



WHITE-LABEL SHOP FOR DIGITAL INTELLIGENT ASSISTANCE AND HUMAN-AI COLLABORATION IN MANUFACTURING

Title	D3.7 – NFT royalty computation & federated shop integration
Document Owners	Georgios Lagos, Fotis Paraskevopoulos
Contributors	ICCS
Dissemination	Public
Date	30/07/2024
Version	1.0



Co-funded by the Horizon Europe programme
of the European Union under Grant Agreement
N° 101092176

VERSION HISTORY

Nr.	Date	Author (Organization)	Description
0.1	02/06/2024	Giorgos Lagos (ICCS)	Deliverable structure
0.2	12/06/2024	Giorgos Lagos (ICCS)	Initial Draft
0.3	25/06/2024	Fotis Paraskevopoulos (ICCS)	Draft Review – Changes
0.4	05/07/2024	Giorgos Lagos (ICCS)	First version for review
0.5	10/07/2024	Fotis Paraskevopoulos (ICCS)	Introduction / Document Review
0.6	20/07/2024	Alessandro Canepa (IDEAL)	Review
0.7	21/07/2024	Arno Cuypers (KU Leuven)	Review
1.0	27/07/2024	Giorgos Lagos (ICCS)	Final Draft

DISCLAIMER

This document does not represent the opinion of the European Commission, and the European Commission is not responsible for any use that might be made of its content. This document may contain material, which is the copyright of certain WASABI consortium parties, and may not be reproduced or copied without permission. This document is supplied confidentially and must not be used for any purpose other than that for which it is supplied. It must not be reproduced either wholly or partially, copied or transmitted to any person without the authorisation of the Consortium.

ACKNOWLEDGEMENT

This document is a deliverable of the WASABI project. This project has received funding from the European Union's Horizon Europe programme under grant agreement N° 101092176



CONTENT

1. EXECUTIVE SUMMARY	4
2. INTRODUCTION	5
2.1 Purpose and scope of the deliverable	5
2.1.1 Purpose	5
2.1.2 Scope	6
3. Smart Contract	7
3.1 Environment Setup	7
3.2 Code analysis.....	7
3.3 Compilation and deployment.....	10
3.4 Testing.....	10
3.5 ABI Extraction.....	11
3.6 Gas fees on Ethereum and the NFT Royalty Module	11
4. White Label SHop (prestashop) integration	12
4.1 Environment setup.....	12
4.2 Admin Dashboard: Smart Contract deployment	13
4.3 Skill bridging into Web3: Products minted on-chain	15
4.4 Donation logging on-chain upon checkout.....	15
4.5 Admin Dashboard: Royalty payout.....	16
4.6 Author Dashboard: Royalty allocation on will.....	16
5. Demonstration.....	19
6. Conclusion.....	25
7. Bibliography	26



FIGURES

Figure 1 - End result: Multiple systems running simultaneously capturing royalties from all Marketplaces.....	12
Figure 2- Module Installation in the PrestaShop backend.....	13
Figure 3 - After successfully installing, one needs to configure the module.	13
Figure 4 - An overview of the Instance Owner dashboard.	14
Figure 5 - The Instance owner administrator dashboard.	14
Figure 6 - Royalty amount owed to Skill developers.....	15
Figure 7 - NFT skill minting by the Instance owner.	15
Figure 8 - An overview of the authors' dashboard.	16
Figure 9 - The Skill Author Dashboard.....	17
Figure 10 - Author-specific stats related to their skills and their royalties.	17
Figure 11 - The mechanism allowing royalty transfer from authors.....	18
Figure 12 - Extracting the Instance Owner private key.	19
Figure 13 - Instance owner public wallet address stored on database.	19
Figure 14 - Upon filling the private key and public wallet address the system can be deployed.....	20
Figure 15 - Deploying the Smart Contract Instance.	20
Figure 16 - Upon a successful Deployment a Smart Contract address will be shown.	21
Figure 17 - Navigating to the Skill minting backend Tab.	21
Figure 18 - The same product cannot be minted twice as an on chain skill.....	22
Figure 19 - Mint three distinct skills.	22
Figure 20 - Proceeding with minting the three mentioned skills.....	23

1. EXECUTIVE SUMMARY

The principal aim presented in the previous Deliverable has been achieved: the development of a Smart Contract in accordance with the ERC721 specification, which has transformed traditional marketplace items into Non-Fungible Tokens (NFTs). This system, known as the "Royalty Distribution Module," now enables the fractionalization of open-source developers' contributions in a manner that is both transparent and verifiable. Established on the Ethereum network, this sophisticated royalty management platform efficiently handles the rights and rewards associated with digital asset ownership. Skills are uploaded and tokenized into unique NFTs, serving as transferable digital assets that embody specific contributions. The system performs real-time royalty calculations and distributions using smart contract logic to satisfy dependencies and process transactions. Ownership transfers are executed securely and transparently through blockchain transactions, ensuring that all parties recognize the legitimacy of the transactions. The module integrates seamlessly with the outcomes of Task 3.1, enhancing the provisioning of skills as NFTs and the tracking of royalties.

The Marketplace Administrator functions as the Smart Contract owner, provided with administrative privileges. Skill Creators/Developers (Authors) are the beneficiaries of the royalty system, their contributions are fractionized and stored within the Smart Contract. The Royalty Distribution Module operates as a decentralized application (DApp) on the Ethereum network, characterized by its resilience, security, and constant availability, attributed to the blockchain's decentralized nature. This established system acts as a conduit from the web2 space to the web3 ecosystem, facilitating the minting of NFTs that encapsulate metadata about the digitized assets or skills, distinctly separating them from conventional cryptocurrencies.

The functioning Smart Contract is integrated within a specialized PrestaShop module, enabling straightforward interactions for administrators and authors alike, within the e-commerce framework. Having been successfully deployed, this decentralized application revolutionizes compensation methods for open-source developers by ensuring innovation and equitable royalty distribution. The use of blockchain technology guarantees a transparent, secure, and efficient mechanism for the tokenization of skills and management of royalties.



2. INTRODUCTION

The primary goal of Deliverable 3.2 “Smart contract specification” was to provide the specification of a software tool for managing developers’ royalties using blockchain technology. In this deliverable we provide details as to the implementation of the NFT- Blockchain Based Royalty Distribution Module (W3RDM), which is based on that specification.

The objective to create a Smart Contract on the Ethereum network has been realized, with the system now facilitating the conversion of conventional marketplace products into Non-Fungible Tokens. This conversion allows for the fractionalization of contributions by open-source developers, creating unique, transferable digital assets that capture royalties in an immutable and traceable form. The Royalty Distribution Module, now in operation, manages these royalties efficiently and is integrated with Task 3.1 to simplify skill provisioning and enhance royalty tracking.

The module is composed of two main subcomponents. The main component, which is a smart contract implemented using Solidity and deployed on the Ethereum network which contains the core logic of royalty tracking, computing and distribution. And a second component, in the form of a Prestashop¹ module, which provides an interface between Prestashop and the Smart Contract .

Throughout the implementation phase we made sure of several key factors which will allow the continued extension of W3RDM. Initially we implemented the contract following Solidity development best practices, using established libraries such as OpenZeppelin², allowing us to adhere to a set of ERC specifications such as ERC-721, ensuring the interoperability of the contract with the Ethereum ecosystem. Furthermore we compartmentalized our module development in such a manner, allowing us to seamlessly support other web2 environments, including storefronts such as Magento³, CS Cart⁴, or any web2 environments and content management systems.

As a future extension of the W3RDM, we will examine expanding the integration interface into other environments and paradigms, which will allow us to seamlessly track and distribute royalties, such as package management integration and GIT hooks.

2.1 Purpose and scope of the deliverable

2.1.1 Purpose

The completed development of the Smart Contract, aligned with the ERC721 specification, now tokenizes open-source developers' skills as Non-Fungible Tokens (NFTs). The "Royalty Distribution Module" provides a secure, transparent, and automated framework for capturing royalties generated from the use of these skills.

¹ <https://prestashop.com/>

² <https://www.openzeppelin.com/>

³ <https://about.magento.com/Magento-Commerce>

⁴ <https://www.cs-cart.com/>



2.1.2 Scope

The Royalty Distribution Module leverages blockchain technology to transform traditional marketplace items into Non-Fungible Tokens (NFTs), ensuring transparent and verifiable compensation for open-source developers. This comprehensive system encompasses several key functionalities:

Skill Tokenization: Skills provided by developers are minted into unique digital assets or NFTs.

Royalty Management: These NFTs capture and distribute royalties to developers whenever their skills are utilized, directly or as dependencies.

Ownership and Transactions: The module ensures secure and transparent transactions via smart contract interactions on the Blockchain.

Administration and User Interaction: The system supports interactions from Marketplace Administrators and Skill Developers, each benefiting from unique functionalities tailored to their roles.



3. SMART CONTRACT

3.1 Environment Setup

Interacting with the Ethereum blockchain necessitates the use of the Solidity programming language. The development process was conducted in Microsoft's Visual Studio Code, with NPM packages installed to facilitate the compilation of Solidity code and the deployment of Smart Contracts. For blockchain interaction, an Ethereum-native wallet and a node connected to the blockchain are essential. In our case, we utilized a public node provided by Infura⁵. Account creation on Infura's website allows for the generation of an API key, which is critical for executing all Smart Contract operations, including deployment and integration into a white-label platform. It is recommended that users of this software obtain their API key for security purposes. Additionally, deploying the Smart Contract and invoking its functions require an Ethereum wallet with adequate funds in ETHER, the blockchain's native currency. For testing convenience, MetaMask, a widely-used third-party wallet, was selected. MetaMask simplifies the interaction with the Smart Contract through a front-end interface and enables the retrieval of private keys necessary for backend automation. The deployment was carried out on the SEPOLIA Ethereum Testnet, which requires test Ether for operational use. Test funds were obtained from faucet.org, a testnet faucet that necessitates account registration for its services. Utilizing a blockchain explorer like etherscan.io is beneficial for identifying and understanding errors in an accessible manner during the test phase.

3.2 Code analysis

The Solidity Smart Contract code, designed as a standalone module capable of compilation, deployment, and extraction of the ABI (Application Binary Interface), which is crucial for subsequent steps, is accessible in the repository:

<https://github.com/giorgos208/NFT-Royalty-Management>

Project Name:	RoyaltyNFT
Platform	Ethereum Blockchain (Smart Contract)
Language	Solidity
Main Libraries Used	OpenZeppelin (ERC721Enumerable, Ownable)

⁵ <https://www.infura.io/>



Core Features:

- **NFT Integration:** The Contract is based on the ERC721 standard, allowing the creation of unique, non-fungible tokens (NFTs) for each skill.
- **Royalty Management:**
 - **Royalty Allocation:** The Contract maintains a system for allocating royalties to different authors for their contributions to a particular skill.
- **Dependency Management:** Skills can have dependencies on other skills, with allocations specified for each dependency.
- **Skill Structuring:**
 - Each 'Skill' is structured with identifiers, authors, allocations, and a dependency tree.
 - Skills are stored in an array, and each has its unique attributes and dependencies.
- **Donation Functionality:**
 - Allows donations to be made to a specific skill.
 - Royalties are then distributed based on the allocation percentages among the authors and dependencies.
- **Royalty Distribution:**
 - Maintains a balance of royalties for each address.
 - Includes functions for distributing royalties and managing royalty debts.
- **Utility Functions:**
 - Functions for checking if a skill exists, if an address is registered, and the balance associated with an address.
 - Ability to get details of a skill, including its authors and allocation percentages.
- **Author and Skill Management:**
 - Functions to retrieve all skills associated with an author.
 - Ability to mint new skills with specific allocations and dependencies.
 - Royalty reallocation and author replacement features.
- **Contract Administration:**
 - Only the Contract owner can mint new skills and pause the Contract.
 - Includes a function to load the Contract with Ether for royalty distribution.
- **Security and Validations:**
 - Several require statements ensure that transactions meet specific conditions before execution, enhancing security and correctness. (The above mentioned are depicted in Figure)

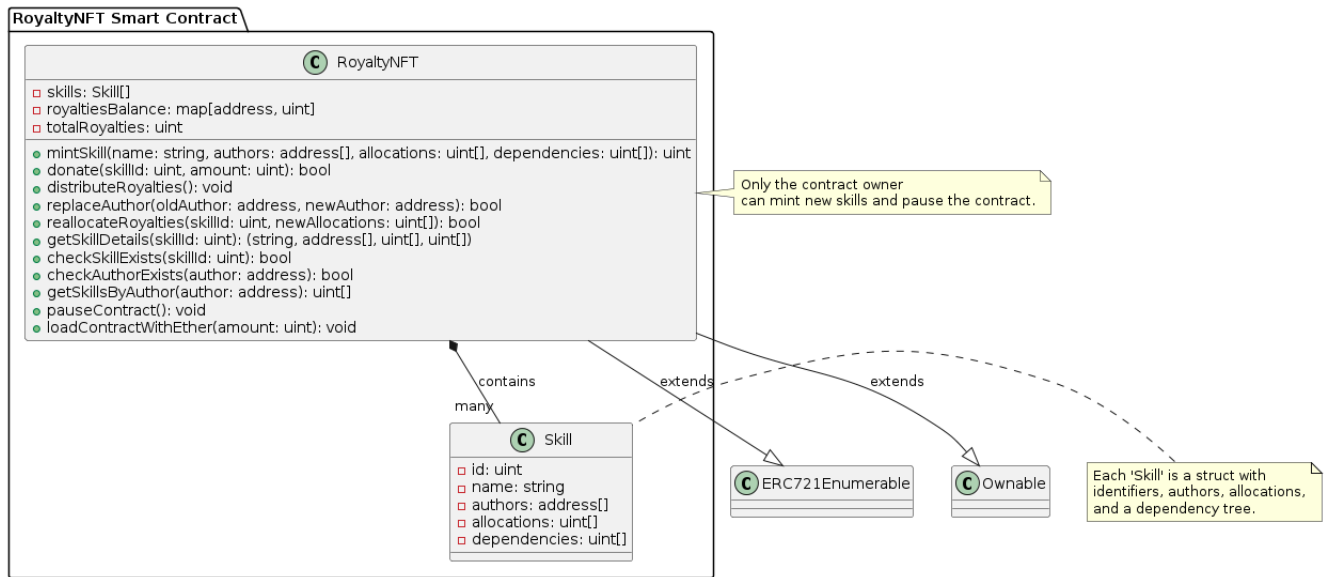


Figure 1 - UML Diagram for the Royalty System.

Here’s a breakdown analysis of the most Important Structs/Functions:

Skill (Struct)

- id: uint - The ID of the skill.
- authors: address[] - An array of author addresses.
- allocations: uint[] - An array of allocation amounts.
- dependencies: uint[] - An array of dependency IDs (if they exist).

mintSkill (Function)

Parameters:

- uint: ID - The ID of the skill to mint on-chain.
- authors: address[] - An array of author addresses.
- allocations: uint[] - An array of allocation amounts.
- dependencies: uint[] - An array of dependency IDs.

Donate (Function)

Parameters:

- skillId: uint - The ID of the skill to which the donation is made.
- amount: uint - The amount to donate.

getSkillsByAuthor (Function)

Parameters:

- author: address - The address of the author whose skills are being fetched.

Output:

uint[] - An array of skill IDs authored by the given address.

reallocateRoyalties (Function)



Parameters:

- skillId: uint - The ID of the skill for which royalties are being reallocated.
- newAuthors: address[] - An array of new author addresses.
- newAllocations: uint[] - An array of new allocation amounts.

3.3 Compilation and deployment

After installing the module from the GitHub repository, you can compile and deploy the Smart Contract to the Sepolia testnet by adhering to the instructions provided in the README file. It's essential for the reader or user to define the variables in the ``.env`` file.

After following the README file's directions, the Smart Contract has been deployed on the testnet. The Contract's address is now logged in the terminal, making it identifiable through blockchain explorers like Etherscan.

3.4 Testing

The system underwent extensive testing, encompassing all conceivable user and administrator interactions. To simulate real-world scenarios, multiple account addresses and wallets representing skills developed by various individuals were employed. Particular attention was given to edge cases, notably in royalty allocation, where a skill might be entirely disassociated from its creators. The demonstration section will include test transactions that illustrate the functionality of all working components.

Figure 2 illustrates the successful integration of a centralized marketplace environment with a decentralized network. The Royalty Distribution System handles the collection and distribution of royalties through an independent layer, ensuring secure and transparent processing of royalties within the decentralized network.

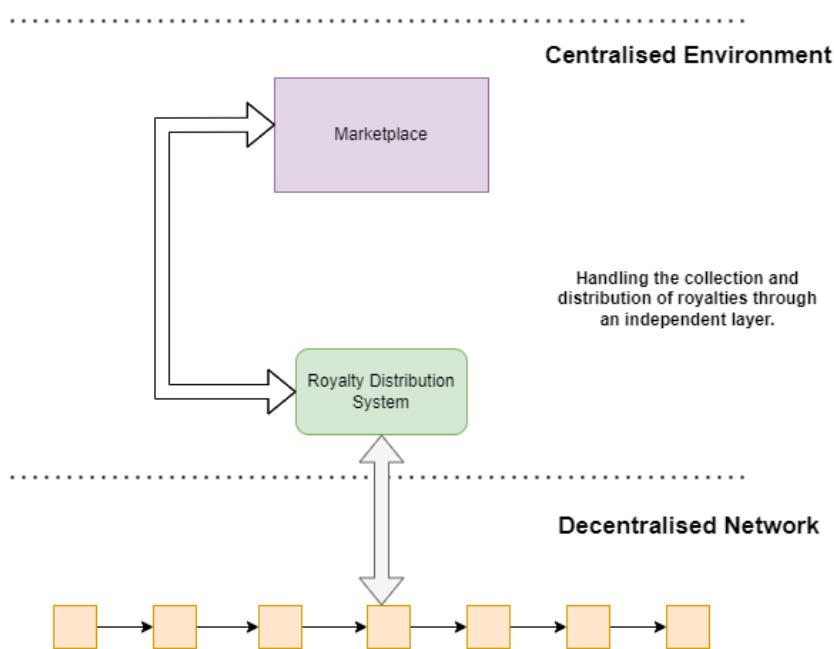


Figure 2 - Royalties are securely processed in a decentralized Network.

3.5 ABI Extraction

The integration of the White Label Shop involves creating a distinct Smart Contract for each Marketplace instance, necessitating multiple duplications or forks of the system. This approach is due to each Marketplace Instance functioning as an independent marketplace, requiring the establishment of its own royalty system. Although these instances will facilitate intercommunication, such connectivity is not necessary for the operation of the decentralized royalty layer.

3.6 Gas fees on Ethereum and the NFT Royalty Module

Gas fees on Ethereum are the costs required to perform transactions or execute smart contracts on the Ethereum network. These fees are paid in Ether (ETH) and vary depending on network congestion and the computational complexity of the transaction. To ensure timely processing of transactions by the NFT Royalty Module, we fetch the latest gas prices in real-time for each transaction. This approach helps to include transactions in the next block as quickly as possible, enhancing the efficiency and reliability of the module. However, this is a design choice that can be adjusted based on specific needs or preferences, allowing for flexibility in how gas prices are managed.

Specifically for the Sepolia Testnet network, where the module was initially deployed, the current gas price is fetched using the API provided by [GasNow](#). This allows us to ensure accurate and up-to-date gas pricing for all transactions handled by the module.

4. WHITE LABEL SHOP (PRESTASHOP) INTEGRATION

4.1 Environment setup

The final product of this development is a PrestaShop-compatible, standalone module, enabling installation within any PrestaShop 8.2.0 environment. For experimental purposes, PrestaShop was installed locally on a macOS system. The most recent version of PrestaShop was obtained directly from the official GitHub repository for local installation. MAMP was utilized to establish the required local server environment and to support the MySQL database tables essential for the PrestaShop setup.

We successfully integrated [Web3 PHP](#) into the PrestaShop module to enable blockchain communication within the e-commerce platform. By leveraging the Web3 PHP library, we established a connection to the Ethereum blockchain, allowing seamless interaction with the smart contract and blockchain transactions. This integration involved setting up the necessary environment, creating a structured module in PrestaShop, and implementing key functions such as retrieving smart contract balances and sending transactions. These blockchain capabilities were then made accessible through PrestaShop's both back-end and front-end by creating hooks and templates, enabling users to perform and view blockchain-related activities directly from the platform. This integration enhances the functionality of the PrestaShop store, providing a decentralized and secure method for handling transactions and verifiable data on the blockchain.

Figure 3 illustrates the deployment and interaction of multiple marketplace instances with distinct Royalty Distribution Systems on the Ethereum blockchain. Each marketplace instance, managed by different administrators and authors, deploys its own smart contract to handle skill tokenization and royalty distribution, capturing royalties from various marketplaces simultaneously.

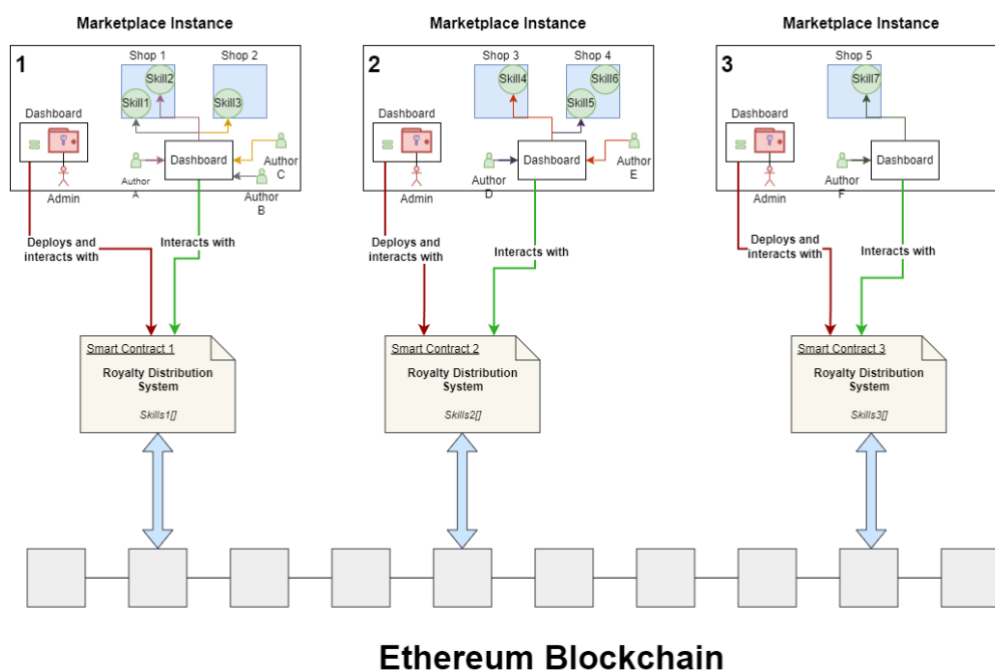


Figure 1 - End result: Multiple systems running simultaneously capturing royalties from all Marketplaces.

To integrate the software into a PrestaShop environment, one should navigate to the Modules section found in the left menu of PrestaShop and proceed to add the custom module, which can be uploaded as a zip file or directly included within the filesystem. Following the successful installation of the module, a new tab labeled “Admin Dashboard” will be accessible in the PrestaShop backend menu, marking the completion of the module's integration.

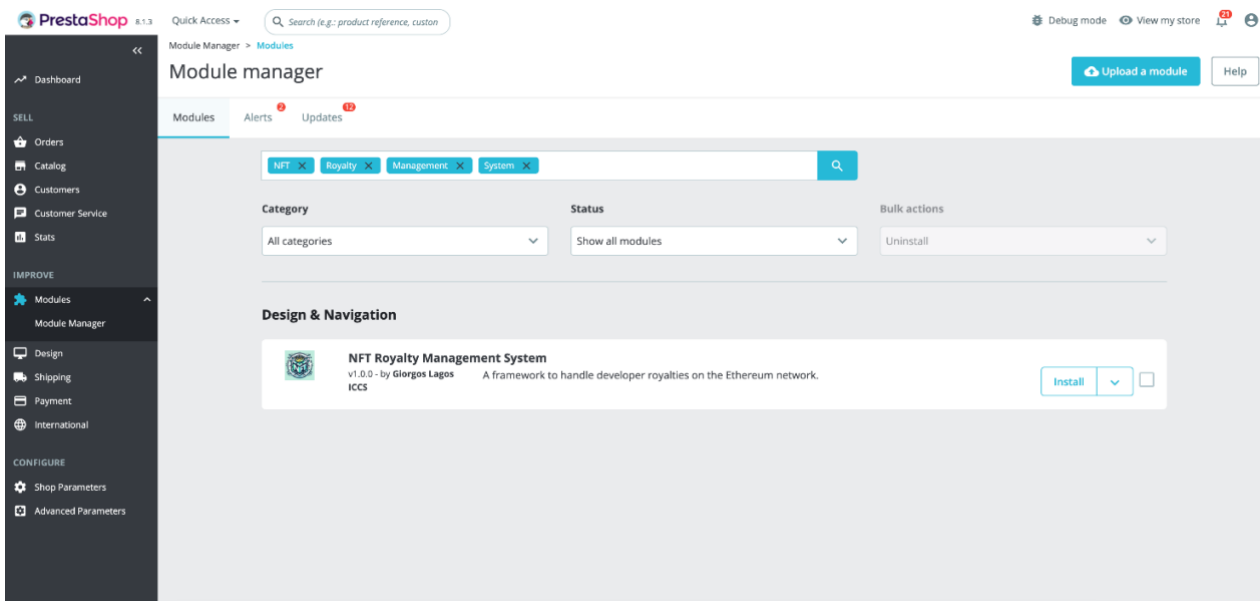


Figure 2- Module Installation in the PrestaShop backend.

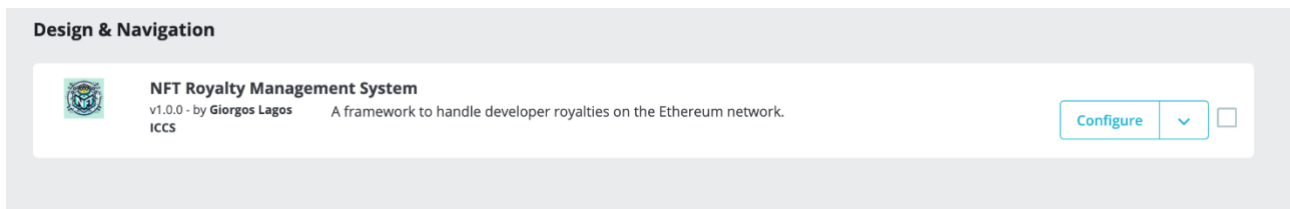


Figure 3 - After successfully installing, one needs to configure the module.

4.2 Admin Dashboard: Smart Contract deployment

The Instance owner Administrator (referred to as "Admin") is tasked with supplying a mnemonic seed phrase for an Ethereum wallet. Once this mnemonic phrase is provided, the entire Web3 system will be activated as shown in figure 6, which provides an overview of the Web3 Admin Dashboard (PrestaShop Panel) for the Instance Owner. The Marketplace Administrator uses this dashboard to manage the Royalty Distribution System by supplying an Ethereum Wallet with adequate funds, loading the Smart Contract with these funds, and distributing royalties to authors. The administrator can monitor the total Ether owed to authors and initiate simultaneous royalty payouts once the contract is funded.

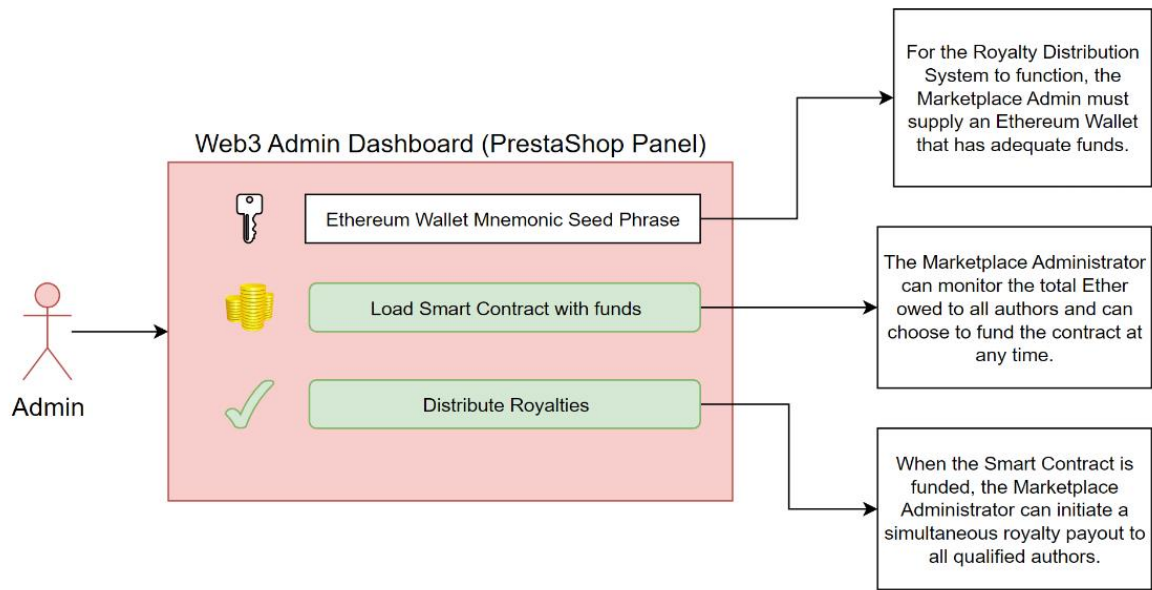


Figure 4 - An overview of the Instance Owner dashboard.

The initialization and setup of the royalty system commence through the Admin Dashboard, where the administrator is required to input all essential sensitive data to ensure the system's operational capacity. For the backend to effectively initiate Smart Contract calls as needed, the inclusion of the administrative wallet's public address and private keys is imperative. This sensitive information will be securely stored within custom PrestaShop database tables and accessed as required by specific actions.

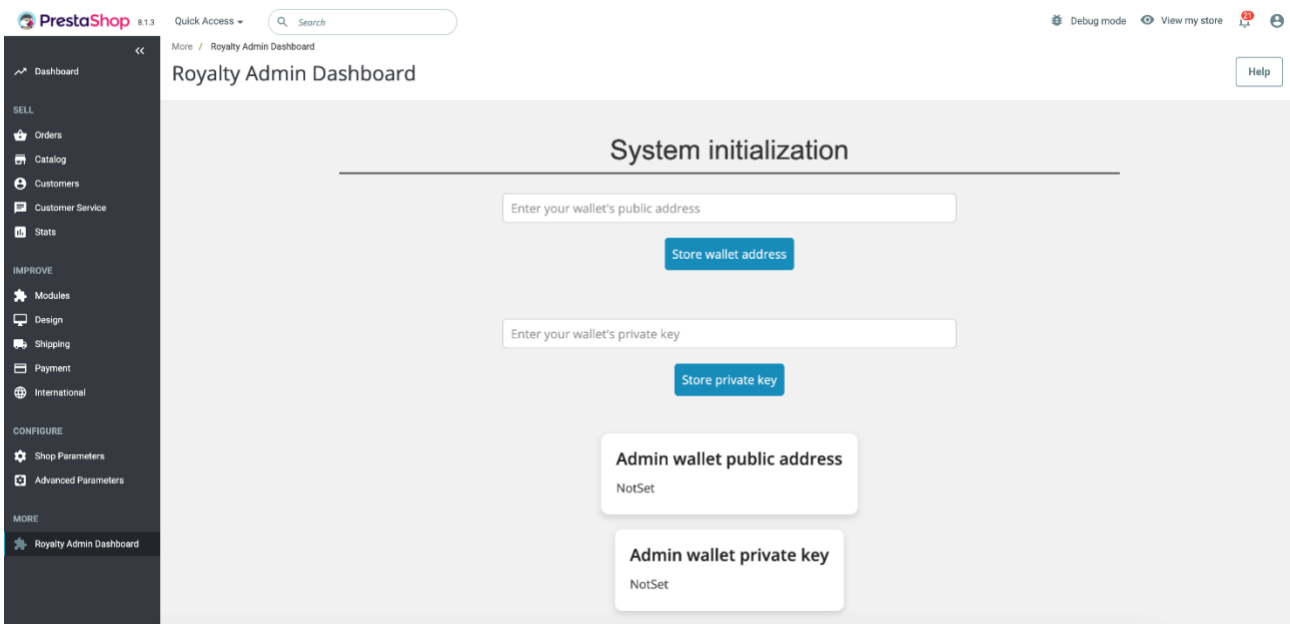


Figure 5 - The Instance owner administrator dashboard.

The latter segment of the royalty administration dashboard encompasses the system monitoring feature, enabling administrators to continually track the current sum of royalties due. This functionality provides

administrators with the discretion to determine the timing for royalty payouts, ensuring they can manage and distribute earnings efficiently.

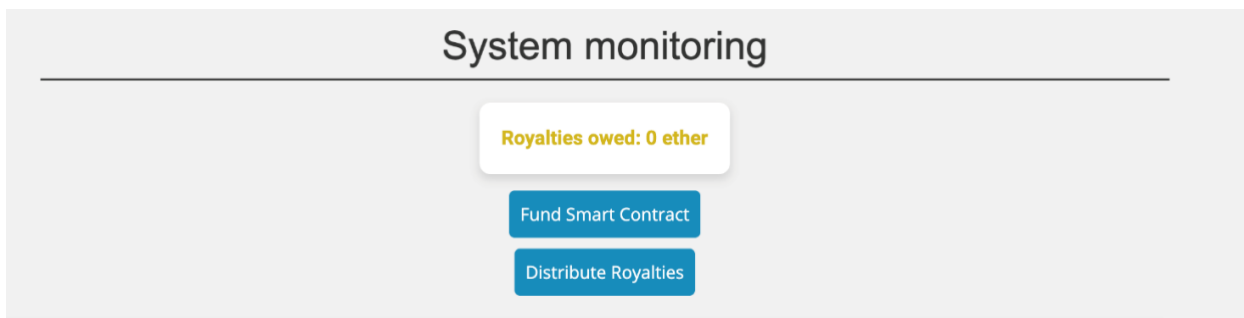


Figure 6 - Royalty amount owed to Skill developers.

4.3 Skill bridging into Web3: Products minted on-chain

To ensure a smooth user experience, the skill bridging process is overseen by the marketplace administrator.

After a skill is successfully uploaded, the administrator can proceed to the skill module tab in the backend to mint and encapsulate the skill into a non-fungible token (NFT) on the blockchain.

Upon the completion of the minting process, the skill is then configured to generate royalties with each subsequent purchase, establishing a direct link between the creator's work and their compensation.

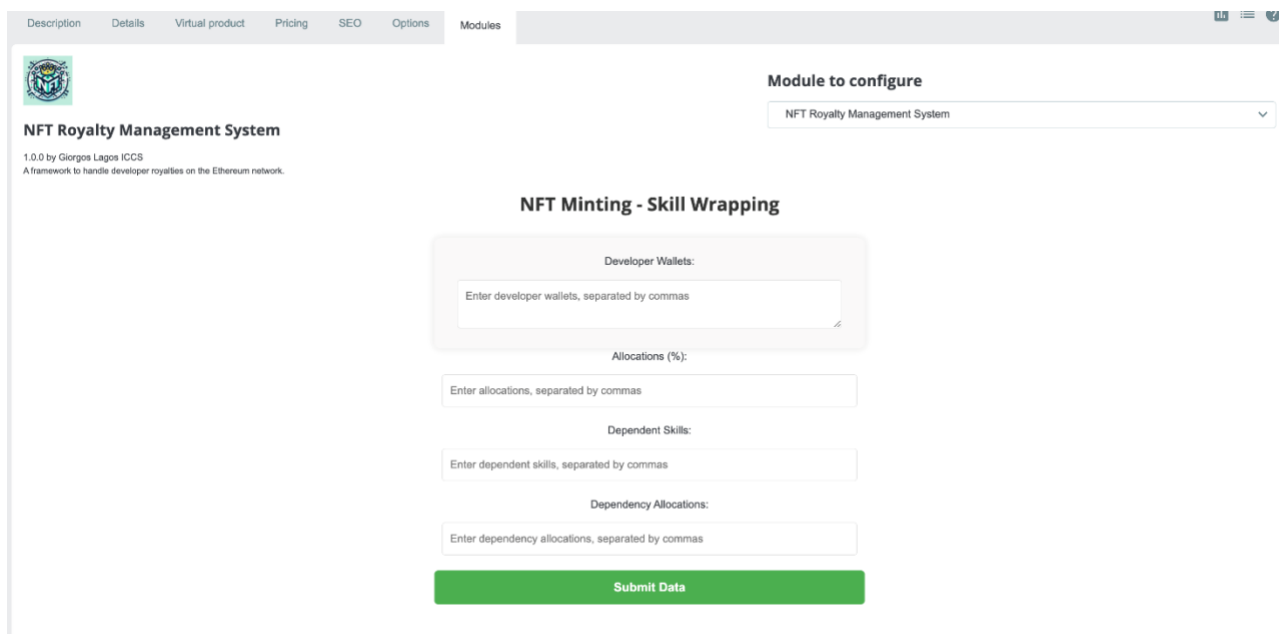


Figure 7 - NFT skill minting by the Instance owner.

4.4 Donation logging on-chain upon checkout

The process of logging donations for skill products is seamlessly integrated into the checkout experience. When a minted skill is purchased, the details of royalty management remain transparent to the buyer, thereby enhancing the user experience.

Buyers are kept unaware to the underlying blockchain mechanics, as a transaction is carried out in the background using the marketplace admin's key to finalize the checkout. This approach guarantees that although royalties are recorded at the time of each skill purchase, their actual distribution is postponed to a future date, as decided by the marketplace administrator.

4.5 Admin Dashboard: Royalty payout

The administrator has the ability to access the NFT royalty dashboard at any time to review the current state of royalty payments, displaying the total amount of royalties owed to all creators. It falls upon the administrator to ensure that the Smart Contract is adequately funded to facilitate the distribution of royalties.

Nonetheless, the administrator is only aware to the total sum of royalties due, without insight into the specific amounts owed to each creator. This level of detail is not required, as the Smart Contract independently manages the distribution of royalties on the blockchain.

4.6 Author Dashboard: Royalty allocation on will

Authors can link their Ethereum wallets within the dashboard to view all skills that have been successfully attributed to them on-chain. In addition, they can monitor their royalty balance. They can also assign royalty points (and consequently upcoming royalties) from skills they own to other individuals as shown in figure 10, which provides an overview of the authors' dashboard. Authors can connect their Web3 Metamask Wallet to view all skills attributed to them, monitor their royalties balance, and assign royalty points (and consequently upcoming royalties) from skills they own to other individuals.

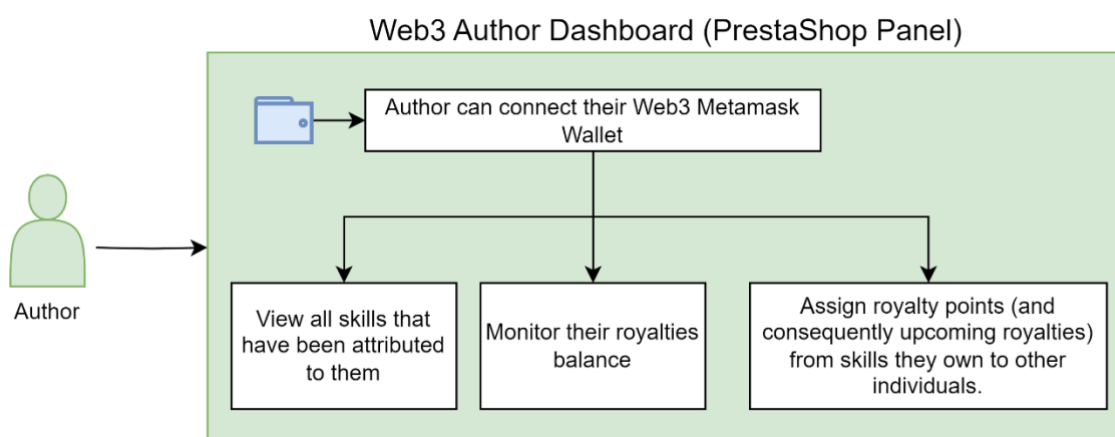


Figure 8 - An overview of the authors' dashboard.

The Author Dashboard provides a straightforward and user-friendly interface that allows authors to connect their wallets and access information about their skills and royalties. Authors must use the METAMASK wallet extension, a widely recognized Ethereum wallet, to log in to the dashboard.

Welcome to the Web3 Dashboard
Smart Contract Address: 0x8da31eddd4190695d120c3577428a256f37d74d

Developer information

[Connect to MetaMask](#)

Wallet address
No wallet connected

Royalties
Royalties debt... Loading

Your skills
Loading skills.. Loading

Allocate Royalty
Skill ID:

Allocation Amount:

Author Address:

[Allocate](#)

Figure 9 - The Skill Author Dashboard

Once connected, authors can view a list of skills associated with their Ethereum wallet address. The dashboard displays each skill by its unique identifier (ID) and shows the percentage of royalty rights the author holds over it, such as "50%" for a skill labeled "X." Additionally, the dashboard provides real-time updates on the amount of royalties currently owed to them, denominated in Ether, offering transparency and immediate insight into their potential earnings.

Developer information

Wallet address
0x5d7A3eBe20800708a5f87Fa101643799a8130CdC

Royalties
Total Debt: 0 ETH

Your skills
Skills for author:
Skill ID: 22
Allocation: 100

Figure 10 - Author-specific stats related to their skills and their royalties.

Authors have the flexibility to allocate royalties to new authors at any time through the dashboard. This process involves specifying the amount of royalty and the particular skill to which this allocation should apply. Authors can only allocate royalties if they have sufficient rights to do so. Upon successful allocation, the new author associated with a distinct Ethereum wallet address becomes eligible to receive royalties from each subsequent donation related to that skill.

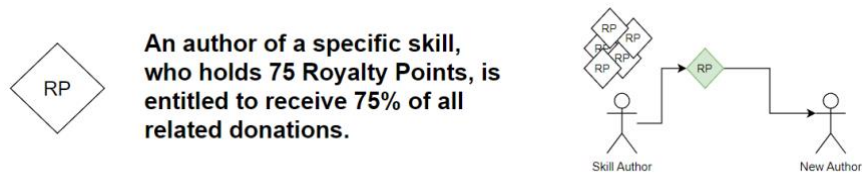


Figure 11 - The mechanism allowing royalty transfer from authors.

The dashboard's functionality is underpinned by robust security measures to ensure the safe management and transfer of royalty allocations. The system's reliance on blockchain technology not only secures financial transactions but also ensures that all changes to royalty allocations are transparently and immutably recorded, safeguarding the interests of all parties involved.

5. DEMONSTRATION

Now, we'll showcase a fully functional procedure of the system, illustrating how, from the initial administrative setup, the system is enabled to manage royalties within the decentralized blockchain framework.

You can also find a recorded video demonstration here: <https://www.youtube.com/watch?v=P6vmiOyo4lw>

After installing the module into the custom PrestaShop instance, one must navigate to the Admin Dashboard, a newly available panel within the Marketplace administrator's backend.

The user is then required to provide their public wallet address and their wallet's private key. This wallet, referred to as the Admin Wallet, will be responsible for handling all system transactions. A straightforward method to accomplish this is by using the most popular Web3 Ethereum Wallet – MetaMask. If MetaMask is utilized, the administrator simply needs to select their wallet account, click on details, and then copy and paste both the public address and private key.

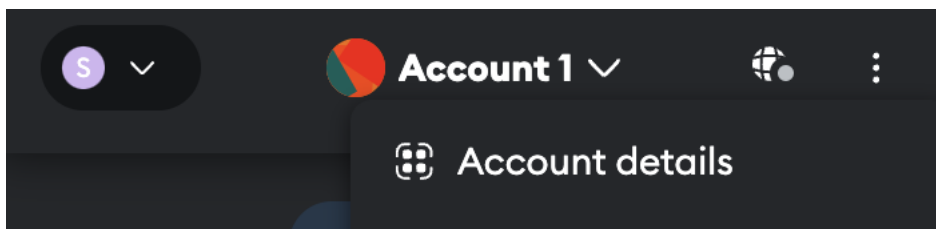


Figure 12 - Extracting the Instance Owner private key.

Once the public address and the private key are supplied, the system can officially be initialized. Upon submitting this information, the administrator will receive confirmation that their data has been successfully stored in the Marketplace Database, signifying the completion of the system's bootstrap process.

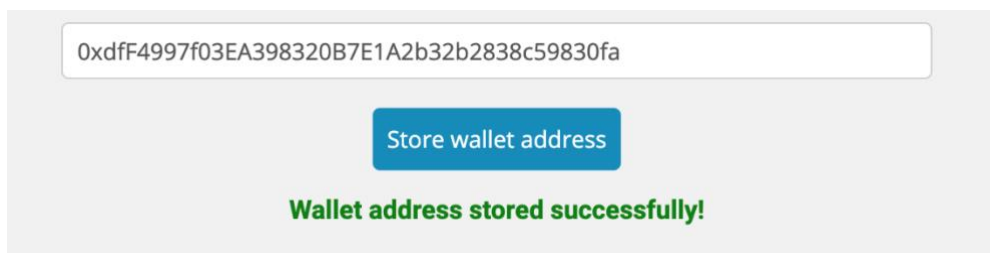
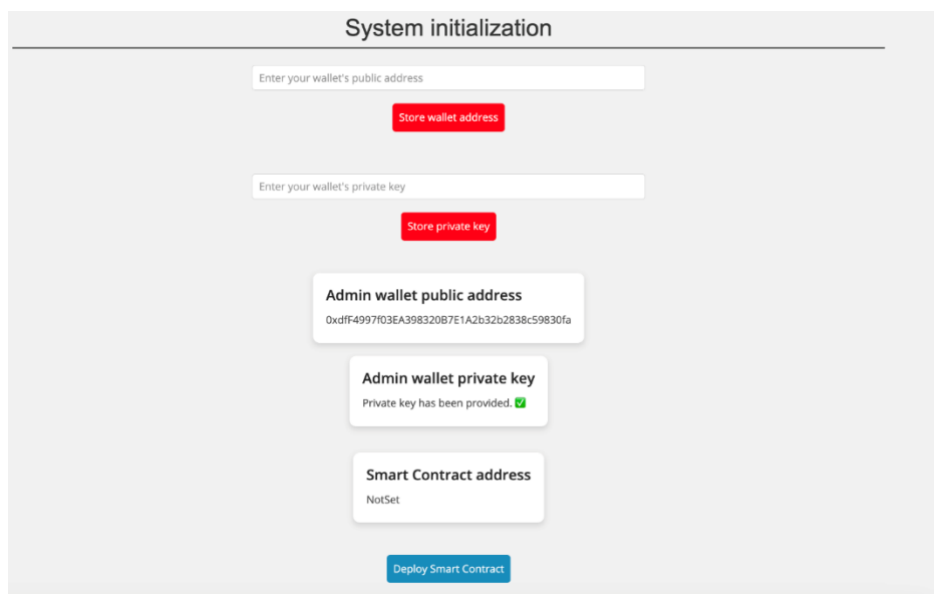


Figure 13 - Instance owner public wallet address stored on database.

Simultaneously, the sensitive portion of the submitted data, specifically the private key, will never be visible on the Admin Dashboard for security purposes. Administrators will only receive a verification message confirming that a private key has been submitted. They have the option to update this sensitive information at any time. However, it's important to note that changing the signing information will render any previous Contract instances inoperative.

After providing the necessary sensitive information, the system is ready to be activated with the simple action of clicking the 'Deploy Smart Contract' button.



The image shows a web form titled "System initialization". It contains the following elements:

- An input field labeled "Enter your wallet's public address" with a red "Store wallet address" button below it.
- An input field labeled "Enter your wallet's private key" with a red "Store private key" button below it.
- A box labeled "Admin wallet public address" containing the hexadecimal string "0xdfF4997f03EA398320B7E1A2b32b2838c59830fa".
- A box labeled "Admin wallet private key" containing the text "Private key has been provided." with a green checkmark icon.
- A box labeled "Smart Contract address" containing the text "NotSet".
- A blue "Deploy Smart Contract" button at the bottom.

Figure 14 - Upon filling the private key and public wallet address the system can be deployed.

Proceeding with the demonstration, we'll simulate the deployment of the Smart Contract followed by the minting of hypothetical skills on the blockchain, which will then accrue royalties.

Initiating this process is straightforward: by clicking the designated button, the deployment of the Smart Contract commences. This step marks the beginning of the system's operational phase, where skills can be digitally represented on the blockchain and set to generate royalties from transactions associated with them.

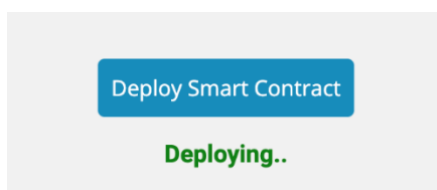


Figure 15 - Deploying the Smart Contract Instance.

The process of deploying the Smart Contract can take up to a minute, as it occurs in real time. This duration encompasses the entirety of communication with the blockchain, including the verification of the newly deployed Contract, all initiated by the button click. This time frame reflects the live interaction and transaction confirmation within the blockchain network, ensuring that the deployment is successfully recorded and verified.

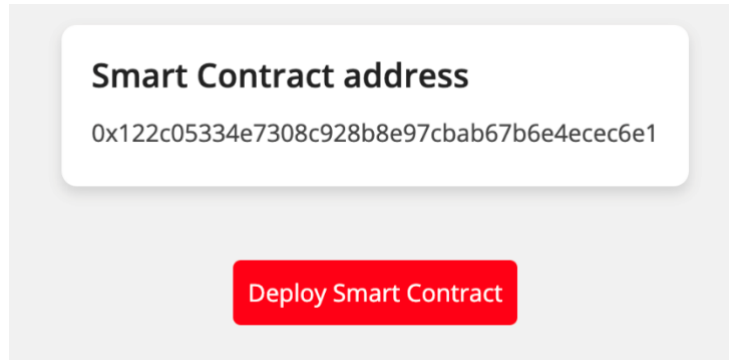


Figure 16 - Upon a successful Deployment a Smart Contract address will be shown.

Once the Contract deployment is finalized, by refreshing the page, the administrator will be able to view the address of the Smart Contract. It's important to note that although the deployment button may appear inactive, clicking it again will initiate a new deployment process. This effectively acts as a system reset and should be approached with caution.

With the system now successfully initialized, the next step involves proceeding to the product upload section of the backend. Here, the administrator can select the specific skill they wish to mint on the blockchain. The option for minting is found under the 'Modules' section of the default PrestaShop Product Item page, specifically within the Minting Tab. This feature enables the seamless transition of digital skills into minted non-fungible tokens (NFTs) on the blockchain, facilitating their entry into the digital economy.

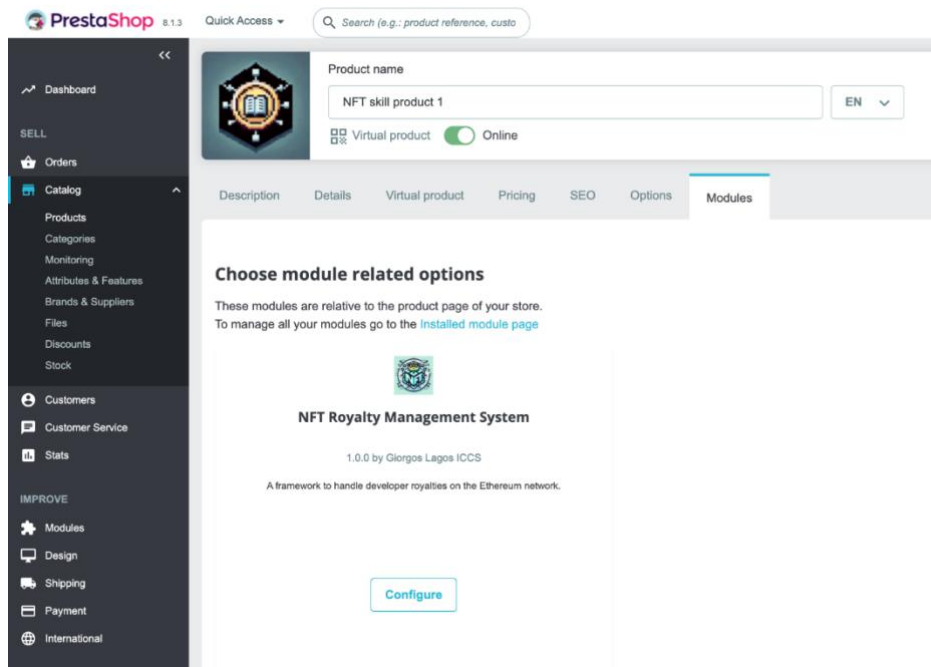


Figure 17 - Navigating to the Skill minting backend Tab.

After selecting the Minting Tab, you will encounter the module and have the opportunity to choose the 'Configure' option. It's crucial to understand that if the current product has already been minted on the blockchain, the minting process cannot be bypassed or duplicated. In such cases, the administrator must consider alternative

solutions, such as resetting the system or creating a new product. This measure ensures the uniqueness of each minted product on the blockchain, preventing duplicates and preserving the integrity of the digital assets within the marketplace.

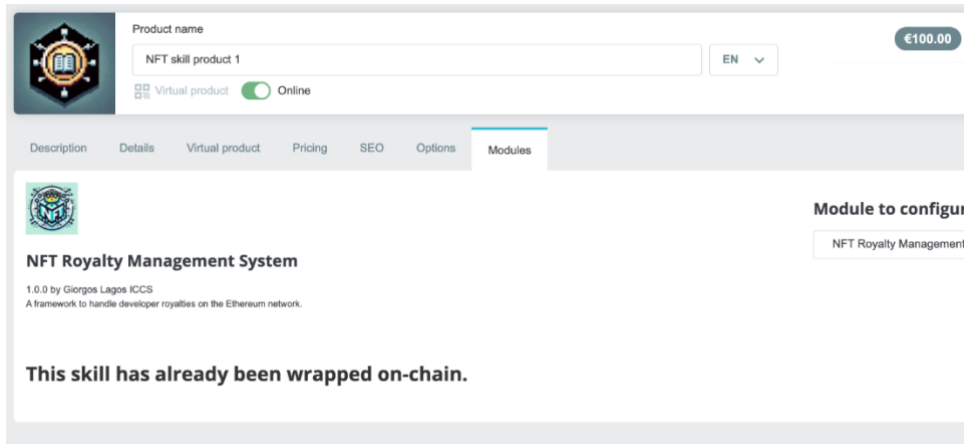


Figure 18 - The same product cannot be minted twice as an on chain skill.

To continue with our demonstration of a royalty payout, we will mint three distinct skills into non-fungible tokens (NFTs) on the blockchain. These three skills will be interconnected in the following manner:

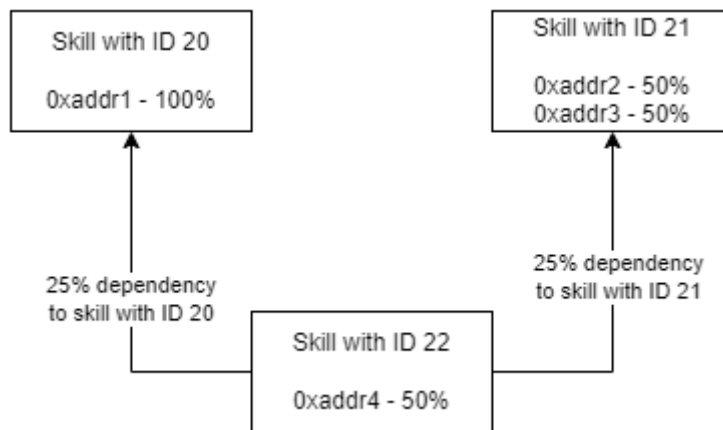


Figure 19 - Mint three distinct skills.

To establish the described relationship among the skills, we will mint each skill separately through the mint skill interface. This process involves selecting each skill individually and utilizing the interface to mint them as non-fungible tokens (NFTs) on the blockchain, ensuring each skill is uniquely represented and can interact within the predefined relationship framework.

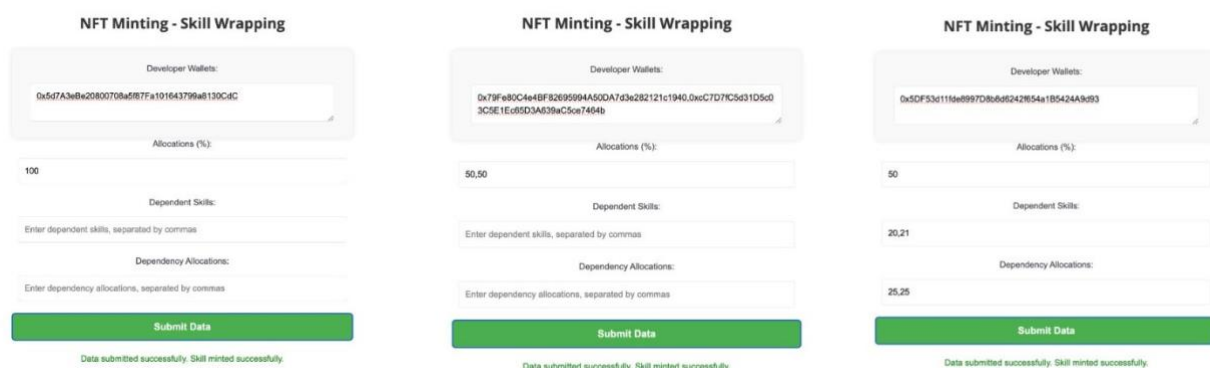


Figure 20 - Proceeding with minting the three mentioned skills.

Next, we will proceed to simultaneously purchase the three minted skills. By purchasing these skills, we activate the royalty mechanisms embedded within their smart contracts, setting the stage for demonstrating how royalties are accrued and distributed among the creators according to the predefined relationships and terms encoded in the blockchain.

When purchasing the skills, payment can be made using any of the accepted methods available in the shop. It's important to note that, at this stage, our module does not incorporate any blockchain-related payment methods. This approach allows for flexibility in payment options, catering to a wide range of users while maintaining the distinction between the transactional aspects of acquiring skills and the blockchain operations related to minting and royalty distributions. This ensures that the process remains accessible to users unfamiliar with blockchain technology, while still leveraging its capabilities for the backend mechanics of royalty management.

Once the order is confirmed, a standard receipt for the purchase is issued, mirroring the conventional PrestaShop checkout experience. From the client's perspective, there is no visible indication of the underlying royalty system in operation. The process appears identical to any other online transaction within the PrestaShop environment.

However, when accessing the system as the Marketplace administrator, a noticeable change is observed: the amount of royalties owed has increased from zero. In this example, the total amount due is the aggregate of ['0.0285', '0.0143', '0.0143']. These figures represent the royalties derived from the skill prices, the allocated royalty percentages, and the conversion of these values into Ethereum, which is then translated back into euros for clarity and consistency in financial reporting. This demonstrates the seamless integration of blockchain functionalities within a traditional e-commerce framework, enhancing the administrative oversight of digital asset transactions and royalty distributions.

The marketplace administrator has the discretion to either distribute the owed royalties to the skill authors immediately or postpone the distribution until a future date or when a more substantial amount has accumulated. For the purpose of this demonstration, we will proceed with the distribution of the current owed amount. It's important to note that the administrator does not have access to specific details regarding which author is owed what amount. All relevant information is securely stored on the blockchain, and the Smart Contract is precisely programmed to know how much is owed to each author, provided it has been supplied with the necessary funds.

After completing the funding transaction, we can observe that the Smart Contract has indeed been adequately funded on the blockchain. This step ensures that the system has the required financial resources to fulfill the royalty payments, maintaining the integrity and functionality of the automated royalty distribution process.

After funding the Smart Contract, executing the royalty payout is as simple as clicking a button. The transaction for the royalty payout on the blockchain is conducted seamlessly, illustrating the efficiency and automation of the system. This step finalizes the distribution process, ensuring that the royalties are transferred to the authors' wallets according to the predefined allocations within the Smart Contract. The ability to manage these transactions with a single click not only simplifies the administrative process but also underscores the power and flexibility of blockchain technology in automating and securing digital transactions.

To thoroughly track the path of on-chain blockchain transactions related to the Smart Contract, you can follow them via the provided link to Sepolia's Etherscan:

<https://sepolia.etherscan.io/address/0x9098ba86efadb3f60e35c751e6ebc83b58064fd9>

This link leads to a detailed view of all transactions for the specified Smart Contract address on the Sepolia testnet version of Etherscan, a popular Ethereum blockchain explorer. Here, you can explore transaction histories, including contract interactions, transfers, and the deployment of the contract itself, providing a transparent overview of the contract's activity on the blockchain.

6. CONCLUSION

Through the development and deployment of a sophisticated smart contract system on the Ethereum blockchain, we have successfully transformed traditional methods of skill and asset management into a decentralized, transparent, and secure framework.

Key accomplishments of this effort include the establishment of the Royalty Distribution Module(W3RDM), which effectively tokenizes skills and contributions into non-fungible tokens (NFTs), ensuring that creators receive equitable and verifiable royalties. The integration with PrestaShop is just one example of how we can seamlessly integrate blockchain-based solutions with existing web2 ecosystems, providing users a secure interface for managing and distributing digital assets.

The practical applications of this system extend beyond mere tokenization. By enabling real-time royalty computation and fostering interoperability across multiple marketplaces, we have laid the groundwork for a future where digital content creators are fairly compensated for their contributions, as such, in future versions of W3RDM we will examine the integration with other paradigms. An important one is package managers such as NPM / MVN / Gradle, where we will analyze dependencies between modules into a dependency tree, and subsequently augment this information by performing SCM analysis. The end result will be a detailed contribution distribution map, which can be managed through W3RDM, and provided as a tool which will compensate developers, especially in the open source community.

7. BIBLIOGRAPHY

- Nakamoto, S. (2008). Retrieved from Bitcoin: A Peer-to-Peer Electronic Cash System: <https://bitcoin.org/bitcoin.pdf>
- Buterin, V. (2013). Retrieved from Ethereum white paper: <https://ethereum.org/en/whitepaper/>
- Wood, G. (n.d.). Retrieved from Ethereum: A secure decentralized transaction ledger: <https://ethereum.github.io/yellowpaper/paper.pdf>
- Madine, M., Salah, K., Jayaraman, R., & Zemerly, J. (2023). NFTs for Open-Source and Commercial Software Licensing and Royalties. *IEEE Access*. Retrieved from <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10024941>
- Christidis, K., & Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*. Retrieved from <https://ieeexplore.ieee.org/document/7467408>
- Gatteschi, V., Lamberti, F., Demartini, C., Pranteda, C., & Santamaría, V. (2018). Blockchain and smart contracts for insurance: Is the technology mature enough? Retrieved from <https://www.mdpi.com/1999-5903/10/2/20>
- Catalini, C., & Gans, J. S. (2016). *Some simple economics of the Blockchain*. Retrieved from <https://www.nber.org/papers/w22952>
- Conoscenti, M., Vetro, A., & De Martin, J. C. (2016). Blockchain for the Internet of Things: a systematic literature review. Retrieved from <https://ieeexplore.ieee.org/document/7945805>
- Commission, E. (n.d.). Retrieved from What is EBSI: <https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/What+is+EBSI>
- Risius, M., & Spohrer, K. (2017). A Blockchain Research Framework. *Business & Information Systems Engineering*.
- Enriken, W., Shirley, D., Evans, J., & Sachs, N. (2018). Retrieved from ERC-721 Non-Fungible Token Standard: <https://eips.ethereum.org/EIPS/eip-721>
- Böhme, R., Christin, N., Edelman, B., & Moore, T. (2015). Bitcoin: Economics, Technology, and Governance. *Journal of Economic Perspectives*.