Nuno Marques da Silva

Design and Development of a Blockchain-Based Platform for Loyalty Programs

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Jacinto Estima and by Engineers Raul Fonseca and Tiago Ferreira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Nuno Marques da Silva

Design and Development of a Blockchain-Based Platform for Loyalty Programs

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Jacinto Estima and by Engineers Raul Fonseca and Tiago Ferreira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Departamento de Engenharia Informática

Nuno Marques da Silva

Design e Desenvolvimento de uma Plataforma baseada em Blockchain para Programas de Fidelização

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor Jacinto Estima e pelos Engenheiros Raul Fonseca e Tiago Ferreira e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2023

Acknowledgements

I want to express my gratitude to my parents for all the support they have given me along the way and for the many sacrifices they have made so I can concentrate on my work as a student.

I want to thank my friends who accompanied me on this journey and made each day funnier and more enjoyable.

To WIT, I want to thank you for the trust and opportunity. I'd like to thank my advisor Raul Fonseca, tutor Tiago Ferreira, and business analyst Ane Polito for all of their assistance and guidance.

To Professor Jacinto Estima, I want to thank you for all the advice and suggestions, and for the guidance.

Thank you!

Abstract

Loyalty programs are one of the most effective and successful strategies for retaining customers. It is also noticeable that traditional loyalty programs can sometimes lack transparency and support for new partners, potentially making the customers lose their trust and interest. Therefore, it is important for these programs to prioritize transparency and security in transactions, while also facilitating partnerships between businesses in order to accommodate a larger set of benefits.

This dissertation aims to address these challenges by leveraging blockchain technology to enhance security and transparency in loyalty program transactions. The traditional currency (points, stamps, etc.) and rewards are replaced with blockchain tokens, while maintaining the same concept of collecting currency to then redeem a reward. The reward tokens are unique as they can be customized by the business who created them to have different real-world values, ranging from digital discounts or coupons to physical items.

Additionally, a RESTful API was developed to manage these loyalty programs and aims to facilitate partnerships between businesses, enabling loyalty programs to offer a wider range of benefits and attract more customers. Furthermore, this dissertation analyses the throughput of the developed platform to assess its performance and determine its suitability for real-world implementation. The results show that this idea is applicable in the real-world context, despite the limitation found in the throughput tests.

Overall, this research contributes to the improvement of loyalty programs by harnessing blockchain technology and developing an innovative management system. The findings and insights obtained from this work have the potential to enhance the effectiveness and trustworthiness of loyalty programs in various industries.

Keywords

Loyalty Programs, Blockchain, Transparency, REST API

Resumo

Os programas de fidelização são uma das estratégias mais eficazes e bem sucedidas para reter clientes. É também evidente que os programas de fidelização tradicionais por vezes possuem falta de transparência e suporte para novos parceiros, o que pode resultar na perda de confiança e interesse por parte dos clientes. Assim, é importante dar prioridade aos aspetos de transparência e segurança nas transações, e ao mesmo tempo promover a facilidade de desenvolver parcerias entre empresas, para que estas consigam oferecer um maior conjunto de benefícios nos seus programas.

Esta dissertação tem como objetivo abordar estes desafios baseando-se na tecnologia *blockchain*. Aproveitando o mesmo conceito de acumular moeda (pontos, selos, etc..) dos programas tradicionais para eventualmente redimir uma recompensa, podemos substituir a moeda e a recompensa por *tokens* da *blockchain*. Os *tokens* de recompensa podem ser personalizados pela empresa que os criou para poder ter diferentes valores no mundo real, o que permite fazer com que possam representar desde cupões ou descontos em formato digital até bens físicos.

Além disso, foi desenvolvida uma *API RESTful* para gerir estes programas de fidelização em *blockchain* e facilitar as parcerias entre empresas, dando-lhes a oportunidade de oferecer uma maior variedade de recompensas e, consequentemente, atrair mais clientes. A plataforma desenvolvida teve também a sua capacidade de *throughput* testada e analisada para determinar se o desempenho é suficiente para que possa ser implementada em contexto real. Os resultados mostram que esta ideia é aplicável no contexto do mundo real, apesar da limitação encontrada nos testes de *throughput*.

Em suma, esta pesquisa contribui para a melhoria dos programas de fidelização fazendo proveito da tecnologia da *blockchain* e ao ser desenvolvido um sistema inovador. As descobertas obtidas têm o potencial para aumentar a eficácia e a credibilidade dos programas de fidelização em várias indústrias.

Palavras-Chave

Programa de Fidelização, Blockchain, Transparência, REST API

Contents

Acronyms

ABI	Application Binary Interface.
API	Application Programming Interface.
CU	Compute Unit.
dApps	Decentralized Applications.
DPoS	Delegated Proof of Stake.
IPFS	InterPlanetary File System.
JWT	JSON Web Token.
LPoS	Liquid Proof of Stake.
NFT	Non-Fungible Token.
PoET	Proof of Elapsed Time.
PoS	Proof of Stake.
PoW	Proof of Work.
SDK	Software Development Kit.
TPS	Transactions per Second.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Client retention is crucial for any business to maintain sustainability, as acquiring new customers is often more expensive than retaining existing ones. Brands prioritize building solid relationships with their customers to ensure long-term success. Loyalty programs are among the most successful strategies for retaining customers [1]. Research shows that approximately 78% of consumers are more likely to continue doing business with a company if it offers a loyalty program. Additionally, around 65% of customers claim that their spending is influenced to maximize the benefits received from these programs [2].

On average, customers participate in 16.6 loyalty programs but only actively use 7.6 of them, indicating that they are truly loyal to only 46% of the programs they participate in. This leaves 9 programs that a customer could easily leave and switch to a competitor if a better offer comes along [2].

1.2 Motivation and Objectives

The high proportion of inactive customers can be attributed to loyalty programs whose benefits and rewards are not appealing to clients. Furthermore, the abundance of available programs can overwhelm customers, potentially causing them to become confused and forgetful about how a particular program functions or their own membership status.

To tackle the issue of unappealing rewards, companies often try to increase the variety of rewards and benefits by forming partnerships with other businesses. However, sharing personal information with intermediaries can raise concerns around customer privacy and increase risks [3]. Traditional loyalty programs can also lack transparency [4], with issues like fake stamps and tampering with point systems, to name just a few examples. Such problems occur when only one party has direct access to the records.

The prototype developed in this project aims to address every issue listed above by leveragin the advantages of blockchain technology. Firstly, all loyalty programs are centralized in a single platform, allowing users to easily manage and utilize them. Additionally, blockchain technology provides the security and transparency sought by customers, preventing transaction forgery and unauthorized access to their assets, and eliminates the need for businesses to share customers' personal information - as customers are represented by public addresses - and minimizes the risk of information leakage while reducing the costs due to the absence of intermediaries [3, 4]. Table 1.1 summarizes the problems encountered in traditional loyalty programs and the Blockchain approach to tackle them.

Problems	Blockchain's approach
Customers' personal information exchange	Each customer is represented by a public address
Intermediary necessity	No intermediaries: no risk of leakage, and low costs

Table 1.1: Blockchain solutions to traditional loyalty program problems.

On a high level, the platform allow companies to deploy and manage their customized loyalty programs, and trade with their clients. On the other hand, customers are able to sign up for different loyalty programs, and trade with the relevant company or companies (in the case of a multiple company program), or even with other clients.

The end result was achieved through the development of the following components:

- Smart Contracts to handle programs, tokens and rewards;
- Token metadata storage;
- Application Programming Interface (API) to allow interaction with the described platform.

1.3 Methodology

This project follows the Design Science Research (DSR) methodology, which aims to address practical problems through the creation and evaluation of innovative artifacts. The methodology encompasses a series of iterative cycles that include problem identification, solution design, artifact development, evaluation, and reflection.

For instance, each feature in the project followed this iterative process. Firstly, the problem was identified - the absence of a specific feature. Then, a solution was designed and carefully planned for implementation. Subsequently, the feature was developed and prepared for testing and evaluation. Afterward, a critical analysis and reflection were conducted to assess the design decisions, address any implementation challenges, and evaluate the outcomes of the feature.

The DSR methodology provides a structured and systematic approach to addressing practical problems by combining scientific rigor with design principles. It allows for the creation of valuable knowledge and practical solutions with real-world impact [5].

1.4 Document Outline

This document consists of nine chapters, each addressing different stages of the project.

In the first chapter, an introduction is provided, presenting the a contextualization, problem statement, and objectives to be achieved by the end of the internship.

The second chapter dives into the background and state of the art information about loyalty programs, blockchain, and explores how these two concepts can be combined.

Moving on to the third chapter, the proposed solution and its underlying idea are presented.

The fourth chapter focuses on the requirements analysis, describing the specific requirements for the platform and defining the success criteria.

In the fifth chapter, the system design and architecture are discussed, along with an examination of the identified risks.

The sixth chapter explains the adopted development methodology and analyzes the project planning, including both first and second semester plans.

Chapter seven delves into the development process, covering topics such as the development environment, smart contract implementation, the structure of the database, and backend and frontend development.

In the eighth chapter, the testing and validation phases take place, where the developed platform is tested and verified for validation of the requirements.

Finally, in the ninth chapter, a conclusion is provided, summarizing the internship experience and highlighting future work, including suggestions for further development of the platform.

Chapter 2

Background and State of the art

The present chapter aims to provide an overview of the concepts of loyalty programs and blockchain technology. By exploring these topics, readers will gain a better understanding of the context surrounding the theme of this project. This chapter commences by examining loyalty programs, their nature, how they operate, and providing real-world examples. Following that, the blockchain is introduced, what it is, it's mechanics and advantages, and various blockchain technologies are explored. Lastly, an explanation on how loyalty programs are leveraging blockchain technology in the present is discussed, along with some more noteworthy examples.

2.1 Loyalty Programs

Loyalty programs are an important instrument to improve brand loyalty by creating stronger economic and habitual relationships [6]. These programs offer benefits, prizes, and discounts to customers in exchange for their loyalty and ongoing purchases. The companies promote these programs as they show great success in retaining customers [1].

The type of loyalty programs is determined by the nature of interactions between the company and the customers. This leads to a wide variety of structures [7–9] that can be classified into four main categories:

- Point System (Section 2.1.1);
- Tier System (Section 2.1.2);
- Subscription System (Section 2.1.3);
- Value-based System (Section 2.1.4).

2.1.1 Point System

This is the most traditional and simple system, where a currency such as points, collectibles, or stamps is used and traded for the final reward. The more money customers spend, the more points they receive in return. Customers earn points by making purchases, and as soon as these points are available for them to spend, they can get discounts, vouchers, free rewards, or even some form of cashback.

Example: Vodafone Clube Viva [10]

Vodafone, a telecommunications company, gives their customers two points for every euro they spend on a subscription for one or more of their services. These points can then be redeemed for discounts on communication devices (e.g., cellphones, tablets, hotspots, SIM cards) and accessories. A schematic representation of this loyalty program is shown in Figure 2.1.

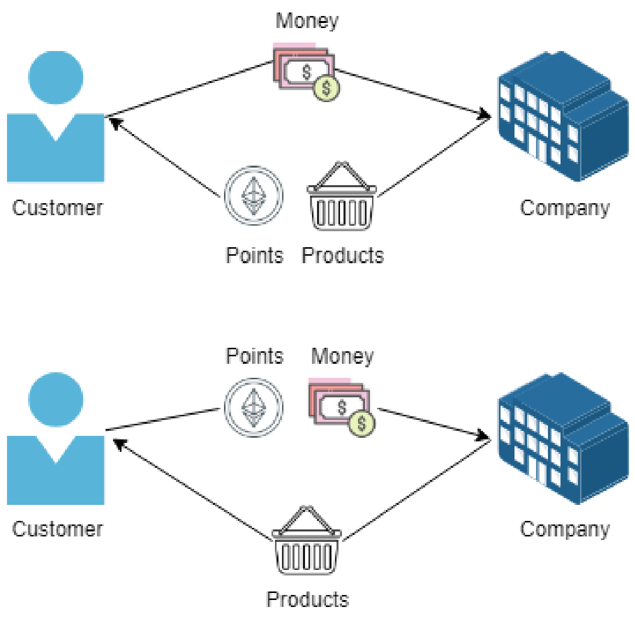


Figure 2.1: Schematic representation of Vodafone Clube Viva loyalty program.

2.1.2 Tier System

Tier systems are structured as a ranking system where customers begin at the base-level and gain access to initial benefits upon joining. As customers continue to engage with the company and make purchases, they have the opportunity to progress through the tiers, unlocking increasingly valuable rewards. These rewards often include discounts, exclusive deals, and complimentary products or services.

Example: Nespresso [11]

The Nespresso Company uses a tiered loyalty program with 3 levels named *Connoisseur*, *Expert* and *Ambassador* (Figure 2.2). As the clients enlist themselves in the program, they are given the *Connoisseur* tier, where they receive the following basic benefits: free deliveries, a decalcification kit, and the ability to schedule future orders.

Then, after purchasing every year the equivalent of approximately 2 or 3 capsules per day, on average, and being in the program for more than 5 years, they rank up to the *Expert* tier. At this level, they continue to receive all the *Connoisseur* benefits, a 20€ discount on accessories, and a free product.

Reaching the mark of 10 consecutive years of purchasing from Nespresso and maintaining an average of 3 or more capsules per day throughout each year, customers ascend to the highest *Ambassador* tier. As *Ambassadors*, they unlock a host of exclusive benefits, including all the perks from previous tiers, access to personalized deals, and a brand new coffee machine.

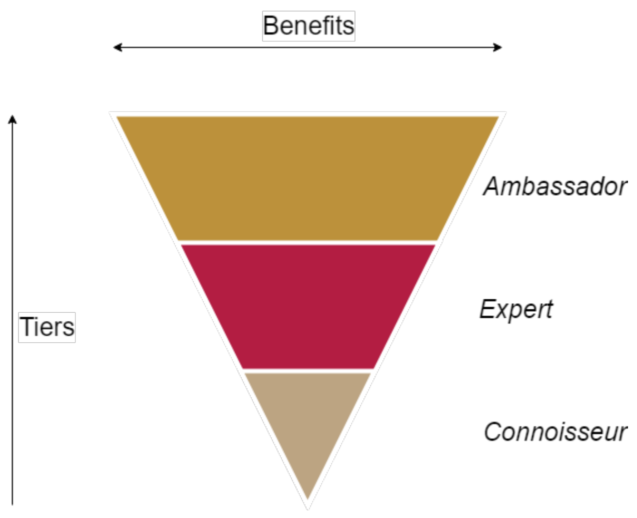


Figure 2.2: Schematic representation of Nespresso's loyalty program.

2.1.3 Subscription System

The subscription program allows clients to enroll and enjoy a range of exclusive services, discounts, and access to exclusive content. As the name suggests, clients pay a regular fee, which can be monthly, quarterly, semiannual, or even annual, to maintain their subscription and continue receiving these exclusive benefits.

Example: Amazon Prime [12]

Amazon has a subscription-based loyalty program that benefits customers who frequently use products from the Amazon ecosystem. For a monthly or annual fee, a Portuguese client will get:

- Free shipping and faster deliveries on Amazon purchases;
- Access to Prime Video, where he can watch movies and series for free;
- Prime Gaming, which gives him a free Twitch subscription and access to exclusive content for a number of games;
- Prime Priority Access provides access to Flash Offers 30 minutes prior to launch;
- Amazon Photos, offers unlimited storage capacity for photos in the Amazon Drive.

2.1.4 Value-based System

Value-based loyalty programs are implemented by businesses to build a string emotional bond with their customers. Unlike traditional loyalty programs where the vendor gives something back to the client, value-based programs focus on fostering a deeper connection by allowing customers to contribute to charitable organizations that align with their values and interests.

Example: Amazon Smile [13]

Amazon also includes a value-based program in their portfolio called Amazon Smile where customers can select one organization from a list of charities to support. Then, as shown in Figure 2.3, Amazon donates 0.5% of the total purchase price to the designated charity, assisting those who are most in need.

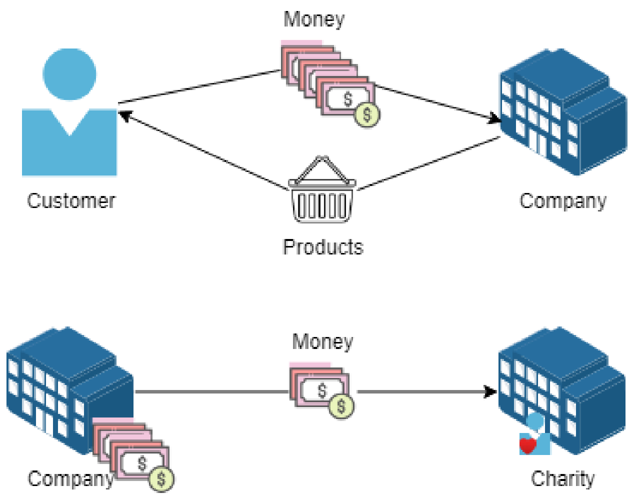


Figure 2.3: Schematic representation of Amazon Smile.

2.2 Blockchain

In 2008, an individual or group known as Satoshi Nakamoto introduced the concept of a decentralized blockchain. One year later, this concept was materialized in the creation of Bitcoin [14].

A blockchain, as its name suggests, is a series of interconnected blocks called ledger, each of which contains the information that needs to be recorded along with the hash of the preceding block, as depicted in Figure 2.4 [15]. This inherent structure makes the blockchain immutable, because changing a block in the middle of the ledger would cause the entire chain to break, with everyone noticing it [15].

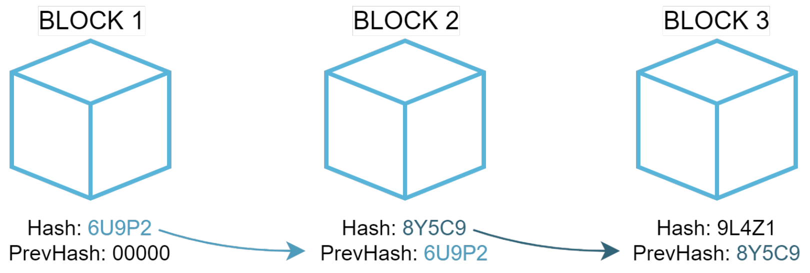


Figure 2.4: Representation of the ledger.

Periodically, depending on the blockchain technology, a new block is added to the chain after its transactions are verified and validated by a node or a participant called validator that in return receives some reward. In order to validate a new block on the ledger, validators rely on a synchronizing mechanism to handle concurrency and maintain consistency across all nodes. This enables them to verify and finalize transactions effectively (Section 2.2.1).

To achieve decentralization (Figure 2.5b) and overcome the challenges associated with centralization, wit is essential to establish a network of interconnected peers that each maintain a copy of the ledger [14].

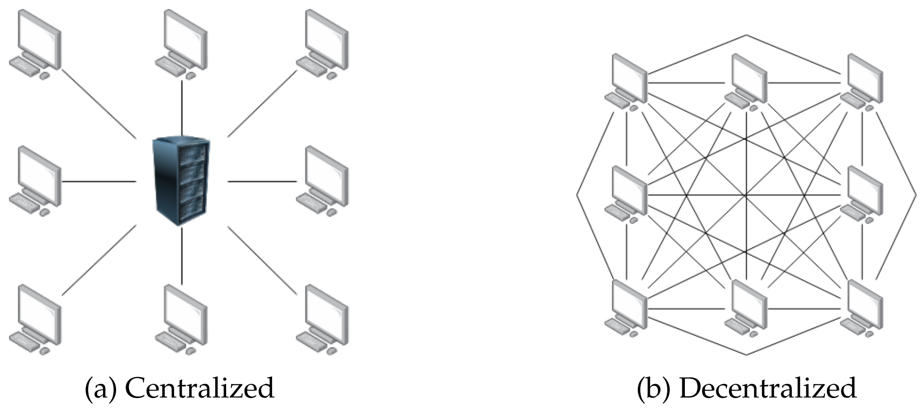


Figure 2.5: Representation of centralized and decentralized networks.

And since there are no intermediaries among transactions, how can anyone trust that the other entity will uphold their end of the deal? Well, thanks to smart contracts (Section 2.2.2), the entire transaction commits as a single act, safeguarding both the buyer and the seller [16].

In terms of control and governance, blockchains can be categorized into four main types: (i) public, (ii) private, (iii) hybrid, or (iv) consortium, as depicted in Figure 2.6 [17, 18].

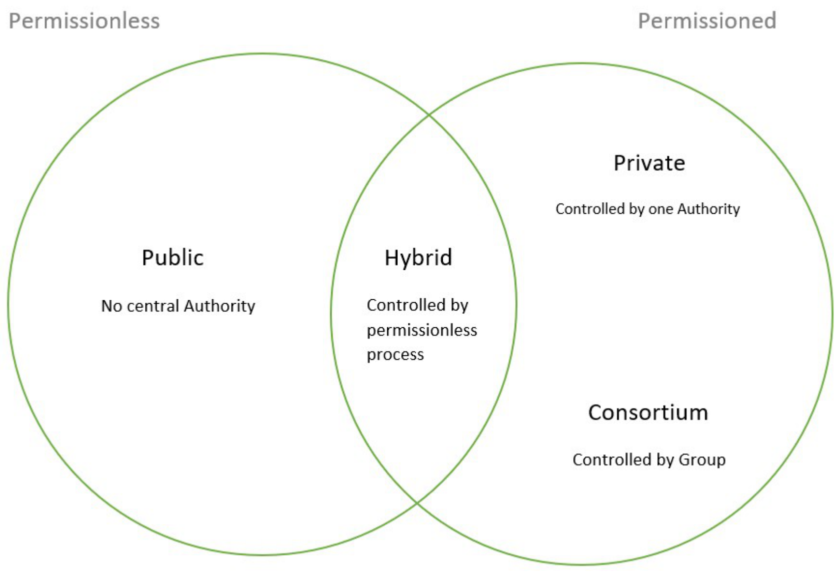


Figure 2.6: Types of Blockchain.

Public blockchains have no central authority, which means nobody owns them. Due to their decentralized nature, anyone with access to the internet can join and take part in the network. Additionally, given their permissionless nature, everyone can see and follow every transaction.

As opposed to public blockchains, private blockchains are centralized, meaning that one authority — which could be one person or entity — controls the entire network. The authority has the power to choose the participants of the network (permissioned), completely concealing the transactions from the general public.

Consortium blockchains, which are also permissioned, allow a group of authorities (i.e., two or more people or entities) to determine who can access their network. In this scenario, the network's architecture is no longer centralized, and the more decentralized it is, the more authorities it has and the more secure it is against corruption of an authority.

Hybrid blockchains combine both public and private blockchain features. A public permissionless network is set up alongside a private permissioned one, allowing them to hide transactions from the general public. An authority can invite users to the private network but cannot change any transactions.

Each has advantages and disadvantages, which explains why they are used in different ways. Table 2.1 lists the benefits and drawbacks of the different types of blockchains [19].

	Public	**Private**	**Hybrid**	**Consortium**
Advantages	Independence Transparency Trust	Access control Performance	Access control Performance Scalability	Access control Scalability Security
Disadvantages	Performance Scalability Security	Trust Auditability	Tranparency Upgrading	Transparency
Use Cases	Cryptocurrency Document validation	Supply chain Asset ownership	Medical records Real estate	Banking Research Supply chain

Table 2.1: Blockchain types: advantages and disadvantages.

2.2.1 Consensus Mechanisms

Consensus occurs when all the peers, or nodes, in a network reach an agreement on the current state of the ledger, ensuring that there is only one authoritative version. Following are a few of the numerous mechanisms or algorithms available to reach consensus:

Proof of Work (PoW)

Nodes compete with one another in order to solve extremely difficult puzzles, and the first to do so earns the right to produce the new block and verify its transactions. Once produced, the node is rewarded for his efforts (block rewards - newly generated coins). This process, also known as mining, although secure and completely decentralized, requires large amounts of computational resources for a node to beat his competition and, consequently, energy. The costs of maintaining a node are notoriously high, pushing away new miners or nodes, leading to concerns about centralization (fewer nodes) and scalability (too expensive) [20].

Proof of Stake (PoS)

To resolve the previously mentioned PoW problems, a more recent algorithm was created. The action is known as staking, and the nodes are required to pledge a stake of a currency in exchange for the opportunity to be selected at random as a validator. A node has a better chance of being chosen if he stakes more coins. The chosen validator will go on to verify and validate the block's transactions, insert the new block into the ledger, and collect a reward (i.e., the transaction fees paid by the actors involved in those transactions).

PoS is viewed as a more sustainable and environmentally-friendly alternative to PoW, and more secure against 51% attack. However, because nodes with larger stakes are favored and nodes with smaller stakes have a very slim chance of being rewarded, there is a tendency toward centralization [20].

Delegated Proof of Stake (DPoS)

This algorithm forks from PoS and achieves consensus through a reputation-based voting system. Users of the network stake their coins to be able to vote in witnesses (also known as block producers). The votes are then weighted according to the size of each voter's stake, and a list of the top witnesses is elected (the size of the list varies from platform to platform). The responsibility of validating and producing a new block circles through the elected block producers. A block producer can also be voted out and replaced if an attempt to validate a fraudulent transaction is made. Despite being less popular than PoS, DPoS is a more effective, democratic, and financially inclusive alternative [20].

RAFT

The RAFT algorithm was designed for better understandability of how consensus can be achieved considering its predecessor PAXOS (another consensus algorithm, but very complex, thus not being explored here). To begin with, RAFT acknowledges that each node can be in one of three states: leader, candidate, or follower. The leader is elected through an election process and leader node serves as the central coordinator, receiving client requests and logging them to the follower nodes. Once the majority of followers have registered and processed the logs provided by the leader, they can be considered committed and be applied. A new leader is elected if the current one fails or becomes unavailable [21].

Proof of Elapsed Time (PoET)

The PoET algorithm aims to achieve consensus in a distributed network by utilizing a fair and efficient lottery-based selection process. Each node participant is assigned a random wait time for it to "sleep", and the first "waking up" wins the permission to propose and produce a new block. Then, the new information is broadcasted throughout the whole network and this process repeats.

This algorithm promotes efficiency by reducing the need for resource-intensive tasks like PoW mining. Instead of expending computational power, participants in PoET simply wait for their assigned time period to pass, significantly reducing energy consumption and computational requirements, while maintaining a fairness in the "leader" selection process [22].

Liquid Proof of Stake (LPoS)

The LPoS algorithm was designed to address the limitations of traditional PoS algorithms, such as the lack of liquidity and the concentration of power among a few stakeholders. The key concept of LPoS is the delegation of stake which allows stakeholders to delegate their tokens to a trusted validator in the network. By delegating their stake, stakeholders can participate in the transactoin validation process and earn rewards without the need to actively run a validator node themselves. Another important advantage of LPoS is its liquidity. Unlike some PoS algorithms where tokens are locked up for a certain period, LPoS allows stakeholders to freely unstake their tokens whenever they choose while still receiving rewards for their participation [23].

Tendermint

At its core, Tendermint utilizes a Byzantine Fault Tolerant as it assumes that up to one-third of the participating nodes can be faulty or malicious, and the algorithm aims to achieve consensus among the non-faulty nodes. The key components of the Tendermint algorithm include a round-based consensus process, a leader-based block proposal mechanism, and a voting mechanism for block validation.

Every round, the "leader" role is assigned to a node, allowing him to propose a block of transactions. Once this block is broadcasted to the network, other nodes validate it by voting on its validity. The block is committed if at least two-thirds of the nodes agree on its validity.

To prevent malicious behavior, Tendermint employs a punishment mechanism. If a node behaves dishonestly, it can be detected by other nodes through its signature, and it may then face penalties, such as being temporarily or permanently excluded from the consensus process [24].

2.2.2 Smart Contracts

A smart contract is an agreement between two or more parties over the internet. It is self-executed and thus eliminates the need for intermediaries. It is programmed and supports various currencies, tokens, or kinds of property in addition to performing a number of different tasks like buying, selling, and trading those assets. To write a smart contract, we need to identify (i) the parties involved in the transaction, (ii) the type of assets exchanged, and (iii) the conditions of the transaction. Following deployment, the smart contract, like a transaction, is stored on the blockchain, is immutable, ensuring security, and is accessible to all users, ensuring transparency [25, 26]. The potential applications are endless, for instance, if we assume that a Non-Fungible Token (NFT) can represent a physical document, such as a house deed, the holder could sell their home by simply exchanging their NFT for cash.

2.2.3 Tokens

A token is a representation of something in the blockchain: money, time, services, shares in a company, anything [27]. There are native tokens and contract tokens. The native tokens serve as the universal currency within that blockchain platform, they are used to stake, pay transaction fees, and are rewarded to the validators, while the contract tokens are tokens produced by a smart contract. The smart contract links addresses that represent one person/entity (deeper explanation in section 2.2.4) to their balance of that respective token, like a common database, and has some methods to add to or subtract from those balances: "sending tokens" actually means "calling a method on a smart contract that subtracts and adds a certain value" [28].

There is a significant difference between having two shares of a company and two deeds of ownership: each share is equal to all others, whereas houses are not always equal - one deed of ownership is worth more than the other. This property is called fungibility [28]:

- Fungible Tokens - goods are equivalent and interchangeable;
- NFTs - goods are unique and distinct.

Because of their non-fungible nature, NFTs require special attention: a block of the ledger has a certain size, so storing large data like an image on the blockchain is too expensive and not scalable. The solution is to save all that metadata (like the image and its name and description) somewhere else and store an identifier or link to that metadata on the blockchain.

The InterPlanetary File System (IPFS) is a decentralized peer-to-peer file sharing and storage system. It aims to create a global, distributed network where users can store and retrieve files in a more efficient and resilient manner compared to traditional centralized systems. In IPFS, files are identified using a content-addressable approach, where each file is assigned a unique cryptographic hash based on its content. This ensures that identical files are always represented by the same hash, enabling efficient deduplication and eliminating the need for multiple copies of the same file, and also allows files to remain accessible even if their original hosting nodes go offline, since they are referenced by their content rather their location [29, 30].

Overall, IPFS presents an approach to file storage and sharing, leveraging decentralization, content addressing, and peer-to-peer networking to create a more efficient, resilient, and censorship-resistant file system, ideal to host the token metadata mentioned earlier.

Token standards are the conditions, guidelines, and functions that define how a token can be created, issued, and deployed on blockchain platforms that support smart contracts. Table 2.2 displays the most prevalent contracts implemented on the Ethereum blockchain [31]:

Standard Name	Creation Date	Description
ERC-20	19-11-2015	Supports fungible tokens.
ERC-721	24-01-2018	Supports NFTs.
ERC-1155	17-06-2018	Supports both fungible and non-fungible tokens.

Table 2.2: Most Common ERC Token Standards.

2.2.4 Token Storage

A user is represented by a pair of keys in the blockchain world: a public key that serves as the address for other users to send tokens to and a private key used to authenticate the user and authorize transactions. Such pairs of keys can be stored in digital wallets.

Digital wallets allow a user to create and manage these pairs of public and private keys. They can be of two natures: (i) custodial wallets, a user-friendly approach in which both keys are held by a third party; (ii) non-custodial wallets, the more advanced and secure way: both keys are fully controlled by the user, and he cannot forget the private key.

2.2.5 Blockchain Platforms

This section examines several blockchain platforms that support smart contracts because every platform has advantages and disadvantages, implying that each was created with a specific purpose.

Ethereum [32]

It was launched in 2015 and is public and permissionless. Its native token is Ether (ETH), and the consensus mechanism is PoS. Solidity is the main programming language used in smart contracts. And Ethereum has the largest programming community [33]. Interoperability is another key feature. Ethereum appears to be the foundation for many scaling options, resulting in a large ecosystem with many blockchains that can exchange information among them. The main disadvantage is the transaction speed. As demand exceeds supply, the network is extremely congested, driving up transaction fees.

Hyperledger Fabric [34] and Hyperledger Sawtooth [35]

Both of these platforms belong to the same family and are quite similar. They are neither public nor permissionless but can be configured to be either private, hybrid, or consortium. Fabric uses RAFT for consensus, whereas Sawtooth can use RAFT and Practical Byzantine Fault Tolerance (PBFT) or switches to PoET in real time in a running network.

Since these platforms are not public, companies have to deploy and maintain the network, but on the other hand, transaction fees would be free and transaction speeds would be maximized (such as 3500 Transactions per Second (TPS)). Both support various programming languages for their smart contracts, as detailed in Table 2.3.

Programming Language	Blockchain Platform	
	Hyperledger Fabric	Hyperledger Sawtooth
Go	X	X
JavaScript	X	X
Java	X	X
Rust		X
Python		X
C++		X

Table 2.3: Hyperledger Fabric and Hyperledger Sawtooth's smart contract programming languages.

Corda [36]

Corda is a consortium blockchain developed by R3 with a main focus on performance and instantaneous transactions. Conventional blockchains use a system of blocks and block production with multiple transactions within them, while in Corda, as soon as one transaction is submitted, and if consensus is achieved, the transaction is deployed to the ledger in real-time. There is no need to wait for transactions to come along to form an entire block or a "block interval". For validators to achieve consensus, validity and uniqueness must be verified: validity determines if a transaction is accepted by the smart contract it references, and uniqueness prevents double-spending. The smart contract programming languages supported are Java, Kotlin, and any other Java Virtual Machine (JVM) compatible language.

Tezos [37]

Tezos is a public blockchain platform, and its native token is Tez (XTZ). The founders didn't want it to be forked into another variation with another native token with two distinct prizes, so it uses a system where anyone with XTZ cryptocurrency can vote on possible changes to Tezos rules, and once decided, the software would automatically update to ensure that changes are implemented [38].

Another interesting fact is that the consensus mechanism, LPoS, is an evolution of DPoS where the idea is to dilute even more the activity and increase inclusion among users to participate in the network. Participants, in order to become block producers, need to stake at least 6000 XTZ (a roll) in a process called baking; if they don't have that amount, they can delegate their tokens to other bakers with the goal of winning some rewards (more tokens) [39].

The smart contract language is the Michelson language, a low-level language that is similar to assembly language. Like assembly, there are a number of high-level languages that compile into the Michelson language (e.g., SmartPy, Archetype, or Ligo) [40].

EOSIO [41]

Founded in 2017 by Block.One [42], EOSIO is a permissioned hybrid blockchain platform with EOS as the native token. The consensus mechanism is composed of a two-layer mechanism: firstly, there is a DPoS layer to generate a list of 21 block producers and handle the voting system, and for block production and validation, the asynchronous Byzantine Fault Tolerant (aBFT) layer enters into action. These two processes are independent and can be executed in parallel.

There are no transaction fees. Developers need to buy CPU, memory, and network bandwidth with EOS for their Decentralized Applications (dApps) to interact with the network. Users can also delegate or rent such resources.

EOS tokens are limitless, which means they can be minted indefinitely, but they can also be burned and deleted. Currently there is an inflation rate of 3% per year, where 1% is distributed through the block producers and the remaining 2% is headed into a savings account for future sponsorships, funding, and development.

Officially, the smart contract programming language is C++, however there is an award-winning community project that bridges the gap between Solidity and the native language called "eosio.evm" [43].

It is possible to use JavaScript, Swift, and Java to interact with the blockchain. But there are also some issues: (i) Windows Operating System is not officially supported, and (ii) EOSIO Java Software Development Kit (SDK) [44] has not been updated in over 2 years, meaning it is outdated.

Stellar [45]

Stellar is a public and permissionless blockchain, and its native token is called Lumen (XLM). Stellar's purpose is to facilitate fast, low-cost, and secure transactions, with a focus on cross-border payments and financial services. It offers built-in features that allow for seamless currency exchange, making it easier and more cost-effective to send money across different currencies. Stellar's native cryptocurrency, XLM, serves as both a means of exchange and an anti-spam mechanism to prevent network abuse.

Another key feature of Stellar is its ability to facilitate the issuance and transfer of digital assets, including cryptocurrencies and tokens. It provides users with a platform to create and manage their own assets, which can represent various types of value, such as currencies, commodities, or even real-world assets like stocks or bonds.

Polygon [46]

Polygon is a layer-two scaling platform that aims to enhance the scalability of Ethereum while retaining its core features. It achieves this by operating as an Ethereum sidechain, which means it runs parallel to Ethereum's main blockchain and is connected to it. This connection enables seamless interoperability and the exchange of information between the two networks.

Polygon's main network, called Polygon POS, utilizes a PoS consensus mechanism. This means that users can stake their MATIC tokens (the native token of Polygon) and potentially become validators, responsible for validating and adding blocks to the blockchain. Alternatively, users have the option to delegate their tokens to other validators, thereby participating in the consensus process indirectly.

A few advantages of Polygon are its seamless integration with Ethereum, the transaction speed, and the low costs per transaction. Because it is built as an Ethereum sidechain, developers can utilize familiar tools and programming languages, particularly Solidity, which is Ethereum's primary smart contract language. This compatibility allows for easy migration of existing Ethereum projects to Polygon and promotes developer adoption. Also, by leveraging its sidechain architecture, Polygon offers faster transaction speeds and significantly reduced fees compared to the Ethereum network. This improvement allows users to enjoy a more efficient and cost-effective experience when interacting with dApps and executing transactions.

Cosmos [47]

Cosmos, often referred as the "internet of blockchains" [47], aims to create a network of interoperable networks, allowing transactions between them. Each new blockchain created with Cosmos (referred to as a "zone") is then connected to the Cosmos Hub, a PoS blockchain whose native token is ATOM.

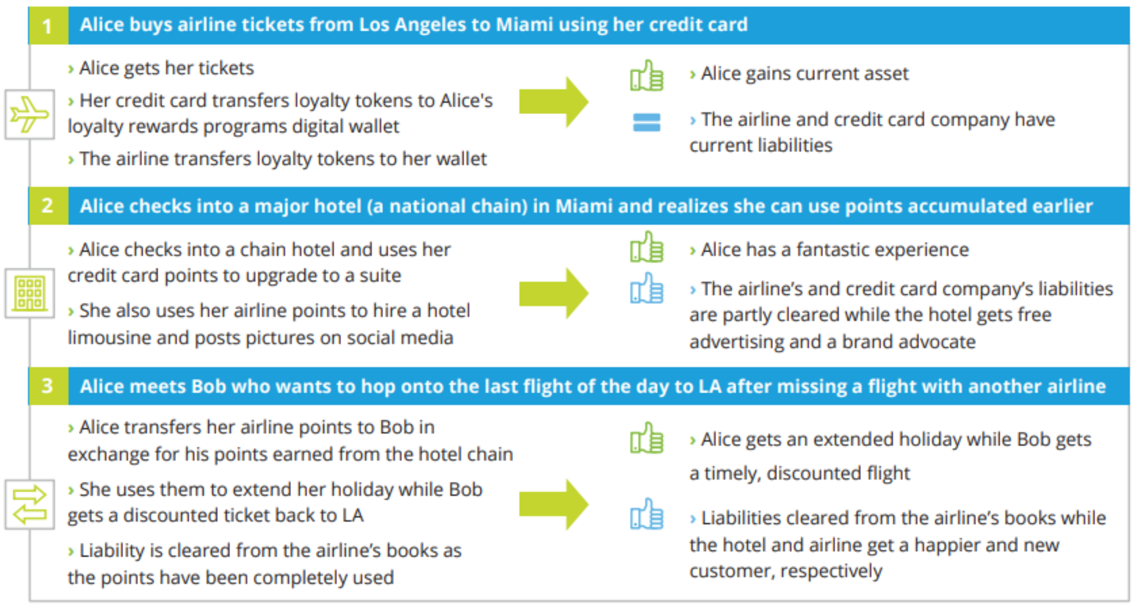
The Cosmos Hub was the first blockchain to be launched within the Cosmos network. Its primary role is to be the intermediary between all the zones. Each zone is able to carry out its essential functions on its own and can communicate with other zones via the Inter-Blockchain Communication Protocol (IBC) [48].

Since this process seems very complex, the Cosmos team developed the Cosmos SDK that allows developers to build blockchains using the Tendermint consensus mechanism. The SDK minimizes complexity by offering the most common functionalities like staking, governance, and tokens [48].

2.3 Blockchain based Loyalty Programs

As already mentioned, blockchain-based loyalty programs enable user interaction through a system without middlemen and without jeopardizing their privacy. Ideally, businesses don't have to worry about scalability and can manage and monitor their programs in real time, leaving great potential for partnerships with other entities: "Blockchain is a system facilitator, not a replacement for an existing system" [3, 6, 27].

After searching on the current market for these programs, it is possible to conclude that all follow some sort of a point system where the token is something spendable, like a coin, a voucher, or a stamp — something that the customer spends in exchange for those benefits [27] (a general example is represented by Figure 2.7 [6]).



1 Alice buys airline tickets from Los Angeles to Miami using her credit card

› Alice gets her tickets
› Her credit card transfers loyalty tokens to Alice's loyalty rewards programs digital wallet
› The airline transfers loyalty tokens to her wallet

› Alice gains current asset
› The airline and credit card company have current liabilities

2 Alice checks into a major hotel (a national chain) in Miami and realizes she can use points accumulated earlier

› Alice checks into a chain hotel and uses her credit card points to upgrade to a suite
› She also uses her airline points to hire a hotel limousine and posts pictures on social media

› Alice has a fantastic experience
› The airline's and credit card company's liabilities are partly cleared while the hotel gets free advertising and a brand advocate

3 Alice meets Bob who wants to hop onto the last flight of the day to LA after missing a flight with another airline

› Alice transfers her airline points to Bob in exchange for his points earned from the hotel chain
› She uses them to extend her holiday while Bob gets a discounted ticket back to LA
› Liability is cleared from the airline's books as the points have been completely used

› Alice gets an extended holiday while Bob gets a timely, discounted flight
› Liabilities cleared from the airline's books while the hotel and airline get a happier and new customer, respectively

Figure 2.7: Typical customer journey with blockchain-based loyalty programs.

Amergent Hospitality Group [49] & Mobivity [50]

Amergent Hospitality Group, the former Chanticleer Holdings Group, are the current owners of Jaybee's Chicken Palace [51], PizzaRev [52], Little Big Burger [53], Burgers Grilled Right [54], and American Burger Co. [55]. In January 2018, they planned a partnership with Mobivity, a publicity and marketing company, to create a blockchain-based loyalty program where their customers would receive Mobivity Merit (the token name) that could be used in those restaurants to make purchases or traded with other users: "Mobivity Merit is real cryptocurrency, leveraging the same infrastructure and principles of Bitcoin, Ethereum, Ripple, Litecoin, and more" - Michael D. Pruitt, Amergent Hospitality Group CEO [56].

American Express [57] & Boxed [58]

In May 2019, American Express, an American bank, partnered with Boxed, a supermarket chain, to create a loyalty program. When Boxed customers bought their groceries, if they paid with a certain special American Express card, they would win points that could be exchanged for discounts in Boxed stores. Today, this partnership has expanded, and many companies like Amazon [59], Best Buy [60], Staples [61], Ticketmaster [62], and more have joined [63]. All transactions are registered on a private Hyperledger blockchain.

Singapore Airlines [64]

In June 2018, Singapore Airlines (SIA) became the first company in the airline sector with a blockchain-based loyalty program. Customers earn tokens called miles by flying with SIA; those tokens can be used, via the application Kris+, to redeem prizes and discounts at more than 1,000 different companies that have partnered up. It is also possible to convert miles to other loyalty program currencies (from other companies) that SIA allows [65]. All transactions are registered on a SIA-owned private blockchain [66].

Etisalat [67]

In May 2017, Etisalat, a telecommunications company in the United Arab Emirates (UAE), launched the Stisalat Smiles app [68]. When paying the monthly bill for mobile, TV, and/or internet services, customers receive points that can be redeemed throughout a pool of partners in the form of discounts and prizes [69].

The Etisalat Smiles program received an upgrade in December 2020, allowing customers to add other loyalty programs to the Smiles app and convert points from one program to others (if the loyalty program being added signs up for it) [70].

Chapter 3

The Proposed Blockahin-based Loyalty Platform

This chapter presents the idea and concept of our proposed solution for enhancing loyalty programs. The first part focuses on discussing potential improvements to existing blockchain-based loyalty programs, while the second part provides an overview of the developed platform.

3.1 Loyalty Program

In a point system, tokens can function as currency, allowing customers to accumulate a balance of tokens that can be exchanged for rewards at a later time. These tokens are generated in batches and assigned unique IDs for storage in the blockchain. Each batch can be associated with an expiration date, either indefinite or for a specific period. Token owners can easily engage in transactions with other users, as illustrated in Figure 3.2. When a token reaches its expiration date, it remains in the customer's possession but loses its value, as depicted in Figure 3.1. This approach enables flexible token usage while ensuring that expired tokens cannot be redeemed.

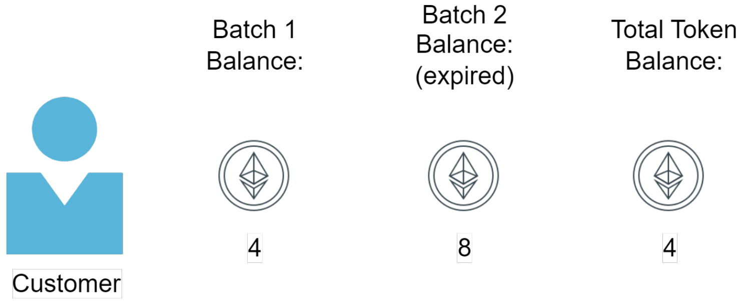


Figure 3.1: Token balance calculation.

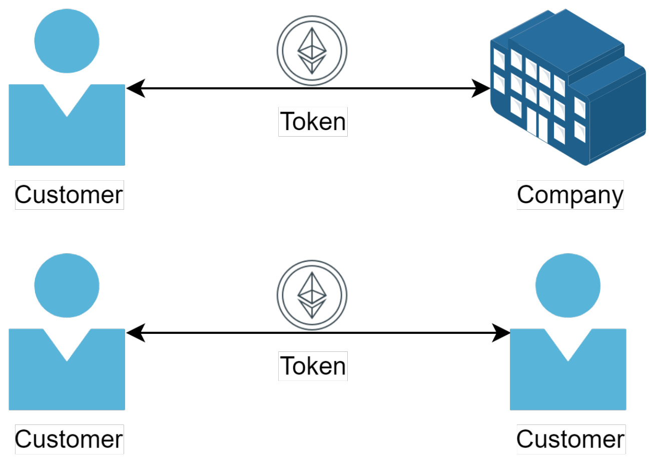


Figure 3.2: Token transmission in a point blockchain-based loyalty program.

As previously discussed in Section 2.3, all the analysed blockchain-based programs revolved around some form of point system. This led us to explore the concept of subscription and tier systems, which can be considered as essentially a membership with an expiration date. The underlying idea is that when a user possesses a NFT with a designated expiration date, it grants them the same privileges as a member of the corresponding tier or subscription (refer to Figure 3.3). To ensure exclusive access, the token can only be transferred between the vendor (the token's issuer) and the consumer, thereby preventing unauthorized sharing (as illustrated in Figure 3.4). Once a token expires, it can still be retained by the customer, but it loses its value.

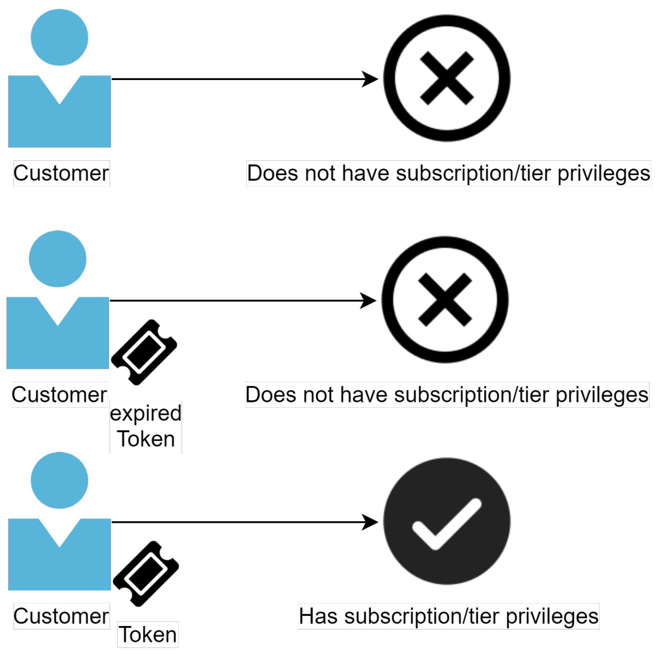

Customer
Does not have subscription/tier privileges
Customer
expired Token
Does not have subscription/tier privileges
Customer
Token
Has subscription/tier privileges

Figure 3.3: Subscription and tier privilege access.

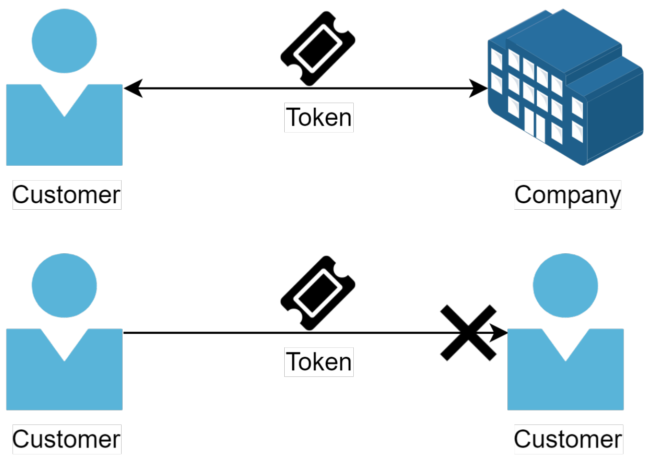


Figure 3.4: Token transmission in a subscription and tier blockchain-based loyalty program.

3.2 The Platform

The developed platform provides a range of features for both vendors and clients. Vendors enjoy the convenience of minting tokens, monitoring and transferring their balances, as well as deploying and managing loyalty programs, including the ability to edit and disable them. On the other hand, clients have the capability to check and transfer their token balance, search for, join/leave, and interact with any program, including the process of redeeming rewards. The platform strives to be user-friendly and accessible, ensuring that even individuals without prior knowledge of blockchain technology can navigate it easily.

Similar to the tokens used as currency within the programs, rewards are also represented by tokens. Redeeming a reward involves exchanging tokens, it is like purchasing a specific token with other tokens. However, the reward tokens differ from the program tokens in that they are not stored within the program tokens smart contract. Instead, they reside in a separate contract created and maintained by the vendor, allowing the vendor to have control over the reward handling in the "real world" (this allows vendors to use reward tokens as coupons or discounts). To facilitate this, the vendor needs to deploy his rewards smart contract and add it to the platform, enabling the platform to mint tokens based on it. It's worth noting that the platform aims to be as user-friendly and accessible as possible, so a default rewards contract is already deployed and available for use by every vendor.

Chapter 4

Requirements Analysis

The purpose of this chapter is to establish the requirements for the platform. Defining these requirements helps describe the features of the platform and enables the creation of a development plan and timeline. This process allows for determining whether there is sufficient time to implement specific features.

The chapter begins by defining the functional requirements, accompanied by user stories to enhance comprehension of each feature. Next, the non-functional requirements are outlined. Finally, the success criteria are presented.

4.1 Functional Requirements

The functional requirements collect every feature associated with the product. However, it is not implied that the final product will include all of them. To provide a comprehensive overview, the requirements are grouped into five modules and listed in Table 4.1, with each feature assigned a priority label based on the MoSCoW scale: (M) Must Have, (S) Should Have, (C) Could Have, and (W) Won't Have This Time [71]. For a more detailed understanding, user stories in Section 4.2 can provide additional information and clarity.

ID	Actor	Feature	Priority
		Authentication	
FR1	Vendor/Client	Log in with email and password credentials	M
FR2	Vendor/Client	Register new account with email and password credentials	M
FR3	Vendor/Client	Log in with MetaMask Wallet	M
FR4	Vendor/Client	Register new account with MetaMask Wallet	M

Table 4.1 continued from previous page

ID	Actor	Feature	Priority
		Tokens	
FR5	Vendor	Create a token template (only upload the token metadata, including image, to the IPFS network)	M
FR6	Vendor	Soft delete a token template	W
FR7	Vendor/Client	Fetch all the details of a token by its ID	M
FR8	Vendor/Client	Fetch the list of token templates given a list of IDs	S
FR9	Vendor/Client	Fetch the list of token templates created by a certain Vendor	M
FR10	Vendor/Client	Fetch the list of token templates that a certain user has balance of	M
FR11	Vendor/Client	Fetch the balance of a token owned by a certain user	M
		Rewards	
FR12	Vendor	Create a reward token template (only upload the token metadata, including image, to the IPFS network)	M
FR13	Vendor	Soft delete a reward token template	W
FR14	Vendor/Client	Fetch all the details of a reward token by its ID	M
FR15	Vendor/Client	Fetch the list of reward token templates given a list of IDs	S
FR16	Vendor/Client	Fetch the list of reward token templates given a certain reward smart contract ID	S
FR17	Vendor/Client	Fetch the list of reward token templates created by a certain Vendor	M
FR18	Vendor/Client	Fetch the list of reward token templates that a certain user has balance of	M
FR19	Vendor/Client	Fetch the balance of a reward token owned by a certain user	M
FR20	Vendor	Insert the details of a reward smart contract	S
FR21	Vendor	Soft delete a reward smart contract	W
FR22	Vendor/Client	Fetch the details of a reward smart contract by its ID	S

Table 4.1 continued from previous page

ID	Actor	Feature	Priority
FR23	Vendor/Client	Fetch the list of reward smart contracts given a list of IDs	S
FR24	Vendor/Client	Fetch the list of reward smart contracts inserted by a certain Vendor	S
		Programs	
FR25	Vendor	Create a new program	M
FR26	Vendor	Edit a program	M
FR27	Vendor	Soft delete a program	W
FR28	Vendor/Client	Fetch all the details of a program by its ID	M
FR29	Vendor/Client	Fetch the list of programs created by a certain Vendor	M
FR30	Vendor/Client	Fetch the list of programs a certain user has joined	M
FR31	Vendor/Client	Fetch the list of programs given a name	M
FR32	Vendor/Client	Fetch the list of programs given a location	S
FR33	Vendor/Client	Join/leave a certain program	M
		Transactions	
FR34	Vendor	Mint a new batch of a certain token to some user	M
FR35	Vendor/Client	Transfer balance of a certain token to some other user	M
FR36	Vendor/Client	Fetch the transaction history of a certain token	S
FR37	Vendor/Client	Redeem a reward	M
FR38	Vendor/Client	Send an email notification to a user when he was enough token balance to redeem a certain reward from a certain program	W
FR39	Vendor/Client	Send an email notification to a user when he was token balance about to expire	W

Table 4.1: Functional Requirements.

4.2 User Stories

To improve organization, User Stories (US) are grouped into five modules per actor. Each user story follows the format "as an [actor], I want to [goal] so that [reason]", providing a clear explanation of the feature.

4.2.1 Vendor

Authentication module

US1: Log in

As a vendor, I want to authenticate myself on the platform so that I can use all my functionalities. I can do that by using any of the following methods:

- Email and password credentials;
- MetaMask digital wallet.

US2: Registration

As a vendor who does not have a registered account on the platform, I want to create a new account so that I can use the functionalities. I can do that by:

- Contact the platform support to enter the vendor whitelist;
- Register with either email and password credentials, or my MetaMask digital wallet.

Tokens module

US3: Create a token template

As an authenticated vendor, I want to create a new token template so that it can be used in programs. And for that, I need to specify:

- The name of the token;
- The description of the token;
- The type of token (if it is Transactable between users, like points, or not, like subscription memberships);
- The image of the token.

US4: Soft delete a token

As an authenticated vendor, I want to hide my token so that it does not shown up when searching for it.

US5: Fetch all the details of a token

As an authenticated vendor, I want to fetch all the details, including my balance (slows down the request), of a certain token so that I get all the information about that token.

US6: Fetch a list of tokens

As an authenticated vendor, I want to fetch a list of tokens given a certain filter so that I can list them. I can filter the tokens by:

- An array of IDs (get all tokens in the given array);
- The creator of the token template (get all tokens created by a certain vendor);
- The client (get all tokens that a certain user has balance of).

Rewards module

US7: Create a reward token template

As an authenticated vendor, I want to create a new reward token template so that it can be used in programs. And for that, I need to specify:

- The ID of the reward smart contract (relates to the smart contract where the reward token is going to be minted);
- The name of the reward;
- The description of the reward;
- Extra information/attributes about the reward that needs to go to the token metadata when minted;
- The image of the reward.

US8: Soft delete a reward

As an authenticated vendor, I want to hide my reward so that it does not shown up when searching for it.

US9: Fetch all the details of a reward

As an authenticated vendor, I want to fetch all the details, including my balance (slows down the request), of a certain reward so that I get all the information about that reward.

US10: Fetch a list of rewards

As an authenticated vendor, I want to fetch a list of rewards given a certain filter so that I can list them. I can filter the rewards by:

- An array of IDs (get all rewards in the given array);
- An array of IDs of the reward smart contract (get all rewards whose smart contract ID is in the given array);
- The creator of the reward token template (get all rewards created by a certain vendor);
- The client (get all rewards that a certain user has balance of).

US11: Insert the details of a reward smart contract

As an authenticated vendor, I want to add the details of an already deployed smart contract so that the platform mints the rewards in the smart contract deployed by me. And for that, I need to specify:

- The address of the smart contract;
- The standard of the smart contract (ERC-1155 or ERC-721);
- The ABI of the smart contract.

US12: Soft delete a reward smart contract

As an authenticated vendor, I want to hide my reward smart contract so that no more reward templates can be associated to it.

US13: Fetch a list of reward smart contracts

As an authenticated vendor, I want to fetch a list of reward smart contracts given a certain filter so that I can list them. I can filter the contracts by:

- An array of IDs (get all contracts in the given array);
- The creator of the contract (get all contracts inserted by a certain vendor).

Programs module

US14: Create a new program

As an authenticated vendor, I want to create a new loyalty program so that clients can use it. And for that, I need to specify:

- The name of the program;
- The type of program (if it is a Points or Subscriptions type of program);
- The description of the program;
- The array of the IDs of the tokens supported by the program;
- The array of the IDs of the rewards with the respective the price supported by the program;
- The array of locations (coordinates and city names) of the vendor's physical stores;
- The status of the program (if the program is Active or Deactivated);
- The terms of service file of the program.

US15: Edit a program

As an authenticated vendor, I want to edit information about my loyalty programs so that they can stay up to date. I can edit:

- The description of the program;
- The array of the IDs of the tokens supported by the program;
- The array of the IDs of the rewards with the respective the price supported by the program;
- The array of locations (coordinates and city names) of the vendor's physical stores;
- The status of the program (if the program is Active or Deactivated);
- The terms of service file of the program.

US16: Soft delete a program

As an authenticated vendor, I want to hide my deactivated loyalty program so that it does not shown up when searching for it.

US17: Fetch all the details of a program

As an authenticated vendor, I want to fetch all the details, including the ones saved on the blockchain (slows down the request), of a certain loyalty program so that I get all the information about that program.

US18: Fetch a list of programs

As an authenticated vendor, I want to fetch a list of loyalty programs given a certain filter so that I can list them. I can filter the programs by:

- The vendor (get all programs created by a certain vendor);
- The client (get all programs a certain user has joined);
- The name of the program (get all programs that match a certain name);
- The location (get all programs that have a physical store inside a certain radius of a certain location).

US19: Join/leave a certain program

As an authenticated vendor, I want to join a certain loyalty program so that I get use it, or I want to leave a certain loyalty program because I don't want to use it anymore.

Transactions module

US20: Mint a new batch of a certain token to some user

As an authenticated vendor, I want to generate balance of a certain token into another user's account as a result of his loyalty so that he can use it. Additionally, I can set an expiration date for the generated balance or leave it indefinite.

US21: Transfer balance of a certain token to some other user

As an authenticated vendor, I want to transfer some balance of a certain token from my account to another user so that he can use it.

US22: Fetch the transaction history of a certain token

As an authenticated vendor, I want to list all the transactions related to a certain token so that I can track them.

US23: Redeem a reward

As an authenticated vendor, I want to use my balance of one or more tokens to redeem a reward so that I use it.

US24: Send email notifications

As an authenticated vendor, I want to receive email notifications either when I have enough token balance to redeem a reward in any program I've joined, or if I have token balance about to expire, so that I'm aware of it.

4.2.2 Client

Authentication module

US25: Log in

As a client, I want to authenticate myself on the platform so that I can use all my functionalities. I can do that by using any of the following methods:

- Email and password credentials;
- MetaMask digital wallet.

US26: Registration

As a client who does not have a registered account on the platform, I want to create a new account so that I can use the functionalities. I can do that by using any of the following methods:

- Email and password credentials;
- MetaMask digital wallet.

Tokens module

US27: Fetch all the details of a token

As an authenticated client, I want to fetch all the details, including my balance (slows down the request), of a certain token so that I get all the information about that token.

US28: Fetch a list of tokens

As an authenticated client, I want to fetch a list of tokens given a certain filter so that I can list them. I can filter the tokens by:

- An array of IDs (get all tokens in the given array);
- The creator of the token template (get all tokens created by a certain vendor);
- The client (get all tokens that a certain user has balance of).

Rewards module

US29: Fetch all the details of a reward

As an authenticated client, I want to fetch all the details, including my balance (slows down the request), of a certain reward so that I get all the information about that reward.

US30: Fetch a list of rewards

As an authenticated client, I want to fetch a list of rewards given a certain filter so that I can list them. I can filter the rewards by:

- An array of IDs (get all rewards in the given array);
- An array of IDs of the reward smart contract (get all rewards whose smart contract ID is in the given array);
- The creator of the reward token template (get all rewards created by a certain vendor);
- The client (get all rewards that a certain user has balance of).

US31: Fetch a list of reward smart contracts

As an authenticated client, I want to fetch a list of reward smart contracts given a certain filter so that I can list them. I can filter the contracts by:

- An array of IDs (get all contracts in the given array);
- The creator of the contract (get all contracts inserted by a certain vendor).

Programs module

US32: Fetch all the details of a program

As an authenticated client, I want to fetch all the details, including the ones saved on the blockchain (slows down the request), of a certain loyalty program so that I get all the information about that program.

US33: Fetch a list of programs

As an authenticated client, I want to fetch a list of loyalty programs given a certain filter so that I can list them. I can filter the programs by:

- The vendor (get all programs created by a certain vendor);
- The client (get all programs a certain user has joined);
- The name of the program (get all programs that match a certain name);
- The location (get all programs that have a physical store inside a certain radius of a certain location).

US34: Join/leave a certain program

As an authenticated client, I want to join a certain loyalty program so that I get use it, or I want to leave a certain loyalty program because I don't want to use it anymore.

Transactions module

US35: Transfer balance of a certain token to some other user

As an authenticated client, I want to transfer some balance of a certain token from my account to another user so that he can use it.

US36: Fetch the transaction history of a certain token

As an authenticated client, I want to list all the transactions related to a certain token so that I can track them.

US37: Redeem a reward

As an authenticated client, I want to use my balance of one or more tokens to redeem a reward so that I use it.

US38: Send email notifications

As an authenticated client, I want to receive email notifications either when I have enough token balance to redeem a reward in any program I've joined, or if I have token balance about to expire, so that I'm aware of it.

4.3 Non-functional Requirements

Non-functional requirements, also referred to as quality attributes, represent the characteristics associated with the final product. Similar to functional requirements, each attribute is assigned a priority label based on the MoSCoW scale [71]. To describe and evaluate these attributes, scenarios are formulated, and the expected correct responses serve as evidence of the system possessing the desired characteristics. The quality attributes for this platform are outlined in Table 4.2.

ID	Quality Attribute	Priority
NFR1	Security	M
NFR2	Throughput	M
NFR3	Fault Tolerance	S

Table 4.2: Non-functional Requirements.

RNF1 - Security

In any modern software system, ensuring some sort of security is super important to protect sensitive information, prevent unauthorized access, and make sure that users have the appropriate privileges to perform specific actions. The scenarios in Tables 4.3 and 4.4 relate to this requirement.

Source of Stimulus	Unauthenticated user
Stimulus	Unauthenticated user attempts to access restricted resources or perform actions he is not authorized for
Artifact	Application Programming Interface (API)
Environment	Normal conditions
Response	The server responds with an 401 status error message
Response Measure	The server should not allow unauthenticated users to access restricted resources or perform actions that they are not authorized for

Table 4.3: Scenario 1: Unauthenticated access - Security (NFR1).

Source of Stimulus	Authenticated user
Stimulus	Authenticated user attempts to access restricted resources or perform actions he does not have permission for
Artifact	API
Environment	Normal conditions
Response	The server responds with an 401 status error message
Response Measure	The server should not allow authenticated users to access restricted resources or perform actions that they do not have permission for

Table 4.4: Scenario 2: Unauthorized access - Security (NFR1).

RNF2 - Throughput

When designing a platform that aims to centralize numerous loyalty programs and accommodate a substantial user base, it becomes crucial to address the non-functional requirement of throughput. Throughput refers to the system's capacity to handle a high volume of requests within a given time frame. Given the potential exposure of this platform to the real world, it is anticipated that a massive influx of requests will be directed towards it. Therefore, ensuring sufficient throughput is essential to maintain the platform's performance and responsiveness, guaranteeing a seamless user experience. The scenarios in Tables 4.5 and 4.6 relate to this requirement.

Source of Stimulus	Authenticated user
Stimulus	Authenticated user sends a request that communicates with the Blockchain
Artifact	API
Environment	Overloaded
Response	The server sends a response
Response Measure	The server should not take more than 15 seconds to send a response

Table 4.5: Scenario 3: Requests with access to Blockchain - Throughput (NFR2).

Source of Stimulus	Authenticated user
Stimulus	Authenticated user sends a request that does not communicate with the Blockchain
Artifact	API
Environment	Overloaded
Response	The server sends a response
Response Measure	The server should not take more than 5 seconds to send a response

Table 4.6: Scenario 4: Requests without access to Blockchain - Throughput (NFR2).

RNF3 - Fault Tolerance

Fault tolerance refers to the system's ability to remain operational and provide its intended services even in the presence of faults or failures. Considering the potential exposure of this platform to the real world, where it will encounter various challenges and potential points of failure, ensuring robust fault tolerance mechanisms is crucial to maintain uninterrupted service and prevent disruption to loyalty program operations. The scenarios in Tables 4.7, 4.8 and 4.9 relate to this requirement.

Source of Stimulus	Authenticated user
Stimulus	Authenticated user sends a request with invalid inputs
Artifact	API
Environment	Normal conditions
Response	The server responds with an 422 status error message with information about the invalid inputs
Response Measure	The server should catch and handle the exception and give feedback about the error

Table 4.7: Scenario 5: Exception Handling - Fault Tolerance (NFR3).

Source of Stimulus	Authenticated user
Stimulus	Authenticated user sends a request that accesses the Blockchain or the IPFS
Artifact	API
Environment	The platform lost the connection to the Blockchain or to the IPFS
Response	The server responds with an 424 status error message with the respective information
Response Measure	The server should catch and handle the exception and give feedback about the error

Table 4.8: Scenario 6: Blockchain or IPFS Failure - Fault Tolerance (NFR3).

Source of Stimulus	Authenticated user
Stimulus	Authenticated user sends a request that accesses the database
Artifact	API
Environment	The platform lost the connection to the database
Response	The server responds with an 500 status error message and switches to redundant or backup database
Response Measure	The server should detect the lost connection and initiate the recovery process of switching to a redundant or backup database and keeping down time minimal

Table 4.9: Scenario 7: Database Failure - Fault Tolerance (NFR3).

4.4 Success Criteria

For a project to be described as successful, there are some criteria that need to be achieved before the final delivery. In this case, the intern must meet the following criteria:

- The project must be finished on time, with all "Must Have" requirements met (both functional and non-functional);
- Every architecture and/or requirement decision must be approved by the product owner;
- The documentation produced with the project is such that it will also be helpful in the future.

Chapter 5

System Design

This chapter provides an overview of the system design for the platform. While detailed information is available in Chapter 7, the present chapter offers a high-level understanding of system architecture, its components, and the users who interact with it. Additionally, it discusses the technologies chosen for each component and presents a risk analysis.

5.1 System Architecture

A high-level view of the project's architecture is shown in Figure 5.1 (the frontend is out of the project's scope). This enables us to comprehend how each element is related to the others and how each serves a specific purpose.

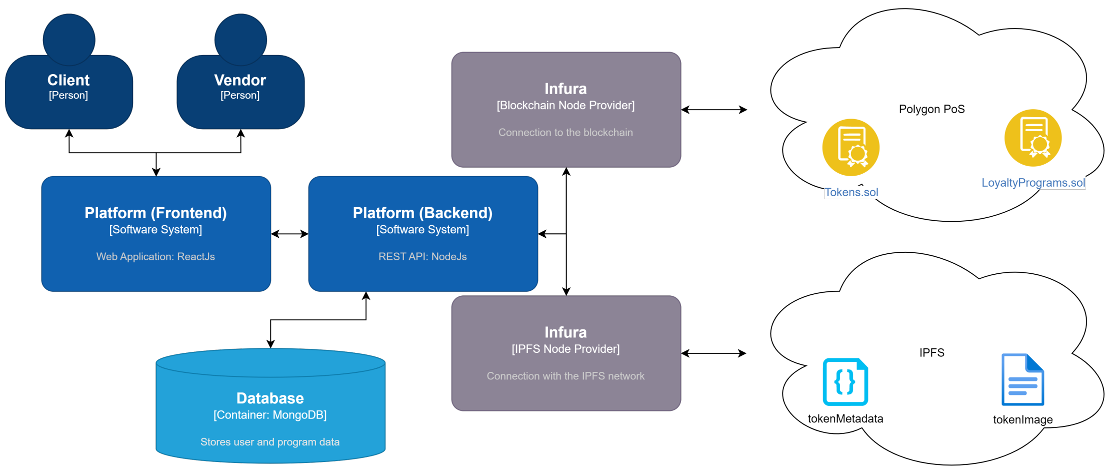


Figure 5.1: System architecture.

The developed prototype acts as an interface between the user and the blockchain, offering the functionalities explained in the requirements chapter (Chapter 4). To accomplish this, it uses a database to store and retrieve data, a connection to the blockchain network to interact with it, and integration with the IPFS network to facilitate the storage and retrieval of NFT metadata.

5.2 Technologies

Blockchain

Section 2.2.5 introduces a few blockchain platforms, but most of them are not public, and the project would benefit from having a public blockchain. In light of this, Table 5.1 consolidates the public blockchains that offer adequate NFT and smart contract development support. It also includes information on their maximum TPS capacity and average cost per transaction.

Blockchain Platform	Programming Language	Maximum TPS	Average Cost per Transaction (USD)
Ethereum	Solidity	20	$2.02
Tezos	Michelson	40	-
Polygon	Solidity	65,000	$0.003999
Cosmos	Ethermint	10,000	-

Table 5.1: Comparison between different blockchain platforms [1].

When selecting a public blockchain for our project we prioritized characteristics such as high TPS, strong community support, interoperability, and low transaction costs to ensure maximum transparency. Upon reviewing Table 5.1, Polygon was the chosen blockchain technology as it aligns best with these requirements.

Blockchain and IPFS networks access

There are two ways to connect to the blockchain and IPFS networks: (i) either we host a blockchain and an IPFS nodes, which entails requiring the resources to maintain the nodes constantly running, or (ii) we use an external node provider who enables us to communicate with the networks via their API. So we chose option two and compiled a comparison table (Table 5.2) featuring various node providers that support connection to the Polygon POS network. This allows us to evaluate and select the service that best aligns with the requirements of the project.

Observing Table 5.2, it becomes evident that certain node providers present their specifications in terms of requests, while others utilize Compute Units (CU). A CU represents the basic unit of processing power and is used to quantify the computational intensity of a request. For instance, a simple request such as "block-

[1]Values verified in December 2022 [72, 73]. Empty fields denote missing or outdated information.

Characteristics	Node Providers					
	Infura [74]	Moralis [75]	Alchemy [76]	Chainstack [77]	GetBlock [78]	QuickNode [79]
Plan (CU/month)	3.000.000 requests	10.000.000	300.000.000	3.000.000	3.000.000 requests	10.000.000
Throughput (CU/s)	-	25	330	-	60 requests	25 requests
IPFS Support	✓	✓	✗	✓ (Early Access)	✗	✗

Table 5.2: Comparison between node providers' services.

Number" consumes 10 CU, whereas a more complex "eth_call" request utilizes 26 CU [80].

Alchemy, GetBlock, QuickNode, and Chainstack were disqualified as potential options due to their lack of, or early access, integration with the IPFS network, and was excluded because its IPFS API is still in early access. As a result, Moralis and Infura emerged as the only remaining choices. Infura was selected over Moralis due to its utilization of both the web3.js [81] (blockchain connection) and the ipfs-http-client [82] (IPFS connection) libraries, both open-sourced solutions that offer significant advantages in terms of flexibility and migration options, while Moralis only support their own libraries. By leveraging web3.js and ipfs-http-client, we can seamlessly transition from one node provider to another, if necessary, or even transition from a node provider to hosting a local node. This freedom of choice enables us to adapt and optimize our infrastructure based on evolving needs, avoiding vendor lock-in and ensuring a more scalable and adaptable solution for the future.

Prototype Blockchain Loyalty Platform

After compiling information about node providers, we have decided to build the platform using Node.js [83] instead of Python [84]. Node.js benefits from a larger developer community, which increases the likelihood of finding online solutions in case any issues arise during the development process.

Loyalty Program's Database

A non-relational database, such as MongoDB [85], is a suitable choice for storing the specific details of each loyalty program. This is because the structure of each loyalty program can vary, and a non-relational database offers the advantage of faster performance compared to a relational database. The decision to use MongoDB was made based on the preference of the author for this technology.

5.3 Risks

By identifying the risks associated with the project, mitigation plans can be created to lessen their effects in the event that the risk materializes in the middle of the development phase. Each risk is identified by an ID, has a description, and a mitigation plan, and is quantified by its probability and impact. The risks identified are:

R1: Technical Dependencies

Description	This platform relies on third-party services for the communication with the Blockchain. This external component introduces the risk of changes in API specifications or on the conditions of the free plan used to have access to the services.
Probability	Low
Impact	High
Mitigation Plan	An alternative solution that offers the same services can be used.

R2: Change of Requirements

Description	The utilization of an agile development methodology introduces the inherent risk of requirements changing as a result of client feedback. In the event of requirement changes, the development phase can become longer.
Probability	Medium
Impact	Medium
Mitigation Plan	Treat each change as a new feature during meetings and assign it a new priority status. This approach prevents the development phase from stagnating and ensures that evolving requirements are appropriately acknowledged and prioritized.

R3: Inexperience with Technologies

Description	The technologies this project works with are new to the intern and the lack of knowledge can lead to delays, suboptimal solutions, and potential errors in the system.
Probability	High
Impact	High
Mitigation Plan	Adequate training, mentorship, and research can equip the intern with the necessary expertise he needs.

As soon as the risks have been characterized, we can display them in the risk matrix shown in Figure 5.2.

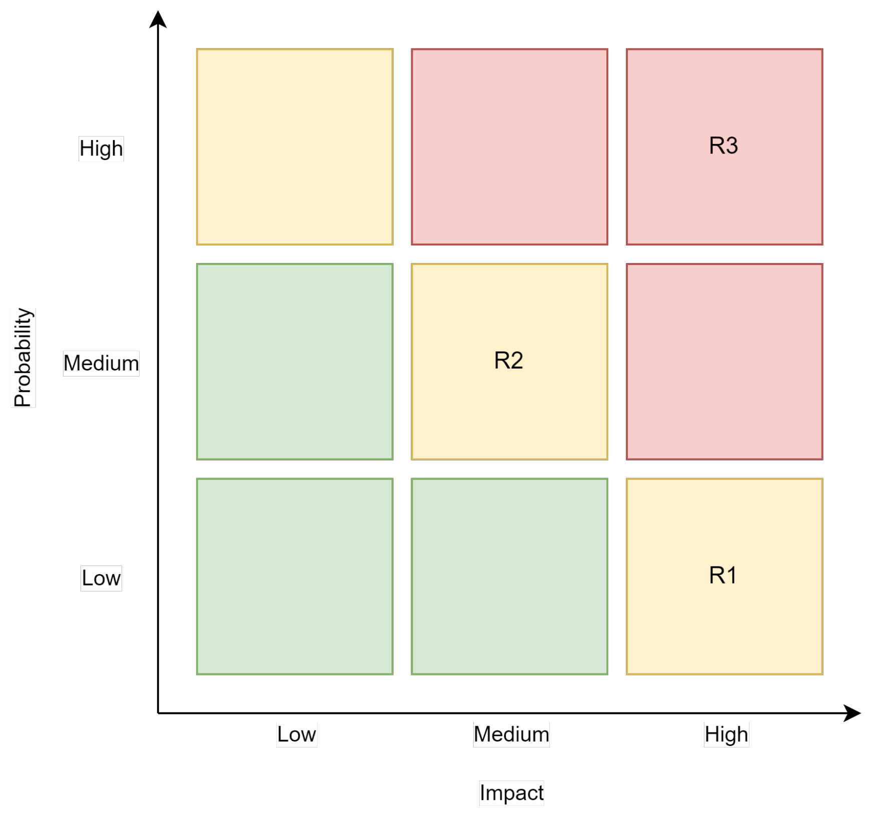


Figure 5.2: Risk Matrix.

Chapter 6

Development Methodology and Planning

This chapter discusses the topics of development methodology and planning of the project. The first section provides details about the methodology employed during the development phase. The second section presents the planning for both the first and second semesters, along with a discussion on the disparities between planned and actual plans.

6.1 Development Methodology

Project management is essential for steering the project in the right direction and setting the right pace during the development phase by establishing goals and timelines. Therefore, deciding on the most effective methodology is crucial.

Scrum is the most widely used methodology at WIT Software, and it was selected for the internship as it fits it better. It is regarded as a type of agile methodology that is iterative and incremental and seeks to deliver functionalities frequently to receive ongoing customer feedback, thereby minimizing bugs and errors. Each iteration is called a Sprint and, typically, has a duration of 2 or 3 weeks. During each Sprint, the development team works through the tasks on the Sprint Backlog. And in every other Sprint, the Product Owner transfers tasks from the Product Backlog to the Sprint Backlog, updating it with new challenges and objectives.

Scrum Roles

The Scrum Methodology consists of 3 actors: (i) the Product Owner, (ii) the Scrum Master, and (iii) the Development Team.

The Product Owner has the most authority and represents the client. Its principal concern is to guarantee that the requirements are matched and achieved, and he keeps the Product Backlog updated.

The Scrum Master leads the whole team and is responsible for scheduling daily meetings and promoting good team interaction and productivity.

The Development Team is the one who develops the project, works through the Sprint Backlog and tests functionalities.

Scrum Artifacts

The Product Backlog is a list of all the features and requirements that have to be present in the final product. The Sprint, as mentioned before, is an interval of time, typically of 2 or 3 weeks, in which a smaller list of tasks designated Spring Backlog have to be done. The Daily Scrum is a small meeting for the whole team to discuss any doubts and challenges that may be facing, and the progress accomplished. It is worth noting that these meetings can actually be non-daily, depending on the project set-up.

To best align scrum methodology with the development phase, we established a 2-week interval for the Sprints, and the daily scrum meetings took place on Mondays, Wednesdays, and Fridays. The intern actively participated in monthly presentations within WIT, showcasing his progress and receiving valuable feedback from other advisors. Furthermore, we scheduled bi-monthly meetings with the advisory team, with the frequency increased to weekly during the month of June. This approach ensured regular progress updates, timely feedback, and effective collaboration throughout the project's lifecycle.

6.2 Planning

This work spanned two semesters, and for each, a plan with a list of suggested tasks was developed. A comparison of their estimated and real completion times is also presented.

6.2.1 First Semester

Here are the tasks assigned to the first semester, along with their estimated and actual durations, as shown in Figure 6.1 and Figure 6.2, respectively:

- Research - Build the state of the art of both blockchain-based and traditional loyalty programs, and study the different types of blockchain and their applications in loyalty programs;
- Requirements - Create a detailed requirements list and corresponding user stories;
- Solution Proposal - Propose a suitable solution, including its architecture and development plan;

- Intermediate Report - Compile all the previous tasks into an intermediate report.

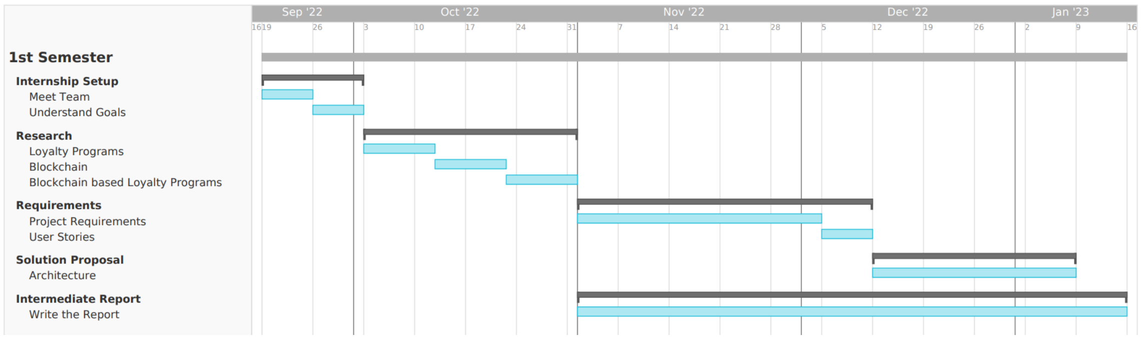


Figure 6.1: Gantt chart with the estimated planning for the first semester.

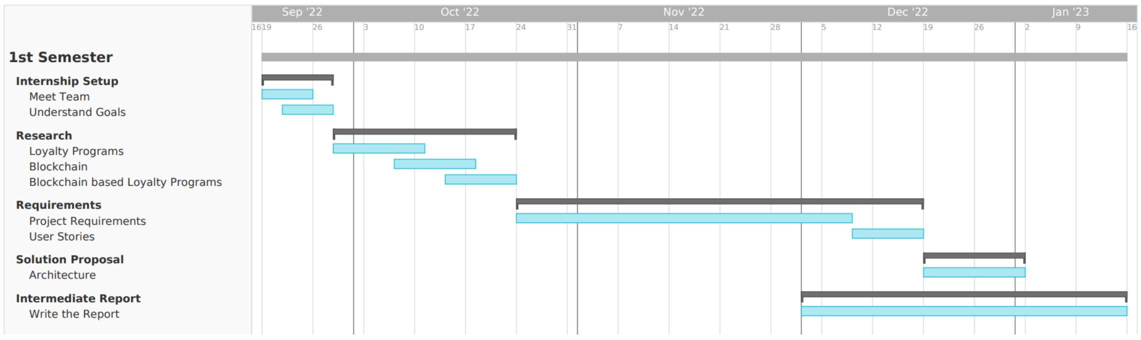


Figure 6.2: Gantt chart with the real timeline of the first semester.

Upon reviewing the timelines, it is evident that the actual progress did not deviate significantly from the initially estimate. The most noticeable discrepancy lies in the writing of the intermediate report, which started approximately one month later than originally anticipated.

6.2.2 Second Semester

Here are the tasks assigned to the second semester, along with their estimated and actual durations, as shown in Figure 6.3 and Figure 6.4, respectively:

- Project Setup - Elaborate the development plan, and learn how to interact with the new technologies;
- Development - Implement and build the product;
- Testing - Conduct testing to ensure functionality and quality;
- Final Report - Prepare the final report documenting the project;
- Scientific Paper - Write an article for a conference on blockchain and loyalty programs.

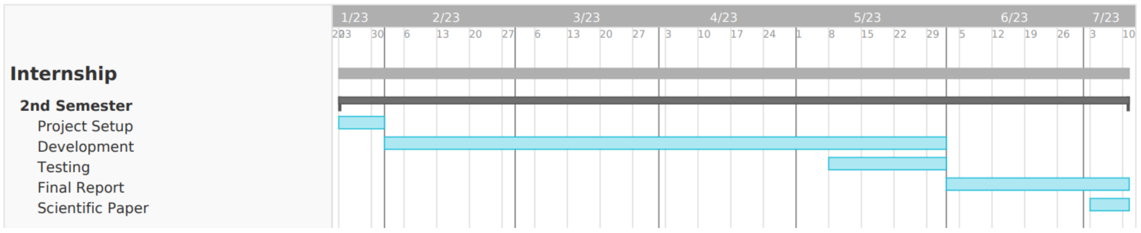


Figure 6.3: Gantt chart with the estimated planning for the second semester.

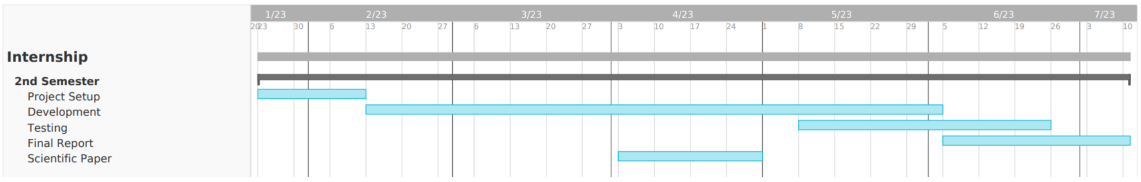


Figure 6.4: Gantt chart with the real timeline of the second semester.

Upon comparing both charts, it is evident that there are minor extensions in the durations of the Project Setup, Testing, and Scientific Paper phases. The longer duration of the Project Setup phase can be attributed to the initial implementation of the Moralis service as the node provider (refer to Section 5.2 for more details on the "Blockchain and IPFS networks access" topic), which was subsequently replaced with the Infura service. The prolonged testing phase allowed for the identification of bugs and the establishment of a more robust testing environment to assess throughput. As for the Scientific Paper, it was completed and submitted for the ISD 2023 conference on May 1, 2023. Although it was not accepted, the feedback received mainly highlighted the need for real-world data to substantiate the theoretical claims made by the platform.

Chapter 7

Development

This chapter focuses on the development phase of the project, providing insights into the development environment and the tools utilized. Next, it dives into the smart contract development, highlighting its significance. The chapter also covers the database structure and the relationships between various collections. An overview of the backend is provided, including detailed information about the controllers. Lastly, a preview of the frontend, which was developed exclusively for demonstration purposes, is presented.

7.1 Development Environment

At the start of the second semester, the project entered the setup phase, which involved configuring and establishing various tools on the computer provided by WIT. Microsoft Visual Studio Code [86] was selected as the primary code development platform due to its extensive features and functionalities. For smart contract development and deployment, the Remix Online IDE [87] was utilized through the Google [88] browser, complemented by the MetaMask [89] extension for establishing connectivity to the blockchain and enabling digital wallet functionalities. The MongoDB [85] database was installed as a service, remaining operational whenever the computer was powered on. Additionally, the MongoDB Compass [90] app provided an intuitive interface for accessing and managing stored data. To ensure code version control and backup, a private repository was created on GitHub [91]. The sprint planning and task allocation was handled in a Jira Software [92] project. Lastly, the developed frontend web app, designed to showcase and demonstrate the platform, had its mockups created using Figma [93].

7.2 Smart Contracts

During the development phase, the Polygon Mumbai network was chosen as a replacement for the Polygon POS network due to its role as a testing network.

Testnets are commonly used by blockchain developers to build and test their applications, as they offer free utilization and a controlled environment. As previously mentioned, the Remix Online IDE was utilized for smart contract development and deployment. Prior to that, a pair of public and private keys was generated within the MetaMask extension. These keys were used to establish the account for the platform. To ensure the availability of funds for transactions and interactions with the smart contracts on the Polygon Mumbai network, the account was then recharged with MATIC balance. This balance was obtained from the Polygon Faucet [94], a tool designed to provide developers with test tokens to facilitate their development and testing processes.

Loyalty Programs

In order to uphold the objective of promoting transparency in the realm of loyalty programs, it was decided that the platform should also store program-related data on the blockchain. However, ensure cost-effectiveness, only critical information would be stored in this manner. The essential details include the program's ID, the public address of the vendor responsible for creating the program, the list of supported tokens, the available rewards (including their redemption prices), and the program status (as illustrated in Figure 7.1). By limiting the data stored on the blockchain to these essential elements, the platform can achieve its transparency goals while minimizing the storage costs associated with blockchain transactions.

```
contract LoyaltyPrograms {
    struct Reward {
        uint256 id;
        uint256 price;
    }
    struct Program {
        address creator;
        uint256[] supportedTokens;
        Reward[] rewardsCatalog;
        bool isActive;
    }
    mapping(uint256 => Program)
        private programTable;

    ...
}
```

Figure 7.1: *LoyaltyPrograms.sol* code snippet.

Tokens

As detailed in Section 3.1, each token within the platform consists of multiple batches, each with a distinct expiration date and information about its ability to being transferred between clients. In this context, each batch serves as an individual token on the blockchain, whereby the batch ID corresponds to the token ID. Considering this structure, the ERC-1155 token standard is deemed more suitable for the platform's requirements compared to the ERC-20 and ERC-721 standards, as depicted in Figure 7.2. The ERC-1155 standard allows for the representation of multiple fungible and non-fungible tokens within a single contract.

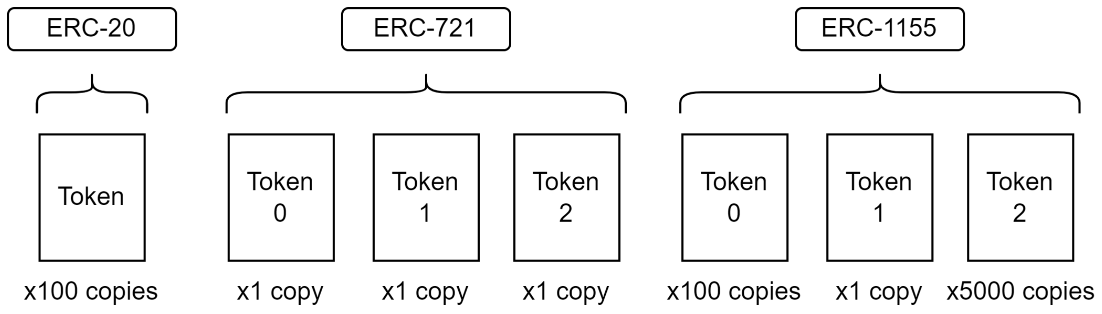


Figure 7.2: Comparison between ERC-20, ERC-721, and ERC-1155 token standards.

Rewards

The platform offers the flexibility for vendors to deploy their own smart contracts to store their rewards, which are represented as blockchain tokens. This approach opens up numerous possibilities for integrations with other projects, such as an ongoing project at WIT, where these tokens can be utilized as coupons during the checkout process for purchases made from a Shopify [95] store.

To facilitate this process, vendors are required to design their smart contract using a customized ERC-721 or ERC-1155 template provided by the platform. Additionally, they must assign the platform with the "minter role" to enable it to mint the reward tokens. For vendors with limited understanding of blockchain technology, the platform simplifies the process by deploying one of these templates and allowing every vendor to use it. This way, vendors can skip the process of deploying a smart contract at the cost of unlimited customization capabilities, such as integration with other platforms.

7.3 Database

The database serves as a storage solution for storing data fields that are considered not suitable for blockchain storage. Two primary rules help determine whether a field is appropriate for blockchain storage: (i) whether the changes and updates to the field should be trackable, and (ii) whether the field will be frequently searched or filtered. For instance, it is sensible to store the list of tokens supported by a program on the blockchain as it allows for easy tracking and verification. On the other hand, storing the name of the program or the terms of service file on the blockchain may not be necessary as they do not require tracking and are frequently accessed or retrieved.

Figure 7.3 illustrates the relationship between the various entities, followed by an explanation of the purpose of each collection.

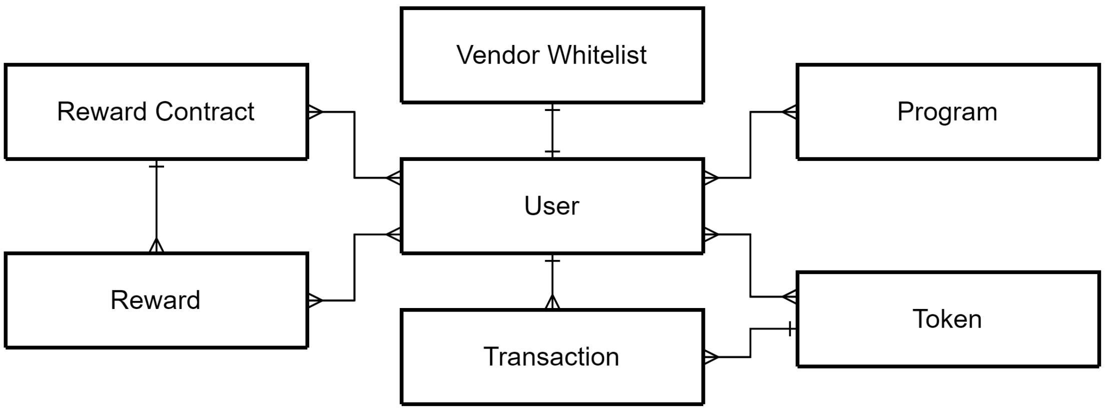


Figure 7.3: Representation of the relationship between collections.

Users

Figure 7.4 illustrates the user segment of the database, which consists of two collections. The first collection, named "user," stores user-related data such as login credentials, digital wallet details, role (client or vendor), the list of joined programs, and the balance of tokens and rewards they possess. Additionally, for vendors, the collection also includes their API key. The second collection, called "vendor whitelist," is responsible for storing the public addresses or emails of users who will be granted the vendor role upon registering on the platform.

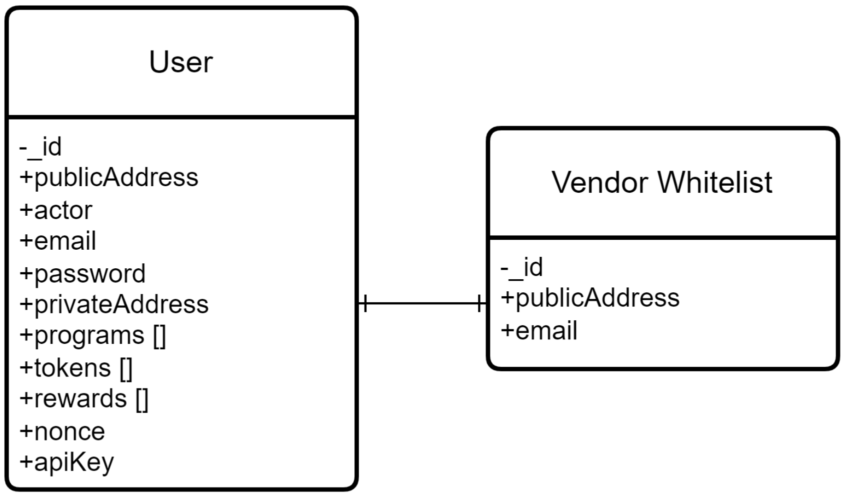


Figure 7.4: Representation of the *User* and *Vendor Whitelist* collections, and their relationship.

Loyalty Programs

The "programs" collection is responsible for storing data related to loyalty programs that are not stored on the blockchain. The fields stored in this collection are specifically chosen as they do not require tracking changes and updates or are frequently accessed or retrieved in queries. Figure 7.5 provides a visual representation of the stored fields, which include the name of the program, description, type (points or subscriptions), terms of service file, public address of the user who created the program, and an array of locations containing the geographical coordinates of physical stores supporting the program.

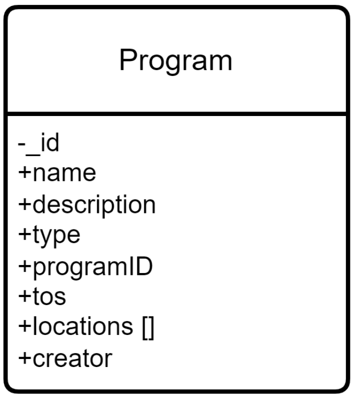


Figure 7.5: Representation of the *Programs* collection.

MongoDB offers a notable feature, which is the capability to index geographical coordinates. This feature allows for efficient retrieval of programs that have physical stores located within a specified distance from a given point. For instance, if a user is at a shopping mall, he can fetch all the loyalty programs of the stores in the area of the mall by conducting a search based on his location and a designated radius. This capability allows for streamlined access to loyalty programs specifically tailored to the user's current location, enhancing their browsing experience and convenience.

Tokens

The "tokens" collection holds information related to tokens. It includes the token template, as depicted in Figure 7.6. The token template contains fields such as name, description, transferability between clients, public address of the vendor who created it, image, and the smart contract address where it resides. Additionally, it stores a batch ID that does not have an expiration date, as well as a list of all remaining batch IDs associated with the token.

By storing the token template in the database, vendors have the ability to "create a token" without actually minting anything on the blockchain. This concept is referred to as the "token template" since it serves as a blueprint for creating tokens.

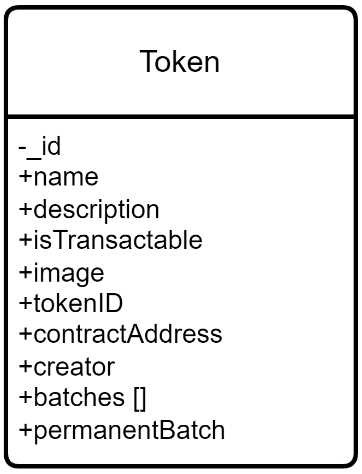


Figure 7.6: Representation of the *Tokens* collection.

Rewards

The rewards section encompasses two collections: "rewards" and "reward contracts". In the "rewards" collection, as depicted in Figure 7.7, reward templates are stored. Each reward template includes attributes such as the name of the reward, description, image, and other relevant customized attributes by the reward creator. The collection also contains information about the smart contract where the reward is deployed and the associated batch IDs.

The "reward contracts" collection specifically stores information related to the smart contracts associated with the previous rewards. It includes the smart contract address, the token standard followed by the contract, and the Application Binary Interface (ABI). The ABI describes the methods available in the smart contract, enabling the platform to interact with it effectively.

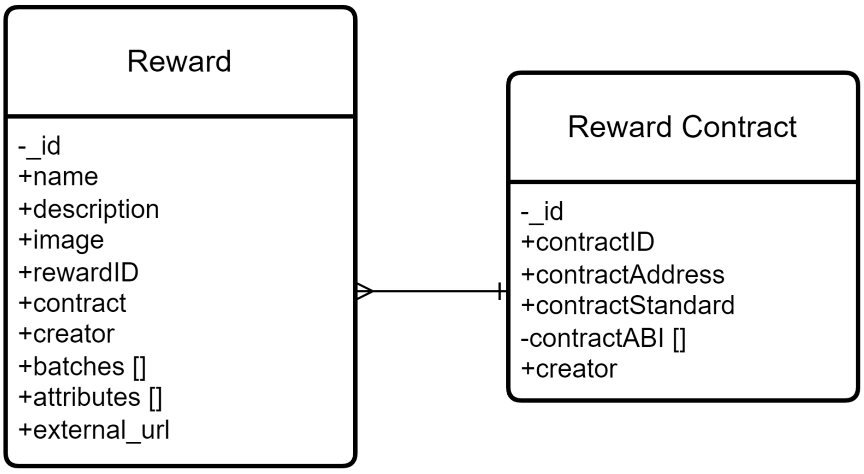


Figure 7.7: Representation of the *Reward* and *Reward Contract* collections, and their relationship.

Transactions

The "transactions" collection serves as a caching system for transactions related to tokens. Its main purpose is to facilitate faster retrieval of information regarding token minting, transferring, and redeeming. By caching previous transactions, the platform can focus solely on searching for new transactions, starting from the largest block number saved up to the current block number. This approach eliminates the need to search the entire blockchain, resulting in significant performance improvements. Figure 7.8 provides an overview of the stored fields, which include the block number where the transaction is saved, the addresses of the sender and receiver, the token sent, and notably, the batch and amount associated with each transaction.

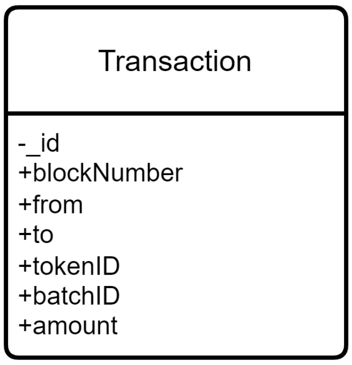


Figure 7.8: Representation of the *Transaction* collection.

7.4 Platform Backend

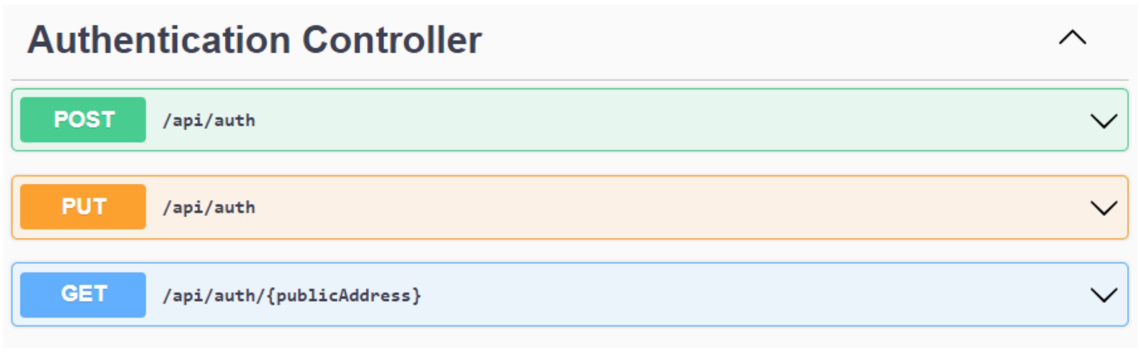
The backend service is constructed using the Express module from Node.js, which enables the creation of a RESTful API. The API consists of five controllers: Authentication, Program, Token, Reward, and Transaction controllers. These controllers facilitate the interaction between the frontend and backend by handling HTTP requests sent to specific endpoints.

To facilitate easier integration with the API, a Swagger UI [96] documentation is provided. This documentation describes all the endpoints available in the API, including the request and response formats, enabling developers to understand and interact with the API more efficiently.

Authentication

The Authentication controller (see Figure 7.9) is responsible for handling all registration and login requests to authenticate users accessing the backend service. Two authentication methods are supported: API key authentication for vendors and JSON Web Token (JWT) authentication for all users.

Vendors are provided with an API key upon registration, which they use to authenticate their requests. On the other hand, users can obtain a JWT through either an email and password login or a MetaMask login. The MetaMask login mechanism is based on the concept that a wallet consists of a pair of public and private keys. By signing a specific nonce with their private key, users can verify their ownership of the corresponding public key, thereby authenticating themselves.


Authentication Controller

POST /api/auth

PUT /api/auth

GET /api/auth/{publicAddress}

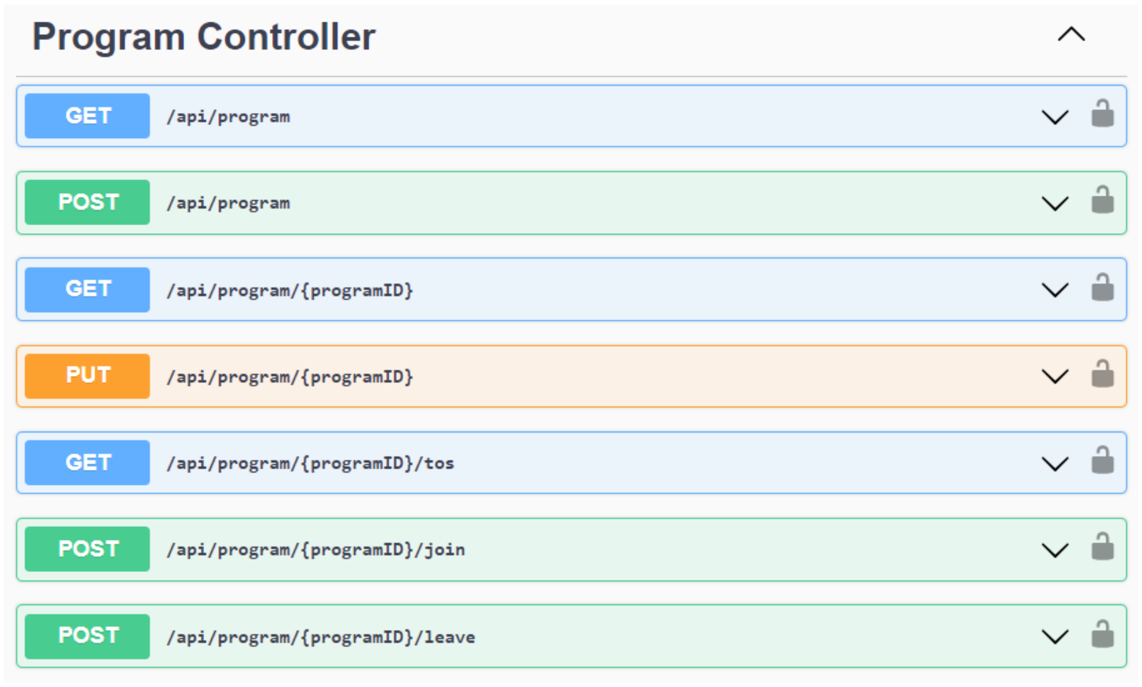
Figure 7.9: Authentication Controller.

Program

The Program controller (see Figure 7.10) handles various functionalities related to loyalty programs within the platform. It offers a range of features to enhance the user experience and facilitate program management for vendors.

Users can search and filter programs based on specific criteria to find programs that align with their preferences. Additionally, they can download the terms of service document associated with a program to review the program's terms and conditions. Users also have the ability to join or leave programs as per their preference. This allows them to participate in loyalty programs and, eventually, redeem rewards.

For vendors, the Program controller enables them to create and update programs. They can define essential program details such as the program name, description, supported tokens, rewards catalog, list of locations, and program type. They can also update program information as needed, ensuring the program stays relevant and up to date.



Program Controller

GET /api/program

POST /api/program

GET /api/program/{programID}

PUT /api/program/{programID}

GET /api/program/{programID}/tos

POST /api/program/{programID}/join

POST /api/program/{programID}/leave

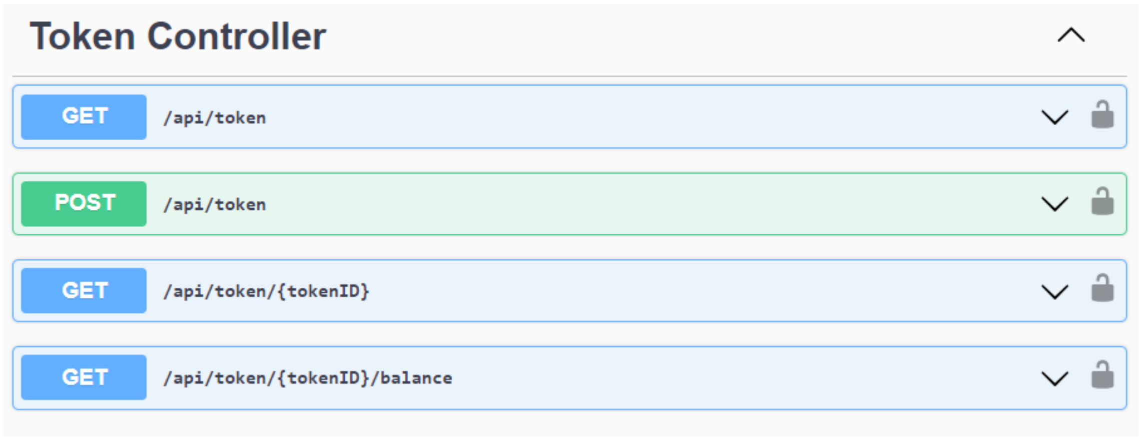
Figure 7.10: Program Controller.

Token

The Token controller (see Figure 7.11) is dedicated to handling various actions related to tokens within the platform, excluding the minting process.

Vendors are granted the capability to create new token templates. They can input specific details such as the token's name, description, image, and determine whether the token can be transferred between clients or not.

For all users, the Token controller enables them to search and filter tokens based on specific criteria. Additionally, they can verify the balance of a particular token for any user.



Token Controller

GET /api/token

POST /api/token

GET /api/token/{tokenID}

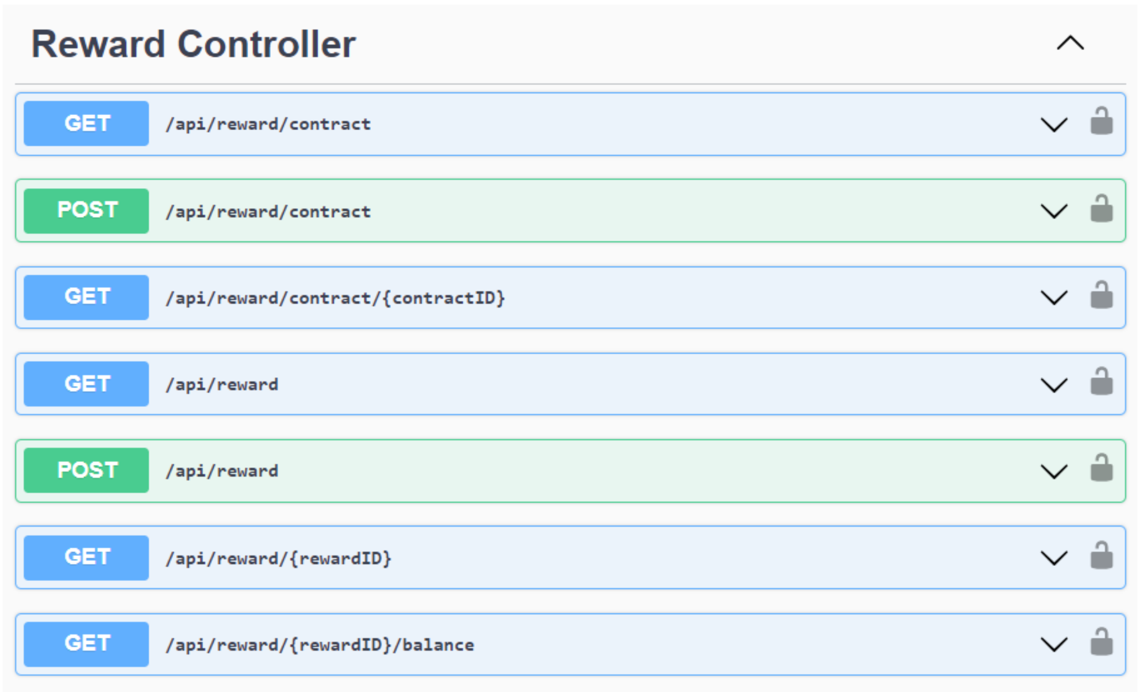
GET /api/token/{tokenID}/balance

Figure 7.11: Token Controller.

Reward

The Reward controller (see Figure 7.12) functions similarly to the Token controller, enabling vendors to create reward token templates. However, in the case of rewards, vendors are required to specify the smart contract where the rewards can be minted. Vendors have the option to use an existing deployed contract or insert a new one that they have deployed themselves. When creating a reward token template, vendors must provide the contract address, the token standard, and the ABI.

In addition to vendor functionalities, all users have access to features such as searching and filtering for contracts and rewards. Users can also verify the balance of a specific reward for a particular user, enabling them to track their accumulated rewards. This provides users with transparency and visibility into their reward holdings.


Reward Controller

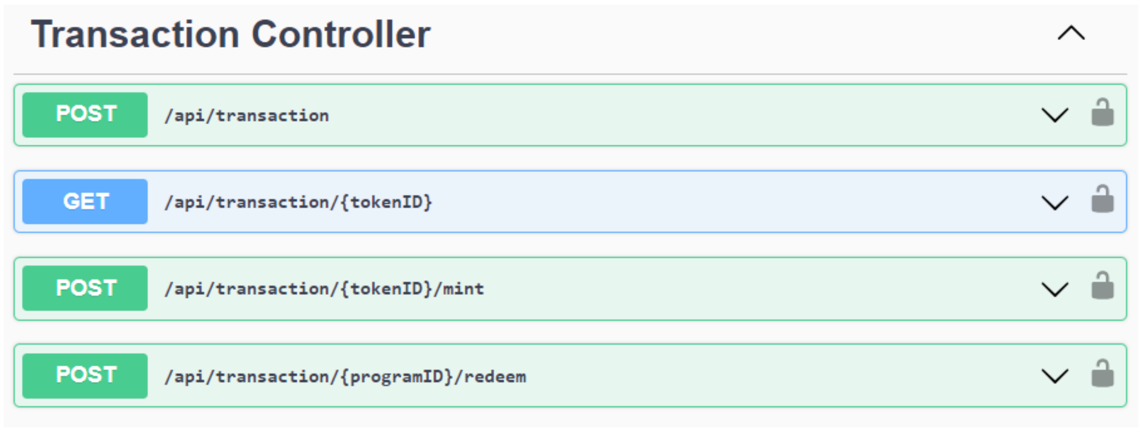
```
GET   /api/reward/contract
POST  /api/reward/contract
GET   /api/reward/contract/{contractID}
GET   /api/reward
POST  /api/reward
GET   /api/reward/{rewardID}
GET   /api/reward/{rewardID}/balance
```

Figure 7.12: Reward Controller.

Transaction

Finally, the Transaction controller (see Figure 7.13) holds all the functionalities related with minting or transferring tokens. One of the key features of the Transaction controller is the minting process, which allows users to receive tokens as they engage with a specific program. This enables them to accumulate a sufficient balance of tokens to redeem for rewards. The minting process also includes minting the corresponding reward when a user redeems their accumulated tokens.

Additionally, users have the ability to transfer tokens to other users, as long as the token allows for transfers, and the Transaction controller also allows for users to access the transaction history of a specific token.


Transaction Controller

POST /api/transaction

GET /api/transaction/{tokenID}

POST /api/transaction/{tokenID}/mint

POST /api/transaction/{programID}/redeem

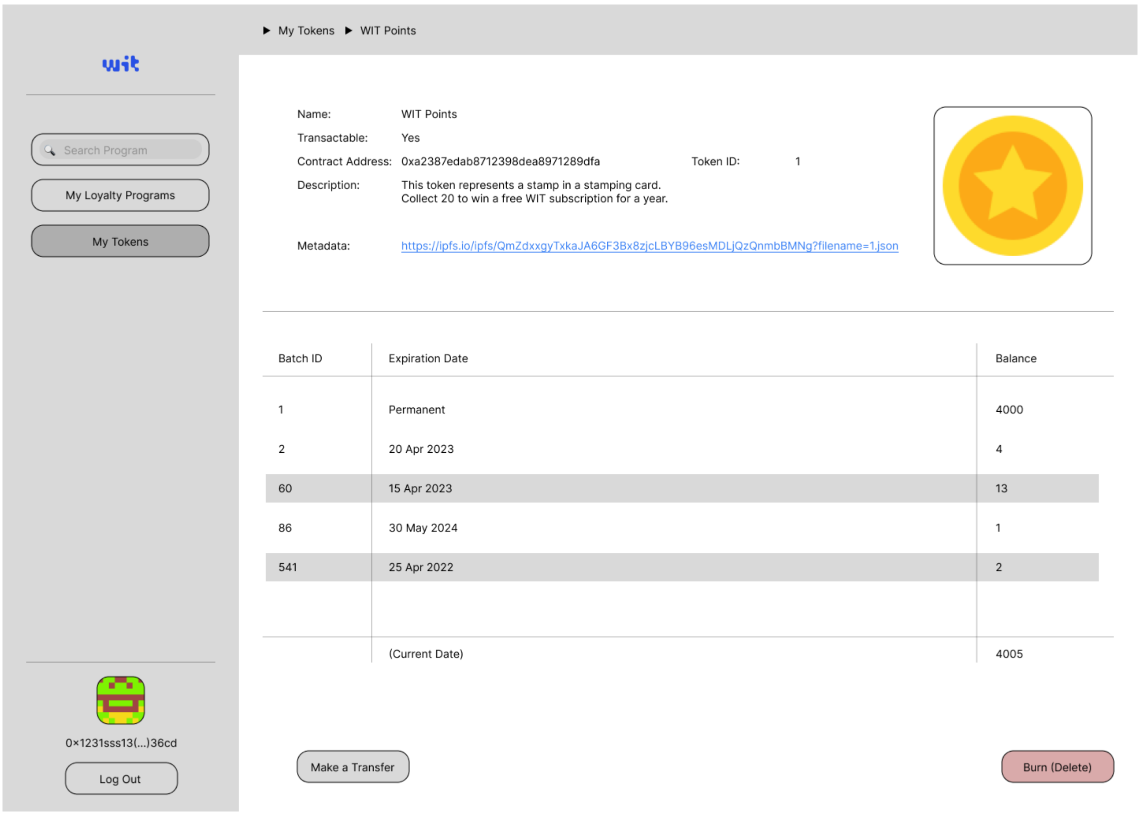
Figure 7.13: Transaction Controller.

7.5 Platform Frontend

The frontend of the platform was developed using React [97], a popular JavaScript library for building user interfaces. Its main purpose is to provide a visual representation of the platform's features and serve as a demonstration tool. To design the user interface of the frontend, mockups were created using Figma. These mockups outline the layout and visual elements of the platform's various pages and components. An example of a mockup is the token details page, as shown in Figure 7.14.

The frontend interacts with the backend through the provided RESTful API. It supports authentication methods such as MetaMask and email plus password logins. It is worth mentioning that the frontend does not directly access the blockchain. Instead, it relies on the backend API to interact with the blockchain on behalf of the user.

Additionally, it's important to note that the frontend primarily serves as a dashboard for the platform. It focuses on showcasing the platform's features. However, it doesn't demonstrate the functionality of minting tokens since this process is typically integrated into the vendor's platform. The vendor would initiate the token minting request after a customer makes a purchase.



Name:	WIT Points
Transactable:	Yes
Contract Address:	0xa2387edab8712398dea8971289dfa
Token ID:	1
Description:	This token represents a stamp in a stamping card. Collect 20 to win a free WIT subscription for a year.
Metadata:	https://ipfs.io/ipfs/QmZdxxgyTxkaJA6GF3Bx8zjcLBYB96esMDLjQzQnmbBMNg?filename=1.json

Batch ID	Expiration Date	Balance
1	Permanent	4000
2	20 Apr 2023	4
60	15 Apr 2023	13
86	30 May 2024	1
541	25 Apr 2022	2
	(Current Date)	4005

Figure 7.14: Mockup of token details page.

Chapter 8

Testing and Validation

This chapter focuses on testing and validating the platform's requirements. It begins by describing the testing environment and the specific tests conducted on the platform. Subsequently, the results of these tests are analyzed to determine whether the platform fulfills the initial requirements or not.

8.1 Testing

The testing phase is a crucial aspect of the project as it enables the evaluation of the platform. During testing, the platform was subjected to various scenarios and conditions to identify any existing bugs or issues. The primary objective was to eliminate as many bugs as possible, prioritizing critical and high-priority issues, while low-priority issues could be addressed at a later stage or deemed acceptable within the project timeline.

8.1.1 Testing Environment

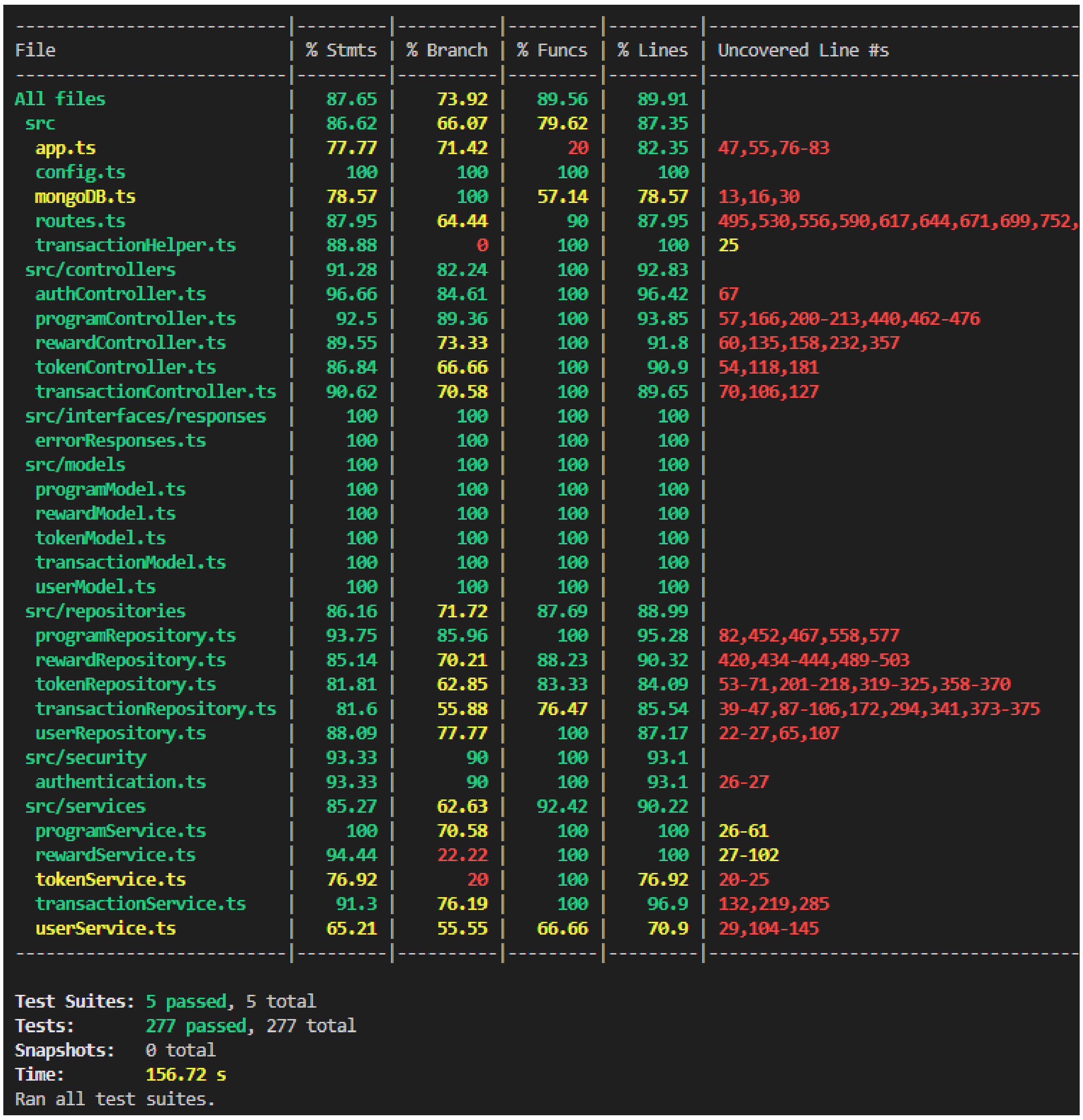
Two types of tests were conducted to assess the backend of the platform: unit testing and throughput testing. Unit tests were executed in the development environment to ensure the correctness of individual components and functions. On the other hand, throughput testing was carried out after compiling the source code. For the throughput testing, a dual virtual machine setup was employed. The backend and database were deployed on one virtual machine, while a separate virtual machine was dedicated to running the throughput testing script.

8.1.2 Unit Testing

Unit testing focuses on testing individual units, or the smallest testable components of a software system. The purpose of unit testing is to verify the correctness of these units in isolation, ensuring that they behave as expected and produce the desired outputs for a given set of inputs.

With the combination of Jest [98] and SuperTest [99] frameworks, the backend of the platform was tested by writing unit tests for each endpoint. These tests involved sending mock HTTP requests to the endpoints and asserting that the responses matched the expected outcomes. The unit tests covered different scenarios and error conditions to ensure that the backend functions correctly and handles various situations appropriately.

During the unit testing process, a high level of code coverage was achieved, with nearly 90% coverage as indicated by the Jest code coverage report presented in Figure 8.1. This level of code coverage demonstrates that the majority of the back-end code was exercised and tested during the unit testing phase, increasing confidence in its overall reliability and stability.



```
---------------------------|---------|----------|---------|---------|-------------------------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-------------------------------------
All files                  |   87.65 |    73.92 |   89.56 |   89.91 |
 src                       |   86.62 |    66.07 |   79.62 |   87.35 |
  app.ts                   |   77.77 |    71.42 |      20 |   82.35 | 47,55,76-83
  config.ts                |     100 |      100 |     100 |     100 |
  mongoDB.ts               |   78.57 |      100 |   57.14 |   78.57 | 13,16,30
  routes.ts                |   87.95 |    64.44 |      90 |   87.95 | 495,530,556,590,617,644,671,699,752,
  transactionHelper.ts     |   88.88 |        0 |     100 |     100 | 25
 src/controllers           |   91.28 |    82.24 |     100 |   92.83 |
  authController.ts        |   96.66 |    84.61 |     100 |   96.42 | 67
  programController.ts     |    92.5 |    89.36 |     100 |   93.85 | 57,166,200-213,440,462-476
  rewardController.ts      |   89.55 |    73.33 |     100 |    91.8 | 60,135,158,232,357
  tokenController.ts       |   86.84 |    66.66 |     100 |    90.9 | 54,118,181
  transactionController.ts |   90.62 |    70.58 |     100 |   89.65 | 70,106,127
 src/interfaces/responses  |     100 |      100 |     100 |     100 |
  errorResponses.ts        |     100 |      100 |     100 |     100 |
 src/models                |     100 |      100 |     100 |     100 |
  programModel.ts          |     100 |      100 |     100 |     100 |
  rewardModel.ts           |     100 |      100 |     100 |     100 |
  tokenModel.ts            |     100 |      100 |     100 |     100 |
  transactionModel.ts      |     100 |      100 |     100 |     100 |
  userModel.ts             |     100 |      100 |     100 |     100 |
 src/repositories          |   86.16 |    71.72 |   87.69 |   88.99 |
  programRepository.ts     |   93.75 |    85.96 |     100 |   95.28 | 82,452,467,558,577
  rewardRepository.ts      |   85.14 |    70.21 |   88.23 |   90.32 | 420,434-444,489-503
  tokenRepository.ts       |   81.81 |    62.85 |   83.33 |   84.09 | 53-71,201-218,319-325,358-370
  transactionRepository.ts |    81.6 |    55.88 |   76.47 |   85.54 | 39-47,87-106,172,294,341,373-375
  userRepository.ts        |   88.09 |    77.77 |     100 |   87.17 | 22-27,65,107
 src/security              |   93.33 |       90 |     100 |    93.1 |
  authentication.ts        |   93.33 |       90 |     100 |    93.1 | 26-27
 src/services              |   85.27 |    62.63 |   92.42 |   90.22 |
  programService.ts        |     100 |    70.58 |     100 |     100 | 26-61
  rewardService.ts         |   94.44 |    22.22 |     100 |     100 | 27-102
  tokenService.ts          |   76.92 |       20 |     100 |   76.92 | 20-25
  transactionService.ts    |    91.3 |    76.19 |     100 |    96.9 | 132,219,285
  userService.ts           |   65.21 |    55.55 |   66.66 |    70.9 | 29,104-145
---------------------------|---------|----------|---------|---------|-------------------------------------

Test Suites: 5 passed, 5 total
Tests:       277 passed, 277 total
Snapshots:   0 total
Time:        156.72 s
Ran all test suites.
```

Figure 8.1: Jest Report - Code Coverage.

8.1.3 Throughput Testing

Throughput testing is a crucial aspect of performance testing that evaluates a the capability of a system to handle a significant number of transactions or requests under high load conditions. To conduct throughput testing, the Artillery [100] framework was utilized.

It's worth noting that, in this particular test, the backend was modified to simulate the absence of connections to the blockchain and IPFS networks. Dummy data was used in place of actual interactions with these external services. Also, to maintain uniform testing conditions, the database data was reseted before each test. This approach allowed for the evaluation of the platform without dependencies on third-party services, focusing solely on the performance of the internal components that can be controlled, and ensured all tests were conducted under the same initial conditions.

During testing, a script consisting of four user journeys was executed to evaluate the backend performance. These user journeys covered various actions such as joining a program, receiving tokens or redeeming rewards, and checking the corresponding balances. For vendors, the user journeys included creating new program, token, and reward templates, as well as updating the program details.

The Artillery framework creates virtual users that represent regular users in a real-world scenario. Upon creation, these virtual users randomly select one of the four predefined journeys and follow it until completion. The framework also provides feedback organized by each endpoint, enabling detailed analysis of performance metrics.

8.2 Validation

This section assesses whether the developed platform meets the specified requirements and can be considered a success. This is achieved by analyzing the results obtained from the testing phase and comparing them against the initial requirements.

8.2.1 Functional Requirements Validation

The functional requirements outlined in Section 4.1 were validated to ensure that they were implemented correctly. Table 8.1 provides an overview of the requirements and whether they behave as expected. A feature is considered to behave as expected if it passes all of the related unit tests.

ID	Feature	Priority	Expected Behaviour
	Authentication		
FR1	Log in with email and password credentials	M	Yes
FR2	Register new account with email and password credentials	M	Yes
FR3	Log in with MetaMask Wallet	M	Yes
FR4	Register new account with MetaMask Wallet	M	Yes
	Tokens		
FR5	Create a token template (only upload the token metadata, including image, to the IPFS network)	M	Yes
FR6	Soft delete a token template	W	-
FR7	Fetch all the details of a token by its ID	M	Yes
FR8	Fetch the list of token templates given a list of IDs	S	Yes
FR9	Fetch the list of token templates created by a certain Vendor	M	Yes
FR10	Fetch the list of token templates that a certain user has balance of	M	Yes
FR11	Fetch the balance of a token owned by a certain user	M	Yes
	Rewards		
FR12	Create a reward token template (only upload the token metadata, including image, to the IPFS network)	M	Yes
FR13	Soft delete a reward token template	W	-
FR14	Fetch all the details of a reward token by its ID	M	Yes
FR15	Fetch the list of reward token templates given a list of IDs	S	Yes
FR16	Fetch the list of reward token templates given a certain reward smart contract ID	S	Yes
FR17	Fetch the list of reward token templates created by a certain Vendor	M	Yes
FR18	Fetch the list of reward token templates that a certain user has balance of	M	Yes

Table 8.1 continued from previous page

ID	Feature	Priority	Expected Behaviour
FR19	Fetch the balance of a reward token owned by a certain user	M	Yes
FR20	Insert the details of a reward smart contract	S	Yes
FR21	Soft delete a reward smart contract	W	-
FR22	Fetch the details of a reward smart contract by its ID	S	Yes
FR23	Fetch the list of reward smart contracts given a list of IDs	S	Yes
FR24	Fetch the list of reward smart contracts inserted by a certain Vendor	S	Yes
	Programs		
FR25	Create a new program	M	Yes
FR26	Edit a program	M	Yes
FR27	Soft delete a program	W	-
FR28	Fetch all the details of a program by its ID	M	Yes
FR29	Fetch the list of programs created by a certain Vendor	M	Yes
FR30	Fetch the list of programs a certain user has joined	M	Yes
FR31	Fetch the list of programs given a name	M	Yes
FR32	Fetch the list of programs given a location	S	Yes
FR33	Join/leave a certain program	M	Yes
	Transactions		
FR34	Mint a new batch of a certain token to some user	M	Yes
FR35	Transfer balance of a certain token to some other user	M	Yes
FR36	Fetch the transaction history of a certain token	S	Yes
FR37	Redeem a reward	M	Yes

Table 8.1 continued from previous page

ID	Feature	Priority	Expected Behaviour
FR38	Send an email notification to a user when he was enough token balance to redeem a certain reward from a certain program	W	-
FR39	Send an email notification to a user when he was token balance about to expire	W	-

Table 8.1: Functional Requirements Validation.

Upon reviewing the table, it is evident that all the Must Have priority requirements were successfully implemented and behaved as expected. Similarly, all of the Should Have requirements were addressed and implemented according to their expected behavior.

It is important to note that none of the Won't-Have requirements were developed as they were intentionally excluded from the project scope. Therefore, the "Expected Behavior" column for the Won't Have requirements is left blank, as there were no specific expectations associated with them.

8.2.2 Non-functional Requirements Validation

The non-functional requirements presented in Section 4.3 were also evaluated.

RNF1 - Security

To validate the security requirement, the scenarios 4.3 and 4.4 were analysed. The expected response in both cases should be "401 - Unauthorized". Based on the information presented in Table 8.1, it is evident that these scenarios were included in the tests for each feature, and they behaved as expected. Therefore, the security requirement was successfully validated.

RNF2 - Throughput

The throughput requirement includes two specific scenarios: Scenario 4.5 and Scenario 4.6. To simulate sufficient load on the platform, the Artillery script was configured to create 6 virtual users per second and gradually increase this number to 12 over a span of 5 minutes (300 seconds), as depicted in Figure 8.2.

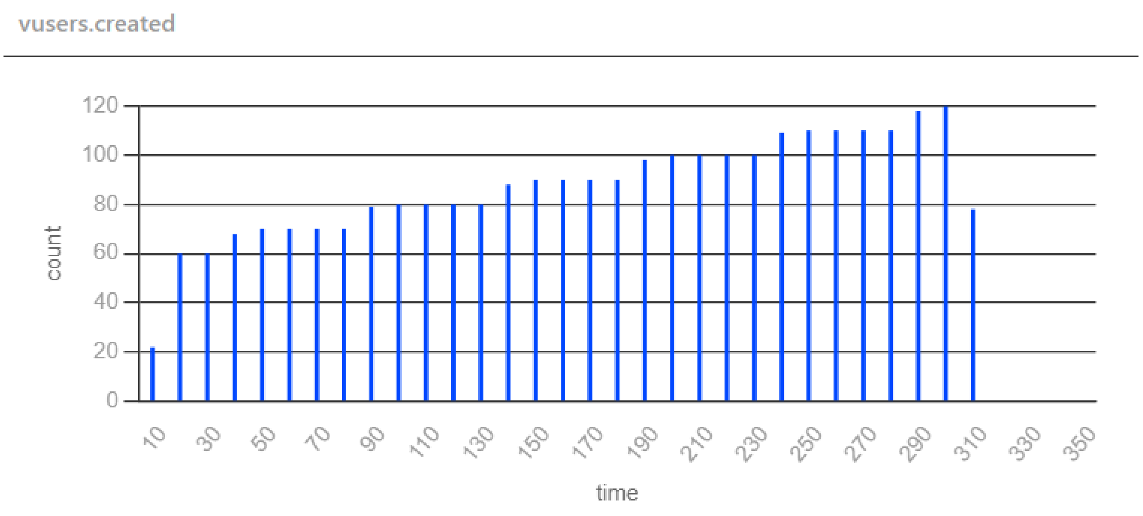


Figure 8.2: Graphical representation of the virtual users created along the span of an Artillery script run.

Figure 8.3 displays the number of requests sent during a test run, while Figure 8.4 provides the corresponding responses. Both graphs show a consistent number of requests and responses per second, except for a spike between seconds 330 and 340. During this time interval, the metrics reached approximately 90 requests per second (900 requests in 10 seconds) and 140 responses per second (1400 responses in 10 seconds).

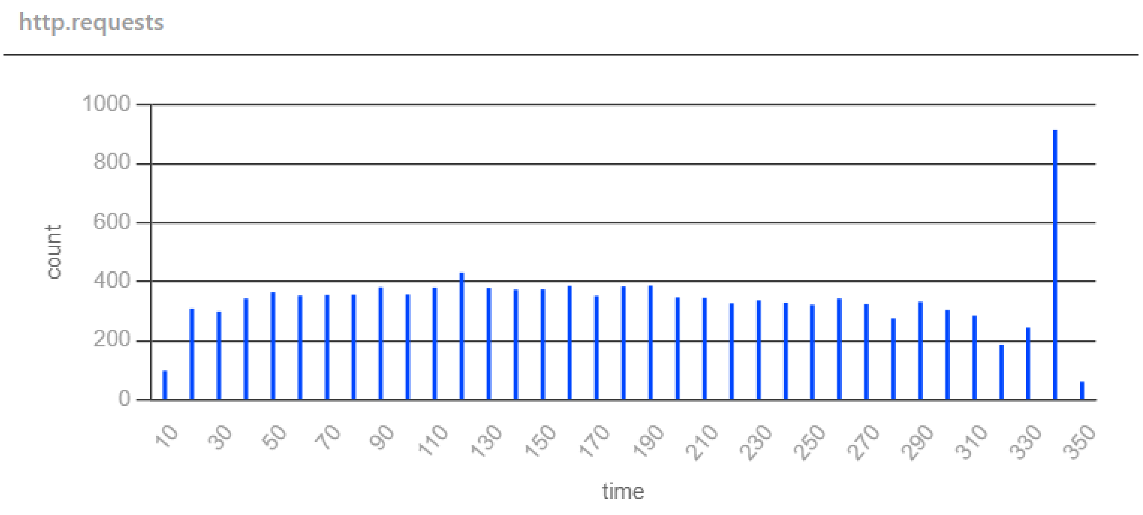


Figure 8.3: Graphical representation of the sent requests along the span of an Artillery script run.

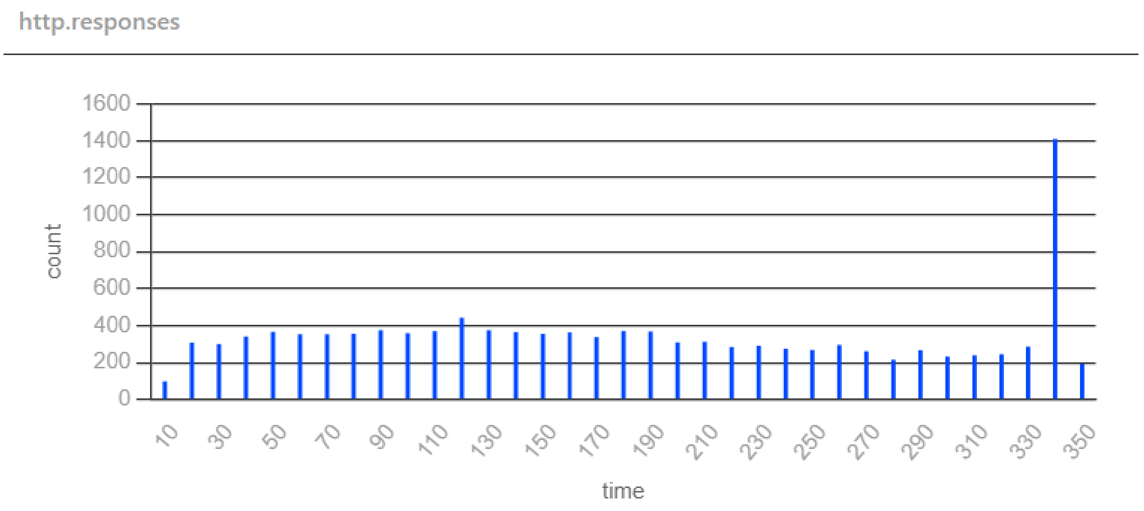


Figure 8.4: Graphical representation of the received responses along the span of an Artillery script run.

The presence of this spike was unexpected, as we initially anticipated a linear increase in both the number of requests per second and the corresponding responses, aligned with the increasing number of created virtual users. The sudden increase in requests and responses suggests that the virtual machine used for testing may have experienced resource starvation. This resource limitation likely hindered the process of sending requests until virtual users stopped being created. However, further investigation is required to precisely determine the cause of this anomaly and to address any potential resource limitations or networking bottlenecks (connection between the virtual machines). Nonetheless, for the purpose of the evaluation, we can proceed with the analysis of the results obtained so far.

The first scenario focuses on the response time of the requests that access the blockchain. According to the requirement, the platform should respond within 15 seconds in this scenario. Figure 8.5 displays the average response times of all the endpoints that would access the blockchain if the platform was not modified. To simulate the blockchain access, an offset of the average block time for the Polygon POS network, which is 2.22 seconds as of July 4th [101], can be added to the response times obtained in the tests. This calculation provides an estimate of the overall response time, including the response time measured during the tests and the average block time.

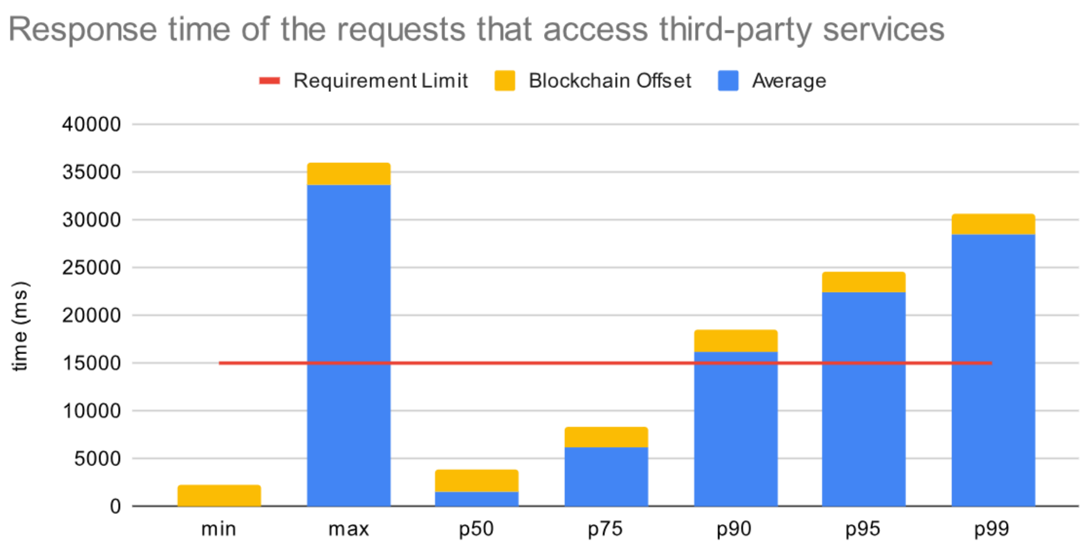


Figure 8.5: Graphical representation of the response times of the endpoints that access the blockchain.

The second scenario addresses the response time of the requests that do not access the blockchain. This scenario specifies that the platform should respond within 5 seconds when the platform is overloaded. Figure 8.6 exposes the average response times of all the endpoints that do not access the blockchain.

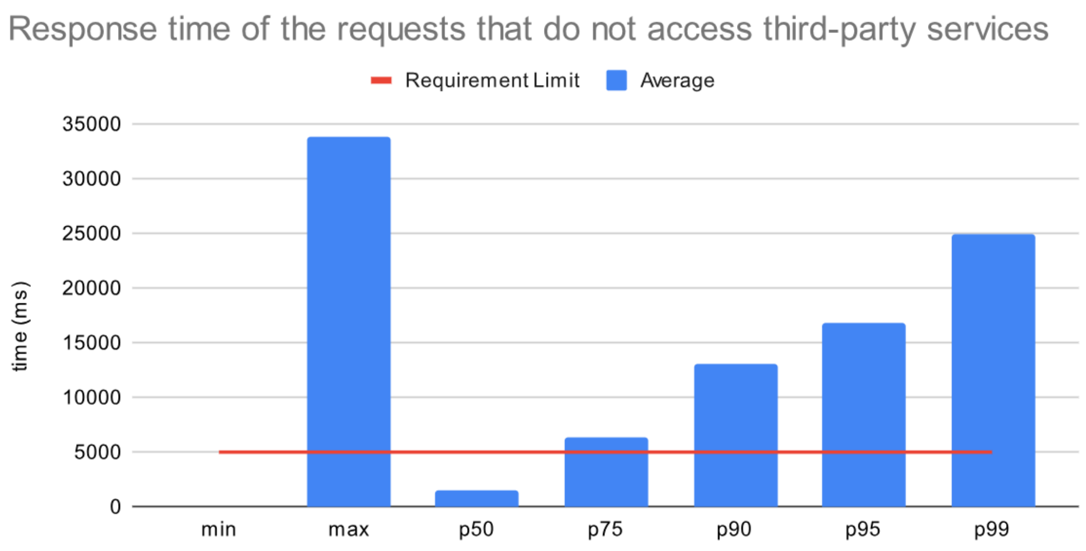


Figure 8.6: Graphical representation of the response times of the endpoints that do not access the blockchain.

Based on this analysis, the observed average load of 30 requests per second (excluding the spike at the end) does not exert sufficient load on the platform. More significant load would be expected in a platform of this kind. Additionally, the existence of the spike in requests distorts the obtained results, as the spike contains a great amount of entries with inflated response times. In a real-world scenario, a spike like this is typically more linear and gradual in its increase of requests.

RNF3 - Fault Tolerance

In order to validate the fault tolerance requirement the scenarios 4.7, 4.8, and 4.9, were analyzed. The first one involves testing the platform's ability to handle invalid inputs, and every endpoint performs input validation, and if an invalid input is detected, the platform responds with a detailed message specifying the validation failure. This scenario was included in the unit tests, ensuring that the platform handles invalid inputs appropriately.

The second scenario revolves around handling the requests when the blockchain or the IPFS connection breaks. Currently, the platform reports to the user with a detailed message if the blockchain connection fails. However, in the case of the IPFS network connection breaking, the platform simply returns a "500 - Internal Server Error" message. Further enhancements could be made to provide more informative responses in the case of IPFS connection failures.

As for the third scenario, it focus on how the platform handles requests when the connection to the database breaks. Currently, on occurrence, the platform returns a "500 - Internal Server Error" message. Improvements can be made to provide more meaningful feedback to users in such cases.

Summary

After evaluating the various scenarios, it is evident that the security requirement has been successfully validated. The fault tolerance requirement, on the other hand, can be considered partially validated, as some scenarios demonstrated its effectiveness while others did not. However, the throughput requirement cannot be validated at this stage, as the test results lack credibility and do not provide sufficient evidence to assess the performance of the platform in handling high loads. Further investigation and testing are needed to accurately validate the throughput requirement.

Chapter 9

Conclusion

This concluding chapter provides reflections and considerations on the project, along with an exploration of future work. It serves as an opportunity to summarize the key findings and experiences gained throughout the project journey.

The primary objective of this project was to explore the integration of blockchain technology with loyalty programs and understand the potential benefits that can be derived from this relationship. The initial phase involved conducting an in-depth literature review and research on both blockchain and loyalty programs, which provided the necessary background knowledge to come up with an approach for the problem presented by WIT Software. This led to the definition of requirements and user stories. By the end of the first semester, the proposed solution and its architecture were finalized, along with a preliminary selection of technologies.

Starting the second semester, the intern and their advisors adopted an adapted version of Scrum methodology, which proved effective in managing the project. At the beginning of the development phase, a project setup phase allowed the intern to familiarize himself with the initially selected technologies. However, as the intern gained a deeper understanding of the pros and cons of various technologies, adjustments and refinements were made in the technology selection. Throughout the development phase, minor requirement changes occurred, but they were properly prioritized and did not significantly impacted the overall project plan. Additionally, a scientific paper was submitted to the ISD 2023 Conference, although it was rejected due to the lack of real-world results at the time of submission.

Towards the end of the second semester, the developed platform underwent rigorous testing and evaluation to assess its compliance with the defined requirements. However, the results of the evaluation did not meet the success criteria, primarily due to limitations in validating the throughput requirement, as the obtained data lacked credibility.

Despite the testing limitations, the concept of a blockchain-based loyalty program platform shows promise. However, further testing is necessary to validate its throughput capacity and cost-effectiveness, which are crucial factors for real-world adoption.

This project allowed the intern with valuable opportunities to work and learn from other professionals in the field, gaining hands-on experience. From researching blockchain and working with smart contracts to developing a platform in Node.js, the intern acquired important knowledge and skills.

Moving forward, there are several areas of future work that can be considered to enhance the developed platform:

1. Throughput Testing:

   The data obtained from previous tests did not provide sufficient evidence to conclusively prove that the platform can meet the requirements. Further testing should be conducted to obtain more reliable and credible data.

2. Bug Fixes:

   During the throughput testing, it was observed that the platform encounters conflicts when concurrently creating programs. This is due to the calculation of the "programID" field, which counts the number of entries in the database. To address this issue, the database entities need to be restructured so that MongoDB utilizes the default "_id" field instead of the custom "programID" field, which was created for readability purposes.

3. Expand Feature Set:

   The platform can be further enhanced by implementing additional features and functionalities. For example, when fetching a list of programs, tokens, or rewards, the platform could allow multiple filters and calculate the intersection between them. Additionally, integration with more blockchain platforms in the rewards module could be explored, as the platform currently allows vendors to add their own reward smart contracts.

4. Cost Optimization:

   An analysis of the cost to run the platform can be conducted to identify potential optimizations. This includes evaluating transaction fees, resource utilization, and different node provider plans. Additionally, considering the possibility of hosting our own blockchain and IPFS nodes can further reduce reliance on external services and their associated costs. By hosting our own nodes, we can have more control over the infrastructure and potentially achieve cost savings in the long run.

References

[1] Mark D. Uncles, Grahame R. Dowling, and Kathy Hammond. Customer loyalty and customer loyalty programs. *Journal of Consumer Marketing*, 20 (4):294–316, Jan 2003. ISSN 0736-3761. doi: 10.1108/07363760310483676. URL https://doi.org/10.1108/07363760310483676.

[2] Bond. Pathways to Growth Guide. https://info.bondbrandloyalty.com/pathways-to-growth-guide, 2022. Accessed: 2022-12-12.

[3] Steve Fromhart and Lincy Therattil. Making blockchain real for customer loyalty rewards programs. *Deloitte center for financial services*, 2016.

[4] Manuel Utz, Simon Johanning, Tamara Roth, Thomas Bruckner, and Jens Strüker. From ambivalence to trust: Using blockchain in customer loyalty programs. *International Journal of Information Management*, 68:102496, 2023. ISSN 0268-4012. doi: https://doi.org/10.1016/j.ijinfomgt.2022.102496. URL https://www.sciencedirect.com/science/article/pii/S0268401222000275.

[5] Jan vom Brocke, Alan Hevner, and Alexander Maedche. *Introduction to Design Science Research*, pages 1–13. 09 2020. ISBN 978-3-030-46780-7. doi: 10.1007/978-3-030-46781-4_1.

[6] Ioannis ANTONIADIS, Stamatis KONTSAS, and Konstantinos SPINTHIROPOULOS. Blockchain and Brand Loyalty Programs: A Short Review of Applications and Challenges. *International Conference on Economic Sciences and Business Administration*, 5(1):8–16, November 2019. URL https://ideas.repec.org/a/icb/wpaper/v5y2019i18-16.html.

[7] Asnan Furinto, Teddy Pawitra, and Tengku E Balqiah. Designing competitive loyalty programs: How types of program affect customer equity. *Journal of Targeting, Measurement and Analysis for Marketing*, 17(4):307–319, December 2009.

[8] E-satisfaction. 8 types of customer loyalty programs! https://www.e-satisfaction.com/8-types-of-customer-loyalty-programs. Accessed: 2023-01-09.

[9] Jessica Huhn. Referralrock: 8 types of loyalty programs: Which is right for you? https://referralrock.com/blog/types-of-loyalty-programs, 2022. Accessed: 2023-01-09.

[10] Vodafone. Vodafone clube viva. https://www.vodafone.pt/loja/clube-viva.html. Accessed: 2022-12-15.

[11] Nespresso. Nespresso & more. https://www.nespresso.com/pt/pt/beneficios#!/entrega-gratuita. Accessed: 2022-12-15.

[12] Amazon. Amazon prime. https://www.amazon.es/amazonprime?language=pt, . Accessed: 2022-12-16.

[13] Amazon. Amazon smile. https://smile.amazon.com/, . Accessed: 2022-12-16.

[14] Wikipedia. Blockchain. https://en.wikipedia.org/wiki/Blockchain, . Accessed: 2022-12-16.

[15] Shubham Hapse. Velotio - blockchain 101: The simplest guide you will ever read. https://www.velotio.com/engineering-blog/introduction-to-blockchain-and-how-bitcoin-works. Accessed: 2022-12-20.

[16] Arun Sekar Rajasekaran, Maria Azees, and Fadi Al-Turjman. A comprehensive survey on blockchain technology. *Sustainable Energy Technologies and Assessments*, 52:102039, 2022. ISSN 2213-1388. doi: https://doi.org/10.1016/j.seta.2022.102039. URL https://www.sciencedirect.com/science/article/pii/S2213138822000911.

[17] Geeks for Geeks. Types of blockchain. https://www.geeksforgeeks.org/types-of-blockchain, . Accessed: 2022-12-20.

[18] Bhabendu Kumar Mohanta, Debasish Jena, Soumyashree S. Panda, and Srichandan Sobhanayak. Blockchain technology: A survey on applications and security privacy challenges. *Internet of Things*, 8:100107, 2019. ISSN 2542-6605. doi: https://doi.org/10.1016/j.iot.2019.100107. URL https://www.sciencedirect.com/science/article/pii/S2542660518300702.

[19] Christine Campbell. Techtarget: What are the 4 different types of blockchain technology? https://www.techtarget.com/searchcio/feature/What-are-the-4-different-types-of-blockchain-technology. Accessed: 2022-12-20.

[20] Crypto.com. Consensus mechanisms in blockchain. https://crypto.com/university/consensus-mechanisms-in-blockchain, . Accessed: 2022-12-21.

[21] Geeks for Geeks. Raft consensus algorithm. https://www.geeksforgeeks.org/raft-consensus-algorithm/, . Accessed: 2023-07-03.

[22] Geeks for Geeks. Proof of elapsed time (poet) in blockchain. https://www.geeksforgeeks.org/proof-of-elapsed-time-poet-in-blockchain/, . Accessed: 2023-07-03.

[23] Bybit Learn. Liquid proof of stake (lpos). https://learn.bybit.com/glossary/definition-liquid-proof-of-stake-lpos/. Accessed: 2023-07-03.

[24] Tendermint Core. What is tendermint. https://docs.tendermint.com/v0.34/introduction/what-is-tendermint.html. Accessed: 2023-07-03.

[25] Moralis. What are web3 contracts? exploring smart contracts. https://moralis.io/what-are-web3-contracts-exploring-smart-contracts/, . Accessed: 2022-12-21.

[26] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. An overview of smart contract and use cases in blockchain technology. In *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–4, 2018. doi: 10.1109/ICCCNT.2018.8494045.

[27] Osman Sönmeztürk, Tolga Ayav, and Yusuf M. Erten. Loyalty program using blockchain. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 509–516, 2020. doi: 10.1109/Blockchain50366.2020.00074.

[28] OpenZeppelin. Tokens. https://docs.openzeppelin.com/contracts/2.x/tokens. Accessed: 2022-12-21.

[29] Decrypt. How are nfts stored? on-chain, off-chain and decentralized storage. https://decrypt.co/resources/how-are-nfts-stored-on-chain-off-chain-and-decentralized-storage. Accessed: 2022-12-21.

[30] Juan Benet. Ipfs - content addressed, versioned, p2p file system, 2014.

[31] Crypto.com. What are token standards? an overview. https://crypto.com/university/what-are-token-standards, . Accessed: 2022-12-21.

[32] Ethereum. https://ethereum.org/en/. Accessed: 2022-12-22.

[33] ConsenSys. Ethereum has 4x more developers than any other crypto ecosystem. https://consensys.net/blog/developers/ethereum-has-4x-more-developers-than-any-other-crypto-ecosystem/. Accessed: 2022-12-22.

[34] Hyperledger Fabric. https://www.hyperledger.org/use/fabric. Accessed: 2022-12-22.

[35] Hyperledger Sawtooth. https://www.hyperledger.org/use/sawtooth. Accessed: 2022-12-22.

[36] Corda. https://corda.net/. Accessed: 2022-12-22.

[37] Tezos. https://tezos.com/, . Accessed: 2022-12-22.

[38] Kraken. What is tezos? (xtz). https://www.kraken.com/learn/what-is-tezos-xtz, . Accessed: 2022-12-22.

[39] Open Tezos. Liquid proof-of-stake. https://opentezos.com/tezos-basics/liquid-proof-of-stake/,. Accessed: 2022-12-23.

[40] Open Tezos. First contracts - nfts. https://opentezos.com/smart-contracts/simple-nft-contract-1/,. Accessed: 2022-12-23.

[41] EOSIO. https://eos.io/,. Accessed: 2022-12-23.

[42] Block.one. https://b1.com/. Accessed: 2022-12-23.

[43] EOSIO. eosio.evm: Code in solidity while leveraging eosio speed and scalability. https://eos.io/innovations/eosio-evm/,. Accessed: 2022-12-23.

[44] GitHub. Eosio sdk for java - api for integrating with eosio-based blockchains. https://github.com/EOSIO/eosio-java,. Accessed: 2022-12-23.

[45] Stellar. https://www.stellar.org/. Accessed: 2022-12-24.

[46] Polygon. https://polygon.technology/. Accessed: 2022-12-24.

[47] Cosmos. https://cosmos.network/. Accessed: 2022-12-24.

[48] Kraken. What is cosmos? (atom). https://www.kraken.com/learn/what-is-cosmos-atom,. Accessed: 2022-12-24.

[49] Amergent Hospitality Group. https://amergenthg.com/. Accessed: 2022-12-27.

[50] Mobivity. https://www.mobivity.com/. Accessed: 2022-12-27.

[51] Jaybee's Chicken Palace. https://jaybeeschicken.com/. Accessed: 2022-12-27.

[52] PizzaRev. https://pizzarev.com/. Accessed: 2022-12-27.

[53] Little Big Burger. https://littlebigburger.com/. Accessed: 2022-12-27.

[54] Burgers Grilled Right. https://bgrtheburgerjoint.com/. Accessed: 2022-12-27.

[55] American Burger Co. https://americanburgerco.com/. Accessed: 2022-12-27.

[56] Hospitality Technology. How this restaurant operator uses blockchain to reimagine loyalty. https://hospitalitytech.com/how-restaurant-operator-uses-blockchain-reimagine-loyalty. Accessed: 2022-12-27.

[57] American Express. https://www.americanexpress.com/,. Accessed: 2022-12-27.

[58] Boxed. https://www.boxed.com/. Accessed: 2022-12-27.

[59] Amazon. https://www.amazon.com/,. Accessed: 2022-12-27.

[60] Best Buy. https://www.bestbuy.com/. Accessed: 2022-12-27.

[61] Staples. https://www.staples.com/. Accessed: 2022-12-27.

[62] Ticketmaster. https://www.ticketmaster.com/. Accessed: 2022-12-27.

[63] American Express. Membership rewards. https://www.ticketmaster.com/,. Accessed: 2022-12-27.

[64] Singapore Airlines. https://www.singaporeair.com/,. Accessed: 2022-12-27.

[65] Singapore Airlines. Kris+. https://www.singaporeair.com/en_UK/us/ppsclub-krisflyer/use-miles/krisplus/,. Accessed: 2022-12-27.

[66] Singapore Airlines. Krisflyer to launch world's first blockchain-based airline loyalty digital wallet. https://www.singaporeair.com/en_UK/es/media-centre/press-release/article/?q=en_UK/2018/January-March/ne0518-180205,. Accessed: 2022-12-27.

[67] Etisalat. https://www.etisalat.ae/,. Accessed: 2022-12-27.

[68] Etisalat. Etisalat smiles. https://www.etisalat.ae/en/c/mobile/smiles.html,. Accessed: 2022-12-27.

[69] Khaleej Times. Etisalat launches 'smiles' customer engagement programme. https://www.khaleejtimes.com/article/etisalat-launches-smiles-customer-engagement-programme. Accessed: 2022-12-27.

[70] Telecom Review. Etisalat's smiles unveils uae's first blockchain-powered rewards exchange. https://www.telecomreview.com/index.php/articles/telecom-operators/4385-etisalat-s-smiles-unveils-uae-s-first-blockchain-powered-rewards-excha
Accessed: 2022-12-27.

[71] Wikipedia. Moscow method. https://en.wikipedia.org/wiki/MoSCoW_method,. Accessed: 2023-07-04.

[72] PolygonScan. Polygon (matic) blockchain explorer. https://polygonscan.com/,. Accessed: 2022-12-24.

[73] Messari. Ethereum (eth) average transaction fees. https://messari.io/charts/ethereum/txn-fee-avg. Accessed: 2022-12-24.

[74] Infura. https://www.infura.io/. Accessed: 2023-01-14.

[75] Moralis. https://www.moralis.io/,. Accessed: 2023-01-14.

[76] Alchemy. https://www.alchemy.com/. Accessed: 2023-01-14.

[77] Chainstack. https://chainstack.com/. Accessed: 2023-01-14.

[78] GetBlock. https://getblock.io/. Accessed: 2023-01-14.

[79] QuickNode. https://www.quicknode.com/. Accessed: 2023-01-14.

[80] Alchemy Docs. Compute units. https://docs.alchemy.com/reference/compute-units. Accessed: 2023-01-14.

[81] Web3.js. https://web3js.readthedocs.io/en/v1.9.0/. Accessed: 2023-06-15.

[82] IPFS in JavaScript. https://docs.ipfs.tech/reference/js/api/#javascript-libraries. Accessed: 2023-06-15.

[83] Node.js. https://nodejs.org/en. Accessed: 2023-01-15.

[84] Python. https://www.python.org/. Accessed: 2023-01-15.

[85] MongoDB. https://www.mongodb.com/. Accessed: 2023-01-15.

[86] Microsoft Visual Studio Code. https://code.visualstudio.com/. Accessed: 2023-06-22.

[87] Remix Online IDE. https://remix.ethereum.org/. Accessed: 2023-06-22.

[88] Google Chrome. https://www.google.com/intl/en/chrome/. Accessed: 2023-06-22.

[89] MetaMask. https://metamask.io/. Accessed: 2023-06-22.

[90] MongoDB Compass. https://www.mongodb.com/products/compass. Accessed: 2023-06-22.

[91] GitHub. https://github.com/,. Accessed: 2023-06-22.

[92] Atlassian. Jira software. https://www.atlassian.com/software/jira. Accessed: 2023-06-23.

[93] Figma. https://www.figma.com/. Accessed: 2023-06-23.

[94] Polygon Faucet. https://faucet.polygon.technology/. Accessed: 2023-06-22.

[95] Shopify. https://www.shopify.com/. Accessed: 2023-06-22.

[96] Swagger UI. https://swagger.io/tools/swagger-ui/. Accessed: 2023-06-26.

[97] React. https://react.dev/. Accessed: 2023-06-26.

[98] Jest. https://jestjs.io/. Accessed: 2023-06-29.

[99] GitHub. Supertest. https://github.com/ladjs/supertest#readme,. Accessed: 2023-06-29.

[100] Artillery. https://www.artillery.io/. Accessed: 2023-06-29.

[101] PolygonScan. Polygon pos chain average block time chart. https://polygonscan.com/chart/blocktime,. Accessed: 2023-07-05.