LAB University of Applied Sciences

Blockchain DApps.

LAB University of Applied Sciences

Bachelor of Information Technology

2022

Juan Francisco Vañó Francés

Abstract

Author(s) Vañó Francés, Juan Francisco	Publication type Thesis, UAS	Completion year 2022
	Number of pages 67	

Title of the thesis

Blockchain DApps

Degree

Technology

Name, title, and organization of the client

Abstract

This work is about decentralized applications. These applications are deployed on the blockchain, making them interesting and different from traditional centralized applications.

First, it is discussed theoretically what a blockchain is, how this technology is built, and some tools for developing decentralized applications. This theoretical part is to put the reader in context and better understand the practical case. This practical case is a decentralized application programmed by the author of this document to show how important this technology can be and how it can modify how a business is managed now.

The main goal is to show that blockchain technology is beyond supporting cryptocurrencies and NFTs are beyond art speculation.

Keywords

Blockchain, DApp, NFT, Ethereum, Solidity, Truffle, Ganache, NodeJS.

Contents

Glossary.

ABI: Application Binary Interface. Kind of file which is autogenerated JSON when the smart contracts are compiled. ABI files are the standard way to interact with the smart contract, either outside the blockchain or between contracts.

Bytecode: Set of instructions readable by the Ethereum Virtual Machine (EVM).

CID: Content Identifier. It is a label used to point to material in an IPFS.

Cryptocurrency exchange: Platform in which it is possible to buy and sell cryptocurrencies.

DApp: Decentralized Application. It is a distributed software application that runs on a peer-to-peer blockchain instead of a single centralized server.

Ether: Main cryptocurrency of Ethereum Blockchain.

Etherscan: Tool for scanning the transactions taking place on the Ethereum blockchain.

EVM: Ethereum Virtual Machine. Machine used in Ethereum to execute the code programmed in smart contracts. This code is read by the machine in a bytecode way, but before reading it, it is necessary to carry out a compilation process in order to convert the code written in the smart contract into a bytecode.

Ganache: A personal blockchain for rapid Ethereum and Corda distributed application development.

Gas fees: Price to pay for executing a transaction on a blockchain. This fee is in addition to the transaction price.

Holder: Person who manages the account that possesses some digital assets.

Main-net: Blockchain production network.

Mint: Process to add to the blockchain new NFTs or add a cryptocurrency.

NFT: Non-Fungible Token. A digital asset that is stored on a blockchain and represents something unique.

NFT Marketplace: Platform where it is possible to trade with NFTs.

Remix IDE: Online platform designed by Ethereum to program and deploy smart contracts.

Solidity: High-level programming language used to program smart contracts that will be deployed on the Ethereum blockchain or EVM-compatible blockchains.

Sign message: Cryptographic measure to prove ownership via the wallet.

Smart contract: It is a programming script deployed on a blockchain. It contains instructions that are automatically triggered when an event occurs.

Test-net: Blockchain test network.

Truffle: Suite of software programs that helps to deploy and test DApps in EVM-compatible environments.

Twelve words of a wallet: Security phrase with which a digital wallet account is exported or recovered. It is not the same as a private key of an account.

URI: Uniform Resource Identifier. Used to denote specific content in a particular context.

1 Introduction

Nowadays, users increasingly use and adopt technologies like artificial intelligence, virtual reality, the internet of things, etc. Another new technology called blockchain appeared a few years ago and is being adopted by many users and organizations to jump to the following internet era, the decentralization era. From the emergence of the first blockchain to the present day, many other blockchains have seemed to improve the first one. One of the most valuable improvements is the possibility of creating new applications on it.

Creating applications on a blockchain is the main topic to discuss. Still, firstly, to better understand how to develop these applications, it is necessary to go deeper on a blockchain basis and know its main features. In this work, the smart contracts and some blockchains that use them to add some blockchain functionalities are also presented theoretically. After it, decentralized applications and their main components are explained, followed by the new generation of web pages born from the union of technologies discussed before. The last theoretical topic is Non-Fungible tokens (NFTs). This section introduces this new technology, its properties, and future applications.

The concepts discussed before are put together in a practical case that shows how these different technologies work and bring a new and disruptive point of view about the internet.

This work is for those who want to learn more about this technology. Some parts are very technical, so some programming or computer science concepts are necessary to understand completely.

The main goal of this work is to develop a decentralized application (DApp) able to create and manage NFTs in order to provide some benefits to the holders. This DApp is programmed to run on whichever EVM-compatible blockchains. The application demonstrates that whichever business can take advantage of NFTs and generate benefits for both the company itself and the customer.

2 Blockchain and new improvements

2.1 Briefly blockchain review

Fourteen years ago, between 2008 and 2009, the first cryptocurrency appeared in our world in the middle of an economic crisis, and new technology came with it. This technology is called blockchain, and the reason is that in this kind of data structure, the transactions are stored in blocks. After filling a block with transactions, a new one is created, and this new block is connected to its previous one throughout the parent block hash number, thus forming a blockchain. Each block is mined from time to time, and this time is different in each blockchain. The time that needs a block to be mined is known as block time. (Haynes 2022.)

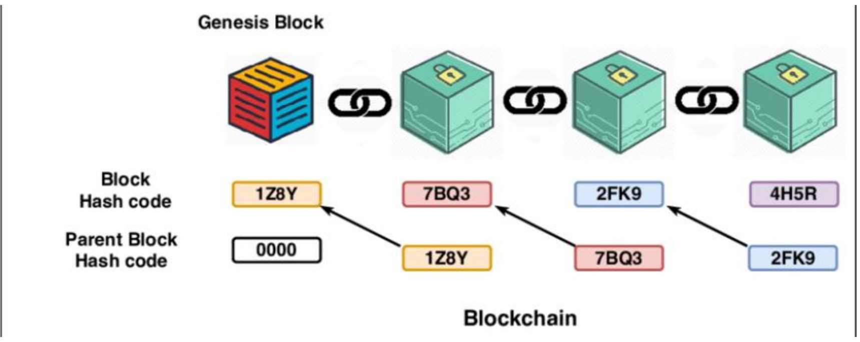


Figure 1. Blockchain is a sequence of hashed blocks (Torky and Hassanien 2020)

This data structure, called ledger or distributed database in a lot of resources, is maintained by different nodes located in other parts of the world that are fully connected. When a new block is added to the blockchain, its information is stored in each node's hard disk. This concept of not keeping the data in a unique server or location is known as decentralization and is the most crucial blockchain feature. Information is decentralized because there is no centralized server or a set of them that stores the data, but the data are distributed over the entire network. Using decentralized devices is a very secure way to keep the data because if a hacker wants to remove a block or modify a transaction, it is necessary to hack 51% of the machines by which the network is composed. If this occurs, the other 49% of the nodes will modify their information because 51% of nodes have different information. (Haynes 2022.)

The first blockchain was called Bitcoin, like its cryptocurrency. The bitcoin blockchain is specifically created to store and manage bitcoin cryptocurrency transactions, but it is slow. For this reason, nowadays, new blockchains and different types of blockchains are appearing. Ethereum, Solana, and Monero are examples of new popular blockchains that try to improve the Bitcoin blockchain lacks. (Haynes 2022.)

Ethereum blockchain specializes in managing smart contracts and the information stored in them. This approach is different from bitcoin because blockchain technology is well suited to handle other information beyond cryptocurrency transactions. Another interesting use case could be tracing packages or tracking the product manufacturing process. For example, when a user buys a box and the asset to be tracked reaches some stage, this information is added to the blockchain. Then, the buyer can check the package status. Due to blockchain immutability property, nobody can modify information stored on it. Therefore, the user can trust that the package is where the blockchain says. (Ditsche and Streichfuss 2021.)

Currently exists two main types of blockchains:

- Public or permissionless: On these blockchains, anyone can create blocks or be a bookkeeper without needing permission from an authority. Bitcoin is an example of a public blockchain. (Seth 2021.)
- Private: These blockchains are formed by verified participants, so they are not truly decentralized because the organization controls them. But the information is not stored on a central server. It is stored among distributed ledgers. Blockchains used by private organizations are examples of it. (Seth 2021.)

Other types of blockchain exist, but it is unnecessary to explain because it is beyond this work's scope.

2.2 Smart contract basis

Some new blockchains appear to solve other blockchains' problems or improve some aspects of them. Ethereum seems to improve Bitcoin in programming aspects. In the book *Mastering Ethereum: Building smart contracts and DApps*, Antonopoulos and Wood say that *"Unlike Bitcoin, which has a very limited scripting language, Ethereum is designed to be a general-purpose programmable blockchain that runs a virtual machine capable of executing code of arbitrary and unbounded complexity."* (Chapter 1, Section 1, 43.)

Ethereum was built to allow developers to create their own decentralized applications (DApps) and deploy them over the Ethereum blockchain. In this way, the developers make Ethereum worthwhile. This way of proceeding is very interesting because developers bring new functionalities and applications to the blockchain. The good applications make the popularity and the use of a blockchain grow. Of course, this increases the cryptocurrency and blockchain's usefulness and value. (Reiff 2022.)

Therefore, due to Ethereum giving importance to the developers, the company needs to build good tools to carry out the programming task in a comfortable and user-friendly way. Ethereum got it thanks to tools developed by itself like Remix IDE or by third parties like Truffle or Ganache. (GeeksforGeeks 2022.)

Thanks to smart contracts, it is possible to program the applications that will run on a blockchain. Smart contracts were mentioned for the first time in 1996 by Nick Szabo in his paper *Smart Contracts: Building Blocks for Digital Markets*. Szabo defines smart contracts as a "*set of promises, specified in digital form, including protocols within which the parties perform on the other promises."* (Szabo 1996.)

Smart contract definition changes depending on the context. Antonopoulos and Wood define smart contracts in their book Mastering Ethereum: Building smart contracts and DApps as "*immutable computer programs that run deterministically in the context of an Ethereum Virtual Machine as part of the Ethereum network protocol."* (Chapter 7, section 1, 246.)

This definition is uniquely valid in the context of the Ethereum blockchain because only the smart contracts deployed either on Ethereum or EVM-compatible blockchains, like Binance Smart Chain, accomplish this definition. For this reason, blockchains like Solana have their smart contract definition as well. The smart contract is defined in Solana's documentation as a "*program on a blockchain that can read and modify accounts over which it has control."* As can be read, Solana talks about modifying accounts, but the other definitions don't talk about this aspect. (Solana A.)

Notice that defining the smart contract term in a general way is complicated because it depends on the context. Based on the above definitions and my experience working with it, a smart contract could be described as a programming script or set that runs on a blockchain and triggers actions when an event occurs.

2.3 Blockchains that use SC

Due to smart contracts rising popularity, other developers began to develop their blockchain using Ethereum's strategy, i.e., allowing the community to develop their own applications for the blockchain. This is the case of Terra or Solana, among others. (Barker 2021; Terra.)

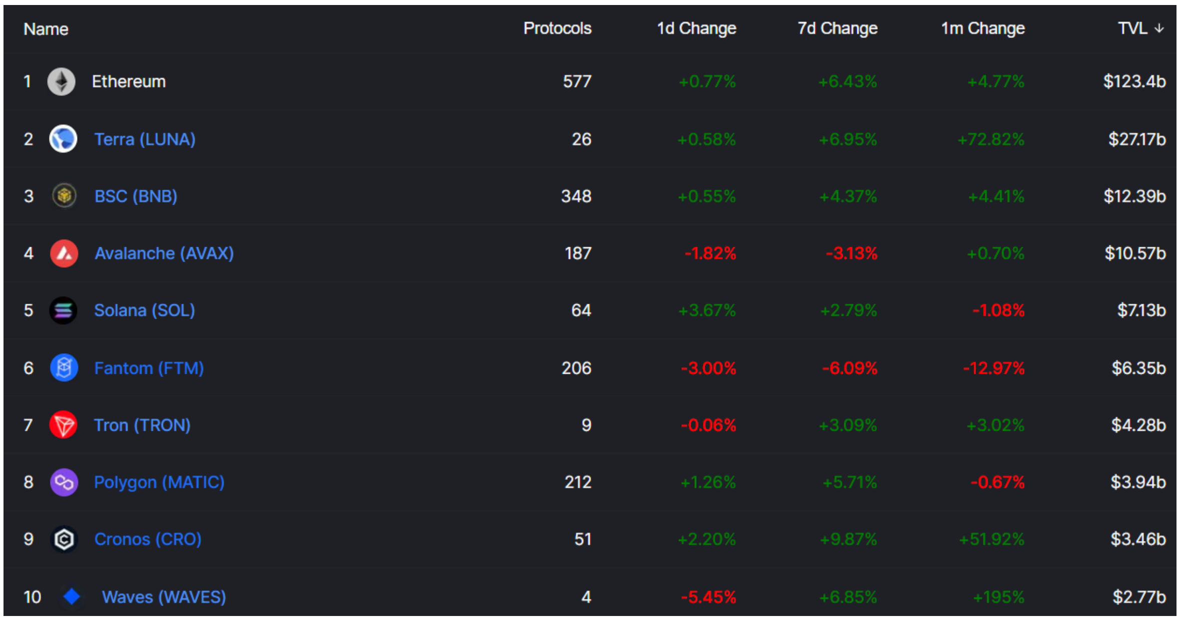
It is also possible to build an EVM-compatible blockchain, like Binance Smart Chain (BSC). This means that an independent blockchain exists from Ethereum, but the EVM engine is used to compile the smart contracts deployed on an EVM-compatible blockchain. This approach allows using smart contracts in the same way that Ethereum does. (Temitope B.)

Furthermore, there exist blockchains that use different approaches, like Bitcoin, which is the most famous blockchain and the first one. Bitcoin blockchain doesn’t use smart contracts in this same way. It uses smart contracts to manage transactions but does not create DApps or allow developers to do it. Each blockchain has its particularities to code and deploy smart contracts, for example:

- Ethereum: Smart contracts deployed on Ethereum run on the Ethereum Virtual Machine (EVM), a virtual machine that runs EVM bytecode. It is like CPUs that run machine code. Because programming using bytecode language is too hard, programmers make use of high-level languages to program smart contracts, like LLL, Serpent, Solidity, Vyper, and Bamboo. Once the smart contract is programmed using one of the languages mentioned, the code is ready to be compiled into bytecode and read by the EVM. (Antonopoulos & Wood, 2019, 250, 251.)
- Binance Smart Chain: This blockchain works with smart contracts compatible with EVM. Therefore, whichever programmed smart contract with Solidity or Vyper language is suitable to be deployed on BSC and Ethereum. Moreover, a cross-chain system is implemented to transfer tokens from different blockchains to BSC or vice-versa. (Binance.)

  Furthermore, other blockchains make use of EVM to run their own smart contracts. Avalanche or Fantom are two good examples, but more than a hundred blockchains take advantage of EVM to compile their smart contracts. (Coinguides 2021.)

  To figure out the importance of EVM, in the next image top ten blockchains are displayed with the most asset management. Six blockchains (Ethereum, BSC, Avalanche, Fantom, Polygon, and Cronos) are EVM-compatible in this list. And their TVL (Total value Locked), which represents the total number of assets stacked in a blockchain, amounts to 160.11 billion dollars. The TVL on Non-EVM-Compatible blockchains (Terra, Solana, Tron, and Waves) amounts to 41.35 billion dollars. (DeFi Llama 2022.)



	Name	Protocols	1d Change	7d Change	1m Change	TVL ↓
1	Ethereum	577	+0.77%	+6.43%	+4.77%	$123.4b
2	Terra (LUNA)	26	+0.58%	+6.95%	+72.82%	$27.17b
3	BSC (BNB)	348	+0.55%	+4.37%	+4.41%	$12.39b
4	Avalanche (AVAX)	187	-1.82%	-3.13%	+0.70%	$10.57b
5	Solana (SOL)	64	+3.67%	+2.79%	-1.08%	$7.13b
6	Fantom (FTM)	206	-3.00%	-6.09%	-12.97%	$6.35b
7	Tron (TRON)	9	-0.06%	+3.09%	+3.02%	$4.28b
8	Polygon (MATIC)	212	+1.26%	+5.71%	-0.67%	$3.94b
9	Cronos (CRO)	51	+2.20%	+9.87%	+51.92%	$3.46b
10	Waves (WAVES)	4	-5.45%	+6.85%	+195%	$2.77b

Figure 2. Top ten asset managing blockchains (DeFi Llama 2022.)

Of course, other blockchains that use their own virtual machine and their own languages to develop and deploy smart contracts exist. The two ones most used are:

- Solana: This blockchain allows the developers to use Rust, C, or C++ to develop smart contracts that will be executed via the Solana Runtime. Unlike Ethereum, Solana uses well-adopted programming languages to deploy its smart contracts and DApps, which is very interesting because new programmers don't need to learn a new language to develop a DApp on Solana's blockchain. The compiler used by Solana to compile the smart contracts is LLVM, another well-adopted tool by programmers. (Solana B.)

- Terra: Terra is another blockchain that takes advantage of smart contracts to extend its capabilities. As well as Solana, Terra makes use of well-known programming languages to develop its smart contracts. The programming language used by Terra is Rust. (Jonah 2022.)

2.4 Smart contracts: Advantages and disadvantages

One of the advantages of blockchains that make use of smart contracts to build applications is that developers make it worth the blockchain. From a blockchain point of view, attracting developers who work in applications that will be deployed on top of the blockchain is like hiring employees who work for free. Developers can deploy applications like games, NFT's marketplaces, cryptocurrency exchanges, and whichever application they can imagine. An attractive application attracts new consumers, and these consumers could spend money on this application, making growth the blockchain and cryptocurrency popularity. Moreover, these applications could help other developers to build their ideas or think about improvements to this application and create their own. But this is just an advantage only for blockchains that apply this kind of approach. (Anwar 2018.)

Independently of the blockchain nature where smart contracts are used, some common advantages exist due to the intrinsic smart contract characteristics.

- Task automation. In smart contracts, it is possible to trigger an action when an event occurs. Let's imagine that a railway company sells tickets using smart contracts. And in this smart contract, a function is programmed to automatically return 25% of the ticket price in case the train is ten minutes late. Then customers don't have to do anything to claim their money back. (Smith 2019.)

- Transparency: This is one of the most well-known advantages in the blockchain world. All the transactions are recorded and made public, and accessible by all users through platforms like etherscan. The information about the transactions is shown on these websites, but not only the transactions data, but the contract details are public as well. This makes it possible to search the contract and display the code just as was programmed. Furthermore, the possibility to read the smart contract code adds a security layer because someone could analyze the code and then determine if exists the possibility of being scammed. In the train example exposed in the above point, users would be able to verify that effectively, if a train suffers some delay, 25% of the total ticket price would be returned. (Smith 2019.)

- Speed: The last advantage is the speed at which things are done. Because no bureaucracy exists, processes like sending money to whoever in wherever world part or other kinds of transactions are done very quickly. Just waiting for the time needed while the transaction is verified. (Smith 2019.)

These were the main advantages of smart contracts, but of course, there are some disadvantages as well:

- Immutability: When some data is added to a blockchain, the changes can not be undone. For this reason, once a new smart contract is added to the blockchain, its content will be impossible to be modified. If a smart contract is unsafe and a way to exploit this security breach is found, this might cause many troubles to the users in the network. It is possible to use the self-destruct function to destroy the smart contract and avoid this unfortunate situation. (Smith 2019; Solidity by Example A.)

- Gas fee: Pay fees when the smart contract is deployed could cause some troubles, like, for example, don’t having enough money to deploy it. Depending on the smart contract complexity and the blockchain where the smart contract it’s being deployed, the contract deployment can be either expensive or cheaper. In any way, a wallet with some funds is needed to deploy the smart contract. In case of more than one smart contract has to be deployed, who deploys it must pay for all of them. The cost of these fees depends in part on blockchain nature. (Antonopoulos & Wood, 2019, 293.).
- Scalability: Proof-of-work is an algorithm used to validate blocks and their transactions. It consists of a computational problem that must be solved to validate a block and its transactions, so it takes some time. The time needed to solve this problem is adaptative, so when a problem is solved in less than a certain time, the difficulty is increased. For this reason, the number of validated transactions per time unit is always in the same range. It does not matter if one transaction or one million are waiting to be validated. It is not possible to increment the number of proceeded transactions per time. In case of blockchain receives a lot of transactions, the users would have to wait some minutes until the transaction is processed. (Khan et al. 2021, 13.)

3 Decentralized Applications (DApps)

3.1 DApps basis

On many websites, decentralized applications (DApps) are defined as a smart contract with a frontend, but this definition is wrong. If a centralized company manages the smart contract and their data are also centralized, this is not a DApp. In fact, it is a traditional centralized application. A real DApp is formed by different components that must be truly decentralized to accomplish the DApp purpose. (Antonopoulos & Wood, 2019, 475.)

A DApp is an application whose data storage, frontend, backend, message communications, and name resolution are partially or entirely decentralized. (Antonopoulos & Wood, 2019, 475.) Technically this is tricky to achieve but not impossible. Many tools to decentralize an application are being deployed due to the rising up blockchain popularity. For example, Pinata is a very nice site to store data, Ethereum Name Service allows to buy a decentralized domain name, and other tools exist to achieve the goal of creating a DApp itself.

3.2 DApp's components

DApp is composed of some elements that must be decentralized in order to build a real DApp. These elements are:

Backend: Also known as a smart contract. It is the part where the business logic is programmed, or in other words, what a DApp is able to do. The backend should be kept as light as possible because deploying a smart contract or executing a function costs gas. The larger the smart contract or function to be executed, the more expensive it will be to deploy it. The gas cost acts as a security system because, without this gas, it would be possible to execute a function that never ends, which could cause a lot of trouble to the network. Then it is necessary to pay a transaction cost each time a transaction that modifies the blockchain (Upload or modify blockchain data, upload smart contract, etc.) is executed. It's possible to run functions as long as the economy allows. (Antonopoulos & Wood, 2019, 477.)

Frontend: Programming languages like HTML, CSS, or JavaScript can be used to program the graphical user interface part. It is unnecessary to have advanced knowledge about EVM because the programmer doesn't interact with it. It only uses well-known tools, libraries, and frameworks to make DApp usage user-friendly. This front-end is usually programmed with JavaScript and uses libraries like web3.js or ethers.js. These libraries help communication between the front-end and the smart contract. (Antonopoulos & Wood, 2019, 478.)

Data storage: Because of gas cost, smart contracts are not well suited to storing and processing a large amount of data. For this reason, the information needs to be stored off-chain, i.e., out of the blockchain. It is possible to achieve this using traditional centralized servers, but IPFS is the best option. (Antonopoulos & Wood, 2019, 479.) There exist two different ways to store the information in a decentralized way:

- IPFS: Interplanetary File System. It is a decentralized file system where the users of a P2P network maintain a blockchain's data. Each piece of data is tagged with a hash that identifies it. Then, the user can retrieve this data when necessary. (Antonopoulos & Wood, 2019, 479.)
- Swarm: This is another decentralized file system created by the Ethereum Foundation similar to IPFS. As same as IPFS, the data can be accessed by hash. (Antonopoulos & Wood, 2019, 479.)

Message communication: Exchange of messages between applications, users, or different instances of DApp is also an essential part. For this reason, the Ethereum Foundation offers the Whisper protocol, a protocol to exchange messages over a decentralized P2P network. (Antonopoulos & Wood, 2019, 480.)

Name resolution: The Ethereum Name Service (ENS) is the same as DNS, but it acts decentralized. Traditional DNS translates a domain name into an IP address, making human-friendly access to this IP address. ENS works the same way, but now a public key is translated into a *name.eth* structure. Public keys are formed by forty-two hexadecimal characters, which are more difficult to remember for humans. Thanks to ENS, it is now possible to give some nicknames to this public key, making it easier to remember and use. ENS can be used to create a domain name or provide an alias to a wallet address, making it easier to transfer funds to this account. (Vardai 2021.)

3.3 DApps main features

Some advantages or features could never be part of centralized applications. Antonopoulos and Wood describe those three features in their book:

- Resiliency: The DApp backend is stored on a blockchain, and the DApp will be available as long as this blockchain is operating. Maybe some part of the DApp is not decentralized. Therefore, access to the application could be restricted, e.g., think about the domain name being translated using a traditional DNS instead of ENS. The company could modify the web URL or finalize the contract with the DNS company. Then the users couldn’t find the website but could fund the smart contract with tools like etherscan. Unless the whole blockchain is down, the smart contract always is part of it and can always be found in one way or another. (Antonopoulos & Wood, 2019, 475.)

- Transparency: Because the backend is a smart contract, everyone can inspect the code. Furthermore, the transactions on the blockchain are public, then whoever can inspect the transactions carried out with this smart contract. In this way, the users can check if the smart contract is legitimate or if there is something that is not legitimate. (Antonopoulos & Wood, 2019, 475.)
- Censorship resistance: As with a blockchain transaction, a smart contract can not be modified after being uploaded to the blockchain or replaced by another one, so it can be bad because developers are unable to modify functions or correct bugs. (Antonopoulos & Wood, 2019, 475-476.)

4 Web 3.0

4.1 Web 3.0 introduction

Web 3.0 is a new term used to refer to the latest version of traditional webs. But another term called Web3 was coined by Dr. Gavin Wood (Ethereum founder) to refer to Web 3.0 focusing on blockchain purposes. Many web pages and writers use Web 3.0 and Web3 terms in the same way, which causes some confusion to newbies. In order to avoid this possible confusion to the reader, these two different terms are exposed in the next lines.

Web 3.0: Tim Berners-Lee, inventor of the World Wide Web in 1989 and Director of the World Wide Web Consortium currently, coined this term as a data web that can be processed by machines. (Investopedia team 2022.)

Web 3.0 can be explained as a web that uses artificial intelligence technologies, like machine learning or natural language preprocessing, to interpret data introduced by the users in a human way. This way of interpreting the data provides accurate results and improves the user experience. Tim Berners-Lee makes use of the term "semantic web" to refer to this kind of webpages. (Vermaak 2022.)

Web3: Ethereum refers to this term on its documentation as "*decentralized apps that run on the blockchain. These are apps that allow anyone to participate without monetizing their personal data*". (Richards 2021.)

Because these kinds of websites are allocated on a blockchain, the information uploaded by users can not be modified once uploaded, e.g., tweets could not be censored. Another difference with web 2.0 is that no personal data is required in Web3 to make a payment. In addition, it is difficult for a webpage to go down because it is maintained by many nodes, not by a centralized server. (Richards 2021.)

4.2 Web3 history

Web3 is a very new topic in the blockchain world, so its information is scarce and constantly evolving. Firstly, to understand Web3 and its origins, it is necessary to explain web 1.0 and web 2.0.

Web 1.0: Also known as static websites. This kind of webpage was the first on the internet, was used from the earlier nineteens to 2005 approximately. These websites were read-only, which means the information was written by the developers on the webpage, then the users read this information. The webpage could not be edited by users, e.g., by adding comments or creating a post. (Bhattacharya 2021.)

Web 2.0: In the mid-2000s, an evolution of web 1.0 appeared and is still used. Now the website can be edited by users because they can add data to it. This kind of web was readable-writeable. The negative side is that once this data is on the internet, it can not be controlled by users. Furthermore, the platform and data are centralized, which means that a hacker only needs to attack the company servers to steal data or provocate bad platform behavior. Web3 appears to solve these problems. (Bhattacharya 2021.)

Web 3.0: Web3 is DApp's frontend allocated on a blockchain. No personal data are involved because users are registered using a crypto wallet, like Metamask, Coinbase, or another wallet allowed by the website. The process to connect a user with a Web3 website would be similar to the next one: A wallet extension is installed in the user's browser. Then, when the user wants to be registered on a new website, only the wallet must be connected to the website. This is an easy step because only a click on the webpage button to link the wallet to this website is needed. The information provided by the wallet is anonymous because, from the website point of view, only a public address (wallet public key actually) has been connected. There are no user names or emails that can be linked with a person. These data are decentralized because the DApp backend is on a blockchain. Then, nobody can control it, and companies can not sell users' data. (Dabit 2021.)

4.3 Web 3.0 architecture

Similar to DApps, these websites need smart contracts where business logic is programmed. A frontend is also necessary to allow user interactions whit it. In addition to these two components already explained in section 2.2, web 3.0 also needs another two essential elements to work correctly:

- Blockchain: This is the base of Web3, and without it would not be possible to create this new web concept. This kind of website has smart contracts as the backend, and there is no other way to execute a smart contract than run it on a blockchain. (Bhattacharya 2021.)
- A virtual machine to execute smart contracts on a blockchain is also needed. Not every server around the world is able to compile Solidity code or other languages related to smart contract programming. For this reason, specialized machines that can do it are needed. (Bhattacharya 2021.)

The next image shows the Web3 architecture:

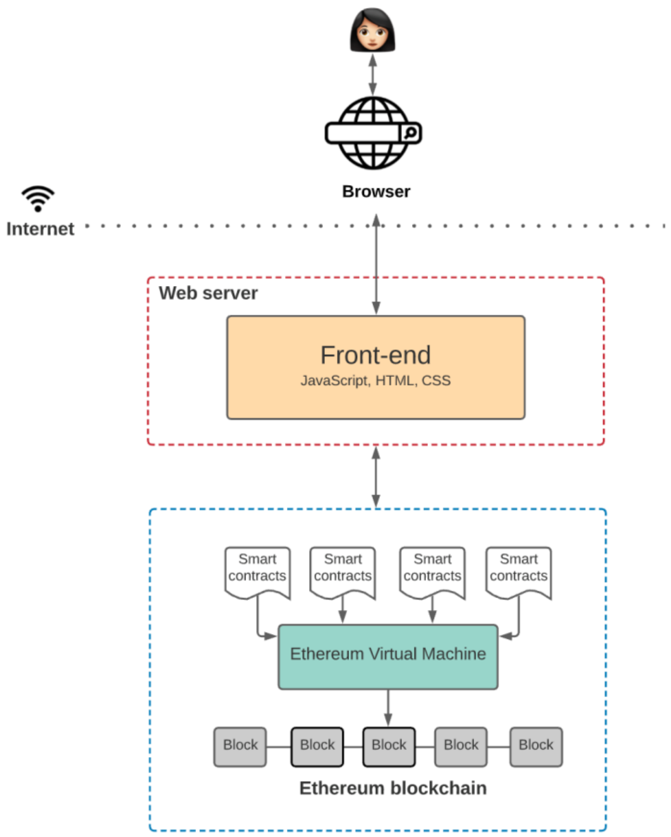


Figure 3. Web 3.0 application (Bhattacharya 2021.)

Figure 3 shows the different Web3 components and how each one interacts with the next one. Throughout a browser, and with an internet connection, the user can connect to whichever website, as always done. But this time, instead of working with centralized servers, this frontend is directly connected to smart contracts, which act as the backend. In this case, smart contracts are deployed on the Ethereum blockchain. Then the EVM is used to compile the smart contract code and convert it to bytecode and interact directly with the Ethereum blockchain.

4.4 Web3 limitations on Ethereum

In order to build a DApp on top of a blockchain, it is essential to know its limitations and know what can be done or not. The Ethereum blockchain has some limitations that the organization is trying to solve, or at least trying to mitigate. Ethereum company is developing a new version of Ethereum called Ethereum 2.0. Explanation of what Ethereum 2.0 is and how it works is beyond the scope of this document, but Ethereum 1.0 limitations are listed next:

- Scalability: All the transactions must be processed, validated, and mined to be added to the blockchain. This process takes between 12 and 14 seconds, which is the time to mine a block on Ethereum. Block time, or the time needed to mine a block, is a constant in this blockchain, so it never will be less than twelve seconds. In case of an increment in Ethereum transactions, this block time would be a bottle-neck, and many transactions would wait in the transaction pool. (Richards 2021; Smith 2022.)
- Cost: Another problem related to scalability is the cost of a transaction, also known as the gas fee. An additional charge is added to the base transaction price (if any) to avoid unnecessary transactions. When a transaction is executed, this fee can be modified directly by the user throughout the wallet. Due to Ethereum prioritizes transactions with expensive fees, other transactions with less cost could remain per-manently in the transaction pool. Furthermore, blockchain popularity makes rise the price of executing a transaction. (Richards 2021; Richards 2022.)
- Accessibility: Different browser extensions will be needed to register users into Web3 applications. The problem is that some browsers are not actually prepared to integrate all these requirements. (Richards 2021.)

4.5 Main Web3 advantages and disadvantages

Like any other technology, Web3 has some advantages and disadvantages:

The main advantages are:

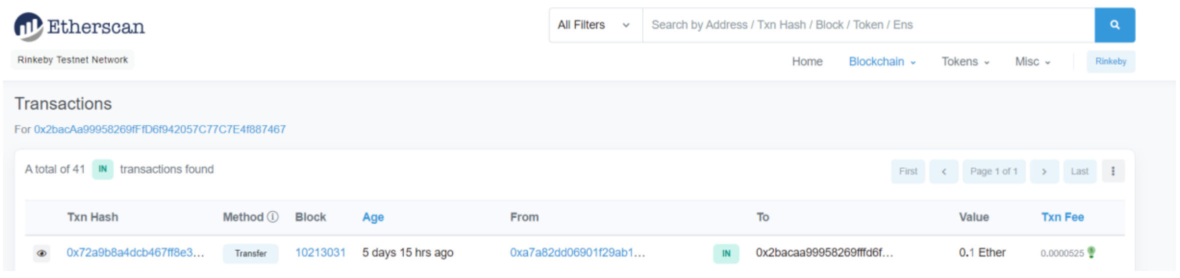
- Transparency: As in all blockchain-based technologies, the transactions carried out in Web3 are public. (Bhattacharya 2021.)
- Single profile creation: As commented in the 3.0 section, the users will need only one profile, or their wallet, to register the wallet on a website. No email addresses, nicknames, identification numbers, or bank accounts are required in order to interact with blockchain-based applications. A wallet provides a unique ID that can be used as a nickname and all the necessary elements to make payments or receive money. (Bhattacharya 2021.)
- Decentralization: A real Web3 website is not allocated over a single server. This means that no unique organization is responsible for maintaining the website but is maintained by an entire network of nodes disturbed worldwide. Therefore, the web will always be operative, and an inaccessible state is almost impossible to reach. (Richards 2021.)

The main disadvantages are:

- Data privacy: On many websites, one can read that data in blockchain are private because blockchain uses cryptography to encrypt the data. This affirmation is false, at least in Ethereum. In fact, Antonopoulos and Wood write the following lines in their book Mastering Ethereum: Building smart contracts and DApp's: "*At the time of publication, no part of the Ethereum protocol involves encryption; that is to say all communications with the Ethereum platform and between nodes (including transaction data) are unencrypted and can (necessarily) be read by anyone*." (Antonopoulos & Wood, 2019, 138.)

  In some blockchains, cryptography is used to sign transactions but not to encrypt data. Therefore, data is not private. It is suitable to say that data is private because it is generated through a public key and not from a nickname or email like in web 2.0, so data is anonymous. Then, knowing who realizes a transaction is difficult to guess because only the public key is visible, but it is not impossible to link this key with someone. (Glover 2021; Eichler & Jansen 2017.)

  It is complicated to know the wallet owner's identity, but it is not difficult to watch all the wallet movements. In fact, tools like etherscan are built with this purpose, and the entire transaction information is published and displayed in a way that everyone can understand. For this reason, having multiple accounts and registering them in different applications to avoid this tracking is recommended by some experts.

  Transactions

  For 0x2bacAa99958269fFfD6f942057C77C7E4f887467

  A total of 41 IN transactions found

	Txn Hash	Method	Block	Age	From		To	Value	Txn Fee
	0x72a9b8a4dcb467ff8e3...	Transfer	10213031	5 days 15 hrs ago	0xa7a82dd06901f29ab1...	IN	0x2bacaa99958269fffd6f...	0.1 Ether	0.0000525

  Figure 4. Etherscan transactions. (https://rinkeby.etherscan.io/)

  As shown in the last image (Figure 4), it is possible to search information by filtering using an account, then all the transactions that involve this account are displayed. In the previous image can be appreciated a transaction five days old, from one account to another 0.1 ether worth.

- Complicated functionality: This technology is difficult to understand for new users, and sometimes it could be tedious to use. Sometimes, it is necessary to learn how to interact with these applications and manage a wallet; that is not always easy. (Bhattacharya 2021.)

- Not ready for widespread adoption: Due to it being a new technology, many users are not ready to use or program advanced Web3 applications and websites. More understanding and learning are required for the full adoption of this technology. (Bhattacharya 2021.)

5 NFTs

5.1 NFTs introduction

NFTs are a very new concept and are a very disruptive technology. NFT is the acronym for Non-Fungible Token. The word “fungible” means that something exactly equal or with the same value exists. Because these tokens are non-fungible, it can be said that there are not two NFTs exactly equal, and this is the main characteristic of this new technology. Then an appropriate definition of NFT could be a type of cryptographic asset representing a unique item (Hedera.)

NFTs was firstly launched in the Ethereum blockchain. This first project was launched in 2015 and was called Etheria. Currently, blockchains like Solana support NFTs as well. One advantage of Ethereum regarding other blockchains is the creation of standards (smart contract samples) for this type of token, which makes NFTs development tasks more manageable. But because of high fees in the Ethereum blockchain, the NFT market is moving to cheaper blockchains like Solana (White-Gomez 2021; Joshua 2021 A; Canny 2022.)

There exist other blockchains like Tezos or Cardano that support NFTs. Even blockchains like Flow have been created exclusively to support NFTs and their applications. These blockchains and their support for NFTs were developed because Ethereum has high fees, so the transaction to mint or buy NFTs is expensive. For this reason, Ethereum is losing clients in favor of Solana’s blockchain. (Chatfield 2021.)

NFTs are considered an excellent way to check if someone poses something. Therefore, once this information is checked, someone (an organization, company, club) could offer some benefits to the NFT holder (who possesses the asset). And this is the reason why the NFTs are interesting. Influencers, entrepreneurs, big brands, artists, and a lot of companies could build better communities using NFTs. Furthermore, other use cases like collectibles, game assets, or art are other different scenarios where NFTs can be helpful. (Jaswal 2021.)

These use cases are possible thanks to the NFT attributes:

- Indivisibility: This is one of the main NFT features that makes it exciting and powerful. Indivisibility means that a part of an NFT can’t be bought. One always has to buy a whole NFT. (Geroni 2021.)
- Ownership: Because of indivisibility, NFT is only held by one user. In other words, an NFT belongs exclusively to one wallet account simultaneously. NFT holders are able to transfer it from one account to another quickly. (Geroni 2021.)

- Uniqueness: Probably this is the most valuable NFT feature. When an NFT is bought, the holder can be sure that there does not exist another NFT like his one. Typically NFTs are represented by an image, then thinking that an NFT can be duplicated is reasoning. In fact, it is possible, but the reality is that NFTs are tagged with an identification number and have associated a smart contract address. Therefore, creating an NFT represented by the same images as the original one is possible. But there is no way to link this non-original NFT to the smart contract of the original collection. Then, the original NFT is unique. (Geroni 2021.)
- Scarcity: Sometimes, the NFT value is given by its scarcity. Infinite NFTs can be minted, but to create a good relationship between the offer and demand is a good practice to create a limited quantity of tokens. (Hedera.)
- Transparency: Like all blockchain transactions, NFTs transactions are public as well. Therefore, it is possible to trace a particular NFT to know how many times the NFT has been traded, which could help to know if the NFT traced is the original one. Transparency helps to read the NFT smart contract, and this is very useful to see if it is a good collection or a scam. (Hedera.)

5.2 Past, present, and future of NFTs

Nowadays, some people consider NFTs a scam, which could be partially true. Once something generates money, the scammers appear to take advantage of the situation and try to scam newbies. For example, Lana Rhoades, a famous adult film actress, scammed her fans with her collection. (Conroy 2022; Gogo 2022.)

Scams partially appear because some people don’t know how NFTs should be bought. Purchase an NFT is not only connecting the wallet to the marketplace and acquiring it. It is about getting information about the project. Knowing the founders, their experience in NFTs, the NFT utility, and if they can keep their word and accomplish the promises. These all are essential points to check before buying an NFT. (Yang 2022.)

Another reason why people think that NFTs are a scam is that NFTs are currently mainly used to sell digital art pieces. Although NFTs are used in other areas, such as gaming, art is where the most money is spent. In 2021, $22 billion were spent by collectors and investors in digital art. NFTs are mainly used to sell art, and historically good artwork has increased in value over time. Then, many NFTs are bought by people who hope that the NFT value will rise and makes them rich. Unfortunately, this is far from reality because only a few collections are good enough to increase their value. This belief that an NFT could be worth two or three times as much in the future comes from collections like CryptoPunks. Five years ago, these collection items were gifted or sold for around $3, but now the most expensive one is “CryptoPunk 7523,” which is $11.75 million worth. (Escalante-De Mattei 2021; Bureau 2021)

Another similar case exists. In April 2021, the first NFT of a collection named “Bored Ape Yacht Club” was sold for $188 approximately. Currently, the cheapest one is around $187.000. Because of these collection growths, some people try to find the next millionaire-maker collection, so when this does not happen and NFT turns cheaper or loses value, the users feel scammed. Thinking that most NFT collections will increment their value is wrong. According to Gary Vaynerchuk, entrepreneur and NFT investor, most NFTs will be worthless in a few years, and only a few collections will increase their value. (NonFungible 2022; Rosen 2022.)

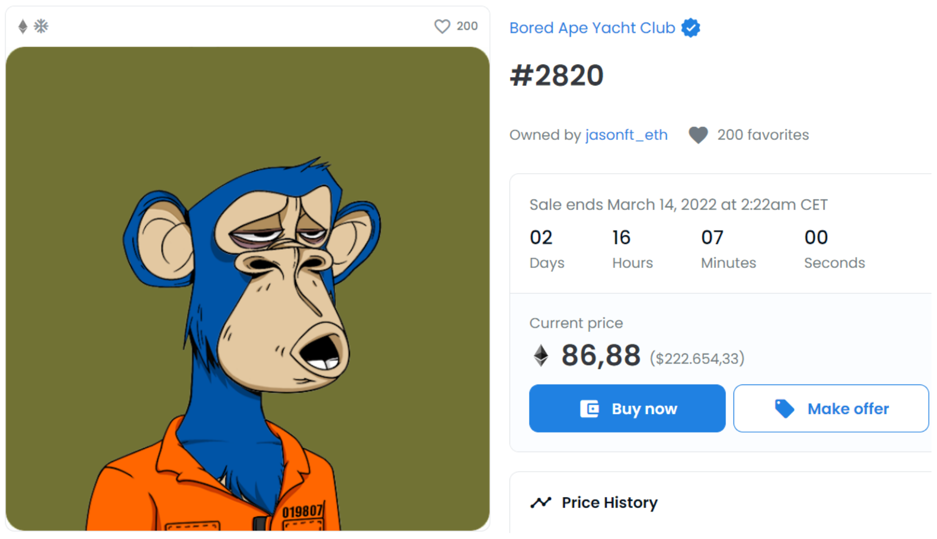


Figure 5. Bored Ape 2820 and his NFT price. ([2820 - Bored Ape Yacht Club | OpenSea]())

The possibility that most NFTs will be worthless does not mean that NFTs don’t have a bright future or don’t have any utility. Due to NFT's properties, like the possibility of knowing who poses what, the NFTs are a perfect way to create digital contracts to sell something physical. In fact, NFTs are being used in the real estate industry. Companies like Propy sell houses throughout NFTs, which is possible thanks to NFTs properties. Uniqueness, ownership control, and non-divisibility allow buying physical products and storing the information in a blockchain. Users will always be able to demonstrate that they are the house owner. In this case, NFTs are acting like registering on public records. (Quiroz-Gutierrez 2022.)

Also, in the near future, NFTs could be used as tickets. This way of using NFTs is easier to control than a rental agreement because there is no legal bureaucracy involved. NFTs can provide some benefits in the ticket sale area. (LeewayHertz.)

These benefits are listed here:

- Authenticity: Selling non-authentic tickets is a typical way to scam people. NFTs reduce this risk because there are no physical tickets. In addition, the smart contract of these NFTs can be accessed to check if the ticket is legitimate. (LeewayHertz)
- Possibility to collect: An NFT can be saved for life. Tickets can be collected as a souvenir of having assisted at an event. (LeewayHertz.)
- Possibility to revaluation: In the case that an event turns into a historical event, for example, a football match, some people could want to buy the NFT of this event, then the original buyer could sell it at a higher price. (LeewayHertz.)
- Resale royalties: In some events, re-selling tickets is illegal, so this can be easily controlled in the smart contract programming a rule to avoid re-sell the ticket. On the other hand, it would be possible to allow re-sell the ticket and destinate part of the re-sell price to the main organization. This functionality acts like royalty, so each time a ticket is re-sold, its creator receives a percentage of the total amount. This is a significant advantage in ticket selling. (Thune 2022.)

NFTs can also be used in businesses, and the main reason is, again, digital ownership. There already exists NFTs collections that provide some benefits to the holders. These benefits are not monetary, but the holders have access to exclusive events, restaurants, and courses. And this is the main utility of NFTs, providing benefits to communities. I.e., Gary Vee has his own collection, and whoever acquires one NFT of this collection will have access to his three-day event. (Veefriends.)

6 Practical case: DApp development

6.1 Introduction and goals

This practical case is used mainly to expose all the concepts explained in this document graphically. Also, how the different tools explained in this chapter interact between them to compose the final DApp, is exposed. The DApp is based on a fast-food restaurant that sells its own NFTs in order to offer special sales for those who acquire one of these NFTs. Whichever company could use an application similar to the one created in this practical case, this is not uniquely valid for restaurants or similar businesses.

The DApp consists of two different parts.

- Backend: That is composed of two smart contracts programmed in Solidity.
- Frontend: In this part, the basic graphical user interface is programmed. It is composed of HTML and JavaScript code, responsible for establishing a link between smart contracts and HTML code.

But before deep diving into these elements, it is necessary to talk about some tools that are necessary to build the DApp, like programs, libraries, programming languages, and so on.

6.2 Software

In order to build a functional DApp, some tools are needed to develop the necessary environment. These tools save a lot of time for the developer and simplify all the tasks that must be carried out, like programming or deploying the contract in a production or test environment. These are the tools used to program the case DAPP:

- Truffle: This framework is used to deploy, compile, debug, and test smart contracts. Solidity smart contracts can be deployed on whatever EVM-compatible blockchain. Truffle is easy to use and easy to understand but doesn't have GUI, so it is necessary to know how the command-line interface works at a basic level in order to use this tool. (Truffle Suite C.)
- Ganache: A potent tool that simulates a blockchain on a local network. Ganache, as truffle, belongs to "Truffle Suite," so there are no compatibility problems between both programs. Thanks to this local blockchain, the developer can deploy, develop and test his entire DApp locally. Ganache provides a GUI with a lot of information

about blocks and transactions. Then Ganache is also helpful in learning how a blockchain works and how the transactions are recorded. Also, a command-line interface is available to interact with Ganache for more advanced users. Ganache is available for Windows, Mac, and Linux. (Truffle Suite B.)

- Node.js: A widely used JavaScript runtime environment that allows building whichever programming project. It is open-source and cross-platform (Node.js.)
- Visual Studio Code: A mighty easy-to-use code editor is available for Windows, Linux, and Mac. It supports JavaScript and Node.js. Furthermore, have extensions to support Solidity. (Visual Studio Code.)
- VSC Solidity extension: This extension makes easier the task of programming with Solidity. It has many interesting features, like text highlighting, keyboard shortcuts to compile contracts, code completion, etcetera. (Blanco 2021.)
- Metamask: This is a browser extension that allows the storage and managing of cryptocurrencies. It can be used to pay for an asset, make a money transaction, or interact with a blockchain, e.g., minting NFTs. Metamask is the most widely used wallet. Also, it is able to connect with EVM-compatible main-nets and test-nets, like Binance Smart Chain. In addition, metamask has a mobile version that allows viewing the acquired NFTs. This feature is not available in the browser extension version. (Hussey & Phillips 2020.)

6.2.1 Programming languages and libraries

To build a DApp that accomplices these characteristics are required some programming languages, like:

- Solidity: It is an imperative high-level programming language created by Dr. Gavin Wood, similar to JavaScript, C++, or Java. It is the most widely used programming language for smart contracts programming. (Antonopoulos & Wood, 2019, 251.).

  Based on my experience, Solidity is intuitive, easy to use, and easy to learn if the user has some programming knowledge and has coded with similar programming languages before. Maybe this is the reason for its widespread adoption.
- JavaScript: Programming language used to develop dynamic websites, or the same, a dynamic frontend with buttons or something interactable by the user. (MDN Contributors 2021.)

- HTML and CSS: Hypertext Markup Language and Cascading Style Sheets. Both languages are used to develop web pages. With HTML, the web page structure is built, and with CSS, the visual part is programmed.

Also is need to use a way to format the text:

- JSON: JavaScript Object Notation. It is a structured text representing some data on JavaScript object syntax. Commonly used to transmit data between user and server or between server and user. (MDN Contributors 2022.)

  In the practical case, some JSON files have been created automatically by truffle when smart contracts have been deployed. The purpose of these JSON files is to create a bridge between the JavaScript front-end and Solidity backend. When the smart contract is deployed, a file type called "ABI" is automatically generated in JSON format. This ABI contains all the information about the smart contract, like the functions programmed, compiler version to be used, function parameters and outputs of these, etc. Once created, it is possible to import these ABI files from the frontend page and use it to interact with the smart contract. Furthermore, the author of this work created other JSON files. These files are the NFT metadata that stores some data about the NFT features.

In addition to these two programming languages, installing some JavaScript and Solidity libraries is necessary to achieve a good application performance.

- Web3.js: Thanks to this library, developers can establish a connection between the user and the smart contract, or the same, between the user and the blockchain. Connecting the wallet to the site is necessary to allow user-blockchain interaction, which is possible with this library. Also, it is possible to connect the front-end with the back-end in a few lines of code. (McCubbin 2022.)
- React js: A JavaScript library that helps in user interface programming tasks. (React.)
- Axios: This library allows to communicate with servers using the HTTP protocol. (GeeksforGeeks 2021.)
- openzeppelin/contracts: This library is used not in the front-end scripts, as the other ones explained before, but is used in the smart contract. With this library, the user can import some standard smart contracts, like ERC1155 or ERC20. Once this library is installed, the developer must import some of these standard smart contracts and use the functions within. Furthermore, this library of smart contracts contains

interfaces that can be imported and implemented by the programmer. (Openzeppelin.)

6.2.2 Relevant project files

In the case of a DApp deployed with Truffle, it is essential to know how to edit the *truffle-config.js* file. This file is automatically created when the truffle project is initiated. By default, it is located in the project root directory. Another feature to comment on is that it is a JavaScript file. A lot of important information is configured in this file, like the blockchains where the project will be launched, the compiler version to use, the wallet account to use, or where the ABI files and smart contracts are allocated. (Truffle Suite A.) Some important fields of the truffle configuration file are:

- Networks: The blockchain where the project will be deployed must be specified in this file. By default, it is selected a testing network called *development* in this file. This network is used for testing the application on localhost. For this reason, it is necessary to simulate a blockchain using ganache software in the machine where the project is being developed. The developer can add other different networks in this file, e.g., the Rinkeby test-net can be added for testing in a real Ethereum test-net. When the DApp is finished, a real main-net like Ethereum or Polygon is added to deploy the DApp in a production environment. (Truffle Suite A.)

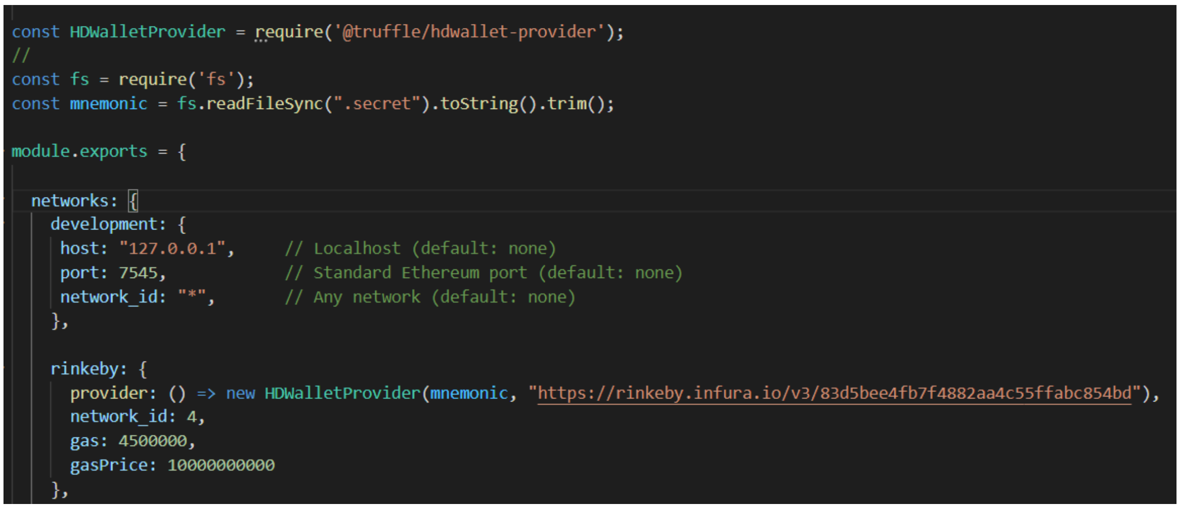
  Some parameters must be configured in the *networks* field to achieve a good deployment and a good DApp performance, such as:

  - Network_id: A network identification number that must be provided by the network provider. In the case of ganache, the user can select whichever ID when the project is created in ganache and write this same ID on this field in the configuration file.
  - Gas: In the case of works with a non-local network, this field should be filled to select the maximum gas to deploy the smart contract. If this field is not filled, a default amount of gas is used. (Truffle Suite A).
  - GasPrice: Gas price used for deploying the project. (Truffle Suite A.)
  - Provider: This allows both vendor selection, with whom a project will be launched on the blockchain, and the wallet account to deploy the contract. The provider field is essential because when a smart contract is put into production, it is necessary to pay gas to deploy it, and the

way to define which wallet will pay the gas is by filling this field. (Truffle Suite A.)

In order to set a provider, an external package is required. This package is called *hdwallet-provider,* and it is necessary to import it into the *truffle-config.js* file. Once imported, configuring the provider for the current network is needed to create a new *HDWalletProvider* object, and two parameters must be passed to it. The first parameter is the file where the private phrase necessary to import the wallet is written. This file is typically named *.secret*, and the variable to save it is usually called *mnemonic*. The second parameter is the Ethereum client to send transactions nonrelated to Web3. (npm 2022.)

To better understand this *networks* field, the reader can look at Figure 6.



```
const HDWalletProvider = require('@truffle/hdwallet-provider');
//
const fs = require('fs');
const mnemonic = fs.readFileSync(".secret").toString().trim();

module.exports = {

  networks: {
    development: {
     host: "127.0.0.1",     // Localhost (default: none)
     port: 7545,            // Standard Ethereum port (default: none)
     network_id: "*",       // Any network (default: none)
    },

    rinkeby: {
      provider: () => new HDWalletProvider(mnemonic, "https://rinkeby.infura.io/v3/83d5bee4fb7f4882aa4c55ffabc854bd"),
      network_id: 4,
      gas: 4500000,
      gasPrice: 10000000000
    },
```

Figure 6. Truffle-config.js: Networks configuration

- Compilers: The compiler used to compile the smart contracts is selected in this field. The compiler version has to match the version used on the smart contract. In another way, the contract will not compile. (Truffle Suite A.)

- Contracts_directory: This line defines the directory where the smart contracts are stored. By default, smart contracts are in the ./contracts-folder, but the standard directory can be modified if the developer desires it. (Truffle Suite A.)

- Contracts_build_directory: This is very similar to contracts_directory, but this keyword refers to where smart contracts ABIs are stored. The ABIs are stored in ./build/contracts by default, but this setting can be modified. (Truffle Suite A.)

Other information can be added, and other keys can be modified, but the keys mentioned above are the most relevant and used in the practical case.

The migrations file is another important file in a DApp project. It is a file programmed in JavaScript, created automatically when the truffle project is initiated. The migrations file's primary function is to deploy a smart contract, or a set of them, on the selected network. (Truffle Suite D.)

Another important detail related to migrations files is the file name. The name of this file starts with a number that indicates the smart contract deploying position. In the case of having three programmed smart contracts and their respective migrations files, the first smart contract deployed will be the one whose migration file name starts by 1. This file deployment is followed by the deployment of the file, which migration file starts by 2. The name is not only composed of a number but also is added a suffix with human readability purpose. An example of migrations file name could be: *1_example_migration.js, 2_example_migration.js* (Truffle Suite D.)

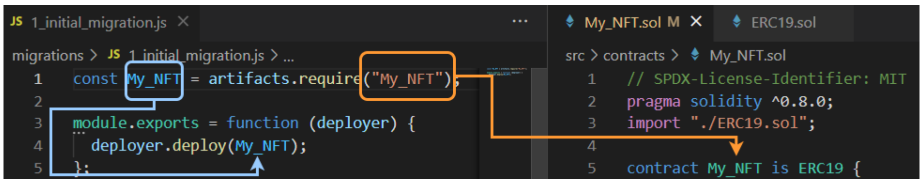
Migrations files are concise scripts and are composed of three essential parts:

- Artifacts.require(Contract): This sentence appears at the first line in the migration file, and in it is selected the contract to interact with. It is necessary to store the result of this instruction in a variable in order to use it later in the script. (Truffle Suite D.)

- Module.exports: All migration files must export a function that accepts a *deployer* object as the first parameter. This function provides a clear syntax to deploy smart contracts and carry out other tasks like saving deployed artifacts for later use. (Truffle Suite D.)

- Deployer.deploy(Contract): The developer gets the contract object stored in the variable created on the first line. Then this object is specified as a parameter in deployer.deploy(ContractObject). This function provides an address for the contract that will be modified each time that migration is executed. This function accepts more parameters but explaining all of them is beyond the scope of this document. (Truffle Suite D.)

To understand how the migrations files are built, look the figure 7.


1_initial_migration.js
migrations > 1_initial_migration.js > ...
My_NFT.sol M
ERC19.sol
src > contracts > My_NFT.sol

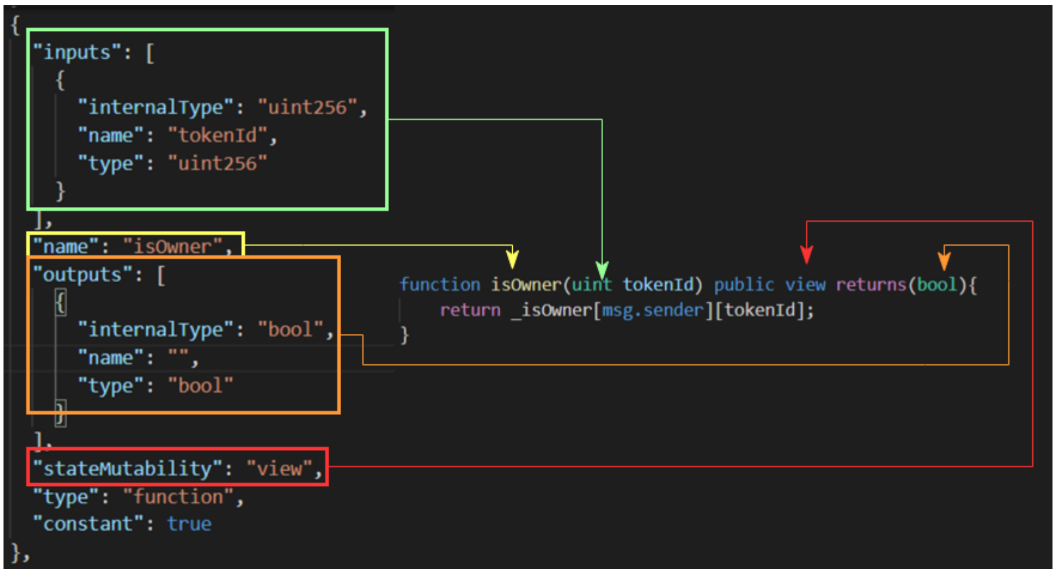
```javascript
const My_NFT = artifacts.require("My_NFT");

module.exports = function (deployer) {
  deployer.deploy(My_NFT);
};
```

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./ERC19.sol";

contract My_NFT is ERC19 {
```

Figure 7. Relation between migration file (left) and SC to migrate (right).

The last kind of file to comment on are the ABIs file. ABI comes from Application Binary Interface, and this file is responsible for allowing communication between the frontend and contract or between contracts themselves. These files define the contract functions that can be invoked. And describe the accepted parameters, outputs, and other relevant information about it. Furthermore, it defines how data structures and methods are accessed in machine code. In conclusion, front-end files and other smart contracts use ABI files to know how to invoke a function or use a data structure, like mappings. One ABI .JSON file is created for each smart contract used, either directly programmed by the developer or imported throughout a smart contract (GitHub contributors A 2021; Antonopoulos & Wood, 2019, 259.)

The next image (Figure 8) shows the relation between the ABI file and the smart contract described above.



```
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "tokenId",
      "type": "uint256"
    }
  ],
  "name": "isOwner",
  "outputs": [
    {
      "internalType": "bool",
      "name": "",
      "type": "bool"
    }
  ],
  "stateMutability": "view",
  "type": "function",
  "constant": true
},
```

```
function isOwner(uint tokenId) public view returns(bool){
    return _isOwner[msg.sender][tokenId];
}
```

Figure 8. Relation between ABI file and Smart Contract.

On the left side of figure 8, the JSON file is presented. This is the ABI file that contains information about smart contract functions. On the right side, the function isOwner is presented.

The function inputs are marked in light green. The information about the inputs contains the internal parameter type, the input name given by the programmer, and the type selected by the programmer, in this case, an unsigned integer. The next rectangle, in yellow, is used to tag the function name, which is the same name used by the programmer to refer to the function.

Next, highlighted in orange, the function output is shown. This function returns a boolean value. And finally, in red, the function state is shown, which refers to the function type.

6.2.3 Test-net

When a transaction is ordered and registered, there is no way to erase or modify it from the blockchain, and this happens as well when a smart contract or an entire DApp is deployed in a real blockchain. For this reason, the test-nets are useful. Test-nets are networks where a production environment is simulated. In these networks, fake money is used, which allows users to make use and learn about blockchain, wallets, Web3, etc., without spending real money. Usually, a blockchain has its main network (also known as main-net) and test blockchain. For example, Ropsten, Rinkeby, or Kovan are Ethereum's test networks. Before the smart contracts are deployed on the main-net, they should be deployed on a test-net. This practice allows to test the DApp in a real environment and check that it works properly. (Albert 2022.)

6.3 Solidity specifications

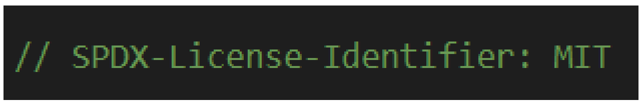
6.3.1 License identifier

Before deep-diving into the practical case itself, it is good to know some aspects of Solidity, the programming language used to program the DApp backend.

The first line that a smart contract should implement is the license identifier. This identifier is included in the smart contract as a commentary. It is not validated by the compiler but is included in the bytecode metadata. (GitHub Contributors B 2020.)

In the case of Solidity, the licenses are provided by SPDX (Software Package Data Exchange), an international open standard, so it allows to trust in smart contract code. (GitHub Contributors B 2020; SPDX A.)

The following image (Figure 9) represents a typical license identifier widely used on smart contracts.



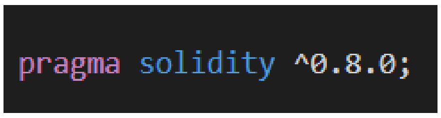
```
// SPDX-License-Identifier: MIT
```

Figure 9. Smart contract license

Notice that the license identifier is MIT. This kind of license indicates that the smart contract is open source. Then whoever is free to do whatever with it, like distributing, selling copies, copying, or modifying it. Another kind of license exists but explaining all of them is beyond the scope of this document. (SPDX B.)

6.3.2 Solidity version

In order to select the Solidity version with which a smart contract is programmed, it is necessary to add a line like the one shown in figure 10 in each smart contract:



```
pragma solidity ^0.8.0;
```

Figure 10: Smart contract version

The necessary code to select the version is divided into two different parts. The first is the keyword *pragma*, which enables compiler-specific features or checks. (GitHub Contributors B 2020.)

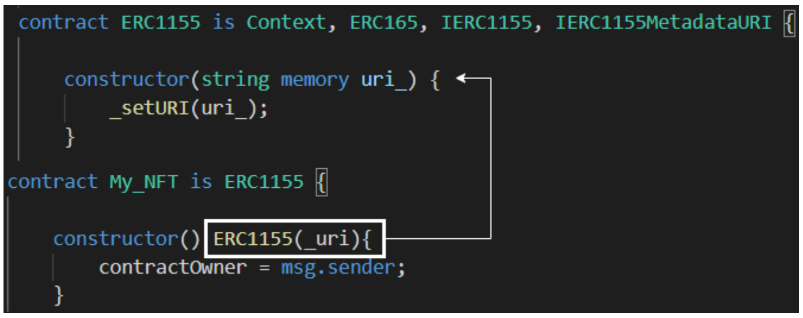
The second part is the word Solidity and the version of it. When this line is added, the code version is not the only thing selected on the smart contract but the compiler version with which the code must be compiled. If the 0.8.0 version is set only compiles with the 0.8.0 compiler, neither earlier nor future compiler versions will work (GitHub Contributors B 2020.)

6.3.3 Constructor

When a contract is created, this means deployed to the blockchain, the constructor is called, and it will not be called again. The variables can be initialized in the constructor. If the variables are not initialized in the constructor or out of it (that is also possible), they are initialized using a default value. (Solidity docs A 2021.)

Only one constructor is allowed per each smart contract. In the case of missing a constructor, Solidity assumes the default one. This is known as an *empty constructor*. Once the constructor is executed, the rest of the code is stored on the blockchain. External and public functions are deployed on the blockchain, but neither the constructor code nor internal functions aren't deployed. (Solidity docs A 2021.)

In the case of the current smart contract being a child of another smart contract, the constructor of the parent smart contract must be called from the child smart contract with the arguments needed. The parent constructor is called after the keyword *constructor* at the definition of the current contract. (Solidity docs A 2021.)



```
contract ERC1155 is Context, ERC165, IERC1155, IERC1155MetadataURI {

    constructor(string memory uri_) {
        _setURI(uri_);
    }
contract My_NFT is ERC1155 {

    constructor() ERC1155(_uri){
        contractOwner = msg.sender;
    }
```

Figure 11. My_NFT contract: Constructor.

On the top of figure 11 is shown the ERC1155 contract definition and the constructor. Notice that contract ERC1155 is a child of more than one contract, but none of them is called in the ERC1155 constructor. It is because any ERC115 parents have a constructor. At the bottom of the image is defined the contract My_NFT, which is the child of ECR1155. Because ERC1155 has its own constructor, this must be called in the My_NFT constructor and pass the same arguments. The argument is the collection URI, the URL where the NFT metadata is stored. In addition, in the MY_NFT constructor, the variable *contractOwner* is initialized. When the contract is deployed and the constructor executed, the deployer's address is stored in the *contractOwner* variable. Then the contract owner is who deploys the contract.

6.3.4 Functions

Smart contracts are very similar to classes in object-oriented programming languages. This means that the variables state and machine state are modified through functions. If the user wants to execute some action using a smart contract, the function that carries out this action needs to be programmed and called. (Solidity docs A 2021.)

Defining a function can be challenging due to the wide variety of keywords added to its definition. The next image shows how a function has to be defined in Solidity.

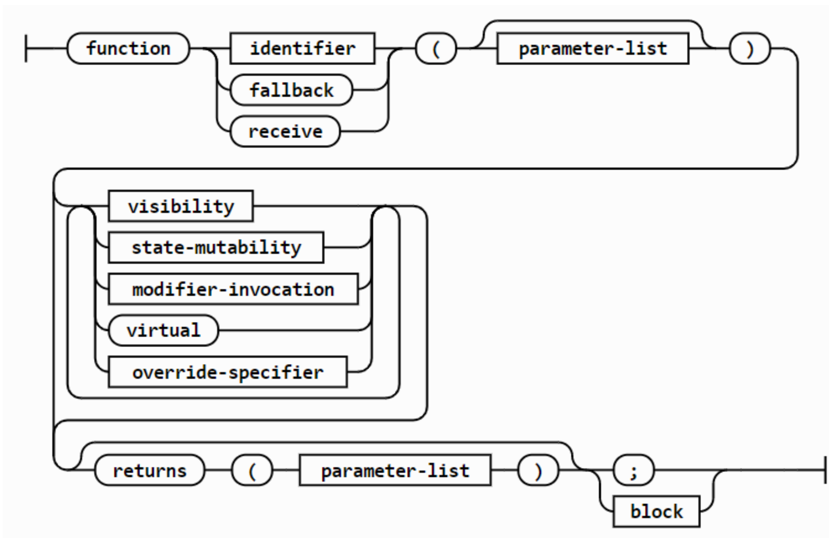


Figure 12. Grammar definition for Solidity functions. (Solidity docs B.)

As shown in the Solidity grammar definition (Figure 12), a function needs to be defined by introducing the keyword *function* at its beginning. Followed by either an identifier, commonly a given function name selected by the developer, or *fallback*/*receive* keywords. In the second case, Solidity tags these functions as special functions. Neither *fallback* nor *receive* functions need to be preceded with the *function* keyword. (Solidity docs A 2021.)

The characteristics of these special functions are:

- Fallback: This function is executed either when the given function identifier does not match other functions or when neither no arguments are provided with the function call nor receive Ether function is programmed. This function always receives data, but it must be marked as *payable* if it receives Ether. A contract contains at most one fallback function and must have external visibility. Also, it can be virtual, can override a function in the parent contract, and have modifiers. (Solidity docs A 2021.)
- Receive: A receive function is executed when the contract is called with empty calldata. At most, one *receive* function can be declared per contract. Furthermore, this function does not allow to have arguments and cannot return anything. This function must be external and payable, can override a function in the parent contract, and have modifiers if required. (Solidity docs A 2021.)

Once the function name is declared, or the function itself is defined as a special function, the parameters list must be specified if needed. Parameters act like variables and can only be used inside the function scope, meaning parameters work as function local variables. As variables, the parameter type needs to be known when defined. Parameters are optional so no parameter list would be specified. (Solidity docs A 2021.)

After specifying the function parameters, the visibility of the function must be defined. Solidity programming language makes it possible to select among four different visibility modifiers in order to adjust the function scope. These four visibility specifiers are:

- External: These functions can be called from other contracts and via transactions but cannot be called internally from the contract where they are programmed. (Solidity docs A 2021.)

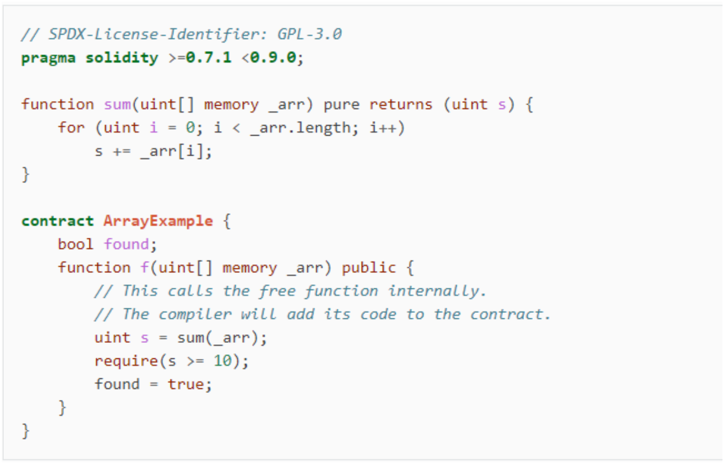
- Internal: An internal function can be called either from the current contract (where the method is programmed) or from children's contracts. These functions can't be called externally, either in other contracts or the front-end. (Solidity docs A 2021.)

- Public: These types of functions can be accessed from the current contract or via message calls, or what it is the same, from the front-end. (Solidity docs A 2021.)

- Private: Like an internal function, but inaccessible from derived contracts. (Solidity docs A 2021.)

The above visibility modifiers define the function scope from which a function can be called but need to be specified only when the function is defined inside the contract. This is because Solidity accepts defining functions outside of a contract. They are called *free functions*. The visibility of these kinds of functions is internal. Therefore, it only can be reached from the current contract, and the function context is the same as the contract context. Access directly to variables and functions outside the scope of free functions is not possible. (Solidity docs A 2021.)

The following figure shows how free functions are used and programmed.



```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.1 <0.9.0;

function sum(uint[] memory _arr) pure returns (uint s) {
    for (uint i = 0; i < _arr.length; i++)
        s += _arr[i];
}

contract ArrayExample {
    bool found;
    function f(uint[] memory _arr) public {
        // This calls the free function internally.
        // The compiler will add its code to the contract.
        uint s = sum(_arr);
        require(s >= 10);
        found = true;
    }
}
```

Figure 13. Free functions in a smart contract. (Solidity docs A 2021.)

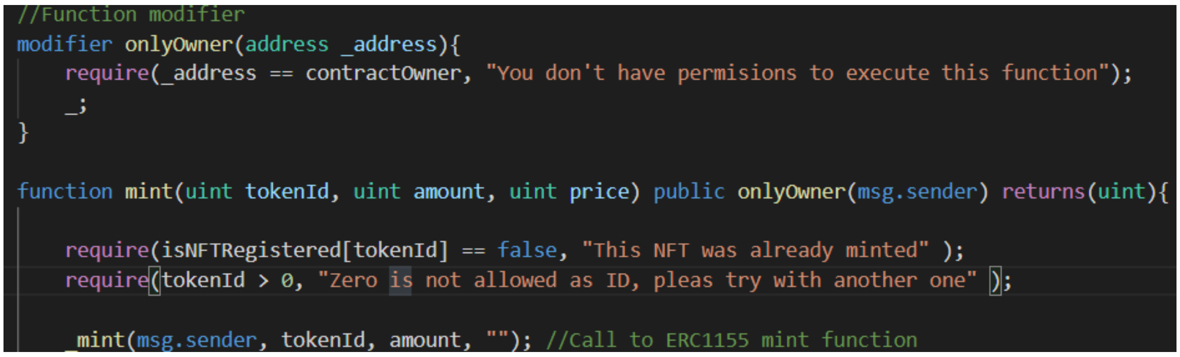
As can be seen in the previous figure (Figure 13), the first two lines define the license and the program version. In this case, the selected license is GPL-3.0, and the compiler versions are those between 0.7.1 and 0.9.0. This last version is not included. Next, a free function is programmed, so no visualization specifier is added because these functions are internal intrinsically. After function definition, the contract itself is programmed. In this case, the contract is not a derivation from another contract and doesn't contain a constructor. Just a function to make use of the free function is being programmed.

In the next step, the function state mutability is defined. That is how Solidity specifies if a function will be able, or not, to modify the blockchain state through variables or transactions. These three different types of states are the next ones. (Solidity docs A 2021.)

- Pure: These type of functions only makes use of local variables. So no state variables are neither modified nor read (Solidity docs A 2021.)
- View: These are functions where just a value is returned without modifying state variables. It could seem like pure functions, but the view functions are able to read values stored on state variables. (Solidity docs A 2021.)
- Payable: In the methods where it is necessary to allow the contract to receive ethers, the keyword *payable* must appear as a visibility modifier. Otherwise, no ether transactions are allowed. This kind of function can carry out modifications in state variables. (Solidity by Example B.)

In the case of no mutability specified, Solidity marks this function as non-payable, and the function will be able to modify and read state and local variables. (Solidity docs B.)

Once the state mutability is defined, it is possible to add *function modifiers*. These modifiers are a very useful tool that allows controlling the function flow but without the necessity of executing the function itself. Modifiers are similar to adding an external layer that must be overcome to execute the method. Furthermore, parameters are accepted in modifiers, but the modifier itself cannot access parameters used in functions it modifies. It is possible to use more than one modifier in a function. Another important characteristic is that only modifiers programmed in the current contract or those belonging to the derived contract can be accessed. (Solidity docs A 2021.)



```
//Function modifier
modifier onlyOwner(address _address){
    require(_address == contractOwner, "You don't have permisions to execute this function");
    _;
}

function mint(uint tokenId, uint amount, uint price) public onlyOwner(msg.sender) returns(uint){

    require(isNFTRegistered[tokenId] == false, "This NFT was already minted" );
    require(tokenId > 0, "Zero is not allowed as ID, pleas try with another one" );

     _mint(msg.sender, tokenId, amount, ""); //Call to ERC1155 mint function
```

Figure 14. Modifier example.

At the top of figure 14, the modifier declaration is presented. A requirement condition is programmed inside it where the owner's address is passed as a parameter. After the modifier definition, a *mint* function that uses the previously defined modifier is programmed. Each time this function is called, the modifier filter acts and checks if the account from where the function is being called is the same as the contract owner's address. If not, an exception is thrown up, and the function is not executed.

Once the modifier has been added or not, it is time to specify whether the function will be overridden. When the *override* keyword appears in the function, it means that this function is being declared and defined in the parent contract, but in the current contract, it is necessary to re-define this function to adapt it to the required purposes. (Modi 2018.)

Because Solidity allows polymorphism, this means declaring two or more functions with the same name. It is necessary to specify if the current function could be "overridden" in future children contracts. Also, it is necessary to specify if the function is currently "overriding" a function at the parent contract. The keyword *virtual* indicates that the current function could be replaced in a child contract. On the other hand, the *override* keyword indicates that the current function replaces a function in the parent contract. These keywords are non-exclusive between them. (Solidity by Example C; Solidity docs A 2021.)

And now, to finalize the function declaration, it can be necessary to specify the type of the returned value. If the function does not return anything, this part can be omitted. It is possible to return more than one value. In that case, the variable types must be specified. The return variable identifier can be omitted in the parameter list, but the variable's type must always appear. (Solidity docs A 2021.)

6.3.5 Variables.

Solidity allows storing values in different kinds of variables to make the management of some functions easier. Three main variable types exist in Solidity: (Solidity by Example D.)

- Local: Those variables are declared in the context of a function. These kinds of variables are not stored in the blockchain. (Solidity by Example D.)
- State: Variables declared in a contract context. These types of variables are stored in the blockchain. (Solidity by Example D.)
- Global: Special variables that provide information about the blockchain. An example of these variables could be the ethers quantity sent in a transaction or who is calling a function. (Solidity by Example D; Solidity docs A 2021.)

State variables are declared by the programmer, so the visibility of the variable must be specified. In addition, an immutable state can be declared if needed.

Solidity accepts three types of visibility:

- Internal: Variables are accessible from the current and child contracts but not externally, as the front-end. By default, the variables are declared in this visibility. (Solidity docs A 2021.)
- Public: Very similar to internal variables, but the compiler automatically generates getter functions to allow read their values from other contracts. Because setters are not generated automatically, other contracts will not be able to modify the value of these variables, but these setters can be programmed explicitly by the programmer. (Solidity docs A 2021.)
- Private: Variables can be accessed in the current contract. Children's contracts can't access the value stored in these variables. Public getters and setters can be programmed in order to access the variables from other contracts. (Solidity docs A 2021.)

It's possible to declare immutable variables. Solidity uses two keywords to specify it:

- Constant: Variables in which the value stored on it is fixed at compile-time, then once the contract is constructed, the value can't be modified. These variables are initialized in the constructor. (Solidity docs A 2021.)
- Immutable: Like in constant variables, the value stored on an immutable variable can't be modified after contract construction. But in this case, the variable value is

defined at construction time, which means assigning a variable value explicitly when it is declared outside the constructor. (Solidity docs A 2021.)

To allocate the different variables used in a smart contract, depending on the variable type (local, state, or global), Solidity stores the variable value in one of the following places. (Desai 2019.)

- Storage: State variables are allocated in the storage location, similar to a hard disk drive. Variables kept in storage are permanent. (Desai 2019 A.)
- Memory: This location stores temporary variables, similar to RAM, which is the best option to treat local variables. (Desai 2019 A.)
- Calldata: The best option to allocate parameters is the calldata location. It Is non-modifiable and non-persistent data location. (Desai 2019 A.)

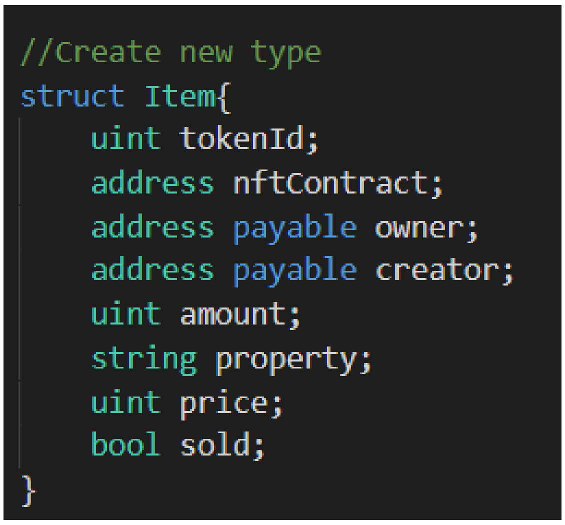
It is unnecessary to specify where the variables will be allocated during the smart contract execution. Depending on the context of the variable, the storage class is assigned automatically. (Desai 2019 A.)

6.3.6 Solidity data structures (Reference Types)

Solidity allows different kinds of data structures. Following is a list of available options:

- Arrays: Solidity also implements this well-known structure used in several programming languages like Python or Java. Thanks to this structure, several values with the same data type can be stored in the same variable and accessed by their position inside this structure. Solidity allows using fixed-size arrays or dynamic size arrays. (Desai 2019 B.)
- Mappings: This structure is similar to Python dictionaries because mappings are key-value structures. This reference type stores two variables, the first one is the key, and the second one is the value linked to the key. To retrieve a value key must be known. If known, this value can be efficiently accessed because mappings work like a hash table where values are accessed not by position like arrays but by hash, which is more efficient and straightforward. (Solidity docs C.)
- Structs: Thanks to structs, programmers can create their own data types. Struct is composed of different data types, including reference types. (Solidity docs C).

The following image is an example of a struct used in the practical case:



```
//Create new type
struct Item{
    uint tokenId;
    address nftContract;
    address payable owner;
    address payable creator;
    uint amount;
    string property;
    uint price;
    bool sold;
}
```

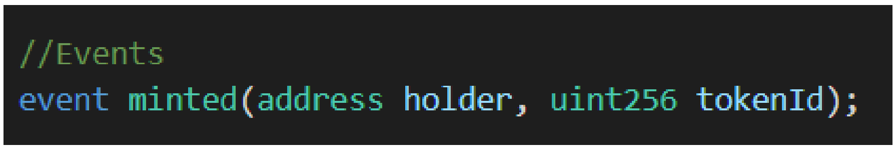
Figure 15. My_NFT smart contract: Item struct.

Figure 15 shows that the definition starts with the struct keyword, followed by the identifier name. Furthermore, two payable address variable types are being used in this struct. These variables store a wallet address (the public key). The payable keyword determines that these addresses can receive money in their wallets.

6.3.7 Events

Events are used to register transaction information in blockchain logs. Logs are associated with contract addresses and are accessible till the contract is on the blockchain. Events are inheritable so that they can be called from derived contracts. (Solidity docs A 2021)

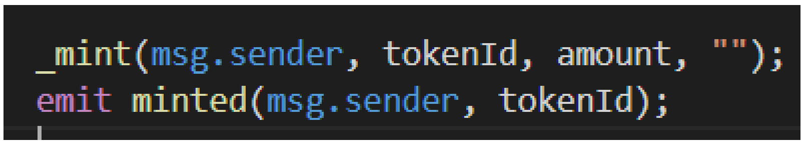
Creating an event is an easy task. It is necessary to specify only the keyword *event,* followed by the event identification name, and after it, declare the arguments. The next image (Figure 16) is an example of an event declaration:



```
//Events
event minted(address holder, uint256 tokenId);
```

Figure 16. Event declaration

The event programmed before is used in the mint function, as shown in figure 17. The information registered in the blockchain would be who mints the item (msg.sender) and the identifier of the minted token.



```
_mint(msg.sender, tokenId, amount, "");
emit minted(msg.sender, tokenId);
```

Figure 17. Emitting an event.

6.4 Practical case elements

6.4.1 ERC Standards

ERC (Ethereum Request for Comments) are documents that establish the rules required to develop a token being either cryptocurrencies or NFTs. (EthHub.)

Currently, Ethereum has three primary ERC standards:

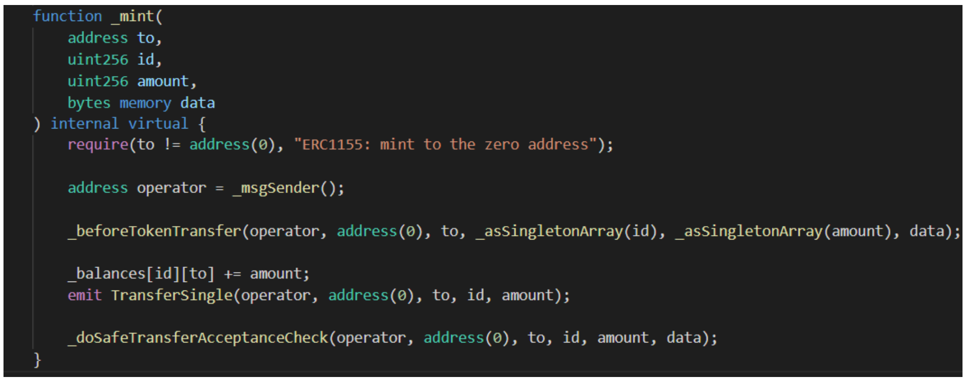
- ERC-20: Standard used for creating cryptocurrencies. (Joshua 2021 B.)
- ERC-721: Standard that allows creating a collection of unique NFTs. (Joshua 2021 B.)
- ERC-1155: Standard with which both cryptocurrencies and NFTs can be created, but creating non-unique NFTs is also possible. So it is possible to create more than one NFT with the same identification number, therefore containing the same data. This standard is the chosen one in this case. (Joshua 2021 B.)

These three standards are Solidity contracts that contain the functions to manage the cryptocurrencies or NFTs. Therefore, depending on the nature of the project, one type of standard will be more suitable than the other.

In order to program a smart contract that creates a new NFT collection or manages NFTs or cryptocurrencies in some way, the standard that fits better with the final purpose must be imported. In addition, it is necessary to specify that the new smart contract is a child of the standard imported. Establishing a relationship between own smart contract and an officially recognized standard is the only way to proceed. In this practical case, the ERC-1155 has been used as a parent of the smart contract created.

Due to this standard containing all the functions to manage information, these functions don’t have to be programmed twice. Hence, the child contract uniquely calls these functions and creates other ones to manage its own information. The following functions from the ERC-1155 will be used in the child smart contract:

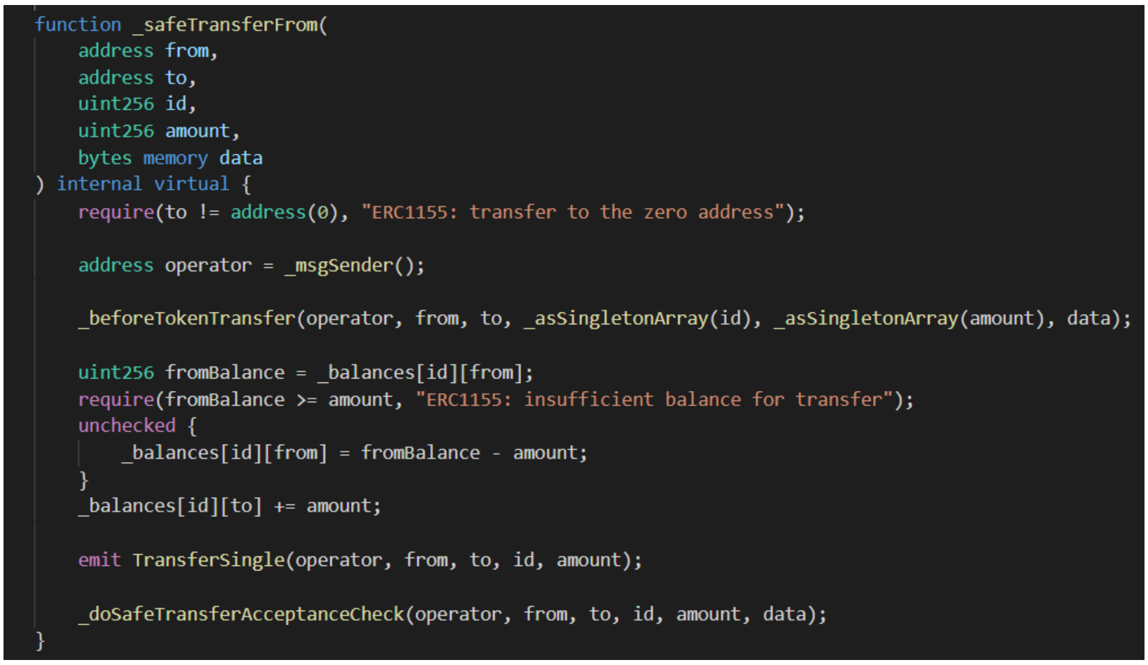
- _mint: The minting process is carried out in this function. This means registering in the *_balances* mapping that the account which mints the token now possesses a concrete amount of them. Once the mapping is updated, the corresponding event is emitted. This function is shown in figure 18.



```
function _mint(
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) internal virtual {
    require(to != address(0), "ERC1155: mint to the zero address");

    address operator = _msgSender();

    _beforeTokenTransfer(operator, address(0), to, _asSingletonArray(id), _asSingletonArray(amount), data);

    _balances[id][to] += amount;
    emit TransferSingle(operator, address(0), to, id, amount);

    _doSafeTransferAcceptanceCheck(operator, address(0), to, id, amount, data);
}
```

Figure 18. ERC-1155: _mint function.

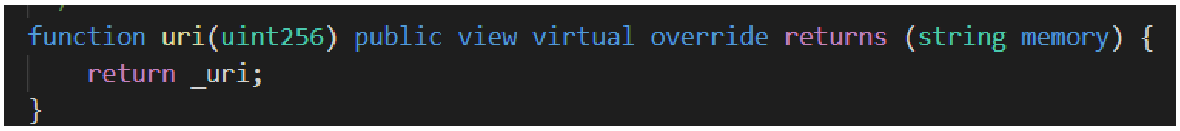
- _ safeTransferFrom: This function's main task is to transfer items from one account to another. The items from the *origin (from account)* are subtracted, then added to the destination (to account), using *_balances* mapping. Then, the event which informs of this transfer is triggered. This function is shown in figure 19.



```
function _safeTransferFrom(
    address from,
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) internal virtual {
    require(to != address(0), "ERC1155: transfer to the zero address");

    address operator = _msgSender();

    _beforeTokenTransfer(operator, from, to, _asSingletonArray(id), _asSingletonArray(amount), data);

    uint256 fromBalance = _balances[id][from];
    require(fromBalance >= amount, "ERC1155: insufficient balance for transfer");
    unchecked {
        _balances[id][from] = fromBalance - amount;
    }
    _balances[id][to] += amount;

    emit TransferSingle(operator, from, to, id, amount);

    _doSafeTransferAcceptanceCheck(operator, from, to, id, amount, data);
}
```

Figure 19. ERC-1155: _safeTransferFrom function.

- _uri: This function is responsible for returning the URI where the NFT metadata are allocated. It is not a function called from the child smart contract but is “overriding” the function in the parent smart contract because it is necessary to use different metadata for each NFT. This function is shown in figure 20.



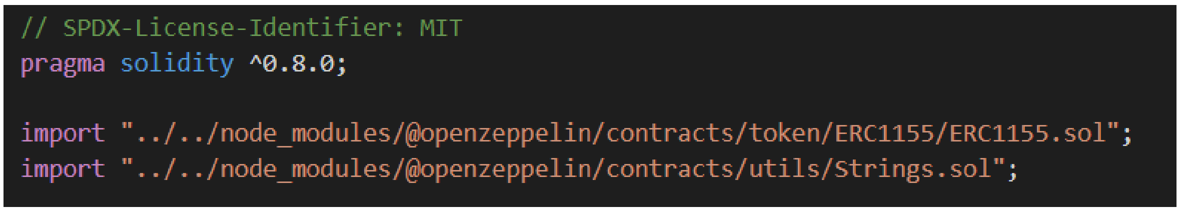
```
function uri(uint256) public view virtual override returns (string memory) {
    return _uri;
}
```

Figure 20. ERC-1155: URI function.

6.4.2 Smart contract

In order to develop a marketplace that controls the NFT flow, such as creating, buying, selling, removing, and displaying the NFTs purchased from an account, it is necessary to program a smart contract that manages all of this information.

When a Solidity smart contract is being programmed, the first thing to do is select the license and the version. Then, before starting with the smart contract itself, it is necessary to import the required standards. Figure 21 shows how to do it.



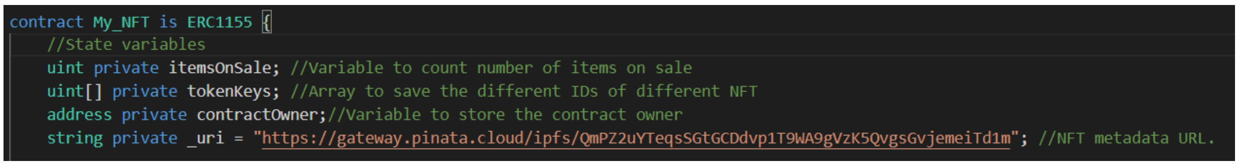
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "../../node_modules/@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
import "../../node_modules/@openzeppelin/contracts/utils/Strings.sol";
```

Figure 21. My_NFT smart contract: license, version, and imports.

The next step is initializing the contract itself, if and when the contract does not require free functions that must be programmed before initializing it.

If the contract is a child of another smart contract, this must be specified using the keyword *is.* If not, only the *contract* keyword and the contract name are necessary.

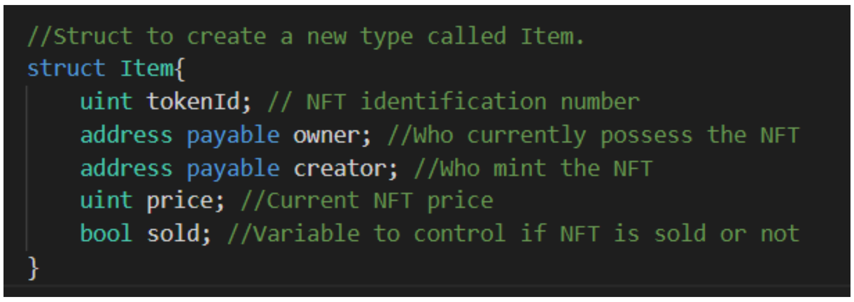
Once the smart contract is initialized, an excellent way to start a smart contract is by defining the state variables on top. Initializing the variables here is not a rule or a convention. It is only a way to proceed to avoid variables being scattered throughout the smart contract. Figure 22 shows a way to do it.



```
contract My_NFT is ERC1155 {
    //State variables
    uint private itemsOnSale; //Variable to count number of items on sale
    uint[] private tokenKeys; //Array to save the different IDs of different NFT
    address private contractOwner;//Variable to store the contract owner
    string private _uri = "https://gateway.pinata.cloud/ipfs/QmPZ2uYTeqsSGtGCDdvp1T9WA9gVzK5QvgsGvjemeiTd1m"; //NFT metadata URL.
```

Figure 22. My_NFT smart contract: declaration and state variables.

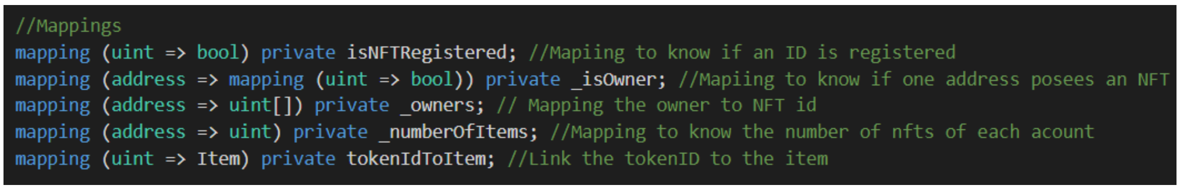
Creating a struct containing all this information is practical to keep the NFT information organized. This struct makes it easier to control the NFT state. Figure 23 shows how to create a struct.



```
//Struct to create a new type called Item.
struct Item{
    uint tokenId; // NFT identification number
    address payable owner; //Who currently possess the NFT
    address payable creator; //Who mint the NFT
    uint price; //Current NFT price
    bool sold; //Variable to control if NFT is sold or not
}
```

Figure 23. My_NFT smart contract Item type declaration.

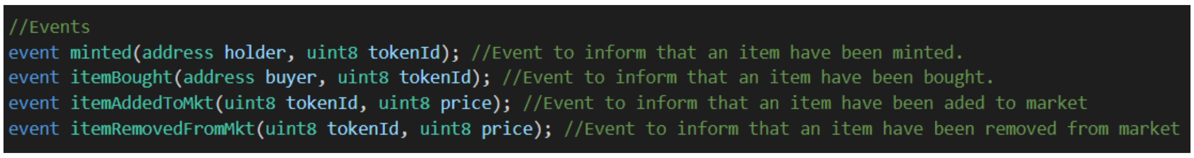
After declaring the state variables and following a pattern to maintain the smart contract structure, a suitable way to continue is to declare the necessary mappings, like in figure 24.



```
//Mappings
mapping (uint => bool) private isNFTRegistered; //Mapiing to know if an ID is registered
mapping (address => mapping (uint => bool)) private _isOwner; //Mapiing to know if one address posees an NFT
mapping (address => uint[]) private _owners; // Mapping the owner to NFT id
mapping (address => uint) private _numberOfItems; //Mapping to know the number of nfts of each acount
mapping (uint => Item) private tokenIdToItem; //Link the tokenID to the item
```

Figure 24. My_NFT smart contract mappings.

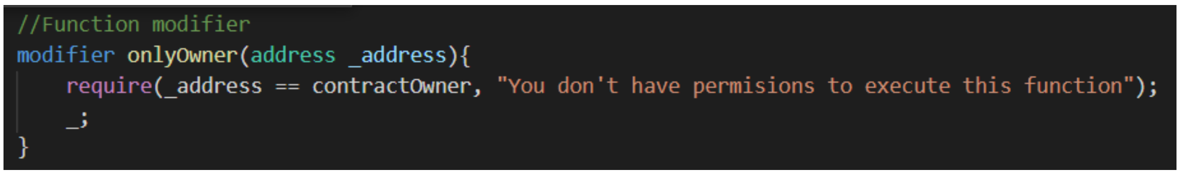
There is no more information to store. The next step is to create the necessary events to ensure that the blockchain log stores the relevant information. It is not a good idea to use more than the essential events because they increment the complexity of the functions, so emitting an event increment the function execution costs. Figure 25 shows the events used in this DApp.



```
//Events
event minted(address holder, uint8 tokenId); //Event to inform that an item have been minted.
event itemBought(address buyer, uint8 tokenId); //Event to inform that an item have been bought.
event itemAddedToMkt(uint8 tokenId, uint8 price); //Event to inform that an item have been aded to market
event itemRemovedFromMkt(uint8 tokenId, uint8 price); //Event to inform that an item have been removed from market
```

Figure 25. My_NFT smart contract events.

After declaring the events, the required function modifiers are programmed. In this case, only one modifier is necessary, but it is possible to add more than one. This modifier checks that the account with which the function is being executed is the owner account. If it is not being executed by the contract owner, an error is thrown. Then the text typed inside the *require* condition will appear in the transaction data. In a production environment, web pages like etherscan are helpful for watching this information. In etherscan, a transaction can be accessed even if it fails, and the failure information can be easily recognized. The unique programmed modifier is shown in figure 26.

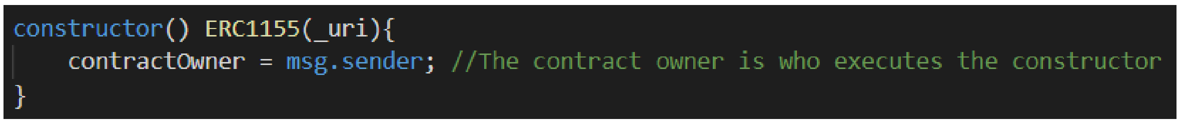


```
//Function modifier
modifier onlyOwner(address _address){
    require(_address == contractOwner, "You don't have permisions to execute this function");
    _;
}
```

Figure 26. My_NFT smart contract function modifier.

Once all the state variables, events, modifiers, etcetera are implemented, it is time to proceed with the constructor and the functions.

The constructor is formed by two parts: the constructor itself and the call to the parent contract constructor with the required argument. In this constructor, only the *contractOwner* variable is defined. This is shown in figure 27.

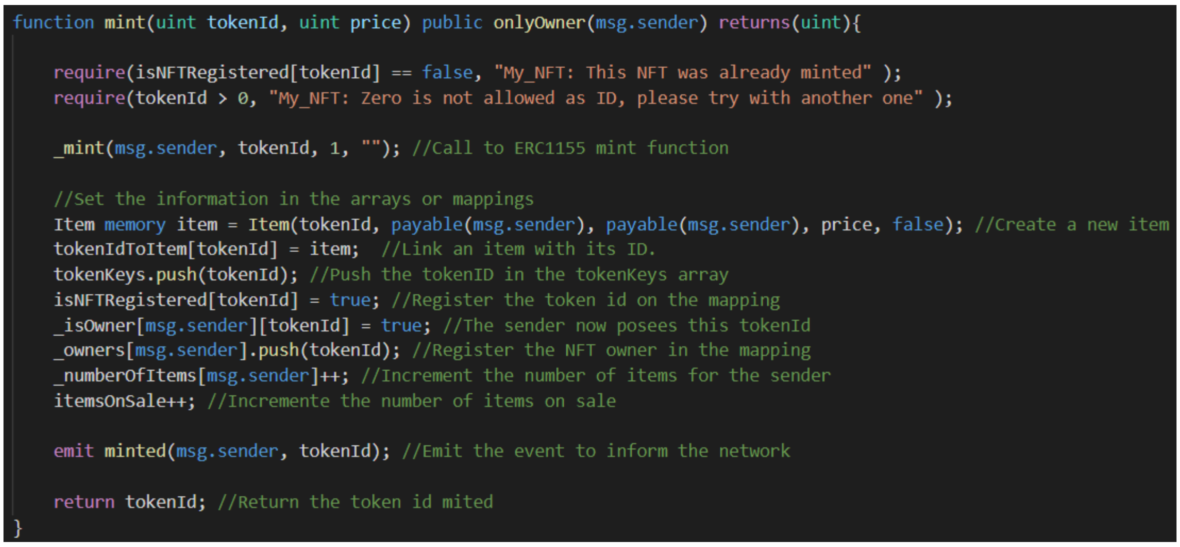


```
constructor() ERC1155(_uri){
    contractOwner = msg.sender; //The contract owner is who executes the constructor
}
```

Figure 27. My_NFT smart contract constructor.

After the construction definition, the core part of the smart contract must be programmed. This part is the last one, and it is composed of all the functions that the contract is able to carry out.

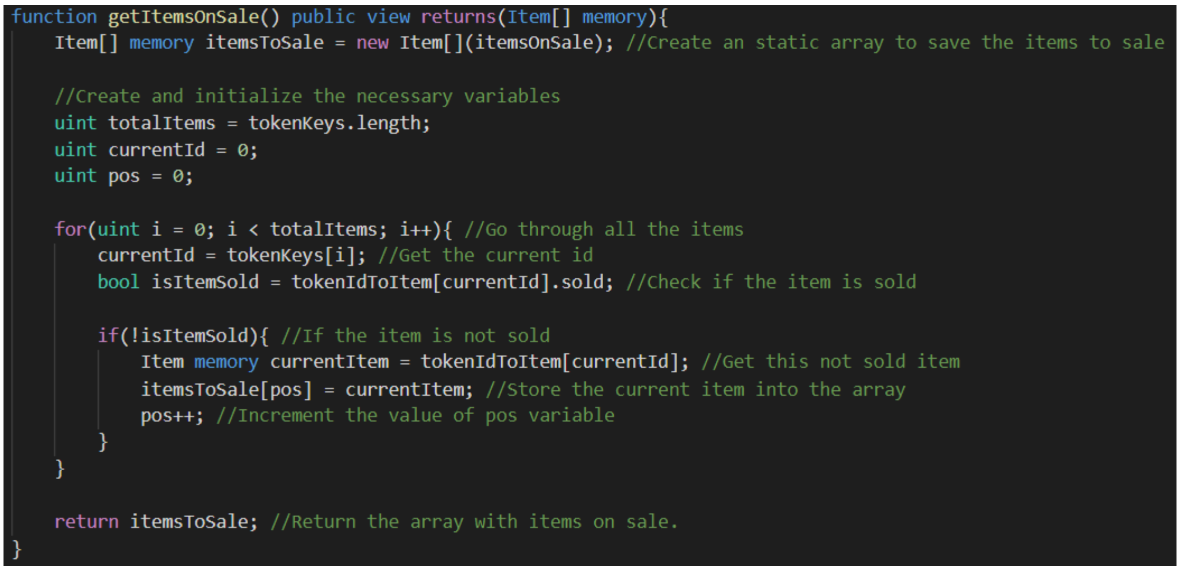
The first one is the function to mint an NFT. But before starting to mint and store the information in the variables, it is necessary to establish certain security layers. The first one is the function modifier which avoids executing this function by anyone other than the owner. If the owner is minting, two different security layers must be passed. The first one is to avoid minting an existent NFT, and the second one is to avoid minting an item identified with a negative number or zero. If these three security checks are passed, the token is minted, and the different variables, mappings, and arrays are updated. This entire process is shown in figure 28.



```
function mint(uint tokenId, uint price) public onlyOwner(msg.sender) returns(uint){

    require(isNFTRegistered[tokenId] == false, "My_NFT: This NFT was already minted" );
    require(tokenId > 0, "My_NFT: Zero is not allowed as ID, please try with another one" );

    _mint(msg.sender, tokenId, 1, ""); //Call to ERC1155 mint function

    //Set the information in the arrays or mappings
    Item memory item = Item(tokenId, payable(msg.sender), payable(msg.sender), price, false); //Create a new item
    tokenIdToItem[tokenId] = item;  //Link an item with its ID.
    tokenKeys.push(tokenId); //Push the tokenID in the tokenKeys array
    isNFTRegistered[tokenId] = true; //Register the token id on the mapping
    _isOwner[msg.sender][tokenId] = true; //The sender now posees this tokenId
    _owners[msg.sender].push(tokenId); //Register the NFT owner in the mapping
    _numberOfItems[msg.sender]++; //Increment the number of items for the sender
    itemsOnSale++; //Incremente the number of items on sale

    emit minted(msg.sender, tokenId); //Emit the event to inform the network

    return tokenId; //Return the token id mited
}
```

Figure 28. My_NFT smart contract *mint* function.

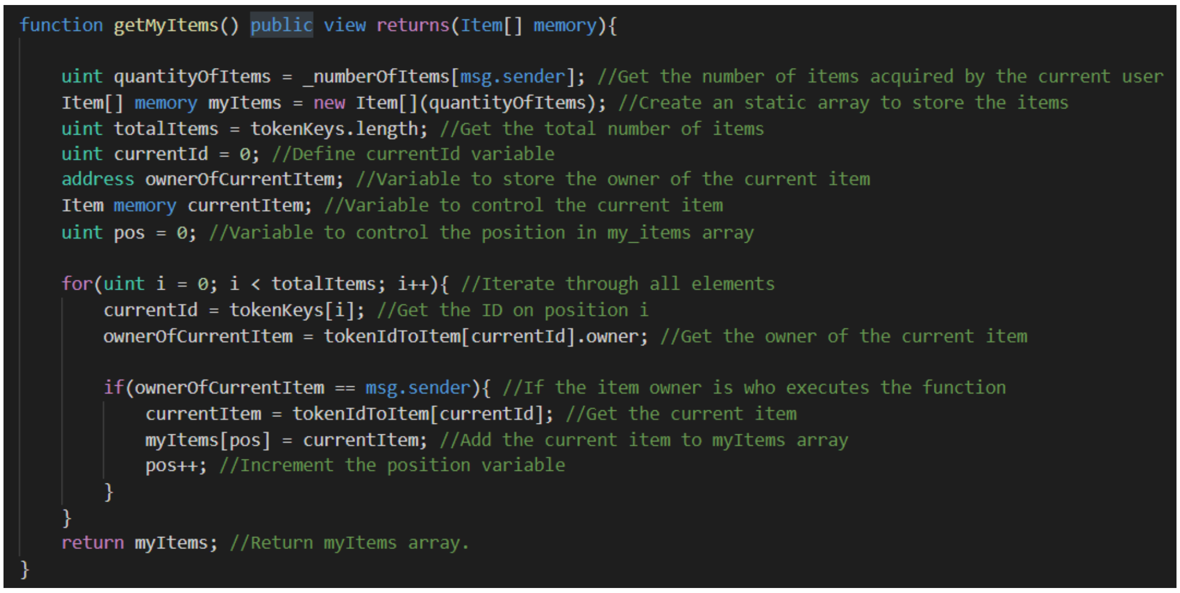
In order to be able to display all items on sale at the market, it is essential to program a function that gets all these items and returns them. This function must be public and must not modify any state variable. Then, it can be specified as a view function. It can not be a pure function because it needs to get state variables values to work. Note that the array is stored in memory because it is only accessed by this function, and it is not necessary to save it into storage. This function is shown in figure 29.



```
function getItemsOnSale() public view returns(Item[] memory){
    Item[] memory itemsToSale = new Item[](itemsOnSale); //Create an static array to save the items to sale

    //Create and initialize the necessary variables
    uint totalItems = tokenKeys.length;
    uint currentId = 0;
    uint pos = 0;

    for(uint i = 0; i < totalItems; i++){ //Go through all the items
        currentId = tokenKeys[i]; //Get the current id
        bool isItemSold = tokenIdToItem[currentId].sold; //Check if the item is sold

        if(!isItemSold){ //If the item is not sold
            Item memory currentItem = tokenIdToItem[currentId]; //Get this not sold item
            itemsToSale[pos] = currentItem; //Store the current item into the array
            pos++; //Increment the value of pos variable
        }
    }

    return itemsToSale; //Return the array with items on sale.
}
```

Figure 29. My_NFT smart contract *getItemsOnSale* function.

A very similar function, shown in figure 30, is needed to get the items bought by the current user.

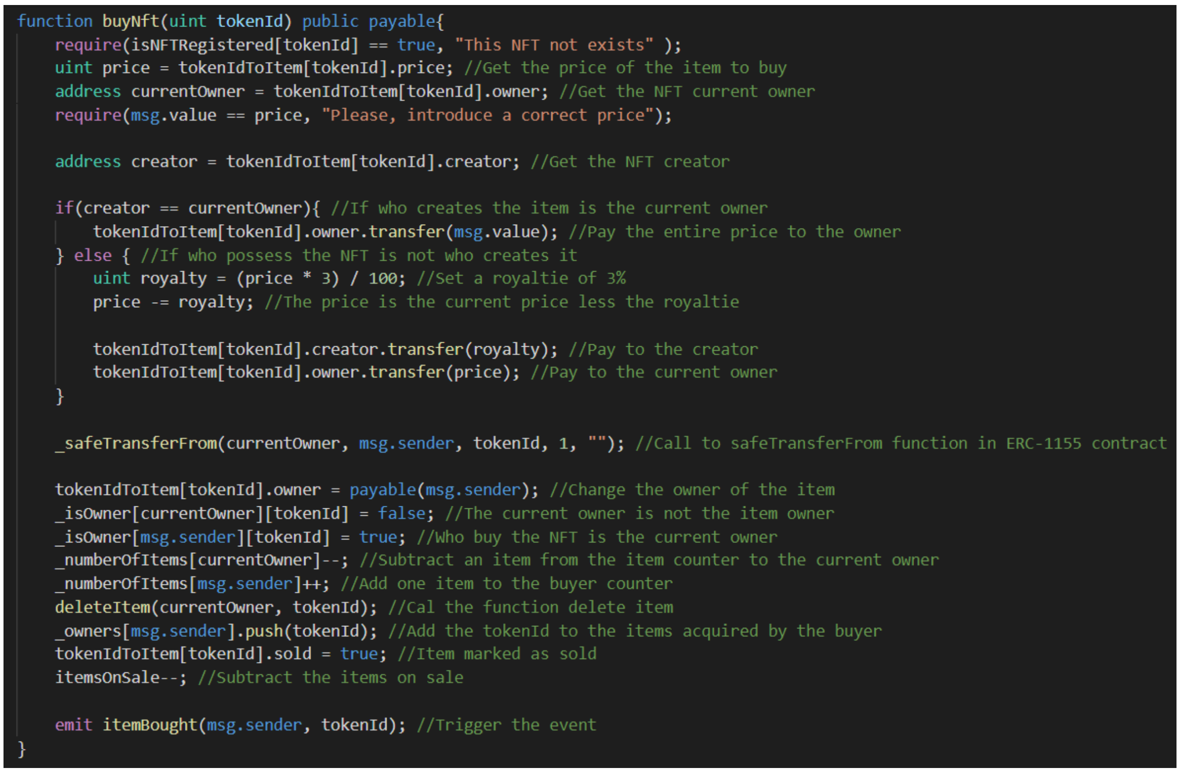


```
function getMyItems() public view returns(Item[] memory){

    uint quantityOfItems = _numberOfItems[msg.sender]; //Get the number of items acquired by the current user
    Item[] memory myItems = new Item[](quantityOfItems); //Create an static array to store the items
    uint totalItems = tokenKeys.length; //Get the total number of items
    uint currentId = 0; //Define currentId variable
    address ownerOfCurrentItem; //Variable to store the owner of the current item
    Item memory currentItem; //Variable to control the current item
    uint pos = 0; //Variable to control the position in my_items array

    for(uint i = 0; i < totalItems; i++){ //Iterate through all elements
        currentId = tokenKeys[i]; //Get the ID on position i
        ownerOfCurrentItem = tokenIdToItem[currentId].owner; //Get the owner of the current item

        if(ownerOfCurrentItem == msg.sender){ //If the item owner is who executes the function
            currentItem = tokenIdToItem[currentId]; //Get the current item
            myItems[pos] = currentItem; //Add the current item to myItems array
            pos++; //Increment the position variable
        }
    }
    return myItems; //Return myItems array.
}
```

Figure 30. My_NFT smart contract *getMyItems* function.

Another important function to talk about is the function that allows buying of items. It must be marked as payable because some money transactions are carried out in this function. The function must be public as well. Otherwise, it would not be possible to buy items.

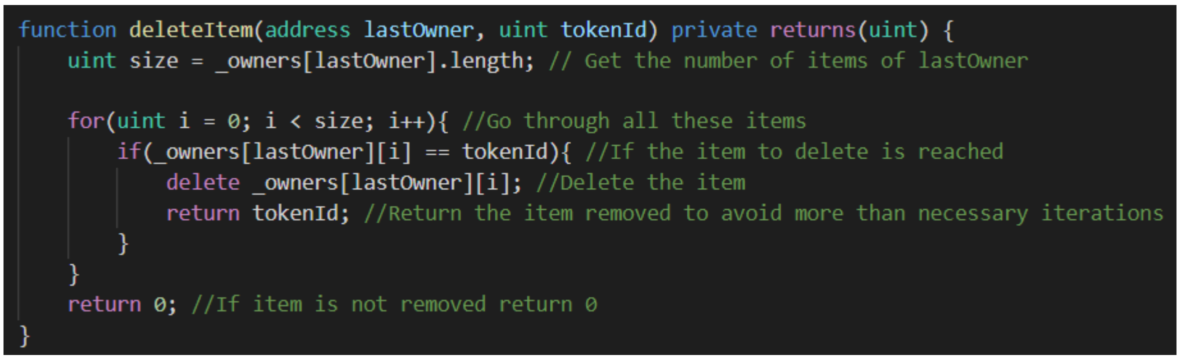
It is necessary to overcome two requires statements to execute this function. The first one is to avoid buying inexistent NFTs, and the other is to avoid paying a different price than the selling price. After checking that both *require* statements are passed, the payments are carried out, and the mapping and other variables are updated. Figure 31 shows this entire process.



```
function buyNft(uint tokenId) public payable{
    require(isNFTRegistered[tokenId] == true, "This NFT not exists" );
    uint price = tokenIdToItem[tokenId].price; //Get the price of the item to buy
    address currentOwner = tokenIdToItem[tokenId].owner; //Get the NFT current owner
    require(msg.value == price, "Please, introduce a correct price");

    address creator = tokenIdToItem[tokenId].creator; //Get the NFT creator

    if(creator == currentOwner){ //If who creates the item is the current owner
        tokenIdToItem[tokenId].owner.transfer(msg.value); //Pay the entire price to the owner
    } else { //If who possess the NFT is not who creates it
        uint royalty = (price * 3) / 100; //Set a royaltie of 3%
        price -= royalty; //The price is the current price less the royaltie

        tokenIdToItem[tokenId].creator.transfer(royalty); //Pay to the creator
        tokenIdToItem[tokenId].owner.transfer(price); //Pay to the current owner
    }

    _safeTransferFrom(currentOwner, msg.sender, tokenId, 1, ""); //Call to safeTransferFrom function in ERC-1155 contract

    tokenIdToItem[tokenId].owner = payable(msg.sender); //Change the owner of the item
    _isOwner[currentOwner][tokenId] = false; //The current owner is not the item owner
    _isOwner[msg.sender][tokenId] = true; //Who buy the NFT is the current owner
    _numberOfItems[currentOwner]--; //Subtract an item from the item counter to the current owner
    _numberOfItems[msg.sender]++; //Add one item to the buyer counter
    deleteItem(currentOwner, tokenId); //Cal the function delete item
    _owners[msg.sender].push(tokenId); //Add the tokenId to the items acquired by the buyer
    tokenIdToItem[tokenId].sold = true; //Item marked as sold
    itemsOnSale--; //Subtract the items on sale

    emit itemBought(msg.sender, tokenId); //Trigger the event
}
```

Figure 31. My_NFT smart contract *buyNFT* function.

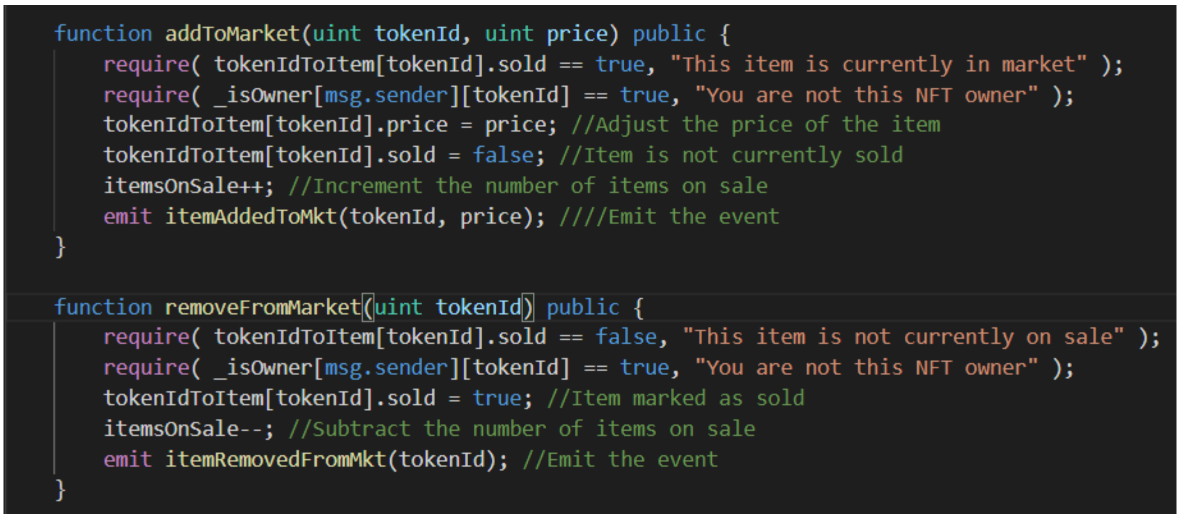
When an item is bought, meaning sold by someone, it is crucial to remove this item from the seller's items mapping. This function needs to be private. Otherwise, items could be removed by whoever. The *deleteItem* function in figure 32 accomplishes it.



```
function deleteItem(address lastOwner, uint tokenId) private returns(uint) {
    uint size = _owners[lastOwner].length; // Get the number of items of lastOwner

    for(uint i = 0; i < size; i++){ //Go through all these items
        if(_owners[lastOwner][i] == tokenId){ //If the item to delete is reached
            delete _owners[lastOwner][i]; //Delete the item
            return tokenId; //Return the item removed to avoid more than necessary iterations
        }
    }
    return 0; //If item is not removed return 0
}
```

Figure 32. My_NFT smart contract *deleteItem* function.

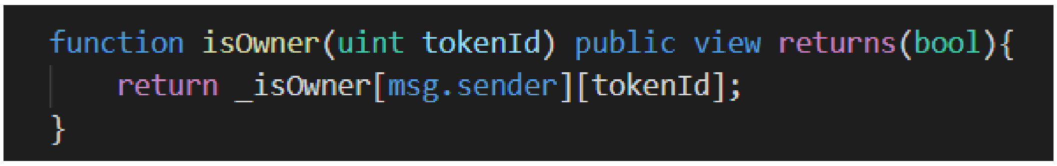
The other two important functions are the ones that allow the users to add and remove items from the market. Both functions are very similar but act oppositely and are shown in figure 33.



```
function addToMarket(uint tokenId, uint price) public {
    require( tokenIdToItem[tokenId].sold == true, "This item is currently in market" );
    require( _isOwner[msg.sender][tokenId] == true, "You are not this NFT owner" );
    tokenIdToItem[tokenId].price = price; //Adjust the price of the item
    tokenIdToItem[tokenId].sold = false; //Item is not currently sold
    itemsOnSale++; //Increment the number of items on sale
    emit itemAddedToMkt(tokenId, price); ////Emit the event
}

function removeFromMarket(uint tokenId) public {
    require( tokenIdToItem[tokenId].sold == false, "This item is not currently on sale" );
    require( _isOwner[msg.sender][tokenId] == true, "You are not this NFT owner" );
    tokenIdToItem[tokenId].sold = true; //Item marked as sold
    itemsOnSale--; //Subtract the number of items on sale
    emit itemRemovedFromMkt(tokenId); //Emit the event
}
```

Figure 33. My_NFT smart contract *addToMarket* and *removeFromMakret* functions.

The last function to discuss is to check if the current user is the owner of the item passed as a parameter or not. This function returns true or false depending on it, as shown in figure 34.



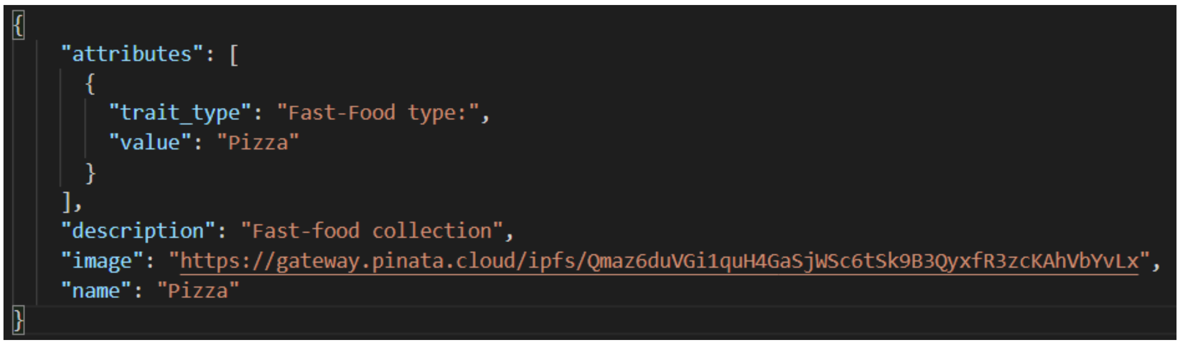
```
function isOwner(uint tokenId) public view returns(bool){
    return _isOwner[msg.sender][tokenId];
}
```

Figure 34. My_NFT smart contract *isOwner* function.

6.4.3 Metadata

To store the metadata of an NFT, an IPFS is required. NFT image, properties, name, and description are part of NFT metadata. Some NFT marketplace, like OpenSea or Rarible, uses metadata to represent all the elements about NFTs and allow the users to modify this information when required.

This metadata is arranged in a JSON file and could follow a structure similar to the next one:



```
{
    "attributes": [
      {
        "trait_type": "Fast-Food type:",
        "value": "Pizza"
      }
    ],
    "description": "Fast-food collection",
    "image": "https://gateway.pinata.cloud/ipfs/Qmaz6duVGi1quH4GaSjWSc6tSk9B3QyxfR3zcKAhVbYvLx",
    "name": "Pizza"
}
```

Figure 35. Metadata JSON file.

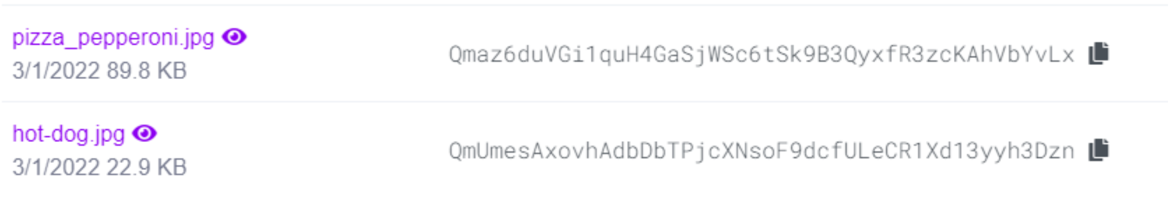
The example of a metadata file shown in figure 35 only contains one attribute, but more than one or no attribute can be specified. Furthermore, a description, an image, and a name can be defined. As can be seen in the previous figure, the NFT image is passed as a string URL. This URL belongs to Pinata's IPFS, and it is the specific link to access the image. The image must be uploaded to the IPFS, then to represent the NFT with an image, the image link is pasted in this field.

It is important to mention that, in this practical case, each NFT has its own metadata because, traditionally, each NFT in a collection is represented by a different image, and all of them have different names and attributes.

In this practical case, a collection that accepts non-unique NFTs has been created, meaning it is possible to create more than one NFT with the same ID number. Therefore, the NFTs with a certain ID are represented by the same image and contain the same metadata. Conversely, NFTs with a different id will be represented with other information because metadata files are different.

The process to create, upload and retrieve the correct metadata for each NFT is the next:

- The first step is to register an account in whichever IPFS and then upload the file by which the NFT will be represented. This file can be an image, a GIF, a 3d model, a music file, etcetera. Figure 36 shows the image files uploaded to pinata IPFS.



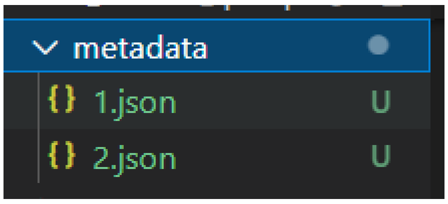
pizza_pepperoni.jpg
3/1/2022 89.8 KB
Qmaz6duVGi1quH4GaSjWSc6tSk9B3QyxfR3zcKAhVbYvLx

hot-dog.jpg
3/1/2022 22.9 KB
QmUmesAxovhAdbDbTPjcXNsoF9dcfULeCR1Xd13yyh3Dzn

Figure 36. Images uploaded to Pinata IPFS and their CID.

- Once all the images are uploaded and accessible through the URL, it is time to create the metadata file for each NFT created. The information added to the metadata file must follow a JSON structure. In this practical case, the structure proposed by OpenSea marketplace, shown in figure 35, is used, but it would be possible to use different formats.

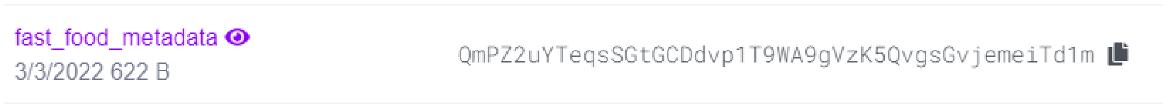
  The name of each metadata file must follow the next structure: *NFTidentificationNumber.json.* An example of it can be seen in figure 37.



```
metadata
{} 1.json   U
{} 2.json   U
```

Figure 37. Metadata file names.

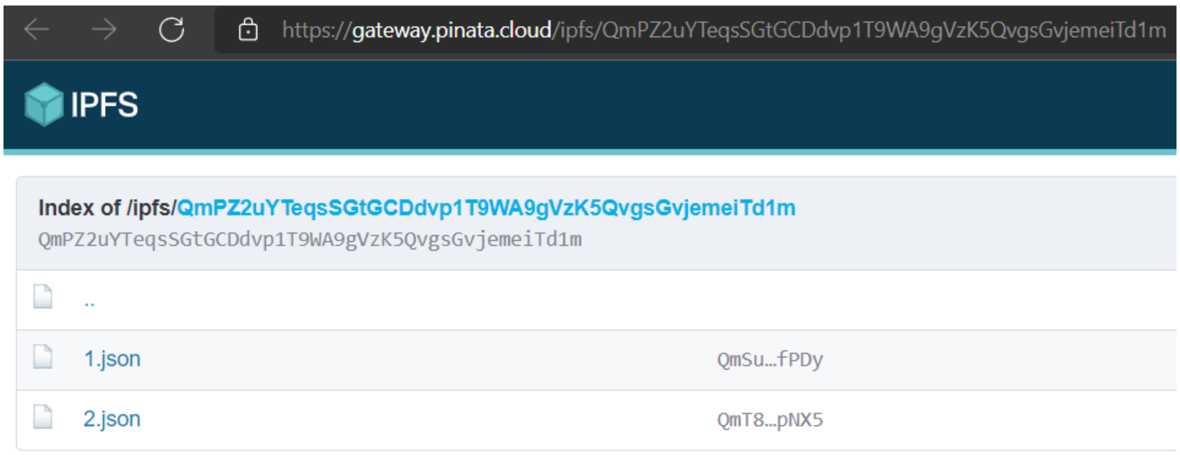
- Once the required files are created and filled, it is the moment to upload this folder to an IPFS, as figure 38 shows.



fast_food_metadata
3/3/2022 622 B
QmPZ2uYTeqsSGtGCDdvp1T9WA9gVzK5QvgsGvjemeiTd1m

Figure 38. Metadata folder uploaded to Pinata IPFS.

- After uploading the folder, its URL must be copied and pasted in the smart contract, concretely in the *_uri* state variable. The URL to copy and paste is the one that appears at the top of figure 39.



https://gateway.pinata.cloud/ipfs/QmPZ2uYTeqsSGtGCDdvp1T9WA9gVzK5QvgsGvjemeiTd1m

IPFS

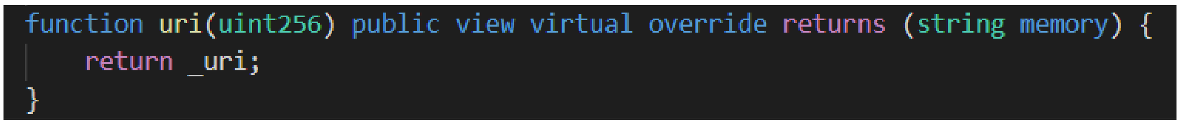
Index of /ipfs/QmPZ2uYTeqsSGtGCDdvp1T9WA9gVzK5QvgsGvjemeiTd1m
QmPZ2uYTeqsSGtGCDdvp1T9WA9gVzK5QvgsGvjemeiTd1m

..	
1.json	QmSu...fPDy
2.json	QmT8...pNX5

Figure 39. fast_food_metadata folder content.

- Once all these steps are completed, the folder metadata URL must be pasted into the corresponding state variable, called *_uri* in the MY_NFT contract. And then, a new function in this smart contract, which gets the content of these files for each NFT uploaded, must be created.

  The function to create is slightly different from the ones created before because a function that returns the *_uri* variable is already defined in the parent contract, so it is necessary to override this function in the child smart contract, like in figure 40.

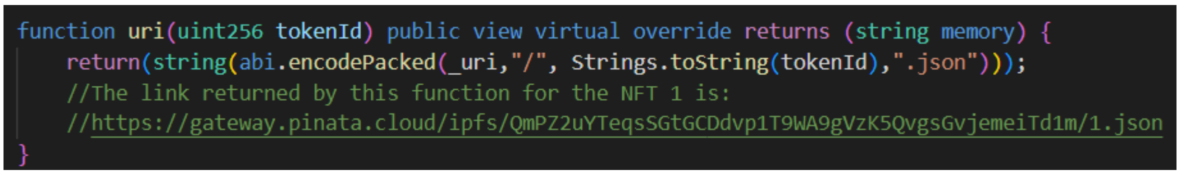


```
function uri(uint256) public view virtual override returns (string memory) {
    return _uri;
}
```

Figure 40. ERC-1155 function to get the URI (Parent contract).

In the way that the standard ERC-1155 treats the URI, it is impossible to create different NFTs with different images and attributes, meaning that by default, in ERC-1155, the URI is static. Therefore, in the case of requiring dynamic URIs, where each NFT will have its own metadata, it is necessary to program the function in the child smart contract which carries out this task.

This function is defined in the same way as in the parent contract, but the content is different. In this case, the function gets the value in the *_uri* variable, a slash character, NFT id converted to a string, and the .json termination. Then all these parts are concatenated, forming the URL of the metadata file. After concatenating the string, this link allows access to each metadata file uploaded at the IPFS. Figure 41 shows the function where this entire process is carried out.

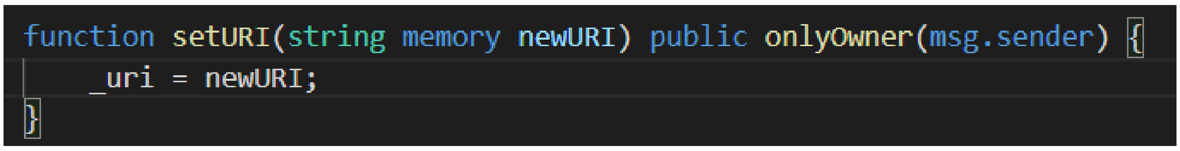


```
function uri(uint256 tokenId) public view virtual override returns (string memory) {
    return(string(abi.encodePacked(_uri,"/", Strings.toString(tokenId),".json")));
    //The link returned by this function for the NFT 1 is:
    //https://gateway.pinata.cloud/ipfs/QmPZ2uYTeqsSGtGCDdvp1T9WA9gVzK5QvgsGvjemeiTd1m/1.json
}
```

Figure 41. My_NFT smart contract *URI* function.

For this reason, it is important to name each file as the NFT ID that it belongs to.

- In addition to these steps, it is possible to add a function that allows modifying the *_uri* state variable. Then the owner would be able to modify some attributes like the image of each NFT if needed and display them. In the DApp being programmed in this case, the function that carries out this *_uri* variable modification is shown in figure 42.



```
function setURI(string memory newURI) public onlyOwner(msg.sender) {
    _uri = newURI;
}
```

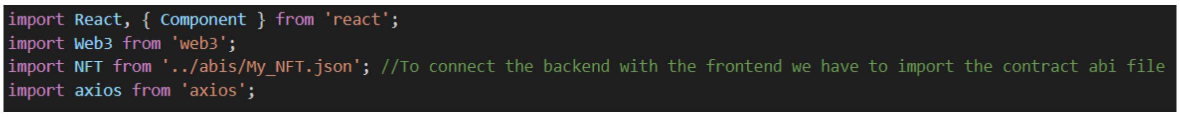
Figure 42. My_NFT: *setURI* function.

6.4.4 Front-end

The main goal of the front-end is to allow the communication between user and smart contract and show the necessary information in a user-friendly way.

The front-end comprises four main scripts. But before going deeper into each of them, it is necessary to expose some common parts of all of them. These parts are:

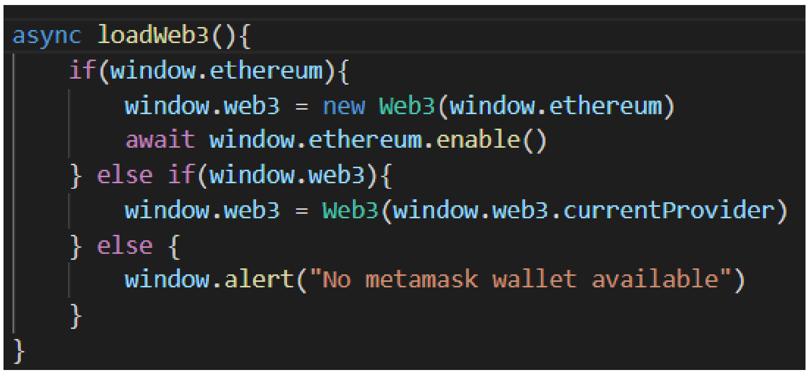
- Imports: Some libraries and the ABI file must be imported to develop a user interface and interact with the desired smart contract, as in figure 43.



```
import React, { Component } from 'react';
import Web3 from 'web3';
import NFT from '../abis/My_NFT.json'; //To connect the backend with the frontend we have to import the contract abi file
import axios from 'axios';
```

Figure 43. Front-end: necessary imports.

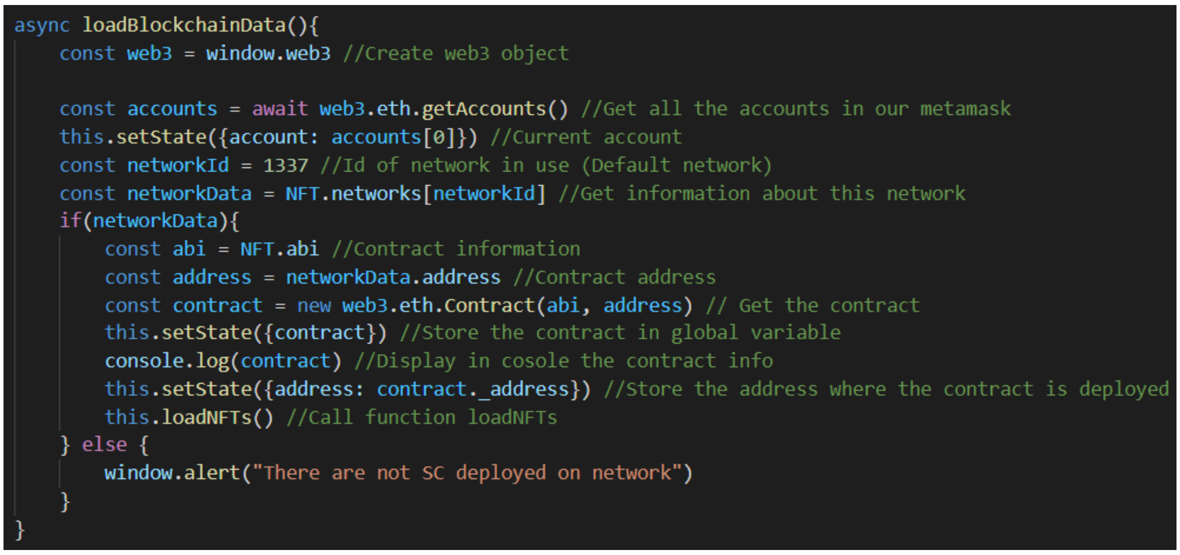
- Function to load the wallet: Because a Web3 is deployed, a procedure is required to connect the user's wallet to the website. Otherwise, the users would not be able to interact correctly with the website. For example, if a wallet is not connected to a website, an NFT could not be bought by the user interacting with the web page. The function to load the metamask wallet could be similar to the function in figure 44.



```
async loadWeb3(){
    if(window.ethereum){
        window.web3 = new Web3(window.ethereum)
        await window.ethereum.enable()
    } else if(window.web3){
        window.web3 = Web3(window.web3.currentProvider)
    } else {
        window.alert("No metamask wallet available")
    }
}
```

Figure 44. Front-end: *loadWeb3* function.

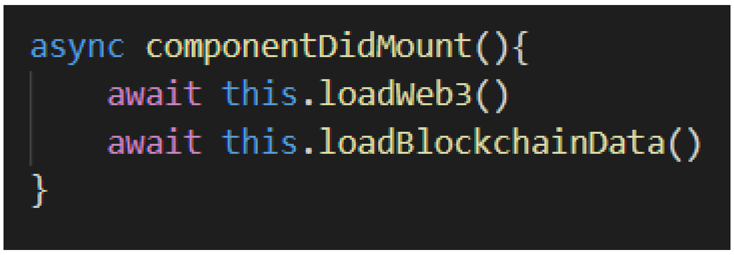
- Function to load the information in the smart contract: This function accesses the deployed contract in the network selected. Note that this network must be added in the *truffle-config* file. Furthermore, this function stores some information in variables accessible from the script itself. Figure 45 shows how this process is being programmed.



```
async loadBlockchainData(){
    const web3 = window.web3 //Create web3 object

    const accounts = await web3.eth.getAccounts() //Get all the accounts in our metamask
    this.setState({account: accounts[0]}) //Current account
    const networkId = 1337 //Id of network in use (Default network)
    const networkData = NFT.networks[networkId] //Get information about this network
    if(networkData){
        const abi = NFT.abi //Contract information
        const address = networkData.address //Contract address
        const contract = new web3.eth.Contract(abi, address) // Get the contract
        this.setState({contract}) //Store the contract in global variable
        console.log(contract) //Display in cosole the contract info
        this.setState({address: contract._address}) //Store the address where the contract is deployed
        this.loadNFTs() //Call function loadNFTs
    } else {
        window.alert("There are not SC deployed on network")
    }
}
```

Figure 45. Front-end: *loadBlockchainData* function.

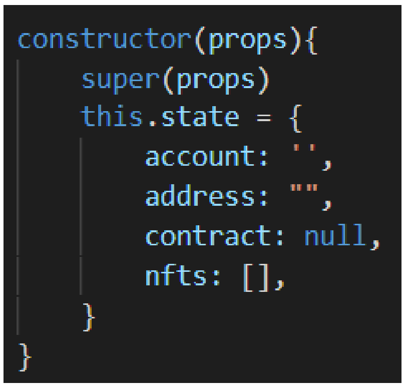
- ComponentDidMount function: These two last functions must be called when the application is initialized in a computer. The function programmed in the figure 46, *componentDidMount() accomplish with it.*



```
async componentDidMount(){
    await this.loadWeb3()
    await this.loadBlockchainData()
}
```

Figure 46. Front-end: *componentDidMount* function

- Constructor: Finally, it is required to use a constructor for each script created. All the constructors are very similar because they are only used to store values that will be used in different functions in the script. An example of this constructor is shown in figure 47.



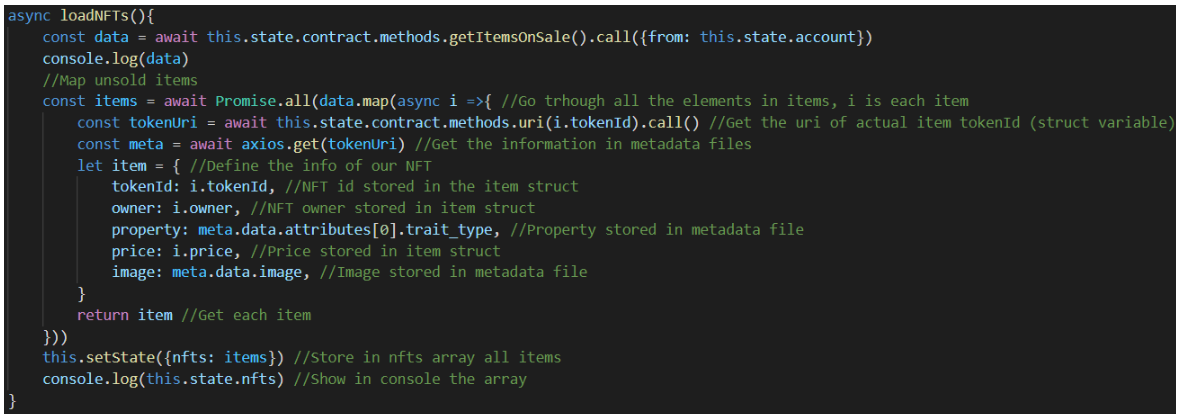
```
constructor(props){
    super(props)
    this.state = {
        account: '',
        address: "",
        contract: null,
        nfts: [],
    }
}
```

Figure 47. Index page: constructor.

The four scripts mentioned above are:

- Index: This page is the one that will be displayed when the webpage is accessed by a user, the NFTs on sale are shown here.

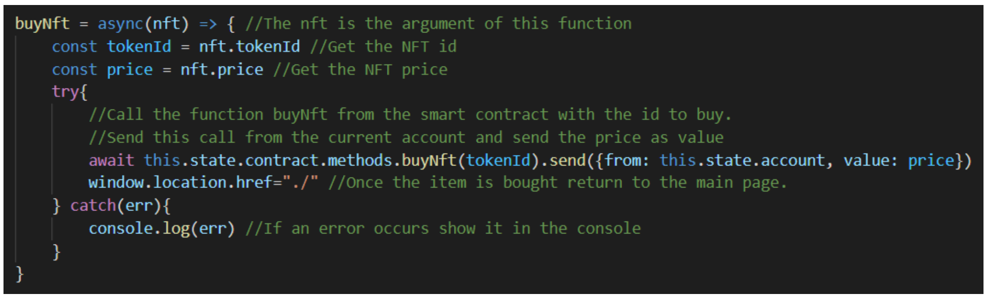
  Due to the fact that NFTs must be displayed, a function like the one in figure 48 to get the item objects on sale is essential.



```
async loadNFTs(){
    const data = await this.state.contract.methods.getItemsOnSale().call({from: this.state.account})
    console.log(data)
    //Map unsold items
    const items = await Promise.all(data.map(async i =>{ //Go trhough all the elements in items, i is each item
        const tokenUri = await this.state.contract.methods.uri(i.tokenId).call() //Get the uri of actual item tokenId (struct variable)
        const meta = await axios.get(tokenUri) //Get the information in metadata files
        let item = { //Define the info of our NFT
            tokenId: i.tokenId, //NFT id stored in the item struct
            owner: i.owner, //NFT owner stored in item struct
            property: meta.data.attributes[0].trait_type, //Property stored in metadata file
            price: i.price, //Price stored in item struct
            image: meta.data.image, //Image stored in metadata file
        }
        return item //Get each item
    }))
    this.setState({nfts: items}) //Store in nfts array all items
    console.log(this.state.nfts) //Show in console the array
}
```

Figure 48. Index page: *loadNFTs* function.

In addition, another function is required to buy the items and send the money. It is mandatory to specify the value to pay for an NFT. Otherwise, the wallet will not send the required amount to the seller, and the function will throw an exception because the price to pay is incorrect. This function can be called *buyNft* and is shown in figure 49.

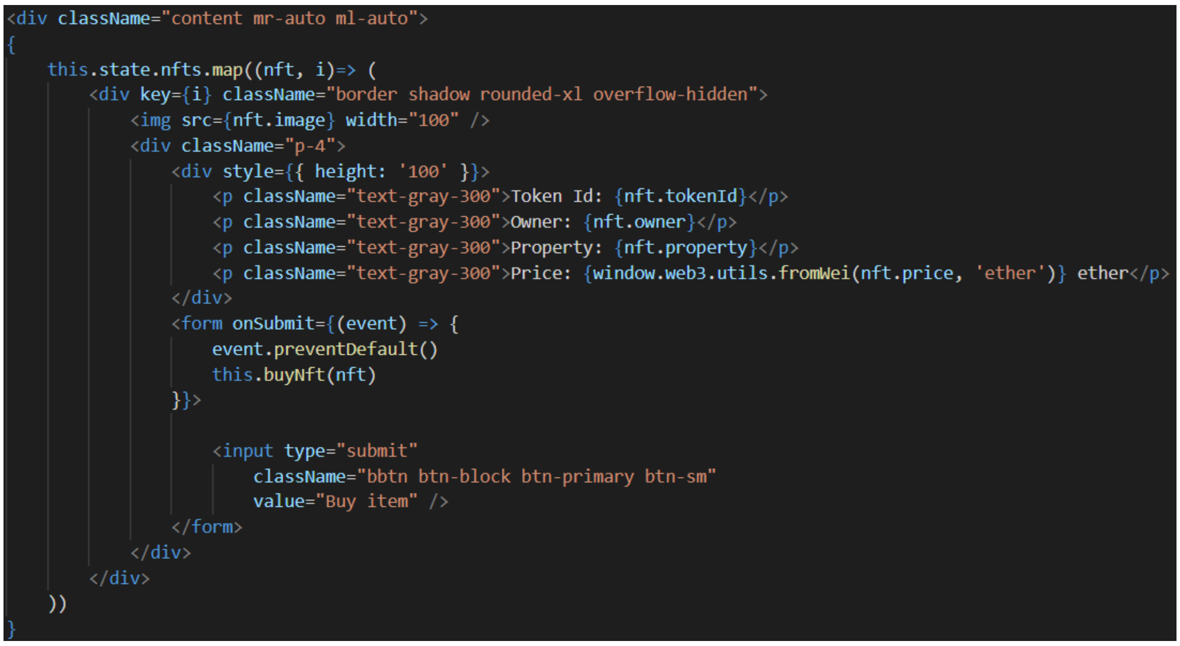


```
buyNft = async(nft) => { //The nft is the argument of this function
    const tokenId = nft.tokenId //Get the NFT id
    const price = nft.price //Get the NFT price
    try{
        //Call the function buyNft from the smart contract with the id to buy.
        //Send this call from the current account and send the price as value
        await this.state.contract.methods.buyNft(tokenId).send({from: this.state.account, value: price})
        window.location.href="./" //Once the item is bought return to the main page.
    } catch(err){
        console.log(err) //If an error occurs show it in the console
    }
}
```

Figure 49. Index page: *buyNFT* function.

Once the necessary functions to interact with the smart contract are programmed, programing the buttons that call these functions and page appearance is required.

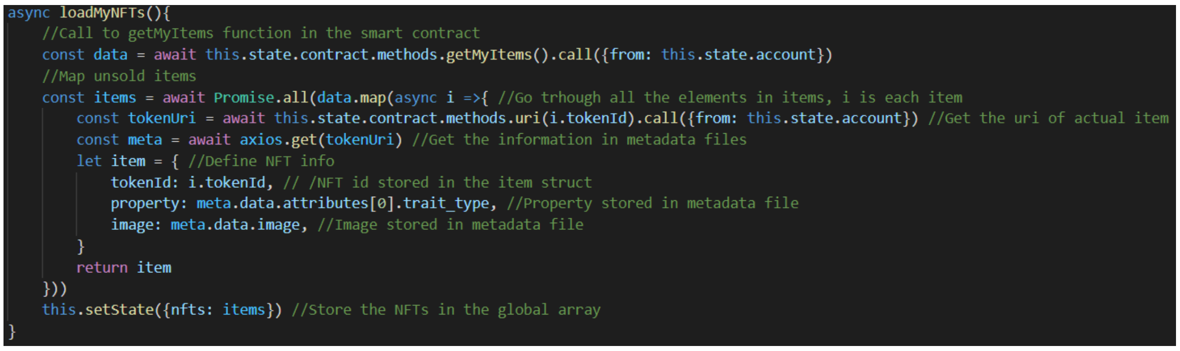
The HTML code is shown in the next image (Figure 50) to access each NFT saved on the global array. This code shows the required information of each NFT on sale. A button to call the function *buyNFT()* is displayed at the end of the function.



```
<div className="content mr-auto ml-auto">
{
    this.state.nfts.map((nft, i)=> (
        <div key={i} className="border shadow rounded-xl overflow-hidden">
            <img src={nft.image} width="100" />
            <div className="p-4">
                <div style={{ height: '100' }}>
                    <p className="text-gray-300">Token Id: {nft.tokenId}</p>
                    <p className="text-gray-300">Owner: {nft.owner}</p>
                    <p className="text-gray-300">Property: {nft.property}</p>
                    <p className="text-gray-300">Price: {window.web3.utils.fromWei(nft.price, 'ether')} ether</p>
                </div>
                <form onSubmit={(event) => {
                    event.preventDefault()
                    this.buyNft(nft)
                }}>

                    <input type="submit"
                        className="bbtn btn-block btn-primary btn-sm"
                        value="Buy item" />
                </form>
            </div>
        </div>
    ))
}
```

Figure 50. Index page: HTML code.

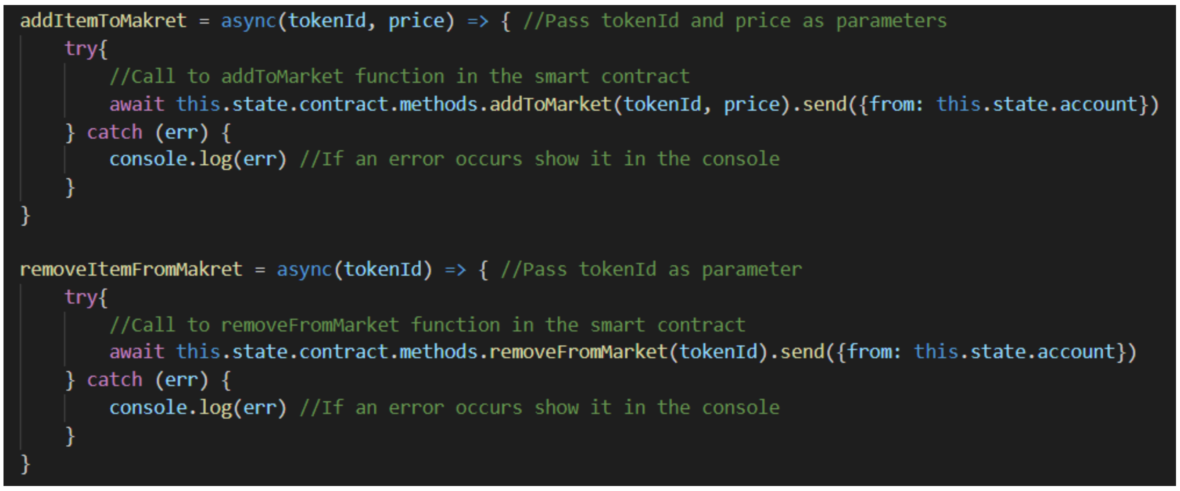
- Ownership script: In this script, a function to display the NFTs acquired with the wallet account connected to the page is programmed, and it is shown in figure 51.



```
async loadMyNFTs(){
    //Call to getMyItems function in the smart contract
    const data = await this.state.contract.methods.getMyItems().call({from: this.state.account})
    //Map unsold items
    const items = await Promise.all(data.map(async i =>{ //Go trhough all the elements in items, i is each item
        const tokenUri = await this.state.contract.methods.uri(i.tokenId).call({from: this.state.account}) //Get the uri of actual item
        const meta = await axios.get(tokenUri) //Get the information in metadata files
        let item = { //Define NFT info
            tokenId: i.tokenId, // /NFT id stored in the item struct
            property: meta.data.attributes[0].trait_type, //Property stored in metadata file
            image: meta.data.image, //Image stored in metadata file
        }
        return item
    }))
    this.setState({nfts: items}) //Store the NFTs in the global array
}
```

Figure 51. Ownership page: *loadMyNFTs* function.

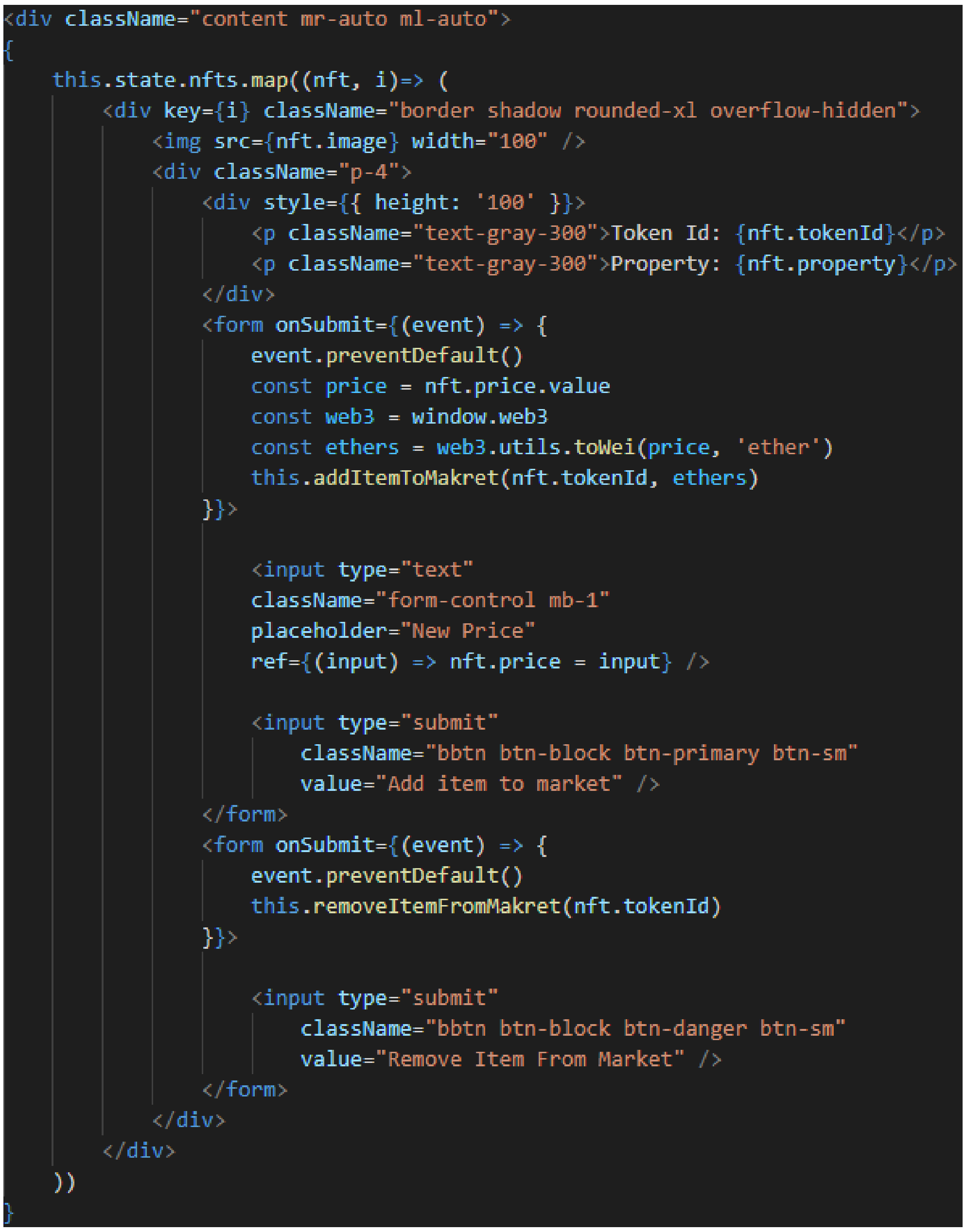
Furthermore, two additional functions have been added. The first one is to add an item to the market, and the second one is to remove it from the market. These two functions are the ones displayed in figure 52.



```
addItemToMakret = async(tokenId, price) => { //Pass tokenId and price as parameters
    try{
        //Call to addToMarket function in the smart contract
        await this.state.contract.methods.addToMarket(tokenId, price).send({from: this.state.account})
    } catch (err) {
        console.log(err) //If an error occurs show it in the console
    }
}

removeItemFromMakret = async(tokenId) => { //Pass tokenId as parameter
    try{
        //Call to removeFromMarket function in the smart contract
        await this.state.contract.methods.removeFromMarket(tokenId).send({from: this.state.account})
    } catch (err) {
        console.log(err) //If an error occurs show it in the console
    }
}
```

Figure 52. Ownership page: *addItemToMarket* and *removeItemFromMarket* functions.

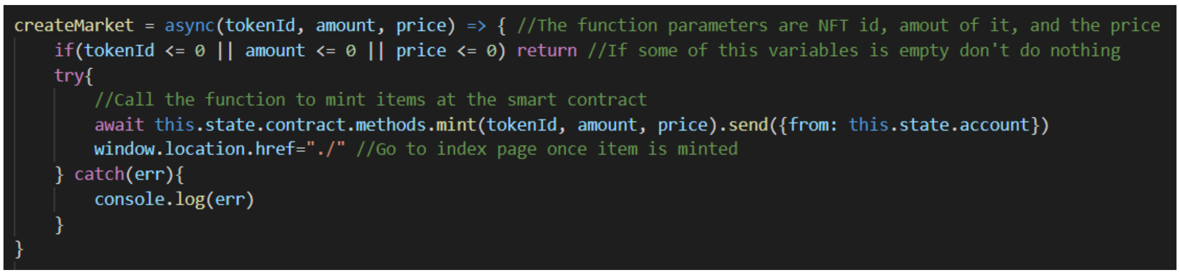
The function to display the acquired NFTs is called automatically when the ownership page is loaded. For each NFT displayed, the ID, the total amount of items in possession by the user who calls the function, and the NFT property are shown. It is also added a field to introduce the sale price and two buttons that call the functions of the previous image, add the item and remove it from the market. How these two buttons are programmed is shown in the next figure (Figure 53).



```
<div className="content mr-auto ml-auto">
{
    this.state.nfts.map((nft, i)=> (
        <div key={i} className="border shadow rounded-xl overflow-hidden">
            <img src={nft.image} width="100" />
            <div className="p-4">
                <div style={{ height: '100' }}>
                    <p className="text-gray-300">Token Id: {nft.tokenId}</p>
                    <p className="text-gray-300">Property: {nft.property}</p>
                </div>
                <form onSubmit={(event) => {
                    event.preventDefault()
                    const price = nft.price.value
                    const web3 = window.web3
                    const ethers = web3.utils.toWei(price, 'ether')
                    this.addItemToMakret(nft.tokenId, ethers)
                }}>

                    <input type="text"
                    className="form-control mb-1"
                    placeholder="New Price"
                    ref={(input) => nft.price = input} />

                    <input type="submit"
                        className="bbtn btn-block btn-primary btn-sm"
                        value="Add item to market" />
                </form>
                <form onSubmit={(event) => {
                    event.preventDefault()
                    this.removeItemFromMakret(nft.tokenId)
                }}>

                    <input type="submit"
                        className="bbtn btn-block btn-danger btn-sm"
                        value="Remove Item From Market" />
                </form>
            </div>
        </div>
    ))
}
```

Figure 53. Ownership page: HTML code.

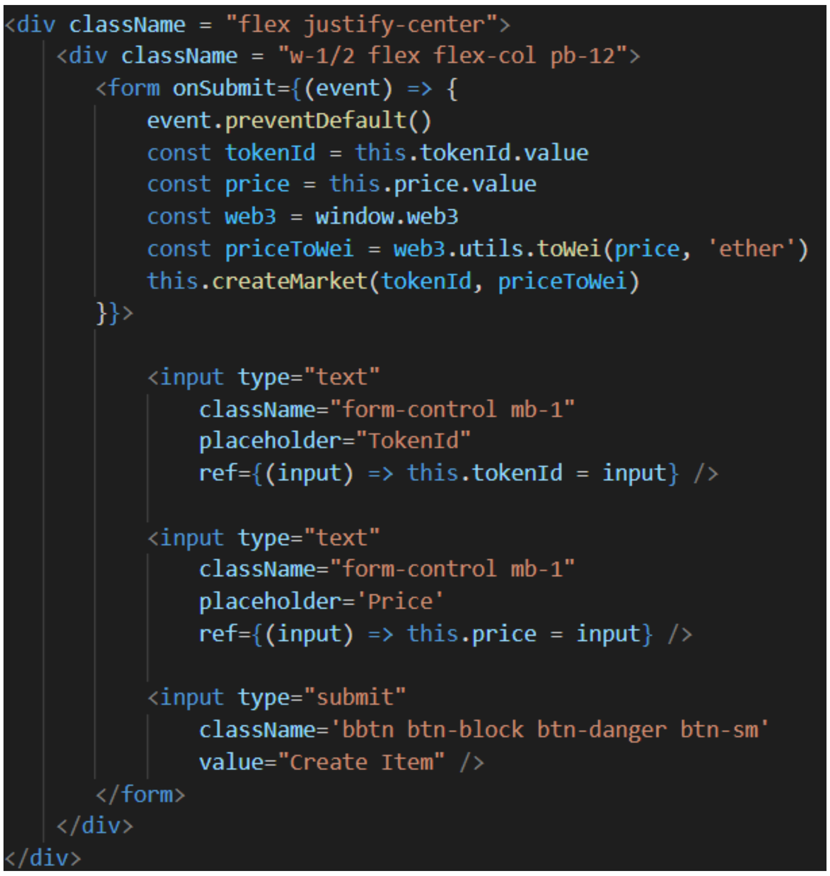
- Creator: In this script, the function to create NFTs and add these new items to the marketplace is programmed. The code of this function is shown in figure 54.



```
createMarket = async(tokenId, amount, price) => { //The function parameters are NFT id, amout of it, and the price
    if(tokenId <= 0 || amount <= 0 || price <= 0) return //If some of this variables is empty don't do nothing
    try{
        //Call the function to mint items at the smart contract
        await this.state.contract.methods.mint(tokenId, amount, price).send({from: this.state.account})
        window.location.href="./" //Go to index page once item is minted
    } catch(err){
        console.log(err)
    }
}
```

Figure 54. Creator page: Function to add items to the market.

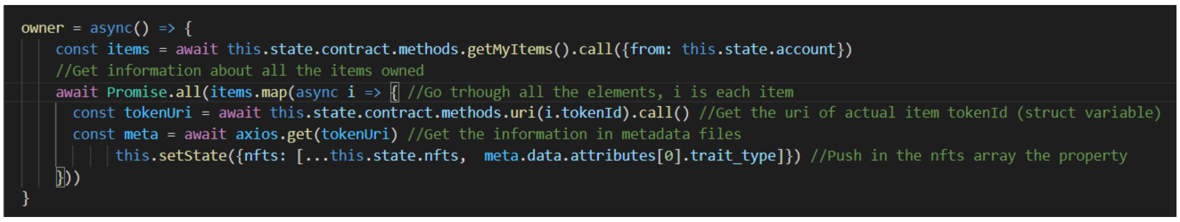
The HTML code to call the last function is a formulary that asks for the item ID and the price of the NFT. In the end, a button is displayed to add the item to the market. All of these elements are shown in figure 55.



```
<div className = "flex justify-center">
    <div className = "w-1/2 flex flex-col pb-12">
      <form onSubmit={(event) => {
          event.preventDefault()
          const tokenId = this.tokenId.value
          const price = this.price.value
          const web3 = window.web3
          const priceToWei = web3.utils.toWei(price, 'ether')
          this.createMarket(tokenId, priceToWei)
      }}>

          <input type="text"
              className="form-control mb-1"
              placeholder="TokenId"
              ref={(input) => this.tokenId = input} />

          <input type="text"
              className="form-control mb-1"
              placeholder='Price'
              ref={(input) => this.price = input} />

          <input type="submit"
              className='bbtn btn-block btn-danger btn-sm'
              value="Create Item" />
      </form>
    </div>
</div>
```

Figure 55. Creator page: HTML code

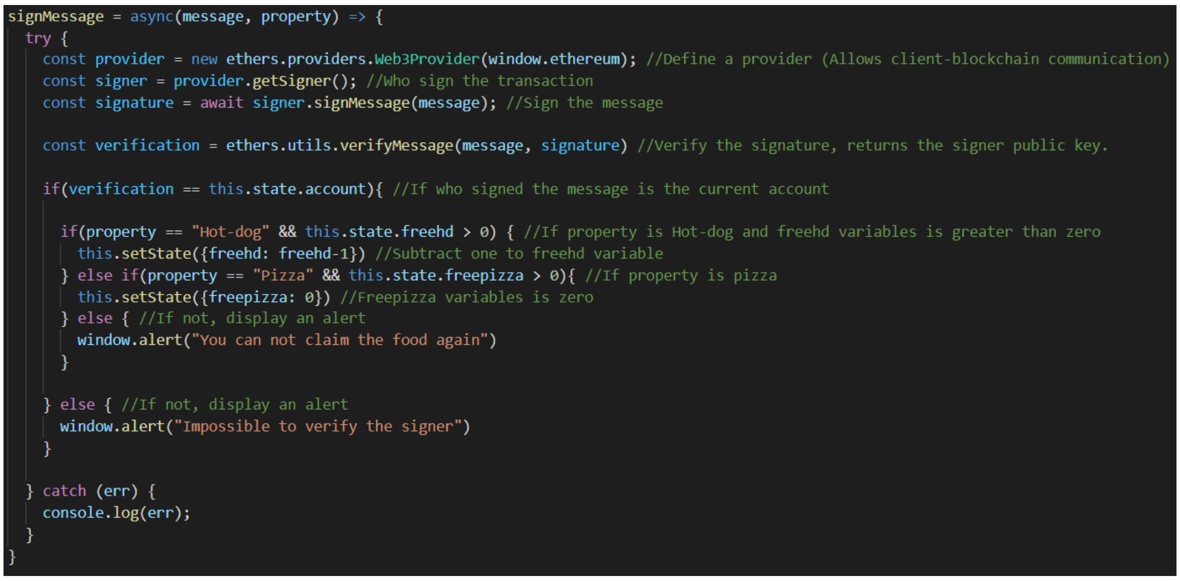
- Offers: This is the last script programmed. Here is where the special sales for the clients that bought an NFT are displayed. A function that gets the acquired NFTs' properties is necessary to display the correct offer. This function is programmed in the way that is shown in figure 56.



```
owner = async() => {
    const items = await this.state.contract.methods.getMyItems().call({from: this.state.account})
    //Get information about all the items owned
    await Promise.all(items.map(async i => { //Go trhough all the elements, i is each item
      const tokenUri = await this.state.contract.methods.uri(i.tokenId).call() //Get the uri of actual item tokenId (struct variable)
      const meta = await axios.get(tokenUri) //Get the information in metadata files
          this.setState({nfts: [...this.state.nfts,  meta.data.attributes[0].trait_type]}) //Push in the nfts array the property
    }))
}
```

Figure 56. Offers page: Function to get the item properties.

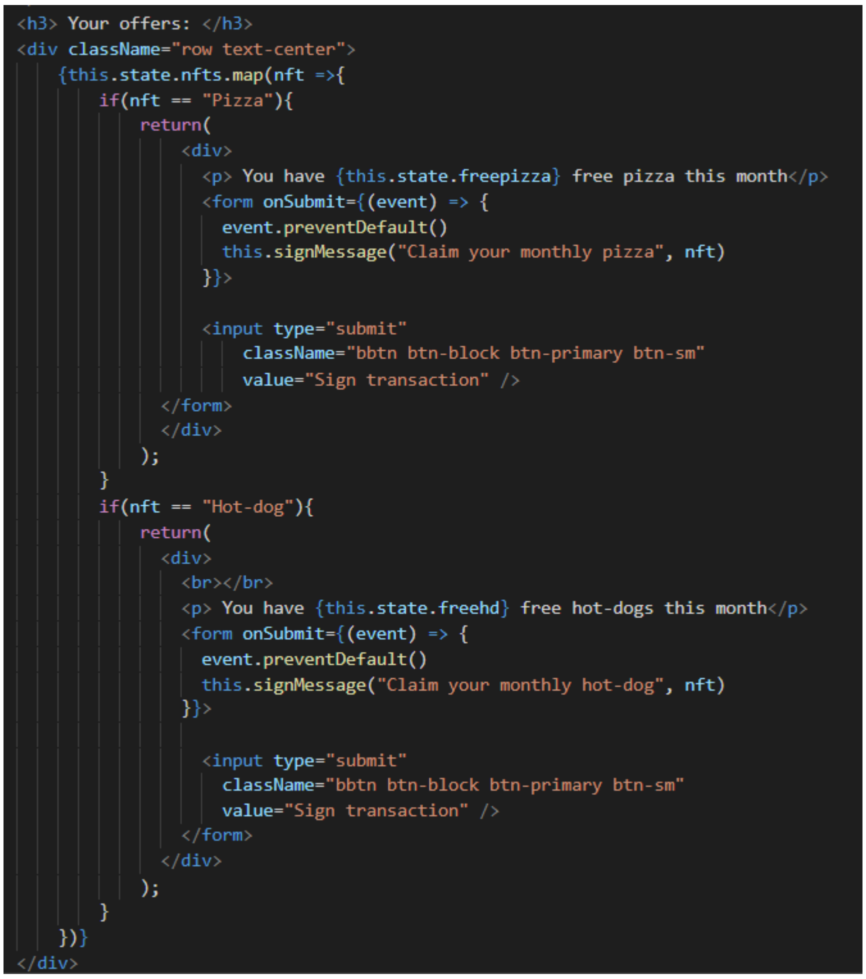
Furthermore, a function is required to demand the gift offered by a fast-food restaurant and register it in the blockchain. It is necessary to check who claims the gift is the NFT owner and avoid gas fee costs to the owner to carry out this action. A way to satisfy all these requirements is by signing a transaction. A message is passed to the function, then the account connected to the Web3 signs this message with the private key without exposing it publicly. Once this message is signed, the variable that controls the claimed gift is updated because only one gift is available per month. The needed code to carry out this whole process is shown in figure 57.



```
signMessage = async(message, property) => {
  try {
    const provider = new ethers.providers.Web3Provider(window.ethereum); //Define a provider (Allows client-blockchain communication)
    const signer = provider.getSigner(); //Who sign the transaction
    const signature = await signer.signMessage(message); //Sign the message

    const verification = ethers.utils.verifyMessage(message, signature) //Verify the signature, returns the signer public key.

    if(verification == this.state.account){ //If who signed the message is the current account

      if(property == "Hot-dog" && this.state.freehd > 0) { //If property is Hot-dog and freehd variables is greater than zero
        this.setState({freehd: freehd-1}) //Subtract one to freehd variable
      } else if(property == "Pizza" && this.state.freepizza > 0){ //If property is pizza
        this.setState({freepizza: 0}) //Freepizza variables is zero
      } else { //If not, display an alert
        window.alert("You can not claim the food again")
      }

    } else { //If not, display an alert
      window.alert("Impossible to verify the signer")
    }

  } catch (err) {
    console.log(err);
  }
}
```

Figure 57. Offers page: Function to sign a transaction and claim the gift.

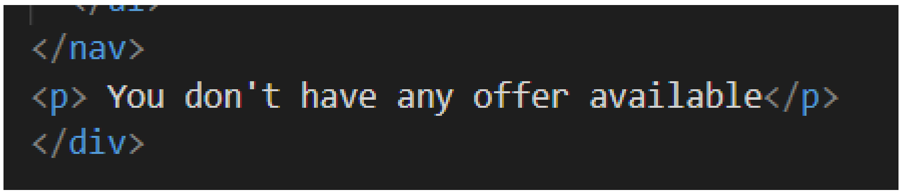
The HTML code is divided into two parts. The first one is the case that the holder possesses one or more NFTs. In this case, the number of available monthly offers and a button to sign the transaction to claim the gift are displayed. The function that accomplishes this process is shown in figure 58.



```
<h3> Your offers: </h3>
<div className="row text-center">
    {this.state.nfts.map(nft =>{
        if(nft == "Pizza"){
            return(
                <div>
                    <p> You have {this.state.freepizza} free pizza this month</p>
                    <form onSubmit={(event) => {
                      event.preventDefault()
                      this.signMessage("Claim your monthly pizza", nft)
                    }}>

                    <input type="submit"
                        className="bbtn btn-block btn-primary btn-sm"
                        value="Sign transaction" />
                  </form>
                  </div>
            );
        }
        if(nft == "Hot-dog"){
            return(
              <div>
                <br></br>
                <p> You have {this.state.freehd} free hot-dogs this month</p>
                <form onSubmit={(event) => {
                  event.preventDefault()
                  this.signMessage("Claim your monthly hot-dog", nft)
                }}>

                  <input type="submit"
                    className="bbtn btn-block btn-primary btn-sm"
                    value="Sign transaction" />
                </form>
              </div>
            );
        }
    })}
</div>
```

Figure 58. Offers page: HTML code, first part.

And in the second part, where the holder's case does not have any NFT, a message is shown. The message is the one displayed in figure 59.

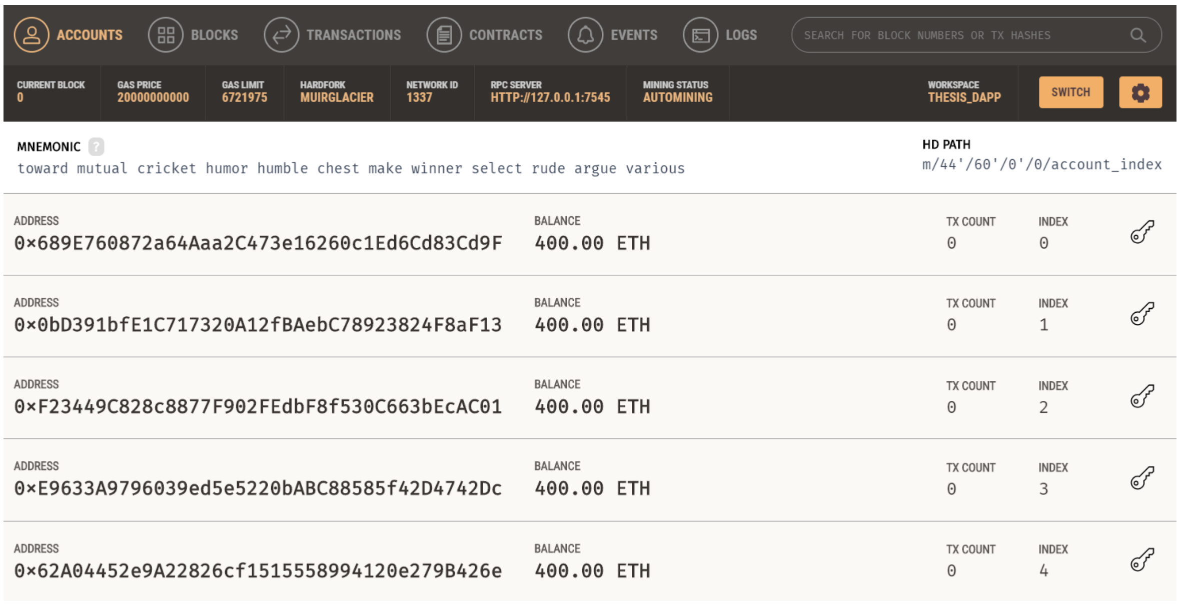


```
</nav>
<p> You don't have any offer available</p>
</div>
```

Figure 59. Offers page: HTML code, second part.

6.4.5 DApp deployment

Once the DApp is completely programmed, it is necessary to deploy it on a test-net or a main-net. Independently of the network where the project is deployed, it is essential to have an account with some real or fake money. Ganache is very useful here because it helps create a blockchain in the local network providing a network id, accounts with money, and other information about blocks, logs, and events. This information, or a tab to access it, is displayed in the main panel when a new project is created in Ganache. As can be appreciated in the following figure 60, ganache also provides the mnemonic phrase used in the *truffle-config.js* file. These are the twelve words that must be pasted into the *.secret* file, as figure 61 shows.



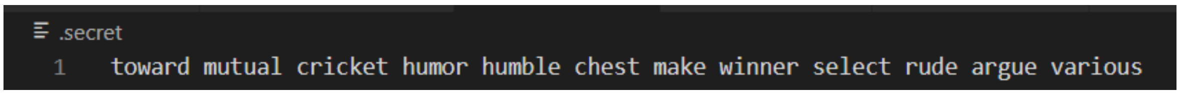
CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
0	20000000000	6721975	MUIRGLACIER	1337	HTTP://127.0.0.1:7545	AUTOMINING	THESIS_DAPP

MNEMONIC
toward mutual cricket humor humble chest make winner select rude argue various

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×689E760872a64Aaa2C473e16260c1Ed6Cd83Cd9F	400.00 ETH	0	0
0×0bD391bfE1C717320A12fBAebC78923824F8aF13	400.00 ETH	0	1
0×F23449C828c8877F902FEdbF8f530C663bEcAC01	400.00 ETH	0	2
0×E9633A9796039ed5e5220bABC88585f42D4742Dc	400.00 ETH	0	3
0×62A04452e9A22826cf1515558994120e279B426e	400.00 ETH	0	4

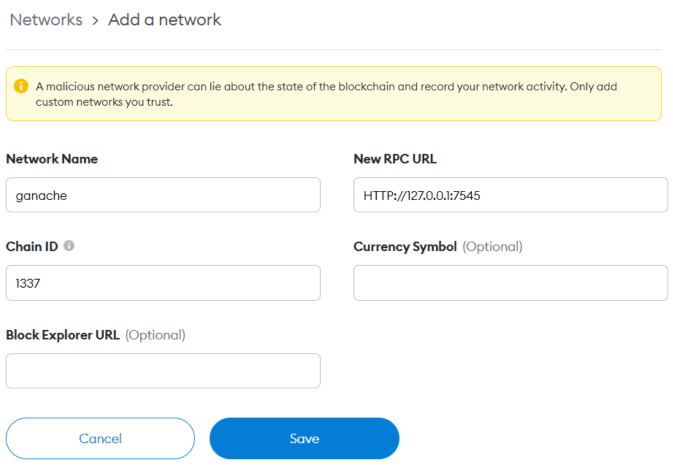
Figure 60. Ganache: accounts panel.



```
toward mutual cricket humor humble chest make winner select rude argue various
```

Figure 61. .secret file.

It is necessary to add the ganache network to metamask to import the ganache accounts in the wallet as a real account. The essential information required to create the ganache network in metamask is on top of the ganache account panel, and the way to proceed is shown in figure 62.


Networks > Add a network

A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.

Network Name	ganache
New RPC URL	HTTP://127.0.0.1:7545
Chain ID	1337
Currency Symbol (Optional)	
Block Explorer URL (Optional)	

Figure 62. Adding ganache network to metamask wallet.

The accounts shown in figure 60 can be imported into the metamask wallet and used to connect the wallet to the DApp and carry out the necessary transactions. Figure 63 shows the first ganache account and its balance imported into the metamask wallet.

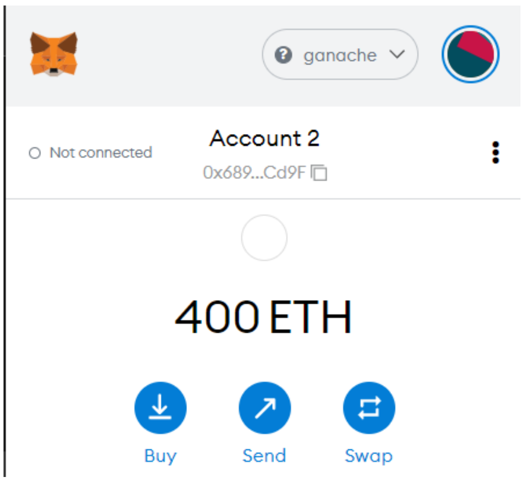


Figure 63. Ganache account imported into metamask wallet.

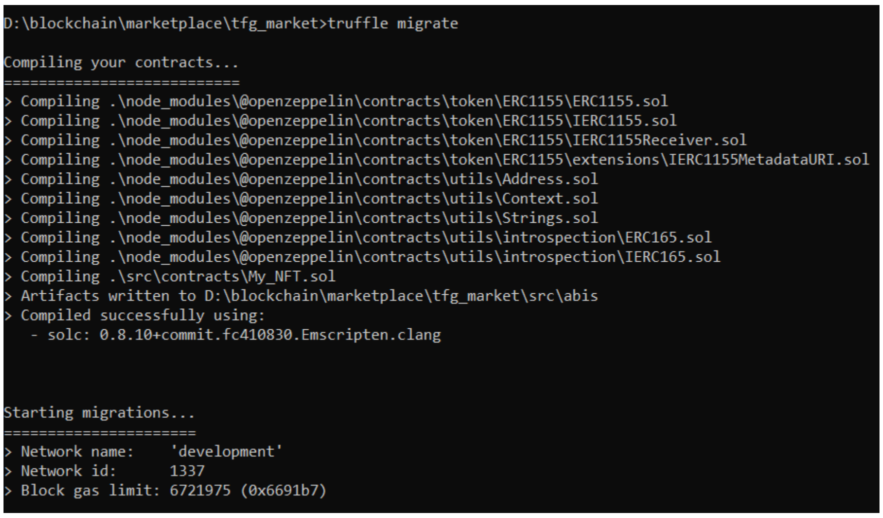
Once the *.secret* file contains the security phrase and the *truffle-config.js* file contains the network where the smart contracts will be deployed, it is possible to upload the smart contract to this blockchain.

Smart contracts are deployed in the default network with a simple command provided by truffle. This command is `truffle migrate`. The default network is *development* network that deploys the contract on localhost.

This is the base command and allows the deployment of new smart contracts without modifying the information of the contracts already deployed. It is also possible to add flags to modify some information, e.g., the network where the smart contracts are deployed can be changed using the `flag –network networkName`. Furthermore, if the smart contracts were deployed and modified later, adding the flag `--reset` allows re-deploying a smart contract and resetting its information. It is important to mention that each time a smart contract is deployed again, it could seem like it was possible to edit it whenever required. But this is not possible because when it is re-deployed, the address changes, so it actually is a new and different smart contract.

When the command is executed, the process starts. The first step in the deploying process is to compile the contracts. This action does not uniquely compile the smart contracts programmed but also the smart contracts imported by parents and children. In case there is a problem during the compilation, the smart contracts are not uploaded to the blockchain. The migration process starts if all the programmed and imported smart contracts compile.

Firstly, some information about the network where the project is being deployed is shown when the migration starts. This network information is shown at the bottom of figure 64.

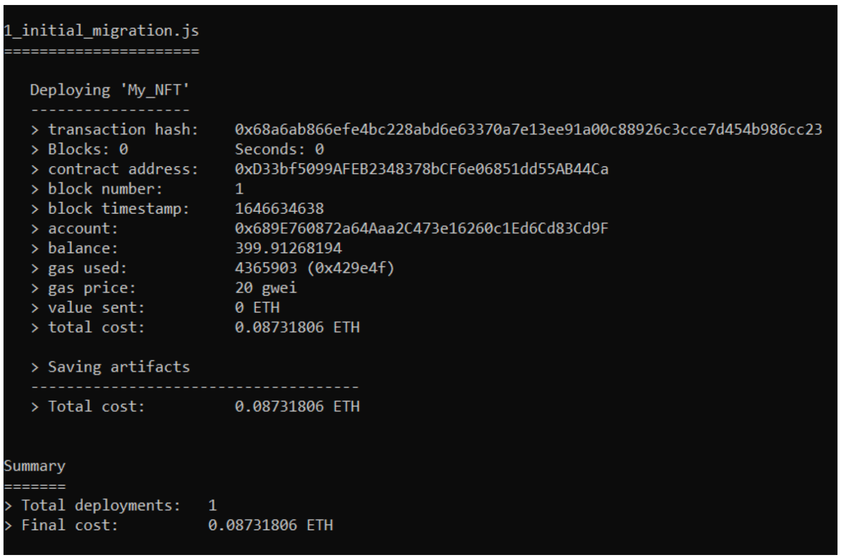


```
D:\blockchain\marketplace\tfg_market>truffle migrate

Compiling your contracts...
===========================
> Compiling .\node_modules\@openzeppelin\contracts\token\ERC1155\ERC1155.sol
> Compiling .\node_modules\@openzeppelin\contracts\token\ERC1155\IERC1155.sol
> Compiling .\node_modules\@openzeppelin\contracts\token\ERC1155\IERC1155Receiver.sol
> Compiling .\node_modules\@openzeppelin\contracts\token\ERC1155\extensions\IERC1155MetadataURI.sol
> Compiling .\node_modules\@openzeppelin\contracts\utils\Address.sol
> Compiling .\node_modules\@openzeppelin\contracts\utils\Context.sol
> Compiling .\node_modules\@openzeppelin\contracts\utils\Strings.sol
> Compiling .\node_modules\@openzeppelin\contracts\utils\introspection\ERC165.sol
> Compiling .\node_modules\@openzeppelin\contracts\utils\introspection\IERC165.sol
> Compiling .\src\contracts\My_NFT.sol
> Artifacts written to D:\blockchain\marketplace\tfg_market\src\abis
> Compiled successfully using:
   - solc: 0.8.10+commit.fc410830.Emscripten.clang



Starting migrations...
======================
> Network name:    'development'
> Network id:      1337
> Block gas limit: 6721975 (0x6691b7)
```

Figure 64. Deploying process: Smart contract compilation and network information.

Once the information about the network is shown, the deployment starts. Note that the first smart contract to compile is deployed throughout the initial migration file.

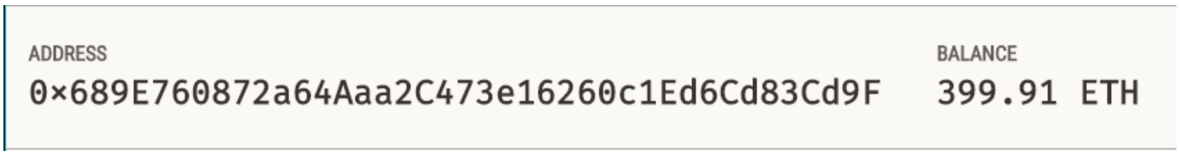
Some interesting information is displayed in the command-line interface, like the block where this transaction is stored, the contract address, the account which deploys the contract (the contract owner account), the price, et cetera. This information is shown in figure 65.



```
1_initial_migration.js
======================

   Deploying 'My_NFT'
   ------------------
   > transaction hash:    0x68a6ab866efe4bc228abd6e63370a7e13ee91a00c88926c3cce7d454b986cc23
   > Blocks: 0            Seconds: 0
   > contract address:    0xD33bf5099AFEB2348378bCF6e06851dd55AB44Ca
   > block number:        1
   > block timestamp:     1646634638
   > account:             0x689E760872a64Aaa2C473e16260c1Ed6Cd83Cd9F
   > balance:             399.91268194
   > gas used:            4365903 (0x429e4f)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.08731806 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.08731806 ETH


Summary
=======
> Total deployments:   1
> Final cost:          0.08731806 ETH
```

Figure 65. Deploying process: Smart contract migration information.

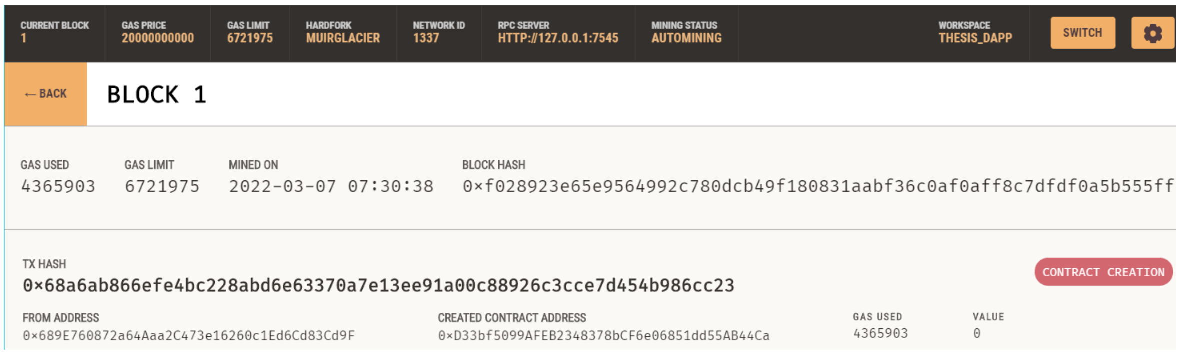
Some of this information can also be checked in ganache. For example, figure 66 shows that the balance of the account throughout the contract is deployed has been decremented.



```
ADDRESS                                       BALANCE
0×689E760872a64Aaa2C473e16260c1Ed6Cd83Cd9F    399.91 ETH
```

Figure 66. Ganache: Balance of contract owner.

Also, the block's information has been updated in the ganache blockchain.



```
CURRENT BLOCK  GAS PRICE    GAS LIMIT  HARDFORK     NETWORK ID  RPC SERVER              MINING STATUS  WORKSPACE
1              20000000000  6721975    MUIRGLACIER  1337        HTTP://127.0.0.1:7545   AUTOMINING     THESIS_DAPP

BLOCK 1

GAS USED   GAS LIMIT  MINED ON             BLOCK HASH
4365903    6721975    2022-03-07 07:30:38  0×f028923e65e9564992c780dcb49f180831aabf36c0af0aff8c7dfdf0a5b555ff

TX HASH                                                                                  CONTRACT CREATION
0×68a6ab866efe4bc228abd6e63370a7e13ee91a00c88926c3cce7d454b986cc23

FROM ADDRESS                                CREATED CONTRACT ADDRESS                     GAS USED   VALUE
0×689E760872a64Aaa2C473e16260c1Ed6Cd83Cd9F  0×D33bf5099AFEB2348378bCF6e06851dd55AB44Ca   4365903    0
```

Figure 67. Ganache: Block information.

As figure 67 shown, a block has been added, and this block contains the transaction belonging to the contract creation. In addition, the transaction hash, the account with which the transaction is carried out, and the contract address are part of the information saved in the block.

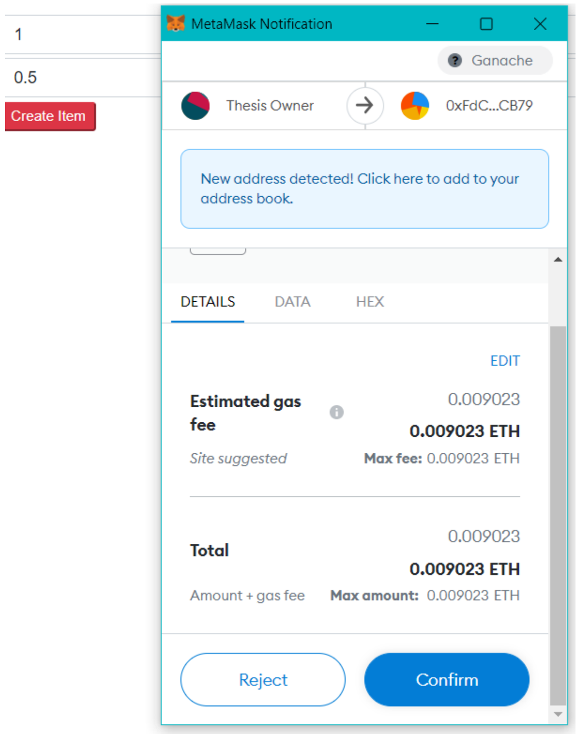
The deployment task does not need much time when the contract is deployed in the ganache network. But in the case of deploying the smart contracts in other test-nets or a main-net, this process can take some time. This is because the transaction must be mined, and this depends on the block time.

6.4.6 Final result (GUI).

Finally, once the smart contract is deployed, it is possible to interact with it using the user interface programmed in JavaScript. The GUI is composed of different sections with which the users can interact depending on what they want to do. These sections are:

- NFT creator tab: It is necessary to fill out the form displayed in this section to upload new NFTs to the marketplace. The information needed to create a new NFT is the NFT identifier and the price in ethers. Image and property are on the metadata file, then is not necessary to specify anything about them. The number of NFTs minted is fixed to one in the smart contract, so only one NFT can be mined per time.

  Once the user clicks on *create item* button, the metamask wallet is displayed to confirm the transaction. The cost of this transaction is uniquely the gas fees. Figure 68 shows this process.

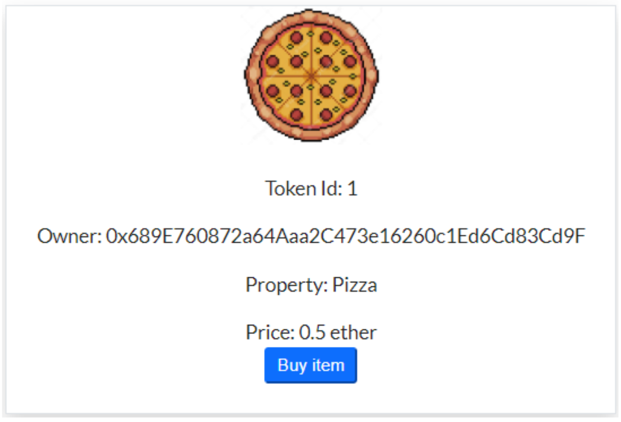


New address detected! Click here to add to your address book.

Estimated gas fee	0.009023
	0.009023 ETH
Site suggested	Max fee: 0.009023 ETH
Total	0.009023
	0.009023 ETH
Amount + gas fee	Max amount: 0.009023 ETH

Figure 68. Form and metamask wallet before confirming the transaction.

- Marketplace: In this section, the NFTs on sale are displayed. Some information about these NFTs is also displayed, like the NFT image, identification number, number of tokens available on sale, NFT property, and the price in ethers. At the bottom of each NFT appears a button to buy it, as shown in figure 69.


Token Id: 1

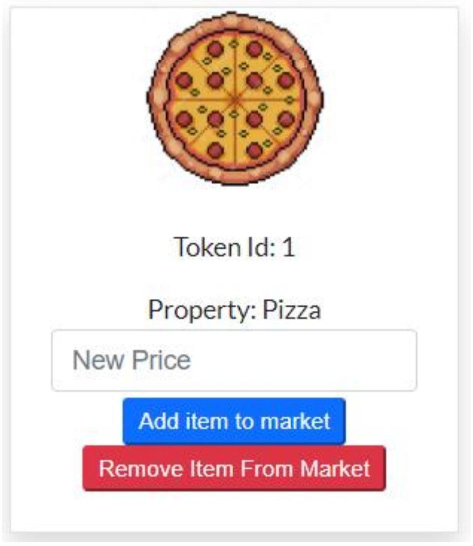
Owner: 0x689E760872a64Aaa2C473e16260c1Ed6Cd83Cd9F

Property: Pizza

Price: 0.5 ether

Figure 69. NFT is on sale in the marketplace section.

- My collection: The purpose of this page is to display the NFTs acquired by the current user. It is also possible to interact with the NFT. Users can add the item to the market or remove it from it. Each time the user wants to perform an action, the gas fee must be paid. An example of an NFT on *my collection* page is shown in figure 70.


Token Id: 1

Property: Pizza

Figure 70. User collection.

- Offers: This is the last section which displays the offers to those who bought an NFT. Depending on the NFT property, the displayed offer will change. When the claim button is clicked, the metamask wallet extension is opened to sign the message. If it is signed, the number of free food for that month is decreased by one unit. This process of signing a transaction is displayed in the next figure (Figure 71).

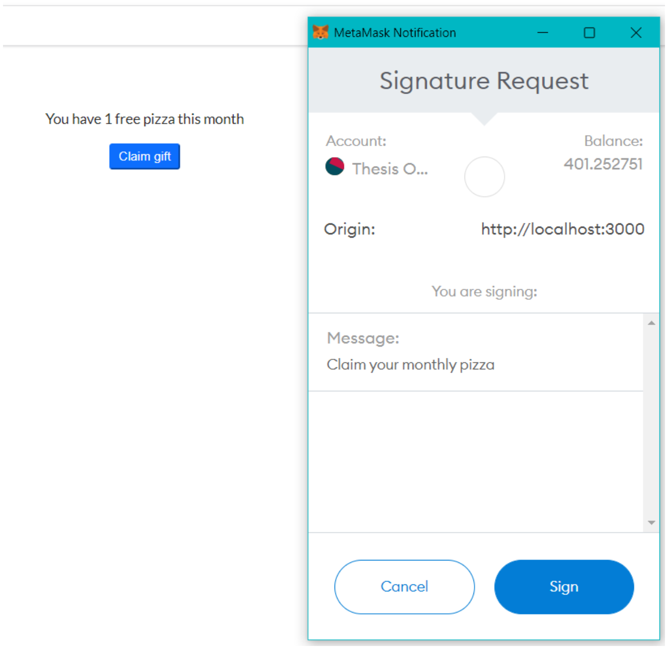


Figure 71. Offer to users who bought pizza NFT.

7 Summary

Currently, there still exists a lot of ignorance about blockchain, DApps, smart contracts, NFTs, and all these terms. Some people think that blockchain only manages cryptocurrencies, and NFTs are only used to speculate with digital art. Maybe this is because there are only reported cases of people turning rich or poor in a few days or a few hours in traditional media. This traditional media does not inform society about the powerful technology that supports cryptocurrencies and NFTs. They do not inform about the benefits of using this technology in a business or daily life. They only publish about extreme cases where money is involved. This is what only matters, the money.

Only those curious, and maybe slightly visionaries, who do not consider these traditional media and get information from books or other specialized fonts, understand this technology and know its real power. It is essential to avoid biased media and get information from legitimate sources. Then, once the topic is understood, it is suitable to judge if it is interesting or not.

For sure current blockchains have defects. There is a lot of work to do in order to improve this technology and simplify the way to use it, and this is my recommendation. Simplifying how a user creates a wallet, buys cryptocurrencies, and uses them on Web3 websites is necessary and will help expand its use. Also, educate the people to make that these people understand the technology and the tools to interact with web3.

The final goal of this work was to demonstrate that this technology goes beyond the current applications and it is well suited to create new and different approaches to managing a company or other kinds of organizations. This task has been accomplished by programming a DApp that could be used in business and giving examples about new companies that use this technology differently. The importance of the use case presented in this work is the philosophy acquired and the different points of view offered to manage a business and build communities.

8 Bibliography.

Albert 2022. Networks. Ethereum. Retrieved on 17 February 2022. Available at Networks | ethereum.org

Antonopoulos & Wood 2019. Mastering Ethereum: Building smart contracts. 1005 Gravenstein Highway North, Sebastopol, CA 95472. Published by O'Reilly Media, Inc. Available at GitHub - ethereumbook/ethereumbook: Mastering Ethereum, by Andreas M. Antonopoulos, Gavin Wood

Anwar 2018. What Are The Benefits Of Ethereum Decentralized Platform?. 101 Blockchains. Retrieved on 7 January 2022. Available at What Are the Benefits of Ethereum Decentralized Platform? (101blockchains.com)

Barker 2021. Getting started with Solana Development. Solana. Retrieved on 10 February 2022. Available at Getting Started with Solana Development

Bhattacharya 2021. What Is Web 3.0? The Future of the Internet. Single Grain. Retrieved on 12 January 2022. Available at What Is Web 3.0? The Future of the Internet - Single Grain

Bhattacharya 2021. Web3.0 application. Retrieved on 12 January 2022. Available at What Is Web 3.0? The Future of the Internet - Single Grain

Binance. Introduction of Binance Smart Chain. Binance. Retrieved on 10 February 2022. Available at Introduction - Binance Chain Docs

Blanco 2021. Solidity. Visual Studio Marketplace. 20 January 2022. Available at Solidity - Visual Studio Marketplace

Bureau 2021. Pak, Beeple, CryptoPunks, the Internet's source code – these are the 15 most expensive NFTs sold in 2021. Business Insider. Retrieved on 17 January 2022. Available at Pak, Beeple, CryptoPunks, the Internet's source code – these are the 15 most expensive NFTs sold in 2021 | BusinessInsider India

Canny 2022. JPMorgan Says Ethereum Is Losing NFT Market Share to Solana. CoinDesk. Retrieved on 19 January 2022. Available at JPMorgan Says Ethereum Is Losing NFT Market Share to Solana (coindesk.com)

Chatfield 2021. Move Over Ethereum: 5 Blockchains That Support NFTs. MUO. Retrieved on 19 January 2022. Available at Move Over Ethereum: 5 Blockchains That Support NFTs (makeuseof.com)

Coinguides 2021. List of all EVM blockchains and how to add any EVM network to Metamask. Coinguides. Retrieved on 24 March 2022. Available at List of all EVM blockchains and how to add any EVM network to Metamask (coinguides.org)

Conroy 2022. Opinion—One year later: NFTs are (still) a scam. The North Wind. Retrieved on 28 February 2022. Available at Opinion—One year later: NFTs are (still) a scam – The North Wind (thenorthwindonline.com)

Dabit 2021. What is Web3? The Decentralized Internet of the Future Explained. freeCodeCamp. Retrieved on 7 February 2022. Available at What is Web3? The Decentralized Internet of the Future Explained (freecodecamp.org)

DeFi Llama. 2022. Total Value Locked All Chains. DeFi Llama. Retrieved on 1 February 2022. Available at Chain TVL - DefiLlama

Desai 2019. Data Locations In Solidity. C# Corner. Retrieved on 2 March 2022. Available at Data Locations 🗺 In Solidity (c-sharpcorner.com)

Ditsche and Streichfuss 2021. More than just hype: Blockchain technologies can bring greater efficiency in supply chain management. Roland Berger. Retrieved on 10 March 2022. Available at Massive potential for industry: Blockchain beyond the cryptocurrencies | Roland Berger

Eichler & Jansen 2017. BLOCKCHAIN AND DATA PROTECTION LAW: WHEN ANONYMOUS DATA BECOMES PERSONAL. Dotmagazine. Retrieved on 17 March 2022. Available at Blockchain and Data Protection Law: When Anonymous Data Becomes Personal - Who's Naughty Now? Can Santa's Business Model Survive the GDPR? - Safe Xmas - Issues - dotmagazine

Escalante-De Mattei 2021. $22 B. Spent on NFTs in 2021: Market for Burgeoning Medium Rapidly Expanded, Report Says. ARTnews. Retrieved on 16 January 2022. Available at $22 B. Spent on NFTs in 2021, Report Reveals – ARTnews.com

EthHub. ERC Token Standards. EthHub. Retrieved on 3 March 2022. Available at ERC Token Standards - EthHub

GeeksforGeeks 2021. Difference between Fetch and Axios.js for making HTTP requests. GeeksforGeeks. Retrieved on 28 January 2022. Available at Difference between Fetch and Axios.js for making http requests - GeeksforGeeks

GeeksforGeeks 2022. Best Ethereum Development Tools to Create Dapps. Geeksfor-Geeks. Retrieved on 25 February 2022. Available at Best Ethereum Development Tools to Create Dapps - GeeksforGeeks.

Geroni 2021. Understanding The Attributes Of Non-Fungible Tokens (NFTs). 101 Block-chains. Retrieved on 19 January 2022. Available at Understanding the Attributes of Non-Fungible Tokens (NFTs) - 101 Blockchains

GitHub Contributors A 2021. Contract ABI Specification. GitHub. Retrieved on 21 February 2022. Available at Solidity/abi-spec.rst at v0.8.12 · ethereum/Solidity · GitHub

GitHub Contributors B 2020. Layout of a Solidity Source File. GitHub. Retrieved on 22 Feb-ruary 2022. Available at https://github.com/ethereum/Solidity/blob/v0.6.8/docs/layout-of-source-files.rst

Glover 2021. The privacy dangers of web3 and DeFi – and the projects trying to fix them. Tech Monitor. Retrieved on 17 March 2022. Available at How will web3 and DeFi impact privacy? - Tech Monitor

Gogo 2022. Porn Star Lana Rhoades Makes Off With $1.5M in Apparent NFT Scam. Bein-crypto. Retrieved on 28 February 2022. Available at Porn Star Lana Rhoades Makes Off With $1.5M in Apparent NFT Scam - BeInCrypto

Hayes. 2022. Blockchain Explained. Investopedia. Retrieved on 10 February 2022. Availa-ble at Blockchain Definition: What You Need to Know (investopedia.com)

Hedera. What is a non-fungible token (NFT)? Hedera. Retrieved on 19 January 2022. Avail-able at What is a non-fungible token (NFT)? | Hedera

Hussey and Phillips. 2020. MetaMask: What It Is and How To Use It. Decrypt. Retrieved on 20 January 2022. Available at What is MetaMask? | The Beginner’s Guide - Decrypt

Investopedia team. 2022. Web 2.0 and Web 3.0. Retrieved on 13 May 2022. Available at Web 2.0 and Web 3.0 Definitions (investopedia.com)

Jaswal 2021. Gary Vaynerchuk Views on Crypto and NFTs. Crypto bulls club. Retrieved on 19 January 2022. Available at Gary Vaynerchuk on Crypto and NFTs (cryptobullsclub.com)

Jonah 2022. Developing Terra smart contracts. LogRocket Blog. Retrieved on 4 January 2022. Available at Developing Terra smart contracts - LogRocket Blog

Joshua 2021 A. ERC-721 NON-FUNGIBLE TOKEN STANDARD. Ethereum. Retrieved on 19 January 2022. Available at ERC-721 Non-Fungible Token Standard | ethereum.org

Joshua 2021 B. Token Standards. Ethereum. Retrieved on 4 March 2022. Available at Token Standards | ethereum.org

Khan, Low Tang Jung, Manzoor Ahmed Hashmani & Moke Kwai Cheong. 2021. Systematic Literature Review of Challenges in Blockchain Scalability. MDPI. Retrieved on 14 February 2022. Available at Sensors | Free Full-Text | Empirical Performance Analysis of Hyperledger LTS for Small and Medium Enterprises | HTML (mdpi.com)

LeewayHertz. HOW CAN NFT TICKETING DISRUPT THE TICKETING INDUSTRY?. LeewayHertz. Retrieved on 16 February 2022. Available at How can NFT Ticketing disrupt the ticketing industry? (leewayhertz.com)

McCubbin 2022. Intro to Web3.js · Ethereum Blockchain Developer Crash Course. Dapp University. Retrieved on 28 January 2022. Available at Intro to Web3.js · Ethereum Blockchain Developer Crash Course | Dapp University

MDN Contributors A. 2021. What is JavaScript?. MDN Web Docs. Retrieved on 21 January 2022. Available at What is JavaScript? - Learn web development | MDN (mozilla.org)

MDN Contributors B. 2022. Working with JSON. MDN Web Docs. Retrieved on 17 February 2022. Available at Working with JSON - Learn web development | MDN (mozilla.org)

Modi 2018. Solidity programming essentials. Packt. Retrieved on 29 March 2022. Available at Method overriding | Solidity Programming Essentials (packtpub.com)

Node.js. Introduction to Node.js. Retrieved on 20 January 2022. Available at Introduction to Node.js (nodejs.dev)

NonFungible. 2022. CryptoPunks market history. NonFungible. Retrieved on 17 January 2022. Available at CryptoPunks | Market History and sales trends | NonFungible.com

Npm 2022. @truffle/hdwallet-provider. Npm. Retrieved on 4 March 2022. Available at @truffle/hdwallet-provider - npm (npmjs.com)

Openzeppelin. Constantly updating. Openzeppelin Contracts. GitHub. Retrieved on 23 January 2022. Available at GitHub - OpenZeppelin/openzeppelin-contracts: OpenZeppelin Contracts is a library for secure smart contract development.

Quiroz-Gutierrez 2022. Someone just bought a Florida home for $653,000 through an NFT sale. Fortune. Retrieved on 16 February 2022. Available at NFT sale of Florida home nets owner the equivalent of $653,000 | Fortune

React. Getting Started. React docs. Retrieved on 28 January 2022. Available at Getting Started – React (reactjs.org)

Reiff 2022. Bitcoin vs. Ethereum: What's the Difference?. Investopedia. Retrieved on 25 February 2022. Available at Bitcoin vs. Ethereum: What's the Difference? (investopedia.com)

Richards 2021. WEB2 VS WEB3. Ethereum documentation. Retrieved on 07 February 2022. Available at Web2 vs Web3 | ethereum.org

Richards 2022. Gas and Fees. Ethereum Documentation. Retrieved on 17 March 2022. Available at Gas and fees | ethereum.org

Rosen 2022. Gary Vaynerchuk says 98% of NFT projects will fail after the gold rush fades. Here's why 2 experts think he might be right. Insider. Retrieved on 16 February 2022. Available at Gary Vaynerchuk Says Most NFTs Will Fail. Here's What 2 Experts Said. (businessinsider.com)

Seth 2021. Public, Private, Permissioned Blockchains Compared. Investopedia. Retrieved on 10 February 2022. Available at Public, Private, Permissioned Blockchains Compared (investopedia.com)

Smith 2019. Advantages and disadvantages of using smart contracts – How to create a smart contract?. Know Techie. Retrieved on 7 January 2022. Available at Advantages and disadvantages of using smart contracts (knowtechie.com)

Smith 2022. Blocks. Ethereum documentation. Retrieved on 17 March 2022. Available at Blocks | ethereum.org

Solana A. Terminology. Solana documentation. Retrieved on 4 January 2022. Available at Terminology | Solana Docs

Solana B. Overview. Solana documentation. Retrieved on 4 January 2022. Available at Overview | Solana Docs

Solidity by Example A. Self Destruct. Solidity by Example. Retrieved on 7 January 2022. Available at Self Destruct | Solidity by Example | 0.8.10 (Solidity-by-example.org)

Solidity by Example B. Payable. Solidity by Example. Retrieved on 23 February 2022. Available at Payable | Solidity by Example | 0.8.10 (Solidity-by-example.org)

Solidity by Example C. Inheritance. Solidity by Example. Retrieved on 1 March 2022. Available at Inheritance | Solidity by Example | 0.8.10 (Solidity-by-example.org)

Solidity by Example D. Variables. Solidity by Example. Retrieved on 2 March 2022. Available at Variables | Solidity by Example | 0.8.10 (Solidity-by-example.org)

Solidity docs A. 2021. Contracts. Solidity. Retrieved on 23 February 2022. Available at Contracts — Solidity 0.8.12 documentation (Soliditylang.org)

Solidity docs B. Language grammar. Retrieved on 23 February 2022. Available at Language Grammar — Solidity 0.8.12 documentation (Soliditylang.org)

Solidity docs C. Types. Retrieved on 2 March 2022. Available at Types — Solidity 0.8.12 documentation (Soliditylang.org)

SPDX A. Overview. SPDX. Retrieved on 22 February 2022. Available at About - Software Package Data Exchange (SPDX)

SPDX B. MIT License. Retrieved on 22 February 2022. Available at MIT License | Software Package Data Exchange (SPDX)

Szabo 1996. Smart Contracts: Building Blocks for Digital Markets. Uva. Retrieved on 11 February 2022. Available at Nick Szabo -- Smart Contracts: Building Blocks for Digital Markets (uva.nl)

Temitope Akintade 2022. Blockchains that are Ethereum Virtual Machine Compatible. Crypto TV plus. Retrieved on 23 March 2022. Available at Blockchains that are Ethereum Virtual Machine Compatible | CryptoTvplus: DeFi, NFT, Bitcoin, Ethereum Altcoin, Cryptocurrency & Blockchain News, Interviews, Research, Shows

Terra. Terra core modules. Retrieved on 4 January 2022. Available at Terra Core modules — Terra Docs documentation

Thune 2022. What Are NFT Royalties & How Do They Work?. Seeking Alpha. Retrieved on 28 February 2022. Available at How Do NFT Royalties Work? | Seeking Alpha

Torky and Hassanien 2020. Integrating Blockchain and the Internet of Things in Precision Agriculture: Analysis, Opportunities, and Challenges. ResearchGate. Retrieved on 25 February 2022. Available at (PDF) Integrating Blockchain and the Internet of Things in Precision Agriculture: Analysis, Opportunities, and Challenges (researchgate.net)

Truffle Suite A. Configuration. Retrieved on 21 January 2022. Available at Configuration - Truffle Suite

Truffle Suite B. Ganache Overview. Retrieved on 20 January 2022. Available at Ganache | Overview - Truffle Suite

Truffle Suite C. Truffle Overview. Retrieved on 20 January 2022. Available at Truffle | Overview - Truffle Suite

Truffle Suite D. Running Migrations. Retrieved on 18 February 2022. Available at Running Migrations - Truffle Suite

Vardai 2021. What is ENS (Ethereum Name Service) and how does it work?. Forkast. Retrieved on 03 February 2022. Available at What is ENS (Ethereum Name Service) and how does it work? (forkast.news)

Veefriends. FAQ’s. Veefriends. Retrieved on 28 February 2022. Available at FAQ's | VeeFriends.com

Vermaak 2022. What is Web 3.0?. Alexandria. Retrieved on 15 February 2022. Available at What Is Web 3.0? | Alexandria (coinmarketcap.com)

Visual Studio Code. Getting Started. Retrieved on 20 January 2022. Available at Documentation for Visual Studio Code

White-Gomez 2021. The Definitive Timeline of Early NFTs on Ethereum. One37pm. Retrieved on 19 January 2022. Available at The Definitive Timeline of Early NFTs on Ethereum (one37pm.com)

Wikipedia. Semantic Web. Wikipedia. Retrieved on 12 January 2022. Available at Semantic Web - Wikipedia

Yang 2022. How to Find and Evaluate NFT Projects. Creator economy. Retrieved on 31 March 2022. Available at How to Find and Evaluate NFT Projects (creatoreconomy.so)