SAPIENZA
UNIVERSITÀ DI ROMA

Study, design and implementation of a blockchain-based traceability system in the context of used cooking oil recycling

Faculty of Information Engineering, Computer Science and Statistics
Master's degree in Computer Science

Marco Raffaele
ID number 1799912

Advisor
Prof. Claudio Di Ciccio

Co-Advisor
Valerio Goretti

Adjunct Advisor
Prof. Daniele De Sensi

Academic Year 2022/2023

Study, design and implementation of a blockchain-based traceability system in the context of used cooking oil recycling
Master thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: raffaele.1799912@studenti.uniroma1.it

Contents

Chapter 1

Introduction

In recent years, waste management within small or large cities has been assisted by technological innovations to facilitate disposal and avoid waste dispersion. In particular, used cooking oil (UCO) is a waste that is too often ignored but is extremely harmful to the ecosystem, necessarily requiring more effective, sustainable, and smart management for its recycling. In many cases, this type of waste is disposed of incorrectly by citizens, generating extremely negative environmental impacts and huge economic losses [78]. The proper management of UCO during the recycling phase would allow the life of this product to be extended, enabling to creation of energy, transforming it into biofuels, or cosmetics and soaps, applying the right chemical transformations [54, 76]. The production of these products can only take place if this waste is carefully treated and processed into regenerated used cooking oil (RUCO). This thesis aims to develop a UCO tracking system based on Blockchain technology in response to this challenge. The goal is to capture and maintain all information about the UCO during its life cycle from the moment it is deposited by citizens or food service businesses, through the transformation stage into RUCO, to the moment it is reused for new product creations. Such a system also aims to promote a type of economy based on the reuse of existing resources, namely, the circular economy. By developing a system like this, it is possible to decrease the environmental impact of this waste, reducing the use of primary resources, and saving a lot of money for the production of new products such as biofuels and cosmetics [21]. Blockchain technology offers features that are highly suitable for the current case study. Several use cases leverage blockchain to reduce the environmental impact of existing systems or to provide innovative solutions in this respect. In addition, thanks to its transparency, immutability, and security, it enables the creation of particularly efficient tracking systems, limiting the waste of resources and ensuring reliability along the entire path followed by the supervised elements. Digital tracking in the case study described is a key element in data collection and analysis. Indeed,

it allows the entire supply chain to be monitored more accurately, decreasing the error rate and enabling the organizations involved to offer a more efficient and secure service. With the right tracking system, it is also possible to decrease criminal activities related to the illicit sale of UCO. This use case also fits into the context of BPM (Business Process Management), which is an approach aimed at optimizing business processes, thanks to the analysis and optimization of operations within an organization. In this case, utilizing the benefits of blockchain technology can ensure efficiency, transparency, and security for various company operations [99].

Blockchain technology is particularly appropriate for developing tracking systems. There are many works in which it is used to develop tracking systems that exploit its characteristics to reduce the environmental impact of existing systems and promote the circular economy. Some of these are listed and described below. The research by Stroumpoulis et al. (2021) studied how the implementation of tracking systems based on blockchain technology could bring benefits within supply chains, trying to reduce food waste. Blockchain helps coordinate food supply chains, breaking them into more manageable parts and allowing managers to improve control over food [92]. Blockchain technology is well suited to the recycling sector and is increasingly considered for the creation of use cases involving waste disposal, just as in the case of this paper. One of the many works related to this area is the study carried out by Khadke et al. (2021) who describes how the use of blockchain to automate the separation and tracking of plastic waste using unique codes and digital badges would allow for more efficient management of this waste, increasing collaboration between the actors involved [39]. A final case study that is interesting to mention is the one analyzed by Feng Tian (2016), which describes how the agri-food logistics model has become a problem of primary importance and how this requires change. In this regard, Tian proposes a tracking system based on blockchain and RFID (Radio-Frequency Identification) to improve the management of the agri-food supply chain by guaranteeing food safety through the collection, transfer, and sharing of authentic data relating to all the phases involved [96].

In the work described in the following chapters, I explore the current challenges related to the disposal and recycling of used cooking oil and examine the benefits that tracing this waste through blockchain can provide. Furthermore, aspects related to the world of BPM in the context of UCO recycling will be analyzed and described.

1.1 Contribution

The thesis work carried out is based on the use of technologies such as blockchain to create a digital tracking system for Used Vegetable Oils. The goal is to create a certified sequence of information related to UCO movements. The work exploits the

potential of blockchain technology to ensure the immutability and transparency of the information stored in it. It enables the implementation of regulations regarding the documents associated with the waste, which are too often subject to tampering. It allows for complete management by the entity in charge of every aspect of the supply chain, from data read and write permissions to data visibility. The work also involves a reward-based system, which is also guaranteed by the features of the blockchain and allows for the creation of a greater incentive for companies dedicated to the regeneration of this waste.

1.2 Structure of the thesis

The thesis is structured into seven chapters, each devoted to the description of one or more aspects of the proposed system, starting from background concepts to the implementation phase. The realized project is named OilTracker throughout all the following chapters. The first chapter introduced the problem this system aims to solve and its contribution to doing so. Chapter 2 proposed a detailed description of all topics that are involved or necessary to fully implement and understand the work done. Chapter 3 proposes a motivating scenario that contextualizes the involvement of the system within the case study. Chapter 4 describes and motivates all the design choices that influenced the implementation of the OilTracker system. Chapter 5 elaborates on what was described in the preceding chapter from an implementation and technology-involved perspective. Chapters 6 and 7 describe, respectively, the evaluation of the costs involved in OilTracker, and the conclusions defined following the completion of this work while also describing the possible developments that this project could aim for.

Chapter 2

Background and Related Work

2.1 Background

This chapter describes in detail all information regarding aspects, technologies, and concepts that are part of the proposed work. In particular, all the information regarding the background, of blockchain technology and its features, InterPlanetary File System(IPFS) and its operation, circular economy and finally business process management are presented.

2.1.1 Circular Economy

The concept of circular economy cannot be traced back to a precise date or author. An initial introduction of this concept took place in 1989 by economists Pearce and Turner who explained the change from the traditional linear economic system to a more open and circular one, showing a system where the waste produced in the various phases was reused as input materials [73]. To date, one of the best and most widespread definitions of the circular economy is the one proposed by the Ellen MacArthur Foundation in 2016: "A circular economy is one that is restorative and regenerative by design and aims to keep products, components, and materials at their highest utility and value at all times, distinguishing between technical and biological cycle" [26]. The circular economy stands in contrast to the more widely used model of the economy, called the linear economy, where a supply-produce-use-discard policy is followed. The circular economy exploits a more virtuous policy based on the concept of reusing existing materials and products within a renewal cycle, yielding significant benefits from social, environmental, and economic perspectives [46]. It is usual to encapsulate the core concepts of the circular economy in the so-called 3Rs (reduce, reuse, recycle), which have created a reductive perception of what this economic policy can offer. Based on these three concepts, it is easy to think that the circular economy is only applicable to the management of waste and other materials,

however, the 3Rs go beyond this concept. The first R, "reduction", represents the goal of creating production that is "eco-efficient", meaning that it aims to increase value by decreasing environmental impact. The second R, "reuse", represents the goal of creating from a design and engineering perspective, business models based on a cyclic sequence. Finally, the third R, "recycle", refers to all those operations that recover organic and non-organic materials, and resources such as energy, regenerating utility, or creating new purposes for these elements [28]. Numerous research studies have shown how the circular economy offers benefits and is applicable in numerous sectors such as construction, fashion industry, or waste management [1, 44, 72]. There are several challenges and obstacles that this type of economy is forced to face to spread and consolidate in today's economy. These include, for example, problems with designing systems aimed at reuse and recycling, problems with logistics or initial funding, as well as a general low level of interest in these issues [51]. There are currently many technologies that can be used to encourage and facilitate the implementation of circular economy-based systems. Prominent among them are the Internet of Things(IoT), Artificial Intelligence(AI), and Blockchain [29, 45, 105]. Each of these technologies can offer great benefits, IoT allows data to be easily collected and shared, AI enables better planning of activities based on the predictions it makes, and finally, blockchain enables efficient tracking of materials [19].

2.1.2 Business Process Management

Business Process Management is the practice that integrates insights from both information technology and management sciences and then applies these insights to the execution of operational business processes. BPM is based on the concept that every product that is delivered by a company is the result of a series of activities, where business processes are the elements dedicated to managing these activities and their relationships [110]. Business Process Management has its origins in 1911 when Frederick Taylor published a research in which he exposed the benefits brought to productivity by applying the scientific method [95]. The purpose of BPM is to automate processes so as to increase efficiency and decrease operating costs [100]. The business process lifecycle begins with goal setting and appropriate analysis of all aspects that impact business processes. Then there is the design phase in which all processes that require automation or redesign are identified, and during this step, information is also collected on variables such as the purpose and deliverables of the processes. The implementation phase consists of transferring the process models within manual or automated operational environments. Finally, process instances are executed and monitored in real time [60]. Information technology(IT) is the core element of Business Process Management and plays a key role in managing

process activities. IT provides solutions for the design and modelling phase by automating processes through techniques such as process mining that analyze log files. It also enables automated implementation and execution of processes and allows control and measurement of solutions by semi-automatically managing process control and exception management. Finally, it offers tools for innovation and context-driven process improvement by exploiting techniques such as self-learning and enables better program management through the use of tools that give decision support [104]. Business Process Management can be implemented by leveraging different methodologies, such as Six Sigma, Lean, or BPMN. The Six Sigma methodology takes an error-reduction-oriented approach and is based on the logic of Total Quality Management(TQM), an organizational model for quality management within companies, and continuous improvement. This approach is based on the so-called DMAIC (Define, Measure, Analyze, Improve, Control) cycle, which defines the concepts needed to identify and manage all process-related problems [59]. The Lean approach, also called Lean thinking or lean management, is instead an approach that aims to minimize the waste of resources, energy, and time, maximizing the value of the services or products [53]. Finally, there is the BPMN, an acronym for "Business Process Model and Notation", which is a graphical representation of the different phases involved in a business process. This approach offers a common notation for designing process documentation to improve understanding and management of processes. The BPMN exploits graphical elements for the representation of the diagrams, such as flow objects to describe events, activities, and gateways, connection objects to represent the flow of sequences, swimlanes to represent the different participants in the process, and the so-called artifacts, i.e. additional information to increase the level of detail of the diagrams [111].

2.1.3 Blockchain Technologies and DLT

Before discussing what a blockchain is, it is important to define what this technology is more generally, namely, a decentralized distributed system. Distributed systems are a type of system composed of multiple nodes, which work together, coordinating to achieve a common result. The nodes within a distributed system form a network, through which they can communicate with each other, sending and receiving messages. Since nodes are independent of each other, they can take on different behaviors, and thus be honest, defective, or malicious. Nodes that engage in behaviours harmful to the network are called Byzantine [7]. This designation is from the Byzantine Generals Problem, through which L. Lamport described the difficulties in reaching consensus within distributed systems [48].

Among the distributed systems that have attracted increasing interest in recent

years are distributed ledger technologies (DLTs). DLTs are built upon a concept known as a distributed ledger, which is a kind of file used for maintaining and tracking transactions that occur within the network of computer servers called nodes. The special feature of DLTs lies in the fact that a copy of this register is maintained by every node on the network, and if a change occurs this reflects on every existing copy of the register [20].

Blockchain is a form of DLT that utilizes cryptographic techniques and algorithms to generate and authenticate a continuously expanding chain of blocks. In this particular variant of DLT, the chain itself functions as the ledger [62]. Blockchain technology has its origins in the early 1990s when Stuart Haber and W. Scott Stornetta proposed a method for creating secure timestamps for digital documents to prevent tampering and fraud. The proposed system was based on using cryptographic techniques, like hashing, to create timestamps that could be used to verify the authenticity of a document. The hash is a function that takes data as input and returns a string of characters of fixed size, called a digest or hash value, which is unique and can be used to verify the integrity of the data [90]. The authors also proposed allowing multiple parties to participate in the service in order to make it decentralized with each of the parties maintaining a copy of the timestamp database [30].

The study carried out by Haber and Stornetta laid the groundwork for the development of blockchain technology. But the true first implementation came to light in 2008 when an anonymous person (or group) using the pseudonym Satoshi Nakamoto proposed a so-called “Peer-to-Peer Electronic Cash System”, a decentralized digital currency system based on the cryptocurrency called Bitcoin. The study describes the problems with traditional payment systems, such as commission costs and the need to trust central authorities. In particular, the paper describes how the Bitcoin system allows users to send and receive transactions without the need for an intermediary. The generated transactions are recorded on a public ledger, namely the blockchain, which is maintained by a network of nodes that validate and confirm each transaction [61]. A transaction is an operation that involves the transfer of cryptocurrency, or assets, between two accounts within a blockchain, and represents the information stored on the ledger [112]. In order for transactions to be sent between nodes on the network, they must generate a pair of cryptographic keys, one public and one private. The private key has the purpose of signing and therefore authorizing the sending of a transaction, while the public one represents the address with which the user is identified on the network and the account or wallet where all the exchangeable assets are stored [9]. Blocks are the fundamental elements that make up the blockchain and form the system’s ledger. Each block contains various information, including a timestamp, a group of transactions, and the hash value of the previous block.

Thanks to the hash value, each block is linked to the next. If any tampering occurs with a block, it is immediately detected, thanks to the uniqueness of the hash value. Even the slightest change in the data contained in a block generates a different value, which ensures the immutability of the blockchain [66]. Another very important piece of information contained within the block headers is the Merkle root. The Merkle root is a cryptographic hash value of all transactions contained in the block. It is calculated by constructing a binary tree, called a Merkle tree, in which each leaf represents a transaction and each non-leaf node is the hash value of its child nodes. The Merkle root value is the hash at the top of the tree. When the hash of the block containing the Merkle root is included in the header of the next block, an unbreakable link is created between the blocks and the transactions in them [55].

The evolution of blockchain technology over the years has led to the emergence of new platforms with different functionalities and features. The platform that has attracted the most interest is Ethereum, proposed by Vitalik Buterin in 2013 [12]. Ethereum is a decentralized platform created with the goal of enabling the creation of so-called smart contracts, which are described in detail in the following sections, and decentralized applications(DApps).

Consensus mechanism

Blockchain technology is based on concepts such as decentralization, immutability and autonomous organization. These characteristics are guaranteed by consensus mechanisms [113]. Consensus mechanisms were created to enable the verification of generated transactions in a way that was resistant to failure. In fact, in blockchain technology, the problem of maintaining a valid state along all nodes of the network can be seen as a fault-tolerant state-machine replication problem [79]. Consensus mechanisms enable the preservation of agreement between nodes in the distributed network and are responsible for preventing possible attacks from bad actors, making the network secure and reliable [49]. The first consensus mechanism realized for blockchain technology was the one proposed by Satoshi Nakamoto [61] with the creation of Bitcoin, named Proof of Work (PoW). As explained by the author, Bitcoin's Proof of Work is similar to the HashCash system proposed by Adam Back [6]. The PoW encapsulates the entire process required for a new block, and the transactions it contains, to be added to the blockchain. The process that constitutes Proof of Work is called mining, since through it new Bitcoin tokens are minted. The nodes that participate in this process are called miners who compete with each other in order to solve a very complex mathematical problem. The problem, or cryptographic puzzle, is to find a nonce value, which is added to the header of the new block, such that the result of the hash function applied to the block header itself results in a value that begins with a given one number of zeros. To give a more

formal definition, we denote by $H(\cdot)$ the hash function and by x the value of the binary string created by concatenating the information contained in the new block. Given a value of difficulty h, solving PoW means finding the nonce value, contained in x, such that, the result of the hash function applied to x is less than a value $D(h)$.

$$bh = H(x \parallel nonce) \leq D(h)$$

for some fixed lenght of bits L, $D(h) = 2^{L-h}$, where bh stands for block header [109]. Once the proof of work is satisfied, the block cannot be changed except by doing the work again. Consequently, as the blockchain length increases, changing the content of one block requires satisfying all PoWs of subsequent blocks. Depending on some aspects such as the hash rate of the network, and the mining hardware used, the time for creating a new block can vary. To ensure that the time with which a new block is added to the blockchain remains constant over time, the protocol provides for an adjustment of the difficulty of the cryptographic puzzle, based on the average time for mining a block [36]. The first node that solves the PoW, broadcasts the newly created block on the network, and this is validated by the other nodes by verifying that the nonce value produces a valid hash and the transactions are correct. The miner is also rewarded with newly minted cryptocurrency and the sum of transaction fees in the block [27]. In the case of Proof of Work, a problem that can occur is that of forking, which consists of the division of the blockchain into two or more different branches, each with its version of the information history [82]. Forking of a blockchain can occur in the case where two miners, resolve the proof of work almost at the same time which leads to both blocks being hung on the blockchain. When a fork occurs the nodes choose which version of the blockchain to follow, usually based on the length of it, as it means more computational effort was spent on that branch [64]. This mechanism makes it very difficult for a malicious actor to create its version of the blockchain containing false information. This is because in order for its branch to be the longest it would need more computational power than the rest of the nodes in the network combined ("51% attack") [85].

The mining process requires a very high computational effort, and it also involves a large energy consumption. The increase of users taking part in the PoW has led to too high and no longer acceptable energy consumption [18]. The first solution that emerged to solve the energy problem due to mining was the Proof of Stake (PoS) and it follows a different philosophy than the one proposed by the PoW [83]. In fact, it is based on investing some crypto-commodities (stake) in the system, avoiding computational efforts [41]. The stake represents the chances of being selected for the validation of the new block. The higher the stake, the greater the chances of being chosen. Nodes that want to take part in the consensus phase by becoming

validators must deposit their stake, usually expressed in terms as native tokens, in a dedicated wallet or to a specific address [65]. The choice of validators can be based on different aspects, such as the amount of tokens in staking, or by the age of the tokens. Coin age represents the length of time a coin remains in a user's possession; the older the coin, the more value it takes on in the PoS [101]. This is a concept that was already being used by Bitcoin for transaction prioritization but was developed more thoroughly by PPCoin, which represents the first implementation of a Proof of Stake algorithm [42]. Typically, with this type of algorithm, the reward for validators consists of an amount of cryptocurrencies proportionate to the stake deposited. One of the possible problems that may arise with the use of PoS is that of "nothing-at-stake". In the case of Proof of Stake, validators do not risk losing their invested resources, which gives nodes an incentive to continue the staking process. In the case that a fork is present, validators can support multiple versions of the blockchain without risking consequences or loss of stake. An event like this can lead to critical situations, which can make the system vulnerable to malicious attacks [81].

Blockchain types

There are several different types of blockchain, and the best known are Public, Private, Consortium, and Hybrid [3, 23, 63, 106]. Public blockchains, also called permissionless, have no restrictions on joining. In fact, anyone can become an authorized node [40]. The features that this type of blockchain offers are the possibility for the user to view the transaction history and take part in the consensus mechanism without needing any permission [88]. The major advantage of this type of blockchain lies in its independence from any type of organization and the transparency it offers. In addition, the ability to take part in the consensus mechanism makes it extremely complex to enact a security attack since a bad actor needs at least 51% power of the network [71]. The main drawback is performance, which can be low due to factors such as scalability or the complexity of the consensus mechanism in use in the system [91].

Private, or permissioned, blockchains act very similarly to public ones in terms of connection types and level of decentralization. The difference lies in the fact that these platforms are closed and under the control of a central entity which organizes and manages access. These networks tend to be much smaller, as scalability is not an important factor for these types of systems [77]. The main advantage of a blockchain managed by a central authority lies in security, as this entity can define the accessibility to the network and the levels of permissions granted. In addition to this, it also enjoys higher performance, when compared to a public blockchain, given its closed and restricted nature. The disadvantages lie in the fact that centralized

nodes can determine for themselves what is valid, by inserting false information into the network, since participation in the consensus mechanism is not open to all nodes in the network [106].

Consortium-based blockchains are systems similar to private ones but have limited access to a restricted group. Nodes that can take part in the consensus mechanism are preselected [40]. Consortium-based blockchains offer a higher level of security than the other types. Downsides include low transparency and the fact that if a member node is compromised, limitations could be created due to the regulations imposed by the system [23].

Hybrid blockchains try to bring together the advantageous features of public and private blockchains to create a system that is both reliable and secure [77]. These types of blockchains are based on a public network, within which private systems can be created where permissions are required to access them. Being based on a public network, the data is visible, but restrictions can be implemented, ensuring a high level of customization [88]. Some positive aspects lie in the good performance that this type of blockchain offers and good scalability compared to the public blockchain, as well as providing an excellent degree of transparency [71].

Smart contracts

One of the most interesting aspects of blockchain technology is smart contracts. The first to introduce this concept was Nick Szabo, who argued that it was possible to integrate a wide variety of contracts, relating to multiple real-life use cases, into hardware and software systems in such a way that it would be very difficult to violate them [94]. Szabo represented smart contracts by offering the following definition, "A smart contract is a computerized transaction protocol that executes the terms of a contract." In work published later, Szabo offers a new definition: "A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises." [93]. Different definitions have sprung up over the years to describe smart contracts. Currently, a smart contract is considered an intelligent agent, a computer program that can make decisions when previously defined conditions are met. Specifically, the intelligence of the agent is defined by the complexity of the operations and controls it can perform [43]. Initially, smart contracts were born as an isolated concept, but it is with the advent of blockchain technology that they are expressing their true potential. Considering the smart contract as a program that runs on the blockchain, it can be executed on a network to build trust between nodes that are part of it without the need for trusted third parties [115]. A smart contract receives a transaction as input and executes the corresponding code, triggering and changing the result to output based on it. Smart contracts allow for such a level of security that enables the blockchain network

to send transactions between peers and keep the ledger information in a secure and reliable environment. In addition, smart contracts are characterized by the properties of being trackable and irreversible, providing a high level of transparency. Another key aspect of smart contracts is that once created they do not need to be monitored and allow processes to run autonomously without the need for any additional intervention [57]. As mentioned in one of the previous sections, the diffusion of smart contracts is mainly driven by the creation of Ethereum [12]. Ethereum introduced Solidity, a high-level programming language for smart contract development [16]. Solidity has played a key role in the evolution of smart contracts, enabling the implementation of increasingly complex contract terms through data structures, functions, and events. Solidity was born to provide a language that was very similar to Javascript and C++ and had a high level of readability and ease of understanding. Furthermore, Ethereum smart contracts encoded with the Solidity language are executed on the Ethereum Virtual Machine(EVM), an environment that enables the execution of these contracts and their interoperability. Such a virtual machine also allows the execution of the same contracts on multiple blockchain platforms, obviously based on EVM [12].

Smart contracts allow contract clauses to be converted into executable code, providing transparency and predictability of what happens if these clauses are met. Developers can specify permissions for each function implemented, thus ensuring the level of openness and security that is desired. Furthermore, each individual execution of a contract is saved immutably on the blockchain by sending a transaction [114]. Generally, a smart contract experiences a life cycle consisting of four main phases, creation, freezing, execution, and finalization. The first phase consists of encoding the clauses that you want to implement and deploying the contract on the ledger. The freezing phase consists in which it needs to receive majority confirmation from the nodes in the network, so as to avoid network flooding. The third phase is the execution phase, in which the contract can be invoked and its functions executed. This phase results in the creation of a set of new transactions. The set of transactions is submitted to the ledger and goes through the validation phase by the consensus mechanism. The last phase of its life cycle is finalization, in which all transactions have been validated and added to the blockchain [89]. The possible applications of smart contracts are very varied and constantly increasing. This particular feature of blockchain technologies can be used for decentralized finance, ensuring greater security between the parties involved in the trades and increasing their efficiency, for supply chain management, allowing for greater data transparency and resilience, or for the Internet of Things(IoT) facilitating the exchange of information between different devices [69, 74, 86].

In addition to the different applications of smart contracts, there are several

challenges and limitations, which smart contracts have to face [108]. Among these, is the "Reentrancy vulnerability", which consists of a recursive call to the contract that performs continuous withdrawals, which can lead to the total consumption of the contract resources [84]. "Transaction-Ordering Dependence (TOD)", is a problem that occurs when a single block contains multiple transactions that depend on each other, making them vulnerable to attack if they are not executed in the correct order, as the order is decided by the miner [34]. The "Timestamp Dependence" is a problem that can occur if there are conditions of the contracts that are activated based on the timestamp, which is usually defined by the miner's local system and therefore can be exploited by attackers, using timestamps of different blocks, to manipulate the result of a call to the contract [35]. Finally, there are problems related to privacy, mainly due to the storage of the transaction history on the blockchain through which it is sometimes possible, by analyzing the transaction structure graphs, to obtain private user information, such situations are defined as deanonymization attacks [80].

Oracles

Automation of smart contracts is a crucial factor in the development of DeFi systems, and if we exclude data from the outside world, the feasible use cases are extremely limited. Precisely to find a solution to this problem and thus expand the possibilities of smart contract use cases, oracles were born. An oracle represents an entire ecosystem that allows collection, transfer and input of real data, to and from decentralized applications, particularly to smart contracts [13].

Oracles are an important aspect of blockchain technology, and the role they play is as a bridge between smart contracts and information from the world around us, also serving as a tool for authenticating, verifying and querying this data. The need has arisen to introduce oracles precisely to make up for the blockchain's inability to have access to data external to it. There are different types of oracles adaptable to the different application contexts that may arise during the development of decentralized applications. Among the most common are Inbound oracles and Outbound oracles [11]. These types of oracles differ in the mode of interaction they offer; the inbound ones transfer data from the real world to the blockchain, and can, for example, be leveraged for the transfer of financial data, for example, obtained through an API call on the Internet, within a smart contract, which will perform certain actions based on the data received.

In contrast, outbound oracles enable the transfer of information from smart contracts to the outside world in order to trigger specific actions, as is often the case with DeFi systems that exploit IoT technology [2]. Another important distinction that needs to be made, is between centralized and decentralized oracles. A centralized oracle is controlled by a single entity and is the sole provider of information,

this however has an important criticality since the blockchain cannot verify the truthfulness of the data, subjecting smart contracts to the risk of receiving incorrect or even fraudulent information.

A decentralized oracle, on the other hand, will exploit the presence of multiple sources of information in order to verify the integrity of the data being entered into the blockchain [8]. From a more technical point of view, the connection between the outside world and the blockchain is made by two components, one off-chain and one on-chain, which communicate with each other and together represent the oracle. The off-chain component is represented by any code that receives information from a source, such as an API, while the on-chain component is represented by a smart contract, which represents an access point to the blockchain [70]. The major limitation that arises from adopting Oracle-based solutions is the same one that is most present in the case of centralized oracles, which is the authenticity of the data. This limitation is called the "Oracle problem" and is constantly being studied to achieve a system that is as reliable as possible [14].

Tokens

An interesting and extremely widespread property of blockchain technology is the token. A token is the digital representation of an asset and offers a high level of customization which allows it to adapt to many use cases of decentralized finance and more generally of blockchain technology [87]. One of the very first traces of the concept of token dates back to the Bitcoin whitepaper, considered by many to be the first example of a cryptocurrency and digital currency [61]. Subsequently, a big step forward for the diffusion of tokens within decentralized applications was made thanks to the introduction of the ERC20 standard developed by Ethereum, which allows the creation of cryptocurrencies and fungible tokens exchangeable on the Ethereum network [103]. Subsequently, Ethereum also defined the ERC-721 standard, through which it is possible to create non-fungible tokens to which a logical value can be attributed based on the context of use [25].

The high level of customization of tokens is due to the various functionalities they can offer. Indeed, tokens can be used as a current currency and thus a medium of exchange for goods and services [67]. Or as expendable or exchangeable assets for the purchase of goods and services [97]. They can also be called "Utility tokens", tokens intended for example, to represent a level of satisfaction regarding a specific good or service [56]. Finally, a final important utility is related to the security aspect, exploited for example to secure ownership of an asset, or decision-making power in the context of the case study [47]. Another important aspect that characterizes tokens is their nature. There are fungible tokens, meaning that each individual unit is identical and can be replaced with another unit of the same value, a concrete

example being fiat coins [15]. The second type of tokens, particularly popular in recent years, are non-fungible(NFT) tokens, which unlike the first type have unique characteristics that make them non-repeatable and are often used to represent digital assets such as works of art [107]. Finally, there are the semi-fungible tokens, a type that shares characteristics of the first and second natures [75].

2.1.4 InterPlanetary File System

The InterPlanetary File System (IPFS) is a particular protocol born to create a large global network of peer-to-peer nodes that allows each computer that takes part to upload and download files in a distributed way. Within the peer-to-peer network, resources are identified through an address derived from the hash of the content of the resource itself, called Content Identifier (CID) [10]. IPFS is essentially a distributed hash table (DHT), a decentralized structure within which data are distributed based on their hash values and where each node is responsible for storing a portion of the total set of stored data [37]. In the case of IPFS, the different CIDs related to the stored data are mapped to the node responsible for it, which is also represented through an identifier called PeerID that is assigned at the time it joins the peer-to-peer network in the process called Peer Addressing. The "Merkle Directed Acyclic Graph (DAG)" concept is central to the operation of IPFS. When a new file is uploaded by a peer on the network, it is split into smaller chunks, each of which is assigned its own CID. These CIDs are then integrated into a Merkle DAG, forming a tree structure representing the relationships between these blocks. This interconnected structure allows you to easily reconstruct the original file and also allows you to maintain a sort of version history through which you can access different versions of files over time. The process of creating these Merkle DAGs is one of the key concepts of IPFS and has the name of "Content Addressing".

IPFS offers among its benefits secure and duplicate-free file sharing through data encryption, has a high level of scalability given the peer-to-peer nature of the network, and greatly reduces reliance on central servers. It is also a very resilient and failure-resistant system as all the contents are distributed along all the nodes of the network [98]. This technology offers great development opportunities. Thanks to its properties, it can be used, for example, as an off-chain storage system for blockchain-based projects or exploited in the IoT environment, allowing better management of data between interconnected devices [24].

2.2 Related Work

Blockchain and InterPlanetary File System technologies are often used together as they offer a high level of compatibility. A very interesting research is the

one described by Marangone et al. (2023) where a Multi-Authority Approach to Transaction Systems for Interoperating Applications (MARTSIA) is proposed [52]. The system exploits the Ethereum blockchain as a medium for process execution through the use of smart contracts containing resource locators related to information stored in an IPFS-based P2P repository. The resources are made available to their respective owners in an encrypted manner, and only they can provide the decryption keys. Hao et al. (2018) propose a system for managing the traceability of agricultural products based on the Internet of Things(IoT), Blockchain, and IPFS [32]. This research aims to avoid falsification of data regarding the origin of products and thus safeguard food safety. To do this, a system is proposed that monitors products in real-time and saves the data collected thanks to IPFS, subsequently exploiting the Ethereum Blockchain to store the hash generated by this data. Another example of a system based on Blockchain and IPFS is the one described by Di Francesco et al. (2023) [22]. The idea proposed in this research, called "Kryptosafe", exploits the IPFS system for storing datasets, regulating access to them through smart contracts developed on the Ethereum blockchain, and ensuring a decentralized and reliable system for data management and exchange.

Business Process Management often produces a large amount of data that needs particular attention, especially in the world in which they are stored. For example, Verdonck et al. (2020) propose a solution for the management of Electronic Health Records (EHR) through the use of the InterPlanetary File System and blockchain technology [102]. The use of IPFS solves the inefficiencies of blockchain technology in storing large amounts of data while also avoiding the high costs associated with the use of smart contracts for storing records. Hamida et al. (2019) propose a decentralized IPFS-based system for storing data about institutions of higher education [31]. The idea behind this research is to make the information available so that it can be accessed by students and people who are choosing their educational path. Business process management research also includes many that exploit blockchain technology alone. For example, that proposed by Lopez-Pintado et al. (2017), describes a system that maintains the states of process instances on the Ethereum blockchain, managing workflow routing through smart contracts generated by a BPMN-to-Solidity (Business Process Model and Notation to Solidity) compiler [50]. In the area of insurance policies, Hassan et al. (2021) propose a framework for creating secure insurance processes [33]. The problems that this research aims to solve are those related to manual processes that are slow and error-prone. This solution leverages blockchain and smart contracts by implementing the necessary conditions for different insurance policies, ensuring transparency and authenticity. In the field of business process management in the financial sector, on the other hand, Molnar et al. (2023) explore the benefits of using blockchain

technology such as improvements in process efficiency and security, and propose blockchain-based models for managing credit and compensation claims in the banking and insurance fields [58].

Among the topics covered in this paper, circular economy plays an important role as it underlies the idea being presented. Also in this topic, IPFS and Blockchain play a key function in many cases, such as in the one proposed by Dasaklis et al. (2020) where is presented a traceability and auditing framework for reverse logistics (RL) of electronic equipment [17]. Specifically, the solution uses the Ethereum blockchain for traceability and IPFS for storing RL information. Alnuaimi et al. (2023) propose a system that leverages blockchain technology in order to track the path of plastic waste from the collection to the recycling stage [4]. The system also offers a reward mechanism to promote proper recycling. In this case, IPFS is used to save an image of the compressed plastic bales that will be auctioned through dedicated smart contracts. The last case that is interesting to mention regarding the circular economy and the use of blockchain and IPFS, is the research proposed by Omar et al. (2023) which aims to decrease food waste in the hospitality industry [68]. The system being proposed leverages a permissioned type Ethereum network which runs some smart contracts dedicated to managing food orders and other conditions in order to avoid waste due to inadequate planning. IPFS technology is leveraged for off-chain storage of a map containing the ingredients consumed by food and beverage retailers based on the menus being prepared. In order to incentivize proper waste behaviour, rewards and penalties are also assigned to the parties involved in the system.

Chapter 3

Motivating Use Case Scenario

The thesis work was carried out in collaboration with two very important entities within the context of the case study, the national consortium for the collection and treatment of used vegetable and animal oils and a research-oriented leading company in the field of bio-based material production from organic waste. From the next chapter, the entities just mentioned are represented by two distinct actors, the Owner and the Regeneration Company. With these two entities two live meetings [1] were held, through which the needs of the system and the requirements to satisfy them emerged. As already mentioned in the previous chapters, and described in detail in this one, the most important requirements were, more generally speaking, to create a system that would allow the tracking of UCO and RUCO movement documents, certify the quality and origin of used vegetable and animal oil batches, and the correct completion of the entire supply chain through the release of ad hoc certifications. Regarding more specific requirements, the need emerged to be able to hide some of the data within the documents, allowing a high level of privacy, and to implement a token-based reward system in order to incentivise good behaviour at every stage involved in the supply chain.

A new system called OilTracker aims to certify the origin and destination of used cooking oil. Alice is a citizen who cares a lot about correct waste disposal. In particular, used oil produced following the preparation of fried foods or resulting from products preserved in oil. When Alice produces waste of this type, she stores it in special plastic containers. If Alice is responsible for commercial activity in the food sector, she should store the waste in the same way but by filling out a specific form to be delivered to the transporter who will collect the waste. At this point, Alice disposes of the accumulated UCO in special bins, or at collection centres. These bins are placed strategically within urban areas such as near schools, churches or oratories, as they facilitate collection by citizens. Depending on the municipality

[1] Meetings held on 3 November 2023, and 6 April 2023.

of residence, the disposal operation could take place differently, such as through a door-to-door or condominium collection system.

The role of waste collection and transport is assigned to a municipal transport company. Bob, the operator assigned to transport the UCO, collects the product from the various collection points provided and assigns them a document accompanying the waste, called "Waste Identification Form (WIF)". This document is then stored within the OilTracker system, through which it will be possible to consult it at any time for information regarding the traceability of that specific batch of UCO.

Bob delivers the UCO batch to a storage facility, where it will be held until a regeneration company is ready to accept the waste. Upon determining the destination centre, Bob transports the UCO to a specialized waste Regeneration company. During this, a new version of the WIF document is issued, within which there is updated data on the waste to be recycled. The new document is saved on the OilTracker system, allowing the information on the UCO batch to be updated. The Regeneration company performs some chemical analysis to determine the quality level and verify that the product is suitable for regeneration. The regeneration process is carried out in order to transform the UCO into a regenerated raw material, which is able to avoid waste problems and can be used for the production of biofuels such as biodiesel and Hydrotreated Vegetable Oil (HVO), or other types of products such as cosmetics. The regeneration center performs analysis of certain aspects of the product such as moisture and acidity. As a result of these tests, the UCO then undergoes a chemical transformation to become a new and reusable product that goes by the name regenerated used cooking oil (RUCO), thus preventing it from having to be disposed of in landfills. Once the operations on the product are completed, the regeneration company issues a document that allows the transport and contains all useful information about the regenerated product and certifies its quality. This document, known as the "transport document", is stored within the OilTracker system to ensure the permanent certification of the product's quality and its origin. At this point, the entity that owns the system gives to the regeneration centre a digital coin as a symbol of the successful completion of the tracking process. This currency may later, at the discretion of the regeneration centre itself, be exchanged for a second currency that can be used as an exchangeable asset for obtaining specific services offered by entities outside the system.

At this point, Bob goes back to the regeneration centre and picks up the batch of RUCO. RUCO is transported to companies specialized in creating new products from regenerated waste oil. Once Bob arrives, he delivers the RUCO to these companies which are concerned with reusing transported RUCO and exploiting it to create new products, effectively implementing a circular economy system. By applying the necessary chemical transformations, the company is able to make products such as

biodiesel fuels, soaps, cosmetic skin care products, and more. In addition, each of these reprocessed products will have on them a reference to the batch of RUCO from which it was produced so that a citizen like Alice can verify the origin and path followed by the product she purchased.

Chapter 4

Design

In this chapter, the design choices and their underlying motivations for the implementation of the case study described in the preceding chapter are presented. Methodological and design choices are outlined, seeking to provide a detailed and comprehensive description of the work undertaken. Initially, attention is directed towards the overall infrastructure of the OilTracker system, briefly describing the functioning and objectives of the components involved in the project. Subsequently, the various components comprising the system are examined in detail, analyzing their respective use cases and highlighting the decisions made for each of them.

4.1 Infrastructure

OilTracker was conceived to address the issue described in the preceding chapters, providing a system capable of tracking and ensuring the integrity of data involved throughout all stages of transportation and recycling that the UCO undergoes. Among its features, it includes a reward-based system, issuing certifications as a form of recognition for the proper completion of recycling operations, among other functionalities. The system involves several actors, each of whom is involved in the processes differently:

- Owner
- Transporter
- Regeneration Company

The former is the entity that owns and is in charge of managing the system itself. His role is of fundamental importance within OilTracker since he is the direct stakeholder in the proper execution of the processes involved. Its purpose is to manage and possibly maintain certain aspects of the system components, as well as to view the

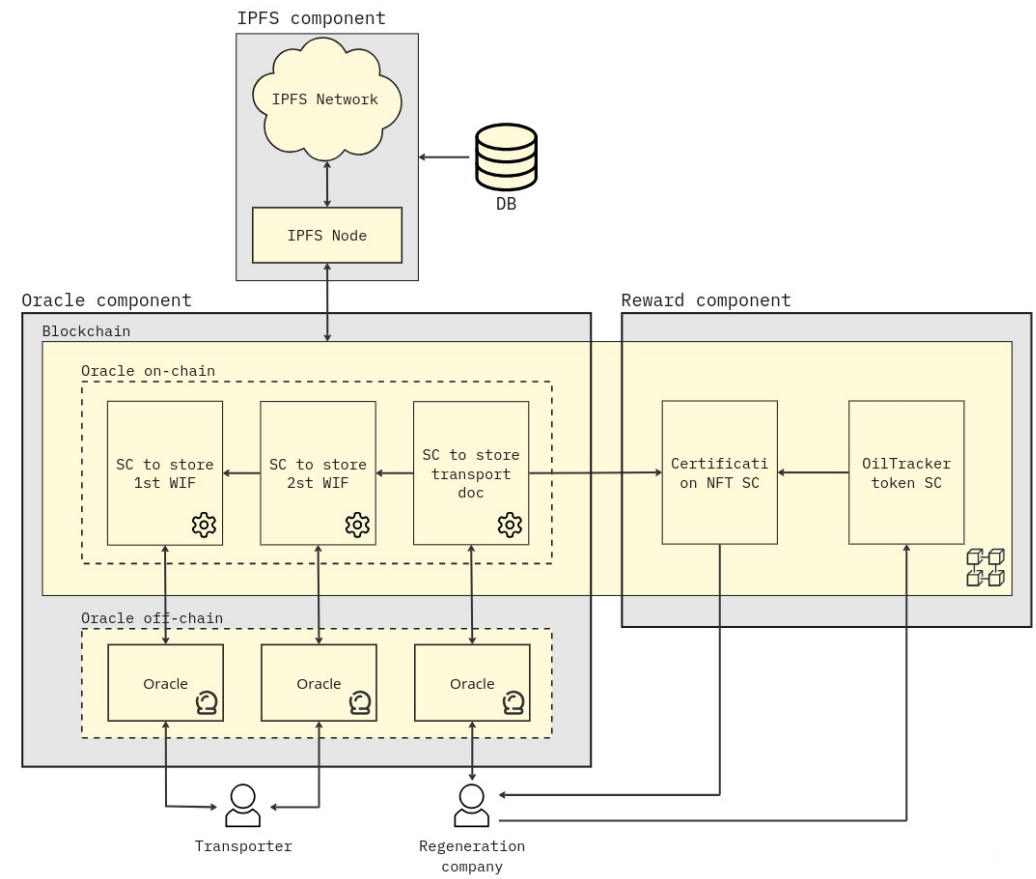

Figure 4.1. OilTracker architecture

stored data. The Transporter, on the other hand, plays a simpler role, in fact, its only task is to initiate the data storage procedure within OilTracker in the initial two phases of the tracking chain, allowing the Owner to view the data entered. The documents that this actor is charged with handling are two separate versions of the "Waste Identification Form", the first containing transport data from the producer to the storage facility, and the second from the storage facility to the regeneration centre. Finally, there is the Regeneration Company, the one who is in charge of the most crucial phase within the proposed system, as it handles the recycling and regeneration operations of the waste involved. The regeneration phase ends with the creation and subsequent storing of the "Goods Transport Document", which is the final point in the tracking sequence. In addition, the Regeneration Company is the actor arranged by the system to receive a certification of successful completion of operations in the form of a Non-Fungible Token (NFT). This NFT can also be used as a ticket to request a second token, this time fungible, which can be exchanged in the future for the issuance of certain services. The tokens involved in the OilTracker system are briefly described below:

- Certification NFT (CERT): Represents the completion of tracking operations, as well as a certificate of waste quality, carefully checked by the Regeneration Company.
- OilTracker token (OT): Represents a fungible token that can be exchanged in the future for the acquisition of goods or services.

At the heart of the OilTracker system is blockchain technology, which is closely linked to all components of the architecture. Concerning blockchain technology, the Ethereum Virtual Machine (EVM) was exploited, which made it possible to create a system compatible with multiple Ethereum platforms, a very important aspect as described in the chapter on the evaluation phase.

As shown in Fig. 4.1, OilTracker consists of three main components, which encapsulate the functionalities that the system provides:

- Oracle component, represents the element through which actors interact with the system.
- IPFS component, dedicated to maintaining the information of the tracked data.
- Certification component, part of the system dedicated to managing rewards and issuing the certification token.

4.1.1 Oracle component

As already described in the chapter 2 Oracles are a service that provides data from the real world to smart contracts, allowing them to act on that information. To do so, Oracles are composed of an on-chain and an off-chain part, which by definition interact with the blockchain and the outside world, respectively. The Oracle component represents the connection point between the user and the blockchain and plays a crucial role within OilTracker. Its purpose is to collect data from users who invoke it and then interact with the IPFS component to obtain referential information to be saved within the smart contracts, which is the on-chain part of the Oracles.

Using the blockchain requires users to possess a key pair, one public and one private, representing Ethereum credentials, which are needed to be able to send and sign transactions. The functionalities that this component offers are different depending on the user who invokes them. The Transporter and Regeneration Company can send documents assigned to them to the off-chain part of the Oracles, which will result in sending a transaction to the smart contract dedicated to handling that specific type of document. At the end of this process they also receive a refund

of what they spent. As can be guessed, it is the Owner who has the greatest freedoms within this component since they can, again through sending direct transactions to the smart contracts, decide which users are enabled to enter new data within the system, as well as view those already in the contracts. In conclusion, the interaction component must ensure the communication and proper transfer of data from the user to the storage component and then to the blockchain, also allowing communication in the opposite direction.

4.1.2 IPFS component

The IPFS component plays a key role within the OilTracker system. IPFS allows excellent archiving and management of the data involved in the case study, offering a decentralized and highly resilient storage system, thanks to peer-to-peer data sharing between the nodes of the network that composes it. The choice to use a system like IPFS was made given the importance of how and where to store the data involved in tracking used cooking oil. By its nature, blockchain is not particularly suitable for storing large quantities of data, especially if we consider the necessary expenses. For this reason, decentralized storage, the main feature of the IPFS system, offers great freedom of data management when used in conjunction with blockchain technology. The IPFS node is directly invoked by the off-chain components of the Oracles creating an indirect connection with the actors, even in the case of information retrieval.

4.1.3 Certification component

The certification component is reserved for use by the Owner and the Regeneration Company, as the Transporter has no interest in the role this component plays. This component is very important in the context of OilTracker because it makes possible to achieve the initial aim of the project by providing a system capable of certifying the completeness of the entire supply chain and issuing an element that can represent the entire tracking chain. The decision to represent this certification through the use of a Non-Fungible Token was quite straightforward as this aspect of blockchain technology is well suited for this use case, allowing the creation of a virtual object with a dual purpose, the first being to represent the correctness of the tracking chain, and the second to be a ticket with which to redeem a second coin, this time fungible. This fungible coin represents the second feature of this component, namely, a reward-based system. This token is distributed based on an exchange rate that can be changed by the Owner at any time. Due to its fungibility, this coin can be used as an exchange currency for useful goods and services, opening up the system to many future developments.

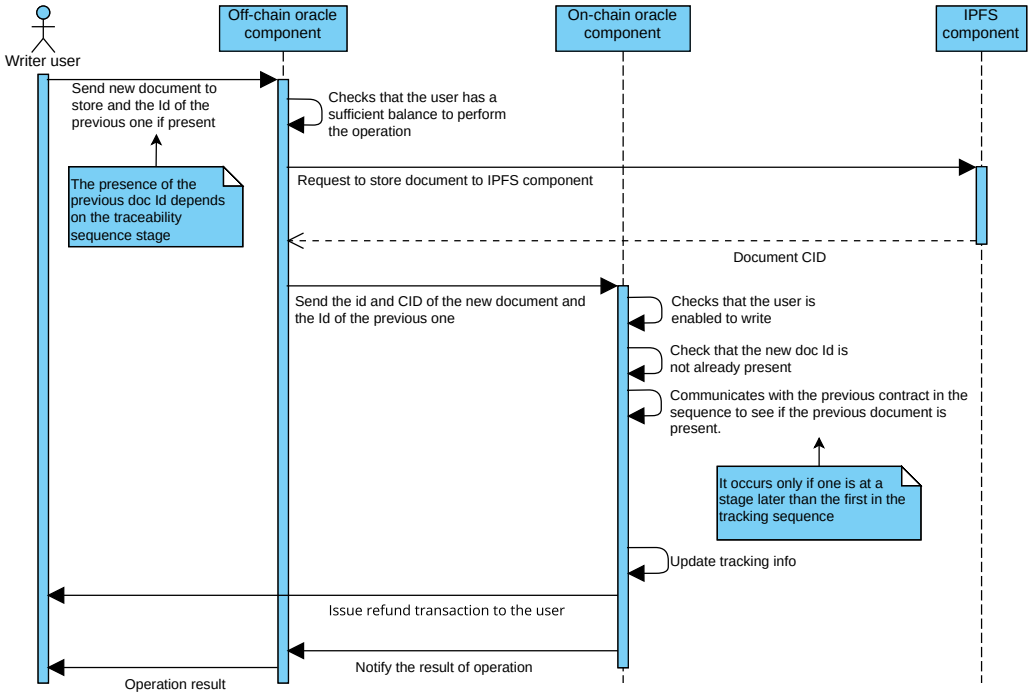

Figure 4.2. Sequence diagram to describe the addition of new tracking information

4.2 Oracle interaction

As illustrated in Fig. 4.1, the Oracles are the point of direct interaction that users have with the system. Specifically, the actors involved, the Owner, the Transporter and the Regeneration Company, interact directly with the off-chain part of the Oracles within the system. The operations in which the Oracles take part involve the steps of reading and writing data to the Blockchain, or management operations performed by the owner, such as sending funds needed for contracts to generate refund transactions, or adding and removing enabled users to the list of addresses holding read and write permissions. Below, through the use of some sequence diagrams, the use cases in which this component is involved are described.

4.2.1 Adding new tracking information

As described in Fig. 4.2 the process of adding new information can be performed by any of the described users. The interaction process is identical for all three phases and is therefore to be understood in a general way and not related to a single contract. The initiation of the interaction with the off-chain component of the Oracle is done by the user uploading a new document to the system, and if it is a phase later than the first, the user is also asked to include the ID of the

previous document in the tracking sequence. Upon receiving the data, the off-chain component performs an internal function to verify that the writer user has sufficient funds to complete the transaction. This is done before the interaction phase with the IPFS component, in order to limit cases where the document is stored on the IPFS network but the transaction directed to the smart contract fails due to lack of funds. If the check returns a positive result, the off-chain component communicates with the IPFS component, a process described in the next section about IPFS storing process, and obtains the CID value from it. Following the response from the IPFS component, the on-chain component of the Oracle is finally invoked, which performs some checks necessary to ensure the integrity and security of the data being recorded on OilTracker. As a first step, it verifies that the address of the user who initiated the storing operation is registered among the addresses enabled to write within the contract. The second check is necessary to verify whether the received document ID is already present within the system, so that information about the same document cannot be added twice, preventing this data from being modified later. The third check is done through an interaction between contracts to verify that the previous document ID is actually present in the on-chain component dedicated to the management of the previous phase, obviously, this check is performed only in the case of phases after the first one. Once all the checks have been successful, the system finally adds the new data on the on-chain component, and it is done in two different ways depending on the contract invoked. If it is the first one then the only data to be recorded are the current document ID and its CID value, while in case it is the second or the third contract, the storing step involves adding the ID of the previous document, so as to create a backward link between the saved documents. Upon completion of the update operations, the contract issues a refund to the address that sent the transaction, calculating the amount spent by the user to perform the operation of writing data to the blockchain.

4.2.2 Getting tracking information

Fig. 4.3 illustrates, through a sequence diagram, the operations involved and the interactions between the components involved, for obtaining the data stored on blockchain within the smart contract state. The user communicates with the off-chain component of the Oracle requesting the tracking information related to the ID of the document provided as input. The off-chain Oracle invokes its corresponding on-chain component by passing the data provided. At this point, the smart contract invoked performs a check to verify that the user is on the list of users enabled to interact with the contract to retrieve the data saved on the blockchain. The smart contract checks the state to verify that there is tracking data related to the provided

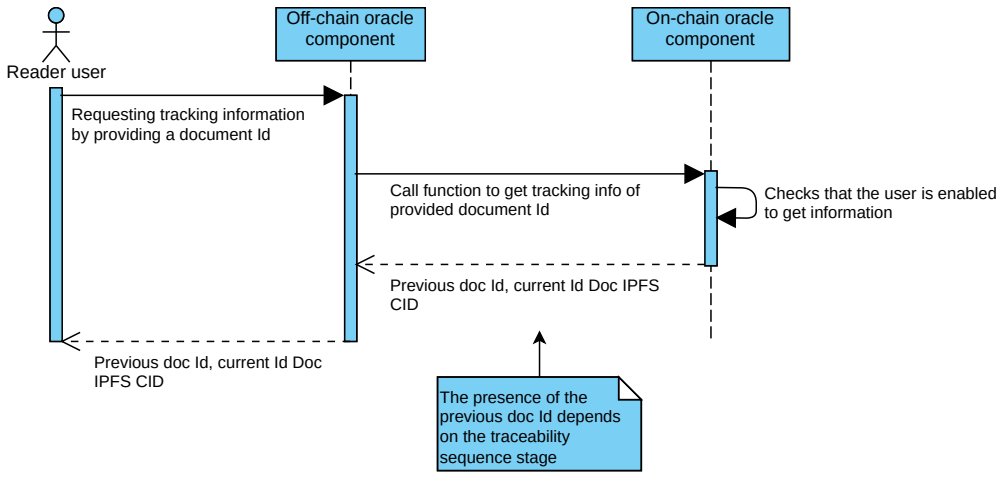

Figure 4.3. Sequence diagram to describe the process of getting tracking information

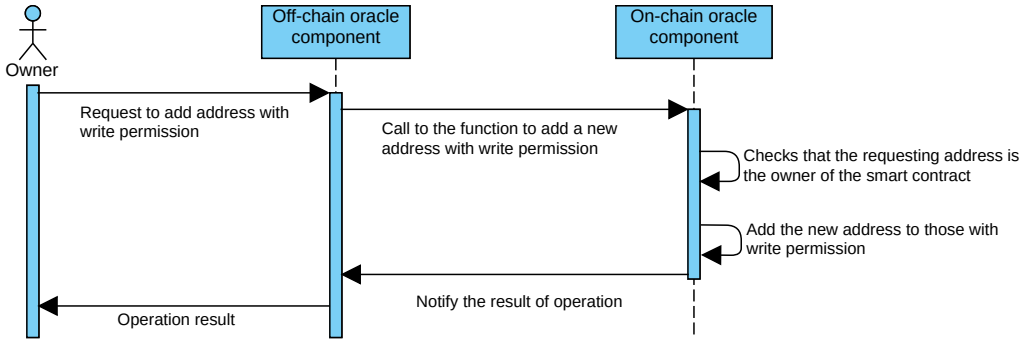

Figure 4.4. Sequence diagram to describe the addition of new address with reading and writing permissions

code. If this data is present, the information that is returned first to the off-chain component of the Oracle and then to the user itself is: The ID of the previous document (present only if the invoked contract is the second or third), the ID of the current document passed in, and the referential IPFS CID code.

4.2.3 Handling writing addresses

Among the management functions that the Owner can perform is the ability to add, and remove addresses enabled to write and read stored information. Given the extreme similarity between performing the operations of adding and removing an address, Fig. 4.4 illustrates only the procedure related to adding a new address to the list stored within the state of the contracts. This functionality also is present in all smart contracts related to the tracking phase. The Owner communicates with

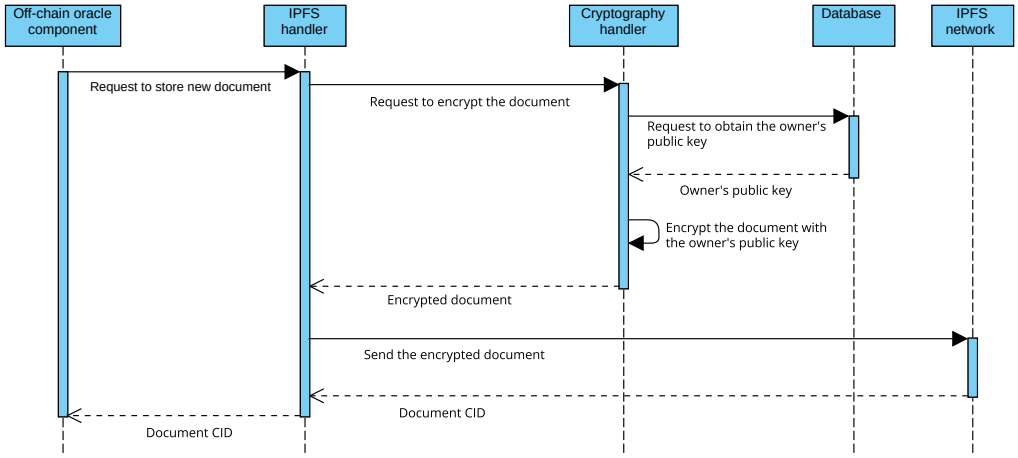

Figure 4.5. Sequence diagram to describe the storing process on IPFS

the off-chain component of the Oracle, invoking a function included in the smart contract, to which it provides a public Ethereum address as an input value. Once the call is received, the smart contract verifies that the user is the Owner of the contract, i.e., the only actor empowered to perform this operation. If the check returns a positive result, the list of write-enabled addresses is modified by adding the input parameter. After completing the update operation, the contract notifies the result of the operation to the off-chain component, which displays it to the user. As already explained, the sequence of interactions between the parties involved is the same even in the case of removing an item from the write-enabled address list.

4.3 IPFS interaction

As mentioned in section 4.2, the data, before being stored on the smart contracts and therefore on the blockchain, is processed in two different phases described in this section. The first is a data encryption phase, aimed at preserving the privacy and anonymity of the producers or holders of the used cooking oil whose tracking is being recorded, while the second one deals with the storage of the encrypted data on the IPFS network and the consequent generation of the reference code for such data, called “Content Identifier(CID)”.

4.3.1 Data storage

The storage process, as described in subsection 4.2.1, is initiated indirectly by users writing on the blockchain, but concretely is handled by the off-chain components of the various Oracles in OilTracker. This process in its entirety is described in Fig. 4.5. The first action that occurs is for the Oracle off-chain to send a request to

the IPFS handler component to start the storage process, providing as input the document to be saved. At this point, the IPFS handler provides the document to the Cryptography handler component, which communicates with the database in the system to obtain the Owner's public key, through which to encrypt the data. The Cryptography handler parses the document provided as input to extract the structure and then encrypts only the values of individual fields, thus keeping the structure itself intact. This choice was made for two reasons: The first relates to a logical point of view since by doing so it will be possible in the future, if necessary, to encrypt only some of the data within the documents and leave the others visible; the second, on the other hand, relates to a more technical aspect, in fact since it is asymmetric encryption it was not possible to encrypt the documents in their entirety, so this solution was much more suitable for the use case. By processing the document provided as input in the manner described, a second version is obtained, which keeps the same structure but has encrypted strings as values. That document is returned to the IPFS handler, which sends it to the IPFS network by interacting with an IPFS node. The document is divided into smaller blocks of data, called "chunks", and distributed over the network of IPFS nodes. After uploading, a Content Identifier (CID) is generated, computed with a cryptographic hash function based on the content of the uploaded data. At this point, the generated CID code plays the role of referential code for documents uploaded by the user through the first interaction with the Oracles. Decentralized storage also allows access to the stored data at any time, thanks to the CID code stored in the various on-chain components of the Oracles.

4.3.2 Data retrieval

The retrieval of documents stored on the InterPlanetary File System network is a very important operation for UCO and RUCO tracking. It is through this operation that the Owner can view that the data stored in the different stages of tracking are consistent with each other. Most likely this operation is initiated by the Owner since he is the only actor who can decrypt and thus make the stored data readable. Fig. 4.6 illustrates all the steps and information involved. In addition, the sequence diagram also includes the part related to obtaining the referential CID code, as already described in Fig. 4.2, since it is necessary for the initiation of the procedure in question. The user initiates the procedure by sending a data recovery request to the IPFS handler, including the CID code of a document. Using this code, the IPFS handler easily retrieves the encrypted document during the step described in Fig. 4.5, as it uniquely identifies it on the IPFS network. At this point, the IPFS handler sends the document to the Cryptography handler, which retrieves the user's private

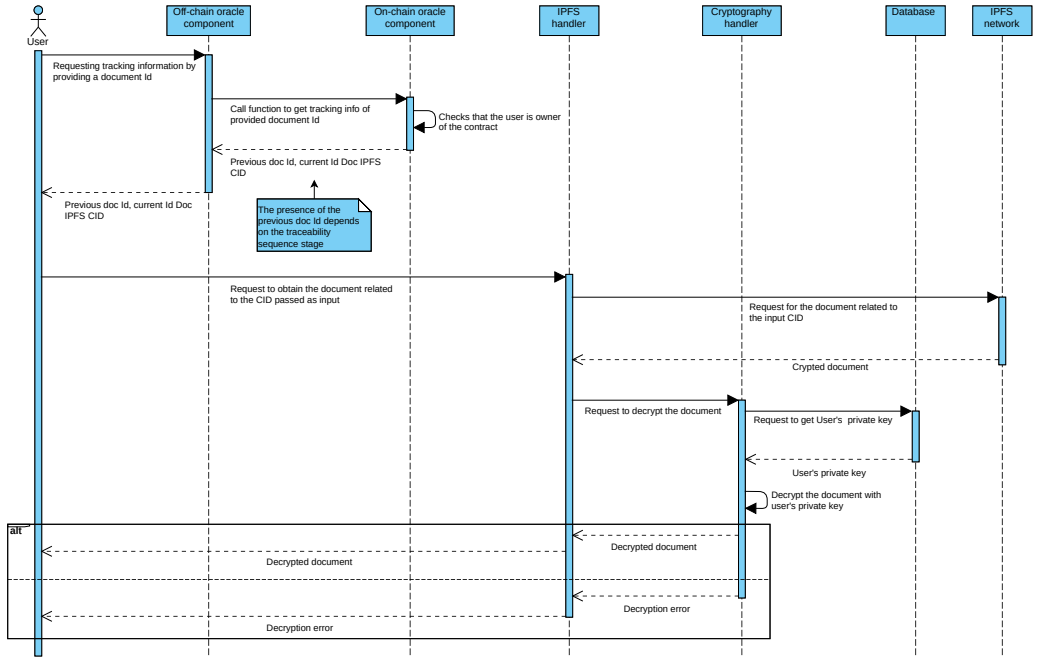

Figure 4.6. Sequence diagram to describe the retrieval data process on IPFS

key through interaction with the OilTracker database and once obtained, uses it to decrypt the document. As in the case of saving, the Crypthography Handler analyzes the structure of the document, so that it performs the decryption procedure only on the values within the fields. This operation can lead to two alternative situations: in the case that the requesting actor is the Owner, then the private key obtained from the database will be the correct one to decrypt and the returned document will be a plaintext readable version of the same; In the case, however, that the actor is not the Owner, then the private key will not be the one paired with the public key used in the subsection 4.3.1, and the Cryptography handler will return an error.

4.4 Certification and reward system

The certification system is divided into two different sequences of operations, which play an independent role, although they need to interact with each other to carry it out. The first concerns the issuance of the Certification NFT, while the second is that of the OilTrackerToken as a reward element. All the steps and controls required for the execution of these two processes are illustrated below using sequence diagrams.

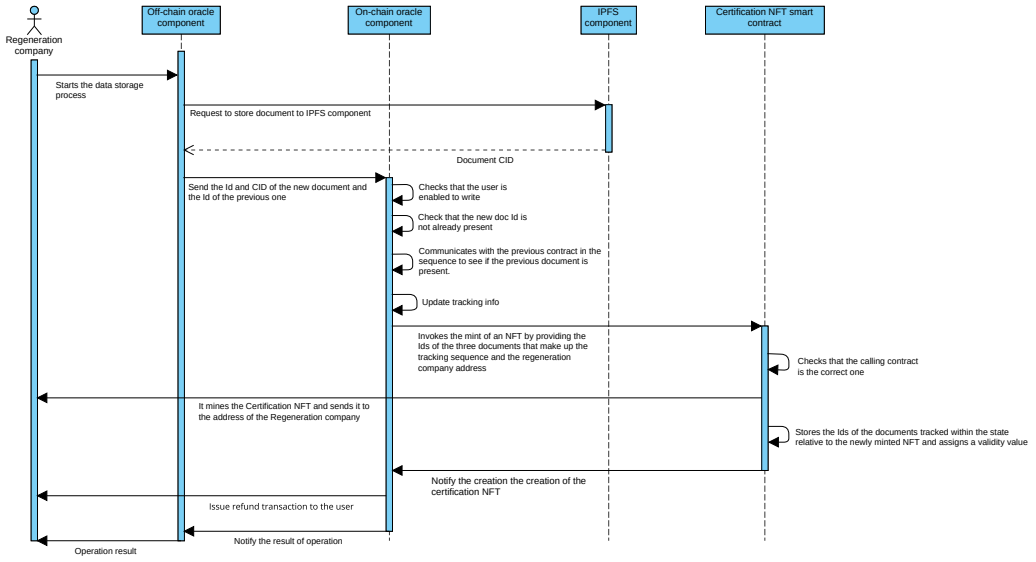

Figure 4.7. Sequence diagram to describe the mint process of the certification NFT

4.4.1 Certification NFT

The certification process is closely linked with the final stage of the tracking sequence. In fact, this is initiated before the completion of the operations necessary for adding new information to the blockchain. The third on-chain component differs in some aspects from the first two. Fig. 4.7 illustrates the process involved in this use case, and as can be seen, the first operations are those related to saving new tracking information. The process leading to the assignment of the Certification NFT(CERT) is initiated by the smart contract for storing the tracking data of the transport document issued by the Regeneration Company. In contrast to what is described in subsection 4.2.1, before issuing the transaction fee refund to the user, the on-chain component of Oracle invokes the contract implementing the Certification NFT, passing it the IDs of the documents that form the just-completed tracking sequence and the address of the user who initiated the storage procedure. The smart contract of the Certification NFT verifies that the calling address matches that of the third on-chain component. At this point, the contract mints the NFT by placing the address of the Regeneration Company as owner. It then stores within its state the IDs of the tracked documents, associating them with the ID of the newly minted CERT. In addition to this data, it also associates a positive validity value, the purpose of which is described in the following subsection. The process concludes with the return to the original execution by the on-chain component of the Oracle, which can finally send the transaction cost refund to the user computed as a result of all operations performed.

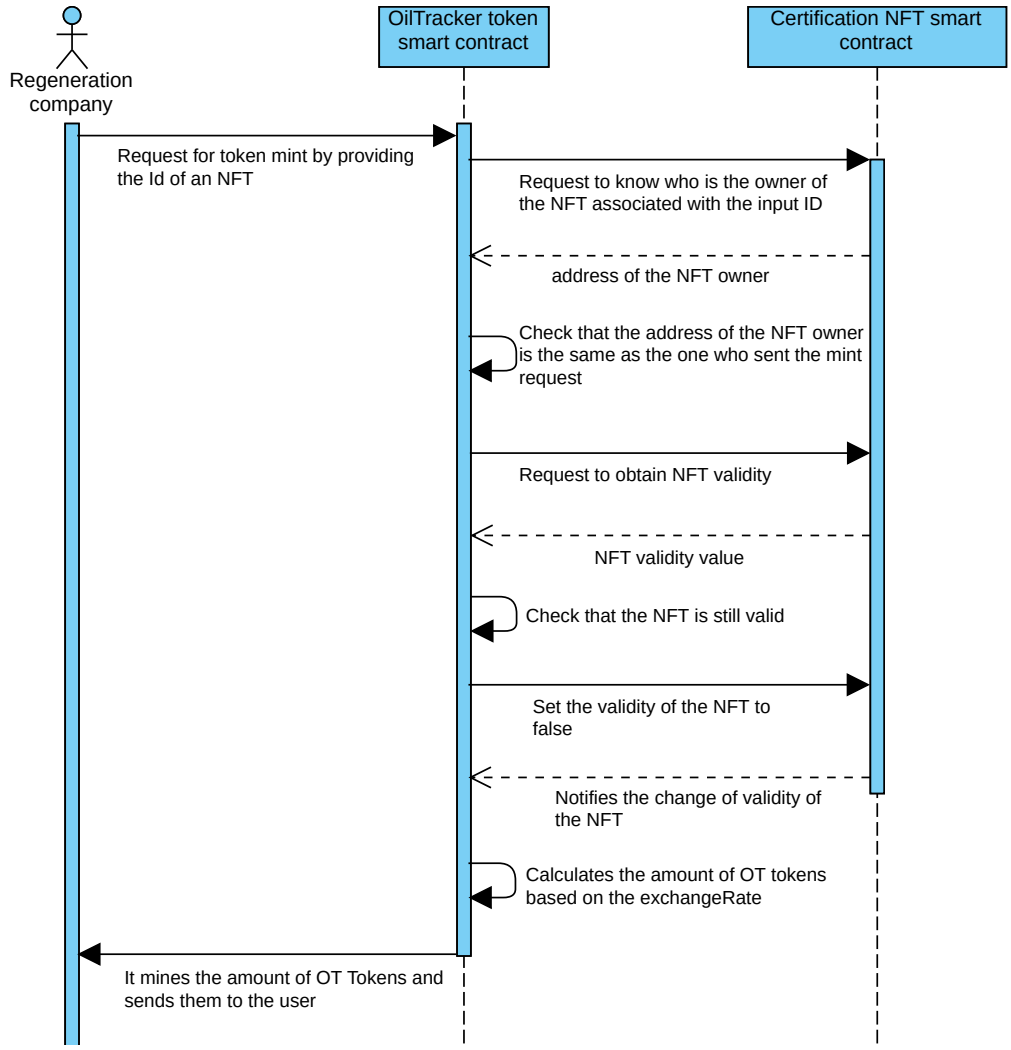

Figure 4.8. Sequence diagram to describe the mint process of the OilTracker token

4.4.2 OilTracker token

The OilTracker token(OT) represents the reward that can be released upon completion of the tracking sequence. Due to its fungible nature, a real value can be assigned to this token. Fig. 4.8 illustrates the sequence of operations that must be performed to mint this coin. As already described in the subsection 4.1.3 the OilTracker token is issued in a specific quantity, which can vary over time according to the description of the Owner of the contract. This quantity is called the exchange rate and is initially set at the time of contract creation and is changeable by the Owner at any time to suit the context conditions. To obtain the release of OTs, the user invokes the function that manages the smart contract mint process related to the OilTracker token, also sending the ID of a Certification NFT(CERT). Upon

receiving the request, the contract performs a few checks so as to make sure that the release of OTs is legitimate. It first invokes the smart contract that implements the Certification NFT by providing the ID of the CERT to obtain the address of its corresponding owner. Once obtained it compares it with the sender of the mint request. If the address matches, it sends a second request, in which it asks for the validity value of the CERT whose ID it provides. If the validity value also returns a positive result, the OilTracker token contract sets that value to false so that that specific CERT can no longer be used to obtain OTs. Finally, the process ends by issuing an amount of OT units to the requesting user computed based on an exchange rate.

Chapter 5

Implementation

This chapter describes the implementation phase of the case study, showing in detail the technologies and services involved in the implementation of the proposed system. In particular, the components of the architecture and the choices made during the design phase are described from an implementation point of view, highlighting the more technical aspects of OilTracker. To conclude this brief overview of the chapter, below is the link through which you can view the implemented code in its entirety:

https://github.com/rafmc98/Blockchain-based-traceability-system-in-the-context-of-used-cooking-oil-recycling

5.1 Technologies and services

For the implementation of OilTracker, a Decentralised App(DApp) was realised, in order to provide a system with a clear and user-friendly graphic interface, both for completeness regarding the work described in this thesis and for potential use by the end user. Its structure is illustrated in Fig. 5.1 and is described in detail below. The frontend side of the DApp was implemented using ReactJs [1], a JavaScript library developed by Facebook that allows the realisation of dynamic HTML-based graphical user interfaces (UIs). This library allows efficient management of the application's state, while also guaranteeing a high level of modelling, both of which proved to be fundamental during the development of OilTracker.

The backend part, on the other hand, was realised by exploiting NodeJs [2], a javascript runtime that allows executing js code and implementing servers and APIs. Another important component that forms part of the OilTracker backend is

[1] https://reactjs.org/
[2] https://nodejs.org/

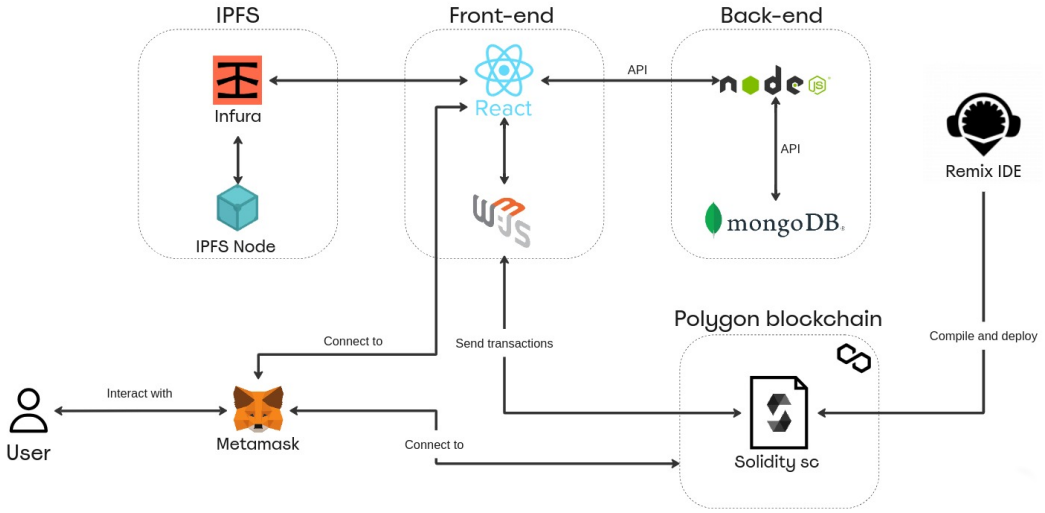

Figure 5.1. DApp architecture

MongoDB [3]. This type of database is scheme-less NoSQL and stores data in the form of documents in "Binary JSON (BSON)" format. Its scheme-less nature also gives it a high level of flexibility in data management. To give an overview, the backend is structured as follows: NodeJs acts as a server and acts as an access point for requests coming from the client, these requests are managed by Express, a javascript framework particularly suited to this purpose, a MongoDB driver enables the establishment of and subsequent communication with the MongoDB database by providing access credentials, finally the API endpoints represent the operations that the system enables. In particular, MongoDB was used to implement an authentication system based on email and password, so as to also store a pair of cryptographic keys generated at the time of registration for each user.

Regarding blockchain technology, Ethereum Virtual Machine(EVM)[4] based platforms were used. This decision was made because EVM-based platforms ensure a high level of interoperability by leveraging Ethereum standards in common with each other, as well as ensuring an intuitive and functional smart contract programming language called Solidity. In addition, these platforms are recognized for having the largest development community, which has made it quite easy to retrieve information and materials. It was decided to use Polygon as the blockchain platform for the implementation of the DApp and to leverage, in addition to this, the Ethereum and Avalanche blockchains with regard to the evaluation phase of the system, described in the next chapter [5, 12, 38].

To enable the DApp to connect and interact with the Ethereum protocol,

[3] https://www.mongodb.com/
[4] https://ethereum.org/en/developers/docs/evm/

a javascript library called Web3Js[5] was used, which provides Ethereum-based Javascript APIs that allow interaction with a local or remote Ethereum node using HTTP, WebSocket, or IPC. Through these APIs, it is possible to manage the keys of an Ethereum account, providing all the necessary tools for creating, signing, and sending transactions. The most important aspect it provided within the OilTracker system is the ability to interact with smart contracts to obtain information regarding their status, or to change it, by exploiting the `call` and `send` methods respectively.

OilTracker, as discussed in the previous chapter, consists of several smart contracts which were developed by leveraging the Solidity[6] language. Solidity is a high-level, object-oriented programming language adopted by several blockchain platforms to implement smart contracts, including Ethereum from which it originated. This language allows the use of dynamic state variables within contracts, functions and modifiers. All these development features allow for the logical and structured implementation of different use cases in the DApp world. Another very important aspect that this language possesses is the fact that it runs on the EVM, which allows it to be compatible with other EVM-based platforms as well, ensuring a high level of interoperability. Its syntax is very intuitive and similar to that of Javascript which makes it an easily understood language. As described already in chapter 2, it provides some standards that allow an intuitive implementation of fungible and non-fungible tokens, an important aspect of the implementation of OilTracker's certification and rewards system.

For the creation of the smart contracts, Remix IDE [7] was used, an online development environment designed for the implementation of smart contracts on the Ethereum blockchain through the Solidity language. It enables the compilation and debugging of smart contracts encoded in its environment. This development environment was used for the writing, compilation, deployment and subsequent interactions of all the smart contracts of which OilTracker is composed, during the development and testing phases. Another important aspect of this IDE is the possibility of integration with Metamask [8], a virtual wallet that can be used via a browser extension and which allows transactions to be handled on Ethereum-based networks. This wallet was fundamental to the development of smart contracts on Remix, as it made it possible to easily manage several Ethereum accounts, use these accounts to interact with smart contracts, and easily change the network on which to execute contracts, in order to carry out evaluations of the performance of the different networks.

[5] https://web3js.org/
[6] https://docs.soliditylang.org/
[7] https://remix.ethereum.org/
[8] https://metamask.io/

Interaction with the IPFS[9] network was mediated by the use of the javascript library "ipfs-http-client", which provides methods for establishing a connection with an IPFS node and sending files to it via HTTP requests. This library turns out to be a very important tool for integrating IPFS within applications, also allowing responses from the IPFS node to be managed following an interaction with it. Finally, to conclude this overview of the technologies and services used in OilTracker, it is important to mention Infura[10], a cloud-based infrastructure service that acted as a connection medium for the IPFS network, allowing communication with a local IPFS node rather than having to configure and manage a local one.

Encryption of documents stored on the IPFS network is handled through the "node-forge" [11] library, which provides several implementations of cryptographic algorithms that can be used on applications developed with NodeJs. The choice of this library is due to its flexibility, and ease of use.

Finally, concerning tracking documents, these are all stored in "Extensible Markup Language (XML)" format and follow a precise structure outlined by current regulations regarding waste disposal documentation. In addition, the XML format makes it possible to implement any controls regarding the type of data or to provide a domain of values that a field can assume.

5.2 Oracles

This section describes functions related to the operation of oracles, referring to code segments. In particular, functions related to blockchain storage of CID codes, and an overview of the differences between the smart contracts with which the off-chain components of oracles communicate are shown from an implementation point of view.

5.2.1 Off-chain components

The off-chain components of the oracles included in OilTracker are three, as many as the tracking documents to be stored. Since the code implemented for tracking the Transport Document is very similar to that for the second WIF document, only the code for the first two steps of the tracking chain, performed by the Transporter, is described in this subsection. The operations performed by the Transporter consist of loading two different versions of the WIF, and the off-chain components of the oracle through which they are handled have been merged and differentiated through appropriate controls.

[9] https://ipfs.io/
[10] https://infura.io/
[11] https://www.npmjs.com/package/node-forge

This single off-chain part of the oracle is represented by a `class component` React. An object of this class is instantiated by providing, the Application Binary Interface (ABI) of the first and second Solidity contracts for recording the WIF data, the corresponding public addresses of these smart contracts, the account of the user who is interacting with the system, the document file to be stored, and the code related to the previous document if it was present. Once the user through a graphical interface uploads or manually enters the data related to a WIF document, they can optionally, depending on the tracking step they are in, also enter the ID code of the previous document and then start the function `storeWifCid`. Depending on the presence of the ID code related to the previous WIF document, an instance of the first or second contract is created. After this operation, the function `checkBalance` is called by providing the instantiated contract as a parameter. This is necessary to verify that the user has the necessary funds to proceed with the blockchain storage operation before the WIF document is sent to the IPFS network. This function retrieves the user's balance in wei by exploiting the `web3.eth.getBalance` function. After that, the function `addWifCid` of the instantiated smart contract is called, providing as parameters two placeholder values of the CID and document ID. Of this operation, an estimate of the gas needed is made through the function `gasEstimate` to return a value, which when compared with the user's balance, makes it possible to understand whether or not it is possible to continue with the procedure.

At this point, execution returns to `storeWifCid` which invokes the `storeWif` function provided by the IPFS handler to store the uploaded document and obtain the corresponding CID. This operation is explained in detail later in this chapter. The execution finally ends by creating and sending a transaction to the on-chain component of the oracle, i.e., the instantiated smart contract. To it, the CID code of the document stored on the IPFS network and the document ID, if any, related to the previous stage of tracking are sent.

```
1  async storeWifCid() {
2      const web3 = new Web3(window.ethereum);
3      let contract = null;
4      if (!this.state.prevDocId) {
5          contract = new web3.eth.Contract(this.state.firstAbi, this.state.
               firstOnChainAddress);
6      } else {
7          contract = new web3.eth.Contract(this.state.secondAbi, this.state.
               secondOnChainAddress);
8      }
9      const balanceResult = await this.checkBalance(web3, contract);
10     if(balanceResult !== undefined) {
11         if (!balanceResult) {
12             console.log('Insufficient balance to cover gas cost');
13             return
14         } else { console.log('Sufficient balance to cover gas cost'); }
15     } else return
```

```
16      const result = await ipfsHandler.storeWif(this.state.fileToUpload);
17      const ipfsCid = result.ipfsCid;
18      const idDoc = result.idDoc;
19      try {
20          let gasEstimate = null;
21          let myData = null;
22          let recipient = null;
23          if (!this.state.prevDocId) {
24              gasEstimate = await contract.methods
25                  .addWifCid(ipfsCid, idDoc)
26                  .estimateGas({from: this.state.account});
27              myData = contract.methods.addWifCid(ipfsCid, idDoc).encodeABI();
28              recipient = this.state.firstOnChainAddress;
29          } else {
30              gasEstimate = await contract.methods
31                  .addWifCid(this.state.prevDocId, ipfsCid, idDoc)
32                  .estimateGas({from: this.state.account});
33              myData = contract.methods.addWifCid(this.state.prevDocId, ipfsCid, idDoc).
                        encodeABI();
34              recipient = this.state.secondOnChainAddress;
35          }
36          const transaction = await web3.eth.sendTransaction({
37              from: this.state.account,
38              to: recipient,
39              gas: gasEstimate,
40              data: myData,
41          });
42      } catch (error) { console.log(error); }
43  }
```

Listing 5.1. Javascript code for invoking the on-chain components of oracles.

5.2.2 On-chain components

The smart contracts with which the oracles interact are described below. Again, only the functions related to the first two smart contracts are shown, highlighting the different aspects and those in common.

The first smart contract, called `FirstWifStorage`, is responsible for storing the data of the first Wif document that the Transporter actor enters. The second one, called `SecondWifCidStorage`, as easily guessed, stores the tracking information of the second WIF document uploaded by the Transporter. Both implement the `addWifCid` function, with the difference that the first contract receives only the CID code and ID of the current document, while the second, in addition to these two parameters, also gets the ID of the previous document. A modifier called `onlyAllowedAddress` is applied to both versions of the function through which it is checked whether the user is enabled to perform that function. In addition to this check, both contracts perform a check on the mapping for information storage to verify that no data exists for a document with the same ID so that no subsequent changes are allowed. As a result of these checks, the function `addWifCid` in the

first smart contract updates a mapping object within the contract state, called `wifStorageInfo`, placing the document ID as the key, and the CID as the value, both in the form of strings. Following this operation, the contract calculates how much the user spent on the execution of the transaction and issues a refund. In the case of the second contract, the invocation of the function `addWifCid`, involves some differences. Specifically, the `SecondWifCidStorage` contract maintains an instance of an interface of `FirstWifCidStorage` through which it can access the `getWifInfo` method of the first contract. This function is invoked by passing as input the ID of the previous document to check for its presence within the mapping field described earlier for the first contract. This operation is necessary to maintain the integrity of the tracking chain and prevent documents related to a certain phase from being mistakenly loaded into a different phase. If the check is also successful, a mapping field, called `WifCidMapping`, formed by the ID of the document as the key and an object called `WifTrackingObj` as the value is modified. This object keeps the ID of the previous document and the CID of the current document, both in the form of a string. Subsequent to this change, a second mapping field is also updated, called `DocIdMapping` which stores the ID of the previous document as the key and the ID of the current one as the value. This mapping was necessary in order to be able to recreate the entire tracking chain later, starting from whatever document ID is desired. As in the case of the first contract, a refund is calculated and issued to the user.

The retrieval of the stored information is very simple from the point of view of off-chain invocation since it consists of a simple interaction of type `call` to the on-chain contract, so it is described in detail in the following section. For the same reason, the invocation of the methods for adding and removing addresses with read and write permissions is also described exclusively in the next section.

As for retrieving the stored information this is done using the same function for both on-chain components, called `getWifInfo`, which receives as input a string corresponding to the ID of the document whose information is to be retrieved. This function applies a modifier to verify that the address corresponding to the sender of the call appears to be among those enabled, and thus has read permissions. At this point the difference between the two smart contracts emerges, in fact in the first case the function simply returns the value that corresponds to the ID of the document within the mapping `wifStorageInfo`, while in the second case, an object of type `WifTrackingObj` is returned containing, as previously described, the ID of the previous document and the CID code of the ID provided as input.

As described in 4, the Owner is the only one who can perform certain operations within contracts. These include adding a new address to the list of those with read and write information enablement, and removing an address from the same list.

The functions that perform these operations are shared by all smart contracts and are called `addEnabledAddress` and `removeEnabledAddress`. The first one verifies that the address is not already present within the list and, if necessary, inserts a new element to the vector `allowedAddresses`. The second, on the other hand, also verifies that the address actually belongs to the list, and if present within it, iterates over it until it finds the address to be deleted of which a `pop` operation is performed.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.8.2 <0.9.0;
3
4  contract FirstWifCidStorage {
5      .
6      .
7      .
8      function addWifCid(string memory _ipfsCid, string memory _docId) public
           onlyEnabledAddress {
9          require(bytes(wifStorageInfo[_docId]).length == 0, "A value already exists for
               this key");
10         wifStorageInfo[_docId] = _ipfsCid;
11         uint256 refundAmount = gasleft() * tx.gasprice;
12         payable(msg.sender).transfer(refundAmount);
13         emit RefundRequested(msg.sender, refundAmount);
14     }
15     function getWifInfo(string memory _docId) public view onlyEnabledAddress returns (
           string memory) {
16         return wifStorageInfo[_docId];
17     }
18     function addEnabledAddress(address _newAddress) public onlyOwner {
19         require(!isEnabledAddress(_newAddress), "Address is already allowed");
20         allowedAddresses.push(_newAddress);
21         emit AddressAdded(_newAddress);
22     }
23     function removeEnabledAddress(address _address) public onlyOwner {
24         require(isEnabledAddress(_address), "The address provided is not among those
               enabled for writing");
25         for (uint256 i = 0; i < allowedAddresses.length; i++) {
26             if (allowedAddresses[i] == _address) {
27                 allowedAddresses[i] = allowedAddresses[allowedAddresses.length - 1];
28                 allowedAddresses.pop();
29                 emit AddressRemoved(_address);
30                 break;
31             }
32         }
33     }
34 }
```

Listing 5.2. Solidity code for the on-chain component of the first oracle.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.8.2 <0.9.0;
3
4  interface FirstWifCidStorageInterface {
5      function getWifInfo(string calldata key) external view returns (string memory);
6  }
7  contract SecondWifCidStorage {
```

```
8       .
9       .
10      .
11      function addWifCid(string memory _prevDocId, string memory _ipfsCid, string memory
            _docId) public onlyEnabledAddress {
12          require(bytes(WifCidMapping[_docId].ipfsCid).length == 0, "A value already
                exists for this key");
13          require(bytes(previousContract.getWifInfo(_prevDocId)).length > 0, "No value
                exists for this key");
14          WifTrackingObj memory newWifTrackingObj = WifTrackingObj(_prevDocId, _ipfsCid);
15          WifCidMapping[_docId] = newWifTrackingObj;
16          DocIdMapping[_prevDocId] = _docId;
17          uint256 refundAmount = gasleft() * tx.gasprice;
18          payable(msg.sender).transfer(refundAmount);
19          emit RefundRequested(msg.sender, refundAmount);
20      }
21      function getWifInfo(string memory _docId) public view onlyEnabledAddress returns (
            string memory, string memory) {
22          WifTrackingObj memory wifTrackingObj = WifCidMapping[_docId];
23          return (wifTrackingObj.prevDocId, wifTrackingObj.ipfsCid);
24      }
25  }
```

Listing 5.3. Solidity code for the on-chain component of the second oracle.

5.3 IPFS

The code related to storing XML tracking documents on the IPFS network is defined by a module called `ipfsHandler`. That module implements a function that enables communication with an IPFS node and indirectly with the IPFS network, named `storeWif`. This function is invoked by the off-chain oracle immediately after it has checked the user's balance, receiving as input the XML file to be stored. The first operation that is performed is to establish a connection with a remote IPFS node through the login credentials provided by Infura, `projectId` and `projectSecretKey`. At this point, exploiting the javascript library `xml2js`, the ID of the current document is extracted from the XML file. Following this operation, the function `encryptDocument` is used, providing it with the file to be stored, of the module `cryptographyHandler`. The function in question transforms the file into a string, again using `xml2js`, and then calls the function `recursiveEncrypt` which iterates recursively over all the fields in the XML file, and once it arrives at the field value, encrypts it using the Owner's public key stored within the system. RSA, an asymmetric encryption algorithm, was used as the encryption algorithm in combination with Optimal Asymmetric Encryption Padding (OAEP). OAEP is a padding scheme used to improve the security of the RSA algorithm by providing greater protection against cryptographic attacks that aim to infer private key information from the public key. Once the encryption phase is complete, the

function returns the new encrypted document, to the original function and then to the `ipfsHandler`. Upon obtaining the encrypted file, the function `storeWif`, initiates communication with the IPFS node, and through the `add` method sends the file to the IPFS network. From this operation, it returns the CID code calculated from the contents of the stored file. This procedure ends with the creation of an object called `params`, formed by the ID and CID of the document stored on IPFS. This object is returned by the function to the oracle that invoked the `ipfsHandler` module, which will initiate the storage of the latter on the blockchain.

The process of retrieving the stored file is done, as already described in subsection 5.2.2, by exploiting some methods of the on-chain components of the oracles. Through these methods, in fact, the CID codes of the stored documents are returned. These codes, when placed as attributes of the URL "https://ipfs.io/ipfs/<CID>", allow access to the stored XML file. This operation is in fact performed by the `ipfsHandler` through the function `getFileFromIpfs` passing the document CID code and the user's password as a parameter. This function can be invoked directly by the user, and subsequent to obtaining the file, proceeds to retrieve the private key of the file. This is done through an HTTP request to the OilTracker database using the "Axios" library, and providing as input the email and password of the account associated with the user. Once the key is obtained, the `cryptographyHandler` is again invoked, which through the two functions `decryptDocument` and `recursiveEncrypt`, returns the decrypted version of the document, provided of course that the user is the Owner and therefore his private key matches is paired with the public key with which the document was encrypted, otherwise a decryption error is returned. To conclude this operation, an automatic download of the XML document is initiated. The modules and functions just described are shown below, and in order to avoid repetition, as far as `cryptographyHandler` is concerned, only the encryption functions are present since the decryption functions are very similar from an implementation point of view.

```
1  const ipfsHandler = {
2      async storeWif(fileToUpload) {
3          const projectId = process.env.REACT_APP_PROJECT_ID;
4          const projectSecretKey = process.env.REACT_APP_PROJECT_KEY;
5          const authorization = "Basic " + btoa(projectId + ":" + projectSecretKey);
6          const ipfs = create({
7              url: "https://ipfs.infura.io:5001/api/v0",
8              headers: {
9                  authorization,
10             },
11         });
12         let idDoc = '';
13         xml2js.parseString(fileToUpload, (err, result) => {
14             if (err) {
15               console.error("Error parsing XML file:", err);
16             } else {
```

```
17              idDoc = result.formData.IdDocumento[0];
18              console.log(idDoc);
19          }
20      });
21      console.log("Encrypting document... ");
22      let fileEncrypted = await cryptographyHandler.encryptDocument(fileToUpload);
23      try {
24          console.log("Storing Wif on IPFS network");
25          const result = await ipfs.add(fileEncrypted);
26          let params = {
27              ipfsCid: result.path,
28              rfj: idDoc
29          };
30          return params;
31      } catch (error) {
32          console.log(error);
33      }
34  },
35  async getFileFromIpfs(cid, password) {
36      try {
37          const response = await fetch(`https://ipfs.io/ipfs/${cid}`);
38          const fileContent = await response.text();
39          const email = sessionStorage.getItem('email');
40          const url = "http://localhost:8080/api/getKeys";
41          const { data: res } = await axios.post(url, {email: email, password:
                password});
42          let privateKey = res.data.privateKey;
43          const decrypted = await cryptographyHandler.decryptDocument(privateKey,
                fileContent);
44          const newXML = { formData: decrypted };
45          const builder = new xml2js.Builder();
46          const xml = builder.buildObject(newXML);
47          const blob = new Blob([xml], { type: 'application/xml' });
48          saveAs(blob, 'xFir.xml');
49      } catch (error) {
50          console.error(error);
51      }
52  }
53 }
54 export { ipfsHandler }
```

Listing 5.4. Javascript code for the implementation of the ipfsHandler module.

```
1  class CryptographyHandler {
2    .
3    .
4    .
5    encryptDocument = async function (fileToUpload) {
6      let encryptedXmlString = '';
7      xml2js.parseString(fileToUpload, (err, result) => {
8        if (err) {
9          console.error("Error parsing XML string:", err);
10       } else {
11         const encryptedData = this.recursiveEncrypt(result.formData);
12         const newXML = { formData: encryptedData };
13         const builder = new xml2js.Builder();
14         encryptedXmlString = builder.buildObject(newXML);
15       }
```

```
16      });
17      return encryptedXmlString;
18    }
19    recursiveEncrypt(obj) {
20      const encryptedData = {};
21      for (const field in obj) {
22        if (obj.hasOwnProperty(field)) {
23          if (typeof obj[field][0] === 'object') {
24            encryptedData[field] = this.recursiveEncrypt(obj[field][0]);
25          } else {
26            let value = obj[field][0];
27            if (value){
28              value = this.publicKeyObj.encrypt(value, "RSA-OAEP");
29              encryptedData[field] = forge.util.encode64(value);
30            } else {
31              encryptedData[field] = '';
32            }
33          }
34        }
35      }
36      return encryptedData;
37    }
38    .
39    .
40    .
41  }
42  const cryptographyHandler = new CryptographyHandler();
43  export { cryptographyHandler };
```

Listing 5.5. Javascript code for the implementation of the cryptographyHandler module.

5.4 Tokens smart contracts

The tokens involved within the OilTracker system are represented by two smart contracts, encoded through the solidity language. These contracts are called `CertificationNFT` and `OilTrackerToken`, and are based on the ERC721 and ERC20 standards, respectively.

5.4.1 Certification NFT

At the time the `CertificationNFT` contract is created, an address is assigned to the `authorizedRegenerationContract` status field, which is essential for checking certain subsequent conditions. This field can be modified by the Owner at any time, as can the `authorizedOTCoinContract` field, which is also required to perform certain checks and is assigned following the deployment of the `OilTrackerToken` contract. The invocation of the `CertificationNFT` contract comes from the last contract that stores tracking data, called `ThirdRegenerationCidStorage`. Unlike the other on-chain components of the oracles, the function for storing the data involves an extra step. In fact, this function, which in this case is named `addRegenerationCid`,

after changing its state with the new information, retrieves from the previous contracts all the IDs of the stored tracking documents. Providing these addresses and the address of the Regeneration Company invokes the `mintNFT` function of the `CertificationNFT` contract. At this point, the newly invoked contract performs a check to verify that the call came from the last tracking contract, and this is done through a modifier called `onlyRegenerationContract`. Verified this check, the function `_safeMint` is executed, through which a new `CertificationToken` or `CERT` is mint and sent to the input address. Next, the contract creates an object of type `CertificationData`, consisting of four elements, the first three corresponding to the IDs of the input documents, and the last one a validity value initially set to true. The newly created object is then added to a mapping called `certificationDataMapping`, where the key is the current value of the NFT counter `_tokenIdCounter`, which as a result of these operations is incremented by one. The execution of the function `mintNFT` ends by issuing an event notifying that the NFT has been mint.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity >=0.8.2 <0.9.0;
3
4  interface CertificationNFTInterface {
5      function mintNFT(address userAddress, string memory id1, string memory id2, string
           memory id3) external;
6  }
7  contract ThirdRegenerationCidStorage {
8      .
9      .
10     .
11     function setNFTMinterAddress(address _newCertificationNFTContractAddress) external
           onlyOwner {
12         certificationNFTContract = CertificationNFTInterface(
               _newCertificationNFTContractAddress);
13         emit CertificationNFTContractAddressSet(_newCertificationNFTContractAddress);
14     }
15     function addRegenerationCid(string memory _secondDocId, string memory _ipfsCid,
           string memory _docId) public onlyEnabledAddress {
16         require(bytes(RegenerationCidMapping[_docId].ipfsCid).length == 0, "A value
               already exists for this key");
17         (string memory firstDocId, ) = previousContract.getWifInfo(_secondDocId);
18         require(bytes(firstDocId).length > 0, "No value exists for this key");
19         RegenerationTrackingObj memory newRegenerationTrackingObj =
               RegenerationTrackingObj(_secondDocId, _ipfsCid);
20         RegenerationCidMapping[_docId] = newRegenerationTrackingObj;
21         DocIdMapping[_secondDocId] = _docId;
22         certificationNFTContract.mintNFT(msg.sender, firstDocId, _secondDocId, _docId);
23         uint256 refundAmount = gasleft() * tx.gasprice;
24         payable(msg.sender).transfer(refundAmount);
25         emit RefundRequested(msg.sender, refundAmount);
26     }
27 }
```

Listing 5.6. Solidity code for the the on-chain component of the third oracle.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.2 <0.9.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract CertificationNFT is ERC721, Ownable {
    .
    .
    .
    constructor(address _authorizedRegenerationContract) ERC721("CertificationToken", "
        CERT") Ownable(msg.sender) {
        authorizedRegenerationContract = _authorizedRegenerationContract;
    }
    function changeRegenerationContract(address newRegenerationContract) public
        onlyOwner {
        authorizedRegenerationContract = newRegenerationContract;
    }
    function setOTCoinContract(address newOTCoinContract) public onlyOwner {
        authorizedOTCoinContract = newOTCoinContract;
    }
    function mintNFT(address userAddress, string memory id1, string memory id2, string
        memory id3) external onlyRegenerationContract {
        uint256 tokenId = _tokenIdCounter;
        _safeMint(userAddress, tokenId);
        certificationDataMapping[tokenId] = CertificationData(id1, id2, id3, true);
        _tokenIdCounter++;
        emit certificationNFTMinted(userAddress, tokenId, id1, id2, id3);
    }
    function getOwnerOfNFT(uint256 tokenId) public view returns (address) {
        return ownerOf(tokenId);
    }
    function getCertificationData(uint256 tokenId) public view returns (string memory,
        string memory, string memory) {
        return (certificationDataMapping[tokenId].id1, certificationDataMapping[tokenId
            ].id2, certificationDataMapping[tokenId].id3);
    }
    function setNFTValidity(uint256 tokenId) external onlyAuthorizedOTCoinCaller {
        require(certificationDataMapping[tokenId].validity == true, "Validity cannot be
            changed anymore");
        certificationDataMapping[tokenId].validity = false;
        emit NFTValiditySet(tokenId, false);
    }
    function getNFTValidity(uint256 tokenId) public view returns (bool) {
        return certificationDataMapping[tokenId].validity;
    }
}
```

Listing 5.7. Solidity code for the certification NFT smart contract.

5.4.2 OilTracker token

The deployment of the `OilTrackerToken` contract is done by passing an address and an integer value as parameters. The former is assigned to the status field `certificationNFTContractAddress`, and the latter to the field `exchangeRate`.

Both of these fields are editable by the contract owner through two different functions, `changeCertificationNFTContractAddress` and `changeExchangeRate`, respectively. The main function of this contract is `mintToken`, and it is invoked by the Regeneration Company address. This function receives as input the parameter `tokenId`, an integer value that corresponds to the ID of a CERT. Using the address stored in `certificationNFTContractAddress`, a contract instance is created. Through this instance, it is verified that the CERT associated with the `tokenId` is owned by the transaction sender, using the function `getOwnerOfNFT`. In case of a positive match, it is checked that the same CERT is still valid and thus exploitable for issuing OTs. This check is done by invoking the `CertificationNFT` contract again through the `getNFTValidity` function, passing the `tokenId` as a parameter. If this second check is also successful, the created contract instance is exploited for the third and final time to invoke the `setNFTValidity` function through which the token is disabled. The execution of the function `mintToken` ends with the effective mint and sends it to the calling user of the OilTracker token, the quantity of which is defined by the value `exchangeRate`. The mint operation is also accompanied by the issuance of an event in order to notify the completion of operations.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5  import "@openzeppelin/contracts/access/Ownable.sol";
6
7  interface CertificationNFTInterface {
8      function getNFTValidity(uint256 tokenId) external view returns (bool);
9      function getOwnerOfNFT(uint256 tokenId) external view returns (address);
10     function setNFTValidity(uint256 tokenId) external;
11 }
12 contract OilTrackerToken is ERC20, Ownable {
13     address private certificationNFTContractAddress;
14     uint256 public exchangeRate;
15     event OilTrackerTokenMinted(address indexed to, uint256 tokenId);
16     event ExchangeRateUpdate(uint256 newExchangeRate);
17     event Burn(address indexed from, uint256 amount);
18     constructor(address _certificationNFTContractAddress, uint256 _exchangeRate) ERC20("
           OilTrackerToken", "OT") Ownable(msg.sender){
19         certificationNFTContractAddress = _certificationNFTContractAddress;
20         exchangeRate = _exchangeRate;
21     }
22     function changeCertificationNFTContractAddress(address
           newcertificationNFTContractAddress) public onlyOwner {
23         certificationNFTContractAddress = newcertificationNFTContractAddress;
24     }
25     function mintToken(uint256 tokenId) public {
26         CertificationNFTInterface certificationNFT = CertificationNFTInterface(
               certificationNFTContractAddress);
27         require(certificationNFT.getOwnerOfNFT(tokenId) == msg.sender, "The user is not
               the owner of this token");
28         require(certificationNFT.getNFTValidity(tokenId), "NFT is not valid anymore");
```

```
29          certificationNFT.setNFTValidity(tokenId);
30          _mint(msg.sender, exchangeRate * (10 ** 18));
31          emit OilTrackerTokenMinted(msg.sender, tokenId);
32      }
33      function changeExchangeRate(uint256 _newExchangeRate) public onlyOwner {
34          exchangeRate = _newExchangeRate;
35          emit ExchangeRateUpdate(_newExchangeRate);
36      }
37      function getExchangeRate() public view returns (uint256) {
38          return exchangeRate;
39      }
40          function burn(uint256 amount) public {
41          _burn(msg.sender, amount);
42          emit Burn(msg.sender, amount);
43      }
44  }
```

Listing 5.8. Solidity code for the OilTracker token smart contract.

Chapter 6

Evaluation

This chapter proposes an evaluation of the costs related to the execution and maintenance of the smart contracts that characterise the OilTracker system. Through the evaluation described below, it is intended to highlight which aspects are more costly from an economic point of view, and which are less so, especially considering the actor involved in a given operation. The chapter is divided into different sections, the first describing in more detail the methodology and tools used to carry out the tests, while the following sections show the results for each smart contract present. In conclusion, a final discussion of the results obtained is proposed.

6.1 Methodology

The economic aspect is the one that is of most interest when implementing a system such as the one proposed, as it is from this aspect that is possible to understand whether the realisation of the project is feasible. Particularly in the case of decentralised EVM-based applications, where it is important to consider transaction costs, which can often prove particularly costly. EVM-based blockchains associate the execution of smart contracts with a fee charged to the address invoking it. This associated cost is called “Gas” and is measured in units and varies in proportion to the computational effort required to execute a function. Gas is associated with smart contract executions to prevent them from remaining in execution indefinitely, since Solidity is a complete Turing language, and would therefore allow such a phenomenon. Infinite execution would lead to essentially unlimited power consumption, as well as blocking every single node belonging to the network. This chapter describes the analysis of this aspect, evaluating its variations on different EVM-based blockchain networks. In particular, the contracts implemented were deployed and executed on the Polygon, Ethereum and Avalanche testnets, called Mumbai, Sepholia, and Fuji, respectively. Since the implementation of OilTracker was realised thanks to

the Remix IDE development environment, it was decided to also use it to carry out this analysis since for each transaction sent it provides the number of Gas units associated with it. Each smart contract in OilTracker was deployed on the three different test networks mentioned above, and tested on each of its functions in an equivalent manner, using real data, thus simulating its real functioning. To provide a consistent and comprehensive evaluation, five costs were selected for the execution of each function, of each contract, for all three networks and an average value was calculated. In addition to the costs in terms of Gas units, there is also the value expressed in terms of ETH, computed using the following formula:

$$\text{Gas Cost} = \text{Gas Price} \times \text{Gas Used}$$

where the "Gas Price" corresponds to the average value for each network expressed in Gwei, while the "Gas Used" corresponds to the units of gas used. The Gas Price for the Polygon [1], Ethereum [2], and Avalanche [3] blockchain networks correspond to 37.7, 40.1, and 25.3 GWei, respectively (Data collected at 11:15 p.m. on 11/12/2023).

6.2 Results

Below are the results obtained in terms of Gas units for each function of each contract involved in OilTracker. The results are shown in tables with four columns, named "Function Name", "Polygon", "Ethereum", and "Avalanche", respectively. The last three columns mentioned also have two sub-columns called "Gas Cost" and "Gas Used". The first describes the cost of execution in terms of units of gas used, while the second is the gas cost, obtained by using the formula described in the previous section and converting the result from GWei to ETH. Before moving on to the description of the costs for each smart contract it is important to note that some functions have no value associated with them, this is either because they return information so do not consume units of Gas, or they are invoked by other contracts so their cost is included in other functions.

Furthermore, all the transactions involved in this analysis can be consulted by using the addresses of each smart contract in the following table, composed of links to the blockchain explorers for each blockchain platform considered.

[1] https://cointool.app/gasPrice/matic
[2] https://cointool.app/gasPrice/eth
[3] https://cointool.app/gasPrice/avax

	Polygon	Ethereum	Avalanche
FirstWifCidStorage	0x90d550541bf02...	0x5b2944e592df...	0xc6fddb477fd45...
SecondWifCidStorage	0xc02039dad0dc3...	0x8bb8cf862f222...	0x0c7023b96e38c...
ThirdRegenerationCidStorage	0x3e728aade6352...	0x812f22874757c...	0x234d2a67d97f1...
CertificationNFT	0x495ec636ca72f...	0x80e0574df65ab...	0x346977e3c20f5...
OilTrackerToken	0xd8eea6ca8b9b8e...	0x5f3dc23d23e70...	0xbe0c53a0735f9...

Table 6.1. Links of smart contract addresses for each analysed blockchain network.

6.2.1 FirstWifCidStorage

FirstWifCidStorage is the smart contract representing the start of the tracking chain. By averaging the values acquired from the three networks, we find that the cost, measured in units of gas used, is 1425912. This operation is handled by the Owner, as this step establishes the Owner as the controller of the smart contract invoking the `constructor()` function. Other operations which only the Owner can perform include `addEnabledAddress()` and `removeEnabledAddress()`, operations which among the networks considered have an average of 61137 and 41792 gas units used. The function of greatest interest in this smart contract, as it performs the data storing operation, is `addWifCid()`, which has an average value of 109047 and is performed by the Transporter. To conclude, FirstWifCidStorage also includes two functions of type `view` that do not incur any cost as they have the sole objective of information, and are `getWifInfo()`, and `getEnabledAddresses()`.

Function Name	Polygon		Ethereum		Avalanche	
	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)
Deployment	1415413	0.0533610701	1415413	0.0567580613	1446910	0.036606823
addWifCid()	108783	0.0041011191	107734	0.0043201334	110622	0.0027987366
addEnabledAddress()	61103	0.0023035831	61104	0.0024502704	61202	0.0015484106
removeEnabledaddress()	37912	0.0014292824	43883	0.0017597083	43580	0.001102574
getWifInfo()	-	-	-	-	-	-
getEnabledAddresses()	-	-	-	-	-	-

Table 6.2. Cost results of FirstWifCidStorage smart contract.

6.2.2 SecondWifCidStorage

SecondWifCidStorage implements the operations necessary for the intermediate phase of the UCO document tracking chain. Given the greater complexity of the operations, as well as a greater number of implemented functions, the cost in terms of gas units used for the deployment phase performed by the Owner is 1895534. The function `changePreviousAddress()`, also executable exclusively by the Owner has an average cost of 30303 units of gas used. Again, the functions `addEnabledAddress()` and `removeEnabledAddress()` have very similar average values to those obtained for the FirstWifCidStorage. On the other hand, the function `addWifCid()`, has a higher cost as it includes additional operations such as saving the referential data related to the initial tracking phase, with an average value of 177321. The functions defined with the type `view` are in these cases `getNextDocId()`, `getWifInfo()` and `getEnabledAddress()`.

Function Name	Polygon		Ethereum		Avalanche	
	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)
Deployment	1882425	0.0709674225	1882425	0.0754852425	1921750	0.048620275
changePreviousAddress()	30300	0.00114231	30300	0.00121503	30308	0.0007667924
addWifCid()	176115	0.0066395355	178351	0.0071518751	177497	0.0044906741
addEnabledAddress()	61222	0.0023080694	61222	0.0024550022	61232	0.0015491696
removeEnabledAddress()	41178	0.0015524106	39862	0.0015984662	41827	0.0010582231
getNextDocId()	-	-	-	-	-	-
getWifInfo()	-	-	-	-	-	-
getEnabledAddresses()	-	-	-	-	-	-

Table 6.3. Cost results of SecondWifCidStorage smart contract.

6.2.3 ThirdRegenerationCidStorage

ThirdRegenerationCidStorage is the smart contract that is invoked at the conclusion moment of the tracking sequence, as well as the point of connection with the NFT's reward system. Its deployment uses an average of 2027900 gas units. The `changePreviousAddress` function, invocable only by the Owner in this case has an average cost of 27525 gas units. The functions `addEnabledAddress()` and `removeEnabledAddress()` have an average cost of 61212 and 38945. The function that implements the storage of the tracking information is called `addRegenerationCid()`

and has an average cost in terms of gas used of 328142. The cost is much higher than the equivalent functions described in the two previous contracts, this is because the invocation of this function involves not only the storage of information but also the invocation of the CertificactionNFT contract through the function `mintNFT()`. In particular, this function is intended for use by the Regeneration Company. The `setNFTMinterAddress()` function, also invocable only by the Owner, has an average cost of 37167. Lastly, the functions `getEnabledAddresses()`, `getRegenerationInfo()`, and `getNextDocId()` complete the set of operations included in this smart contract and have no associated cost since they are of type `view`.

Function Name	Polygon		Ethereum		Avalanche	
	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)
Deployment	2013984	0.0759271968	2013984	0.0807607584	2055732	0.0520100196
changePreviousAddress()	27522	0.0010375794	27522	0.0011036322	27530	0.000696509
addRegenerationCid()	329717	0.0124303309	326884	0.0131080484	327823	0.0082939219
addEnabledAddress()	61217	0.0023078809	61210	0.002454521	61209	0.0015485877
removeEnabledAddress()	39510	0.001489527	38192	0.0015314992	39133	0.0009900649
setNFTMinterAddress()	37494	0.0014135238	38888	0.0015594088	35119	0.0008885107
getRegenerationInfo()	-	-	-	-	-	-
getNextDocId()	-	-	-	-	-	-
getEnabledAddresses()	-	-	-	-	-	-

Table 6.4. Cost results of ThirdRegenerationCidStorage smart contract.

6.2.4 CertificationNFT

CertificationNFT is the first contract representing the token-based reward system. In particular, it implements the CertificationToken(CERT) of type ERC721. The cost for its deployment, which is the Owner's responsibility, is 2659587 gas units. There are also two contract setting functions that can only be invoked by the Owner, `changeRegenerationContract()` and `setOTCoinContract()` with an average gas unit usage of 27945 and 36873 respectively. The function `mintNFT()`, has no associated cost as it is invoked by the ThirdRegenerationCidStorage contract. The same applies to the function `setNFTValidity()`, which is however invoked by the OilTrackerToken contract described below. The functions of type `view` included in

this smart contract are `getCertificationData()` and `getOwnerOfNFT()`.

Function Name	Polygon		Ethereum		Avalanche	
	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)
Deployment	2677095	0.1009264815	2625414	0.1052791014	2676257	0.0677093021
changeRegenerationContract()	27941	0.0010533757	27941	0.0011204341	27951	0.0007071603
setOTCoinContract()	36403	0.0013723931	37803	0.0015159003	36413	0.0009212489
mintNFT()	-	-	-	-	-	-
getOwnerOfNFT()	-	-	-	-	-	-
getCertificationData()	-	-	-	-	-	-
setNFTValidity()	-	-	-	-	-	-

Table 6.5. Cost results of CertificationNFT smart contract.

6.2.5 OilTrackerToken

OilTrackerToken is the smart contract implementing the ERC20 token of the same name, also referred to as OT. Its purpose is to serve as an exchange asset for the acquisition of multiple future services. Compared to the ERC721 type CertificationNFT, its deployment has a lower cost, whose average value among the three networks considered is 1533445 units of gas used. The operations to be performed by the Owner in this smart contract, in addition to the deployment phase, are `changeCertificationNFTContractAddress()`, and `changeExchangeRate()`, which have an average cost of 27920 and 29229 units of gas used, respectively. The contract then includes two functions that can be invoked by the Regeneration Company. The first is `mintToken()` with an average gas used cost of 72167. The second is the `burn()` function with a cost of 35917. Finally, the only function of type `view` within the OilTrackerToken contract is `getExchangeRate()` invocable by any user the value assigned to a single NFT at the time of the OT recommendation.

Function Name	Polygon		Ethereum		Avalanche	
	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)	Gas Used	Gas Cost (ETH)
Deployment	1524344	0.0574677688	1524344	0.0611261944	1551645	0.0392566185
changeCertificationNFTContractAddress()	27918	0.0010525086	27912	0.0011192712	27928	0.0007065784
mintToken()	70591	0.0026612807	70591	0.0028306991	75317	0.0019055201
changeExchangeRate()	28758	0.0010841766	28758	0.0011531958	30169	0.0007632757
burn()	35914	0.0013539578	35914	0.0014401514	35923	0.0009088519
getExchangeRate()	-	-	-	-	-	-

Table 6.6. Cost results of OilTrackerToken smart contract.

6.3 Discussion

After showing the results obtained for each smart contract used, it is possible to discuss the results obtained, also illustrating the conclusions deduced from this analysis. Looking at the results, it is easy to see that they are often similar for some functions of different contracts. This is obviously due to the high level of similarity of the functions themselves. At a general level, it can be seen that the cost in terms of units of gas used to deploy contracts is very similar for the Polygon and Ethereum platforms, while it is slightly higher when executed on the Avalanche network. However, this behaviour may also be due to the variations in gas usage that EVM-based platforms are usually subject to. Despite this aspect, the units of gas used for functions within each contract are consistent across all platforms analysed. Considering the gas prices mentioned, it is easy to see that Avalanche has lower costs in terms of ETH. Although Avalanche needs higher implementation costs in terms of gas units consumed, it proves to have lower costs in ETH than the other two platforms. This can be attributed to the relatively low value of its gas price. In order to better contextualise the purely economic aspect of the creation, setup and use of these smart contracts, the value of a single ETH in EUR is used, which is EUR 2064.29 (value collected at 11:15 p.m. on 11/12/2023 as in the case of gas prices). The deployment costs for all smart contracts are EUR 740.357484023313 for Polygon, EUR 783.21094362582 for Ethereum, and EUR 504.105889725878 for Avalanche. Deployment costs are higher than those for the execution of functions because they encapsulate all the complexity of the smart contract itself. Furthermore, it is important to define certain security standards while trying to keep the code as simple as possible from the point of view of optimisation of memory and the burden of operations. The development on testnets is necessary to avoid additional

costs during the creation phase of the actual system, given the costs involved in using blockchain platforms. The costs of setting up the contracts were calculated considering the execution of the following functions:

- FirstWifCidStorage: Two executions of `addEnabledAddress()` to add the address of at least one Transporter and the address of the contract SecondWifCidStorage with a cost of EUR 9.510527114998 for Polygon, EUR 10.116137368032 for Ethereum, and EUR 6.392737034948 for Avalanche.
- SecondWifCidStorage: Two executions of `addEnabledAddress()` to add the address of at least one Transporter and that of the contract `ThirdWifCidStorage` with a cost of EUR 9.529049163452 for Polygon, EUR 10.135672982876 for Ethereum, and EUR 6.395870627168 for Avalanche.
- ThirdRegenerationCidStorage: One execution of `addEnabledAddress()` to add the address of at least one Regeneration Company, and one execution of `setNFTMinterAddress()` to add the address of the contract CertificationNFT with a cost of EUR 7.682058508163 for Polygon, EUR 8.285915146842 for Ethereum, and EUR 5.030877856136 for Avalanche.
- CertificationNFT: An execution of `setOTCoinContract()` to set the address of the OilTrackerToken contract with a cost of EUR 2.833017352399 for Polygon, EUR 3.129257830287 for Ethereum, and EUR 1.901724891781 for Avalanche.
- OilTrackerToken: No set-up functions following deployment.

Adding up the deployment costs with those just described, we obtain the necessary cost for the owner to set up the infrastructure, with EUR 769.912136162325 for Polygon, EUR 814.877926953857 for Ethereum, and EUR 523.827100135911 for Avalanche.

Finally, with regard to the use of smart contracts by the user Transporter and the user Regeneration Company for the processes of tracking and issuing rewards, the situation is as follows:

- FirstWifCidStorage: Execution of the function `AddWifCid()` by the Transporter with cost per invocation charged to the Owner of EUR 8.465899146939 for Polygon, EUR 8.918008176286 for Ethereum, and EUR 5.777403976014 for Avalanche.
- SecondWifCidStorage: Execution of the function `AddWifCid()` by the Transporter with cost per invocation charged to the Owner equal to EUR 13.705926737295 for Polygon, EUR 14.763544250179 for Ethereum, and EUR 9.270053637889 for Avalanche.

- ThirdRegenerationCidStorage: Execution of the function `AddRegenerationCid()` by the Regeneration Company with cost per invocation charged to the Owner equal to EUR 25.659807773561 for Polygon, EUR 27.058813231636 for Ethereum, and EUR 17.121060038951 for Avalanche.
- CertificationNFT: The cost of the `mintNFT()` function is considered in those of `AddRegenerationCid()` of the ThirdRegenerationCidStorage contract.
- OilTrackerToken: Execution of the `mintToken()` function by the Regeneration Company at a cost to it of EUR 5.493655136203 for Polygon, EUR 5.843383845139 for Ethereum, and EUR 3.933546087229 for Avalanche.

Adding up the data just described, we have an idea of the cost of creating a tracking sequence and issuing the CertificationNFT. The expense associated with the issuance of the OilTrackerToken is not taken into account, as it is covered by the Regeneration Company, which may or may not decide whether or not to request it. The costs are EUR 27.665481020437 for Polygon, EUR 29.524936271604 for Ethereum, and EUR 18.981003701132 for Avalanche. Observing the data obtained and the considerations just made, it is concluded that the Avalanche platform is the cheaper one from the point of view of both the creation of the smart contract infrastructure and the recurring use for tracking the UCO, and this applies to both the Owner and the Regeneration Company, the only two actors that bear costs within the OilTracker system.

Chapter 7

Conclusion and Future Remarks

The work carried out for the realisation of this project was aimed at creating a system to track the movement of used vegetable oil (UCO), from the moment of its collection to the moment of its regeneration and subsequent transformation into regenerated used vegetable oil (RUCO). A second objective was also to create certifications from the regeneration process that could be used as a trading asset for future services. These objectives were achieved by exploiting all the potential that the technologies mentioned in this paper could offer.

The tool at the heart of the project is the blockchain, which played the most important role. In fact, it enabled the realisation of the two core goals of the thesis work, that of offering a platform to store information with which to track the various UCO and RUCO matches, as well as enabling the implementation of a system for issuing NFTs as a certification symbol and as a guarantee for the issuing of fungible tokens. One aspect that it is important to emphasise, however, is the costs involved in using this technology. As can also be deduced from the evaluation described in Chapter 6, blockchain, and EVM-based platforms in particular, are very much linked to the economic factor from the point of view of implementation and maintenance, which is why storing documents in their entirety within the smart contract state would have entailed an exponential increase in costs, also considering the quantity of data that would have to be recorded if one considers the use of this system on a national scale. Precisely to avoid situations such as the one described, it was decided to adopt an alternative solution for storing documents, namely Interplanetary File System (IPFS). This is a technology with enormous potential, especially if exploited as a tool to support systems such as the one realised, allowing large amounts of data to be stored, without necessarily having to do so on platforms such as the blockchain, where it is important to save on every single bit if you wants to create a system that is functional and sustainable over time. IPFS made it possible to create a file storage detached from the smart contracts running on the blockchain, but maintaining a

reference through the CID codes of the different documents stored on the blockchain. The use of EVM-based blockchain platforms made it possible to exploit the ERC20 and ERC721 standards for the representation of fungible and non-fungible tokens. These two standards ensured a high level of customisation of tokens by ensuring that they were linked to each other and by regulating their delivery based on certain requirements, such as the completion of the tracking sequence or the possession of certification NFTs.

The design phase made it possible to create a system that was as robust as possible both from the point of view of security, with the encryption of information, and from the point of view of architecture design and the role-based access system. In this phase, a careful study was made of the characteristics needed for the OilTracker system, which made it possible to understand how to differentiate the use of the system to the different actors involved in it, such as the Owner, the Transporter, and the Regeneration Company. Each of them performs different operations, and only through the design ideas that emerged during this phase was it possible to arrive at the implementation of the final system.

From the implementation point of view, the technologies used were numerous and all played an important role in their way. The backbone on which OilTracker is based is represented by the smart contracts, encoded through the Solidity language, which was designed and subsequently tested on Remix IDE, a clear and complete development environment without which it would not have been possible to perform all the necessary tests during both the implementation and evaluation phases. In order to provide a system that could be used by any type of user, a decentralised application (DApp) was created using state-of-the-art frameworks such as ReactJs, which ensured a dynamic user experience and user interface. Although marginal, the role of the database realised with MongoDB was also relevant within the DApp, as it made it possible to create a place on which to store the users registered to the system and to which keys were assigned for encrypting and possibly, in the case of the Owner, decrypting the stored documents. The cost analysis carried out in the previous chapter finally highlighted the economic aspect involved in the implementation and subsequent use of the OilTracker system. Costs were analysed on three different EVM-based blockchain platforms, Polygon, Ethereum, and Avalanche, exploiting their respective test networks. In order to conduct an exhaustive study, more than 400 transactions were executed, divided equally between the different functions implemented within the smart contracts. Considering the costs obtained, it emerged that the Avalanche blockchain guaranteed lower costs than the other blockchains analysed, but despite this, it was also easy to see the need to further optimise the code for the smart contracts in order to make the implementation costs lower.

The proposed study focused on three fundamental aspects, functional, techno-

logical and economic. Considering these three aspects, the work carried out proved to have the right potential to act as a model of technological innovation and sustainability by allowing the digitalisation of a huge amount of documents, as well as encouraging a more correct behaviour concerning the renewal of waste and UCO in particular.

7.1 Future Remarks

The system has some limitations that future versions could try to solve, first and foremost the cost of management and maintenance. In this regard, it would be interesting to propose the same version of the system, but considering different blockchain platforms, such as Algorand[1], which uses a smart contract development language called TEAL[2] that is completely different from Solidity, both syntactically and logically in terms of transaction generation and token management.

As far as developments that OilTracker could go against, there is also the consideration of technologies or systems other than IPFS for storing data, so as to offer increasingly efficient and cost-effective solutions.

Finally, the most interesting perspective concerns the usefulness of the OilTrackerToken(OT). This could in fact be exchanged in the future for so-called "carbon tokens", which can be issued by third-party projects in order to promote solutions that reduce carbon dioxide (CO_2) emissions, as in the case of the Polygon blockchain. Such tokens could then take on value in the marketplace by serving as an element of certification of a successful decarbonisation process.

Finally, an interesting development could be the integration of Internet-of-Things (IoT) systems to monitor the movements of the UCO in order to provide a more complete view of the actual CO_2 emissions in support of the "carbon token" management system itself, as well as act as a security element against fraud and tampering that production chains such as the one for the collection of used vegetable oils may encounter. These two developments could increase participation on the part of the actors involved and increase the level of transparency towards regeneration processes, increasing the attention paid to environmental impacts and sustainability.

[1] https://algorandtechnologies.com/
[2] https://developer.algorand.org/docs/get-details/dapps/avm/teal/

Bibliography

[1] Katherine Tebbatt Adams, Mohamed Osmani, Tony Thorpe, and Jane Thornback. Circular economy in construction: current awareness, challenges and enablers. In *Proceedings of the Institution of Civil Engineers-Waste and Resource Management*, volume 170, pages 15–24. Thomas Telford Ltd, 2017.

[2] Hamda Al-Breiki, Muhammad Habib Ur Rehman, Khaled Salah, and Davor Svetinovic. Trustworthy blockchain oracles: review, comparison, and open research challenges. *IEEE access*, 8:85675–85685, 2020.

[3] Ahmed Alkhateeb, Cagatay Catal, Gorkem Kar, and Alok Mishra. Hybrid blockchain platforms for the internet of things (iot): A systematic literature review. *Sensors*, 22(4):1304, 2022.

[4] Eiman Alnuaimi, Mariam Alsafi, Maitha Alshehhi, Mazin Debe, Khaled Salah, Ibrar Yaqoob, M Jamal Zemerly, and Raja Jayaraman. Blockchain-based system for tracking and rewarding recyclable plastic waste. *Peer-to-Peer Networking and Applications*, 16(1):328–346, 2023.

[5] Avalanche Platform. Polygon whitepaper, 2019.

[6] Adam Back et al. Hashcash-a denial of service counter-measure. 2002.

[7] Imran Bashir. *Mastering blockchain*. Packt Publishing Ltd, 2017.

[8] Davide Basile, Valerio Goretti, Claudio Di Ciccio, and Sabrina Kirrane. Enhancing blockchain-based processes with decentralized oracles. In *International Conference on Business Process Management*, pages 102–118. Springer, 2021.

[9] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys & Tutorials*, 21(4):3796–3838, 2019.

[10] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[11] Abdeljalil Beniiche. A study of blockchain oracles. *arXiv preprint arXiv:2004.07140*, 2020.

[12] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2013.

[13] Giulio Caldarelli. Overview of blockchain oracle research. *Future Internet*, 14 (6):175, 2022.

[14] Giulio Caldarelli, Cecilia Rossignoli, and Alessandro Zardini. Overcoming the blockchain oracle problem in the traceability of non-fungible products. *Sustainability*, 12(6):2391, 2020.

[15] John P Conley et al. Blockchain and the economics of crypto-tokens and initial coin offerings. *Vanderbilt University Department of economics working papers*, 2017.

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

[17] Thomas K Dasaklis, Fran Casino, and Constantinos Patsakis. A traceability and auditing framework for electronic equipment reverse logistics based on blockchain: The case of mobile phones. In *2020 11th International Conference on Information, Intelligence, Systems and Applications (IISA*, pages 1–7. IEEE, 2020.

[18] Alex De Vries. Bitcoin's growing energy problem. *Joule*, 2(5):801–805, 2018.

[19] Konstantinos Demestichas and Emmanouil Daskalakis. Information and communication technology solutions for the circular economy. *Sustainability*, 12 (18):7272, 2020.

[20] Advait Deshpande, Katherine Stewart, Louise Lepetit, and Salil Gunashekar. Distributed ledger technologies/blockchain: Challenges, opportunities and the prospects for standards. *Overview report The British Standards Institution (BSI)*, 40:40, 2017.

[21] S Di Fraia, N Massarotti, MV Prati, and L Vanoli. A new example of circular economy: Waste vegetable oil for cogeneration in wastewater treatment plants. *Energy Conversion and Management*, 211:112763, 2020.

[22] Marco Di Francesco, Lodovica Marchesi, and Raffaele Porcu. Kryptosafe: managing and trading data sets using blockchain and ipfs. In *2023 IEEE/ACM 6th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 5–8. IEEE, 2023.

[23] Omar Dib, Kei-Leo Brousmiche, Antoine Durand, Eric Thea, and Elyes Ben Hamida. Consortium blockchains: Overview, applications and challenges. *Int. J. Adv. Telecommun*, 11(1):51–64, 2018.

[24] Trinh Viet Doan, Yiannis Psaras, Jörg Ott, and Vaibhav Bajpai. Towards decentralised cloud storage with ipfs: Opportunities, challenges, and future directions. *arXiv preprint arXiv:2202.06315*, 2022.

[25] William Entriken, Dieter Shirley, Jacob Evans, and Nastassia Sachs. Erc-721: Non-fungible token standard. Ethereum Improvement Proposals, January 2018. URL https://eips.ethereum.org/EIPS/eip-721.

[26] Ellen MacArthur Foundation. Circularity in the built environment: Case studies, 2016.

[27] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310. Springer, 2015.

[28] Sylvie Geisendorf and Felicitas Pietrulla. The circular economy and circular economic concepts—a literature analysis and redefinition. *Thunderbird International Business Review*, 60(5):771–782, 2018.

[29] Malahat Ghoreishi and Ari Happonen. New promises ai brings into circular economy accelerated product design: a review on supporting literature. In *E3S web of conferences*, volume 158, page 06002. EDP Sciences, 2020.

[30] Stuart Haber and W Scott Stornetta. *How to time-stamp a digital document*. Springer, 1991.

[31] Carla Hamida, Amanda Landi, and Ziyi Liu. The equity and inclusion in higher education: A proposed model for open data. In *RAIS Conference Proceedings-The 13th International RAIS Conference on Social Sciences and Humanities*, 2019.

[32] J Hao, Yan Sun, and Hong Luo. A safe and efficient storage scheme based on blockchain and ipfs for agricultural products tracking. *J. Comput*, 29(6): 158–167, 2018.

[33] Abid Hassan, MD Iftekhar Ali, Rifat Ahammed, Mohammad Monirujjaman Khan, Nawal Alsufyani, and Abdulmajeed Alsufyani. Secured insurance

framework using blockchain and smart contract. *Scientific Programming*, 2021: 1–11, 2021.

[34] Zheng Hong, Liu Zerun, Huang Jianhua, and Qian Shihui. Verification of transaction ordering dependence vulnerability of smart contract based on cpn. *Journal of System Simulation*, 34(7):1629, 2022.

[35] Md Rafiqul Islam, Muhammad Mahbubur Rahman, Md Mahmud, Mohammed Ataur Rahman, Muslim Har Sani Mohamad, et al. A review on blockchain security issues and challenges. In *2021 IEEE 12th Control and System Graduate Research Colloquium (ICSGRC)*, pages 227–232. IEEE, 2021.

[36] Kose John, Maureen O'Hara, and Fahad Saleh. Bitcoin and beyond. *Annual Review of Financial Economics*, 14:95–115, 2022.

[37] M Frans Kaashoek and David R Karger. Koorde: A simple degree-optimal distributed hash table. In *Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers 2*, pages 98–107. Springer, 2003.

[38] Kevin Sekniqi, Daniel Laine, Stephen Buttolph, and Emin G¨un Sirer. Avalanche platform, 2020.

[39] Swikriti Khadke, Pragya Gupta, Shanmukh Rachakunta, Chandreswar Mahata, Suma Dawn, Mohit Sharma, Deepak Verma, Aniruddha Pradhan, Ambati Mounika Sai Krishna, Seeram Ramakrishna, et al. Efficient plastic recycling and remolding circular economy using the technology of trust–blockchain. *Sustainability*, 13(16):9142, 2021.

[40] Asma Khatoon, Piyush Verma, Jo Southernwood, Beth Massey, and Peter Corcoran. Blockchain in energy efficiency: Potential applications and benefits. *Energies*, 12(17):3317, 2019.

[41] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I*, pages 357–388. Springer, 2017.

[42] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19(1), 2012.

[43] Merit Kolvart, Margus Poola, and Addi Rull. Smart contracts. *The Future of Law and etechnologies*, pages 133–147, 2016.

[44] Małgorzata Koszewska. Circular economy—challenges for the textile and clothing industry. *Autex Research Journal*, 18(4):337–347, 2018.

[45] Mahtab Kouhizadeh, Qingyun Zhu, and Joseph Sarkis. Blockchain and the circular economy: potential tensions and critical reflections from practice. *Production Planning & Control*, 31(11-12):950–966, 2020.

[46] Peter Lacy, Jacob Rutqvist, and Beatrice Lamonica. *Circular economy: Dallo spreco al valore*. EGEA spa, 2016.

[47] Thomas Lambert, Daniel Liebau, and Peter Roosenboom. Security token offerings. *Small Business Economics*, pages 1–27, 2021.

[48] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. 2019.

[49] Bahareh Lashkari and Petr Musilek. A comprehensive review of blockchain consensus mechanisms. *IEEE Access*, 9:43620–43652, 2021.

[50] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, and Ingo Weber. Caterpillar: A blockchain-based business process management system. *BPM (Demos)*, 172, 2017.

[51] Ellen MacArthur et al. Towards the circular economy. *Journal of Industrial Ecology*, 2(1):23–44, 2013.

[52] Edoardo Marangone, Claudio Di Ciccio, Daniele Friolo, Eugenio Nerio Nemmi, Daniele Venturi, and Ingo Weber. Martsia: Enabling data confidentiality for blockchain-based process execution. *arXiv preprint arXiv:2303.17977*, 2023.

[53] Pedro José Martínez-Jurado and José Moyano-Fuentes. Lean management, supply chain management and sustainability: a literature review. *Journal of Cleaner Production*, 85:134–150, 2014.

[54] MC Math, Sudheer Prem Kumar, and Soma V Chetty. Technologies for biodiesel production from used cooking oil—a review. *Energy for sustainable Development*, 14(4):339–345, 2010.

[55] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO'87: Proceedings 7*, pages 369–378. Springer, 1988.

[56] Merriam-Webster. Utility token, 2023. URL https://www.merriam-webster.com/dictionary/utility%20token. Accessed November 9, 2023.

[57] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. An overview of smart contract and use cases in blockchain technology. In *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*, pages 1–4. IEEE, 2018.

[58] Bálint Molnár, Galena Pisoni, Meriem Kherbouche, and Yossra Zghal. Blockchain-based business process management (bpm) for finance: The case of credit and claim requests. *Smart Cities*, 6(3):1254–1278, 2023.

[59] Douglas C Montgomery and William H Woodall. An overview of six sigma. *International Statistical Review/Revue Internationale de Statistique*, pages 329–346, 2008.

[60] Michael zur Muehlen and Danny Ting-Yi Ho. Risk management in the bpm lifecycle. In *International Conference on Business Process Management*, pages 454–466. Springer, 2005.

[61] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[62] Harish Natarajan, Solvej Krause, and Helen Gradstein. Distributed ledger technology and blockchain. 2017.

[63] Till Neudecker and Hannes Hartenstein. Network layer aspects of permissionless blockchains. *IEEE Communications Surveys & Tutorials*, 21(1):838–857, 2018.

[64] Till Neudecker and Hannes Hartenstein. Short paper: An empirical analysis of blockchain forks in bitcoin. In *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 84–92. Springer, 2019.

[65] Cong T Nguyen, Dinh Thai Hoang, Diep N Nguyen, Dusit Niyato, Huynh Tuong Nguyen, and Eryk Dutkiewicz. Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities. *IEEE Access*, 7:85727–85745, 2019.

[66] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59:183–187, 2017.

[67] Luis Oliveira, Liudmila Zavolokina, Ingrid Bauer, and Gerhard Schwabe. To token or not to token: Tools for understanding blockchain tokens. 2018.

[68] Ilhaam Omar, Haya Hassan, Raja Jayaraman, Khaled Salah, and Mohamed Omar. Using blockchain technology to achieve sustainability in the hospitality industry by reducing food waste. *Available at SSRN 4530352*, 2023.

[69] Alfonso Panarello, Nachiket Tapas, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Blockchain and iot integration: A systematic survey. *Sensors*, 18(8):2575, 2018.

[70] Amirmohammad Pasdar, Zhongli Dong, and Young Choon Lee. Blockchain oracle design patterns. *arXiv preprint arXiv:2106.09349*, 2021.

[71] P Paul, PS Aithal, and Ricardo Saavedra. Blockchain technology and its types—a short review. *International Journal of Applied Science and Engineering (IJASE)*, 9(2):189–200, 2021.

[72] Jack Payne, Paul McKeown, and Matthew D Jones. A circular economy approach to plastic waste. *Polymer Degradation and Stability*, 165:170–181, 2019.

[73] David W Pearce and R Kerry Turner. *Economics of natural resources and the environment*. Johns Hopkins University Press, 1989.

[74] Maciel M Queiroz, Renato Telles, and Silvia H Bonilla. Blockchain and supply chain management integration: a systematic review of the literature. *Supply chain management: An international journal*, 25(2):241–254, 2020.

[75] Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien, Eric Binet, and Ronan Sandford. ERC-1155: Multi Token Standard, June 2018. URL https://eips.ethereum.org/EIPS/eip-1155. Ethereum Improvement Proposals, no. 1155.

[76] S Rahayu, KA Pambudi, A Afifah, SR Fitriani, S Tasyari, M Zaki, and R Djamahar. Environmentally safe technology with the conversion of used cooking oil into soap. In *Journal of Physics: Conference Series*, volume 1869, page 012044. IOP Publishing, 2021.

[77] Arun Sekar Rajasekaran, Maria Azees, and Fadi Al-Turjman. A comprehensive survey on blockchain technology. *Sustainable Energy Technologies and Assessments*, 52:102039, 2022.

[78] Tânia Rodrigues Pereira Ramos, Maria Isabel Gomes, and Ana Paula Barbosa-Póvoa. Planning waste cooking oil collection systems. *Waste Management*, 33 (8):1691–1703, 2013.

[79] Michel Raynal. Communication and agreement abstractions for fault-tolerant asynchronous distributed systems. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–273, 2010.

[80] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pages 6–24. Springer, 2013.

[81] David Rose, Edouard Machery, Stephen Stich, Mario Alai, Adriano Angelucci, Renatas Berniūnas, Emma E Buchtel, Amita Chatterjee, Hyundeuk Cheon, In-Rae Cho, et al. Nothing at stake in knowledge. *Noûs*, 53(1):224–247, 2019.

[82] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen. Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):1977–2008, 2020.

[83] Fahad Saleh. Blockchain without waste: Proof-of-stake. *The Review of financial studies*, 34(3):1156–1190, 2021.

[84] Noama Fatima Samreen and Manar H Alalfi. Reentrancy vulnerability identification in ethereum smart contracts. In *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 22–29. IEEE, 2020.

[85] Sarwar Sayeed and Hector Marco-Gisbert. Assessing blockchain consensus and security mechanisms against the 51% attack. *Applied sciences*, 9(9):1788, 2019.

[86] Patrick Schueffel. Defi: Decentralized finance-an introduction and overview. *Journal of Innovation Management*, 9(3):I–XI, 2021.

[87] Mahesh Shirole, Maneesh Darisi, and Sunil Bhirud. Cryptocurrency token: An overview. In *IC-BCT 2019: Proceedings of the International Conference on Blockchain Technology*, pages 133–140. Springer, 2020.

[88] Mahendra Kumar Shrivas and Thomas Yeboah. The disruptive blockchain: types, platforms and applications. *Texila International Journal of Academic Research*, 2019:17–39, 2019.

[89] Christian Sillaber and Bernhard Waltl. Life cycle of smart contracts in blockchain ecosystems. *Datenschutz und Datensicherheit-DuD*, 41(8):497–500, 2017.

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

[91] Siamak Solat, P. Calvez, and Farid Naït-Abdesselam. Permissioned vs. permissionless blockchain: How and why there is only one right choice. *Journal of Software*, 16:95 – 106, 12 2020. doi: 10.17706/jsw.16.3.95-106.

[92] Asterios Stroumpoulis, Evangelia Kopanaki, and Maria Oikonomou. The impact of blockchain technology on food waste management in the hospitality industry. *ENTRENOVA-Enterprise Research Innovation*, 7(1):419–428, 2021.

[93] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2):28, 1996.

[94] Nick Szabo et al. Smart contracts, 1994.

[95] Frederick Winslow Taylor. *The principles of scientific management*. NuVision Publications, LLC, 1911.

[96] Feng Tian. An agri-food supply chain traceability system for china based on rfid & blockchain technology. In *2016 13th international conference on service systems and service management (ICSSSM)*, pages 1–6. IEEE, 2016.

[97] Neil Tiwari. The commodification of crypotocurrency. *Mich. L. Rev.*, 117:611, 2018.

[98] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. Design and evaluation of ipfs: a storage layer for the decentralized web. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 739–752, 2022.

[99] Alaa M Ubaid and Fikri T Dweiri. Business process management (bpm): terminologies and methodologies unified. *International Journal of System Assurance Engineering and Management*, 11:1046–1064, 2020.

[100] Wil MP Van der Aalst. Business process management: a comprehensive survey. *International Scholarly Research Notices*, 2013, 2013.

[101] Pavel Vasin. Blackcoin's proof-of-stake protocol v2. *URL: https://blackcoin. co/blackcoin-pos-protocol-v2-whitepaper. pdf*, 71, 2014.

[102] Michaël Verdonck and Geert Poels. Decentralized data access with ipfs and smart contract permission management for electronic health records. In *Business Process Management Workshops: BPM 2020 International Workshops, Seville, Spain, September 13–18, 2020, Revised Selected Papers 18*, pages 5–16. Springer, 2020.

[103] Fabian Vogelsteller and Vitalik Buterin. Erc-20: Token standard. Ethereum Improvement Proposals, November 2015. URL https://eips.ethereum.org/EIPS/eip-20.

[104] Jan Vom Brocke and Michael Rosemann. *Handbook on business process management 1: Introduction, methods, and information systems*. Springer, 2014.

[105] Konstantinos Voulgaridis, Thomas Lagkas, Constantinos Marios Angelopoulos, and Sotiris E Nikoletseas. Iot and digital circular economy: Principles, applications, and challenges. *Computer Networks*, 219:109456, 2022.

[106] Marko Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM workshop on blockchain, cryptocurrencies and contracts*, pages 3–7, 2017.

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

[108] Shuai Wang, Yong Yuan, Xiao Wang, Juanjuan Li, Rui Qin, and Fei-Yue Wang. An overview of smart contract: architecture, applications, and future trends. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 108–113. IEEE, 2018.

[109] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *Ieee Access*, 7: 22328–22370, 2019.

[110] Mathias Weske. *Business process management architectures*. Springer, 2007.

[111] Stephen A White. Introduction to bpmn. *Ibm Cooperation*, 2(0):0, 2004.

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

[113] Peng Zhang, Douglas C Schmidt, Jules White, and Abhishek Dubey. Consensus mechanisms and information security technologies. *Advances in Computers*, 115:181–209, 2019.

[114] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.

[115] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 47(10):2084–2106, 2019.