

University Degree in Computer Science and Engineering
2021-2022

Bachelor Thesis

“OAuth Authentication System for Web 3.0 based on Blockchain”

Alejandro Puch Marcos

Juan Miguel Gómez Berbis
Leganés, June 2022



This work is licensed under Creative Commons **Attribution – Non Commercial – Non Derivatives**

ABSTRACT

Blockchain technology is getting more and more used. A new paradigm for Internet has appeared with decentralized Blockchains: Web 3.0. Web 3.0 aims to connect the Internet and websites as we know them with decentralized Blockchains. Its main objective is to bring users a full control of their information, identity and assets instead of them being guarded by a centralized institution.

However, since the user is the only one responsible of their assets there is no way to recover them if you get them stolen by any malicious actor. Moreover this environment may seem too complicated for newcomers users and it can be more complex than the standard one.

In consequence, it is really important to establish a secure and easy to use system in order to connect to Web 3.0.

This project presents an authentication systems that ensures user safety and allows him, after setting up his account, to login through external resources (as Discord account or a one-single-use wallet) in order to not expose his assets or accounts to any actor.

Keywords: Blockchain; cryptocurrency: Web 3.0; Solana; token.

DEDICATION

With this project a stage of my life ends after finishing my Bachelor on Computer Science and Engineering. During this four years I have grown as a person as well as learned a lot. Thanks to my friends who shared with me these four years and made everything easier. Thanks to my parents for always supporting me and making me the person who I am now. Finally, thanks to my tutor Juanmi for helping me in the development of this project.

Thanks to all of you.

CONTENTS

1. INTRODUCTION.	1
1.1. Context	1
1.2. Problem	1
1.3. Motivation	2
1.4. Objective	2
1.5. Design Phases	2
1.6. Structure of Document.	3
2. STATE OF ART	4
2.1. Blockchain	4
2.2. Web3	5
2.3. NFTs - Non Fungible Tokens	6
2.4. Solana	7
2.4.1. Solana Network Design	7
2.4.2. Proof of Stake	7
2.5. Solana Items	8
2.5.1. NFTs & Metadata in Solana.	11
2.5.2. Validators	11
2.6. Arweave Protocol	12
2.7. Web Application Frameworks and Tools.	13
2.7.1. Next.js (Full-Stack Framework)	13
2.7.2. React	14
2.7.3. NodeJS	15
2.7.4. MongoDB & Moongose.	15
2.7.5. OAuth	17
2.7.6. Next-auth	17
2.7.7. Discord & Discord OAuth2	18
2.8. Express.js (RestAPI).	19

3. ANALYSIS OF PRODUCT	20
3.1. General Description of the Application	20
3.2. Wallet Ownership Verification module	21
3.3. NFTs Data & Metadata Extraction module	22
3.4. Session Access Control module.	22
3.5. Data Visualization module	23
3.6. Verified Collections Management module	23
3.7. Functional Requirements	24
3.7.1. Wallet Owner Verification module	25
3.7.2. NFTs Data & Metadata Extraction module	26
3.7.3. Session Access Control module.	27
3.7.4. Data Visualization module	28
3.7.5. Verified Collections Management module	29
3.8. Non-Functional Requirements	29
4. DESIGN OF PRODUCT	31
4.1. General Functional Architecture	31
4.2. Wallet Ownership Verification module.	32
4.3. NFTs Data & Metadata Extraction module	33
4.4. Session Access Control module.	34
4.5. Data Visualization module	35
4.6. Verified Collections Management module	36
4.7. Database Design	37
5. IMPLEMENTATION AND TESTS	39
5.1. Wallet Owner Verification	39
5.2. NFTs & Metadata Data Extraction	42
5.3. Session Access Control	47
5.4. Data Visualization	50
5.5. Verified Collections Management	53
6. PROJECT MANAGEMENT, SOCIO-ECONOMIC AND REGULATORY ASPECTS	59
6.1. Regulatory Framework	59

6.2. Socio-Economic Impact	59
6.2.1. Social Impact	59
6.2.2. Economical Impact.	60
6.3. Project Planning	61
6.4. Budget.	61
6.4.1. Cost of Labour	62
6.4.2. Cost of Product.	62
6.4.3. Indirect Costs.	62
6.4.4. Total Cost	63
6.5. Future Work Lines	63
7. CONCLUSION	66
BIBLIOGRAPHY.	67

LIST OF FIGURES

2.1	Centralized, decentralized and distributed systems [3].	5
2.2	Token Program and related actors [7].	9
2.3	Routing structure in a Next.js App.	14
2.4	React component.	15
2.5	MongoDB Document.	16
2.6	Mongoose Schema.	17
2.7	Discord Server [18].	19
4.1	Architecture Diagram.	32
4.2	Database Schemas.	37
5.1	Random Nonce Generation.	39
5.2	Signing Message with Nonce.	40
5.3	Fetch to addWallet API route.	40
5.4	Method to check if client-side and server-side nonce matches.	41
5.5	Signature request to verify wallet.	41
5.6	Successful login after wallet verification.	41
5.7	HTTP request to Solana API to obtain token list.	43
5.8	Method to obtain if a token has a supply of 1.	44
5.9	API route method to obtain NFT list of a wallet.	44
5.10	Collection Interface.	44
5.11	Function to obtain the Metadata of a NFT.	45
5.12	Function to filter the non-verified NFTs which do not use Arweave	46
5.13	NFT Data Extracted example.	46
5.14	Solana Database update after login.	48
5.15	Example of User logged with Discord and 3 wallets linked.	49
5.16	Metadata-Display component.	51
5.17	Collection-Display component.	52
5.18	Data Visualization example.	52

5.19	Example of Verified Collection Object to insert.	53
5.20	Method to obtain Mint List of a collection	54
5.21	Method to insert of the NFTs of a collection	54
5.22	Function to obtain public keys holders of a collection	55
5.23	Stored Data of inserted collection.	56
5.24	Stored Data of inserted NFTs.	56
5.25	Obtained Wallet Holders data.	56
5.26	Obtained Discord Holder data.	57

LIST OF TABLES

3.1	Functional Requirements Wallet Owner Verification module	25
3.2	Functional Requirements NFTs Data & Metadata Extraction module . . .	26
3.3	Functional Requirements Session Access Control module	27
3.4	Functional Requirements Data Visualization module	28
3.5	Functional Requirements Verified Collections Management module . . .	29
3.6	Non-Functional Requirements	30
5.1	Verification of Functional Requirements Wallet Owner Verification module	42
5.2	Verification of Functional Requirements NFTs Data & Metadata Extrac- tion module	47
5.3	Verification of Functional Requirements Session Access Control module .	50
5.4	Verification of Functional Requirements Data Visualization module . . .	53
5.5	Verification of Functional Requirements Verified Collections Manage- ment module	58
6.1	Gantt Diagram	61
6.2	Work Hours dedicated to the project.	61
6.3	Cost of Labour	62
6.4	Cost of Product	62
6.5	Indirect Costs	63
6.6	Total Cost	63

1. INTRODUCTION

In this section, what is pretended is to introduce the reader into this project to such an extent that he has a general idea of it and the related themes the project is surrounded by. This will be achieved by an explanation of the context, the problem to address, the objective of the project and the motivation behind it, the design phases and, finally, the structure of document.

1.1. Context

Nowadays almost everything in Internet is controlled by big entities, from social media (Twitter, Facebook), e-commerce (Amazon, Alibaba), users data (Google, Apple) to banking (central banks as Europe central bank). In consequence, all these entities have a full control of users and their data.

With the appearance of Blockchain and decentralized protocols a new paradigm has emerged: Web 3.0, a decentralized internet by the user for the users. Web 3.0 pursues to transfer these rights, control and ownership to the users instead of concentrating them in an organization. This obviously empowers the user giving him full control of his own information, identity and money. However, such an important and powerful aspect requires huge responsibility.

In the traditional (or centralized) world if you lose your credit card or bank credentials, you can contact the customer support of your bank and ask for a replacement or a solution. On the contrary, on Web 3.0 you are the only owner and person in charge of your money. If you forget or lose the credentials of your wallet you have no way to recover that money and it will be lost forever. In fact, there is a really famous case of a man called Stefan Thomas who has a wallet with a balance of 7002 Bitcoins, which is equivalent to \$211M at current Bitcoin prices, and he can't access it or use the funds as he has forgotten the password [1].

It is pretty clear that users needs to learn the best practices to assume and handle this responsibility. However, this can be really difficult even more for people which is not used to technology. Hence, it is also essential to build different applications, interfaces and services to facilitate everything for any new-coming user.

1.2. Problem

The way this environment works and the fact that all transactions are final, since there is no customer support or entity to claim an error, a fraud or a problem, makes it a perfect scenario for scammers and criminals who try to take advantage of innocent people. These

scammers can make inexperienced users (or even experts if they are not pretty cautious) confirming actions as accepting a fraudulent transaction or giving full access and control of your wallet, mainly through phishing. Users can easily fall for these while they are thinking they are doing a normal transactions and tragically end up realising they have lost all or a big part of their funds.

Moreover, it is really frequent for users to have multiple wallets (a similar to user accounts but in Blockchain) and most applications do not have a system which allows to access their services from multiple wallets at once. This makes the user to constantly have to switch accounts which results in a poor user experience and an increment in risk for the user (the more times a user has to login the more times he is vulnerable to phishing attacks).

1.3. Motivation

The motivation of this project is to contribute to create a safer and easier to use environment for Web 3.0, not only for the safety of currents users but also to encourage and ease the adoption of Web 3.0 and Blockchain. I firmly believe this technology can make the live of a lot of people (mainly of those with worse conditions) easier making this world a better place.

1.4. Objective

The objective of this project is to create a safe and trusted platform where you can connect your wallets in order to access to different features or privileges from different providers without having to directly connect to them, eliminating most possibilities to fall for any kind of scam coming from a malicious actor. This platform must be secure and at the same time save the least or none sensible data to avoid any kind of problem in case there is a successful time. It must ensure the legitimacy of the user in order to avoid any kind of impersonation.

Furthermore, this platform will improve user experience by giving full access to the services from multiple wallets at once, instead of having to continuously switch wallets.

1.5. Design Phases

The development of the project was divided in 4 phases: Analysis of product, Design of product, Implementation and Tests & Validations.

1.6. Structure of Document

The document will be divided in 7 sections.

In this first introduction section a general view of the project will be given. Next section is the State of Art where a generic study and description of the different concepts, topics and technologies related to the project will be done.

Subsequently, the analysis of the product will be the section where the different functionalities and features of the product will be defined and described obtaining the requirements of the application. In the Design of Product section the proposed solution will be described as well as the architecture of the proposed system.

Afterwards, in the Implementation and Tests section, the process of implementation will be explained in detail. Moreover, the different tests made to validate the functionalities and requirements of the application will be included.

Project Management, Socio-Economic and Regulatory aspects section will describe both the impacts of the project and its environment, as well as the different regulations affecting in any way the project. It will also include an explanation of the Project management including the planning, the budget and its future work lines.

Finally, a brief conclusion about the process of development of the project will be done.

2. STATE OF ART

In this chapter the State of Art will be defined and described. The different technologies, concepts and frameworks related to this project or used in it will be explained assuming zero previous knowledge about the theme in order to any person be able to read and understand this project document.

Along this section, the prices of different crypto currencies has been used for calculations. All these price were extracted from CoinMarketCap [2] at the date of 12th of June 11:00 AM (CEST).

2.1. Blockchain

A Blockchain is a distributed database which operates in a way in which the information is saved in blocks. A distributed database is a database formed by different nodes that can be located in the same or different locations. Each of these nodes stores a replica set of the information saved in the database and contribute and coordinates with the rest of nodes on the operation of the database.

Each block of the chain is assigned a Hash. A Hash is a value obtained through passing an input into a formula and used frequently in Cryptography for different reasons. Each block also saves the Hash of the previous and next block. Each Block is immutable since changing any information saved inside them would change the value of its Hash and, consequently, invalidate this replica set of the "edited" Blockchain. The previous block would point to a Hash which doesn't has a block assigned as well as the block after the edited one would have assigned a Hash of a non-existent block. This makes the data both immutable and chronological, since we can keep track of the order in which the blocks have been created and stored.

Blocks store transactions made in the network. In order to save a new block in the chain it needs to get accepted by consensus of a majority of the nodes forming the Blockchain. The most popular use case of the Blockchain is Bitcoin, a decentralized currency system which allows P2P (peer-to-peer) transactions from users ensuring their anonymity. However, not all the Blockchains systems are decentralized. In order to a Blockchain be fully centralized all its data must be transparent, there must be no hierarchy on the nodes and the system has to be free accessible,

- **Transparent Data:** All the blocks and data stored in the Blockchain must be accessible and reliable. This is easily achieved since in a Blockchain system each node has a copy of all the data stored in the Database.
- **No Hierarchy:** All nodes must have the same rights to form part of the consensus

of the creation of new blocks. Voting power is directly proportional to how much a node contributes to the system.

- **Free to Access:** Everyone must have the opportunity to form part of the system if they have the minimum required resources to build a node.

In Fig.2.1 we can see the differences between centralized, decentralized and distributed systems.

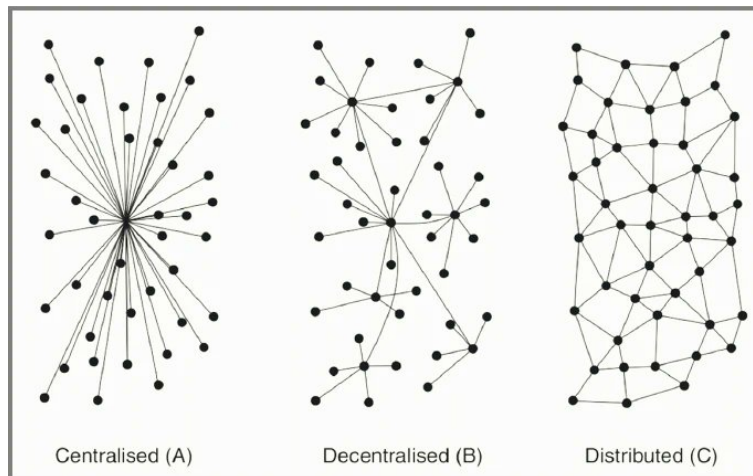


Fig. 2.1. Centralized, decentralized and distributed systems [3].

2.2. Web3

Web 3.0 is the new paradigm for the Internet that has been becoming more and more popular. It refers to the "next era" of Internet and is based on eliminating all the centralised entities and give all the control to the users.

It bases everything, from infrastructure to development and implementation, on decentralized systems. However, for the moment there is no system at production level for infrastructure or network (although there are projects working on developing possible solutions for these systems). Hence, for the moment web3 is just bridging the transactions and contents of internet with different decentralized Blockchains.

An example of a Web3 site would be a clone of AirBNB (a platform which connects property owners with tourist in order to rent out properties for holidays stays) but decentralized. In these decentralized option the marketplace fees would be distributed by token holders (similar to revenue sharing to shareholders in standard companies), the developers and maintainers of the marketplace (they would be periodically chosen by elections where the vote is defined by the amount of tokens holded) and the direction and decision making of the company would be made by the community (in same way as developers election).

There are a lot of projects, called DAOs (decentralized autonomous organization) which are trying to make these transitions from standard web to web3 and you can be part of them just by simply buying any amount of tokens or earning by contributing to the development of the project.

2.3. NFTs - Non Fungible Tokens

A NFT is a non fungible token. The main difference from an standard token is that an NFT is unique identification code in the Blockchain. This tokens are also able to store metadata. This metadata can be the same in two different NFTs but, as explained before, the identification code must be unique.

At the moment the main uses given to NFTs are centered around collectibles (art pieces, sport cards, game skins..) however this technology allows an infinite uses for them. For example a concerts tickets company would be able to tokenize the tickets of an specific concert, store them in the Blockchain and send them to the buyers as the entry ticket for the country. In this case it may not have any extra functionality comparing to standard ticket systems besides a possible reduction in cost.

Another case would be, for instance, an amusement park which decides to sell a certain amount of annual-passes. If you tokenize this pass and send it as an NFT to the customer, he has the option of transferring it to another person at any point. Hence, the customer would be able to rent his pass or even sell it once he does not want to go to the amusement park anymore. It is also important to state that the creator of the NFT (the amusement park in this case) would be able to establish a "creators fee" which is a percentage fee on each transfer of the NFT, giving him a percentage of the total price of the rent/sale of the item.

However, for me these are simple implementations of this technology since, as I stated before, it has no limits. In my opinion, one implementation which we may be able to see in the future and would change the world completely is the use of Blockchain and NFTs in democratic governments. NFTs are a perfect technology to perform secure, transparent and anonymous votes. It would not only eliminate any kind of human error at counting but also any kind of rumour regarding the legitimacy of the elections, since everything would be transparent. Moreover it would decrease a lot the cost of performing these elections. In Solana network it costs 0.00001 SOL to mint (create) an NFT, which is 0.0003774 € at current prices. After making some simple computations, a country like Spain, with a population of 47.35 M, would only spend 17890€ to perform state elections.

In order to put some context, Spain spent almost 139M € in the general elections of April 2019 [4]. This reduce in cost would not only make the country to save a lot of money but also give the opportunity to hold more elections or polls to allow citizens to decide on implementing or modifying new laws.

Even though NFTs are currently seen as inflation and speculation (which may be true

for collectibles NFTs) there is an immense future for their technology to develop.

2.4. Solana

All the information used to explain this section has been extracted from official Solana documentation [5] and Solana Whitepaper [6].

Solana is a decentralized Blockchain which is built around offering fast and cheap transactions. Currently is the network which has successfully make the most amount of transactions with a total of 77.500M of transactions. It current has a TPS (transactions per second) rate of 1000-3000 TPS (depending on network congestion) since it is still in beta phase but it aims for a 65000 TPS rate on its final and production version. Moreover it has the cheapest transaction cost between all the mainly used and relevant networks, with an average cost of \$0.00025 per transaction. This makes it really attractive to people who wants to build dApps (decentralized Apps) which are accessible for everyone and/or require a big amount of transactions.

2.4.1. Solana Network Design

Each time a new block is going to be made the network selects a Leader node that will create the first candidate block to be added to the network. The leader will create a Proof of History sequence and will execute the transactions he had as an input with the data copy or state he has stored obtaining a final state. After that it will publish the final state, the performed transactions and the signature to the verifiers (the rest of the nodes that weren't selected as leader this time). When receiving the information, each verifier will execute the same transactions with his own local state (his own copy of the database stored) and publish their signature confirmation with their final state obtained. Each of these final states published by the leader and validators count as a vote for the consensus algorithm and the selected final state by the algorithm will be the used to create a new block.

2.4.2. Proof of Stake

In order to ensure the data integrity and to prevent any malicious action from a node validator Solana uses a Proof of Stake algorithm for it block creation consensus. This algorithm will select the leader for the next block and punish any node trying to behave in a malicious way.

In order to participate on the consensus a node will have to stake SOL tokens. This stake consist of blocking and freezing the tokens and putting them as guarantee for your validator work. If a validator tries to make a malicious action, for example including a transaction that has not happened, it will lost all the SOL that it has staked. At the same time, each time a block is successfully created a reward, in SOL tokens, will get

distributed between all the validators that contributed to the consensus.

The Proof of Stake algorithm also takes into account the amount of SOL the validator decides to stake. Everything is calculated taking into account what percentage of SOL a validator has staked out of the total SOL staked by every node. The power vote in the consensus, the probability of being selected as leader and the reward distributed is directly proportional to the relative staked tokens amount. This dynamic makes the user and validators to make different groups or strategies to become the top validators. Validators will be explained deeply in a specific subsection.

Solana has an annualized emission of 8%, this emission rate will be reduced by 15% each year since launch until it reaches its final emission rate of 1.5%. All these token emissions are directly used as rewards to the validators when they successfully confirm new blocks as we explained before. This leaves an APR (annualized percentage rate) of 5,3% for staked SOL tokens with current emission rate (already reduced from initial one as mentioned above).

2.5. Solana Items

The objective of this section is to explain what different objects exist as well as how they work and interact with the Solana Network. The objective is not to have a really deep knowledge about the network objects but to understand the principles of the main items that were also used in the development of this project.

Accounts & Wallets

An account is a record in the Solana ledger which can store either data or an executable program. It would be equivalent to a bank account where you store your money but with the difference that a Solana account holds funds or tokens. In case it is an account destined for storing an executable program we can compare it with a file in Linux.

Each account has a key-pair assigned. A key-pair is formed by a public key and a private key. The public key is used to reference to the account, for example, sending funds to an account or running an executable program stored in an account. On the other hand, the private key is used to sign transactions or messages in order to make any kind of actions with this account. This would be equivalent to an email and a password, you share your email so anyone can send you an email but you need the password of the email to be able to make any action (read, send or delete an email) with that email.

A wallet is a device or an app that is used to store the key-pairs of different accounts. Wallets are mostly used by users to store the key-pairs of their accounts, so when we refer to a wallet we always refer to an account of a user (not an account that stores an executable program). There are different types of wallets from physical devices to mobile apps but all of them help users to make transactions and signatures with more security

and easiness.

The most popular and used wallet app for Solana is Phantom which supports both PC (through browser extension) and mobile.

Token Programs & Token Accounts

In order to understand how NFTs work in Solana we have to know how does the creation and storage of a Token work since at the end an NFT is a token with one single token supply.

In Fig.2.2 we can see simple schema about the different actors on a Token creation and storage.

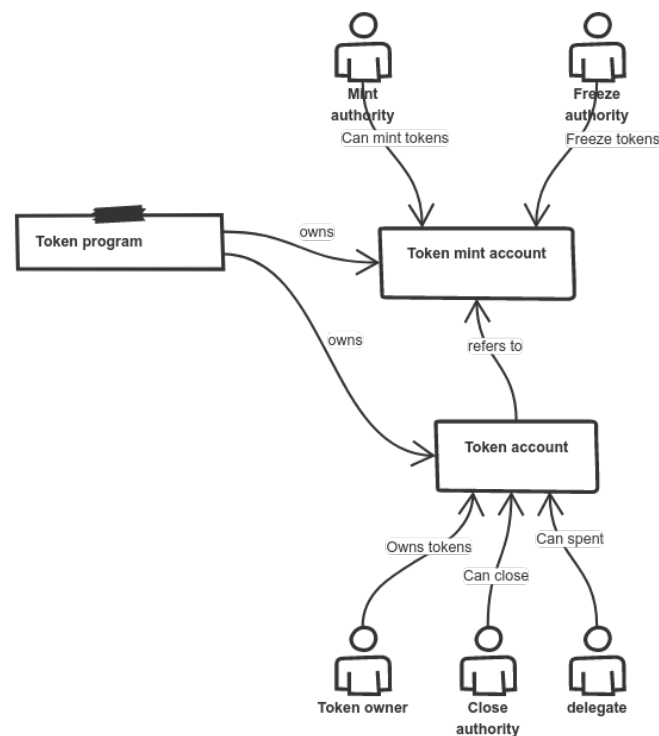


Fig. 2.2. Token Program and related actors [7].

The Token program is the program that defines the implementation of both fungible and non-fungible tokens. Solana provides itself and standardised Token Program and the only thing you have to configure is the data associated to that Token Program.

A Token Mint Account is automatically created each time someone wants to mint (creating and publishing the token info in the Blockchain) a token. It includes the Mint Authority which is a public key of a wallet authorised to mint the token (the one of the person doing the mint) and the Freeze Authority which is a public key of the wallet authorised to lock and freeze the tokens being minted (normally it is set to null since normally none can freeze tokens) [7].

Token Accounts contains the information about the tokens owned by a public key. Although it may seem confusing and non-sense, the Token Account is not owned by the Token Owner but by the Token Program. Token Program is the one which control the access to these tokens. The Token Account store also different data that tells the Token Program how to control the access to these tokens. The Token Owner is the public key that owns these tokens and who has the rights to send or spend these tokens. Token Owner can also give the rights to spend up to a certain amount of tokens, called `delegatedAmount`, to one or more public keys called delegates. Close authority is the public key allowed to close the account [7].

The cost of minting a token is 0.00001 SOL. This cost comes from of the cost of creating a Token Account. Solana Blockchain charges a monthly fee on accounts that are stored on the Blockchain. This amount is charged monthly and if the account does not have enough balance it is automatically closed. However, if the account holds the amount needed to pay for the fee for 2 years the account is not charged anything. This 2 years coverage amount is 0.00001 SOL. Hence, by sending (and losing) this 0.00001 SOL you can create a permanent Token Account and this is what is done through the minting process.

Instruction

An instruction specifies a single program. It can receive as inputs a list of accounts, the ones it will operate on and array of data with any kind of extra data needed (amount of money to send for example).

Signatures

Whenever a program wants to operate on an account (for example modify its data or reduce balance to send money) it will need the authorisation of the account in order to do it. In order to get this authorisation there will be generated a signature, which can only be signed if you have the private key of the account the signature has been made for. In this way it is ensured only the person who knows the private key of an account (the owner) can sign it.

Transactions

In order to execute a program we have to submit a transaction to the cluster. This transaction will include a list of accounts, a list of instructions and a list of signatures. The list of instructions will include all the instructions to execute in a defined order. Furthermore, the transaction will send to the Solana cluster the Account info of the accounts involved as well as the list of signatures. Solana network will check if there is a matching signature for each of the accounts included in the list. It is important to take into account

the fact that we can only pass accounts to the instructions through this account list and since Solana ensures all the accounts of the list have been correctly authorised there is no way to operate with unauthorized accounts. If at any point any instructions throws an error or it is missing a signature from of an account of the list, the transaction would fail immediately.

2.5.1. NFTs & Metadata in Solana

The concept of what is an NFT, which was explained previously, is the same across all Blockchains and systems, however, each Blockchain has a different design in order to create, transfer and store NFT data. In this subsection it will be explained how does an NFT work in Solana Network.

As already explained in previous section, NFT metadata is not directly stored in the Blockchain token account. This is pretty uncommon since most Blockchains (as Ethereum or Polygon) actually store all the NFT data (metadata included) in the NFT itself. Hence the approach to work with NFTs in Solana is completely different from any other chain.

The NFT refers inside its Token Account to a resource link where the metadata of the NFT is stored. In order to keep decentralization and not rely in a centralized resource, the most used approach to store metadata is using Arweave protocol. Arweave protocol is a decentralized protocol for storing files permanently (it will be explained in detail later).

In order to put some context it is also important to explain how are NFTs usually created in Solana network. NFTs in Solana (and in most Blockchains) usually are minted (created) as a part of a collection. A collection is a set of NFTs (each NFT is unique) which is minted through the same Token Program. Collections create a community around it, which is usually only accessible through by owning one of the NFTs. Collections normally grant access to different perks or privileges as services, events or merchandising.

2.5.2. Validators

The concept of a Validator was already introduced in previous sections, nevertheless it is important to have a deeper view of them. A Validator is any kind of node that has any amount SOL staked in order to participate in the consensus validation of transactions, and win rewards in SOL tokens in exchange.

By default, any person can make a call to the Solana network API in order to get any information stored in the Blockchain. However, Solana is still in beta version and has been recently suffering multiple congestion and speed problems. Whenever the Solana network is suffering a bit of congestion or speed problems it blocks any kind of petition made to the main Solana API.

Nevertheless, this is not as bad as it seems since Validators are nodes of the Blockchain

and, in consequence, they have a replica of the data stored in the Blockchain. In order to have an always operational service which depends on making calls to the Blockchain you need to send this calls to a validator which will always be available. In order to do this you have two options, make your own validator (which is currently pretty difficult due to the resource requirements) or to contract a service from a validator which allows you to make calls through it.

These leaves two principal income streams for the Validators. On the one hand, staking platform service. Since you need a validator in order to stake your SOL, all validators offer the possibility to stake with them and charge you a percentage of your profit as a fee for the service. On the other hand, validators can also offer the service of making request to the Blockchain through them (which usually improves a lot the ping, and ensures a permanent availability).

In this second scope it is really important the ping your validators manage to have with the Blockchain and here the election of node leader takes importance. As it was explained before, the transactions performed on a new block are the ones set by the leader node. This is the reason why all validators fight to increase the possibility of becoming leaders as this would increase the possibility of their transactions getting executed and, in consequence, reducing the ping or delay a transaction took since the user made it. The Validators which focus on having the higher rank validator usually charges really low or even zero fees to the users who stake with them since the more SOL is staked with them the more probability they have to be selected as leader. This leaves a really competitive market between Validators which makes the user product better and cheaper what is essential since at the moment the resource requisites to make your own validator are high.

2.6. Arweave Protocol

Arweave is a protocol which was created in order to make a decentralized network where you can upload and store data permanently for the first time [8].

Due to this reason, most Solana apps or dApps decide to use Arweave as the place to store their data, metadata in the case of NFTs, in order to keep all the environment decentralized, which is the main objective. In order to the metadata be accessible, a JSON file with all the information can be uploaded to Arweave system and it will become accessible from the link generated by Arweave. This link is the one referenced in the NFT information that is stored on the Blockchain.

It is also essential to have available and use this type of protocols as support in the Web3 ecosystem in order to maintain the decentralization at all levels.

2.7. Web Application Frameworks and Tools

A web development framework is a set of resources and tools for software developers to build and manage web applications, web services and websites [9].

In web development the two most popular terms are Front-End and Back-End. The part of a website that the user interacts with directly is termed the Front-End. It is also referred to as the visible side of the application [10].

Back-End is the server-side of the website. It stores and manages data, and also makes sure everything on the client-side of the website works fine. It is the part of the website which the user cannot see or interact with.

In this sections the different frameworks, libraries and tools used for the development of the application (both Front-End and Back-End) will be explained.

2.7.1. Next.js (Full-Stack Framework)

Next.js is an open-source Full-Stack (includes both Back-End and Front-End) React framework which was developed by Vercel, a web hosting company. This framework is built on top of Node.js (a JavaScript runtime environment) and allows adding different extra features or tools to React applications.

The main difference with other frameworks and the main reason of its popularity and success is that it allows to use Server-Side-Rendering. Server-Side-Rendering technique consist of rendering the content of the page in the server side instead of the client side generating static sites.

Using already rendered and static sites has a lot of benefits. It can improve a lot the loading time of the pages without mattering the resources of the client. With Client-Side-Rendering if the client has low resources the can take a lot more to load depending on the workload of the rendering. Moreover it is a good practice for web positioning, it can improve the position of your web.

The fact that the server sends static sites makes the content completely and easily readable for the search engines while part of Client-Side-Rendered websites content can be lost for the search engines due to them not being able to render the content. Thanks to this and a faster load of the content sites with Server-Side-Render obtain a better score on the search engines.

Next.js manages the routing of your application itself, based on the file directory structure. Moreover it includes an API route server-side exclusive. It allows to create all the Back-End system and expose an API (Application Programming Interface) that can only be accessed during Server-Side-Rendering making it totally unreachable from the client and, in consequence, private and secure.

In the Fig.2.3 we can see an example of the routing of a Next.js app. Pages folder is

the root folder for the routing of the app and is exposed as '/' in the routing. Each folder will create a sub-route ('/foldername') with no limitation of nesting folders and routes. Each file represents a route of the application, and the name of the file will be the exposed of the application (besides index.js which will be always used for '/' route).

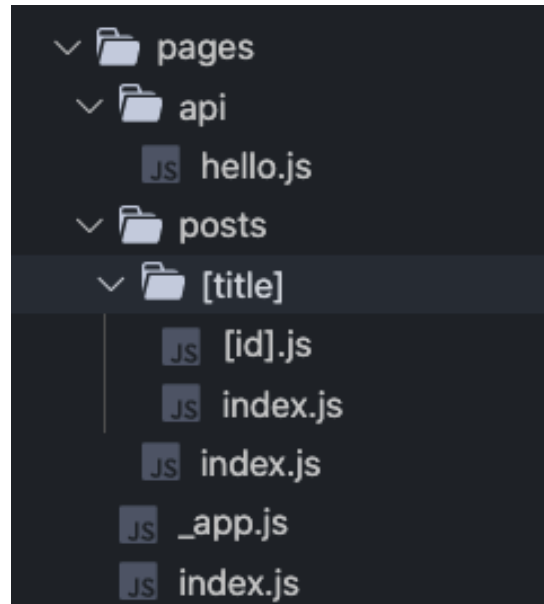


Fig. 2.3. Routing structure in a Next.js App.

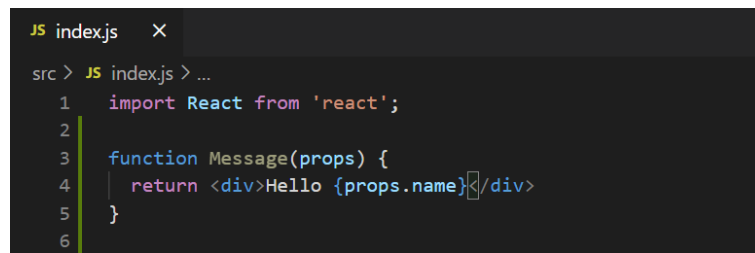
2.7.2. React

React is an open-source Front-End JavaScript library. It is run by Facebook and open-source developer community.

React was created with a single focus: to create components for web applications. A React component can be anything in your web application like a Button, Text, Label, or Grid [11].

Another feature which makes React more attractive and easy to learn is the fact that it is possible to use React in existing apps. React was designed keeping this in mind. It is possible to change a small part of an existing application by using React, and if the change works, then start converting your whole application into React.js. Facebook used the same approach [11].

In Fig.2.3 there is an example of a React component. Message is the component that accepts props (input) and returns JSX. JSX is a special syntax which looks like HTML, which converts React's API calls and finally renders HTML [11].



```
JS index.js  X
src > JS index.js > ...
1  import React from 'react';
2
3  function Message(props) {
4    return <div>Hello {props.name}</div>
5  }
6
```

Fig. 2.4. React component.

2.7.3. NodeJS

Node.js is an asynchronous event-driven JavaScript runtime. It allows to execute JavaScript programs outside a browser. Node.js is designed to build scalable network applications. In Node.js many connections can be handled concurrently [12].

This is in contrast to today's more common concurrency model, in which Operating System threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. A lock is an event that happens when the execution of additional JavaScript in the Node.js process must wait until a non-JavaScript operation completes. This happens because the event loop is unable to continue running JavaScript while a blocking operation is occurring. Almost no function in Node.js directly performs I/O (Input / Output), so the process never blocks except when the I/O is performed using synchronous methods of Node.js standard library. Because nothing blocks, scalable systems are very reasonable to develop in Node.js [12].

2.7.4. MongoDB & Moongose

MongoDB is an open-source No-SQL (non relational) database. No-SQL databases do not store data in relational tables which allows to work with different data structures. No-SQL databases are usually the best option when working with a large sets of distributed data.

MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use [13].

In Fig.2.5 there is an example of the structure of a document from a MongoDB database.

```

{
  _id: 57cd1591c43f2b5670c35e3a,
  cycle_id: 10722,
  cycle_number: 61,
  product: EG55051001,
  source_file_number: 150824,
  day_part: morning,
  date_time: 2015-08-24 09:55:23,
  day_name: Monday,
  machine: KM69,
  process_parameters: {
    cycle_time: 10.47,
    injection_time: 0.9,
    max_inj_press: 1237,
    ...
  },
  alarms_I: {
    A_00064: true
  },
  alarms_II: {
    A_00142: true,
    A_00145: true
  }
}

```

Fig. 2.5. MongoDB Document.

Mongoose is a Node.js library which allows to work with MongoDB with ODM (Object Data Modeling). Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box [14].

The benefits of using Mongoose are to work with an Object model which will ease the relation between the data stored and the application as well as ensuring all the data stored in the database always follows the pattern designed. It allows to obtain some of the main benefits of SQL databases while keeping the scalability and rest of benefits of No-SQL databases, MongoDB particularly.

In Fig.2.6 there is an example of the definition of a Mongoose Schema. As it can be appreciated it is possible to define the different fields of an Object as well as the type of value and other extra parameters (as default value for example) [15].

```

const mongoose = require('mongoose');

export const TrackEntrySchema = new mongoose.Schema({
  keyword: {
    type: String,
    minlength: 1,
    maxlength: 100,
    required: true,
    unique: true
  },
  value: {
    type: Number,
    min: 0,
  },
  trackEntryId: {
    type: Number,
    min: 0,
    required: true,
    unique: true
  },
  type: {
    type: String,
    required: true
  }
});

```

Fig. 2.6. Mongoose Schema.

2.7.5. OAuth

OAuth (Open Authorization) is an open standard authorization framework for token-based authorization on the internet. OAuth enables an end user's account information to be used by third-party services without sharing the account credentials to the third party. Moreover in a OAuth connection the provider can determine which part of the data to be exposed to the third party and not share all the information of the user [16].

Normally the scope of the connection at the type or part of data accessible for the third party is selected by the user or if its defined by the third party. In case the scope is defined by the third party, normally the user is aware of it and needs to accept the scope conditions in order to log in.

2.7.6. Next-auth

Next-Auth is a JavaScript library designed to implement a secure authentication system for Next.js applications. It allows both to authenticate through existing OAuth providers (Google, Discord, Twitter) or to add custom credentials authentication.

Next-Auth ensures an implementation of a secure authentication system through different mechanisms [17]:

- Uses Cross-Site Request Forgery Tokens on POST routes (sign in, sign out).
- Default cookie policy aims for the most restrictive policy appropriate for each cookie.
- When JSON Web Tokens are enabled, they are encrypted by default (JWE) with A256GCM.
- Attempts to implement the latest guidance published by OWASP (Open Web Application Security Project) Foundation

2.7.7. Discord & Discord OAuth2

Discord [18] is a free-to-use text, voice and video chat app. It was firstly designed focusing and aiming to gaming communities although it has become a popular choice for communities of all thematics as streamer or influencer communities, group of friends, developer communities or NFT communities (which are the focus of this project).

It allows to create different chat channels (text channels) or voice channel inside a Discord server. Furthermore, the privacy of the server, channels and everything can be configured through a role based system controlling the access of each user to different channels depending on the user roles.

Discord allows to develop customized Discord Bots that can be added to any channel to automate different type of process as role assignment, message automation or user moderation.

Discord also have an OAuth system, called Discord OAuth2 which allows you to implement OAuth login through Discord in any application. To do this you only have to sign up in Discord for Developers site and create a OAuth Discord Bot (already predesigned by Discord). You will have to configure the different scopes (fields of the user data) that you want to access to when a user logs in and Discord will then give you different keys to implement the OAuth system on your application.

In Fig.2.7 we can see the structure of a Discord server and how it is displayed to the user.

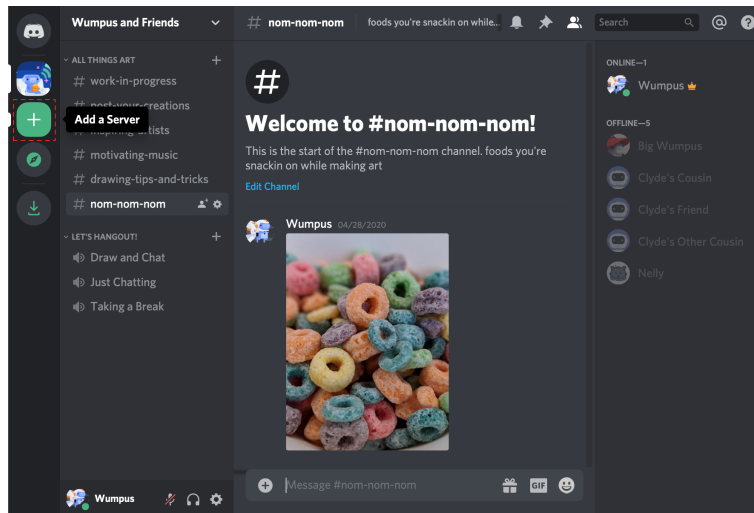


Fig. 2.7. Discord Server [18].

2.8. Express.js (RestAPI)

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web applications and APIs (Application Program Interface) [19]. It offers a myriad of HTTP utility methods and middle-ware, which makes creating a robust API quick and easy.

Express.js is one of the best options to build a RestAPI on Node.js. A RestAPI is an API that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services [20]. REST is a software architecture that defines a set of rules to design web services. It uses http calls to communicate between the client and the service, most of the times using JSON to send messages, and follows the http standards.

3. ANALYSIS OF PRODUCT

In this chapter the product and its functionalities will be defined and described. After a general description of the application, the different functionalities will be separated and organized in modules. Finally, both the functional and non-functional requirements will be defined so that it can be tested and ensured that all the functionalities work as defined after implementation

3.1. General Description of the Application

The product will be a web application that will allow user to connect multiple wallets from Solana and link a Discord application. Once the user has connected multiple wallets and/or his Discord account he will be able to log in with any of the connected accounts (whether wallets or Discord account). Once logged the user will access to his full-account (taking in consideration all connected accounts) functionalities and future services.

Users usually have more than one wallet. The most common wallet set up for an user is to have one "Cold Wallet" , one "Hot Wallet" and a "Burner Wallet". A Cold Wallet is a difficult accessible but high secure wallet where users stores his most valuable assets, which he do not plan to move short-term. for example a ledger Wallet. A Hot wallet is a easier accessible wallet which stores the assets that user plan to move short-term and is used to transact in trusted-sites. A Burner Wallet is a single-use wallet used when interacting with a not 100% trusted site. User only send the strictly necessary assets in order to avoid any bigger loss if the site is malicious. After it has been used it is usually deleted by the user (is not deleted on-chain but the user never uses it again).

In the Solana ecosystem there are multiples dApps that offer different kind of services but most of them have a common problem. All of them require to access with a wallet in order to be used which has two main cons:

- The user has to constantly connect both his Hot an Cold Wallets in order to access the services, which exposes a big vulnerability for the user. All the successful dApps are used by malicious attacker which replicate their interface and use them in order to perform phishing attacks. If user are required to connect their Cold or Hot Wallets in the real application they will easy fall for connection them in a malicious site after a phishing attack.
- If the user has his assets distributed between multiple wallets, which is pretty common, he will not be able to access to all his functionalities at same time, as most of this sites only support a single wallet log in.

The main objective of the application is to be the authentication layer of other services (related to NFTs) which will be built on top in the future. It will allow users to only connect their wallet/s only once, and then even eliminate the permissions of the application in all his wallets. In this way users will be able to access through Discord or a Burner Wallet ensuring a 100% safe connection and prevent any kind of phishing attack. At same time users will be able to access to all the perks of his assets at same time since the assets inside all his linked wallets will be taken into account.

Moreover, it will have a private API (not accessible from main application server) which will be used to obtain different information of the users and NFT collections. This data will be really useful for collection owners. The objective of this API is to whether sell the access to data retrieved from it to collection owners or connect it to different services that may be built in the future.

The functionalities of the application will be divided in different modules.

3.2. Wallet Ownership Verification module

The objective of this module is create a system to verify if an user is the real and only owner of a determined wallet without the necessity of knowing the private key of the wallet. It has been decided to only work with public keys and not knowing or storing any type of sensible data. Although the security of the application will be a priority all the time the best way to protect users is to not store or access data that would make them vulnerable if it gets leaked.

The wallet software that has been decided to be supported and integrated in the application is Phantom. Phantom is the most popular wallet used in Solana. It makes easy and secure to store wallet keys, send or swap tokens on Solana. Moreover it has both a PC browser extension and mobile version which makes possible to any type of user to use it on the application.

This system will be used each time an user tries to login through a wallet or tries to add or link a wallet to an existing account (when the user is already logged).

Also, in order to avoid any kind of reply attacks and impersonating of malicious the system will generate a one-single-use nonce for each verification. A nonce is a random or semi-random number that is generated for a specific single use. It is related to cryptography communication [21].

The user will have to sign a message that will include the generated nonce. Through asking for a signature we ensure the user has access to the key-pair (both public and private key) meaning he is the real owner of the wallet.

Finally the system will check if the nonce signed by the user and the generated by the system matches (discarding reply attacks). If so, it will return the public key of the , now verified, wallet.

3.3. NFTs Data & Metadata Extraction module

This module will be in charge of extracting all the data of the NFTs owned by an user and the related metadata to the NFTs both for the Verified Collections & NFTs and the non-verified NFTs. After extracting the data, the module will return it in an structured way so that the Data Visualization module can read it afterwards.

The process of obtaining the data will consist of, firstly, obtain the wallet public key list from the user. Subsequently, the system will obtain the token list of each of the wallets from the list. It will filter out the NFTs from the standard tokens.

Once the list of NFTs is obtained, they will be classified between verified NFTs and non-verified NFTs. The module will obtain the data of the verified ones from the database, where it is already stored. The data and metadata from the non-verified will be obtained from Solana Blockchain network.

However, a non-verified NFTs can be malicious. In order to avoid references to malicious sites or files only the NFTs which use Arweave protocol for storing their metadata will be included.

Finally, the extracted data will be organized by collections and returned in a structured JSON so that it is easy and clear readable by Data Visualization module or any other module/service that may use it in the future.

3.4. Session Access Control module

The objective of this module is to create and control the sessions of the users. At same time it will save, edit or delete the user stored information in the database.

Users will be able to login with a Solana Wallet or with an existing Discord account. In order to login through a Solana wallet, the user will have to verify his wallet through Wallet Ownership Verification module using Phantom. If the verification success the returned public key will be the used to create the session. In order to login with Discord the system will call Discord OAuth2 API and use the Discord account information for the session. Moreover, the system will also be able the eliminate the current session from client whenever he decides to log out.

Whenever an user logs in with a wallet or a Discord that is not stored in the database, it will be automatically registered and his user data will be stored.

Once an user is logged in, he will be able to connect extra wallets to his user-account. In order to do so, a similar process as login is done. The user will have to verify his wallet through Wallet Ownership Verification module using Phantom. However, this time if the verification is successful the returned public key will be used to be stored in the system database. When a wallet is stored on the database it will be linked to his owner user. In case the wallet was already linked to an user, it will be removed from the old user and will

be linked to the user who has just verified it.

The user will be able to eliminate the different wallets linked to his account. In case he eliminates the wallet which he used to log in, he will log out.

However, due to some incompatibility with Discord OAuth2 API , which is the only way to add Discord login to an application, the user will not be able to link Discord to an existing account. Nevertheless, the system will include an extra functionality to simulate this. This feature will be called "import account".

If an user which has only used to login and linked Solana wallets (he has not linked Discord yet) wants to link his Discord account what he will have to do is to log out and log in with the Discord account. After this, he will be able to the import account feature where he will have to verify a wallet (whichever linked wallet from the account to import). If the verification process success, all the wallets from the account to import will be automatically linked to him.

It is important to only allow this for accounts which do not have a Discord account already linked in order to avoid impersonating. In the case the account to import has already a Discord account, only the wallet that has just been verified would be imported obtaining the same result as if the user had only added the wallet.

Finally, it is essential to keep the integrity of the user data. To do so a wallet can be linked to only **one** user, and at same time a Discord account (identified by its discordId) can also be linked to only **one** user. At same time there must be only **one** wallet for a specific public key since public keys are unique.

3.5. Data Visualization module

This module will be one in charge of displaying the data, obtained from NFTs Data & Metadata Extraction module, in the application. It will receive an input with the data to display. The system must be dynamic according to this since the data extracted from an user may differ a lot depending of the user.

It will display the data and the metadata of the NFTs, which will be given by this input. The NFTs will be organized by collections.

Finally, the system will also include the basic data of the collection. For verified collections, it will also include extra data as quantity holded and vote power.

3.6. Verified Collections Management module

The objective of this module is to manage the verified Collections of the application. On the one hand, it will insert the new verified collections and their NFTs. On the other hand, it will be able to extract collection holder information. Firstly, obtaining the on-chain

information, and then mapping it with database user information to get some extra really valuable data, as collection holders organized by Discord accounts.

In order to insert a new verified collection, the system will receive as input the basic data of the collection (size, max power of vote, name and description) and its Candy Machine Id. Candy Machine is the open-source software used to mint (create) collections and NFTs in Solana Blockchain. It creates a Token Program which will be the one controlling the minting process of tokens, the process is deeply explain in section 2.5. The Candy Machine Id refers to the public key of the generated Token Program and can be used to obtain all the NFTs minted trough that Token Program, which are all the NFTs of the collection.

The system will obtain the list of NFTs of the collection from Solana Blockchain using Candy-Machine Id. It will then extract the data from each of the NFT using the NFTs Data & Metadata Extraction module. Finally the NFTs will be inserted in the database as verified-NFTs.

Moreover, the system will be able to obtain the list of holders of a certain collection. It will verify on-chain the public key which is holding each of the NFTs of the collection. Afterwards, it will organized the obtained public keys in a list which will specify which NFTs do each public key own.

Finally, there will also be the possibility of obtaining the Discord accounts holding at least one of the NFTs and which NFTs do they hold. This will be done by mapping the previous public key dictionary with the database. Only the public keys which are registered in the database will be included. Thereafter, they will be organized by users which has Discord linked (those with no Discord will be omitted). Finally some extra holder information will be included, as the quantity being holded and the power of vote.

3.7. Functional Requirements

In this section the functional requirements of the application will be described.

A functional requirement is a requirement that the system must do in order to operate successfully and accomplishing all the goals defined. It can be understood through reviewing the inputs and outputs of the system. In functional requirements it is specified what the system must do when receiving specifics inputs and what it must output [22].

In order to perform an organised analysis, the requirements will be grouped by module and they will have an unique identifier. The identifier will be formed by a root which will be FR (functional requirement), followed by a module identifier, which will be common for all the requirements of same module, and an unique identifier at the end. The identifier pattern, where XX is the unique identifier, for each module will be the following:

- **FR_WALLET_VER_XX** for Wallet Owner Verification module

- **FR_NFT_EXTRACT_XX** for NFTs Data & Metadata Extraction module
- **FR_SESSION_XX** for Session Access Control module
- **FR_DATA_VIS_XX** for Data Visualization module
- **FR_VER_COL_XX** for Verified Collections Management module

3.7.1. Wallet Owner Verification module

Identifier	Requirement Value
FR_WALLET_VER_1	The system will support Phantom wallet
FR_WALLET_VER_2	The system will support wallet owner verification through Phantom
FR_WALLET_VER_3	The system will be able to generate a random nonce
FR_WALLET_VER_4	The system will be able to send the specific nonce to only the user which is attempting to verify
FR_WALLET_VER_5	The system will generate a new and random-generated nonce for each verification attempt
FR_WALLET_VER_6	The system will generate a message containing the nonce set by then nonce recieved from client-side
FR_WALLET_VER_7	The system will ask the user to sign the generated message in order to create a signature
FR_WALLET_VER_8	The system will be able to obtain the signature performed by the user
FR_WALLET_VER_9	The system will check if the signatures message includes the nonce generated for the verification attempt
FR_WALLET_VER_10	The system will verify a wallet only and only if there is a match between the client-side and the server-side nonce
FR_WALLET_VER_11	The system will return the public key of the wallet when a wallet has been successfully verified

TABLE 3.1. FUNCTIONAL REQUIREMENTS WALLET OWNER VERIFICATION MODULE

3.7.2. NFTs Data & Metadata Extraction module

Identifier	Requirement Value
FR_NFT_EXTRACT_1	The system will receive as initial input the user to obtain NFT data from
FR_NFT_EXTRACT_2	The system will obtain wallet public keys list of the user from the database
FR_NFT_EXTRACT_3	The system will obtain the list of tokens held by the wallets in the wallet list
FR_NFT_EXTRACT_4	The system will filter out the NFTs from the list of tokens obtained
FR_NFT_EXTRACT_5	The system will classify the obtained NFTs between verified or non-verified
FR_NFT_EXTRACT_6	The system will retrieve the data and metadata of the verified NFTs from the database
FR_NFT_EXTRACT_7	The system will retrieve the data and metadata reference of the non-verified NFTs from Solana network
FR_NFT_EXTRACT_8	The system will eliminate the non-verified NFTs which does not use Arweave for metadata referring
FR_NFT_EXTRACT_9	The system will extract the metadata from non-verified NFTs which use Arweave for metadata referring
FR_NFT_EXTRACT_10	The system will output the extracted data through a structured JSON object so that it is easy readable by other modules

TABLE 3.2. FUNCTIONAL REQUIREMENTS NFTS DATA & METADATA EXTRACTION MODULE

3.7.3. Session Access Control module

Identifier	Requirement Value
FR_SESSION_1	The system will allow users to login through Phantom wallet.
FR_SESSION_2	The system will allow users to login through Discord login.
FR_SESSION_3	The system will allow users to log out.
FR_SESSION_4	The system will allow users to connect multiple wallets to their account.
FR_SESSION_5	The system will be able to store and update the information of a logged user.
FR_SESSION_6	The system will allow users to import one or multiple wallets from other account, if that account is not linked to Discord.
FR_SESSION_7	The system will use Wallet Owner Verification Module whenever the user logs in through Phantom, add or import wallets in order to ensure he is the real owner.
FR_SESSION_8	The system will insert, edit or delete database data in order to have a record of the user information.
FR_SESSION_9	The system will maintain the integrity of the user and wallet data of the database.
FR_SESSION_10	When importing accounts the system will check if the account getting imported is not linked to a Discord user.
FR_SESSION_11	When importing accounts if the user who is getting imported has no Discord account linked to it, all his wallets will get imported to the new user.
FR_SESSION_12	When importing accounts if the user who is getting imported has a Discord account linked to it, only the wallet which has been used for the import will be imported.
FR_SESSION_13	The system will be able to delete (disconnect) wallet from an user when he desires so
FR_SESSION_14	The system will automatically log out an user if he deletes the same wallet he used for log in.

TABLE 3.3. FUNCTIONAL REQUIREMENTS SESSION ACCESS
CONTROL MODULE

3.7.4. Data Visualization module

Identifier	Requirement Value
FR_DATA_VIS_1	The system will be dynamic according to the input it receives
FR_DATA_VIS_2	The system will display the content and data of each NFT received from input
FR_DATA_VIS_3	The system will organize these NFT content and data by collections
FR_DATA_VIS_4	The system will include information of the collection
FR_DATA_VIS_5	The system will include the user holder information, quantity and vote power, in each displayed verified collection

TABLE 3.4. FUNCTIONAL REQUIREMENTS DATA
VISUALIZATION MODULE

3.7.5. Verified Collections Management module

Identifier	Requirement Value
FR_VER_COL_1	When adding a verified collection, the system will receive as input the data of the collection and its Candy Machine Id
FR_VER_COL_2	The system will check if the collection is not already inserted in the database
FR_VER_COL_2	The system will insert the collection into the databases, if it is not included in the database
FR_VER_COL_3	The system will extract all the NFT belonging to the collection from Solana network using the Candy Machine id.
FR_VER_COL_4	The system will obtain the data and metadata of the NFTs that belong to the collection through NFTs Data & Metadata Extraction module.
FR_VER_COL_5	The system will insert each NFT, now verified, in the database
FR_VER_COL_6	The system will be able to obtain the list of public key that holds at least one NFT from a determined collection.
FR_VER_COL_7	The system will include in this list which NFTs each of these public keys hold
FR_VER_COL_8	The system will be able to obtain the list of application users that holds at least one NFT from a determined collection.
FR_VER_COL_9	The system will map the public key list data with the user data from the application database, taking into account only users that have a Discord account linked.
FR_VER_COL_10	The system will return a list of the discordId, of the application users which holds at least one NFT.
FR_VER_COL_11	The system will include in the discordId holder list which NFTs each user holds and two extra parameters, quantity holding and power of vote.

TABLE 3.5. FUNCTIONAL REQUIREMENTS VERIFIED
COLLECTIONS MANAGEMENT MODULE

3.8. Non-Functional Requirements

In this section the non-functional requirements of the application will be described.

A non-functional requirement is a requirement that describes how the system must work. Non-functional requirements define how to deliver an specific functionality, while functional requirements define the functionality itself. They do not have any impact of the functionality of the system but they do in its performance. They focus on things as usability, accessibility or availability and security [22].

In order to perform an organised analysis, the non-functional requirements will be grouped by scope (usability, accessibility or security) and they will have an unique identifier. The identifier will be formed by a root which will be NFR (non-functional requirement), followed by an scope identifier, which will be common for all requirements of same scope, and an unique identifier at the end. The identifier pattern, where XX is the unique identifier, for each module will be the following:

- **NFR_ACCESSIBILITY_XX** for accessibility related non-functional requirements.
- **NFR_USABILITY_XX** for usability related non-functional requirements.
- **NFR_SECURITY_xx** for security related non-functional requirements.

Identifier	Requirement Value
NFR_ACCESSIBILITY_1	The application will render all the dynamic data on server-side in order to require the least resources possible for the user
NFR_ACCESSIBILITY_2	The application will optimize load speed as it maximum
NFR_ACCESSIBILITY_3	The application will make use of a private validator to connect to Solana network in order to reduce ping and improve load speed
NFR_USABILITY_1	The interface will include a navigation bar in all the sections of the application so that user can navigate freely at any time
NFR_USABILITY_2	The interface will include information about which section of the application the user is located so that the website is easy to navigate
NFR_SECURITY_1	The system will only store public data (as public keys) that will cause no risk for the user in case it get compromised
NFR_SECURITY_2	The Verified Management Collection module will be deployed in an completely independent environment from main application
NFR_SECURITY_3	The Verified Management Collection module will be only accessible by administrators of the application

TABLE 3.6. NON-FUNCTIONAL REQUIREMENTS

4. DESIGN OF PRODUCT

In this chapter the design of Product will be explained. The technologies, framework and methodologies in order to create a solution that fits the defined requirements will be defined and explained.

4.1. General Functional Architecture

In this section the general functional architecture will be explained.

First of all, Next.js will be the framework which the application will be developed in. It has been decided to use this framework since it allows to develop both the Front-End and Back-End of the application in the same framework and also allows to divide its Back-End in different modules (as the product was initially described) allocating a different API sub-route for each module.

Moreover it improves the user experience because it enable to perform the rendering on the server-side, speeding up a lot the loading times for the user. Furthermore, it allows to work with dynamic data in the Front-End thanks to it uses React.

As explained, each module will have its own API sub-route with another sub-sub-route for each of the functionality or methods of the module itself.

It has also been decided to use a different and completely invisible environment from the Next.js server for the Verified Collection Management module. This module is meant to be only accessible by admins of the application. Including it in an isolated and not visible from Next.js makes it the most secure since even there exist any vulnerability in the Next.js application the potential attacker will have no way to reach this server.

Since this module is planned to be used by other services that may be implemented in the future it will be implemented as an Express.js RestAPI which will allow these future services communicate with it in a easy way.

Finally, in order to store all the user and collection data, MongoDB has been selected as the database. It allows to work with large amounts of information, which is the case. It will be used along with Mongoose library, which enables working with MongoDB with an Object and Schema-Based way which will easy all the process while ensuring all the defined schemas are being followed.

The following schema in Fig.4.1 defines the general architecture of the system.

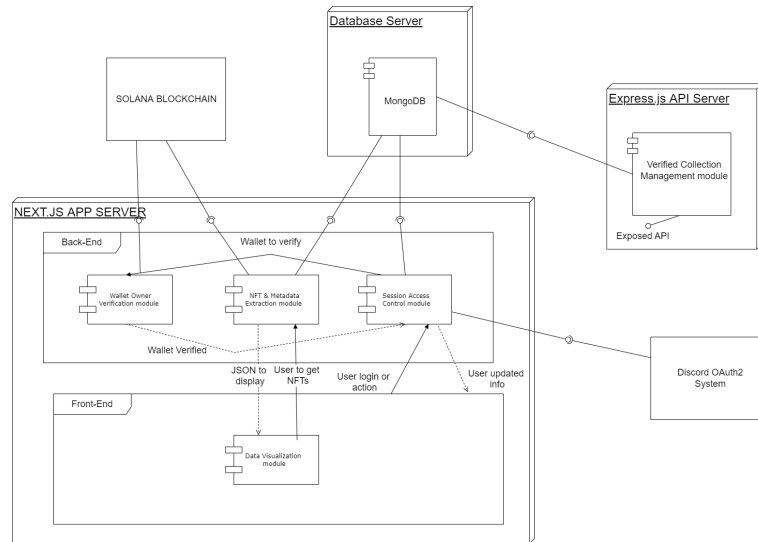


Fig. 4.1. Architecture Diagram.

4.2. Wallet Ownership Verification module

There are many libraries that implement login and wallet connection through Phantom. However, after making some research all the ones that were reliable and most used did not verify on-chain (through Solana Blockchain) if the user is actually owner of the public key. They would just take the public key value stored on the Phantom app (mobile version) or extension (PC version) and this value could get changed by a possible attacker. It would not be a highly critical problem since they would never be able to interact with Blockchain and steal funds since they would need the private key to do so (which they cannot know). However, they would still be able to access the application as an impostor of a real user and accessing their private data. Although, as already explained, only not sensible data is stored, this data must remain private.

In consequence of this problematic, it has been decide to implement an own-built solution that will verify if the user is actually the owner of the wallet on-chain.

In order to do this the solution chosen has been to make the user to sign a message with a nonce auto-generated by the Back-End and then check on-chain that the signature produce contains this nonce.

A nonce is a random or semi-random number that is generated for a specific single use. It is related to cryptography communication and information technology [21]. There are multiple uses cases for nonce but in our case it will be used for authentication. In this case the server will generate a nonce and send it to the client. The client will not only need to send the correct credentials but also the nonce that has just been generated.

In our use case, sending the correct credentials refers to signing the message, what can only done if you have access to the private key (you are the real owner) and including this nonce in the message signed would refer to the part of sending back the nonce to the

server.

The server-side will check if the signature has been correctly executed and will check if the nonce matches the expected one. If so, it will get the public key out of the signature made on chain and the wallet would be verified and its public key returned. If there is any type of problem or the nonce does not match and error would be thrown cancelling the verification of the wallet.

4.3. NFTs Data & Metadata Extraction module

There already exists different APIs from NFT marketplaces or other services that allow you to obtain the NFT data & metadata out of a public key. However, these APIs are not allowed to be used for production or commercial uses, which may be the case for this application in the future. In consequence, this module implements an own-built solution.

The public key list to extract NFT data from will be obtained from database. The wallet list linked to the user will be retrieved.

The solution approached has been to firstly obtain the token data from a public key (or a list of public keys if the user has more than one wallet linked) through Solana API. After obtaining this token list, the NFTs will be filtered by checking the tokenAmount which must be 1, the tokenDecimals which must be 0 and the tokenSupply which must be 1 too. After this process we would have the list of NFTs of an specific user.

The NFTs are classified in the application between verified and non-verified NFTs. The verified NFTs will be the ones from the collections previously verified and added to the database and the non-verified ones will be the rest, those not included in the database.

Regarding the data sent to the Data Visualization Module will be a JSON object containing a list of collections, with the data from the collection and list of owned NFTs and their metadata and a list of non-verified NFTs. This object will be obtained out of the list of public keys owned by a user.

Firstly, the token data from a public key (or a list of public keys if the user has more than one wallet linked) will be obtained through Solana API. After obtaining this token list, the NFTs will be filtered by checking the tokenAmount which must be 1, the tokenDecimals which must be 0 and the tokenSupply which must be 1 too. After this process we would have the list of NFTs of an specific user.

In order to extract the metadata of NFT in the obtained list the module will use a different procedure depending on whether the NFT is a verified one or not. It will be firstly verified if the NFT is included as verified NFT in the database and return its stored metadata if so. If it is not verified the on-chain associated data will be retrieved through Solana API. As previously stated, the Solana Blockchain does not store the metadata on-chain and it only stores a link referring to it. This can be dangerous and make our application vulnerable as it is possible to create an NFT that refers to a link that executes

an script. This could make our application vulnerable for possible attacks. In order to avoid this risk it will be checked if the link refers to Arweave Protocol domain which will ensure it is safe to use.

Finally, a JSON object with the previously defined structure will be returned. This JSON object will be the one used by the visualization module to display the NFTs in the application.

Regarding the insertion of verified collections and NFTs. The Collection & NFT Insertion Module will also call this module to obtain and save the metadata of the NFTs with the only difference that the metadata reference (if it is stored on Arweave Protocol or not) will not be checked since before executing an insertion, the admin has already checked if the NFTs do not refer to a malicious link.

4.4. Session Access Control module

The objective of this module is to create and implement a Session Control and Management system for our users. Next-auth.js library has been selected to implement it since it is a completely secure as well as easy to implement solution for Next.js applications.

Two modes of login will be implement, login through Discord and login through Solana wallet. Since the application will need and use both data from Discord and Solana users will have the opportunity of login to their account through Discord and Solana Wallet as well as linking a Discord account or multiple wallets to an existing account.

Enabling Discord login it is not only important to retrieving data from Discord user, which will be used in other modules, but also ,the most important thing, to enable to login the user to his account without connection to Solana or even with Phantom not installed in the specific browser/mobile.

One of the best practices is to prevent any kind of fishing attacks in Blockchain it to have a separate device/browser where you have installed all the tools and software needed to interact with Blockchain. Inside this secure environment it is also important to access to known and safe websites through direct links and always make the Blockchain related transactions here and only here as well as using this environment only for this.

This will heavily protect the user since in case he clicks a malicious link or falls for a phishing attack the attacker will not have access to all this sensible information since it is not saved inside this environment.

Enabling Discord login will allow the users to only need to connect to the application from the "safe" environment once. After linking all his wallets he will be able to access to the application and access to all the functionalities just by logging with Discord even from an "unsafe" environment with no risks, which is the main purpose of this application.

Next-auth.js has already a built-in functionality to connect with Discord OAuth2 and enable logging with Discord. Since the only needed data from the user is his account info

the Discord OAuth2 has been set up to only prompt identity scope.

Regarding the Solana Wallet implementation, the custom credentials option from next-auth.js will be implemented. In this case when user log in with a Solana Wallet, the Wallet Ownership Verification module will be called. If it returns a public key (user is actually the owner of the wallet) it will login with that public key. Otherwise, if there is any kind of error on the verification process, the log in will be cancelled.

In order to store the user information in the database the callback feature from next-auth will be used. It will call and execute the functions that store and update the database data after the user successfully logs in using the user session data as inputs of these functions.

Finally, we have to enable the option of linking more wallets or a Discord account (if none is linked) to an existing user. The option to add wallets to an existing user is easy to implement an almost similar way as logging but without changing the current user session information on the client side. However, due to the way Discord OAuth2 is implemented in next-auth.js there is no way to access to Discord user information without changing the session information (and missing the previous one) making impossible to link directly a Discord account with the next-auth approach established.

In order to fix this problem, it was decided to implement a "import account" feature. This feature will be only accessible when user has logged in through Discord. If user use this feature he will call the add wallet feature, but will be the only difference that if the wallet is already stored in the database and its owner has no Discord linked it will link all the wallets linked to that owner. Otherwise, whether the owner has already a Discord linked or the wallet is not stored in the database, it will just simply add the wallet itself.

With this the initially desired process of linking a Discord account is simulated. The only change for the user will be the order of the actions it has to do. It will change from the initial "Log in with one of its wallets and then log in with Discord to link" to the final "Log in with Discord and then log in with one of its wallet to link all". In this way we also keep the times the user has to log in to his wallet to one, which is the main objective (for user safety).

4.5. Data Visualization module

This module will be the one in charge of the display of the data obtained from NFTs Data & Metadata Extraction module in the application. The user will be able to see all the NFTs he owns, as well as extra information as vote power for verified collections, displayed on the application.

As the application is built on Next.js it is easily implementable by creating a React component that take a JSON object with the structure of extraction module as input. In order to have a dynamic component (which depends on the input) another nested component

for each verified collection and the non-verified NFT list will be used.

Moreover, to facilitate the design of the component the MUI React UI library will be used. It is a library which contains already designed components for the main necessities of a website interface, from containers to image lists.

4.6. Verified Collections Management module

This module will be only accessible for the admins of the application. It will be an API that will allow to insert new verified collections to the database as well as obtain different data from these collections. The objective of this module is also to design it in a way such that future services can communicate with this services easily.

In order to accomplish this easy communication it has been decided to implement an RestAPI with Express.js. This will allow future services to connect with this API and call its functionalities just by sending different request to it (an allowing to send info through JSON objects).

Its fist functionality will be inserting, updating or deleting data related with verified collections and NFTs. It has been decided to be implement in these separate, admin-access only, service to avoid any kind of option of external attackers or users modify any information related to these fields.

A connection with the database through Mongoose will be established and all the methods for reading (get methods) and editing or deleting (post methods) data will be implemented.

The other functionality will be to retrieve holder information data of an specific collection. Initially, a function to return a dictionary (data structure that provides one key and value matched together) with the wallets that hold at least one NFT from a certain verified collection. In order to make the data easy readable and accessible for future uses, the key of the dictionary will be the public key of the wallet, and the matched data an array with all the mint addresses (unique id to identify an NFT on Blockchain) of the NFTs which the wallet holds.

The other function will contrast this information obtained from the first method with the user data stored in the database. It will map this dictionary with the user information stored, creating a new dictionary which will have as key the discordId of the user and as matched value a list of NFTs owned by the user (aggregating all the NFTs linked to each of his public keys in previous result) and their power of vote. The objective of this function is to be able to obtain a list of holders and related information within the registered Discord users in the application.

One of the future planned improves and new features is to develop a Discord bot that connects to this API. This bot will allow to offer and monetize services as "Role Management for Exclusive access in a Discord server" which could allow to make private

channels inside the collection Discord server which are only accessible for holders of the actual collection. This kind of channels allow to publish holder exclusive information or build DAO voting systems, also only accessible for holders.

However, all the future line works will be explained later in an dedicated section.

4.7. Database Design

In this section the design for the database will be explained. NFT collections usually have a big quantity of NFTs (5000 is the most common size). This added to the fact that also NFTs can contain a huge amount of information themselves a database which can manage big amount of data and has a high scalability will be needed. Hence, MongoDB has been selected as the database to store the data of the application. One of the most used libraries used along when working with Node.js (the server side of a Next.js app) and MongoDB is Mongoose. Mongoose is a library that allows to work with MongoDB with Object Modelling. Not only makes easier to work with data as it is structure in an Object way but also ensure the integrity of the data since it must much the schemas defined.

The database model which has been defined for the database is the one composed by the schemas defined in Fig.4.2.

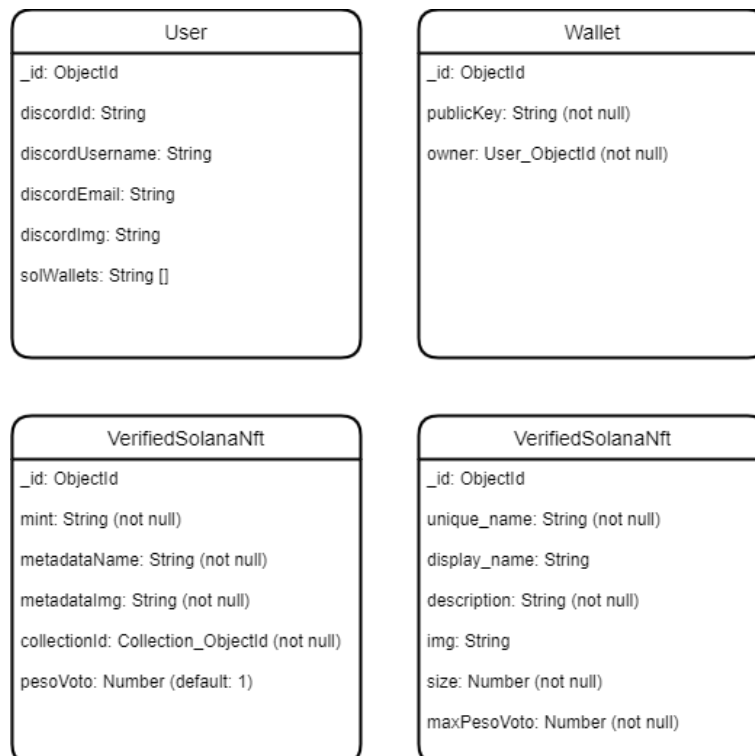


Fig. 4.2. Database Schemas.

All the Objects of the database will be identified by the ObjectId automatically assigned by MongoDB.

The only information saved on the wallet object will be its public key and the owner. The owner field will be an ObjectId which refers to the `_id` of the user who owns the wallet. All fields must be required since all wallets stored in the database must be linked to an user.

Regarding the User, it will include all the Discord linked account information (in case there is any). The stored Discord account information will be `discordId` (unique and immutable number to identify a Discord user) , `Discord username`, `discordEmail` and `discordImg` which will be the profile image of the user, in case he has any, and it will be stored as a string since Discord shares the profile images of users through links. On the other hand, in order to store the information of the wallets linked by the user, a field called `solWallets` will store an array with all the public keys of the wallets owned by the user.

All these fields can be null since an user may only link Discord (making `solWallets` not required) or only link Solana Wallets (making all Discord fields not required). However there will not be the case where there is an user with no information (besides `_id`) since MongoDB would automatically not include in a collection an element with no data.

Regarding the schema to represent the verified Collections it will be stored its `unique_name` (used to identify the collection), a description of the collection (which must be included in all cases), its `display_name` and `img` (also provided as a link) in case the owner of the collection wants to include it and the information regarding its size and `maxVotePower` (this is stored in case all the NFTs of a possible DAO does not have the same vote power, otherwise it will be equal to the size of the collection).

Finally. a verified NFT will contain its mint, which is the address of its mint account and the unique public key which is used to refer to NFTs in Solana, the ObjectId matching the `_id` of the collection it belongs and its voting power (as explained before in some collections some exclusive NFTs have extra privileges and voting powers, if it is not the case it will be 1).

5. IMPLEMENTATION AND TESTS

In this chapter the development process will be deeply explained. The implementation of each module will be explained and different test will be perform and shown in order to verify if the functional requirements have been accomplished.

5.1. Wallet Owner Verification

In order to interact with Solana network the `@solana/web3.js` library (built by Solana team) has been used. It is also possible to communicate with it through http requests, but this library makes it easier to connect with Solana API through methods and Objects.

Firstly we have to establish the connection with main-net Solana validator or any private validator. As it was explained before, main-net validator has some unavailability problems during network congestion so it was decided to user a private validator, Quick Node particularly.

Quick Node allows you to make up to 300.000 request per month to Solana network with its basic membership (which costs 10\$/month). When you have already got your membership Quick Node will automatically generate and set up for you an endpoint. In order to connect to Solana API we just have to change the IP of the main-net validator for the custom one provided by Quick Node.

Firstly, a API route on the Next.js app Back-End is created for generating and obtaining the nonce. In order to generate a random nonce and send it in a secure way to the client side the `crypto` library will be used. `Crypto` library includes different cryptography functionalities following all standards.

When fetching this route from server-side, it will generate a random nonce with `randomBytes` function and then it will be transformed into a string in order to work with it in an easier way and send it to the client-side. In order to send this nonce to the client-side it is stored in a cookie called "auth-nonce". It is important to make this cookie only usable and accessible only from our application site (`sameSite = 'strict'`) in order to protect from reply attacks. After the cookie has been set the nonce will be returned in order to it be temporally stored by the main verification method on server-side.

```
export default function handler(req, res) {
  if(req.method === "GET") {
    const nonce = crypto.randomBytes(32).toString('base64');

    res.setHeader('Set-Cookie', serialize('auth-nonce', nonce, { httpOnly: true, sameSite: 'strict', secure: true }));

    return res.status(200).json({nonce});
  }
  res.status(405);
}
```

Fig. 5.1. Random Nonce Generation.

The method that will prompt the verification will first connect to Phantom wallet (Phantom itself will automatically ask the user to log in if it hasn't already). Then it will call the nonce-generation method, adding to the client cookies the value of the nonce and obtaining the nonce itself. Then this nonce will be used to generate and encoded message. After this, it will send a request to Solana to sign the encoded message using Phantom wallet (which can be done if the owner has real access to the private key) and the signature will be returned only when it has been successfully signed.

```
if(phantom){
  await window.solana.connect();
  const nonce = await fetchNonce();
  const message = `Sign this message for authenticating with your wallet. Nonce: ${nonce}`;
  const encodedMessage = new TextEncoder().encode(message);
  const signedMessage = await solana.request({
    method: "signMessage",
    params: {
      message: encodedMessage,
    },
  });
};
```

Fig. 5.2. Signing Message with Nonce.

If the signature is returned, the signature data and some extra information needed for the action that prompted the verification (login, add wallet or import account) will be sent to the API route of the selected action. It will firstly check if the nonce from client-side is included on the signed message, which would means both the client-side and server-side nonce matches.

```
const dataToSend = {
  signedMessage: signedMessage,
  session: session
}

await fetch('../api/addWallet',{
  body: JSON.stringify(dataToSend),
  headers:{
    'Content-Type': 'application/json'
  },
  method : 'POST'
})
```

Fig. 5.3. Fetch to addWallet API route.

In order to check if both nonces match, we obtain the nonce from the cookies of the request. Afterwards, we create again an encoded message following the same procedure used previously for the signature so that it is exactly equal to the signed one if the nonces matches. At the same time, the public key and the signature is obtained from the signed message.

Finally, nacl library, another cryptography library that also allows to work with key-pairs, will be used to confirm the verification of the wallet. We will check if the content of

the encoded message made with the client-side matches the content of the signature and as well as if the public key obtained from the signed message matches the key-pair that signed the message. If this both conditions are satisfied, we can ensure the user is the real owner and has access to the key-pair (public and private key).

When a wallet is successfully verified the action that prompted the verification (login, add wallet, import account) remains its flow using the public key data , otherwise an error is thrown and the action is automatically cancelled.

```
const signedMessage = req.body.signedMessage;
const session = req.body.session.user;
if (signedMessage && session) {
  try {
    const nonce = req.cookies["auth-nonce"];
    const message = `Sign this message for authenticating with your wallet. Nonce: ${nonce}`;
    const messageBytes = new TextEncoder().encode(message);
    const publicKeyBytes = bs58.decode(signedMessage.publicKey);
    const signatureBytes = bs58.decode(signedMessage.signature);

    const result = nacl.sign.detached.verify(messageBytes, signatureBytes, publicKeyBytes);

    if (!result) {
      throw new Error("user can not be authenticated");
    }
  }
}
```

Fig. 5.4. Method to check if client-side and server-side nonce matches.

In Fig.5.5 it is shown how the user is asked to sign the a message in order to login. In Fig.5.6 we can see that after the user has signed the wallet, the wallet has been verified and the user has been able to log in the application with his public key.

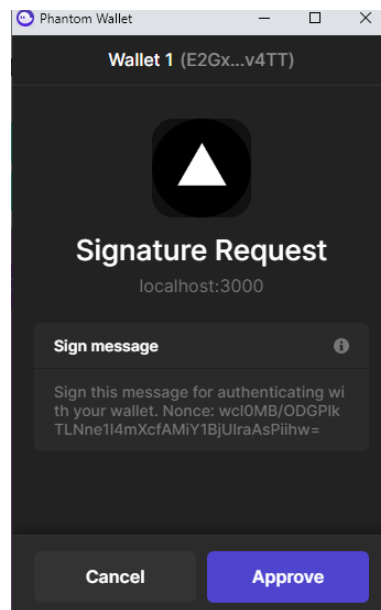


Fig. 5.5. Signature request to verify wallet.

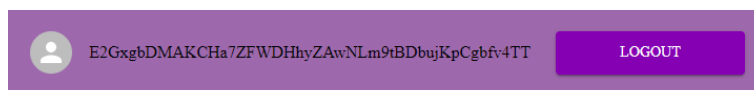


Fig. 5.6. Successful login after wallet verification.

After explaining the process of wallet verification we have to check if all the defined functional requirements are fulfilled.

Identifier	Requirement Value	Verified
FR_WALLET_VER_1	The system will support Phantom wallet	✓
FR_WALLET_VER_2	The system will support wallet owner verification through Phantom	✓
FR_WALLET_VER_3	The system will be able to generate a random nonce	✓
FR_WALLET_VER_4	The system will be able to send the specific nonce to only the user which is attempting to verify	✓
FR_WALLET_VER_5	The system will generate a new and random-generated nonce for each verification attempt	✓
FR_WALLET_VER_6	The system will generate a message containing the nonce set by then nonce recieved from client-side	✓
FR_WALLET_VER_7	The system will ask the user to sign the generated message in order to create a signature	✓
FR_WALLET_VER_8	The system will be able to obtain the signature performed by the user	✓
FR_WALLET_VER_9	The system will check if the signature message includes the nonce generated for the verification attempt	✓
FR_WALLET_VER_10	The system will verify a wallet only and only if there is a match between the client-side and the server-side nonce	✓
FR_WALLET_VER_11	The system will return the public key of the wallet when a wallet has been successfully verified	✓

TABLE 5.1. VERIFICATION OF FUNCTIONAL REQUIREMENTS
WALLET OWNER VERIFICATION MODULE

5.2. NFTs & Metadata Data Extraction

This module will obtain the NFTs, their data and their metadata from a specified user, which will be received as an input. It will not only extract the data but also will organize and output it in an organized way so that other modules can read the information easily.

The @solana/web3.js library will be used again, in order to communicate with the Solana Blockchain. Moreover Metaplex-foundation library will be used in order to work easily with the NFTs. Metaplex-foundation is an open-source protocol that have defined different standards for NFTs in Solana. It allows, to create, transfer and obtain the data from NFTs in an standardized format.

First of all, it will obtain the list of NFTs the specific user has. In order to do so, firstly it will obtain from the database all the wallets linked to that user. If there is at least one (otherwise it would just return an empty object as nothing was found) the system will obtain the list of NFTs owned by the list of wallets,

Getting the owner of an NFT it is really easy since the public key that owns a token is referenced in the token account. However, there is no reference from the public key side to any of the tokens it holds. So, theoretically we would have to look into every single token account stored in the Blockchain and only retrieve the information of the ones owned by the public keys of the user. However, since this a big problem for all community, Solana decide to write an on-chain Program (program that runs on Solana Blockchain) which obtains the list of tokens of a certain public key in the most efficient and fastest way (although it may take few seconds still).

Consequently, in order to obtain this NFT list we will just have to execute this on-chain program once for each of the public keys of the user and then filter out the NFTs from the tokens list. To do so an API route that will receive as query parameter the public key to look for was created.

Firstly, we have to send a request to Solana network, through our validator endpoint, calling the `getTokensAccountByOwner` method from Solana API. To this method we have to pass as arguments, the public key and the program id. In order to execute the http request axios library will be used (a library to perform http request in Node.js). This http request will return the token list of the specific public key.

```
async function getNFTlist(pk) {  
  var myParams = [  
    pk,  
    {  
      "programId": "TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA",  
      "encoding": "jsonParsed"}]  
  var tokensValue;  
  
  var data = JSON.stringify({  
    "jsonrpc": "2.0",  
    "id": 1,  
    "method": "getTokenAccountsByOwner",  
    "params": myParams  
  });  
  
  var config = {  
    method: 'post',  
    url: 'https://small-patient-fire.solana-mainnet.quiknode.pro/HERE GOES THE_CUSTOM_ENDPOINT/',  
    headers: {  
      'Content-Type': 'application/json'  
    },  
    data: data  
  };  
  
  await axios(config).then(function (response) {  
    tokensValue = response.data.result.value;  
  });  
  
  return tokensValue;  
};
```

Fig. 5.7. HTTP request to Solana API to obtain token list.

After this, the API route method will filter out the NFTs from the token list. First filter is applied by checking the `tokenAmount` and `tokenAmount decimals` which must be 1 and 0 respectively. After it is also important to check the supply of the token on-chain. The reason of these is that there are scam tokens made by malicious actors that pass this filter but actually have more supply on-chain. Interacting with this tokens can cause whether the user or the server to run a malicious script.

In order to check the supply on chain, the system will make a request to Solana API with the `getTokenSupply` method. The obtained `tokenSupply` must be 1 in order to the actual token be a real NFT.

```

async function isNFT(mintAddy){
  var mylist = [mintAddy];
  var isANft;
  var data = JSON.stringify({
    "jsonrpc": "2.0",
    "id": 1,
    "method": "getTokenSupply",
    "params": mylist
  });
  var config = {
    method: 'post',
    url: 'https://small-patient-fire.solana-mainnet.quiknode.pro/HERE_GOES_THE_CUSTOM_ENDPOINT/',
    headers: {
      'Content-Type': 'application/json'
    },
    data : data
  };
  await axios(config).then(function (response) {
    isANft = (response.data.result.value.amount === "1");
  })
  .catch(function (error) {
  });
  return isANft;
}

```

Fig. 5.8. Method to obtain if a token has a supply of 1.

Finally, the tokens that passed both filters which can be now considered NFTs are returned.

```

export default async function handler(req, res) {
  var pk = req.query.walletPK;
  var myNftList = [];
  var tokensValue = await getNFTlist(pk);
  for (const element of tokensValue){
    var myTokenData = element.account.data.parsed.info;
    if (myTokenData.tokenAmount.amount === '1' && myTokenData.tokenAmount.decimals === 0 && myTokenData.tokenAmount.uiAmount === 1 ){
      var condition = await isNFT(myTokenData.mint.toString());
      if (condition){
        myNftList.push(myTokenData.mint);
      }
    }
  };
  return res.status(200).json(myNftList);
}

```

Fig. 5.9. API route method to obtain NFT list of a wallet.

Now, once the first step (obtaining the NFT list of an user) has been done, we have to retrieve the data of each NFT obtained and send it in an structured way grouped by collections. In order to do so a "collection" interface is used. Its definition is the one that can be appreciated in Fig.5.10

```

interface colleccion {
  colleccionName: string,
  descripcion: string,
  size?: number,
  maxPesoVoto?: number,
  myPesoVoto?: number,
  metadata: [{
    name: string,
    image: string
  }],
  img?: string
}

```

Fig. 5.10. Collection Interface.

In order to store the "collections" a dictionary will be used, since it allows us to access the determined collection value with its key which will be its name. The NFTs will be

classified between verified and non-verified NFTs. Verified NFTs will be the ones which belong to a verified collection and are stored in the application database and the non-verified will be the rest. The way to extract the metadata of the NFT will differ from verified and non-verified NFTs.

The following process belongs to the Verification Collection Management module, and its implementation is deeply explained and test in Fig.5.5. When a collection is verified it is inserted in the database. After inserting all the NFTs are inserted into the database as well as their metadata obtained for Blockchain.

Since the database application will have always a slower ping than Solana API (without even taking into account congestion problems) it was decided to obtain the NFT data from the database when they are verified ones.

However, for the non-verified NFTs we have to obtain the metadata from the Blockchain. In order to do so we will use metaplex. Firstly we have to get the PDA of the NFT. The PDA is the Program Derived Address with a derived key of ['metadata', metadata_program_id, mint_id] and it is used by Metaplex. Thereafter, using Solana API getAccountInfo method we obtain the account information of the 3 keys of PDA. With this obtained info and the original mintAddress of the NFT, the metadata of the NFT (along with other extra data that is not taken for optimization) is obtained and returned.

```
const getMetadata = async (tokenAddress) => {  
  const metadataPDA = await Metadata.getPDA(tokenAddress);  
  const mintAccInfo = await connection.getAccountInfo(metadataPDA);  
  
  const {  
    data: { data: metadata }  
  } = Metadata.from(new Account(tokenAddress, mintAccInfo));  
  return metadata;  
};
```

Fig. 5.11. Function to obtain the Metadata of a NFT.

It is also important to consider that non-verified NFTs can also be malicious. Hence, it has been decided to eliminate or omit the non-verified NFTs that do not use Arweave protocol or a whitelisted secure domain. Arweave protocol is a decentralized and secure protocol to store data permanently used by most NFT collections to store its metadata. By doing this we ensure the links that the metadata refers to are always secure files and not a malicious file which may make whether the user or the server to run a script.

For testing "https://cdn.tatsu.gg" domain was also whitelisted since it is where an actual verified collection stores its metadata. However they have multiple collections and they are not all verified inside the application due to the huge amount of data it would be. However, their non-verified NFTs are still useful for testing and since they are secure they have been whitelisted.

```

else { // Non - verified NFT
  const result = await getMetadata(input);
  console.log("url");
  console.log(result.url);

  if(result.url.includes(validip2) || result.url.includes(validip2)) { // const validip2 = "https://arweave.net"; const validip2 = "https://cdn.tatsu.gg";
    console.log("Pasa dentro");
    var newResult;
    await axios({method: 'get', url: result.url, headers: {}}).then(function (response) {
      newResult = {name: response.data.name, image: response.data.image};
    }).catch(function (error) {});
    if(verifiedNFTdict["Unverified NFTs"]){
      verifiedNFTdict["Unverified NFTs"].metadata.push({name: newResult.name, image: newResult.image});
    }else{
      var myMetadata = {name: newResult.name, image: newResult.image};
      var newCollection = {collectionName: "Unverified NFTs",
        description: "This is the list of NFTs that hasn't been verified by the platform. Contact support if you want to verify your collection with us",
        metadata: [myMetadata]
      };
      verifiedNFTdict["Unverified NFTs"] = newCollection;
    }
  }
}

```

Fig. 5.12. Function to filter the non-verified NFTs which do not use Arweave

Once the both procedures of obtaining the data of the NFTs have been defined, now its only left to organize them and their data in collections, following the interface. To do so an extra "virtual" collection is created during the organization which will include all the non-verified NFTs.

In Fig.5.13 it is shown an example of the extraction of the data from an user (the JSON response has been passed through a JSON validator so that the structure is easier to appreciate).

```

verifiedNFTlist: [{
  "collectionName": "The Crooks",
  "description": "3,359 Crooks battling for control over the Crookiverse.",
  "size": 3359,
  "maxPesoVoto": 3500,
  "myPesoVoto": 3,
  "metadata": [{
    "name": "Crimson Crook #69",
    "image": "https://www.arweave.net/6I9uJkLm1jI6Z0ghFIZ3HvBp-MnUFQd6nxdMGLxPYo?ext=png"
  }, {
    "name": "Crimson Crook #759",
    "image": "https://www.arweave.net/pmvQqCJOGJYPB_YdZj-N4cpZdHND8fo1lPuD7hu6EwV?ext=png"
  }, {
    "name": "Crimson Crook #549",
    "image": "https://www.arweave.net/p9rTRRlmybhc9GQVbwjexQdz8AHLHGvV_gttb5lNuY?ext=png"
  }],
  "img": "https://dl.airtable.com/.attachmentThumbnails/1b121f0564653a717c92886062fe2423/376d37ee"
}, {
  "collectionName": "Solana Droid Business",
  "description": "SDB is a collection of 5,000 uniquely generated 24-bit droids living on the Solana blockchain.",
  "size": 5000,
  "maxPesoVoto": 5000,
  "myPesoVoto": 2,
  "metadata": [{
    "name": "SolanaDroidBusiness #3734",
    "image": "https://www.arweave.net/3GqAHZw4bsp55ml_uqEbovt1NR0V1gb8mqgnsxhoKs?ext=png"
  }, {
    "name": "SolanaDroidBusiness #4839",
    "image": "https://www.arweave.net/BhSv0hzX0KTKtf2Q8BmzPSlIyApV9TaidUdcqM2g?ext=png"
  }],
  "img": "https://i.imgur.com/3tbiL88C.jpg"
}, {
  "collectionName": "Unverified NFTs",
  "description": "This is the list of NFTs that hasn't been verified by the platform. Contact support if you want to verify",
  "metadata": [{
    "name": "TheDragonClub #1218",
    "image": "https://www.arweave.net/d_E--IP_dnve35n1121Vsh10sH_ggKMLMz1UhwFpJ4?ext=png"
  }, {
    "name": "TheDragonClub #1239",
    "image": "https://www.arweave.net/G91QeaYeyvT1ESsZQIA-7ruS24umwNT9ykFuhGwlagg?ext=png"
  }, {
    "name": "Meekolony PFP #3729",
    "image": "https://cdn.tatsu.gg/mklnpfp/3729.gif"
  }, {
    "name": "Meekolony PFP #4959",
    "image": "https://cdn.tatsu.gg/mklnpfp/4959.gif"
  }, {
    "name": "Meekolony PFP #3521",
    "image": "https://cdn.tatsu.gg/mklnpfp/3521.gif"
  }, {
    "name": "MeekDonald Cap #242",
    "image": "https://cdn.tatsu.gg/mklnpfp/mdcap/md_cap.png"
  }],
  "img": ""
}]

```

Fig. 5.13. NFT Data Extracted example.

After explaining the process of NFTs & Metadata Data Extraction we have to check if all the defined functional requirements are fulfilled.

Identifier	Requirement Value	Verified
FR_NFT_EXTRACT_1	The system will receive as initial input the user to obtain NFT data from	✓
FR_NFT_EXTRACT_2	The system will obtain wallet public keys list of the user from the database	✓
FR_NFT_EXTRACT_3	The system will obtain the list of tokens holded by the wallets in the wallet list	✓
FR_NFT_EXTRACT_4	The system will filter out the NFTs from the list of tokens obtained	✓
FR_NFT_EXTRACT_5	The system will classify the obtained NFTs between verified or non-verified	✓
FR_NFT_EXTRACT_6	The system will retrieve the data and metadata of the verified NFTs from the database	✓
FR_NFT_EXTRACT_7	The system will retrieve the data and metadata reference of the non-verified NFTs from Solana network	✓
FR_NFT_EXTRACT_8	The system will eliminate the non-verified NFTs which does not use Arweave for metadata referring	✓
FR_NFT_EXTRACT_9	The system will extract the metadata from non-verified NFTs which use Arweave for metadata referring	✓
FR_NFT_EXTRACT_10	The system will output the extracted data through a structured JSON object so that it is easy readable by other modules	✓

TABLE 5.2. VERIFICATION OF FUNCTIONAL REQUIREMENTS
NFTS DATA & METADATA EXTRACTION MODULE

5.3. Session Access Control

Session control has been implemented using Next-Auth.js, a Next.js open-source authentication library. In order to set app next-auth you only have to create a file called "[...next-auth].js" and locate it in the directory /api/auth. If Next.js application has Next-Auth installed it will automatically look for this file at build the authentication methods around it.

In order to set up Next-Auth, firstly the different providers (log in methods) will be defined. In our application case it will be Discord (already supported in the library) and custom credentials.

Custom credentials is usually used to login through the standard username + password method which looks at a database. However, we can customize it to our necessities. When customizing it you have to define an authorize function that will be executed with the credentials sent. In our case we will set the credentials input as the public key the user is trying to use to login. The function execute will be the one from Wallet Owner

Verification module. If the wallet is successfully verified the session will be created, setting the session_id (in order to be identified) and session_name (in order to be displayed on app) as the public key returned.

Next-Auth allows to set custom callbacks, which will be run after a successful login with the user data. These callbacks will be used to add a provider field which will be 'discord' or 'solana' depending on the method used to log in. After setting the provider, the callback will also call the method that will check if the user is new, and insert it in the database if so.

This method is included in two different API routes. One for Solana login and another for Discord login. For Discord login it will just make a FindOneAndUpdate on the user collection, updating the user data in case he has changed anything (Discord username for example) or creating a new one in case there was no user linked to that Discord.

However, for the Solana login, both a wallet and a user must be created if its the first time the wallet logs in.

```
const handler = async (req, res) => {
  if (req.method !== 'POST') {
    return res.status(405).send('Method not allowed');
  }
  if (req.body.pk) {
    try {
      var loggedWallet = await Wallet.findOne({publicKey: req.body.pk});
      if (loggedWallet) {
        var loggedUser = await User.findOne({_id: new ObjectId(loggedWallet.owner)});
        return res.status(200).send(loggedUser);
      } else {
        var newUser = await new User({solWallets: [req.body.pk]});
        await newUser.save();
        var newWallet = await new Wallet({publicKey: req.body.pk, owner: new ObjectId(newUser._id)});
        await newWallet.save();
        return res.status(200).send(loggedUser);
      }
    } catch (error) {
      return res.status(500).send(error.message);
    }
  } else {
    res.status(422).send('data_incomplete');
  }
} else {
  res.status(422).send('req_method_not_supported');
}
};
export default connectDB(handler);
```

Fig. 5.14. Solana Database update after login.

The functionality of adding a wallet to a logged user will work the same way but without changing the current session data. After the wallet has been verified, the system will search for the user which matches to current session, create a new wallet in the database (with the verified public key) and link that wallet to the user.

However for the import account method, it will first be checked if the account to import is linked to any Discord account. If so, same process as add wallet will be done. On the contrary, if the account to import has no linked Discord account. The account solWallets list will be inserted in the user, the account to import will be deleted, and all the wallets imported will be updated with the info of the new owner.

When a user wants to disconnect a wallet from its account, it will firstly be checked if he is the owner of that wallet and if so the wallet will be deleted from the database as well as pulled from solWallets user list. In case the deleted wallet was the one used by the user to login the user will be automatically logged out, by calling the Sign-Out method from next-auth. This is done because a session cannot include the information of a wallet not included in the database as it would not be possible to find the user related to that session.

After the module has been implemented different test are made in order to verify if it has achieved all the defined functional requirements. This test will perform the different actions that can be executed (login through wallet, login through Discord, eliminate wallet, add wallet and import account) in all the possible order combinations in order to verify the correct performance of the application and the integrity of the data from the database in all situations.

In Fig.5.15 it is shown an user that logged with his Discord account and linked 3 different wallets.

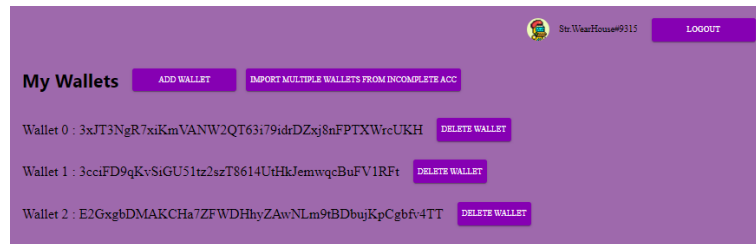


Fig. 5.15. Example of User logged with Discord and 3 wallets linked.

After the successful execution of all the test, it can be confirmed that all the functional requirements of the module are achieved.

Identifier	Requirement Value	Verified
FR_SESSION_1	The system will allow users to login through Phantom wallet.	✓
FR_SESSION_2	The system will allow users to login through Discord login.	✓
FR_SESSION_3	The system will allow users to log out.	✓
FR_SESSION_4	The system will allow users to connect multiple wallets to their account.	✓
FR_SESSION_5	The system will be able to store and update the information of a logged user.	✓
FR_SESSION_6	The system will allow users to import one or multiple wallets from other account, if that account is not linked to Discord.	✓
FR_SESSION_7	The system will use Wallet Owner Verification Module whenever the user logs in through Phantom, add or import wallets in order to ensure he is the real owner.	✓
FR_SESSION_8	The system will insert, edit or delete database data in order to have a record of the user information.	✓
FR_SESSION_9	The system will maintain the integrity of the user and wallet data of the database.	✓
FR_SESSION_10	When importing accounts the system will check if the account getting imported is not linked to a Discord user.	✓
FR_SESSION_11	When importing accounts if the user who is getting imported has no Discord account linked to it, all his wallets will get imported to the new user.	✓
FR_SESSION_12	When importing accounts if the user who is getting imported has a Discord account linked to it, only the wallet which has been used for the import will be imported.	✓
FR_SESSION_13	The system will be able to delete (disconnect) wallet from an user when he desires so	✓
FR_SESSION_14	The system will automatically log out an user if he deletes the same wallet he used for log in.	✓

TABLE 5.3. VERIFICATION OF FUNCTIONAL REQUIREMENTS
SESSION ACCESS CONTROL MODULE

5.4. Data Visualization

In a previous section, it was explained how will be the NFT related data from an user will be obtained. In this section, it will be explained how it will be displayed.

The data to be displayed will be received as an input. This input will be a list of col-

lections objects that follow the interface described in Fig.5.10. Since it will be a dynamic input (it will contain a dynamic amount of collections) we will use React components since they can be nested.

Each collection includes a list of NFTs metadata, in order to display a collection we will firstly define how to display all its metadata since its another dynamic value (the amount of NFTs metadata may differ from one collection to another). In order to do show we will create a Metadata-Display component which will take as input the metadata list of a collection.

In order to implement a nice-looking interface in an easier way, MUI React library will be used. It includes a lot of different ready-to-use components which can be customized. For displaying the NFT metadata, a ImageList (from MUI) will be used as it fits perfectly what we are looking for. A ImageItem will be created for each element of the input list. Each ImageItem title and alt will be set with the value of the NFT metadata_name and the src of the image will be the metadata_image link.

```
function MetadataDisplay(metadataList) {
  return (
    <ImageList cols={10} gap={0}>
      {
        metadataList.metadataList.map((metadata) => {
          <Box>
            <ImageListItem key={metadata.image} sx={{maxWidth:"250px", minWidth:"250px", mr:2, borderRadius:2, p:1, backgroundColor:"#504E6B"}}>
              <ImageListItemBar sx={{color:"white"}}
                title={metadata.name}
                position="below"
              />
              <img
                src={metadata.image}
                alt={metadata.name}
                loading="lazy"
                style={{borderRadius:"5px"}}
              />
            </ImageListItem>
          </Box>
        })
      }
    </ImageList>
  );
}
export default MetadataDisplay;
```

Fig. 5.16. Metadata-Display component.

Finally, a collection display React component will be created. This component will receive as input the list of collections to be displayed.

There can be distinguished two types of collections, verified and non-verified. Verified collections are the ones verified in the application and stored in the database. The non-verified collection will be the one virtually created by the NFTs & Metadata Data Extraction in order to include all non-verified NFTs in a collection that follows the interface created. It is important to take into account that verified collections have extra fields as avatar (which is not required), size or vote power and they also have to be displayed in the application.

In order to display all the information of the collection, first the avatar (in case it has) and the name will be displayed. Then it will be checked if the collection is verified or not by checking if it has the size attribute. If so the size and vote power data will also be displayed. Finally the NFT metadata will be displayed using the Metadata-Display component and passing to it the metadata list input.

```
function VerifiedCollectionDisplay(collectionList) {
  return (
    <Box>
      {
        collectionList.collectionList.map(function(collection, index){
          return (
            <Box sx={{mt:5,mb:2}}>
              <Stack direction="row" justifyContent="start" sx={{p:1}}>
                <collection.img 48>
                <Avatar src={collection.img} sx={{alignSelf:"center", mr:1}} />
                <Typography component="div" variant="h5" sx={{alignSelf:"center", mr:1,fontWeight:600}}>
                  {collection.collectionName}
                </Typography>
                <collection.size 48>
                <Stack direction="row" justifyContent="start" sx={{p:1}}>
                  <Box sx={{mr:2, borderRadius:100, p:1, backgroundColor:"#5D4E6B",alignItems:"center"}}>
                    <Typography component="div" variant="subtitle1" sx={{alignSelf:"center", mr:1,color:"white"}}>
                      Holds: {collection.metadata.length}/{collection.size}
                    </Typography>
                  </Box>
                  <Box sx={{mr:2, borderRadius:100, p:1, backgroundColor:"#5D4E6B"}}>
                    <Typography component="div" variant="subtitle1" sx={{alignSelf:"center", mr:1,color:"white"}}>
                      VoteWeight: {collection.myPesoVote}/{collection.maxPesoVote}
                    </Typography>
                  </Box>
                </Stack>
                <MetadataDisplay metadata={collection.metadata}></MetadataDisplay>
              </Box>
            )
          )
        })
      }
    </Box>
  );
}
export default VerifiedCollectionDisplay
```

Fig. 5.17. Collection-Display component.

In Fig.5.18 it is shown how the data from Fig.5.13, which was an example of the extraction of the data from an user, is displayed in the application.

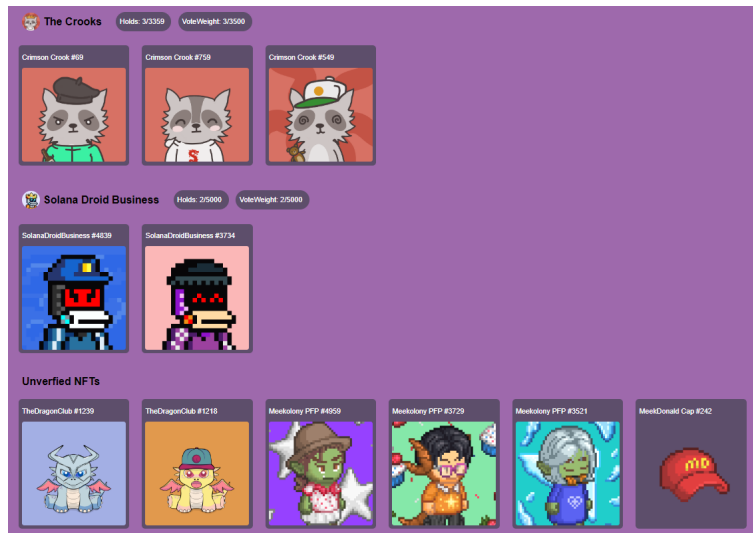


Fig. 5.18. Data Visualization example.

After explaining the process of data visualization we have to check if all the defined functional requirements are fulfilled.

Identifier	Requirement Value	Verified
FR_DATA_VIS_1	The system will be dynamic according to the input it receives	✓
FR_DATA_VIS_2	The system will display the content and data of each NFT received from input	✓
FR_DATA_VIS_3	The system will organize these NFT content and data by collections	✓
FR_DATA_VIS_4	The system will include information of the collection	✓
FR_DATA_VIS_5	The system will include the user holder information, quantity and vote power, in each displayed verified collection	✓

TABLE 5.4. VERIFICATION OF FUNCTIONAL REQUIREMENTS
DATA VISUALIZATION MODULE

5.5. Verified Collections Management

As it was explained in the design chapter, this module will be implemented in a new environment, an Express.js RestAPI. In order to keep our API more organized, a sub-route will be created for each of the functionalities.

On the one hand, we will have the verified collection insertion functionality. It will be implemented in the '/insertCol' sub-route. The different steps will be implemented in different sub-routes of '/insertCol' and the main method will be executed when making a request to the '/' sub-route.

First of all, it is important to define how the input must be sent by the admin using the functionality. The collection will include the information of the collection, its name, description, size, max power of vote and avatar (in case it has) as well as its Candy Machine Id (as explained before, is the Token Program public key used to find get NFTs of the collection) following the format described in Fig.5.19.

```
const collectionExample = {
  unique_name: "Solana Droid Business",
  description: "SDB is a collection of 5,000 uniquely generated 24-bit droids living on the Solana blockchain.",
  size: 5000,
  maxPesoVoto: 5000,
  img: "https://i.imgur.com/1tbl8BC.jpg",
  candyMachineId : "GdXab5GxaU23gpzdY7Zqzfm4t7b3LHkHnnCqsb21yLGV"
};
```

Fig. 5.19. Example of Verified Collection Object to insert.

When a collection is inserted, the main method will firstly call the '/insertCol/getMintList' route which will return the list of mint address of the NFTs of that collection. In this route, in order to obtain the NFT list, a request to Solana network using the getProgrammAccounts method will be done. The program which will be used is the metadata program (an standardized program for Solana metadata) using as filter the public key of the Candy Machine Id (which is the first creator of the NFTs of the collection). After-

wards, the metadata accounts are returned and their information is mapped in order to only keep the mint address.

```
const TOKEN_METADATA_PROGRAM = new PublicKey("metaqbxxJerdq28cj1RbAwkYQm3ybzjb6a8bt518x1s");
const getMintAddresses = async (candyMachineId: String) => {
  const firstCreatorAddress = new PublicKey(candyMachineId)
  const metadataAccounts = await connection.getProgramAccounts(
    TOKEN_METADATA_PROGRAM,
    {
      dataSlice: { offset: 33, length: 32 }, // The mint address is located at byte 33 and lasts for 32 bytes.
      filters: [
        { dataSize: MAX_METADATA_LEN }, // Only get Metadata accounts.
        { memcmp: { // Filter using the first creator.
          offset: CREATOR_ARRAY_START,
          bytes: firstCreatorAddress.toBase58(),
        } },
      ],
    },
  );
  return metadataAccounts.map((metadataAccountInfo) =>
    bs58.encode(metadataAccountInfo.account.data)
  );
};
```

Fig. 5.20. Method to obtain Mint List of a collection

In order to obtain the metadata of an NFT, an API route is created in `'/insertCol/getNftMetadata/'` where the mint address of the NFT is passed as a query parameter. The data is obtained with the same procedure as in NFTs Data & Metadata Extraction module (section 5.11). The same method used is replicated in this API since as explained before, the objective is to keep this API isolated and not connect to the Next.js app under any circumstances.

Finally, in order to insert both the verified NFTs and the collection two different routes are created. On the one hand, for the collection insert (which must be done first in order to be able to reference the NFTs to the collection in the database later) a simple insertion on the database with the collection data will be done. The collection insertion method was located in the `'/insertCol/insertCollection/'` API route.

On the other hand, an API route `'/insertCol/insertMintList/'` will be created. It will receive as input (through the body of the request) the mint list obtained from `'/getMintList'` and the `collection_id` that was automatically assigned by MongoDB during the collection insertion.

The metadata of each element of the list will be obtained using `'/insertCol/getNftMetadata'` API route and the each NFT will be inserted. In case, there is an error on the mintList or a collection is inserted twice `findOneAndUpdate` method is used, creating the NFT in case no NFT is found with the specific mint address.

```
const handler = async (req, res) => {
  const collectionId = req.body.collectionId;
  for(const elem of req.body.mintList){
    const metResponse = await fetch(process.env.BASE_URL+'api/getMetadata/'+elem.mint);
    const resData = await metResponse.json();
    await VerifiedSolanaNft.findOneAndUpdate(
      {mint: elem.mint},
      {metadataName: resData.newResult.name, metadataImg: resData.newResult.image, collectionId: ObjectId(collectionId)},
      {new: true, upsert: true});
  }
  return res.status(200).json("Insertion completed");
}
export default connectDB(handler);
```

Fig. 5.21. Method to insert of the NFTs of a collection

The main method on API route `'/insertCol'` will just call the different API routes in order passing the required inputs in order to the whole process be executed in order. In essence, an admin will just have to make a call including the collection information (as in Fig.5.19) and all the process will be automated.

On the other hand, we have the holder data extraction functionality. It will be located in the API sub-route `'/collectionHolders/'`. It will be divided in two sub-routes it self, one for obtaining the wallet holder list on `'/getPKlist'` and another for obtaining the Discord holder list on `'/getDiscordList'`.

In order to obtain the wallet holder list the `collectionId`, the one stored in the database, will be passed as query argument. All the NFTs from the database with that `collectionId` value at `NFT_collectionId` field will be obtained, keeping only the mint address information which is the only needed field.

For each NFT obtained, a request to Solana network will be done using `getLargestTokenAccounts` methods which will return the biggest Token Accounts (account where the token is stored but not the owner account) holding an specific token. In our case, there will be only 1 account since all NFTs have a supply of 1. Then another request to the Solana network will be made to obtain the account information. It is important to use the `getParsedAccountInfo`, not the `getAccountInfo`, otherwise the owner info will not be readable. From the parsed account info we can get the owner information and its public key.

In order to return the data as it was defined in requirements a dictionary will be used. The key value will be the public key of the owner and the value the list of mint addresses of the NFTs it holds.

```
router.get('/getPKlist', async (req, res) => {
  const mintlist = (await NFT.find({collectionId: new ObjectId(req.query.collectionId?.toString())}, 'mint')).map(elem => {
    return elem.mint;
  });

  var holderDict: {[key:string]: [string]} = {};
  for(const mint of mintlist){
    const tokenAcc = await connection.getTokenLargestAccounts(new PublicKey(mint));
    const parsedInfo: any = await connection.getParsedAccountInfo(tokenAcc.value[0].address);
    const holderPk = parsedInfo.value.data.parsed.info.owner;
    if(holderPk in holderDict){
      holderDict[holderPk].push(mint);
    }else{
      holderDict[holderPk] = [mint];
    }
  }
  res.json(holderDict);
})
```

Fig. 5.22. Function to obtain public keys holders of a collection

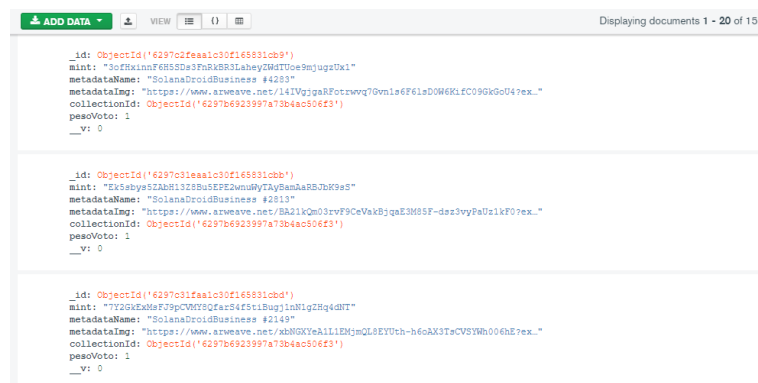
The process of extracting the Discord holder data is similar to the one for public key holder data. In the first place, the public key holders data is obtained by fetching to `'/getPKlist'` route. Thereafter, a join (a established connection between two data sources) is done between the public key list and the user and linked wallets information. For each public key of the dictionary, it will be checked if the public key is stored as a wallet in the database application. If so, those NFTs will be assigned to the owner user of the wallet obtaining as final result a dictionary using as key a discordId (obtained from user and used to identify him) and NFTs owned by that user as the matching value. The users which do

not have a Discord account linked and their wallets are omitted.

Regarding the verified collection insertion, a insertion of a collection of 5000 NFTs is tested. However, due to the to big resources consumption of the action executed and the test being run in a local machine, only the first 150 NFTs of the collection are inserted for the test. The data to make the insertion will be the one shown in Fig.5.19, after the insertion, Fig.5.23 represents how the collection has been successfully stored in the database. Fig.5.24 shows the data of 3 inserted NFTs out of the 150.

```
_id: ObjectId('6297b6923997a73b4ac506f3')
unique_name: "Solana Droid Business"
v: 0
description: "SDB is a collection of 5,000 uniquely generated 24-bit droids living o..."
size: 5000
maxPesoVoto: 5000
img: "https://i.imgur.com/JtbLBBC.jpg"
```

Fig. 5.23. Stored Data of inserted collection.



```
_id: ObjectId('6297b6923997a73b4ac506f3')
mint: "3fFkinnF855D83Phkx83LahyDwIT0e9mju9x0k1"
metadataName: "SolanaDroidBusiness #4283"
metadataImg: "https://www.arweave.net/14TVgjaRfotrvvq70vnl6F61sD0W6KfC09Gk0U4?ex..."
collectionId: ObjectId('6297b6923997a73b4ac506f3')
pesoVoto: 1
__v: 0

_id: ObjectId('6297b6923997a73b4ac506f3')
mint: "Ex5abya51AhH1328Bu5EF2nuuWYTAy8amha8B3bK9e5"
metadataName: "SolanaDroidBusiness #2613"
metadataImg: "https://www.arweave.net/BA21K0h3rrvF9CeVakBjqa5M85F-daz3vyPaU1kF0?ex..."
collectionId: ObjectId('6297b6923997a73b4ac506f3')
pesoVoto: 1
__v: 0

_id: ObjectId('6297b6923997a73b4ac506f3')
mint: "7f2G2x8bF9p0V0Qf8e34F5c18ug1nW1gZHQdWT"
metadataName: "SolanaDroidBusiness #2149"
metadataImg: "https://www.arweave.net/xh8GKXeA11EMjmqL8EYUth-hoA33TcCVSWb006h?ex..."
collectionId: ObjectId('6297b6923997a73b4ac506f3')
pesoVoto: 1
__v: 0
```

Fig. 5.24. Stored Data of inserted NFTs.

After this collection has been inserted in the application we can now also test the wallet holder functionality. Fig.5.25 shows a part of the obtained JSON which includes the public key holders (whole document has not been shown since it is really big as it includes information of the 150 inserted NFTs). It can be seen how the data is structured as it was defined, the public key (white text) is matched with a list of NFT mint addresses (green text)

```
"GUFcR9mK6azb9vcpsxgXy7XRPAKJd4KMHTTVtncGgp": [
  "3xF8XTepG57aU7oh7dpn9AhqN79wuKym7Xdh2iTxFWny",
  "GwkYdJf8mXGGFYnPt4Mt9bmozjzrBqmDEpDHJvEZ5ArP",
  "J6HoF28snUUBuQAzK5VC6xPMaLc8GGInNCBQn5QSpN7D",
  "CTaGkU2gugzCBdK7xHMaPBN7JBBNRJ8MrWQxqNbBuSyw",
  "7b4ZyoLPHYnJ5F4TGDhLGHFusNrIRz5qoU9tttMkaLAz",
  "D7DCNqbBZwKiEjGVrHr23pqQJzuaLJ9MU4UtNRbrLAFP",
  "9S4UMFsYeJKvauKM4gGow6miFpfgNgK9Z5HeBydx2HgL"
],
"7mgBgE1rW3PUQgYFY1TUCdx9sF5HLGaGE8zmYjh875RN": [
  "HU6d1jQCgL2YKJxSer2nGsgdYHUS116K8s4z7dPKCbAY"
],
"AyRHPoSos8xDGRUNhEW6S6RJURBA2Bo3696EFYXJWbTK": [
  "ALsc5qqe2HJtwR2e64H4T4NuCuLdzDfQLV5fLFJ2K5ro",
  "AMVvgZp5oJgDfQWZ61FSLpWjxA8ZdKupuVD9AajF6on5",
  "GMZWfSRE8pb8FNnrSbtceSm7YPsczLFqfLWSLTHxnd7F"
],
```

Fig. 5.25. Obtained Wallet Holders data.

In order to test also the Discord holder data functionality, I purchased two NFTs of this specific collection and link both my wallets and my Discord to the application. In Fig.5.26 it can be seen that all the information from the only holder found (the only one registered in the application) is obtained: discordId, amount of NFTs being holded, list of NFTs being holded and vote (this collection has an equal vote power of 1 for all its NFTs).

```
{
  "discordHolders": {
    "discordId": "572433761250050048",
    "mintList": [
      "EueBZxvdj796X9NsdDEEFzmeYhx8w6YWdix4Pu54f4iU",
      "wTW7LkAf4XA3D3wy25HH5ocxzaPPyDFt6X3BzS3hDmz"
    ],
    "pesoVoto": 2,
    "cantidad": 2
  }
}
```

Fig. 5.26. Obtained Discord Holder data.

With the performed test we can ensure all the functional requirements defined have been accomplished.

Identifier	Requirement Value	Verified
FR_VER_COL_1	When adding a verified collection, the system will receive as input the data of the collection and its Candy Machine Id	✓
FR_VER_COL_2	The system will check if the collection is not already inserted in the database	✓
FR_VER_COL_2	The system will insert the collection into the databases, if it is not included in the database	✓
FR_VER_COL_3	The system will extract all the NFT belonging to the collection from Solana network using the Candy Machine id.	✓
FR_VER_COL_4	The system will obtain the data and metadata of the NFTs that belong to the collection through NFTs Data & Metadata Extraction module.	✓
FR_VER_COL_5	The system will insert each NFT, now verified, in the database	✓
FR_VER_COL_6	The system will be able to obtain the list of public key that holds at least one NFT from a determined collection.	✓
FR_VER_COL_7	The system will include in this list which NFTs each of these public keys hold	✓
FR_VER_COL_8	The system will be able to obtain the list of application users that holds at least one NFT from a determined collection.	✓
FR_VER_COL_9	The system will map the public key list data with the user data from the application database, taking into account only users that have a Discord account linked.	✓
FR_VER_COL_10	The system will return a list of the discordId, of the application users which holds at least one NFT.	✓
FR_VER_COL_11	The system will include in the discordId holder list which NFTs each user holds and two extra parameters, quantity holding and power of vote.	✓

TABLE 5.5. VERIFICATION OF FUNCTIONAL REQUIREMENTS
VERIFIED COLLECTIONS MANAGEMENT MODULE

6. PROJECT MANAGEMENT, SOCIO-ECONOMIC AND REGULATORY ASPECTS

In this chapter, the regulatory frameworks which were applied and affected the development of the project will be explained. Moreover, an analysis of the both social and economical impact of this project and the ones related to the Blockchain technology will be done. Finally, the planning executed and the future work lines of the project will be described.

6.1. Regulatory Framework

There are different European and Spanish laws which regulate different aspects of the topics and scopes of the project.

España Digital 2025 [23] is the agenda for digital transformation for Spain. It was presented and approved by Spanish Government on 23 Jul. 2020. It describes the planning created for digital transformation for Spain including the objectives to be achieved for 2025. It mentions that the Blockchain technology will have an important role, particularly in the European project European Blockchain Services Infrastructure (EBSI). EBSI will be a Blockchain based infrastructure which will allow to provide services along the whole European Union [24].

European commission has also stated a regulatory framework for Blockchain technologies development [25]. It states that they are working on creating a regulatory sandbox (a facility to test innovative solutions which brings together companies, regulators and developers) to impulse the Blockchain technology both for EBSI use cases and private use cases.

6.2. Socio-Economic Impact

6.2.1. Social Impact

Decentralized Blockchain were designed with the main purpose of giving the full control to people of their information and their assets. Currently, all the big companies, especially social media companies not only own a lot of information about their users but also they commercialize it. At the same time, central banks have full control of our money and assets. If due to whatever reason (justified or not) your bank decides to freeze your account you have nothing to do besides waiting them to unlock it when the frozen money is only yours. Decentralized Blockchain technology gives the full control and ownership of their data and assets to the users. However, this full control comes, as any great power, with a high

responsibility as you are the only responsible of any problem that may happen.

Although, a full decentralized world is almost impossible to exist, this technology gives people from all over the world a lot of freedom. At the same time, it promotes an egalitarian and fair society, since everything performed is transparent and there is no barrier-entry for anyone besides internet access. While a person in Africa may not have access to create a bank account he will always be able to open a wallet in any Blockchain and take payments in cryptocurrencies.

Regarding this particular project, it aims to help all type of people (both those more into this technology and those who not) to adopt this technology in a safe and easy way.

6.2.2. Economical Impact

At the same time, this technology has a huge economical impact. Since everything is tokenized in this technology and any token can be traded (bought or sold) it is possible to set a value for everything. This has its pros and cons. Its main, and most popular, con is the fact that it attracts a lot of speculation. Since anything can be tokenized and there is not a tangible or real way to calculate the value of the different Blockchain technologies there exists a lot of FOMO (Fear of Missing Out) and a lot of "all smoke and mirrors" projects along with a lot of volatility (huge variations) on the tokens prices. This makes the crypto world an easy environment to lose huge amounts of money for inexperienced (and even experienced) people.

However, if it is seen as a technology or a tool and not a way to earn money it has plenty of economical benefits. The main one is that it allows anyone from anywhere in the world to make safe payments for an almost non-existing fee (for example, as previously stated the cost of a Solana transaction is \$0.00025). At the same time it can be used as a value protection asset in countries with a bad economic or freedom situation. The best example may be Venezuela, whose currency gets more and more devalued each day. In general the people from South American countries are the ones who are adopting these technologies and cryptocurrencies the most due to the high devaluation of their currency they have been suffering for years.

Lastly, another big impact this technology can have thanks to its transparency is its use for government treasury. If implemented correctly it could avoid corruption entirely. Since all payments are visible (although anonymous) everyone would be able to track where the government spends the country money by publishing the public keys of the government treasury.

6.3. Project Planning

In this section the project plan executed during the development of the project will be explained in detail.

Hereunder, in TABLE 6.1 the Gantt Diagram which was proposed and used will be described :

Task	Jan 22	Feb 22	Mar 22	Apr 22	May 22	Jun 22
Analysis of Product						
Design of Product						
Implementation						
Tests & Validations						
Memory						

TABLE 6.1. GANTT DIAGRAM

Since the beginning, a planning for the project was defined. It was decided to be divided in months in order to have different measurable objectives at the end of each month and at same time have some extra margin in case there is any unexpected problem.

The initial planning estimated a total 320 hours of work. However, there were plenty of problems encountered, mainly during implementation which derived in delays resulting in a total of 415 hours of work. The approach to keep the deadlines of each month and keep following the plan structure was to add this, extra and unexpected, hours of work to the months the specific delayed task was assigned to.

In the Fig.6.2 the planned hour of works, the actual amount of hour of works and the start and end date of each task is described.

Task	Start Date	End Date	Plannified Work Hours	Dedicated Work Hours
Analysis of Product	Jan. 14, 20202	Feb. 28, 2022	50	65
Design of Product	Mar. 1, 2022	Mar. 31, 2022	50	55
Implementation	Mar. 10, 2022	Apr. 30, 2022	120	175
Tests & Validations	Apr. 1, 2022	May 31, 2022	40	55
Memory	Jan. 14, 2022	June 20, 2022	60	60
Total			320	415

TABLE 6.2. WORK HOURS DEDICATED TO THE PROJECT.

6.4. Budget

In this section a detailed description of the project costs will be done. The budget will be divided in different areas depending on the type of expense.

6.4.1. Cost of Labour

The cost of labour, is the budget that was destined to pay all the employees who worked in the project. According to Talent.com [26] the salary for a junior Computer Science Engineer in 2022 is 26.500 €/year which is equivalent to 13.75€/h. According to Glassdoor.com [27] the mean salary for a Software Project Director is 48.000 €/year which is equivalent to 25€/h.

Employee	Salary (€/H)	Hours of Work	Total Cost
Junior Computer Science Engineer	13.75	415	5706
Software Project Director	25	25	625
		SUM	6331

TABLE 6.3. COST OF LABOUR

6.4.2. Cost of Product

The cost of product, if the cost of all the hardware and software used in order to develop the project. Since all the technologies, frameworks and software used were open-source there are no costs derived from them.

The cost of the laptop used to develop the project is included. QuickNode Launch Membership [28] was the one selected in order to use QuickNode validator services since it is the cheapest one and it has more than enough request per month limit for the project. Moreover, there were purchased multiple NFTs with a total cost of 60€ in order to have some NFTs to make test with in the application.

Product	Original Cost	Amortisation	Cost / month	Months Used	Total Cost (€)
MSI Prestige 14 Evo A12M-049ES	1399 €	60 months	23,30€	6	140
QuickNode Launch Membership	8,70 € / month	–	8,70 €	6	52
Windows 10	Free	–	Free	6	0
Visual Studio Code	Free	–	Free	6	0
Next.js Framework	Free	–	Free	6	0
MongoDB	Free	–	Free	6	0
Discord OAuth2 Service	Free	–	Free	6	0
Solana API	Free	–	Free	6	0
Test NFTs purchases	60 €	–	One Time Payment	–	60
				SUM	252

TABLE 6.4. COST OF PRODUCT

6.4.3. Indirect Costs

Indirect cost are the cost derived from the basic resources used in the workplace during the development of the project. This refers, in our case, to electricity, internet and transport expenses.

Service	Cost / month	Months Used	Total Cost (€)
Electricity	30 €	6	180
Internet	35 €	6	210
Transport	20 €	6	120
		SUM	510

TABLE 6.5. INDIRECT COSTS

6.4.4. Total Cost

Finally, after each expense has been explained in detail we can obtain the total cost of the project:

Concept	Total Cost (€)
Cost of Labour	6331
Cost of product	252
Indirect Costs	510
Total Cost without IVA	7093
IVA (21%)	1489,50
Total Project Cost	8582,50

TABLE 6.6. TOTAL COST

6.5. Future Work Lines

As it has been stated on the entire document, Blockchain technology and NFTs have an almost infinite amount of use cases. In consequences, there are a lot of different work lines to keep working in this project.

These are the different non-excluding (which can be developed in parallel) work lines defined:

- **Improvements on Speed.** The speed of loading have been one of the main priorities on the development on the application. However, there is always room for improvement. It can be accomplished whether optimising algorithms, improvements on Solana network speed (which is not in our hands) or achieving lower pings (time to receive a response from a connection) to Solana API by using a faster validator or creating our own validator.
- **FAQ and Documentation for User.** One of the main objectives is to help people which do not have a deep understanding and control of Blockchain technologies. In order to improve user experience and help newcomer users it is planned to create a FAQ (Frequently Asked Questions) section in the application. Moreover, it is also planned to create a complete documentation not only about the functioning of the application but also a full-guide for beginners of Solana Blockchain and the best practices to keep your assets safe.
- **OAuth Public API.** The application was mainly designed to be the OAuth Authentication layer of a bigger application. However, as of now it is implemented to be only accessible for internal services. However, OAuth system are almost always designed to be accessed by external services. This would not only attract a lot of new users to the application but will also spread the benefits of this authentication system to other Apps or dApps (decentralized apps). The main objective is to contribute to have a safer and easier to adopt ecosystem, allowing other apps to implement our application authentication system would contribute a lot to this. In consequence, it is also planned to create a work line focused on making a OAuth public API, which will probably be the most difficult work line raised as keeping the user privacy and safety is a must.
- **Discord Holder & Role assignment Service.** One of the future work lines with highest priority is creating a Discord Bot to offer services to NFT Collections in their Discord servers. It is really common for collections to have Discord servers where the community share their thoughts. Moreover, it is also pretty common to have in these Discord servers exclusive channels which can only be accessed by holders where different information regarding the project being undertaken by the collection or community.

The objective of this Discord Bot is to automatically give the role and permissions to access these different channels to holders of the collection. This bot would be connected to the Verified Collections Management module API and will update the Discord roles and permissions periodically using the data obtained by the API.

Moreover, this bot would also allow to perform decentralized voting inside the Discord server. It will allow to users from a DAO (Decentralized Autonomous Organization) to vote the different proposals raised taking into account the amount of NFTs and their power vote each user holds.

- **User finder.** Another feature which is planned to be added in the future is the User finder. The objective of this feature is to allow users decide whether their profiles is private or public (would be private by default). There will be a section on the application to search users by discordId. It will search for the user which has that discordId linked and if he has set his profile to public all the NFTs he hold will be shared and displayed. This would allow users to show their NFTs to other people without having to publish their public keys and ensuring they are the real owners.
- **Building our own validator.** As it has been explained in section 2.5.2, validators contribute to the operation of Solana network and store a copy of the Solana Blockchain. Moreover the communications with Solana API are made through them. If the application reaches a big amount of users after its production release, it may be considered to create our own validator for the application. This would, firstly, eliminate the dependency on other validator services and give a lot of autonomy. Moreover, the ping to Solana would be highly reduced as using our own validator would remove almost all the ping to the validator itself.

7. CONCLUSION

This project has been a big challenge not only for the problems encountered but mainly due to the fact of working with such new technologies and, in consequence, poorly documented and standardised. However, as a result of this experience I learnt a lot of new concepts, experience and knowledge. Most of these completely new knowledge acquired is related to Blockchain, programming in Solana and the newest web technologies and frameworks.

I had to put in practise lots of the things learned in my Computer Science Engineering Degree at university as database design and management, software engineering, project management or web interfaces. Furthermore, not only I used this knowledge but also reinforced it since I have to study them in a deeper way in order to apply them in the project.

The main problems encountered during the development of the project occurred during the implementation. Working with Blockchain was pretty difficult. Most problems happened mainly since there is not many documentation or guidance regarding how to implement Solana Blockchain in your application and the few which is available is from completely different sources.

This caused some incompatibility problems, which resulted in some changes or even to completely change an entire module. This happened for example with Wallet Owner Verification module, after having implemented it trough a library and started implementing rest of modules taking this into account I found an article talking about a possible risk for user impersonation on different popular libraries used for wallet integration. This problem made me to implement the verification module from zero again as well as changing different parts of the modules that connect to verification module in any way.

It made unusable the work I had been doing the 10 previous days and probably was the moment I felt more frustrated. Due to this, I decided from then to implement everything related to Blockchain with an own solution (using no external libraries besides the official and standardised from Solana team). This decision made the process of implementation a little harder but was completely worth it since I learned a lot more during the process and also obtained a better result on the final product.

In conclusion, this project was a fantastic experience which allowed me to grow both personally and professionally learning lot of things which will be very useful for my future works.

BIBLIOGRAPHY

- [1] S. Goodyear, “This man owns \$321m in bitcoin — but he can’t access it because he lost his password,” *CBC Radio*, Jan. 16, 2021. [Online]. Available: <https://www.cbc.ca/radio/asithappens/as-it-happens-friday-edition-1.5875363/this-man-owns-321m-in-bitcoin-but-he-can-t-access-it-because-he-lost-his-password-1.5875366> (accessed on Jan. 16, 2021).
- [2] “Coinmarketcap - cryptocurrencies prices & charts,” [Online]. Available: <https://coinmarketcap.com/> (accessed on Jun. 12, 2022).
- [3] “The blockchain technology on the music industry - scientific figure on researchgate,” [Online]. Available: https://www.researchgate.net/figure/Centralized-decentralized-and-distributed-systems-Source-NetworkCulturesorg_fig1_327401244 (accessed on Jan. 25, 2022).
- [4] J. R. S., “¿cuánto cuestan las elecciones generales 2019? esto es lo que pagará cada español,” *El Confidencial*, Sep. 18, 2019. [Online]. Available: https://www.elconfidencial.com/elecciones-generales/2019-04-26/cuanto-cuestan-campana-organizacion-votos_1954082/ (accessed on Jun. 12, 2022).
- [5] Solana-Foundation. “Solana official documentation,” [Online]. Available: <https://docs.solana.com/> (accessed on Mar. 21, 2022).
- [6] A. Yakovenko. “Solana whitepaper,” [Online]. Available: <https://solana.com/solana-whitepaper.pdf> (accessed on Mar. 21, 2022).
- [7] J. Londoño, “Understanding solana’s mint accounts and token accounts,” *Medium*, Feb. 26, 2022. [Online]. Available: https://medium.com/@jorge.londono_31005/understanding-solanas-mint-account-and-token-accounts-546c0590e8e (accessed on Mar. 23, 2022).
- [8] “What is arweave,” [Online]. Available: <https://www.arweave.org/#arweave-intro> (accessed on Mar. 27, 2022).
- [9] M. Luna, “Web development framework (wdf),” *Tech Target*, Jan. 2013. [Online]. Available: <https://www.techtarget.com/searchcontentmanagement/definition/web-development-framework-WDF> (accessed on Apr. 2, 2022).
- [10] “Frontend vs backend,” [Online]. Available: <https://www.geeksforgeeks.org/frontend-vs-backend/> (accessed on Apr. 2, 2022).
- [11] S. Surve. “Why you should use react.js for web development,” [Online]. Available: <https://www.freecodecamp.org/news/why-use-react-for-web-development/> (accessed on Apr. 3, 2022).

- [12] “About node.js,” [Online]. Available: <https://nodejs.org/en/about/> (accessed on Apr. 3, 2022).
- [13] “What is mongodb?” [Online]. Available: <https://www.mongodb.com/en/what-is-mongodb> (accessed on Apr. 4, 2022).
- [14] “Mongoose,” [Online]. Available: <https://mongoosejs.com/> (accessed on Apr. 4, 2022).
- [15] “Schemas,” [Online]. Available: <https://mongoosejs.com/docs/guide.html#schemas> (accessed on Apr. 4, 2022).
- [16] S. M. Michael Cobb, “OAuth,” *Tech Target*, Feb. 2020. [Online]. Available: <https://www.techtarget.com/searchapparchitecture/definition/OAuth> (accessed on Jan. 29, 2022).
- [17] A. Pedroti. “Introduction to nextauth.js,” [Online]. Available: <https://next-auth.js.org/getting-started/introduction> (accessed on Apr. 3, 2022).
- [18] “Discord,” [Online]. Available: <https://discord.com/> (accessed on Jan. 27, 2022).
- [19] “Express,” [Online]. Available: <https://expressjs.com/> (accessed on Apr. 3, 2022).
- [20] “What is a rest api?” (May 2020), [Online]. Available: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (accessed on Apr. 8, 2022).
- [21] B. Lutkevich, “Cryptographic nonce,” *Tech Target*, Oct. 2021. [Online]. Available: <https://www.techtarget.com/searchcontentmanagement/definition/web-development-framework-WDF> (accessed on Apr. 25, 2022).
- [22] P. Gorbachenko. “What are functional and non-functional requirements and how to document these.” (Apr. 2021), [Online]. Available: <https://enkonix.com/blog/functional-requirements-vs-non-functional/> (accessed on May 3, 2022).
- [23] “España digital 2025,” [Online]. Available: https://portal.mineco.gob.es/en-us/ministerio/estrategias/Pages/00_Espana_Digital_2025.aspx (accessed on May 20, 2022).
- [24] “España digital 2025 official document,” [Online]. Available: https://avancedigital.mineco.gob.es/programas-avance-digital/Documents/EspanaDigital_2025_TransicionDigital.pdf (accessed on May 20, 2022).
- [25] “Legal and regulatory framework for blockchain.” (Jun. 7, 2022), [Online]. Available: <https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-blockchain#:~:text=The%5C%20EU%5C%20strongly%5C%20supports%5C%20a,ensure%5C%20consumer%5C%20and%5C%20investor%5C%20protection.> (accessed on Jun. 12, 2022).

- [26] “Salario medio para ingeniero informático en españa, 2022.” (2022), [Online]. Available: <https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico#:~:text=El%20salario%20ingeniero%20inform%C3%A1tico%20promedio,hasta%20%E2%82%AC%2034.300%20al%20a%C3%B1o>. (accessed on Jun. 14, 2022).
- [27] “Sueldos para el puesto de jefe de proyecto en españa.” (2022), [Online]. Available: https://www.glassdoor.es/Sueldos/jefe-de-proyecto-sueldo-SRCH_K00,16.htm (accessed on Jun. 14, 2022).
- [28] “Simple, transparent pricing,designed for you.” (2022), [Online]. Available: <https://www.quicknode.com/pricing> (accessed on Apr. 20, 2022).