multimedia tools and applications*, pages 1–24, 2020.

[6] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[7] Krzysztof R. Apt. *Verification of sequential and concurrent programs / Krysztof R. Apt, Ernst-Rudiger Olderog*. Texts and monographs in computer science. Springer, New York [etc, c1991.

[8] Chris Aquino and Todd Gandee. *Front-End Web Development: The Big Nerd Ranch Guide*. Pearson Technology Group, 2016.

[9] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[10] Donald Bell. Uml basics: The component diagram. *IBM Global Services*, 2004.

[11] Vitalik Buterin. Ethereum white paper: A next generation smart contract & decentralized application platform. 2013.

[12] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[13] Usman W Chohan. The limits to blockchain? scaling vs. decentralization. 2019.

[14] Chris Dannen. *Introducing Ethereum and solidity*, volume 1. Springer, 2017.

[15] Marlon Dumas and Arthur HM Ter Hofstede. Uml activity diagrams as a workflow specification language. In *International conference on the unified modeling language*, pages 76–90. Springer, 2001.

[16] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[17] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012.

[18] ICOM Italia. Definizione di museo: Scelta la proposta finale che sarà votata a praga.

[19] Akshita Jain, Sherif Arora, Yashashwita Shukla, T Patil, and S Sawant-Patil. Proof of stake with casper the friendly finality gadget protocol for fair validation consensus in ethereum. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 3(3):291–298, 2018.

[20] Moez Krichen, Meryem Ammi, Alaeddine Mihoub, and Mutiq Almutiq. Blockchain for modern applications: A survey. *Sensors*, 22(14):5274, 2022.

[21] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, jul 1982.

[22] Rana M Amir Latif, Muhammad Farhan, Osama Rizwan, Majid Hussain, Sohail Jabbar, and Shahzad Khalid. Retail level blockchain transformation for product supply chain using truffle development platform. *Cluster Computing*, 24(1):1–16, 2021.

[23] Wei-Meng Lee. Using the metamask chrome extension. In *Beginning Ethereum Smart Contracts Programming*, pages 93–126. Springer, 2019.

[24] Lorenzo Mucchi, Matilde Milanesi, and Claudio Becagli. Blockchain technologies for museum management. the case of the loan of cultural objects. *Current Issues in Tourism*, 25(18):3042–3056, 2022.

[25] Mayukh Mukhopadhyay. *Ethereum Smart Contract Development: Build blockchain-based decentralized applications using solidity*. Packt Publishing Ltd, 2018.

[26] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.

[27] International Council of Museums. Museum definition.

[28] Dirk Ohst, Michael Welle, and Udo Kelter. Differences between versions of uml diagrams. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 227–236, 2003.

[29] Massimo Ragnedda and Giuseppe Destefanis. *Blockchain and Web 3.0*. London: Routledge, Taylor and Francis Group, 2019.

[30] Helene Rey. International trade and currency exchange. *The Review of Economic Studies*, 68(2):443–464, 2001.

[31] M.A. Sandulli. *Codice dei beni culturali e del paesaggio*. Le fonti del diritto italiano. Giuffrè, 2012.

[32] Simanta Shekhar Sarmah. Understanding blockchain technology. *Computer Science and Engineering*, 8(2):23–29, 2018.

[33] Geri Schneider and Jason P Winters. *Applying use cases: a practical guide*. Pearson Education, 2001.

[34] Salvatore Settis. Le radici romane della tutela del patrimonio culturale. *L'Osservatore Romano*, page 1–5, Nov 2009.

[35] Rajeev Sobti and Ganesan Geetha. Cryptographic hash functions: a review. *International Journal of Computer Science Issues (IJCSI)*, 9(2):461, 2012.

[36] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. *arXiv preprint arXiv:2105.07447*, 2021.

[37] Ziyuan Wang, Lin Yang, Qin Wang, Donghai Liu, Zhiyu Xu, and Shigang Liu. Artchain: Blockchain-enabled platform for art marketplace. In *2019 IEEE Inter-*

national Conference on Blockchain (Blockchain), pages 447–454, 2019.

[38] Amy Whitaker, Anne Bracegirdle, Susan de Menil, Michelle Ann Gitlitz, and Lena Saltos. Art, antiquities, and blockchain: new approaches to the restitution of cultural heritage. *International Journal of Cultural Policy*, 27(3):312–329, 2021.

[39] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.

[40] L. Zagato, S. Pinton, and M. Giampieretti. *Lezioni di diritto internazionale ed europeo del patrimonio culturale. Protezione e salvaguardia*. Libreria Editrice Cafoscarina, 2019.