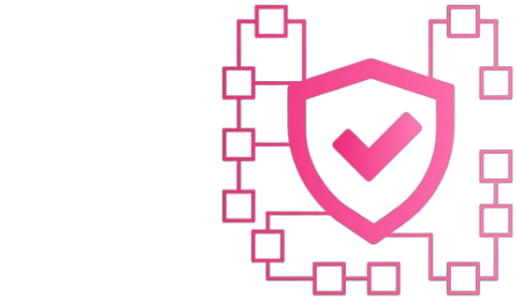
SAPIENZA
UNIVERSITÀ DI ROMA

Blockchain and Distributed Ledger Technologies a.y. 23-24

SmartSupply

Alessio Lucciola - lucciola.1823638@studenti.uniroma1.it
Domiziano Scarcelli - scarcelli.1872664@studenti.uniroma1.it
Danilo Corsi - corsi.1742375@studenti.uniroma1.it

January 24, 2024



SMARTSUPPLY

CREATE IT, SELL IT, CHECK IT!

Contents

1 Preface

1.1 Introduction

The proliferation of fake products, facilitated by the internet, poses serious threats to consumer safety, the economy, and intellectual property rights. Traditional methods of product authentication, such as serialization, RFID, and QR codes, are vulnerable to manipulation. To address these challenges, we propose a **Fake Product Identification system** that leverages the Blockchain technology. This decentralized and tamper-resistant ledger ensures an immutable record of the supply chain, offering traceability, transparency, and enhanced security. Smart contracts automate agreements, decentralization prevents single points of failure, and cryptographic techniques safeguard data. The system involves four key actors **Manufacturers**, **Distributors**, **Retailers**, and **Consumers**. Our goal is to empowering consumers to validate a product's origin and integrity. The project envisions a global collaboration platform to combat counterfeiting and protect stakeholders across complex supply chains.

1.2 Team members and responsibilities

- **Alessio Lucciola:** Smart contracts implementation and tests, initialization of the architecture (e.g. Docker), smart supply frontend and backend implementation.
- **Danilo Corsi:** Editing report and presentation, developing some smart contract, backend and frontend functionality and performing extensive testing.
- **Domiziano Scarcelli:** Frontend UI design and implementation, integration with the Backend and Database via APIs.

2 Background

2.1 The Blockchain

Blockchain technology has its roots in cryptography and distributed systems, emerging as an innovative solution to manage transactions and data securely and transparently. Its history began in the 1990s, in the context of electronic cryptography, when people were looking for a way to secure digital transactions without the intermediation of central institutions. The concept of blockchain was formalised in 2008 by a person or group known by the pseudonym Satoshi Nakamoto, who published a whitepaper entitled 'Bitcoin: A Peer-to-Peer Electronic Cash System'. The paper introduced the concept of a distributed ledger (the blockchain) based on a peer-to-peer network, which would allow the creation of a decentralised digital currency known as Bitcoin. Bitcoin's blockchain represents a public register of all transactions made with this cryptocurrency, guaranteeing immutability and transparency. The basic logic of a blockchain is that of a distributed digital ledger, shared between network participants, which permanently records transactions in concatenated blocks. Each block contains a set of transactions and a reference to the previous block, creating a continuous chain. The security of the blockchain is guaranteed by cryptography, which protects the integrity of the data and prevents retroactive modification of the blocks. The blockchain's decentralised approach eliminates the need for a central authority or intermediaries, allowing direct transactions between network participants. Distributed consensus is achieved through consensus algorithms, such as the proof-of-work used by Bitcoin. Ethereum, introduced in 2015 by Vitalik Buterin, represents a further step in the evolution of blockchain technology. In addition to acting as a cryptocurrency with its Ether coin (ETH), Ethereum introduces 'smart contracts'. These are self-executing computer programmes that automate and manage contractual agreements in a transparent and secure manner. This significantly expands the applications of the blockchain, enabling the creation of decentralised applications (DApps) and more complex blockchain-based projects. Ethereum's blockchain also uses distributed consensus, but is moving towards a proof-of-possession model to improve efficiency. In addition, Ethereum recently updated to Ethereum 2.0, a major upgrade that aimed to improve the performance, scalability and accessibility of the platform thanks to the introduction of the Proof of Stake (PoS) as consensus mechanism.

2.2 Application domain

The application domain of a DApp (decentralized application) that focuses on a Fake Product Identification system lies in the domain of anti-counterfeiting and supply chain management. By utilizing blockchain technology, such a DApp aims to provide a reliable and verifiable record of a product's authenticity, thereby enabling the detection and prevention of counterfeit products within the supply chain. This application domain is crucial for various industries, including pharmaceuticals, luxury goods, and consumer products, where the proliferation of counterfeit products poses significant risks to consumer safety and brand reputation.

3 Presentation of the context

3.1 Aim of the SmartSupply DApp

Our DApp aims to create a system based on the Blockchain to trace the origin and movement of products throughout the entire supply chain. We assume to have four actors that are:

- **Manufacturer:** An individual, company, or entity that produces goods or products through a process of converting raw materials, components, or parts into finished goods that are suitable for use or sale.
- **Distributor:** An intermediary who buys the products from the manufacturers and ships them to the retailers.
- **Retailer:** Establishment or commercial enterprise where goods or services are offered for sale to consumers.
- **Consumer:** The final entity that buys the goods from retailers.

The goal is to allow the consumer to validate a product. This means that the consumer must be equipped with a system to allow him to trace the origin and movements of the product in such a way that he is sure that it has been produced by a certain company and that it has not been tampered with during the creation process and sale.

3.2 Why use blockchain?

Nowadays, manufacturers often implement various tracking and authentication measures to fight this problem, but most of them can be easily manipulated during the supply chain process. For this reason we would like to propose a method to address these challenges based on the Blockchain technology, the main idea is that the decentralized and tamper-resistant ledger ensures an immutable record of the supply chain, offering traceability, transparency, and enhanced security.

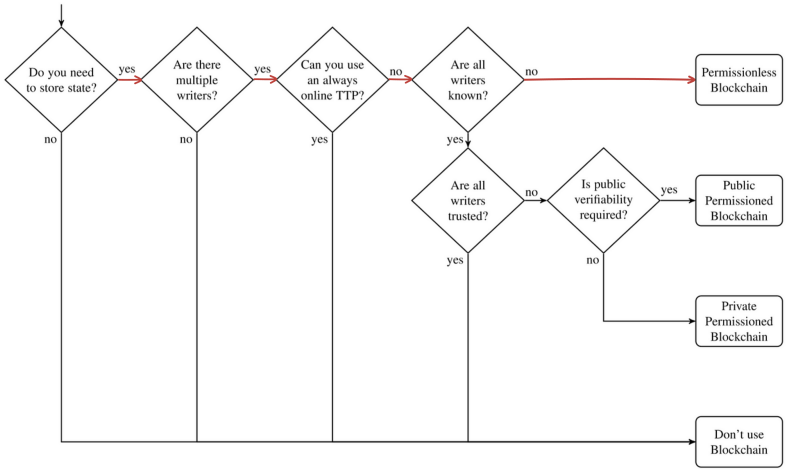


Figure 1: Wüst and Gervais diagram

These were our answers to the following diagram for a **public and permissionless blockchain**

1. **Do You Need To Store state? Yes**, we need to store the product ownership and transaction details.
2. **Are there multiple writers? Yes**, there are multiple users with different roles.
3. **Can you use an always online TTP (Trusted Third Party)? No**, the main aim of the system is to bypass such TTP and have full authority thus promising authenticity.
4. **Are all writers known ? No**, everyone must be able to access it and perform operations on it without the authorisation of a central entity.

4 Software architecture

4.1 Smart Contracts

In order to implement SmartSupply service we created 4 smart contracts:

- **Entities.sol**: It is responsible for managing operations involving the several entities of the systems. It contains the functions for adding, removing and verifying a specific entity in the system.
- **SmartSupply.sol**: It is responsible for implementing the whole supply chain from the production of a good to the final sale to a customer while keeping track of its transfers of ownership. It also contains the methods for managing and distributing the rewards to entities that completed the supply chain.
- **Utils.sol**: This contract is inherited from SmartSupply.sol and simply contains some utils functions (e.g. modifiers), enums and structs used in the main contract.
- **BalanceManager.sol**: It is responsible for managing the balance of the DApp.

Let's go into details and explain how each contract works. **Entities.sol** contains all the required functions to register and possibly remove each entity from the system, in particular it has 4 mappings in which all the roles of the several entities are saved. An entity could register as a manufacturer by calling the *addManufacturer* that simply takes the address of the message sender and adds it to the relative mapping. An entity can also lose its role, and to do so they can call the function *removeManufacturer*. In this case we apply a modifier *onlyManufacturer* that checks if the account has that specific role. The same approach is also used for the other roles that have their dedicated functions. One important operation is the **verification of an entity**. We already said that everybody could potentially join the system. Hence, in order to increase the safety of the platform, we implemented a method to get a **verified badge** that certifies that the user is who they claim to be and this is verified by SmartSupply itself. The process works like this:

1. The entity must first send a proof of its identity (e.g. Partita IVA, IBAN). This request is sent to the admin that must personally certify that the sent information is valid. In our DApp this process is limited to a simple "Accept/Reject" button for simplification matters.
2. If the entity gets accepted, the *grantVerificationPermission* function is called. This saves a boolean value inside the *entityVerificationPermission* mapping that tells that the entity completed the verification step (this operation can only be done by the admin).
3. To get the verified badge and finalize the whole process, the entity has to pay a specific amount of ETH by calling the *verifyEntity* function. Once this happens, the information on the payment is saved in the *verificationStatus* mapping.

The whole process is done so that the users are more willing to buy products from entities directly verified from SmartSupply. Moreover it is a form of financing since coins are transfered to SmartSupply.

Let's move to **SmartSupply.sol** that manages the entire supply chain. The whole process, from the production of a good to its sale and receipt by the customer, is run in a sequential way by 5 functions:

- **produceProduct**: First of all, a product is defined by a struct that contains all its information such as:
  - productID: Unique id of the product which is generated sequentially from a database;

  - productUID: Universal identifier of the product. To clarify, two pairs of shoes of the same model have the same productUID but a different productID;
  - currentOwner: The metamask address of the current owner;
  - previousOwner: The metamask address of the previous owner;
  - creationDate: Unix timestamp that specifies the time of creation of the product;
  - certificationPrice: The amount on coins (Gwei) needed to buy the certification of a product from the retailer;
  - productStage: Enum that specifies the current stage of a product, in particular it can be *produced*, *onSale*, *purchased*, *shipped* and *received*;
  - productLocation: Enum that specifies the current location of a product (different from the stage) and it can be *inFactory*, *inDistributor*, *inRetailer*, *inCustomer* and *Shipping*. We distinguish between stage and location because the stages repeat themselves when a product is received by the next entity;
  - ownership: Enum that saves all the chain of owners as the product is transferred through the supply chain;
  - bankTransactionId: Enum that saves the payments information;
  - rewards: Enum that saves the information on who already received the rewards.

  This function can only be called by manufacturers and adds a new product to the *products* mapping. At first, the *currentOwner*, *previousOwner* and *Ownership.manufacturer* fields are filled with the address of the manufacturer. The product stage is set to *produced* and the product location to *inFactory*.

- **changeOnSale**: In order to be sold, a product must be flagged accordingly. This function takes a *productID* in input and checks that it is the owner that is trying to change the state and that the product is not already purchased or shipped. If these conditions are valid, it changes *productStage* to *onSale*. There is a second version of this function **changeOnSaleRetailer** only callable from retailers. This function is used to define the certification price needed to transfer the ownership of a product from the retailer to a customer. This price is set as *certification percentage* of the price imposed by the retailer. For example, if the price of the product set by the retailer is $20 and the *certificationPercentage* is 10% then the certification price is set to 7900000 Gwei (assuming $1 = 0,0004 ETH of January 2024). To sum up, to finalize the supply chain and allow the customer to confirm its ownership of a product buyed from a retailer, they have to pay a certification price that is dynamically determined and calculated as a percentage of the price of the product.
- **purchaseProduct**: This function takes a *productID* in input and firstly checks that the buyer is not the current owner and the product is flagged on sale. If the conditions are true then we transfer the ownership of the product. For example, if a retailer buys a product from a manufacturer we set *previousOwner* with the address of the manufacturer, *currentOwner* with the address of the retailer and we also set *ownership.retailer* with the address of the buyer. Notice that, as already said, we don't deal with the money transfer (that is managed separately by each entity, even using FIAT currency) so the purchase is just a transfer of ownership. The only entity that must pay the platform when purchasing a product is the customer. We decided to add this feature for 2 reasons:
  - **Certification purchase**: In order to write on the blockchain that the customer finally has the product, they have to pay a small fee. This allows them to download a certification of ownership that can be used to prove that the product is not counterfeit.
  - **Rewards**: The coins sent by the customer to get the certification are splitted equally to the previous 3 owners (25% each) as a form of participation. The remaining 25% is kept by SmartSupply as a form of financing. Later, we'll explain better how this works.
- **shipProduct**: This function takes a *productID* in input and firstly checks that the product has *productStage* purchased and that the entity that is trying to ship it is the previous owner. If the conditions are true then we ship the product by changing *productStage* to *shipped* and *productLocation* to *shipping*.

- **receiveProduct**: This function takes a *productID* in input and firstly checks that the product has *productStage* shipped, *productLocation* shipping and that the entity that is trying to flag the product as received is the current owner. If the conditions are true then we update the *productStage* to *received* and the *productLocation* accordingly (e.g. if the received is a retailer we set it to *inRetailer*).

Figure 2: Concept diagram

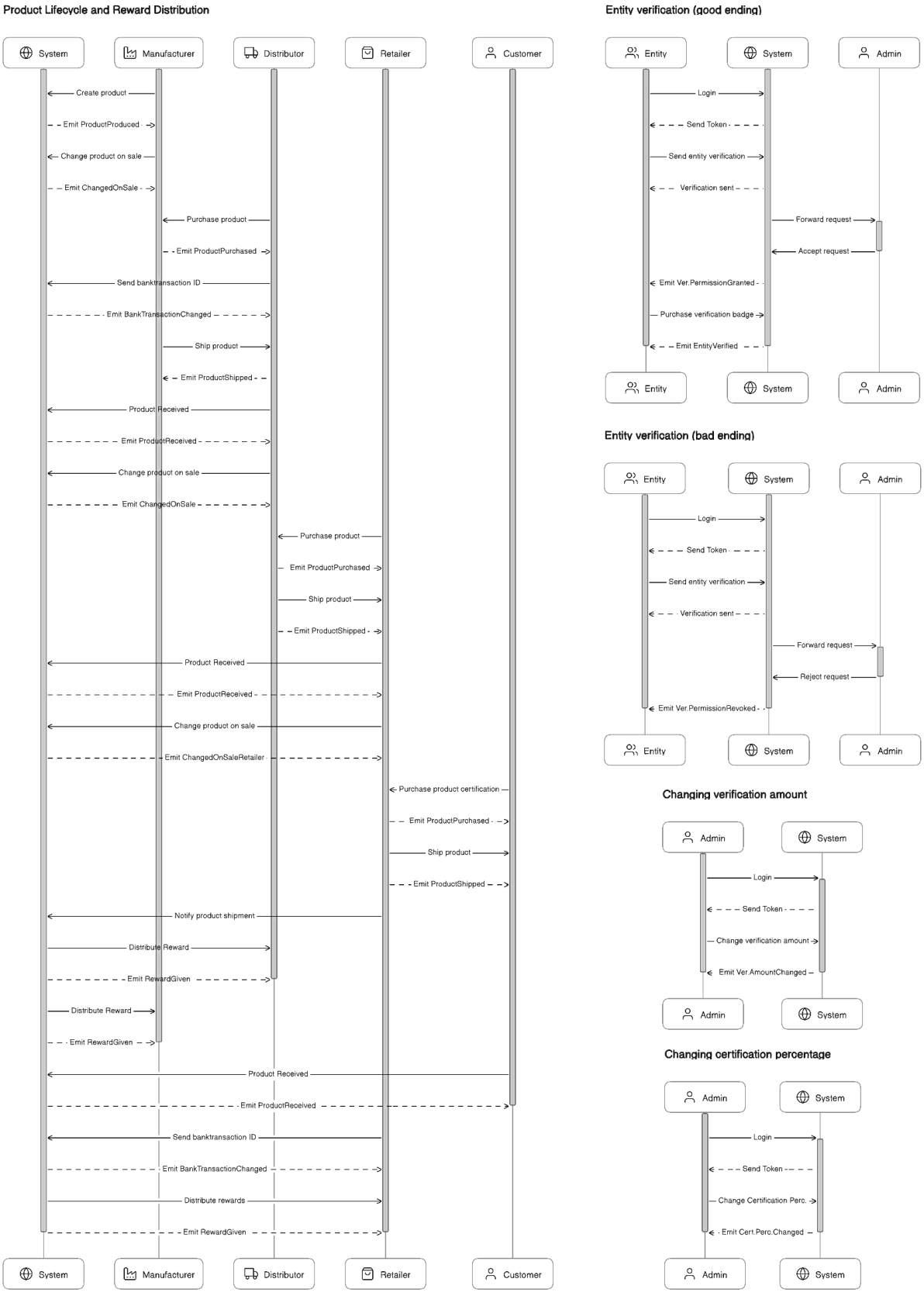


Figure 3: Sequence diagrams

The **reward feature** was implemented with the aim of persuading the business activities to use the DApp. The platform is completely free to use (except for the gas that is paid by each entity performing an action in the blockchain) and to encourage its utilization even more, business activities are awarded with a certain amount of coins. As already said before, in order to get a certification, the customer must pay and the revenues are splitted among the previous owners (25% of the certification cost each and the remaining 25% remains to SmartSupply). The customer sends the coins to SmartSupply when they purchase a product from the retailer. In order to avoid possible scams, it is SmartSupply that is responsible to split and assign rewards when the retailer ships the product. The manufacturer always gets its slice of profit. In order for the distributor and retailer to get the reward they must also send some proof of the payment that is done outside SmartSupply. Just for simplicity, we allow them to send the *BankTransactionID* but in a real system it might not be enough to prove that they actually paid. They get the reward if and only if they uploaded the *BankTransactionID* to SmartSupply using the *changeBankTransactionID* function. Notice that this operation can be done anytime after the product was purchased (even when it has arrived to the customer). Therefore, when changing the id, we check if the conditions to give the reward are valid and possibly send it asynchronously. The function that is responsible for checking the conditions we already stated and distributing the rewards to the entities is *distributeReward*. To sum up, if the entity doesn't specify the *BankTransactionID* when the product is shipped from the retailer, its reward is maintained and transferred later when the transaction id is added to the blockchain. In the sequence diagram above, we show both possibilities (back transaction id sent before and after the shipment of the product to the customer).

Finally, let's briefly explain the **BalanceManager.sol** contract. This contract is responsible for managing the balance of SmartSupply. It contains a function *addFunds* to add funds to the contract balance and a function *withdrawFunds* to withdraw them. The latter operation transfers the specified amount to the admin balance and it is only executable by the admin using the *onlyAdmin* modifier. Moreover it also manages the certification percentage and the verification amount. The certification percentage is used when buying a product from the retailer to obtain the certification of a product as a customer. Its default value is 10% and it is changeable by the admin using the *setCertificationPercentage* function. The verification amount is the amount of coins needed to obtain the verified badge. Its default value is 50000000 Gwei (0.05 ETH) and it is changeable by the admin using the *setVerificationAmount* function. We decided to implement these operations because **if the platform wants to change these values, it is possible to do it without redeploying the contracts**.

4.2 Use cases

The actions each entity can take within the system varies according to its assigned role. Basically all users can **register** and **log in**, going more specific:

- **Verify identity (admin):** Accept or reject entity verification requests;
- **Manage certification percentage (admin):** Percentage to be applied to the price to obtain product certification;
- **Manage verification amount (admin):** Amount required from the various entities to obtain the verification badge;
- **Produce product (manufacturer):** Insert a new product within its own invetario;
- **Request verification (manufacturer, distributor, retailer):** Submit proof of veridicity of the entity to the system and, upon acceptance by the administrator, pay a certain amount to obtain the badge visible to all;
- **Buy product (distributor, retailer, consumer):** Purchases the product from another entity, this action also includes confirmation of shipment and receipt of the product as well as the inclusion of proof of purchase. In this case the customer can purchase the certificate of authenticity from the retailer paying a mark-up percentage on the product price;
- **Sell product (manufacturer, distributor, retailer):** Sale of the product to other entities;
- **Get sold products (manufacturer, distributor, retailer):** List of products sold;
- **Get purchased products (distributor, retailer):** List of purchased products;
- **Get selling products(manufacturer, distributor, retailer):** List of products for sale;

- **Verify product (manufacturer, distributor, retailer, consumer):** Visualization of the product chain from its creation to sale to the end user;

Figure 4: Use case diagram

4.3 DApp structure and modules

Our DApp is based on 3 components:

- **Frontend**: It is the part of our application the user interacts with. We implemented the frontend using the **React** library based on **Typescript**. This library is useful to build interactive and dynamic user interfaces. In order to structure the layout of the DApp, we used **Tailwind CSS**;
- **Backend**: It is the part of the web application that handles the business logic, database interactions, and server-side operations. The server is built using **Node.js** and **Express.js**. We also use a database built with **MySQL** and the framework **Prisma**. The requests to the database are sent using the **REST framework**. We decided to include a database to store all the heavy information which is not recommended to be included in the blockchain. Moreover we also decided to replicate some information that is stored in the blockchain just to facilitate and speed up the communication with the frontend. The backend components are hosted in a **Docker** container to make them easily manageable and deployable.
- **Blockchain**: We implemented the logic of the DApp on smart contracts that we deployed in **Hardhat**. The communication between frontend and smart contracts is accomplished using the **ether.js** library. To test the smart contracts we also made use of **Remix**. We make use of smart contract **events** to trigger operations in the frontend.

In order to use the application, it is necessary to link a **Metamask** wallet that an entity can use to interact with the blockchain. For example, it is used to pay for gas and to pay for SmartSupply services such as the account verification and the sale of a certification.

Concerning the database, we created some tables to support the information already stored in the blockchain:

- **Product:** This table is used to store the universal products, e.g. the information on a model of shoes. When a product is created, it is added to this table and a uid is generated accordingly. The information we save are the name and the description of the product;
- **Product Instances:** The table is used to store the unique products, e.g. the information on the several instances of the same model of shoes. When a product is created, is it associated with the relative uid that links it to a row in the product table. We also save the price of the product given by each entity (i.e. manufacturer, distributor and retailer can choose a different price when reselling the product) and the current owner. We decided not to save prices in the blockchain because this information is public and businesses might want to hide it. We just save the *retailerPrice* that is the price the customer must pay, it is usually public and in out case it is used also to compute the *certificationPrice*;
- **Entity**: The table is used to save the information on an Entity when they register. The saved information is the name and the surname (if it is a customer), the company name and the shop name (if it is a business activity), the email address, the metamask wallet linked with the account and the role;
- **Verifications:** This table is used to store the verification id (e.g. Partita IVA) of the entity. In the blockchain we just save the information on the accepted verification request and the account verified information (after having paid) but not the verification id that could possibly be a string.

Notice that, as already explained before, the *produceProduct* function in the *SmartSupply* smart contracts takes a *id* and *uid* in input. We let the database assign these values when the product is created. We add a new product in the dataset, retrieve those ids and use them to also register the product in the blockchain. This is useful to link the information on the database and the blockchain.

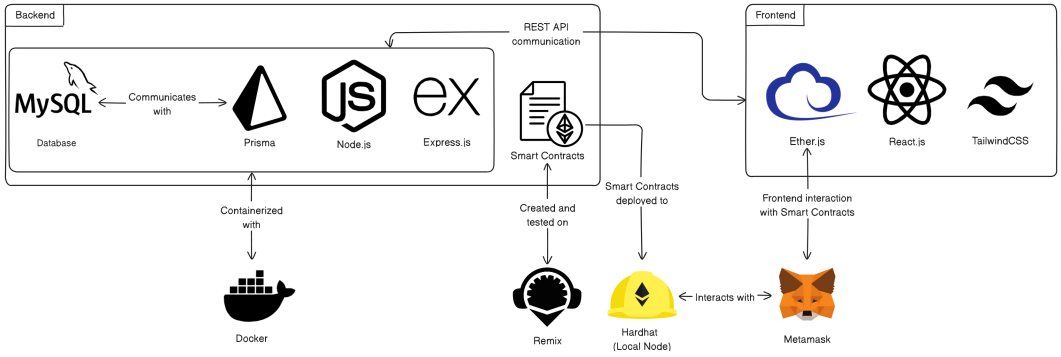


Figure 5: Component diagram that shows the frontend and backend architecture

5 Implementation

Once the app has been opened, the user will find himself on the main screen. Here they can choose whether to register or log in (if they already are). During registration, the user must fill in the following fields and provide a metamask wallet address to be linked to the platform. After confirming successful registration also on the blockchain, they will be able to log in. When logging in, they will have to provide the credentials they entered during the registration phase as well as the address of the associated metamask wallet; depending on the type of user, they will be presented with a screen where they can perform various operations.

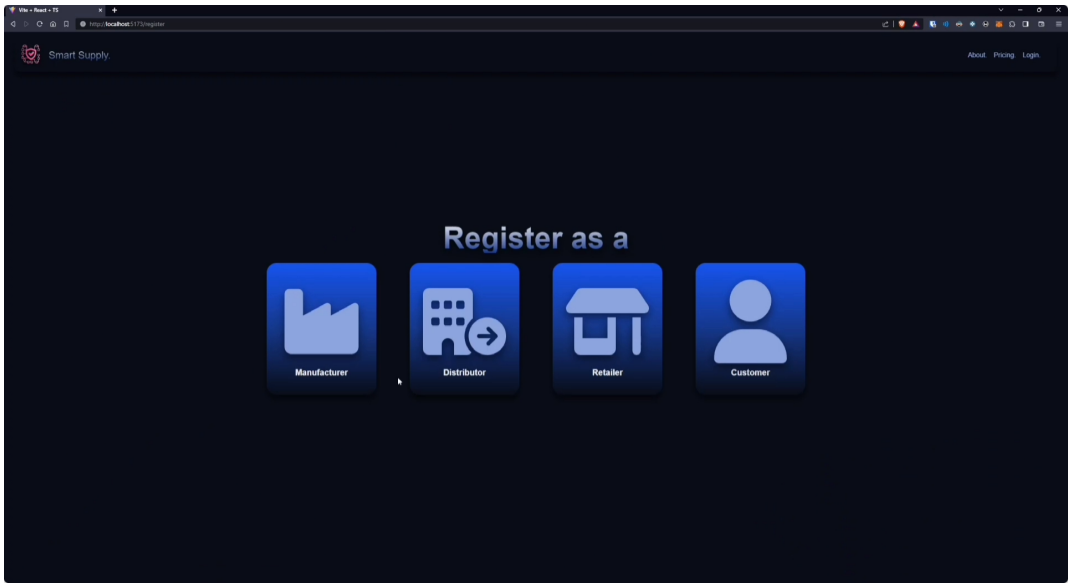


Figure 6: Selection of the entity to be registered

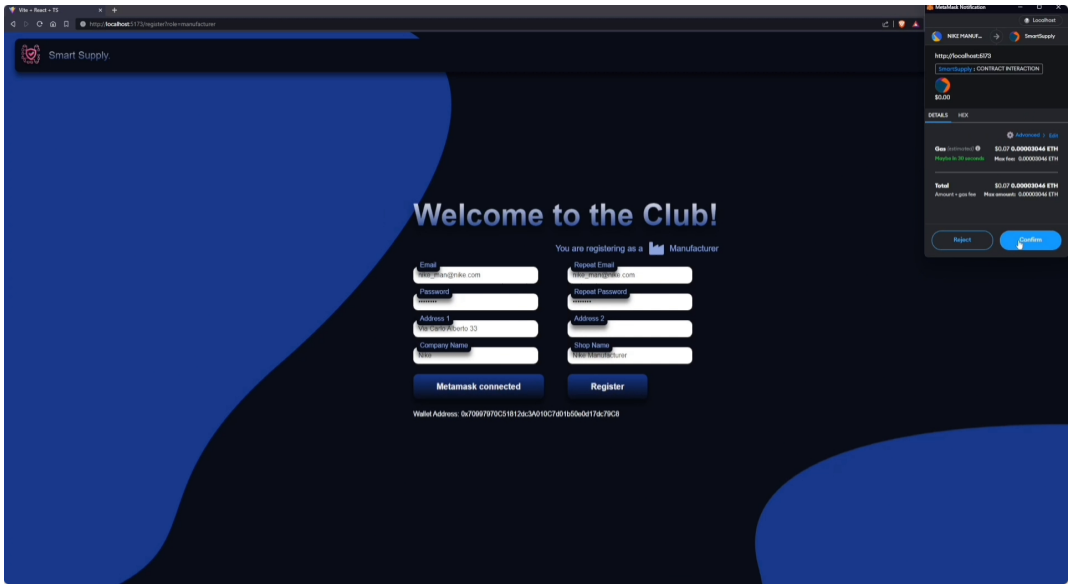


Figure 7: Registration page

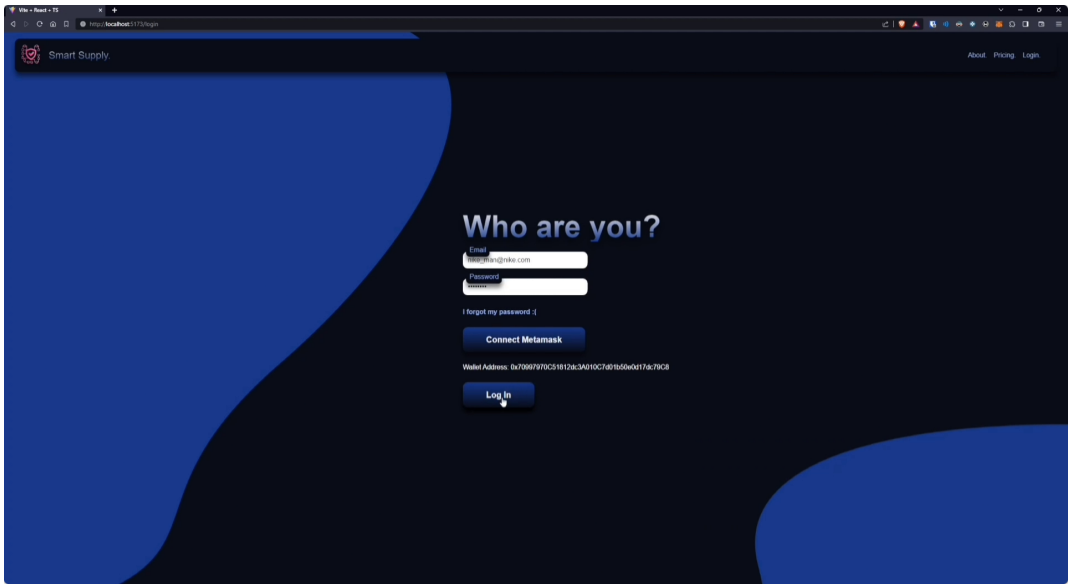


Figure 8: Login page

In the case of the manufacturer, in the *My Shop* page they will be able to enter new products into its inventory, display them and then offer them for sale. Moreover, in the *My Sales* page they can see the history of sold products and flag them as shipped. In the case of the distributor, they will be able to view all articles offered for sale by the various manufacturers, purchase them in the *Purchase Product* page and put them on sale again in the *My Shop* once received from the manufacturer. In the *My Orders* page, they can see the history of purchased products and flag them as received or change the bank transaction id. They also have the *My Sales* page as the manufactureres. In the case of the retailer, it will be able to view all articles offered for sale by the various distributors, purchase them and put them back on sale. The pages and their content are the same as the ones of the distributors.

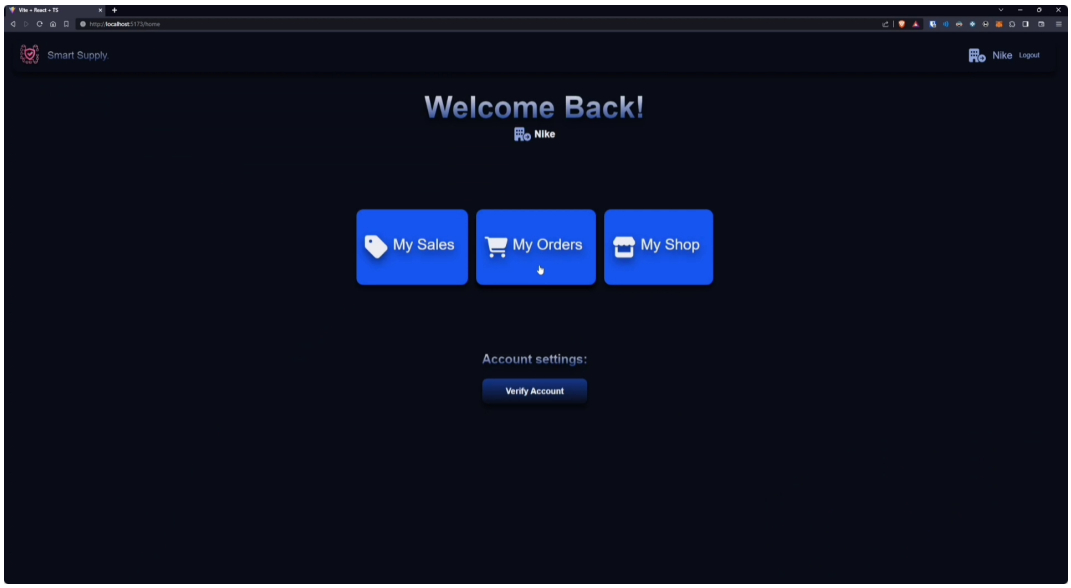


Figure 9: Business activities home page

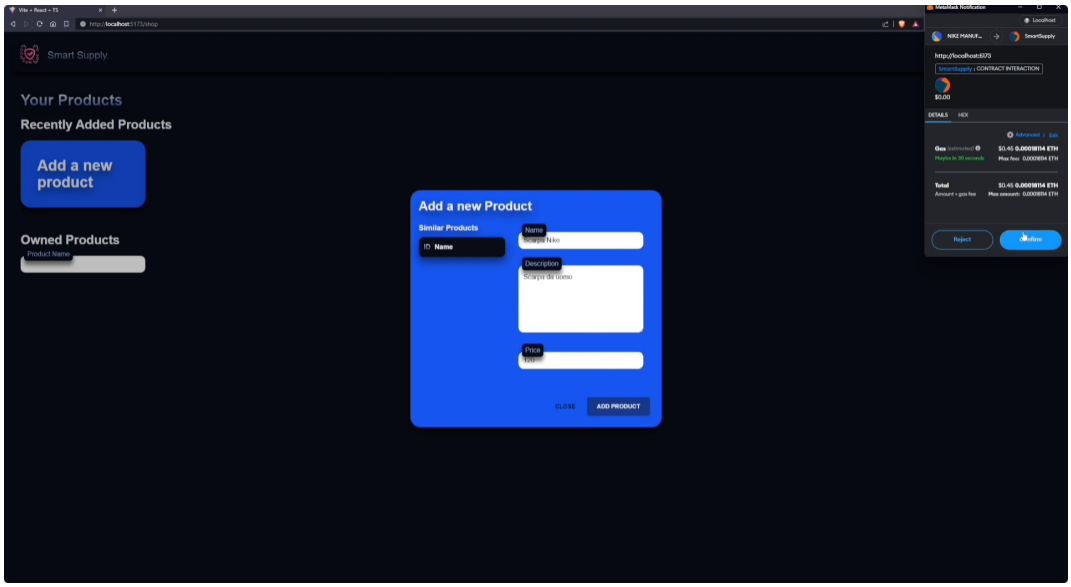


Figure 10: Produce product

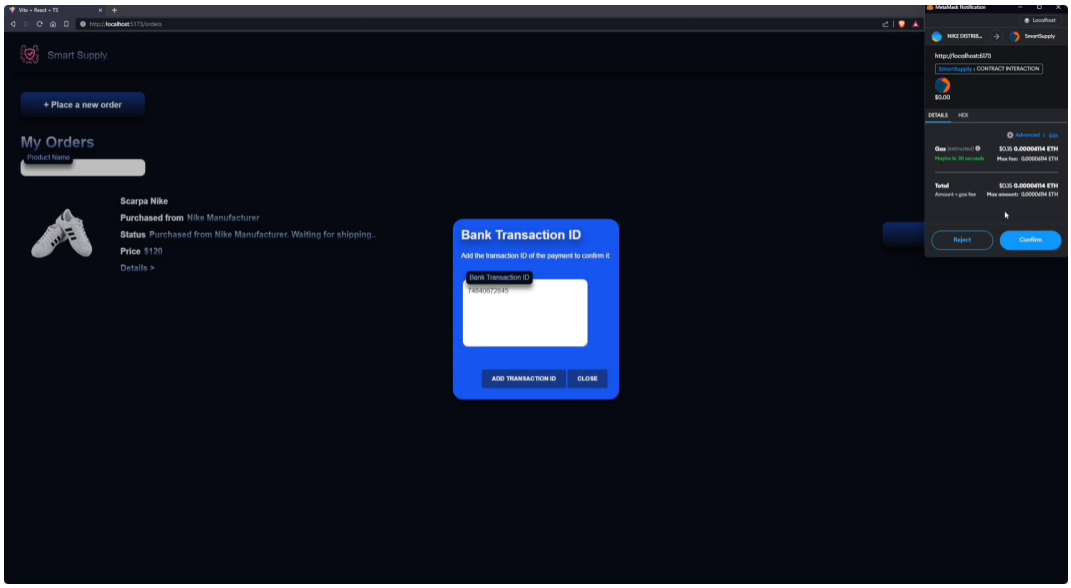


Figure 11: Insert bank transaction ID

As described before, these actors can include a proof of their business (such as a vat number) into the system and, after the system administrator has confirmed it, they will be able to obtain a badge identifying their truthfulness by paying a certain amount of money to the platform. In addition, each time a purchase/sale of a product is made, each actor must mark within the platform the shipment/receipt of it and that the payment has been made by indicating the bank transaction id.

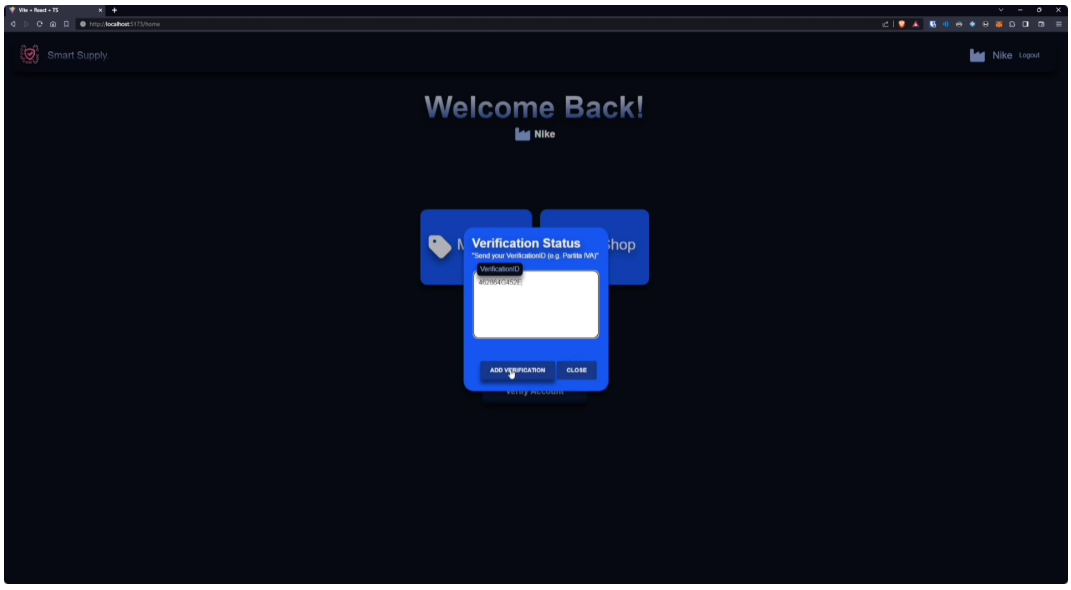


Figure 12: Request for verification of business activity

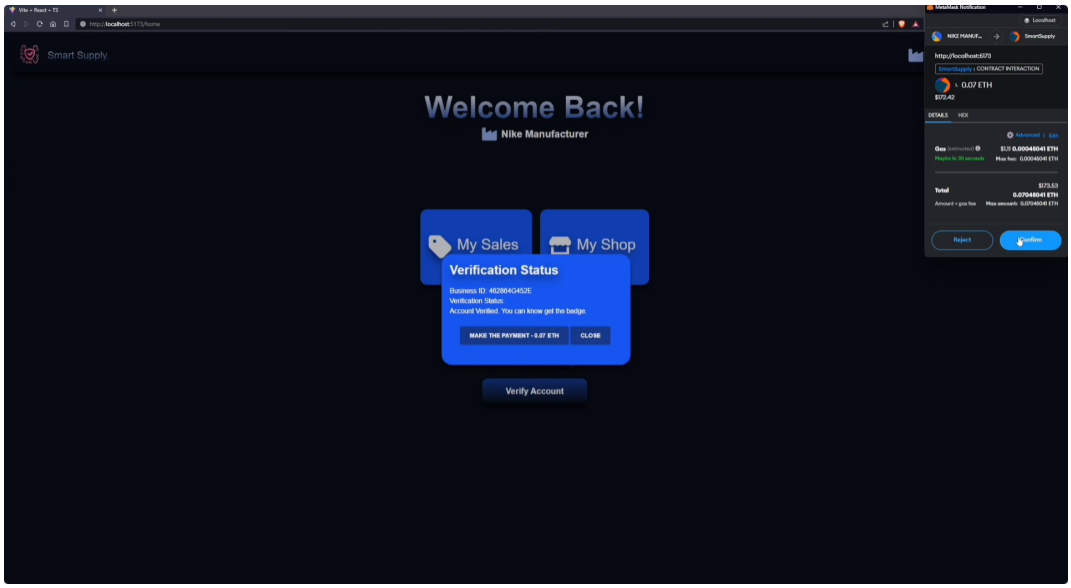


Figure 13: Purchase verification badge

In the case of the customer, they will be able to view all articles offered for sale by the retailers and can purchase the certification of the product by paying a certain amount to the platform. When the retailers ship the product to the customer the product chain is complete and can be viewed by any actor who was part of it. At this point the certification amount paid for that specific product will be divided and given as rewards to the entities that contributed to the creation of the aforementioned supply chain.

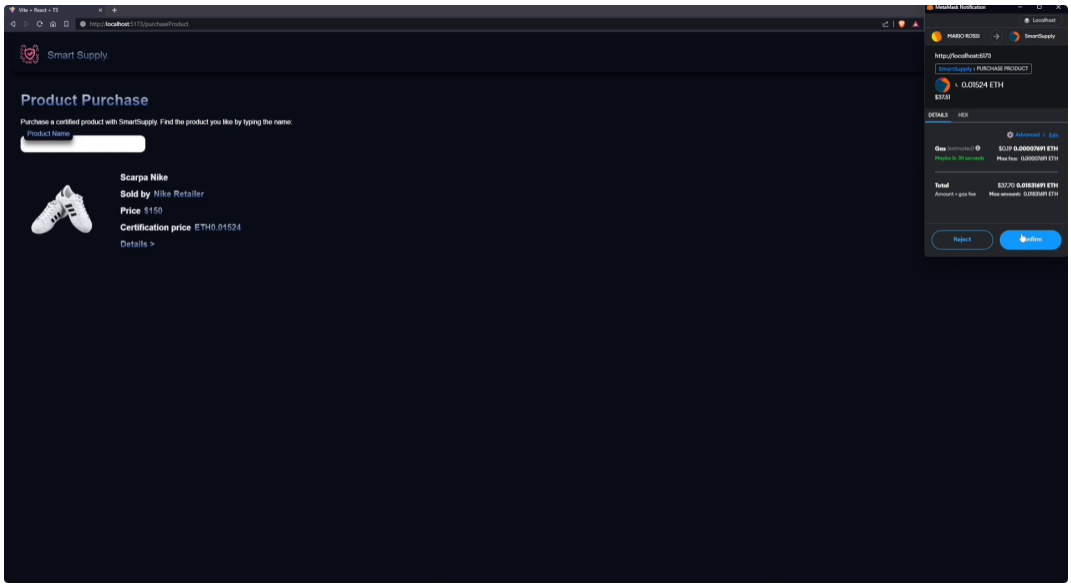


Figure 14: Purchase product certification

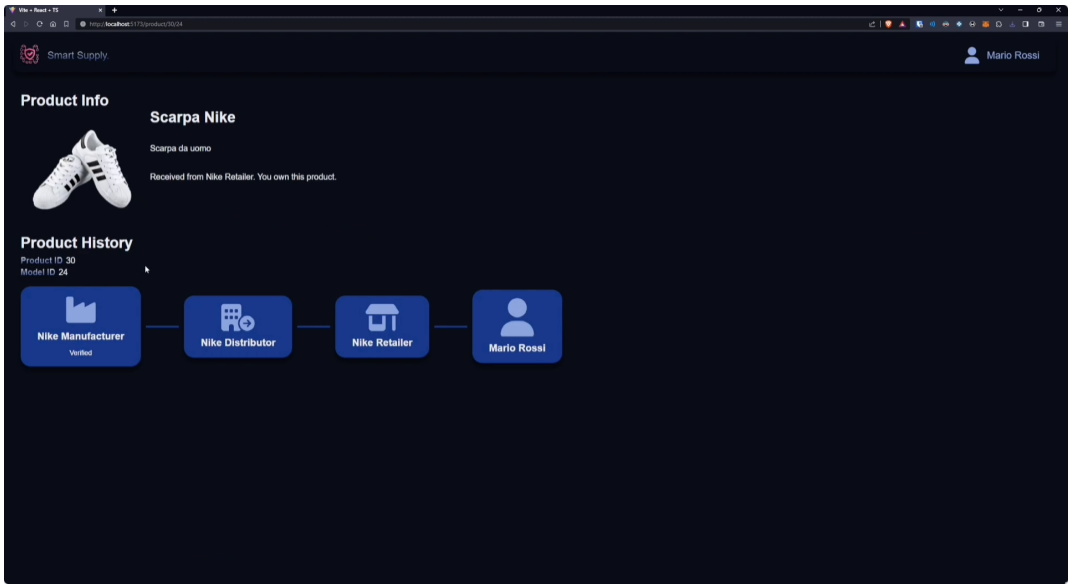


Figure 15: Product details with the complete chain

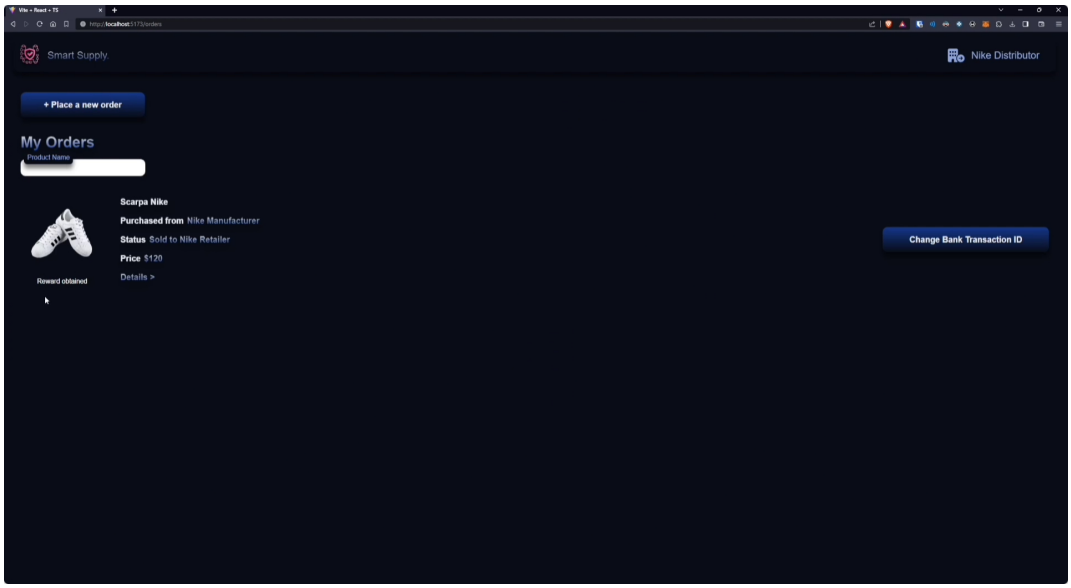


Figure 16: Displaying the seller's "Reward obtained" badge

The system administrator user manages the entire platform and has the ability to accept/refuse verification requests from other users as well as to modify the amount required to obtain the verification badge and the percentage to be applied to the certification price. It is important to notice that a one-off payment will be requested for the verification, while the certification price is calculated applying to the original product price the percentage defined by the administrator.

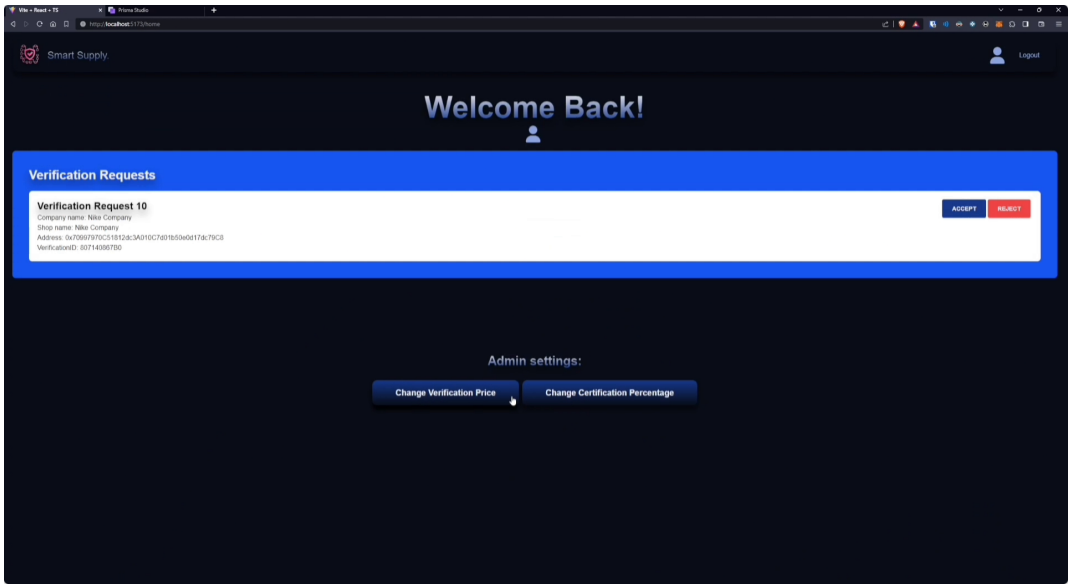


Figure 17: Administrator homepage

6 Demo

A video showing a demo of the DApp is available at this LINK.

7 Known issues and limitations

The DApp we managed to build allows us to successfully track the product from its creation to the final sale to a customer but this process has several limitations. First of all, the system is public hence everybody could potentially join the system. A potential scammer could join the platform and take advantage from a buyer by not shipping a sold product. It is very difficult to solve this problem in a real scenario so we tried to limit it by adding the verification badge. A business activity can confirm its identity by paying a certain amount of coins. The activity shows that it cares about the platform (and it is actually interested in using it) by sending the coins and the user is more willing to buy a product from a verified seller.
Another limitation is that business activities may not be interested in using the platform. For this reason we thought of including the reward system that gives them a small percentage of the certification price given by the customer. The customer may always be interested in using the platform and pay for a small fee to get a product that is certainly certified.
During the design of the DApp we decided to exclude the in-app payment of the product. The purchase of a product is just a transfer of assets and we assume that the real payment is done outside of our application (even using FIAT currency). This is just a design choice but a future upgrade might also include the possibility to pay for a product in ETH using SmartSupply.
Finally, we simplified the bank transaction id system. Currently it is limited to sending a number but in the real system this wouldn't be a user-friendly and secure operation since it would be better to retrieve it automatically from the bank system. This poses some difficulties since we would have to create an oracle to communicate with each (centralized) bank system so interoperability may not be guaranteed. This problem could be easily solved by implementing the payment system in SmartSupply.

8 Conclusions

The proposed **Fake Product Identification** system leverages **Blockchain technology** to address the challenges posed by the proliferation of fake products facilitated by the internet. The software architecture consists of four smart contracts: Entities.sol, SmartSupply.sol, Utils.sol, and BalanceManager.sol. These contracts manage various aspects of the supply chain, including entity registration, product production, ownership transfer, and reward distribution. The system aims to empower consumers by providing a platform for validating a product's origin and integrity, fostering a global collaboration to combat counterfeiting across complex supply chains. In order to receive a certification of the validity of the product they must pay a fee based on the price of the product given by the retailer. The business activities are encouraged to behave correctly with the promise of the final reward when a product certification is bought. In the presented project it is assumed that payment for the products is be made externally. A future extension could be to implement in-app payment with appropriate considerations on the problems that could arise from it.

References

[1] Gautami Tripathi, Mohd Abdul Ahad, Gabriella Casalino (2023) - A comprehensive review of blockchain technology: Underlying principles and historical background with future challenges https://www.sciencedirect.com/science/article/pii/S2772662223001844

[2] Ash-har Quraishi, Amy Corral, Ryan Beard (2023) - $2 trillion worth of counterfeit products are sold each year. Can AI help put a stop to it? https://www.cbsnews.com/news/ai-counterfeit-detection-amazon/

[3] Alice Sherwood (2022) - Spot the difference: the invincible business of counterfeit goods https://www.theguardian.com/fashion/2022/may/10/spot-the-difference-the-invincible-business-of-counterfeit-goods

[4] Baker McKenzie - How Blockchain is Transforming the Fight Against Fake Products https://www.lexology.com/library/detail.aspx?g=83933482-6cba-453b-b093-9390b3c80d05

[5] Shwetabh Raj (2022) - Use of blockchain to protect against counterfeiting https://intellectual-property-helpdesk.ec.europa.eu/news-events/news/use-blockchain-protect-against-counterfeiting-2022-09-16_en

[6] Ellen Glover (2022) - 5 Uses of Blockchain in the Supply Chain https://builtin.com/blockchain/blockchain-in-supply-chain

[7] Huw Lloyd, Mohammad Hammoudeh, Bamidele Adebisi & Umar Raza (2021) - Blockchain-enabled supply chain: analysis, challenges, and future directions https://link.springer.com/article/10.1007/s00530-020-00687-0

[8] Coursera (2023) - Supply Chain Management: Definition, Jobs, Salary, and More https://www.coursera.org/articles/supply-chain-management

[9] McKinsey&Company (2022) - What is supply chain? https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-supply-chain

[10] Amazon AWS - Blockchain for Supply Chain: Track and Trace https://aws.amazon.com/it/blockchain/blockchain-for-supply-chain-track-and-trace/

[11] Jim Keller (2021) - Ensuring Authenticity in the Supply Chain https://www.sdcexec.com/software-technology/supply-chain-visibility/article/21452070/opsec-security-ensuring-authenticity-in-the-supply-chain

[12] Metamask documentation https://docs.metamask.io/wallet/

[13] Hardhat documentation https://hardhat.org/docs

[14] Ethers documentation https://docs.ethers.org/v5/