Leonardo Vicentini
Matteo Zanotto
Luca Vian

UNIVERSITÀ DI TRENTO

ProjectsChain

A new way to store, sell and buy CAD projects

github.com/vicentinileonardo/blockchain-project

Blockchain Course - Final project - Academic Year 2022-2023

Table of Contents

Executive summary

Some major obstacles arise in the daily activities of Freelancer Industrial Designers who also earn money by selling their designs and related assembled products online. Indeed, building a strong online presence in order to get higher visibility and design high-quality projects are fundamental tasks that they need to face in order to be successful. If the first task is tackled by many dedicated stores present online, the second one could be eased by giving the possibility to build their own project using, as components, other successful ones belonging also to other authors. This proposal, by the way, has not a trivial implementation due to the fact that exposing others' projects in order to be used as components, could also end up in an intellectual property violation having, as a consequence, an unfair royalty distribution among all the designers indirectly involved.

ProjectsChain is a plugin developed in order to be integrated with already existing e-shops of industrial designs and is able to grant, using the public blockchain and through smart contracts, the intellectual property of the projects, even without keeping them secret, and a fair royalty distribution between coauthors of the same final project. The straightforward and transparent mechanism developed also will prevent non-payment and miscommunication risks, representing common scenarios among freelancers.

The smart contracts developed in the MVP are 4:

- Master smart contract, that acts as a proxy between the dApp and the other contracts.
- Access smart contract, in order to manage the possibility (limited in time) for the manufacturers of a project to assemble or implement it.
- ProjectNFT smart contract, in order to grant the intellectual property of a project and its sale.
- CustomChainlinkClient, used as an oracle by the ProjectNFT smart contract in order to allow, among other things, the upload of a project on IPFS.

The two main operations provided by the smart contracts (store and sell a CAD project) can be used through the payment of a commission, which can be fixed or variable depending on the functionality itself, to an address specified during the contracts' deployment.

By looking at the context of the solution proposed, many efforts have been dedicated to gas fees optimization, to the access control of the contracts, to the testing phase and the effortless UI of the dApp.

Context of the scenario considered

Problem and target segments

Thanks to the widespread availability of the internet, the number of freelancers across various fields has rapidly increased, also because their job has become accessible to a larger number of potential customers. Today, indeed, there are numerous freelance opportunities available online, including roles such as developers, designers, writers, translators, and more [1]. The ability to choose how, when, where, and with whom to work is just one benefit of these types of jobs. On the other hand, this "independence" can lead to other issues, the main one of which is weaker legal support, which has a fundamental role, instead, even in small companies.

A weaker legal support can bring many serious complications, especially in works that are very creative. Indeed [2,3]:

- It's difficult to enforce the intellectual property related to one's own works.
- There could be miscommunication between collaborating freelancers, the customers, and with other figures involved in the process.
- There could be the risk of non-payment.

Another serious issue [2,4] is related to building a network and a strong online presence. If those steps are indeed fundamental in order to attract new customers and in order to find collaborations and partnerships, they still require a lot of effort and time that, otherwise, would be focused on developing other new creative works.

Among all the freelancers, the target segment identified it's related to Industrial CAD designers, so to designers that use computer software such as Autocad, Solidworks, and Systemes in order to create industrial blueprints that are mainly in 3D. Many of them produce general-purpose drawings whose physical realization can accomplish the needs of multiple customers not known from the former.

The emphasis was given to this specific scenario because, even if there are different competitors in the market that try to ease their identified specific needs, none of them seems to tackle all their problems together, that instead have the necessity to be accomplished by a single platform and in a coherent way in order to be fully overcomed. In addition, the big majority of those platforms are in web2 which doesn't represent a good choice in order to own digital assets (so, in order to maintain the intellectual property) or to establish bonds and duties between a designer with his customers, collaborators, and manufacturers (the ones who realize a physical implementation of the digital projects).

Moreover, these bonds must not be seen as a way to only enforce the designer's rights but also in order to accomplish the ones of the buyers and of the manufacturers themselves. Indeed, solutions that have not a multi-sided business model that takes into account also their needs, won't be successful in order to accomplish the necessity of the designers of building a strong online presence and a strong network neither.

Here is the extended segmentation defined, focused on the designers' segment.

Solution proposed

As it was said, there are already different solutions in order to tackle specific problems identified even if no one seems to consider all of them together. Indeed, there exists a dApp focused on enforcing intellectual property through the use of the blockchain (for example cadchain.com) while other platforms are instead focused solely on designing and selling online CAD projects or their physical implementation (cgtrader.com, vention.io).

The proposed solution aims to address the limitations of other approaches by offering a **plugin**, called "ProjectsChain", that can extend existing 3D industrial engineering platforms, such as CGTrader and Vention.io. These platforms currently lack effective intellectual property recognition for designers, as the only protection offered is the ability to keep design details hidden until a customer purchases them. This approach can hinder collaboration between designers, resulting in lower-quality designs and inadequate compensation. For instance, Vention.io has a "Community Design" section with a leaderboard but lacks an effective reward system.
The proposed solution seeks to promote transparency and collaboration, leading to higher-quality designs that can be fairly rewarded. This is accomplished by utilizing the (public) blockchain to ensure intellectual property recognition through the minting of an NFT belonging to the designer of the related project, as well as the use of smart contracts to define the bonds with which other designers can include other designers' projects as components in their own. Those latter bonds will be used in order to distribute the royalties in a transparent way once a project based on others is sold. Then, the use of blockchain will be also used in order to define transparent contracts between designers and manufacturers (in order to give also to the latter an adequate royalty distribution) and between designers and buyers (to avoid non-payment).

In other words, with our solution, the CAD designers will be able to store and sell their projects, which can also be composed of different subprojects (components) developed by other designers. The manufacturers can pay a certain amount in order to be able to access a certain CAD project and, although not yet implemented in the MVP, will get the legal right to print or assemble it for a certain amount of time. Also, the final buyers of a certain product, interested in the physical implementation of the associated project, will be able to buy it through the plugin developed. All the different prices involved (the one in order to get the manufacturers the right of producing a project, the one with which a physical product will be sold, and the economical conditions with which a project can be used as a component in another one) will be defined by the designer himself of the project considered.

So, here are the main functionalities implemented in the MVP of ProjectsChain:

- Possibility for the designers to store their CAD Projects, encoded as JSON files containing the geometrical information about the CAD themselves and the already existing third-party CAD components that they use. This is done by interacting with a form in which it's possible to select a specific JSON file and to fill in some related fields such as a title, a description, a price, and a royalty price.
- Once a project is uploaded, an associated NFT is minted specifying the author's wallet address, by paying a fixed commission. Through the tokenId of the NFT minted, it is possible to get much of the previously specified information, including the reference to the IPFS URL referring to the JSON project itself. This NFT will enforce the intellectual property of the Author for a project. This will solve the previously identified issue related to the difficulty for freelance designers in claiming their intellectual property rights for their creative works.
- The manufacturers and the buyers, that behave in the same way in the MVP, have the possibility to grant the access to a project for a limited amount of time (3 months) by paying the price of the project and the royalty prices of all the components that belong to it, together with an extra 5% of the previous amount for the plugin/platform commissions. Once a project

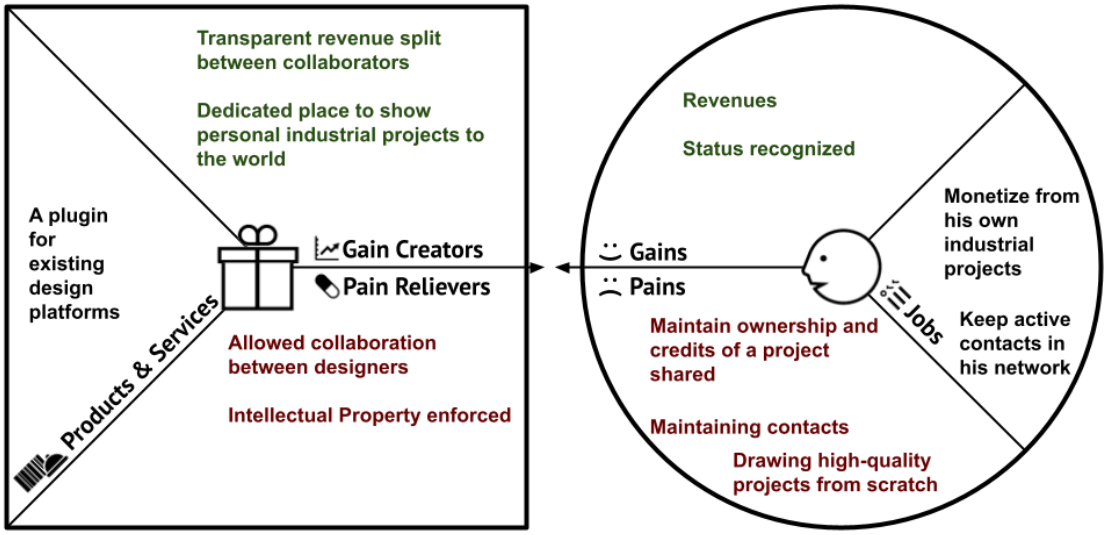
is bought, it will appear in their wallet together with some previously hidden details, such as the JSON representing it. This will solve the well-known issue related to non-payment and miscommunication between a designer with the manufacturers and buyers, ensuring also different rights for all the parties involved in the process.

The MMP (Minimum Marketable Product) will introduce a better distinction between the manufacturers' and buyers' necessities to be accomplished through the plugin proposed that, instead, the MVP flatten.

Value Proposition Canvases

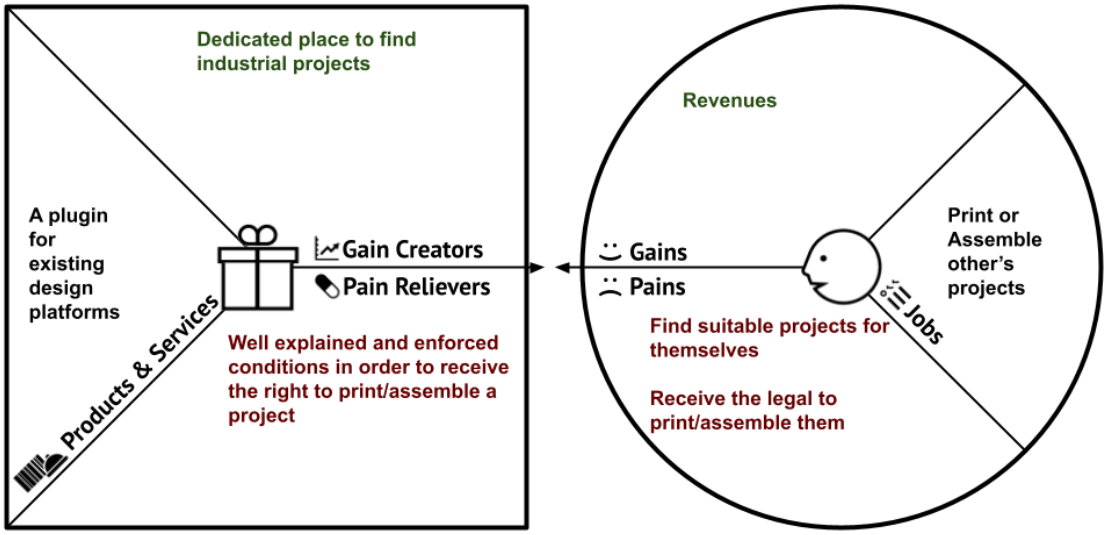
Due to the fact that both the problem identified and the solution proposed are multi-sided, there are three distinct value proposition canvases.

Value Proposition for Designers

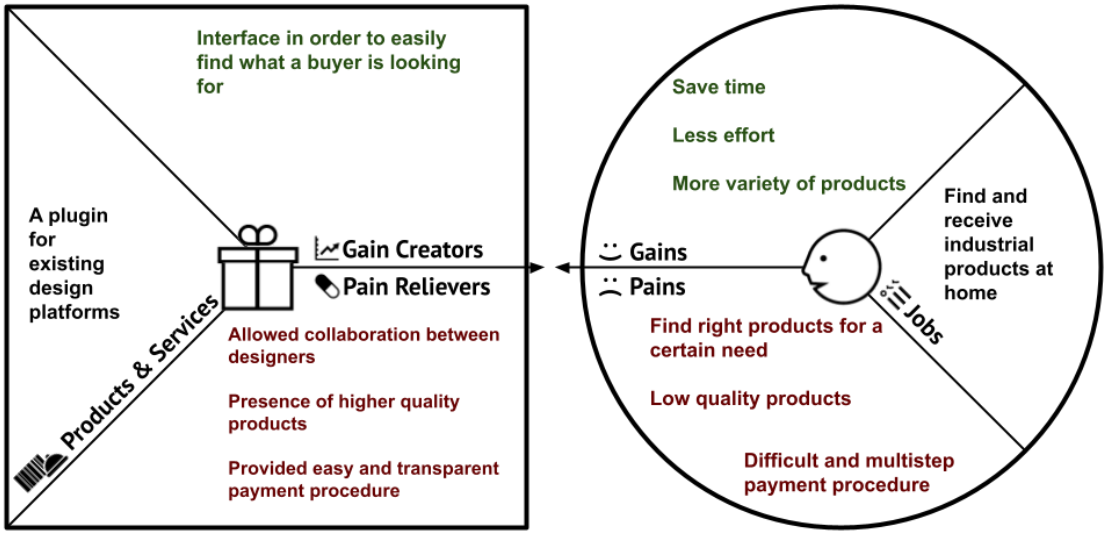


Value Proposition | Customer Profile

Value Proposition for Manufacturers



Value Proposition for Buyers



Business Model Canvas

<table>
<tr>
<td rowspan="2">Key Partners

Accountant

Legal Expertise

Public Blockchain

<u>AND POSSIBLY</u>

Industrial design platforms</td>
<td>Key Activities

Development and maintenance of the plugin

Maintaining relationships with partners and customers

Enhancing the designers' intellectual properties in legal scenarios

Customer service

Marketing</td>
<td rowspan="2">Value Proposition

Intellectual property enforced properly

Collaboration and revenue split made simple and transparent

Easily monetize from your own creativity

Finding and receiving the projects you are looking for in few clicks</td>
<td>Customer Relationships

Through the plugin website and through targeted platforms. Both will include (not in MVP) also a dedicated customer service chat.</td>
<td rowspan="2">Customer Segments

The segments composing the multi-sided business model are:

Industrial freelance CAD designers: who want to increase their revenue

Manufacturers: who want to find a new project on which to monetize

Buyers: who want to find a product that satisfies their needs

<u>OR</u>

Industrial design platforms</td>
</tr>
<tr>
<td>Key Resources

Human resources: developers, cybersecurity experts, legal expertise, help center.

Physical resources: offices, hardware for development. Leveraging cloud services for the deployments of some off-chain components of the product</td>
<td>Channels

Dedicated Website

Targeted platforms where the plugin will be used</td>
</tr>
<tr>
<td colspan="3">Cost Structure

Employees salaries

External Accountant Legal Expertise salaries</td>
<td colspan="2">Revenue Streams

The designers of the platform will pay a fixed commission in order to protect their intellectual property through the blockchain solution.</td>
</tr>
</table>

Software maintenance Cloud computing costs (hosting, …). These costs are present because part of the solution (plugin) is off-chain. Blockchain costs (gas fees on contracts deployment)	Manufacturers and buyers, once they pay in order to get the possibility respectively of assembling a product or getting its physical implementation, will pay a commission dependent on the cost of the project involved. OR By selling the plugin to industrial design platforms

The business model is straightforward: we offer a **blockchain-based plugin** that helps designers and manufacturers monetize their job and creativity by preserving intellectual property and to allow, in a transparent way, projects collaborations and revenue distributions. The customer segment of the plugin, whose problems were identified, represents a very niche market since it's aimed exclusively towards Industrial freelance CAD designers and towards Manufacturers and Buyers of the industrial projects stored. In the MVP the Manufacturers and Buyers are considered as a single customer entity interested in granting access to projects by paying a project-dependent amount. The channels used in order to communicate with and reach the customer segments regard the website of the plugin and the platform where it will be integrated. Through them, it will be both possible to let aware the users of our products and related news. To establish a relationship and to get in touch with our customers the main platforms will be, also in this case, the plugin website and the one of the platforms that are using the plugin itself. Both of them will have a dedicated customer service chat that will refer to the same contact center, able to help the customers in using the plugin and to answer their plugin-related doubts.

Among the key partners, we included figures such as an accountant and a legal expertise, fundamental in order to take care of all the financial matters of the company and to give additional legal support to the value of the intellectual property granted by the plugin respectively. Also the public blockchain, which will have the role of mining and storing the transactions, belongs to our key partners.

The key activities of our company are related to the development and to maintenance of the plugin thanks to the internal software developers of our company itself. This activity, together with the need of maintaining relationships with other partner industrial platforms, is the most important one due to the fact that a plugin of this kind, if not well developed or maintained, cannot guarantee to overcome the customers' needs in the way it promises. Providing the customers' legal support will sustain, in case of debates, the value of the smart contracts signed and the ones of the NFTs minted, in order to enforce their responsibilities and rights given by the plugin itself. Side activities, in order to establish and maintain a good customer relationship, reside in the marketing and in customer service of the company.

The key resources consist of all the human resources involved in the process (developers, cybersecurity experts, legal expertise, help center) together with the physical ones needed in order to develop, maintain and deploy the plugin (offices, hardware for development, cloud services for the deployments of some off-chain components of the product).

The costs reside in the salaries of the internal and external human resources, in the ones related to the physical resources, and in the costs needed to deploy the contracts on the public blockchain (the gas fees for the transactions will be paid by the users).

The revenues, instead, are strictly dependent on the single industrial design platform to which the plugin will be proposed.

Small platforms may not be able to sustain the costs associated with purchasing the plugin. In this scenario, it may be beneficial to propose to them the plugin for free in order to improve the quality of their service and traffic. In exchange for a **revenue-sharing scheme**, the plugin's commissions (0,00059 ETH for minting an NFT representing a project and 5% of the designers' earnings on the sale of their own projects, final amounts TBD) will directly reflect the developing team's revenue. The designer does not have to pay the royalties and associated commissions for his own components if he is interested in granting access to a project that incorporates one or more of his creations as components. Furthermore, in this scenario, the industrial design platform will represent an additional key partner.

In the case of large platforms, it may be more beneficial to **sell** the plugin straight to them with a large down payment. They can probably afford to pay for the plugin because they have a large user base, and they may be scared of the revenue sharing scheme in the long run.
In this situation, the developing team's revenue will be solely comprised of this sale, while the platforms will profit from increased service quality, traffic, and commissions paid directly to them by designers, manufacturers, and buyers. In this situation, the industrial design platforms engaged will be considered a customer rather than a partner in a more complex ecosystem.

The team strategy is to propose at first the plugin to big and known industrial design platforms, in order to earn in terms of plugin **visibility** and in **initial liquid assets** that can be used in order to sustain the initial and fixed costs of our company. Then, with the development of a more complete plugin and with the increase of our company’s EBITDA, it would be more profitable to propose the plugin for free to small companies, in order to have in the long run more cumulative reward given by the users’ commissions.

Technical Implementation of the MVP

Assumptions

The implementation of the plugin proposed relies on some assumptions related to the main platforms to which the plugin itself will be proposed and possibly integrated. Those platforms need to provide a way in order to convert the CAD files uploaded or drawn through them into a well-format and consistent JSON file, in a way that will keep track of the meaningful features (shapes) composing the projects. We assume that the platform itself will be able to automatically identify the components used by a given project.

Frameworks, Libraries, and Technologies

The proposed solution is a fairly complex system comprised of heterogeneous technology. It was kept in mind that other technologies or paradigms around should not have disguised the **critical role performed by the blockchain**, but rather should have strengthened and supported it.

The core of the solution are 4 Ethereum Smart Contracts, namely: ProjectNFT, Access, CustomChainlinkClient and Master. A detailed description of these will be provided in later chapters.

To manage the code compilation and migration of the smart contracts, a **Truffle** project was created from scratch.

CAD Projects metadata, represented as JSON files, are stored on **IPFS** (InterPlanetary File System), using the Moralis Gateway to perform the upload of the file.

A **NoSQL database** was used to store the token metadata. In particular, a docker-based Redis with the RedisJSON and RediSearch modules. The choice of Redis was made solely to learn how to use a new tool since the team already had experience with MongoDB. Since Redis out of the box does not persist data, a proper configuration was set up to achieve immediate persistence of data.

The data models chosen for the database and for IPFS are described in the appendix.

In order to perform CRUD operations on the data, a simple **Node.js server** using Express framework was created.

A docker-based **Chainlink node** was set up in order to fulfill a type of request later described.

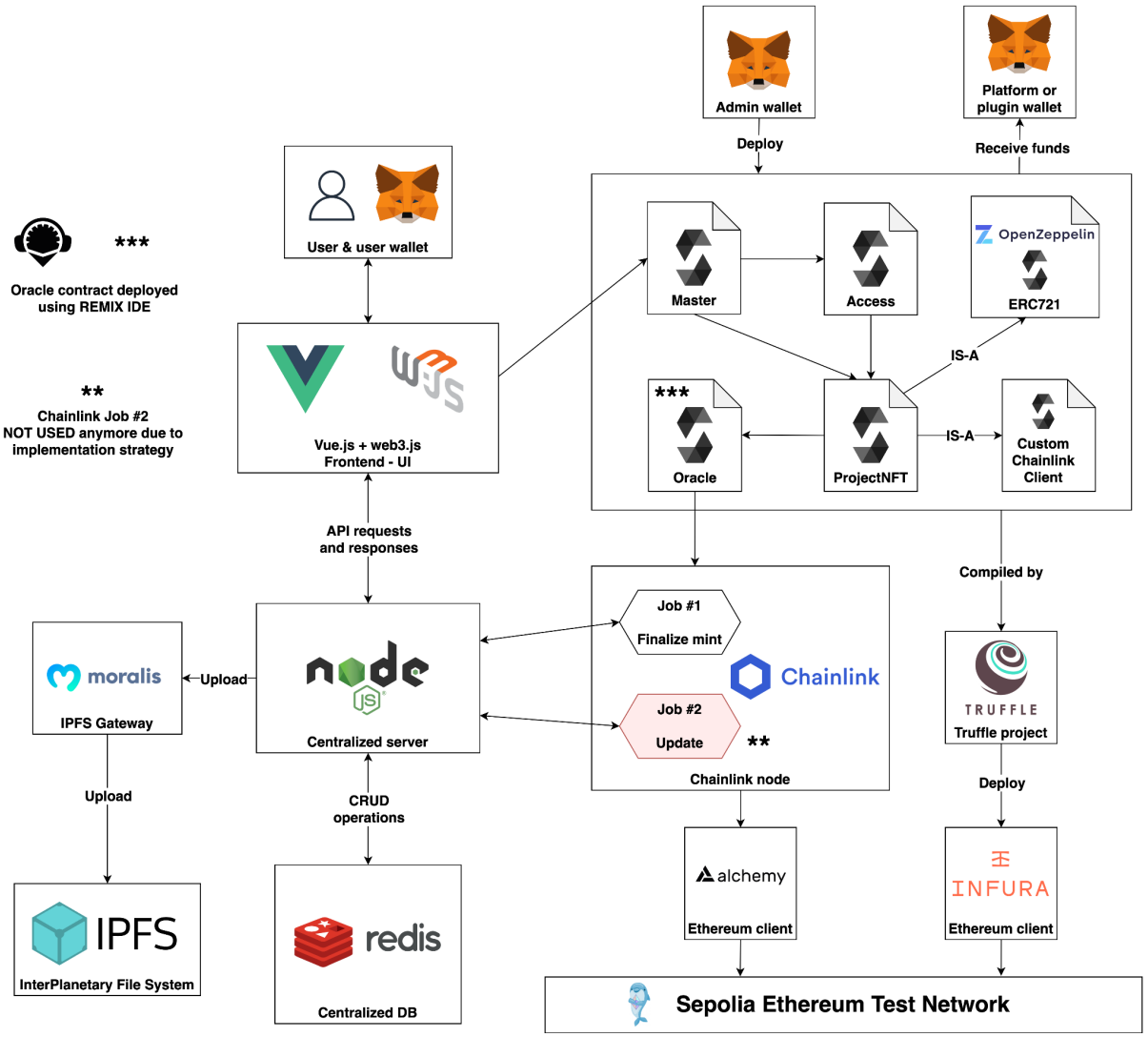
The user interface was created using the **Vue.js** framework.

A note on centralization

Since the solution has a catalog of NFTs, it was deemed hard by the team to devise an effective solution that would have been entirely decentralized. For instance, loading a catalog of items stored only on IPFS is quite problematic, without having an index of the available items.
Theoretically, an achievable solution might have been accomplished with more data loaded on the blockchain, but it would have been slower and more expensive.
Therefore it was tried to achieve a **correct balance** between the two worlds, giving much more importance to the blockchain, not only for didactic purposes but also acknowledging its strength.

Architecture Diagram

The following diagram describes from a high-level perspective the software solution devised.



Smart Contracts

Master

The Master smart contract was designed as a **proxy**, in order to be the only one able to call the mintToken() function of the ProjectNFT contract and the buyToken() function of the Access Smart Contract through access modifiers.
Furthermore, in order to increase the security of the Master contract and to avoid malicious behavior, it was decided to allow users to call its mintToken() method only through the conventional usage of the dApp's UI. This was accomplished by allowing mint requests only if they were accompanied by a request-dependent valid signature provided by the backend of the dApp that will be explained in detail in the subsequent chapters.

Additionally, the proxy approach allows for **upgradability** on the system, as the Master contract exposes the core functions of mintToken() and buyToken() that call the appropriate logic contracts, whose address can be changed in a transparent way for the user/dApp if updated logics are required.

ProjectNFT

The ProjectNFT contract represents a **Non-Fungible Token** as it was implemented to be compliant with the ERC721 Ethereum standard for NFTs. In particular, **OpenZeppelin** implementations were used by making the ProjectNFT inherit the **ERC721URIStorage** contract, as it provides a trusted and compliant foundation that can be extended with custom business logic for specific use cases such as ours.

It is critical to define in detail the **workflow** in which the most crucial smart contract is involved in order to illustrate the core logic of the software solution.

The ProjectNFT smart contract represents the digital assets of 3D CAD projects as an NFTs. It is the mechanism that enforces the **intellectual property rights** of a designer on its project as for each project upload the mint procedure is invoked so that a new non-fungible token is instantiated, unequivocally associating a project (represented as the **unique hash of its JSON representation**) to its **owner** (represented as the wallet address of the owner).

Additionally, the mint procedure also associates to a project its **price**, **royalty price** and **components** used so that it can be bought and integrated with fair compensation in other projects.
The mint procedure finally associates to the new NFT its off-chain file location as a Uniform Resource Identifier that points to our backend server which stores additional metadata of the project and a link for accessing the actual project JSON file on **IPFS**.
On the more technical side, the mint function will safely increment a counter that is used to extract token identifiers that identify each uploaded project: they are used as keys on private mappings to store the project's owner, hash, price, royalty price and components.

The mint procedure is a payable function as it also includes a fund transfer from the wallet of the designer that is uploading the project to the ProjectsChain administrative wallet for the commission fee for the project upload operation. The commission fee is set to a value of 0.00059 ETH per upload (approximately around 1€ at the end of May 2023). The method mintToken() is callable only through the Master contract, leveraging on the use of well-defined modifiers. This choice was taken in order to avoid some malicious users' behaviors (e.g. the possibility of minting a token by omitting, on purpose, the components that belong to it).

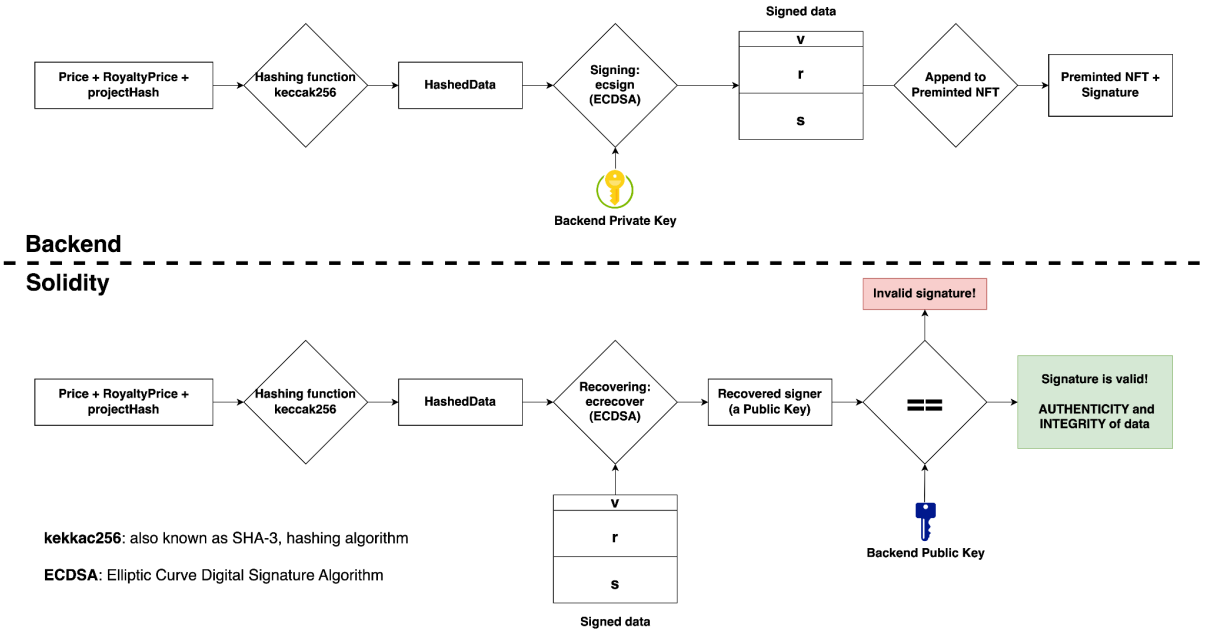
Pre-mint operation

The mint operation is logically preceded in the flow of the project's upload by a **pre-mint phase** that happens by an off-chain HTTP request to our backend: it performs **preliminary checks** on the project's upload, such as: verifying that no other file with such a resulting hash is already uploaded to IPFS, verifying prices meet the allowed range, etc. This approach can help to ensure that the minting process is likely to succeed, and can help to **avoid wasting gas fees** on failed transactions.
Once the pre-mint operation is successfully carried out, the metadata is stored on the server in a "preminted" status (not available) and the mint operation can be invoked on the smart contract. This step could have been exploited by a malicious actor trying to trick the system changing the parameter "on the way" to get inconsistencies to leverage.

Digital Signature mechanism

Therefore a **Digital Signature mechanism** was devised to assure that the pre-minted data correspond to the data used for the on-chain minting.
A signature is created at the end of the pre-mint operation on the backend using a standard signing function (**ecsign**). The function takes as input the data to be secured (price, royaltyPrice, projectHash) and a **private key** of the same type as the ones corresponding to wallets. The function, according to its signing algorithm, returns the signed data divided into 3 parts.
Therefore the proxied mintToken() function of the Master contract, in association with price, royaltyPrice and projectHash, takes as argument also the previously signed data. On the Master contract, the backend **public key** is stored as an instance variable.
If the provided data is not corresponding or the signed data is not original (handcrafted), the smart contract execution flow will lead to a transaction revert due to a **detected invalid mint request**. On the other hand, if the recover function (**ecrecover**) outputs the correct backend public key, it means that the data was signed by the backend and the data match. Therefore with this mechanism, the **authenticity and integrity of the data** are achieved.

The entire procedure is illustrated in the following diagram:



The coordination problem

The **trade-off** of leveraging the benefits of a pre-mint operation is that a lack of coordination between the “two worlds”, decentralized and centralized, can lead to problems of inconsistencies.
In order to cope with this kind of problem, once the minting operation was completed on-chain, a communication with the server was needed in order to signal that the token was minted and ready to be displayed on the user interface of the application.
The naïve solution adopted in the first place was performing a HTTP PUT request once received the “New Token” event from the blockchain on the frontend, in order to modify the status of the token to be available. Because the code on the frontend is available to everyone, this approach could lead to security vulnerabilities. Therefore, in a second iteration, it has been decided to leverage a powerful paradigm: an **oracle**.

Leveraging the paradigm of the oracle using Chainlink

The oracle design pattern needed for solving the problem in this alternative way is the **request-response oracle** and in particular **Chainlink** was chosen to implement the solution of this task. As a matter of fact, the ProjectNFT inherits the interface of CustomChainlinkClient, which were designed particularly to fulfill the contract's needs. This allows for a more secure request because it is sent within the smart contract rather than via the frontend code. Details regarding the use of Chainlink will be discussed in a separate section.

Hash of the project

By storing on-chain the hash of a project's JSON representation, it was possible to uniquely identify (with "require" Solidity checks) whether such a project was already uploaded: this allows to instantly detect and prevent malicious behaviors of users uploading projects originally created by other designers to sell as their own, effectively enforcing the intellectual property of a designers on its works. Simultaneously, this also provides a compressed representation of the project as a string so that it can be more efficiently stored.

Buy operation

The ProjectNFT contract also includes a procedure for correctly attributing the compensations and royalties for projects purchases. This is a payable function that is invoked at the end of the buy process by the Access Smart Contract and will transfer funds sent with the transaction of the buyer to the wallets of the project and components owners, in the amounts specified during the mint operation. The Access Smart Contract is, moreover, the only one able to call directly this transferPayment function through the use of well-defined modifiers. Additionally, this procedure transfers funds from the transaction to the ProjectsChain administrative wallet for the 5% commissions on both the prices of the project and of all the components that it includes. If the caller of the buy operation is also the designer of the project or of one of its components, the amount and the commissions proposed to him to pay won't include also the prices related to his own works.

Access Smart Contract

The Access Smart Contract handles a project's purchasing procedure and enforces buyers' access rights. It makes use of mappings to conveniently store information about which user purchased which project and the time validity of that purchase.

It exposes the buy function, invokable only through the Master contract through the use of modifiers, which is a **payable** function that handles the procedure of purchasing a project NFT: when invoked, it will first check to verify if the user has already purchased the given project, and will then call the transfer payment function on the project NFT for the specific project (identified by the tokenId), along with all funds sent with the transaction, so that the project price and royalties can be paid. Once the payment is completed successfully, the ownership information on the mappings will be created by binding a specific address to the token id that it has purchased, as well as the expiration date of this ownership, which is set in three months from the current block timestamp. Finally, it returns the project buyer's address to notify the Master contract that the operation has been completed correctly, enabling it to emit the NewBuyer event.

The contract also exposes a public view function for retrieving the project NFTs that a certain user has bought. It fetches the current supply of project NFT tokens (the current value of the token id counter) and checks whether the mappings for the user address on the NFT id has a true value and that such ownership is still valid on the current block timestamp. It will then return an array that at each index

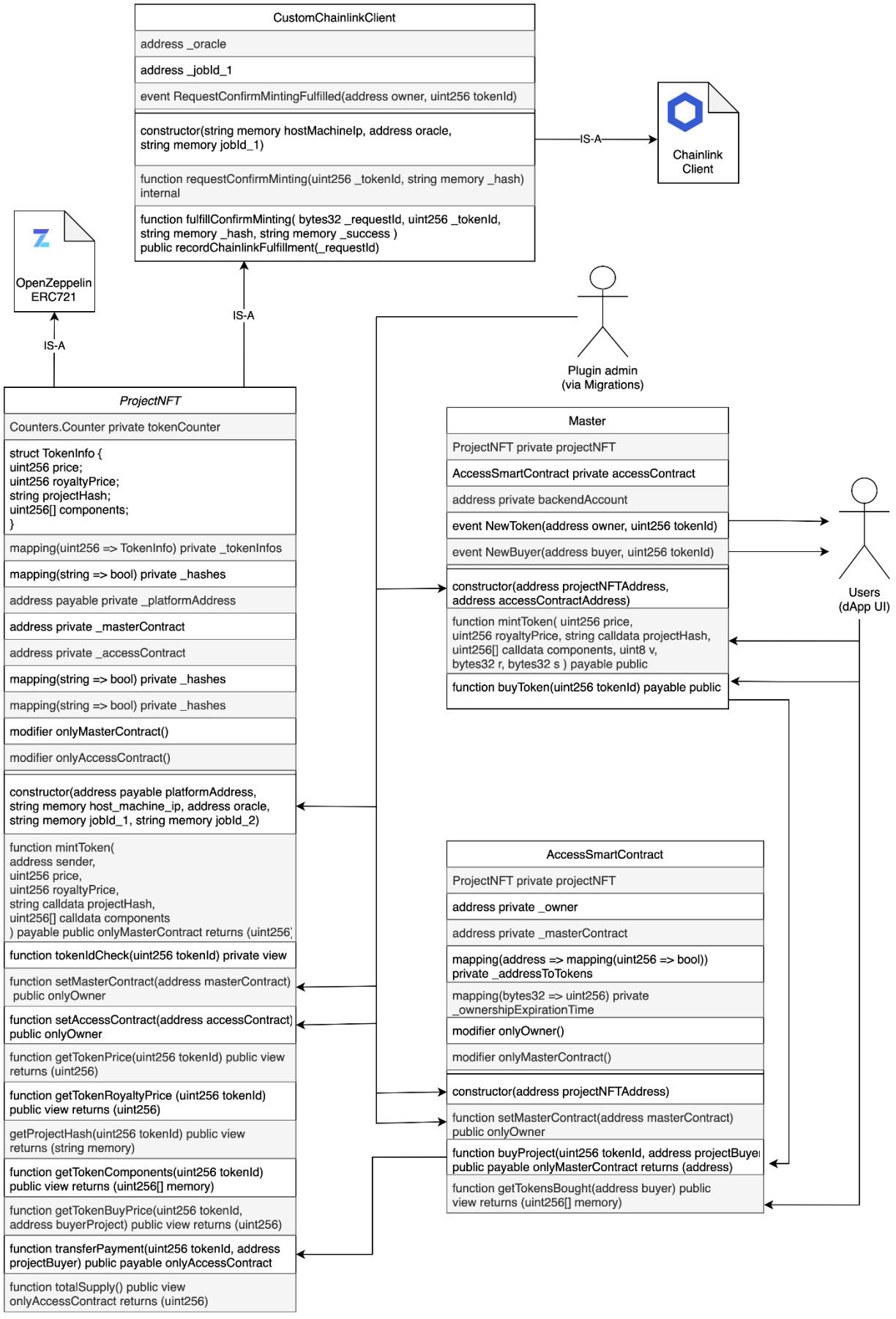
(that corresponds to the project NFT with that id) contains tokenId or 0 depending if the user owns such project NFT or not.
This method is used to enforce an **authorization** mechanism as it allows to determine on the backend whether to show or not the project JSON and IPFS download link on fetch requests for projects: the backend will check the ownership of the projects based on the user's address by interacting with this contract through the Web3.js library.

Custom Chainlink Client

In order to fulfill Chainlink requests, a dedicated smart contract was developed, starting from the Chainlink documentation guidelines. This smart contract is inherited from the official ChainlinkClient interface.
The first thing done was setting up a local **Chainlink node** that will be called by a standard **Chainlink oracle**. The oracle's job is straightforward because it is simply a broker or middleman between the smart contract making the request and the Chainlink node. Fulfilling external requests is done by defining **Chainlink jobs**, which are a list of subsequent tasks.
As a matter of fact, these tasks are often referred to as task pipelines and can be represented by a DAG (Directed Acyclic Graph). A task pipeline is defined through a TOML file which sets the order of the tasks, their type and their corresponding inputs and outputs. Often the input of a task is the output of the previous one.
In particular, a Job was created to make a HTTP PUT request to the backend server to signal the successful minting from the smart contract in order to make available the item on the dApp catalog. Therefore one of the tasks of the pipeline is of type "http" and it is used to send the actual HTTP request to the server endpoint while the subsequent task is of type "jsonparse" for the parsing of the server response. The last task submits a transaction and, on the smart contract, a callback function that emits an event representing the correct update, is activated.
The Chainlink network is based on the LINK token which is used to reward oracles that fulfill clients requests. ProjectNFT, being the client, needs to be funded with LINK on the initial setup.

Smart contracts UML

It must be noted that, for simplicity, the UML also contains the “actors” involved in the use of smart contracts.



Smart Contract Deployment

In order to deploy the smart contracts described, a **migration** file was implemented following the Truffle standard. Because all the contracts are **dependent** on each other in a bidirectional way (e.g. the Master contract needs to know the ProjectNFT's address in order to call its own mintToken method, and the one of the projectNFT itself must be callable only by the Master contract), it was necessary to deploy the contracts **sequentially**, specifying in the constructor of the latter deployed the address of the previous ones. Then, in order to achieve this bidirectional dependence, it was also necessary to specify in the first contract deployed the address of the next ones through a well-defined **setter** callable only by the owner of the specific contract considered, so by the wallet address doing the migration.
The contracts were deployed into a **Ganache** local network for the first part of the development, but in order to interact with a Chainlink Oracle, the deployment was shifted to the **Sepolia** testnet (since Ganache is not supported by Chainlink).

Testing

The testing phase of the developed smart contracts, and in general of the entire plugin, was seriously taken into consideration also due to the legal importance of the scenario considered.

End-to-End testing

During the development of the smart contracts and dApp, it was chosen to perform manual testing of every single functionality developed in order to find and rapidly fix the main vulnerabilities. It was chosen to perform first the **manual End-to-End testing** because of the high number of errors present in those initial phases of the development, which testing in an automated way would result to be more time and effort expensive due to the necessity of updating frequently both the code, both the automated tests themselves.

Test suite

Once the development phase was nearly complete and the interface used to communicate with the smart contracts was not subject to planned future changes, it was decided to create a JavaScript-based **test suite** using the **Truffle** environment and the Mocha and Chai libraries. Running those tests during this phase allowed the team to examine edge cases rapidly, repetitively, without human intervention, and without the need of further updates.
It was decided to rely on JavaScript tests instead of the Solidity ones because of the entire way with which the contracts were defined (only callable through the dApp due to the digital signature mechanism), bringing their tests suitable through the Web3.js JavaScript library, such as the contracts methods were manually invoked by a real user of the plugin.
The tests were defined in 3 test files, one for each contract developed, and through each one of them were tested the methods belonging to the related contract, giving emphasis both on the **successful cases**, both to the ones that should **revert**. Due to the fact that some methods of the smart contracts were protected by a modifier in order to be invoked only by another specific contract, it was necessary to test those methods by the test files belonging to the allowed caller contract in order to cover all the edge scenarios.

Smart contract optimization

Some code size problems were encountered at a certain point during the development. As a matter of fact, since the Spurious Dragon hard-fork of 2016 [5], a **limit of 24 KB** has been imposed by the

EVM. Therefore some actions were put in place with the goal of reducing code size and also gas costs since size is a factor for gas costs.
In the first version of the ProjectNFT smart contract 4 additional mappings were used to represent information about the token given a tokenId. A **struct** with that information was created and used in a single mapping. Some “require” statements were discovered to be repeated therefore a wrapper function was created and utilized.
Moreover, the Solidity compiler **optimizer** was enabled and the parameter runs was set to 1 so that even if a function is called once it will be optimized.

Web3.js integration

The Web3.js library was used both in the backend and dApp in order to interact with the deployed smart contracts through method calls and event listening. To set it up, the compiled contracts’ result JSON files were used for the appropriate method signatures and parameters encoding.

dApp User Interface

The dApp UI consists of a **Vue3** web application that interacts with both the backend through its RESTful APIs and the deployed contracts through the **Web3.js** library for **method calls** and **events listening**. The dApp connects to the user’s wallet through the **MetaMask** browser extension, from which it signs a **web3 token** that is passed in all the HTTP requests allowing for **user authentication**.

The dApp logic is encapsulated in dedicated state management structures, called stores. In particular, the application contains a store for the user’s wallet address, that is set up at the start of the application by requesting the user’s permissions to access the MetaMask wallet information (signing the Web3 authentication token), and another store that is used for interacting with the projects’ NFTs through both HTTP requests and Web3 method call operations: it allows to fetch and cache the NFTs and to carry out the minting and buying operations, from the user input and transactions to the acknowledgments from the blockchain; this store is also set up at the start of the application by loading the compiled ABI files of the smart contracts.

The dApp consists of the following pages:

1. **Homepage**: application starting point for other routes.
2. **Wallet**: allows the user to access the information about the projects that he uploaded as NFTs and of those that he has bought from other designers.
3. **Upload**: a form that allows the user to upload his project JSON and metadata for the mint operation.
4. **Catalog**: an “e-shop” section in which the user can buy projects uploaded by designers.
5. **Checkout**: a page for the checkout phase that triggers the buy operation for a certain project NFTs.

Security

Because of the scope of the problem identified, the security of the plugin developed has been considered a priority since the early development phases, requiring a solution that must be secure in order to grant the protection of the transactions and of the royalty distribution, and a strong legal safeguard of the intellectual property of the projects uploaded. Some of the implementation choices (already mentioned) used to enforce security off-chain and on-chain are highlighted and summarized below:

- Some smart contract methods are protected by **require()** statements, used into well-defined modifiers, that make them only callable through another smart contract and not directly (the goal is to give non-owners the ability to change the contracts state only through the Master one). This choice restricts users from utilizing more powerful lower-level functionalities in an uncontrolled manner.
- The use of **require()** is also used to avoid other undesirable behavior, such as:
    - operations on a non-existent project
    - possibility of purchasing the same project twice
    - possibility of uploading the same project twice
    - not using the Master contract through the dApp (not providing a valid signature)
    - not providing enough money to purchase a project
- **Web3 authentication tokens** were used to identify the user at the backend and hence not give them implementation information about projects they had not purchased or uploaded. Authentication tokens are generated if they are not already stored on the user's browser or if they do not match the active address provided by MetaMask.
- To force the use of the pre-mint operation on the backend, a **digital signature mechanism** was implemented. As a result, only data validated by the backend pre-mint operation can be used for the smart contract minting method.
- To signal to the backend the completion of the mint process by the smart contract in order to make the item minted available for purchase by users, the update request is sent inside the smart contract leveraging a **Chainlink oracle** rather than a simple forgeable request from the frontend.

Operating costs

After having deployed the Master, Access, and ProjectNFT contracts on the Sepolia testnet, some interactions were done with the methods mintToken() and buyToken() in order to understand more about the related operational costs.

According to the tests done, the operations have larger gas fees at Noon when the deployment of the 3 contracts costs an average of 0.012406 SepETH (if it were ETH, it would be around 21€ at the beginning of July 2023), and when the mint and the buy operations (on projects having 2 components) have an average gas fee of 0.001450 and 0.000616 SepETH respectively (in ETH would be around 2,50€ and 1,20€ respectively at the beginning of July 2023). In other time slots, especially during the evening, the costs are slightly lower on average.

Conclusion

Further Improvements

TransferPayment function pattern redesign

It would be interesting to explore the "**withdrawal**" pattern rather than the "**push**" pattern to assess costs and security benefits, assuming they exist. This means that instead of pushing payments to recipients, enable them to withdraw them in a controlled manner. This, however, would require a considerable redesign of the contracts.

Filters on catalog page

A filtering system should be implemented to make project search more user-friendly on the catalog page.

Better project deployment (containerization)

Given that the Chainlink node runs inside a Docker container, in order to communicate with a local server on a host machine (outside Docker), the Chainlink job needs to know the host machine IP address. The shortcut of attaching the containers to the host network unfortunately does not work on Apple machines.
For this reason, the host_machine_ip is a parameter that needs to be provided to the Master contract. To avoid this, spinning up the server as a Docker container and setting up a proper Docker network would allow the Chainlink node Docker container to reach the local server using simply “localhost” and the proper port number.

Final outcomes

In conclusion, the project's primary goal, which was to create an MVP capable of capitalizing on the main benefits of blockchain and smart contracts, was met. The team is satisfied with the solution proposed, able to reconcile in the MVP both basic and **general purpose** blockchain features, such as the enforcement of intellectual property rights through the minting of NFTs, as well as more **context-dependent functionalities**, such as royalty distribution on the sale of a project.
The way in which the project was introduced, defined, and implemented allowed for a deeper understanding of the blockchain's benefits and drawbacks. The curiosity and enthusiasm highlighted in developing a solution based on a real-world case study allowed for a pleasant learning, deeper dive, and application of various new technologies covered during the course.

Team

The team, composed of 3 computer science students having different backgrounds and skills, decided to use the **agile development** approach in order to increase flexibility and collaboration in all parts of the process. During the first two weeks, the problem to be addressed, the solution to be proposed, the technical stack to be used, and the way to arrange the codebase were all agreed collectively. Then, the formalized functionalities were divided into subtasks and each one of them was assigned to each team member, according to his own skills and interests.
Particular care was put to equally distribute the tasks strictly blockchain-related since it was deemed fundamental especially for educational purposes.
The substeps identified were faced, once at the time, during one-week iterations with the care of updating the entire team about each own progress through **daily scrum meetings**, useful also in order to expose personal doubts or implementation issues. When facing particularly challenging tasks, for instance solving a subtle bug, **pair programming** was adopted.
The following paragraph formalizes the personal contribution of each team member.

Personal contribution

Luca

- He developed the main business considerations presented in the report.
- In a coherent way to the business model proposed, he developed, in the smart contracts and in the dApp, the royalties distribution mechanism between designers and the payment of the commissions.
- In the dApp, he contributed to web3 authorization token instantiation and use in the endpoint protection. In the preminting phase, he contributed to the IPFS ping and upload.
- He contributed to the development of automatic javascript tests.

Matteo

- He contributed to the initial high-level smart contracts design
- He implemented the Access smart contract and the buy token procedure, together with the compensation and royalty distribution mechanism on the NFT contract.
- He was responsible for the design and development of the dApp in Vue.js and Web3.js, integrating it with MetaMask, the backend and the smart contracts.
- He carried out end-to-end testing from the dApp to the backend and smart contracts to validate the system in its integrity and the flow of operations from the user perspective.

Leonardo

- He was responsible for the design and implementation of a web server and related RESTful APIs that performs CRUD operations against a properly configured NoSQL database (Redis), which are critical for the pre-mint operation, IPFS upload and catalog fetching.
- He proposed and implemented the Digital Signature mechanism into the server code and into Master Smart Contract.
- Was responsible for the Chainlink node initial setup and jobs configurations together with the CustomChainlinkClient smart contract development and integration with ProjectNFT.
- He contributed to the initial high-level smart contracts design together with final smart contract optimization and testing

References

[1] https://www.indeed.com/career-advice/pay-salary/highest-paying-freelance-jobs
[2] https://n26.com/en-eu/blog/what-is-freelancing
[3] https://qonto.com/en/blog/freelancers/tools-and-tips/best-practices-for-collaboration-between-a-company-and-a-freelancer
[4] https://aicontentfy.com/en/blog/importance-of-strong-online-presence-for-freelance-writers
[5] https://ethereum.org/en/developers/tutorials/downsizing-contracts-to-fight-the-contract-size-limit/

Appendix

Customer Segmentation

DESIGNERS	
Market Density	There are very few companies or businesses offering in order to solve their necessity of stronger legal support, to have better visibility of their works, and that allows a collaboration between them.
Market Size	There are 15,702 industrial designers currently employed in the United States, the country with the most number of industrial designers worldwide (source).
Educational level / Previous knowledge	Should have proficiencies with the basic use of the internet and technology.
Age	Any.
Gender	Any.
Income	Any. The commissions in order to protect their own works are very low.
Occupation	Must be a freelance industrial designer.
Belief	Must see IT innovation as a way to make life easier.
Benefits	The benefits searched regard the ability to expose more of their products by still being able to have strong control over them.
Quantities	There must exist the possibility to upload and sell a potentially unlimited amount of products for each designer.
Readiness	The solution proposed is a disruptive technology compared to the other solutions nowadays. Many designers could have the need to use it immediately.
Attitude	Propositive attitude in order to substitute an already frequently used product, and willing to learn to use a tool that is going to improve their personal working life.

MANUFACTURERS

Market Size	There are many companies or businesses that offer the possibility of printing or assembling others' (industrial) projects. Industrial assembly and print represent a strong reality in the age of Industry 4.0.
Income	Should be able to have initial capital in order to grant access to a project and to have the possibility to print/assemble it. Once this is done, they can monetize from it.
Benefits	The solution should help them in increasing transparency in the process and easier accessibility to the projects.
Quantities	A project must be "bought" multiple times by the manufacturers in order to have granted the right of printing it over a large period of time. Then, the project can be sold to the buyers an unlimited amount of times within this period.
Readiness	Manufacturers have some other lucrative solutions different from the proposed one. For this reason, there couldn't be an immediate necessity to get this service.
Attitude	Propositive attitude in order to substitute other already frequently used solutions.

BUYERS

Market Size	The number of industrial products demanded every day is very high from both the ones interested in a specific final product, for both the DIYers that need only a specific component.
Educational level / Previous knowledge	Should have proficiencies with the internet and technology.
Age	Any.
Gender	Any.
Income	Any. The variety of industrial products is very high and the related prices can vary a lot.
Occupation	Any.
Belief	They must see these kinds of online shops as a consolidated alternative o get better products, cheaper and more personalizable
Benefits	Should help the buyers in getting directly at home the products that they want, characterized also by a higher variety and quality.
Quantities	Any.
Readiness	There are many alternatives in order to buy industrial products online in a similar way from the buyer’s point of view. For this reason, there couldn’t be an immediate necessity to get this product.

Data model

It is useful to show the data model used in the application logic. On the centralized database, the token metadata are a JSON object of the following shape:


```
"nft": {
        "status": "minted",
        "tokenId": 100,
        "name": "sfcsfsf",
        "description": "sdsdsd",
        "price": 1000000000000000,
        "royaltyPrice": 1000000000000000,
        "owner": "0xB79F3D23E01976eEB72e66C9178f592C76Cf73DD",
        "hash": "QmV9ZJiW4DJH1LbTUsug3uh9fUWzwakdTEXuDcL7nP7zxW",
        "imageLink": "project2.jpg",
        "ipfsLink": "https://ipfs.io/ipfs/QmV9ZJiW4DJH1LbTUsug3uh9fUWzwakdTEXuDcL7nP7zxW",
        "projectJSON": {
          "point": {
            "x": 2423232323121212,
            "y": 2.23232321212123,
            "z": 2.3224243231212
          },
          "components": [1,42,77]
        },
        "signature": [
          "27",
          "0x4393cea29524bc50af89b43fd9858434d71522db810f60e1e662f3ad5c9e4b73",
          "0x4f03bf62ac3b59044f5b12fcb398675e59c3f507c000e1ac0c04ce5ae3f16863"
        ]
      }
```


The metadata saved on IPFS has the following shape:


```
{
        "point": {
            "x": 2423232323121212,
            "y": 2.23232321212123,
            "z": 2.3224243231212
          },
          "components": [1,42,77]
}
```


It should be noted that the keys preceding "components" might be any number and have any name.

For development and testing purposes, the “projectJSON” key is present also on the model saved on the database. Ideally, in a production environment, with very large projects encoded into JSON, that field will be removed.