Instituto Superior
de Engenharia
Politécnico de Coimbra

DEPARTMENT OF SYSTEMS AND COMPUTER ENGINEERING

Blockchain-Based Loyalty Management System

Internship Report to fulfill the Master's degree in Informatics Engineering

Specialization Area of Software Engineering

Author

André Francisco Carvalho Santos

Supervisor

José Marinho

Advisors in the organisation WIT Software

Raul Fonseca

Tiago Agostinho

Ane Polito

Coimbra, September 2023

INSTITUTO POLITÉCNICO
DE COIMBRA

INSTITUTO SUPERIOR
DE ENGENHARIA
DE COIMBRA

RESUMO

As plataformas de fidelização são projetadas para aumentar a lealdade dos clientes e, assim, elevar o interesse dos consumidores em realizar compras. Embora tenham sucesso em aumentar o alcance da marca e as vendas, estas plataformas não conseguem alcançar o seu objetivo principal devido à falta de incentivos e estímulo para que os clientes retornem. Além do problema em gerar vendas, estes programas trazem custos excessivos às marcas devido à manutenção e infraestrutura necessária para tornar os sistemas viáveis. Nesse sentido, a mais recente tecnologia blockchain pode ajudar a superar alguns desses problemas, fornecendo recursos como contratos inteligentes, que têm o potencial de reinventar a forma como os sistemas de fidelidade funcionam e resolver problemas atuais. Embora a blockchain seja uma tecnologia relativamente recente, algumas marcas já estão a investigar sua utilidade e a reconstruir os sistemas de fidelidade. No entanto, essas plataformas são independentes entre si e vinculadas diretamente a uma marca. Assim, há a necessidade de uma plataforma genérica capaz de criar e gerir diferentes programas de fidelidade, independentemente do tamanho do negócio. De forma a estudar abordagens inovadoras e explorar tecnologias de ponta, a empresa WIT Software oferece estágios que se aplicam em contextos relevantes. Em concordância com essa visão, o estágio em foco neste relatório pretendeu criar um sistema de gestão de programas de fidelidade descentralizado, especialmente projetado para pequenos comerciantes e capaz de gerir vários mecanismos e programas de fidelidade. O âmbito do estágio envolveu inicialmente uma extensa pesquisa a nível científico e de mercado, permitindo que a WIT obtivesse uma visão sobre as vantagens e desvantagens de várias abordagens. Uma vez concluída a fase de pesquisa, o estágio prosseguiu com o design e a implementação do protótipo. Este documento proporciona uma explicação detalhada dos esforços realizados durante a investigação e implementação do protótipo ao longo do estágio. Como prova do progresso significativo alcançado, foi produzido um artigo científico e posteriormente publicado sobre o assunto, demonstrando a dedicação e o sucesso do estágio.

Palavras-chave: Programas de fidelização; blockchain; contractos inteligentes; gestão de programas de fidelização

ABSTRACT

Loyalty platforms are designed to increase customer loyalty and thus increase consumers' attraction to purchase. Although successful in increasing brand reach and sales, these platforms fail to meet their primary objective due to a lack of incentives and encouragement for customers to return. Along with the problem in originating sales, they bring excessive costs to brands due to the maintenance and infrastructure required to make the systems feasible. In that sense, recent blockchain technology can help to overcome some of these problems, providing capabilities such as smart contracts, which have the potential to reinvent the way loyalty systems work and solve current problems. Although blockchain is a relatively recent technology, some brands are already investigating its usefulness and rebuilding their loyalty systems. However, these platforms are independent and linked directly to a brand. Thus, there is a need for a generic platform capable of creating and managing different loyalty programs, regardless of the size of the business. On a mission to discover innovative approaches and explore cutting-edge technologies, the company WIT Software offers internships that delve into pioneering scenarios. In line with this vision, the internship in this report endeavored to create a decentralized loyalty management system, specifically catered to smaller retailers and capable of managing multiple loyalty mechanisms and programs. The scope of the internship initially involved extensive research at both the scientific and market levels, allowing WIT to gain insights into the advantages and disadvantages of various approaches. Once the research phase concluded, it proceeded to design and implementation of a prototype. This document serves as an exhaustive explanation of the investigation and implementation efforts undertaken throughout the internship. As a testament to the significant progress made, a scientific paper was produced and subsequently published on the subject, showcasing the dedication and success of the internship.

Keywords: loyalty programs; blockchain; smart contracts; loyalty management

ACKNOWLEDGMENTS

Firstly, I would like to extend my sincere gratitude to WIT Software for providing me with the incredible opportunity to undertake an internship in such an innovative field as blockchain. This project not only offered me new experiences but also enriched me with valuable knowledge.

Moreover, I want to express my heartfelt appreciation to Raul Fonseca for his unwavering guidance and expertise in the field, which proved instrumental in making crucial decisions throughout the project. A special note of thanks goes to Tiago Agostinho and Ane Polito for their continuous support and active involvement in the project's development. Both played pivotal roles in the project's success, with Ane Polito contributing to the requirements and UI design and Tiago Agostinho providing consistent help and valuable advice during the development process.

I would also like to convey my deep appreciation to José Marinho, who played a crucial role in guiding and assisting me throughout the internship, particularly in the preparation of the scientific paper and dissertation. His contributions and advice significantly contributed to shaping the content and enhancing the quality of my work.

Lastly, I wish to express my profound gratitude to my family and friends for their unwavering support, encouragement, and patience during the years of my academic journey.

INDEX

FIGURES INDEX

TABLES INDEX

ACRONYMS

Acronyms	Name
AVM	Avalanche Virtual Machine
API	Application Programming Interface
DAO	Data Access Object
DApp	Decentralized Application
EVM	Ethereum Virtual Machine
ERC	Ethereum Request for Comments
FMCG	Fast-moving Consumer Goods
FT	Fungible Tokens
GET	Green Electricity Tariffs
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
IPFS	InterPlanetary File System
ISEC	Coimbra Institute of Engineering (Instituto Superior de Engenharia de Coimbra)
JSON	JavaScript Object Notation
L1	Layer 1
L2	Layer 2
NFT	Non-Fungible Tokens
P2P	Peer-to-peer
PoS	Proof of Stake
PoW	Proof of Work
QR	Quick Response
ROI	Return on Investment
RPC	Remote Procedure Call
SDK	Software Development Kit
SDT	Self-determination theory
TPS	Transactions per second
UI	User Interface
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
USB	Universal Serial Bus

1 INTRODUCTION

This report aims to describe the work conducted during the internship at WIT Software company [1], in the scope of the curricular unit "Internship" of the second year of the Software Engineering Master at Coimbra Institute of Engineering (ISEC) [2]. This internship was supervised by Professor José Marinho (ISEC), by Director of Software Engineering Raul Fonseca (WIT Software), by Business Analyst Ane Polito (WIT Software), and by Tiago Agostinho, Senior Software Engineer (WIT Software)

The internship, performed in a hybrid regime (personally and remotely), aimed to explore the emergence of blockchain technology and improve the current traditional loyalty systems by leveraging blockchain tools and involving several other technologies and tools. This project had two main goals:

- Help the company to gain an overview of how blockchain can impact the loyalty marketing area;
- Create a prototype for demonstration purposes.

Initially, this project/internship was intended to have a reduced scope, focusing on developing only the mobile application for user interaction and connecting it to an already built backend. However, to align with the expectations of a master's internship and accommodate additional requirements, the scope was extended to also include business logic.

The internship comprised two phases. The first phase was dedicated to study the technologies and the state-of-the-art, while the second phase entailed creating and developing a prototype based on the knowledge gained in the initial stage. The obtained prototype consists of a decentralized blockchain-based loyalty management system that aims to manage several and distinct loyalty programs in a single environment for smaller retailers.

1.1 WIT Software

WIT Software [1], a Portuguese company, has amassed over two decades of expertise in the telecom industry. Founded in Coimbra in 2001, WIT Software has expanded its operations to encompass various domains, including Messaging, Voice and Video, Unified Communications, Mobile Money and Fintech Services, M-Commerce, IPTV and OTT-TV, Telecom VAS services, Core Network Services, Self-Care, Internet-of-Things, and Mobile Security [3]. What distinguishes WIT apart is its commitment to providing customers with a comprehensive and quality software journey using the best practices of agile development methodologies, including SAFE [4], Scrum [5] and Kanban [6].

The company's foundation is built upon four core principles: a Global Ambition to tackle new challenges, Success with Integrity by respecting and acting with honesty, Intelligence to Serve Others as problem-solvers who actively learn, and a mindset of being a game changer by thinking big, innovating, and adapting to market demands. Currently, WIT Software employs over 350 professionals across its offices in Coimbra, Porto, Lisbon, Aveiro, Leiria, and the United Kingdom, delivering software solutions to more than 40 countries. Recently, WIT Software invested in blockchain technologies, undertaking pilot projects, and creating prototypes to assess their potential in the market and to expand its business.

1.2 Coimbra Institute of Engineering

Located on the right bank of the Mondego River, the Coimbra Institute of Engineering [2], with over forty years of experience educating and forming students, is now one of the academic units that make up the Coimbra Polytechnic Institute. It emerged in 1974 from the transformation of the former Coimbra Industrial and Commercial Institute (established in December 1921).

Asserting its duty to provide an excellent education in the field of engineering and aiming to be a reference of excellence in both national and international education, ISEC's mission is "the creation, transmission, and dissemination of culture, science, and technology, including providing higher education for professional activities in the field of engineering."

It is also essential to highlight the strong connection that ISEC maintains with various organizations, which allows for a more remarkable interaction between companies and students, thereby enhancing success in the job market.

1.3 Work Plan and Objectives

The internship aimed to provide the company with insights into the potential impact of blockchain on the loyalty marketing field by developing a prototype to showcase its capabilities and functionalities. To ensure the successful accomplishment of this goal, the following objectives were established in the internship proposal (Annex A):

- Acquisition of detailed knowledge about the functioning of blockchain and the development of decentralized applications (DApps) with smart contracts;
- Acquisition of knowledge about existing loyalty mechanisms/types;
- Definition of the prototype's requirements;
- Implementation of the Android mobile app for the users;
- Implementation of the loyalty mechanisms/types through smart contracts;
- Demonstration of the prototype.

To accomplish these objectives as well as to create a detailed report, the internship was organized based on the following tasks:

- T1 - State of the art: understand how blockchain can impact the loyalty programs and study programs with and without it;
- T2 - Requirements: identify, define, and specify the system requirements as well as the user requirements;
- T3 - Solution design: create the mockups and the system architecture;
- T4 - Planning: elaborate on the software development planning;
- T5 - Development: implement the prototype based on the requirements;
- T6 - Tests: test the solution;
- T7 - Documentation: write the documentation of the project and the current report;
- T8 – Scientific Paper: write and publish a scientific paper.

During the internship, a new goal was added, which consisted of the publication of a scientific paper [7]. Table 1-1 presents the schedule of the tasks performed along the internship.

Table 1-1 - Schedule of Tasks

	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	July
T1											
T2											
T3											
T4											
T5											
T6											
T7											
T8											

1.4 Context and methodology

In adherence to WIT Software's guidelines and principles for project development, we employed an adaptation of the SCRUM methodology. SCRUM is an iterative approach that contrasts with the traditional sequential method. This work method aims to provide prompt and dynamic responses based on client feedback, where the workload is divided into multiple sprints, each representing a condensed timeframe instead of delivering the completed product all at once. The division of the workload allows the team to assess the project status, therefore, enabling the adjustment of the project priorities according to the project/client necessities. To maintain project progress, the team conducts regular meetings, either daily or weekly, to review completed and upcoming tasks, as well as identifying any potential obstacles that may delay progress. At the end of each sprint, the team convenes to evaluate the progress made and make necessary adjustments to the project. A SCRUM team consists of three key roles: the scrum master, responsible for managing the project and team; the product owner, responsible for facilitating communication between the customer and the team; and finally, the development team.

In the context of this internship, the development period was divided into twelve sprints, each with two weeks, with two meetings of ten minutes per week to discuss and demonstrate the advancement of the project. Along with these meetings, WIT held a monthly status meeting with the company employees to provide the intern with outside perspectives and train for the final presentation. As previously mentioned, the objective of the internship was to develop a prototype and assess its capabilities. Consequently, the final product is not to be retained. Due to internship nature, the project does not involve an actual customer, as the recipient of the prototype is WIT itself.

1.5 Report Structure

The remainder of this report is organized as follows:

- **Chapter 2 – Loyalty programs and blockchain:** This chapter provides better contextualization about loyalty programs and their current problems. It also presents the main characteristics of the blockchain technology and how it could impact loyalty programs.
- **Chapter 3 – State of the art:** This chapter comprehends an extensive analysis of the scientific work. Also, it identifies competing apps on the market and other pilot projects.
- **Chapter 4 – Technologies and Tools:** This chapter exposes the technologies and tools used for the development of the prototype system.
- **Chapter 5 – Proposed loyalty management system:** This chapter explains the concept of a decentralized application. Moreover, it describes how the

system works, as well as functional and non-functional requirements, the UI design, and provides a detailed description of the system architecture.

- **Chapter 6 – System Implementation:** This chapter presents all the implementation details and considerations during the development. Additionally, it gives an overview of another implemented aspects.
- **Chapter 7 – Discussion:** This chapter aims to discuss about the prototype implemented. Furthermore, a comparison is made with the scientific and market approaches to evaluate the prototype. Also, it provides a list of the system benefits and limitations.
- **Chapter 8 – Conclusions and Future work:** This final chapter aims to gather all the information presented in the document and provide conclusions regarding the results of the internship and the success of the prototype. Additionally, potential future work to enhance the system is defined.

2 LOYALTY PROGRAMS AND BLOCKCHAIN

This chapter first provides an overview of the current state of loyalty programs and the challenges they face. In addition, it clarifies how blockchain technology has the potential to revolutionize these programs by introducing technical concepts that can facilitate a more effective approach.

A loyalty program is a marketing strategy used by companies to reward their most loyal customers. Its other goals include attracting new customers, encouraging repeat purchases, and collecting information about customers. Currently, there is a wide range of loyalty programs, which is an issue since different applications are needed, and the rewards do not suit all customers. Moreover, there is no interaction between the different programs, which makes it difficult to keep track of the status of all of them, and means users have to provide their information multiple times.

Some loyalty programs have found a way to operate with different businesses, but there are many difficulties in tracking information outside of their scope, for example, when a partner provides information for an external service.

Several loyalty management systems have emerged as a way of overcoming these problems. These loyalty management systems provide new possibilities to build loyalty programs, and the tools to monitor them, faster. At the same time, they offer a way to create an ecosystem of loyalty programs that supports the user experience. While they solve drawbacks such as user experience, they still have problems related to third-party tools and incomplete connection between different systems.

Therefore, blockchain may be a viable option to address these challenges. The concept of blockchain consists of a chain of blocks that allows the storage of information. Each block stores a set of transactions, and these blocks are part of a network that any user can access. This technology introduces a new paradigm based on three core ideas: security, decentralization, and disintermediation [8].

For maximum security, this technology uses distributed consensus algorithms that ensure that every transaction is trustworthy. Additionally, it utilizes mechanisms of cryptography and digital signatures to ensure that only the legitimate owner can access its data [8]. A final security-related concept is persistence, which means the data cannot be changed. Concerning decentralization and disintermediation, these two properties play a crucial role. Decentralization allows the user to control and to be the owner of the information it produces, as opposed to the current reality of large companies such as Meta and Google. Another property of a decentralized network is the trustless environment, which means no one needs to know or trust another entity, and each member has a copy of the data. This environment prevents anyone from corrupting the data and optimizes the distribution of resources [9].

Since blockchain does not use third-party tools to analyze and verify transactions, it uses the concept of smart contracts. Smart contracts are self-executing programs stored on the blockchain. These contracts are triggered when the established

conditions are met. Usually, they are used to automate the execution of an agreement between entities so that all of them can be aware of the expected result. This mechanism brings benefits such as performance and efficiency, trust and transparency, and security, as it only works as long as the conditions are met and stored later in the immutable blockchain. Additionally, blockchain may allow people to save money as it avoids delays, fraud, and intermediaries [10].

Summarizing, loyalty programs are one of the main options for brands to increase their accessibility and attractiveness. However, these systems have shortcomings that can influence interaction with users. Blockchain is an innovative technology capable of transforming the behavior of these systems and solving current problems. For better understanding and context, this chapter describes in detail the issues of current loyalty programs and presents how blockchain technology complements these programs. Also, it explains concepts regarding the blockchain, such as smart contracts and meta transactions, for better understanding its behavior.

2.1 Blockchain

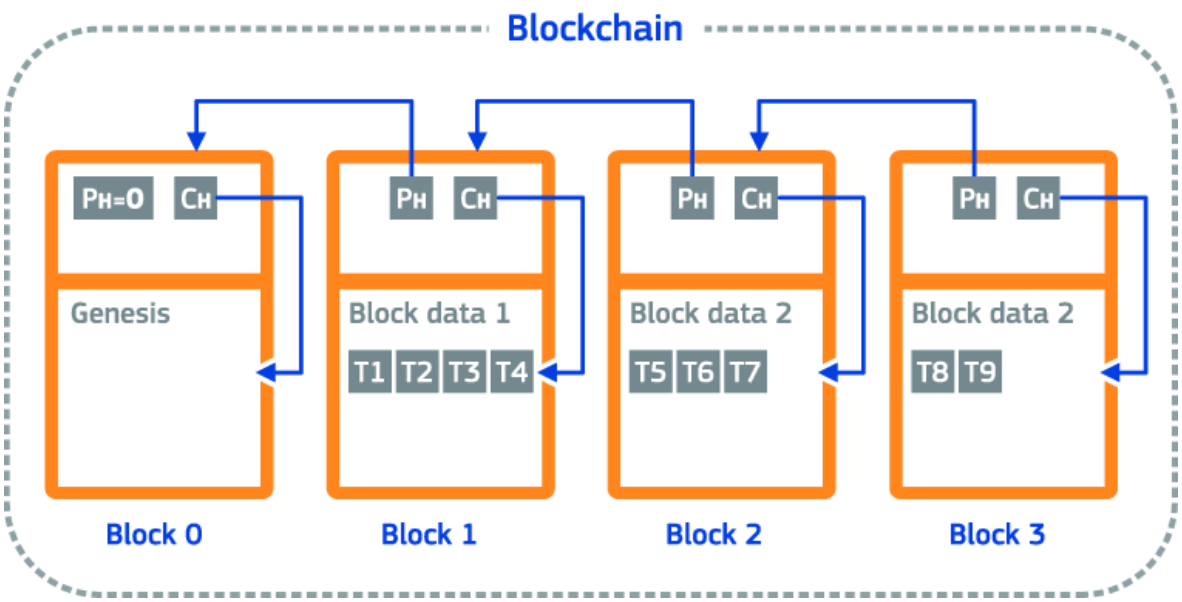
Blockchain is an emerging technology that introduced a new paradigm for secure storage and sharing of information. This technology is essentially a distributed and decentralized database consisting of a chain of interconnected blocks. Each block contains a set of transactions and has a link to the predecessor. The blockchain infrastructure operates as a peer-to-peer (P2P) network, where every computer, referred to as a node, plays a crucial role. Each node holds equal significance and actively contributes to the establishment and management of the network. For achieving maximum security, blockchain utilizes a range of mechanisms such as cryptography, consensus algorithms, and digital signatures [8].

Blockchain functioning involves the collective collaboration of all nodes. These nodes communicate with each other through a protocol and are responsible for validating new blocks. Each node maintains an identical copy of the current network state to ensure data immutability and prevent data loss due to a node going offline. Whenever a new node joins the network, it receives a copy. Regarding validation, to avoid individual or organizational control, blockchain incentivizes the nodes to validate new blocks by rewarding them. The validation process involves utilizing consensus mechanisms that may vary depending on the specific blockchain that is chosen.

Figure 2.1 illustrates the structural organization of blockchain data. As mentioned earlier, the blockchain consists of a series of interconnected blocks. The initial block, often referred to as the genesis block or block 0, does not contain any inherent data but only possesses a hash value. Subsequent blocks added to the chain include their own hash value along the hash value of the preceding block. Additionally, each block stores the number of transactions executed within a time interval. These blocks are in temporal order, meaning the latest block is the most recently created.

In a general sense, transactions carried out on the blockchain can be described as follows. Initially, every transaction is distributed to all nodes within the network. Secondly, all nodes undertake the validation process to verify the authenticity and legitimacy of each transaction, ensuring it is not fraudulent. Once transactions are successfully validated, they are grouped into a candidate block on each node. Finally, a node adds this block to the blockchain once it solves a problem, enabling its publication. All transactions are performed in a decentralized way, removing the need for intermediaries and, once inside a block, the transaction cannot be deleted or reverted. This modern technology is based on the following core principles [11]:

- Decentralization – one of the main goals of blockchain is to remove an entity's ability to have absolute control of all data. Hence, this technology is based on a network in which every node has equal power and the nodes themselves validate the transactions [10];
- Immutability – every node in the network holds a copy of the digital ledger. To add a new transaction, all nodes must check the transaction and most of them must validate it. Validated records become irreversible and cannot be changed. In a scenario of a sudden change in one node, all nodes check the validity of the new state. When all nodes finish the verification, a voting process is started and, if the majority rejects the change, the new state is rejected. Then, a copy of a trusted digital ledger is sent to that node [11];
- Transparency – any user can access any record by accessing the node and consulting the timestamp [8];
- Security – when interacting with the blockchain, it is mandatory for the users to enter the private key every time they want to create a transaction. Each record is individually encrypted. The information inside the blockchain is hashed cryptographically, which means that each piece of data has its own unique identifier [11];
- Disintermediation – in order to avoid the use of third-party entities, blockchain takes advantage of smart contracts. These contracts are used to facilitate the negotiation process but still provide security and reliability to the process.



PH – Previous Hash, CH – Current Hash, Tn – Transaction Number

Figure 2.1 - Blockchain data structure high-level [12]

Although blockchain is a relatively new technology, there is growing interest in it and in research into its applicability across a range of sectors. In [8], the authors describe what distinguishes blockchain from currently known paradigms and characterize their specifications from fundamentals to architecture. After specifying the capabilities of the technology, the authors present recent studies on the incorporation of blockchain in certain areas.

2.1.1 Blockchain types and scalability solutions

Blockchain, as a cutting-edge technology, lacks established standards. Although blockchain networks are commonly public, there are also other permissioned blockchain types. Additionally, given the relative novelty of this technology, several scalability solutions exist that are relevant in the context of the current project. This section explains the current blockchain types and scalability solutions.

When classifying blockchain types, the primary categorization is whether they are public or private. However, there are additional possible classifications, known as consortium and hybrid, based on specific criteria [13]. It is essential to keep in mind that, regardless of their specific type, every blockchain operates as a cluster of nodes collaborating on a peer-to-peer network.

Public blockchains, also called permissionless blockchains, are a prominent type that is both fully accessible and decentralized. Users can join the network without requiring permission and anyone is authorized to participate in transaction validation, with rewards based on their success. There are no restrictions on who can use the network. Anyone can run a node and mining software, access a wallet, and contribute data to transactions as long as they adhere to the blockchain's rules

[13]. The primary characteristics center around trust, as each node in the network does not require prior knowledge or trust in each other thanks to the consensus algorithms. It is secure, as the ability to have numerous participants or nodes makes it increasingly challenging for hackers to tamper with the data since it is distributed across multiple nodes. Furthermore, it is open and transparent, as every transaction is public, and any node can obtain a copy of all records [13], [14]. Bitcoin [15] and Ethereum [16] are the most known blockchains of this type.

Private blockchains, also known as permissioned blockchains, limit access to users. Unlike public blockchains, where any user can access the data and join the network, private blockchains require permissions for users to join and have read and write access to the data. These types of blockchains are typically owned by private individuals or organizations, with a central authority responsible for managing the permissions [13]. The main features encompass privacy, as private blockchains enforce permissions to ensure confidentiality. They also offer faster transactions due to the smaller number of nodes, enabling quicker distribution. Furthermore, private blockchains provide better scalability as entities have the flexibility to adjust the number of nodes to meet specific requirements [14]. Hyperledger [17] projects and Corda [18] are two main examples.

Hybrid blockchains leverage the strengths of both public and private blockchains by combining them. They incorporate elements of centralized and decentralized systems and operate in a closed environment. Despite not being fully open, they uphold crucial features such as integrity, transparency, and security. This technology enables enterprises to establish a private permission-based system alongside a public permissionless system. Such an approach empowers organizations to control access to specific blockchain data while determining which data is made public [13].

Consortium blockchains, also named federation blockchains, are a semi-decentralized type of blockchain that aims to distribute power among a group of individuals or organizations rather than a single entity. It eliminates the concentration of power in one individual or organization. In this type of blockchain, read and write data access may be restricted, and transactions are verified by a specific set of nodes designated by the consortium. Consortium Blockchains are faster and provide higher scalability and transaction privacy [13].

Despite the development of various types of blockchains to accommodate different scenarios, they still have limitations that restrain their usage and performance. To overcome these limitations, several solutions have emerged to enhance blockchain networks. In the context of the project described in this report, Sidechains and Layer 2 can be considered two notable solutions. These solutions present alternative approaches to enhance the functionality and scalability of blockchain networks. To understand these solutions, it is important to mention that blockchain platforms can be built on top of existing blockchains, which act as the base layer. These base layer blockchains, such as Ethereum and Bitcoin, are commonly referred to as Layer 1.

Layer 1 (L1) refers to a network that serves as the foundation for another blockchain platform. These networks, also known as base chains or main chains, handle and validate transactions within their own ecosystem. Each network has its own native token, which is used to pay the associated fees for transactions on that specific network. A prevalent challenge faced by layer 1 networks is their limited scalability, primarily because implementing significant changes can be arduous and achieving consensus among a multitude of users can be difficult to attain [19].

Layer 2 (L2) refers to scaling solutions that work alongside mainnet blockchains. L2 operates as a separate blockchain, extending the capabilities of the underlying layer 1 while inheriting its security features. As the popularity of mainnet blockchains increases, it leads to increased transaction costs, necessitating scaling solutions. Layer 2 networks address this demand. Layer 2 blockchains interact with the mainnet by submitting transaction bundles, ensuring comparable security and decentralization without requiring changes to the layer 1 protocol. By offloading transaction processing to layer 2, the mainnet experiences reduced congestion and massively reduced fees, resulting in improved scalability. Furthermore, layer 2 maintains security by settling transactions on the mainnet and leveraging its security. This solution expands the possibilities for use cases by enabling higher transactions per second, lower fees, and enhanced user experience. As a result, projects can explore new applications with improved efficiency [20].

Sidechains are an independent and parallel blockchain that operates alongside the mainnet. They can have distinct block parameters and use different consensus algorithms, designed, and optimized to suit their needs. However, utilizing a sidechain involves trade-offs as they do not inherit the security properties of the mainnet. Unlike Layer 2 solutions, sidechains rely solely on their security mechanisms. To connect to the main chain, sidechains utilize a two-way bridge that enables the transfer of assets between the sidechain and the mainnet [21].

2.1.2 Consensus algorithms

Consensus mechanisms serve as crucial security measures and protocols within blockchains, enabling a distributed network of nodes to reach a consensus on the chain state. Consensus in a blockchain is a fault-tolerant mechanism, ensuring agreement on a single network state in a distributed multi-node system [22]. These mechanisms also incorporate a rewards system that incentivizes honest validators, penalizes dishonest behavior, and creates a significant cost barrier for attacking the network [23]. Achieving consensus in a blockchain network generally entails the agreement of a majority, such as when 51% of the nodes agree.

Figure 2.2 presents the existing consensus protocol in public and private blockchains. In line with the scope and objectives of the project described in this report, focus is placed on some of the public consensus mechanisms, specifically Proof of Work (PoW) and Proof of Stake (PoS), which are briefly explained next.

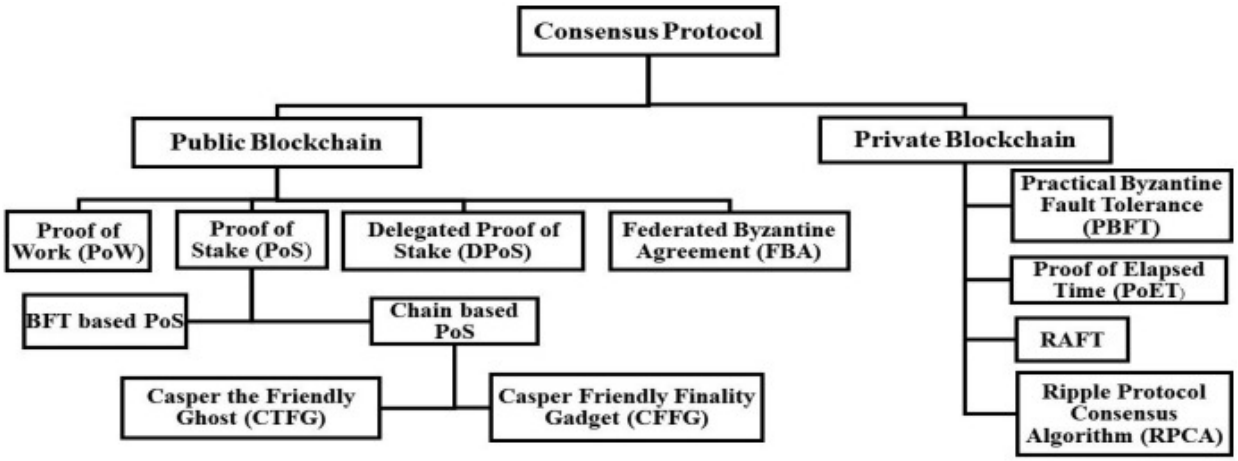


Figure 2.2 - Blockchain consensus mechanisms [22]

Proof-of-Work (PoW), the first crypto consensus mechanism used in projects such as Bitcoin and Ethereum, consists of a resolution of a cryptographic puzzle that miners (nodes that validate) need to complete to record a new block with the validated transactions [22]. The puzzle consists of an intense race of trial and error to find a valid hash for the block. In this cryptographic proof, the prover miner showcases to the verifiers that it has invested a particular amount of computing effort. This effort comes from a constant execution of a mathematically intensive function toward the objective of finding the correct hash that adheres to the predefined rules set by the blockchain. Even though being a time-consuming process with miners attempting millions of solutions, the puzzle is easily proven [22]. When a miner believes it has found a solution, it sends the block along with the hash and the other nodes simply need to verify if the provided hash is correct. Upon successful verification, the miner gets rewards for its efforts.

Proof-of-Stake (PoS) is an alternative to Proof-of-Work (PoW) as a consensus mechanism. With PoS, the selection of the new forger (validator node) occurs in a deterministic manner. Unlike PoW, where any validator can participate by utilizing computational power, PoS requires forgers to possess the native cryptocurrency and their forging capacity is proportional to the amount of native cryptocurrency they hold. With PoS, the selection of a miner or validator to create a new block is based on the wealth or stake of miners in the network [22]. For a validator to be chosen, it must stake or risk a certain amount of cryptocurrency to perform a validation. This process ensures that validators have a financial incentive to act in the best interests of the blockchain, as jeopardizing its integrity could result in the loss of its staked funds. Upon successful validation of a new block, the node is rewarded with the transaction fees associated with all transactions included in the block.

2.1.3 Wallet, tokens, and gas

Within the blockchain ecosystem, several key-aspects must be explained to enhance understanding and provide context. Primarily, a user must possess a crypto wallet (virtual wallet) to interact and perform transactions. Wallets are available in various

formats, including hardware devices (e.g., USB pens), browser extensions, or desktop/mobile applications [24]. Each wallet can accommodate multiple accounts (pairs of private and public keys) that are independent and not interconnected. The primary function of these wallets is to securely store the user's private key and facilitate secure transactions. To ensure transaction security, the wallet cryptographically signs transactions using the private key associated with the specific account [24]. Consequently, each account within the wallet will have a unique signature attributable to its distinct private key. The key pair, consisting of the public key (which can be shared to receive transactions) and the private key, plays a central role in wallet functionality. While the public key can be openly shared, the private key must be kept strictly confidential. The private key serves as the critical element for initiating fund transfers and, if shared, it could potentially lead to unauthorized access and theft of funds. It is imperative to emphasize that user funds are stored within the blockchain, not within the wallet. The wallet's function is confined to safeguarding a pair of cryptographic keys, allowing the user to interact with the blockchain system.

In addition to enabling user interactions with the blockchain, wallets inherently serve as storage for user funds. These funds can comprise native cryptocurrencies of the respective blockchains, as well as tokens. Tokens can be further classified into two categories: Non-Fungible Tokens (NFT) and Fungible Tokens (FT). In contrast to native cryptocurrencies, tokens are built on existing blockchains using smart contracts to represent digital assets. From the perspective of the blockchain itself, these tokens are simply data stored within the blockchain. When considering Non-Fungible Tokens (NFT) and Fungible Tokens (FT), a key differentiating factor is their interchangeability. NFT are unique, meaning that no two units are identical. Each NFT holds distinct characteristics, making it one-of-a-kind. On the other hand, FT are designed to be interchangeable, allowing for a large supply of identical units. FT can be easily exchanged on a one-to-one basis without any differentiation between individual units [25].

In order for users to perform transactions on the blockchain, they must pay fees. These fees, known as gas fees or transaction fees, serve the same purpose across different blockchains. These fees play a vital role in maintaining the efficiency and integrity of blockchain networks. Fees are necessary to cover the costs associated with transaction execution on the blockchain [26]. By imposing fees for each computation performed, the system prevents malicious actors from flooding the network with spam transactions. Additionally, it sets a limit on the computational steps for each transaction helping to prevent unintended or intentional infinite loops or wasteful computations in the code [26]. Moreover, it functions to compensate participants in the network who expend resources to validate the transactions. During periods of high demand, users may find it necessary to offer a higher fee amount in order to potentially outbid other users' transactions. Offering a higher tip increases the chances of the transaction being included in the next block resulting in improved speed and prioritization.

2.1.4 Smart contracts

Smart contracts are self-executing computer programs stored on the blockchain. The primary purpose of these contracts is to provide a more secure way to produce transactions without having to rely on third parties and question whether the information has been altered for personal benefit. These contracts contain the logic to proceed with transactions, with this logic being defined by programmers. Each contract, when stored in the blockchain, receives a unique address, which is the entry point for the interaction.

To interact with smart contracts, every user needs a virtual wallet, as every interaction with the blockchain requires a fee payment due to processing effort. The hosted blockchain's native cryptocurrency is used to pay this fee. During the interaction with a smart contract, the user sends data to the contract that will be processed based on its logic.

The benefits of smart contracts are speed and efficiency due to automatization processes. Once a transaction is sent to a contract, the execution of the associated request is triggered, and the transaction is performed without errors or any kind of bureaucracy. Associated with automatization, these contracts offer trust and transparency, since there are no third-party entities, and the blockchain encrypts every transaction and shares it across the nodes. Additionally, with a consensus mechanism it is nearly impossible to modify transactions for personal benefit. As a result of their nature, smart contracts help entities save money and time by eliminating intermediaries and the associated delays and expenses.

As the adoption of smart contracts grows within the blockchain ecosystem, extensive research has been conducted to enhance their performance and refine various aspects. Given the programmable nature of these contracts, innovative paradigms have been developed and implemented to achieve higher efficiency and cost-effectiveness.

Currently, the adaptability and reusability of smart contracts is an open research issue. In [27], the author proposed a design pattern to implement the reusability of verification rules across multiple contracts, which allows reconfigurability at runtime. The study was conducted using permissioned blockchains and Java as the programming language. The proposed design pattern exploits polymorphism and manages inheritance through sealed classes and includes two layers: abstract and concrete. The abstract layer includes an abstract contract class with a single abstract method, as well as an interface for rule implementation. The concrete layer contains all the contracts, which are classes that extend the abstract contract class and therefore implement its single abstract method (i.e., the contract behavior). Furthermore, the concrete layer includes the definition of rules, where each rule is encapsulated by a class that implements the rule interface. Although a simple pattern, the proposal in [27] reduces code redundancy and allows configuration at runtime.

In the same context, the paper [28] details a design pattern, called the proxy pattern, that allows smart contracts to be updated without losing data. OpenZeppelin provides an extensible plugin based on this pattern. It comes with extensive documentation explaining its functionality [29]. Unlike a single contract that combines both data and business logic, the proxy pattern involves splitting the contracts into a proxy contract and a business logic contract. The proxy contract acts as a dedicated storage and delegates calls to the business logic contract, while the business logic contract handles all the business logic operations. In the proposed design pattern, the read and write operations performed by the delegated contract have an impact on the storage of the proxy contract.

2.1.5 Meta Transactions

Blockchain technology has made significant progress in the market, prompting many companies to explore its potential for enhancing their business operations. However, despite its increasing adoption, this technology faces challenges when it comes to user onboarding. Each day, numerous transactions are transmitted, and smart contracts are deployed. Yet, users encounter difficulties due to the requirement of paying fees to interact with the blockchain. This issue acts as a potential entry barrier and presents challenges to users who are unfamiliar with the blockchain ecosystem. Specifically, the primary problem lies in user onboarding, which requires native cryptocurrency usage to engage with the contracts or send data, consequently impacting the overall user experience. Individuals who lack familiarity with the blockchain environment must first acquire cryptocurrency in order to utilize blockchain features.

To address this concern, meta-transactions have emerged as a mechanism enabling anyone to interact with a blockchain, regardless of their level of knowledge in this area, while bypassing fee payments. Meta transactions enable users without funds to conduct transactions on the blockchain. Rather than requiring the user to have the necessary funds, this approach allows the user to create a transaction containing the required data and sign it. The signed transaction is then forwarded to another entity responsible for covering the associated fees. By decoupling the user initiating the transaction from the entity responsible for paying the fees, meta-transactions provide a solution that facilitates seamless interaction with the blockchain. This approach involves four actors (Transaction Signer/User, Gas Station/Relayer, Forwarder contract, and Recipient contract) for carrying out a transaction [30]. Figure 2.3 illustrates how these actors interact to achieve the desired outcome.

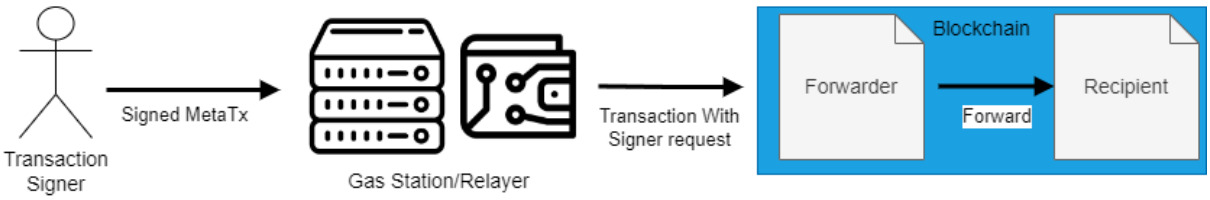


Figure 2.3 - Meta Transactions diagram

1. First, the User, also named Transaction Signer, creates a transaction that contains information about what he would like to execute in the blockchain and afterward signs it off-chain (out of the network). This transaction is equivalent to a usual smart contract transaction, except that it will be signed with the user's private key and additional parameters to "craft" the transaction.
2. Following the transaction request creation and signature phase, the Transaction Signer sends the request to a Gas Relay/Relayer that validates the transactions. The Relayer is a service that possesses its own wallet and is responsible for generating a valid blockchain transaction using the information provided by the Transaction Signer. The Relayer then sends this transaction directly to a contract known as the Trusted Forwarder.
3. The Trusted Forwarder contract has the responsibility of verifying the signature authenticity. It acts as a trusted entity, recognized by the Recipient Contract. If the validation process is successful, the transaction is forwarded to the Recipient contract, which is the target of the Transaction Signer.
4. Finally, the Recipient Contract receives the transaction and performs the action intended by the Transaction Signer.

Meta transactions offer a straightforward yet efficient solution to mitigate blockchain fees, as complete avoidance is impossible. This approach enables companies to onboard users without worrying about their willingness to pay fees or knowledge level, thereby enhancing the overall user experience. By incorporating cryptographic signatures into the data, this design pattern ensures that the entity responsible for transmitting the data to the blockchain cannot manipulate it [31]. Any attempt to tamper with the data would result in a failed verification process within the trusted contract.

2.1.6 Problems of Blockchain

Despite ongoing advancements, blockchain technology still faces several challenges that require attention. These challenges include consensus mechanisms, storage and data management, and chain structures, as well as regulation and governance [32].

Consensus mechanisms, while ensuring the security of blockchains, can be resource-intensive, resulting in lower system throughput and increased latency [32]. As the

network continues to offer free access, an increasing number of nodes will participate in the consensus mechanisms. This influx of participants can potentially lead to further delays in the consensus process.

Scalability and capacity pose technical issues for blockchain networks. As each node must maintain a copy of the current state to back up data, the storage requirements become unrealistic as data continues to grow [32]. Storing an infinite amount of data on a single node is impractical.

Although blockchain has impacted society in some aspects, it has also brought legal challenges. These challenges derive from the lack of adequate legal supervision and the unique features of blockchain. To tackle these issues effectively, it is crucial to understand blockchain's characteristics and establish appropriate laws and regulations. Many countries are actively adopting blockchain technology and working towards improving regulations in various aspects. In the realm of governance, technology sometimes replaces traditional legal solutions, highlighting the interconnection between technology and law. Achieving a balance between technological innovation and supporting legal principles is vital for the positive and sustainable development of blockchain [32].

2.2 Loyalty programs

Loyalty programs are a marketing strategy utilized by companies to foster growth, expand their customer base, and establish a strong market presence. These programs are designed to incentivize customers to remain engaged with the business and facilitate repeat transactions by offering rewards based on customer loyalty. Typically, companies have two primary objectives with these programs: 1) to reward their most loyal customers, cultivating a sense of attachment to the company and encouraging repeat business; 2) to gather customer data and insights in order to enhance their products/services, expand their market share, and attract new clients. The underlying core principle of a loyalty program is to acknowledge and reward customers based on their engagement with the brand. Such programs aim to incentivize customers to make ongoing purchases from a specific business, with rewards being granted upon meeting certain predetermined conditions. Companies adopt this approach due to its minimal effort and cost-effectiveness. Once customers are engaged with the business, it becomes easier to maintain their interest, thereby saving costs on advertising. Additionally, loyal customers can serve as a valuable source for attracting new clients, further contributing to business growth.

Loyalty programs have become a standard practice for companies, with approximately 90% implementing some form of loyalty initiative in 2021 [33]. In 2022, the global loyalty management market reached a value of 5.29 billion USD, with the expectation of 6.47 billion in 2023 [34]. Furthermore, an average American customer belongs to 16.6 loyalty programs [35]. Along with these statistics, these

programs can be shaped and built-in multiple formats/mechanisms, ensuring diversity to please the target audience.

Even though loyalty programs proved to be successful marketing strategies and gained traction in the market, they are now experiencing a decline in effectiveness. Traditional programs are witnessing a customer engagement decrease due to consumerism and an increasingly emotionally detached society. Consequently, many companies are exploring innovative approaches to enhance their programs, aiming to reduce costs while fostering greater user engagement and offering more freedom with rewards. Meanwhile, other companies are transitioning their programs to alternative solutions. The subsequent sections explain several loyalty mechanisms that can be implemented, current challenges of loyalty programs and possible alternatives.

2.2.1 Loyalty programs mechanisms

Companies design loyalty programs according to the target audience and their objectives, adapting them to align with the desired ideology. Despite the immeasurable amount of loyalty programs in the market, traditional loyalty programs currently employ diverse mechanisms, which can be classified into a few major categories: stamp cards, points, tiered, cash back, gamification, and coalition [36]. Every mechanism has its strengths and limitations, and it is up to the company to decide which one best fits their requirements and yields higher returns.

Stamp or Punch cards are considered the oldest and most traditional mechanism used in non-digital loyalty programs. They intend to monitor customer interactions with the brand and provide rewards based on the frequency or quantity of these interactions. For every purchase of a good or service a customer makes, the card receives a punch or stamp. Once the card reaches its designated limit, the customer can exchange it for the predetermined reward. Typically, this type is a default choice by retailers who may not be familiar with digital environments or by more traditional businesses with smaller-scale operations, such as takeaway restaurants, hairdressers/barbers, and grocery stores [37]. While certain companies have adopted this mechanism digitally, the fundamental concept remains unchanged [36]. The advantages of this mechanism include its simplicity in understanding the process and the exact value of the final reward, as well as the avoidance of collecting personal information. On the other hand, the disadvantages are the difficulty in preventing fraud through counterfeit punches/stamps, the limitation of having predefined and uniform rewards for each customer, and the lack of insights into customer interests and data.[37]

Points-based solutions are the most predominant type in digital settings. In essence, this mechanism rewards customers for their transactions. When customers make a purchase, they earn a certain number of points, which can be accumulated and later redeemed for rewards. Recently, companies have been broadening the range of actions that can earn rewards besides purchases of goods and services, aiming to

enhance user engagement. These new possibilities include non-transactional actions like visiting a webpage, downloading an app, and more. Typically, companies opt for this type of mechanism when they have high-frequency purchases and interactions [37]. This loyalty model offers the benefits of attracting new customers efficiently and collecting valuable data for continuous improvement of the business and customer experience. Although it is the most abundant mechanism, it can have a negative impact due to the lack of differentiation compared to other brands. Additionally, despite offering instant points, customers may not always be able to redeem rewards, which can discourage them from experiencing immediate gratification[37].

Tiered loyalty programs are based on ranking the customers by rewarding them with better perks/privileges the more they spend. Similar to the points mechanism, each customer interaction with the brand contributes to the "progress bar" and, upon reaching a milestone, customers receive a promotion to the next tier or rank. These programs foster increased user engagement by offering exclusivity as customers ascend to higher tiers, creating a sense of status. Moreover, while every customer receives rewards, the upgrades in rewards based on rank develop a desire for more among individuals. Individuals on higher ranks are less likely to stop interacting with the brand. Despite favoring the most loyal clients, this mechanism can also affect lower-tier customers. Starting at the lowest tier can induce a daunting sentiment and become an entry barrier along the system's complexity. Furthermore, lower tiers may not provide enough incentive for users to increase their spending. This program can be a double-edged sword because if a user demotes to a lower tier, it can potentially jeopardize the relationship with the brand [38].

Cash back, as the name implies, is mechanism that returns a percentage of the amount spent. Its functioning is similar to a points-based system, where users perform transactions and receive rewards. Unlike other systems, this mechanism can be adapted into three models: 1) Monetary returns - the users get a money percentage back to their wallet/application or a coupon; 2) Non-monetary returns - rather than returning a percentage to the user, the percentage reverts to a charitable cause [36]; 3) Stock returns (unusual) - instead of receiving monetary compensation, users obtain a percentage of market stocks. This type of program can create the perception of products being more affordable as customers can redeem accumulated cash from previous transactions, thereby incentivizing additional sales. In contrast with tiered programs, these programs reward both profitable and unprofitable customers equally and, when facing the market, there is limited differentiation among these programs [38].

Gamification, unlike the previous mechanisms, where users have to make transactions to earn rewards, focuses solely on rewarding customers for their engagement with the business. It integrates various gamification elements into loyalty programs to enhance user engagement without relying on transactions [39]. Instead of requiring users to make purchases, these programs offer tasks or games that users can participate in to earn rewards. This approach makes it more appealing

and interactive for users [40], increasing their time and effort commitment while providing an enjoyable experience and exploring the competitive side. Furthermore, it distinguishes itself from competitors who rely on more conventional mechanisms [40]. The drawback lies in the potential narrowing of the target audience due to the specific interest in gamification elements.

Coalition programs involve multiple companies collaborating to enhance user engagement. This mechanism enables companies to swiftly expand their customer base while sharing operational costs among themselves. Despite customers earning rewards more rapidly and having a vaster selection, these coalition programs do not foster loyalty towards individual companies but rather towards the loyalty program itself. Retailers face challenges in differentiating their loyalty programs. Additionally, there is a risk of customers earning rewards from one brand and redeeming them with a competing brand. As a consequence, this type of program does not provide any competitive advantage in the marketplace [38].

The mechanisms stated above are just the main ones. Many companies employ a combination of multiple mechanisms to leverage the strengths of each to create loyalty programs with fewer shortcomings, aiming to retain a greater number of customers. Regardless of the type chosen, loyalty programs share a common objective of enhancing user engagement, ultimately leading to an increase in the brand's market share.

With the emergence and integration of blockchain technology, a new mechanism based on non-fungible tokens (NFT) has emerged. Companies create NFT and reward customers with them, effectively making customers owners of these unique digital assets. These NFT can include various benefits that the owner can choose to utilize or sell at a price. This new mechanism will be explained in Section 3.2.1.

2.2.2 Traditional Loyalty Programs

Loyalty programs, although prevalent in the marketing field for enhancing user retention, are experiencing a decline in engagement. In 2021, the average American belonged to 16.7 loyalty programs, but this number decreased in 2022 [35]. Despite customers participating in multiple programs, only 7.6 memberships remain active. This indicates that less than half of the programs that customers join retain their relevance and activity. In 2019, Harvard Business Review Analytic Services published "Beyond rewards: Raising the bar on customer loyalty" [41], a research based on a survey of 400 executives globally. The study revealed that while 55% of the executives confirmed they had updated their loyalty strategy, only 42% believed that their organization's approach to customer loyalty was effective. These findings indicate that despite investing in loyalty programs, their effectiveness is declining.

These programs have related problems that can be characterized into two perspectives: customer and company [36].

Starting with the customer perspective, the main current difficulties are as follows:

- The first step in joining a new loyalty program is usually to provide the system with personal information, which nowadays, with unquestionable security concerns, may imply a constraint for some users [42]. Additionally, customers feel that they belong to too many loyalty programs, since each brand has its own program [43];
- With traditional approaches, the customer does not own assets such as points, tiers, and others. Instead, the program controls and stores the assets inside its database. Additionally, the value of the assets is changeable due to the company having the power to decide when to inflate or deflate it. These programs naturally have limits due to the way they are implemented. An example is that rewards only have value within the scope of a specific application and cannot be transferred to another system [42];
- The customer experience of traditional loyalty programs is not satisfactory. Despite their popularity, customers are less motivated to continue interacting with the system due to the effort–reward relationship [44]. Most programs set an expiration date for rewards, which indirectly requires the customers to spend them before the deadline, even if they do not intend to. Another factor that reduces the customer experience is the lack of customization in the rewards for customers, with their needs and desires not usually considered (each customer is different, but compensation is the same for all).

From the perspective of companies, the main difficulties consist of the following:

- Difficulty in measuring the return on investment (ROI) and its impact on net income arises from the unpredictable redemption of rewards and incomplete data from loyalty programs, creating costly liabilities on the firm's financial statements [36];
- Companies are responsible for protecting all the information of customers. Moreover, to process the transactions, they rely on third party entities, which makes them very costly. Apart from the need for intermediaries, a company is also responsible for infrastructure management, which adds overwhelming complexity and increases costs;
- In addition to the above-mentioned statement, companies are also tasked with personalizing loyalty programs to meet the unique needs of individual customers [44]. This poses a significant challenge as each customer has distinct motivations and preferences, making broad approaches ineffective. Moreover, customers are inherently brand-conscious, often prioritizing their preferred brand over the rewards offered by loyalty programs [36];
- Building partnerships with the traditional approach is complex due to the nature of the programs. Usually, brands conceive loyalty programs with the intention of expanding their own business and, therefore, these programs

become restricted to their original scope. Whenever a company decides to build a partnership with another company, the typical solution consists of creating a new loyalty program. Consequently, another issue arises: who owns the assets (since with the traditional approach, the customers do not).

2.2.3 Loyalty Programs alternatives

Loyalty programs serve as a marketing strategy employed by companies to grow their customer base and foster customer retention. While they are the most commonly used approach, there are a few alternative strategies that aim to achieve the same objectives. In this section, we specifically discuss referral programs and membership programs.

Starting with referral programs, both referral and loyalty programs share the common aspect of rewarding customers for specific behavioral actions. However, the similarity ends there, as the way rewards are earned differs between these two approaches. In loyalty programs, rewards are obtained through continuous customer purchases and interactions, whereas referral programs reward customers for recommending a business to others. In a referral program, customers are assigned a unique code or URL, which they promote, and receive rewards for every registration or purchase made using their code or URL [45]. Companies opt for this format because referrals tend to be more effective in advertising. The target audience typically places more trust in recommendations from friends, family, or influencers compared to traditional advertisements. Customers who already have a positive impression of a product are likely to share their personal recommendations, resulting in new referrals already having a favorable perception of the business by the time they interact [45]. An illustration of this approach is the referral system implemented by Dropbox [46]. Through this system, users can invite their friends using a unique URL and receive additional storage when their friends register and subscribe to a plan.

Membership programs are another highly effective marketing strategy for rewarding and retaining customers. In contrast to loyalty programs where customers have to earn their rewards, membership programs offer immediate access to perks and rewards. Customers are required to pay a fee, either through a subscription or one-time payment, to join these programs, which grants them instant benefits [47]. Unlike free-to-join loyalty programs, membership programs often provide a broader range of exclusive perks. Due to their paid nature, membership programs can afford to offer more valuable rewards to customers who join. Additionally, membership programs generate direct income for the business, leading to a more enjoyable, convenient, and valuable customer experience, setting them apart from competitors. A prime example of this is Amazon Prime [48]. Customers pay a monthly or yearly subscription fee and gain access to a wide range of benefits. While some may categorize Amazon Prime as a loyalty program with fees, former Vice President Greg Greeley stated: "Amazon Prime is not a loyalty program. Because it's a great

experience, people who use it become loyal." [49]. This highlights that Amazon Prime goes beyond a traditional loyalty program by providing a remarkable experience that fosters customer loyalty.

2.3 Loyalty Programs Complemented with Blockchain

Blockchain integration will reshape the traditional loyalty programs' work and add some benefits. First, companies will not need to spend more money on infrastructures to manage the assets and the system. Blockchain will manage and be responsible for the infrastructure as necessary, and companies can automate proceedings with smart contracts to establish rules for rewards. Furthermore, with blockchain integration, third parties are removed due to the characteristics of decentralization, disintermediation, and consensus, leading to financial savings. Data security is now guaranteed by blockchain mechanisms, leaving the company less burdened with private data protection, and preventing fraudulent and forging transactions. It will also become simpler to create partnerships with other companies due to the shared ecosystem (blockchain) in which there are no restrictions, as any asset is valid. Companies' coalitions can help the participants to increase their business reachability for new customer networks.

For end users/customers, there are also benefits. The first is the elimination of any physical support for managing rewards, such as stamp/punch cards. Since these loyalty programs with blockchain integration become digital, companies can leverage the ubiquity of mobile platforms making the process more convenient and efficient. Secondly, users who have concerns about their data privacy and security do not need to fear thanks to blockchain security measures. Besides, the users merely need a virtual wallet to interact with these systems, which does not require any introduction of personal information, and to get set to gain rewards. The virtual wallet is universally compatible and can be utilized across various loyalty programs, serving as a centralized storage point for all accumulated rewards, making it easier for customers to track progress. In addition, through partnerships established between companies, customers gain access to a broader range of businesses. This collaborative approach also facilitates the seamless sharing of assets such as points and NFT, among others, enabling faster redemption and greatly improving the overall customer experience and gratification time. Finally, the users are the owners of the assets, which is not the case in traditional systems where companies can inflate/deflate the value of points. Here users can freely share the assets with other users.

3 STATE OF THE ART

This chapter provides an overview of the current state of the art in the field of blockchain associated with loyalty programs. It is divided into two sections: "Related Work," which focuses on scientific research in the area, and "Current Implementations in the Market", which highlights existing practical applications.

3.1 Related work

This section presents an analysis of the current state of the art regarding the use of blockchain in various domains and contexts associated with loyalty programs.

In [50], the authors propose and describe an implemented loyalty program that is based on blockchain. The authors identified issues related to the mechanisms of customer dissatisfaction and loyalty. Afterward, they explain the solution they propose, which consists of a smart contract created in the Ethereum network [16] and capable of producing tokens. These tokens are called TECH tokens (the name given by the authors) and are similar to cryptocurrencies. TECH tokens are stored in Ethereum wallets and can be transferred between wallets to perform payments or exchange goods in the system. Since the Ethereum network requires a fee for each transaction and the cost is expressed in the ether cryptocurrency, the authors implemented and tested the system on Rinkeby [51], an Ethereum-compatible test network. The implementation consisted of three distinct phases. First, the smart contract is written and deployed on the blockchain. Then, the token is created according to ERC20 standards[52] and, finally, the frontend is developed in React.js [53] for user interaction. The system has integrated Web3.js [54] as a way to allow the frontend to communicate with the blockchain.

The authors in [42] propose a universal loyalty platform. Before presenting the solution, they detail how blockchain can improve a loyalty program from both the point of view of customers and companies. The proposed solution consists of an application in which customers can use points to redeem rewards and a website for companies to interact with and manage their business. The system uses React Native [55] and React [53] for mobile and website frontend, respectively. The system also uses Stellar [56] and Hyperledger Fabric [17] blockchains to manage users' information and assets. At the customer level, upon registration, a Stellar account is created. This account allows the user to trade any asset type, being able to exchange points between loyalty programs. At the company level, upon registration, a Stellar account is created. During the registration process, the company can choose other companies to collaborate with. Once this entire process is over, the smart contract allocated in the Hyperledger blockchain is invoked to create and connect the inherent token to that company and link it to the Stellar account.

Two studies from 2019 [57], [58] aimed to explore how blockchain can impact loyalty programs and understand their relation. Both studies presented four theoretical foundations: (1) self-determination theory (SDT) [59]; (2) customer perceived value; (3) loyalty programs' design in loyalty program efficiency; and (4) blockchain application. The authors use SDT to define four dimensions of customer motivation: economy, autonomy, competence, and relatedness. To better understand the link between SDT and customer perceived value, the authors assess customers' perceived value in three domains that they proposed: economic utility, psychological self-fulfillment, and social interaction. For the last two theoretical foundations, the authors focus more on loyalty programs, specifically on design effectiveness and blockchain infrastructure. Both papers chose BubiChain [60] as the target blockchain infrastructure, due to its success in the loyalty program market, to undertake the evaluation. They also chose an exploratory case study on a loyalty points-based system. Although the works are analogous, the depth of the study differs between them.

The authors of the first paper [57] only assess the main characteristics of blockchain integration and its impact on each SDT domain. The second paper [58] evaluates the BubiChain infrastructure itself and how it works, the main elements of the loyalty points-based system, how the code is structured, and, finally, the impact of BubiChain on participation in loyalty programs. While the exploratory case with blockchain has proven to be more effective than existing traditional schemes to satisfy customers, both papers state that more future research is needed to explore the relationship between blockchain design for loyalty programs and customer behaviors.

The work described in [44] aims to review the potential and challenges of blockchain in marketing areas for building brand loyalty. The authors first describe blockchain technology and its unique features. Then, an overview of the characteristics of brand loyalty and current failures is presented. The authors of the paper also explain how integration with blockchain infrastructure can eventually solve and improve the quality of loyalty programs, making them more effective and affordable.

The authors in [61] propose an implementation of a loyalty program for a mobile platform based on the Waves blockchain [62]. The choice for this blockchain platform is due to its lightweight and fast features when compared to Ethereum and to the premise that the cell phone is an essential commodity today. The proposed solution consists of three key components: a token creator in Waves, a web API, and a database to store accounts' information.

In [43], the authors give a simple introduction to blockchain and aim to analyze its impact on the retail industry and customer loyalty. Although the authors focus both on the supply chain and loyalty areas, their emphasis is on the need for a loyalty program as the best solution to retain customers. According to them, customers remain dissatisfied even though numerous companies are now investing in loyalty programs. To support this claim, the authors provide statistics from a survey in

which they list several problems pointed out by the respondents. They also suggest that blockchain integration could potentially address these issues and provide a lower cost system with better security and transparency.

In [63], the author studies the importance of blockchain technology in enhancing loyalty programs. The scope of this paper is restricted to the airline business. Regardless of the study domain, the author explains how blockchain can impact and revolutionize a company's experience. Integrating blockchain technology from a business perspective could reduce maintenance costs and enhance customer service. Blockchain can foster better customer relationships and captivate customers by providing the benefits of traditional loyalty programs and better customer service. From the point of view of the transaction process, blockchain also makes all the validation, storage, and sharing of transactions easier, since intermediaries would no longer be needed. For better context, the author explains the five basic principles of blockchain. This study consists of a survey that was carried out using two sources of information: the primary source, using 450 questionnaires, and the secondary source, analyzing the literature. The 450 questionnaires were distributed among customers and airline managers, and a high-efficiency response rate was achieved. The use of blockchain technology was emphasized by the surveyed airlines. They concluded that the degree of effective use of customer loyalty programs is related to blockchain technology, and the results showed that loyalty systems were effectively improved by blockchain integration.

In [64], the authors propose a blockchain-based framework to enhance loyalty programs in the fast-moving consumer goods (FMCG) industry. This framework aims to revolutionize the conventional process of exchanging coupons by introducing a token called promotion asset exchange (PAX). Instead of customers purchasing products and retaining coupons for later trading with merchants, followed by the cumbersome process of merchants trading numerous coupons with the manufacturing company, which can lead to counting errors and result in confusion, customers will scan a QR code on the product packaging and the promotion (PAX token) will then be stored in their wallet. Once the customer has accumulated enough PAX tokens, they can be traded with the merchant, which will trigger a transfer to the merchant's wallet. To receive payment, the merchant can return the PAX token to the manufacturing company, which will refund it to the customer based on the number of tokens. The authors emphasize that blockchain technology enables the secure tracking of all transactions, enabling manufacturing companies to better understand customer needs while ensuring no coupons or payments are lost due to human error.

The authors of [65] propose a blockchain-based platform for coalition loyalty program management. The primary objective of the platform is to provide customers with the capability to exchange their points across distinct existing blockchain loyalty programs. The system consists of different companies that operate their loyalty programs, customers who accumulate points through these programs, and a platform that facilitates token exchange. The token exchange

platform incorporates smart contracts that define the exchange rates between the tokens of each company. To ease this process, the companies are required to negotiate the rates and establish smart contracts with the rates. To enable interoperability, the authors utilize sidechain technology for conducting transactions across different blockchains. In order to encourage customer engagement, the authors allow customers to participate in the blockchain consensus protocol and receive rewards in return. The authors have selected the proof of stake (PoS) protocol due to its low computational requirements, minimal delay, and potential to attract more customers. The proposed system ensures that customers can effortlessly exchange their points with other companies, and companies can easily join the coalition/partnership without the need for a system overhaul.

The authors of [66] propose a blockchain-based platform that facilitates the exchange of loyalty points among customers that is similar to the work described in [65]. The proposed system leverages a many-to-many matching method based on the call auction approach used in the stock exchange market, thereby eliminating the need for exchange rate negotiations. The call auction method increases the likelihood of orders being executed, prevents points from going unused, and incentivizes customers to accumulate more points. In contrast to the system proposed in [65], in which companies are responsible for handling rate negotiations and defining the corresponding smart contracts, this system [66] uses the call auction method to simplify the points exchange process, making it more efficient and eliminating the need for partnerships and negotiations. Both systems aim to enhance the current blockchain-based loyalty programs and improve customer experience.

A study on the impact of a blockchain-based loyalty program on customer loyalty in the renewable energy sector was conducted by the authors in [67]. More specifically, this paper addresses the problem of green electricity tariffs (GET), where companies often fail to generate the promised amount of renewable energy, leading to dissatisfaction, mistrust, and customers considering switching to competitors. First, the authors present a theoretical study assessing how blockchain can increase institutional trust and decrease institutional distrust. They then propose a loyalty program that allows users to track electricity generation data, monitor their electricity consumption, and optimize their consumption by setting rules for their smart devices. The proposed loyalty program will provide customers with tokens that can be redeemed to reduce the price of GET, donate to charity, use other utility services such as electric scooters or car sharing, or reinvest in renewable energy shares. While affordability is a system requirement in [67], it is not clear whether customers are responsible for paying blockchain fees, which may be impractical due to high transaction prices on the Ethereum network. This problem could be solved by introducing meta transactions. In addition, the smart contract architecture of the proposed system is not demonstrated, and it is unclear whether the system utilizes multiple contracts or any interface for token creation. Although the authors have achieved the desired outcome, the results are specific to the renewable energy sector.

Furthermore, the proposal in [67] is supported by a qualitative analysis that includes interviews and a literature review but lacks quantitative analysis.

Although there are already some proposals for blockchain integration, such as [42], [50], [61], [64], they present a poor user experience with respect to blockchain onboarding, mainly due to the need for user knowledge, the authentication process, the input of personal information, and the lack of meta transactions. Moreover, they contradict the premise of blockchain decentralization using centralized database systems. Systems such as [65], [66] propose innovative concepts and solutions to address the challenges faced by existing loyalty programs that rely on blockchain technology. All these systems use blockchain to manage the loyalty programs' infrastructure.

3.2 Solutions in market

Currently, in the market, there are a few platforms that use blockchain to address inefficiencies in traditional loyalty programs. Despite companies investing in the loyalty field and blockchain being widely recognized as a means to revamp these systems, companies often leverage these networks to create their own independent loyalty programs. Conversely, loyalty management systems that utilize blockchain as their core technology remain limited in number. In this sub-chapter, we will discuss the current solutions in loyalty programs and management systems with blockchain integration. The objective of this study is to understand the specific ways in which companies leverage blockchain technology and gain insights into the key features that hold the greatest significance in this context. Additionally, it aims to identify any existing flaws or limitations that can be overcome in our system. To address these goals, we will first focus on loyalty programs as they serve as a fundamental component in this context before delving into loyalty management systems.

3.2.1 Blockchain-based loyalty programs

KrisFlyer [68] is the loyalty program offered by Singapore Airlines and is recognized as a pioneering initiative in the implementation of blockchain technology. This loyalty program is highly favored by frequent fliers, not only for its straightforward approach to earning miles (points) but also for the extensive range of rewards available for redemption. To commence earning KrisFlyer miles, a registration process is necessary. Once an account is created, KrisFlyer offers multiple avenues for accumulation. The primary method involves flying with Singapore Airlines or its partner airlines, where points are awarded based on the distance traveled and the class of the flight. Another means is through credit card expenditure, as Singapore Airlines provides a debit or credit card that converts spending into miles at a predetermined rate. Additionally, KrisFlyer supports the transfer of points from various bank programs, allowing for conversion into miles. Moreover, Singapore Airlines has established a broad network of partners, including hotels, car rentals,

and other services, enabling members to accumulate KrisFlyer miles through these collaborations. In terms of utilizing these miles, customers have the option to book flights, purchase products and services, or apply them towards partial payment of a bill, among other possibilities. This program is renowned for its extensive scope and comprehensive nature, owing to its partnerships with various businesses. It is worth noting that KrisFlyer operates on a private blockchain exclusively owned by Singapore Airlines [69].

Starbucks Odyssey [70], a leading coffee business, is revamping its loyalty program (Starbucks Rewards [71]) to integrate the latest web3 technologies and provide a more engaging user experience. Starbucks Odyssey [72], the new extension for the loyalty program, gives a greater experience allowing members to participate in a series of entertaining, interactive activities called 'Journeys". Once a Journey is complete, members will earn collectible 'Journey Stamps' (i.e., NFT) and Odyssey Points that will open access to new benefits and immersive coffee experiences [73]. Customers. as they complete more Journeys and store more points, will reach a higher rank giving them higher benefits and rewards. To facilitate the creation and management of NFT assets, Starbucks Odyssey leverages the Polygon PoS public blockchain [73]. Additionally, customers will have the option to buy and sell these NFT in an integrated market, granting them greater control over their assets and enhancing the overall experience. This revamped loyalty program seamlessly combines the traditional mechanisms of points and tiers, empowering users to achieve more rewarding outcomes. Furthermore, by incorporating NFT, Starbucks Odyssey delivers unique and personalized experiences that instill a sense of exclusivity among customers, making them feel truly special [73].

The two loyalty programs mentioned above, despite their significant scale, provide valuable insights into the potential applications of blockchain technology and the emergence of innovative mechanisms. Starbucks, for instance, leverages the intrinsic capabilities of blockchain through the utilization of NFT, thereby enhancing the customer experience by offering unique and exclusive rewards. Importantly, these rewards are fully owned by the customers, granting them the freedom to decide whether to retain or sell them. Meanwhile, Singapore Airlines utilizes a universal token for the whole ecosystem management.

While both companies operate within vast business markets, they have opted for distinct types of blockchain technology. Singapore Airlines has chosen to employ a private blockchain, which allows for a controlled ecosystem and improved transaction speeds. On the other hand, Starbucks utilizes the Polygon PoS public chain for NFT generation and management. This demonstrates that the choice of blockchain technology, regardless of business size or industry, can serve as a means to enhance loyalty programs effectively.

Overall, these loyalty programs showcase the potential of blockchain in revolutionizing customer engagement and loyalty, presenting opportunities for businesses to create more immersive and rewarding experiences.

Currently, various brands are exploring the integration of blockchain technology into their loyalty programs. For instance, American Express, BMW, and Christian Dior are among those researching and developing innovative approaches [74]. Christian Dior, specifically, is concentrating on implementing a multi-tier loyalty program in collaboration with its Beauty division. Another brand, Rebag, has recently introduced new features called "Premium Payouts" and "Spending Bonus" aimed at enhancing brand loyalty by providing substantial rewards to clients who remain engaged with the Rebag ecosystem over the long term and reinvest their payouts.

3.2.2 Blockchain-based loyalty management systems

Loyyal [75] is a renowned loyalty management system that has gained significant popularity. It leverages blockchain technology and smart contracts, making it a universal platform. Notably, esteemed partners like Deloitte and IBM have joined this project [75]. Loyyal supports various business types, such as airlines, banks, and commerce entities. Their primary objective is to unify the fragmented rewards programs industry into a single ecosystem. Each business can establish and customize its own program based on specific requirements. This platform allows for multi-branding of rewards, enabling users to redeem rewards at any participating company within the system [76].

Loyyal utilizes IBM's Hyperledger Fabric, a private blockchain, to greatly reduce the costs associated with establishing and operating a loyalty program. They employ Hyperledger Fabric's "private channels" to ensure secure and private transaction sharing between companies. Within the loyalty platform, smart contracts are used to define loyalty coins as assets. However, this approach requires at least one bank on the network to facilitate the conversion of fiat currencies to loyalty points and vice versa. The limitation of this technique is that loyalty points are confined to the network, and involving a bank becomes necessary for conversion [42]. Given the platform's success and scale, they impose a substantial price for their services, ranging from $5,000 to $15,000 plus.

Qiibee [77] is a leading blockchain-based loyalty marketplace that revolutionizes the rewards economy by creating a decentralized ecosystem where all rewards can be exchanged. They provide a standard Plug & Play platform where businesses can create or integrate their loyalty programs that suit their needs without any tech knowledge. Qiibee blockchain infrastructure is a hybrid solution with two chain types coexisting: 1) Public Ethereum Blockchain; 2) Private Ethereum-based Blockchains. Qiibee for a healthy and sustainable environment bases on "Qiibee's LoyaltyToken Protocol" [78]. This protocol provides a trustless, secure, and provable manner for issuing loyalty tokens backed by a crypto asset. The LoyaltyToken Protocol comprises four essential components: a blockchain hosting a crypto asset that provides real value to loyalty tokens, multiple blockchains housing the loyalty tokens, a set of loyalty contracts governing the behavior and exchange of these tokens, and a cross-chain bridge facilitating secure communication and value

transfer across the blockchains. One of the primary responsibilities of the protocol is to enable communication between the chains. This blockchain infrastructure, along with the QBX token (Qiibee's token), ensures the loyalty tokens possess genuine value. The flow overview of this protocol is illustrated in Figure 3.1.

Here is a breakdown of the protocol flow [78]:

1. The brand establishes a budget for rewards and purchases QBX on the public Ethereum chain;
2. The brand transfers the acquired QBX to the vault;
3. Loyalty tokens are created based on the number of QBX sent;
4. Qiibee transfers the loyalty tokens to the brand's wallet;
5. Brands then distribute the loyalty tokens as rewards to customers;
6. Customers have the option to redeem or exchange their loyalty tokens for QBX.

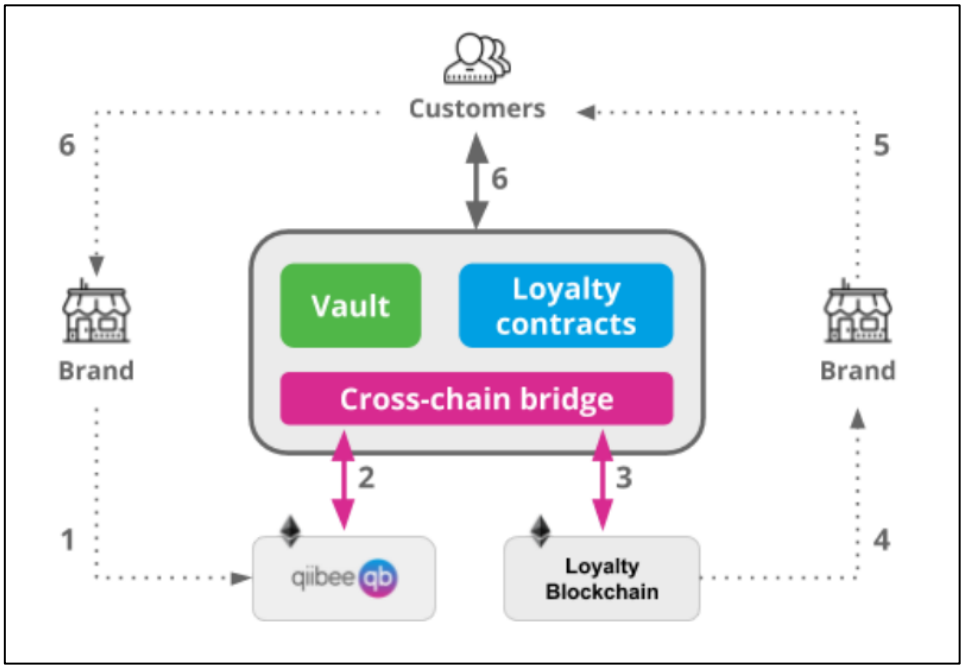


Figure 3.1 - Qiibee protocol flow [78]

By integrating Qiibee's token (QBX), loyalty tokens gain tangible value, allowing them to be converted into real money. Joining the Qiibee ecosystem enables companies to benefit from a network of partners. This fosters the exchange of loyalty points, miles, cryptocurrencies, NFT, and other rewards among participating companies and customers. As a result, engagement and brand loyalty are enhanced. Also, to enhance the process of integration the platform Qiibee developed a service to create wallets for their users. The platform stores wallet information locally on

the user's device browser. However, if the users switch to a different device or clear their browser data, they will need to import the wallet mnemonic phrase. Failure to do so will result in the loss of access to the wallet and funds. One issue related to the business model consists in brands having to define a reward budget, which can difficult the rewards distribution in case of many customers. If a brand bought rewards assuming a number of customers, the entrance of new clients can lead to a rewards shortage.

Loopy Loyalty [79] is a platform designed for small businesses. Its mission is to deliver a digital experience with low effort. Its specialization is the traditional physical stamp card transformation to a digital card. This platform allows any business to create and customize its own card, establishing a brand-new loyalty program within minutes. Even though Loopy Loyalty manages several loyalty programs, it restricts the mechanism to stamp cards, so a business can only choose this type. Also, due to loyalty programs' simplicity, the rewards exchange and redemption require an in-person process where individuals must scan a QR code. Regarding blockchain technology, there is limited available information about the specific blockchain used by the platform and its infrastructure. However, it is mentioned that it employs blockchain technology to record every transaction [80]. Considering the target audience, Loopy Loyalty offers pricing plans ranging from 25$ to 95$ per month, keeping affordability in mind for small businesses.

Appsolutely [81] is a loyalty platform that recently debuted in the Philippine market intending to centralize loyalty programs through blockchain integration. It introduced LoyalCoin, a universal loyalty cryptocurrency that serves as its digital token [36]. LoyalCoin can be earned by making purchases from participating brands within the platform. This approach minimizes the redundancy of multiple loyalty rewards, allowing users to accumulate rewards more quickly and providing the flexibility to exchange them for brand-specific rewards or other digital assets. Additionally, users have the freedom to send the tokens to other users. Specific details about the blockchain and loyalty mechanism used by Appsolutely are not available, except a statement in a report [82] that mentions Appsolutely will use the NEM blockchain [83] to scale transaction rate.

Joyn [84] is a loyalty program management platform based in Belgium. In contrast to the aforementioned platforms, this company does not use blockchain and leverages both digital and physical environments to enhance user experience. It offers brands the opportunity to create their own loyalty programs. However, these programs have limitations in terms of mechanisms and the sharing of rewards with other companies. In terms of the loyalty mechanism employed, Joyn utilizes a combination of a loyalty card and a points system. However, retailers have the option to forgo physical cards and instead focus on digital engagement, prompting users to install Joyn's app. This simplifies the process, alleviating the need for customers to carry physical cards. Additionally, the digital card can be used across various merchants. However, it is important to note that points earned at different

merchants do not accumulate, as each merchant maintains his or her own separate points system.

All the aforementioned solutions, except the last one, are blockchain-powered platforms that specialize in managing loyalty programs. Joyn has been included due to its similarity with the internship goals approach and its focus on smaller businesses. Upon analysis, it becomes evident that these platforms predominantly leverage blockchain technology to facilitate a seamless exchange of rewards across various platforms. Moreover, they embrace digital environments, with a strong emphasis on mobile devices. This highlights the increasing preference among customers to utilize their mobile devices, as technology becomes more ingrained in our daily lives.

Loopy Loyalty and Joyn are more visioned for small businesses, offering simplicity in creating and velocity in implementing operational loyalty programs. The others, despite also being simple, require more information and setup. Also, both systems use location-based services to give a better user experience.

From this analysis, the main features expected to be exposed by blockchain-based loyalty management systems are:

- Registration/Login;
- Simplicity in creating and making operational loyalty programs;
- Location Services;
- Leverage Mobile devices;
- Users must be able to do whatever they want with rewards (redeem/trade);
- Blockchain security as a reliable and transparent way to register transactions;
- Merchants having a business overview;
- Merchants paid subscriptions;
- Unification of loyalty programs as coalitions (merchants' tokens can be used in other merchants or use a universal token for the system).

Loopy Loyalty and Joyn served as great inspiration for our system design and features, due to their similarities with what we intended to achieve.

4 TECHNOLOGIES AND TOOLS

This chapter provides an overview of the technologies and tools employed in the development of the blockchain-based loyalty management system prototype we developed, along with the motives behind their selection.

4.1 Technologies

This section presents the technologies and environments used in developing the system prototype, as well as the reasons for the choice. Table 4-1 summarizes them.

Table 4-1 - Technologies used in prototype development

Name	Description	Usage
Android[85]	Mobile operating system	Mobile operating system the app was developed for
Kotlin[86]	Programming language for multiple platforms	Used to develop the source code of the mobile application
Solidity[87]	Programming language to write smart contracts	Used to write the source code of the smart contracts
JavaScript[88]	Interpreted programming language for multiple platforms	Used to write scripts and tests for smart contracts
Polygon PoS[89]	Blockchain network that uses Proof-of-Stake consensus mechanism	Blockchain used to host and execute the smart contracts

4.1.1 Mobile Operative System: Android

The mobile operative system on which the system was built is Android [85]. Android is a prominent mobile operating system that boasts several noteworthy characteristics. It stands out for being an open-source platform enabling developers to modify the source code and tailor the operating system to their requirements. This flexibility allows for the creation of unique and customized user experiences, encouraging innovation and differentiation among Android devices.

Android has firmly established itself as the leading mobile operating system worldwide. Its extensive presence is evident across a wide range of devices, including smartphones, tablets, smartwatches, smart TVs, and even automotive systems [85]. This broad compatibility has contributed to its widespread adoption and solidified

its position as a dominant force in the mobile industry. A significant aspect of Android's success can be attributed to Google's role as the main contributor to the platform. Google offers Google Play Store service, which serves as the primary app distribution platform, providing users with access to an extensive library of applications, games, and services.

In summary, Android's open-source nature, dominant market presence, broad device support, Google's contributions, and customization options collectively contribute to its widespread popularity and success in the mobile operating system landscape.

The main reason behind the choice of this mobile operating system was the background knowledge of the intern, and the company openness to any choice. Besides Android, one alternative considered was iOS, the mobile operating system for Apple phones.

4.1.2 Kotlin

Kotlin is a modern, statically typed programming language designed to simplify developers' work across multiple platforms [86]. It serves as a versatile tool for creating software on various platforms, including servers, Android, iOS, desktop, and web applications.

In 2019, Google made the significant decision to designate Kotlin as the preferred language for Android development, replacing Java [90]. Kotlin brings several advantages, addressing some of the challenges faced with Java and enhancing the overall developer experience. Notably, Kotlin allows developers to write more concise and precise code, resulting in improved code readability compared to Java.

Another notable feature of Kotlin is its seamless interoperability with Java. Developers can effortlessly integrate Java code, libraries, or frameworks into Kotlin projects, and vice versa, within the same project. This compatibility allows for a smooth transition and utilization of existing Java resources alongside Kotlin.

Kotlin also introduces null safety, providing a safer coding environment. By default, variables in Kotlin cannot hold null values, reducing the occurrence of null pointer exceptions and enhancing code reliability.

Jetpack Compose, the modern UI (User Interface) toolkit introduced by Google, is built on Kotlin, showcasing the language's relevance and adoption within the Android development ecosystem. With Kotlin, developers can leverage Jetpack Compose's capabilities to create intuitive and visually appealing user interfaces.

Furthermore, Kotlin offers structured concurrency, enabling streamlined asynchronous programming through the use of coroutines. This feature simplifies complex tasks such as network or database calls, making asynchronous operations more manageable and less error prone.

In summary, Kotlin stands as a modern, cross-platform programming language that enhances developers' productivity. With its concise syntax, seamless interoperability with Java, null safety, support for Jetpack Compose, and structured concurrency, Kotlin provides a robust foundation for creating robust and efficient software across multiple platforms.

4.1.3 Solidity

Solidity [87] is a powerful object-oriented, high-level programming language specifically designed for creating smart contracts that operate on Ethereum Virtual Machine (EVM) compatible chains. First introduced in 2014, it has become the most widely used language for developing smart contracts on the Ethereum blockchain. Solidity draws significant inspiration from C++, incorporating elements from other languages like Python and JavaScript to enhance its versatility and ease of use.

One of the key strengths of Solidity is its robustness and expressive syntax that allows developers to implement various features, including inheritance, libraries, and custom data structures, stimulating more scalable and reusable smart contracts. It also supports modifiers, which help in code reusability by allowing the definition of common functionalities that can be applied to multiple functions within a contract.

However, being a relatively recent language, Solidity is continually evolving, and developers need to be aware of updates and best practices to ensure the security and reliability of their smart contracts.

As blockchain technology and smart contracts continue to revolutionize various industries, Solidity's role remains instrumental in enabling the creation of decentralized applications and facilitating the transfer of digital assets in a secure and trustless manner on the Ethereum blockchain and other EVM-compatible chains.

The selection of this language was based on the compatibility of the chosen network, which is EVM compatible.

4.1.4 JavaScript

JavaScript [88] is a lightweight interpreted programming language widely used and primarily known for its versatility and widespread adoption across various domains. As a dynamic and high-level language, JavaScript is predominantly used for front-end web development, enabling the creation of interactive and user-friendly web applications. Additionally, it is also utilized for back-end development with the help of Node.js or other tools, allowing developers to build scalable and efficient server-side applications.

One of the key advantages of JavaScript lies in being supported by all major web browsers without the need for additional installations. Moreover, JavaScript boasts a vast ecosystem of libraries and frameworks, such as React, Angular, and Vue.js, which further streamline the development process and enable developers to build

feature-rich and responsive web applications. Its event-driven and asynchronous nature also contributes to enhanced user experiences, as it allows for seamless handling of user interactions without disrupting the overall functionality of the application.

The reason behind the use of JavaScript in our prototype lies in its capability to leverage testing tools for smart contracts and its seamless interaction with web3 technologies.

4.1.5 Polygon PoS

Polygon PoS [89] is a scaling solution offered by the Polygon Lab platform, formerly known as Matic. It functions as an EVM-enabled Sidechain, utilizing the proof of stake consensus mechanism. Being one of the most widely used protocols globally, it boasts thousands of DApps and an average of over 3 million daily transactions. Polygon PoS offers numerous advantages, including lower transaction costs, higher transaction speeds, extensive developer documentation for easy integration, and EVM compatibility, which facilitates the use of Ethereum tools and migration to the Polygon environment. Due to its inherent usage of Ethereum security protocols, Polygon PoS can also be considered a Layer 2 solution. As a result, major companies and projects have embraced it.

The Polygon PoS architecture comprises three layers [91] (Figure 4.1):

1. Ethereum Layer: consisting of contracts on the Ethereum network;
2. Heimdall Layer: comprising proof of stake Heimdall nodes that operate parallel to the Ethereum mainnet. They monitor the staking contracts on Ethereum and commit Polygon Network checkpoints to the mainnet;
3. Bor Layer: comprising block producing Bor nodes, shuffled by Heimdall nodes.

Unlike most side chains that limit the number of validators, Polygon PoS allows anyone to become a validator on the network. Validators are required to stake Matic (Polygon Network's native cryptocurrency, similar to Ethereum) that are stored on the Ethereum main chain and run a full node (Heimdall and Bor layers).

Heimdall [92] is the proof of stake validation layer responsible for aggregating blocks produced by Bor and works in conjunction with the contracts deployed on the Ethereum network. One of its primary duties is managing validators, which involves coordinating, updating, and rewarding them. Heimdall facilitates communication with the contracts in the Ethereum network to enable the entry of new validator stakes, updates, and rewards. These rewards are based on their contributions to securing transactions. By adopting this approach, the system ensures that it doesn't solely rely on validator honesty but also leverages Ethereum's security mechanisms as the validators are maintained in Ethereum network.

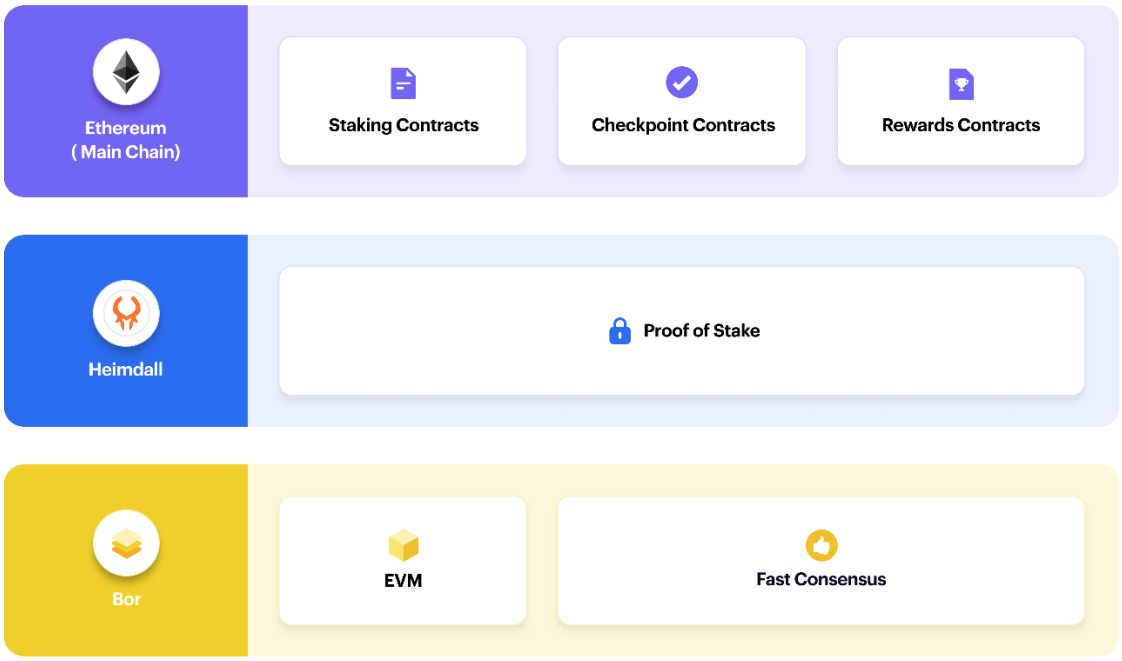


Figure 4.1 - Polygon PoS architecture [91]

Heimdall is also responsible for checkpointing, which consists in blocks aggregation into a single Merkle root and regularly publishing it to the Ethereum main chain. To finalize the checkpoint on the Ethereum chain, a proposer is chosen from the validator group. The proposer's role is to gather all the necessary signatures from validators, ensuring the authenticity of the blocks before they are committed to the Ethereum chain.

Bor [93], on the other hand, acts as the block producer layer, aggregating transactions into two blocks. Block producers are a subset of validators shuffled periodically by Heimdall. These producers are selected to validate blocks for a specific number of blocks, known as a "span" in the Polygon environment. The selection process involves assigning slots based on the amount of Matic staked by each validator. Once the slots are assigned to the block producers, slots suffer shuffling. The selection process then proceeds by choosing a specific number of slots based on the producer count required, determining the producers for the subsequent phase. Also, Bor is the layer where contracts are deployed and executed in an EVM environment.

Figure 4.2 demonstrates the producer selection process. Let's consider an example with three producers: A with 100 Matic staked, and B and C with 40 Matic staked each. Assuming each slot requires 20 Matic, the initial distribution would be as follows: A would obtain 5 slots, while B and C would each receive 2 slots (first array). Once the slots are assigned based on the Matic staked by each producer, they are shuffled into the second array. After the shuffling process, it is selected the number of slots equal to the defined producer count maintained by the validator's

governance (in this case, we defined 5). These selected slots would be taken from the beginning of the shuffled array, resulting in the array on the left side.

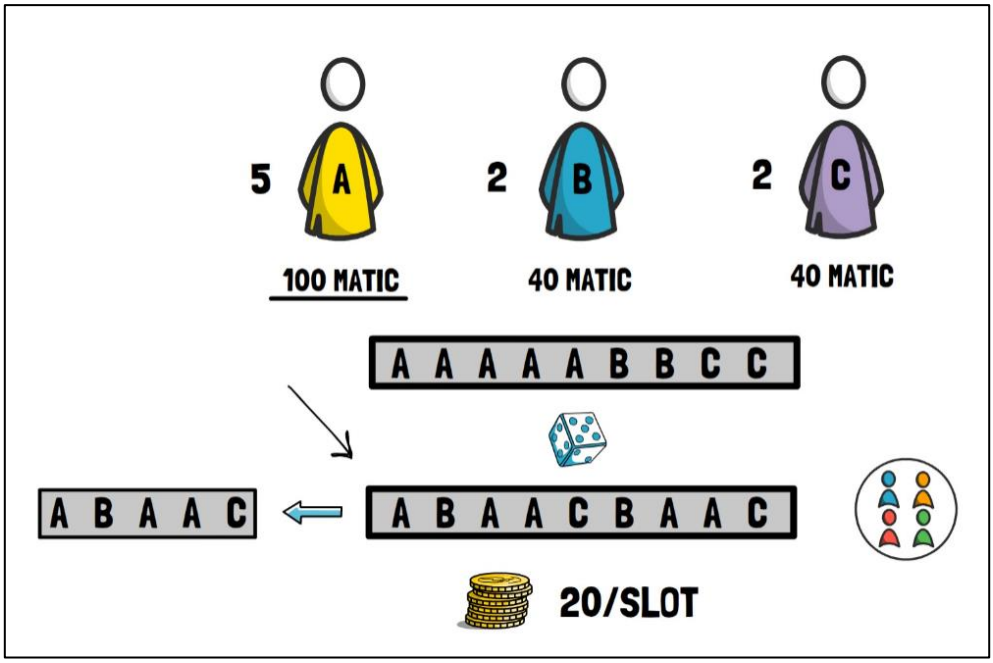


Figure 4.2 - Producer selection process [94]

This random selection model allows anyone to participate in securing the network with any amount of Matic tokens without sacrificing transaction speed, as not all validators need to validate all blocks every time.

Polygon operates in parallel with the Ethereum network and aims to enhance usability and interaction by offering a trustless two-way transaction channel through its cross-chain bridge, known as the Polygon PoS Bridge. This bridge enables users to transfer assets seamlessly between the Polygon and Ethereum networks without facing third-party risks or liquidity limitations in the market.

Through this bridging mechanism, users benefit from near-instant, low-cost, and flexible transactions. To maintain the circulating supply of tokens during the bridge crossing, tokens leaving the Ethereum network are locked, and an equivalent number of tokens are minted in the Polygon PoS. When returning to Ethereum, the process is reversed by unlocking the funds and burning the tokens minted in Polygon PoS.

What sets this bridge apart from others is its near-instant transaction time and the unique feature that all validators participate in securing the bridge. This ensures a robust and reliable cross-chain transaction experience for users.

The reason behind the choice of this blockchain network will be discussed further in the section 5.2.

4.2 Tools

This section introduces the tools employed in developing the system prototype. The main objectives of these tools are to enhance effectiveness, minimize errors and lastly provide various system capabilities. Table 4-2 offers a concise overview of each tool.

Table 4-2 - Tools used in the prototype development

Name	Description	Usage
Android Studio [95]	IDE that provides all tools necessary for android environment and development. Helps minimize the errors and facilitates android development	To write the source code of the app, manage files and dependencies, compile, and installation
Remix IDE [96]	IDE for writing smart contracts. Provides mechanisms to interact and test locally the smart contracts	Used to write the source code, compile, and test manually the developed smart contracts
OpenZeppelin [97]	Company that provides services related to blockchain and provides contracts standards	Utilization of its relayer service for meta-transactions and ERC-1155 contract standard.
IPFS [98]	Peer-to-peer distributed file system, that helps storing files off-chain	Store image files from the retailers and retailers' info
Jira Software [99]	Tool to help manage the software project and define goals.	Used to track the features done and not done and provide an overview of the project flow.
Web3Auth [100]	Infrastructure for wallet management and login	Used to provide a seamless login and wallet to the user
Hardhat[101]	Open-source development environment	Used to create and manage the smart contracts project

4.2.1 Android Studio

Android Studio [95] is a comprehensive integrated development environment (IDE) tailored specifically for Android app development. It serves as the primary tool for

Android developers, providing a user-friendly interface and a rich set of features to streamline the entire development process. With Android Studio, developers can write, test, and debug their Android applications efficiently. Its advanced code editor offers intelligent code completion, syntax highlighting, and refactoring tools to enhance productivity. The built-in Android Emulator allows for easy testing of apps on various virtual devices. Additionally, Android Studio comes equipped with a range of performance analysis tools to optimize app performance. Its seamless integration with the Android SDK (Software Development Kit) and Google services simplifies the development of cutting-edge Android applications. Overall, Android Studio remains an indispensable platform for creating high-quality Android apps, offering a comprehensive suite of features and advantages to cater to the needs of developers.

4.2.2 Remix IDE

Remix IDE [96] is a powerful web-based integrated development environment (IDE) specifically designed for developing smart contracts on the Ethereum blockchain. As a popular choice among blockchain developers with any knowledge level, Remix IDE offers a range of features and advantages to simplify the process of writing, testing, and deploying smart contracts. Its intuitive user interface and real-time code editor provide a seamless development experience. Remix IDE also supports smart contract compilation and debugging, allowing developers to identify and fix issues quickly. Another noteworthy feature is its integration with various Ethereum networks, enabling easy deployment and testing of smart contracts on different networks. Additionally, Remix IDE provides built-in tools for contract analysis and verification, enhancing the security and reliability of smart contracts. With its web-based nature, developers can access and work on their smart contracts from anywhere with an internet connection. Overall, Remix IDE stands as a valuable and versatile tool, offering a range of features and advantages to support the efficient development of smart contracts for the Ethereum blockchain.

4.2.3 OpenZeppelin

OpenZeppelin [97] serves as the industry standard for developing secure blockchain applications. It offers a suite of security products that enable developers to build, automate, and operate decentralized applications with enhanced safety measures. One of its core products, OpenZeppelin Contracts, is a modular library that comprises reusable and secure smart contracts written in Solidity. This library provides developers with a reliable foundation to create their systems without the need to start from scratch. These contracts adhere to well-defined rules and standards, including Ethereum token standards such as ERC-20, ERC-721, and ERC-1155, ensuring compliance with best practices.

Additionally, OpenZeppelin Defender is another essential product that automates various operational aspects. Its suite of tools includes: Admin, designed to automate,

and secure smart contract administration; Relay, to enable private and secure transactions and to facilitate meta-transactions infrastructure; Sentinel, to monitor contracts and send notifications; and Autotasks, to enable the creation of automated scripts for interacting with smart contracts.

OpenZeppelin empowers developers to build more robust and secure blockchain applications, safeguarding organizations' services and products. The library of reusable contracts and the automated operations platform contribute to a smoother and more reliable development process, following industry best practices and standards and supporting multiple blockchains.

4.2.4 IPFS

InterPlanetary File System (IPFS) [98] is a peer-to-peer distributed file system that aims to revolutionize the way we store, share, and access content on the internet. Unlike traditional centralized server-based file systems, IPFS operates in a decentralized manner, utilizing a network of interconnected nodes to store and retrieve data. Each piece of content in IPFS is identified by a unique cryptographic hash, ensuring data integrity, and enabling efficient content addressing. One of the standout features of IPFS is its content-based addressing, which allows identical content to be deduplicated across the network, leading to reduced storage costs and improved network efficiency. Furthermore, IPFS offers built-in versioning and historical content tracking, making it easier to access and retrieve previous versions of data. By creating a distributed and censorship-resistant file system, IPFS promotes greater data availability and resilience, making it a valuable tool for building decentralized applications, content sharing platforms, and improving the overall robustness and accessibility of the internet. This system is especially valuable for blockchain development, as it allows for the storage of large files off-chain while creating immutable and permanent links in transactions. By timestamping and securing content without directly storing the data on-chain, it ensures the reliability and efficiency of blockchain operations.

4.2.5 Jira Software

Jira Software [99] is a powerful project management and issue-tracking tool developed by Atlassian. It is designed to help software development teams plan, track, and manage their projects efficiently. With a user-friendly interface and robust features, Jira Software streamlines the entire development process, from ideation to release.

One of its key features is its flexible and customizable workflow, which allows teams to tailor their project management process according to their specific needs and methodologies. Teams can create custom issue types, define their workflow states, and set up automated transitions, ensuring a smooth and standardized development process. Jira Software also provides a comprehensive set of reporting and analytics

tools, enabling teams to monitor progress, track performance, and identify bottlenecks. The built-in dashboards and real-time reporting help stakeholders stay informed and make data-driven decisions to keep projects on track.

Furthermore, Jira Software facilitates seamless collaboration among team members with features like comments, mentions, and notifications. This fosters communication and knowledge sharing, ensuring everyone is on the same page and can quickly address any issues that arise during development. Overall, Jira Software serves as an indispensable tool for software development teams, enabling them to work more efficiently, collaborate effectively, and deliver high-quality projects on time. Its versatile features and robust capabilities make it a top choice for teams looking to streamline their development processes and successfully manage their projects from start to finish.

4.2.6 Web3Auth

Web3Auth [100] is a cutting-edge pluggable authentication infrastructure designed for Web3 wallets and applications. This innovative system simplifies the onboarding process for both inexperienced and crypto-native users, enabling them to get started in under a minute while offering experiences tailored to their preferences. By incorporating support for various social logins, web and mobile native platforms, wallets, and other key management methods, Web3Auth ensures a seamless and user-friendly authentication process. It results in a standardized cryptographic key provider specific to each user and application, enhancing security and usability. With Web3Auth, users can effortlessly access decentralized applications and services, making it a valuable solution for bridging the gap between traditional and blockchain-based ecosystems.

4.2.7 Hardhat

Hardhat [101] is an open-source development environment and task runner tailored for Ethereum smart contract development. It serves as a robust and flexible platform, streamlining the process of building, testing, and deploying smart contracts on the Ethereum blockchain. Offering a built-in Solidity compiler, developers can easily compile their contracts and take advantage of its optimization features. The development environment also includes a powerful testing framework that enables the creation and execution of unit tests for smart contracts. With support for different Ethereum networks, seamless contract deployment and network management become achievable. Hardhat's scriptable tasks allow developers to automate repetitive tasks, enhancing development efficiency. Additionally, the platform offers debugging and tracing capabilities for comprehensive contract inspection during execution. Its extensible architecture allows integration of community-developed plugins to enhance development further. In summary, Hardhat is a feature-rich and user-friendly environment that empowers Ethereum developers to efficiently create decentralized applications on the Ethereum network.

5 PROPOSED LOYALTY MANAGEMENT SYSTEM

In order to properly design a loyalty management system, the problem must be understood, and some factors considered first. These factors include understanding blockchain capabilities and how they fit into the proposed system, addressing current customer and retailer concerns, and providing a better overall user experience.

The proposed system in the current document is a loyalty management system capable of managing several loyalty programs and types in a single platform. This system is fully decentralized with two core modules: a mobile application and smart contracts. Regardless of the knowledge and role of the user (i.e., retailer or customer), the application is designed to completely abstract users from the underlying blockchain.

5.1 Decentralized application

The core feature of our loyalty management system lies in its complete decentralization. In order to understand the complexities and workings of the system, it is imperative to understand this underlying concept.

A decentralized application (Dapp/DApp) is an application built upon a decentralized network, integrating smart contracts as a backend and a frontend user interface. Smart contracts can be compared to an API (Application Programming Interface), which is accessible and transparent to any user. Unlike the conventional approach running on a centralized server owned by an entity, the DApp's backend code operates on a decentralized peer-to-peer network [102].

A DApp can be characterized by the following attributes [102]:

- Decentralized: DApp operate on a public decentralized platform where no single entity has control;
- Deterministic: they consistently perform the same function regardless of the environment in which they are executed;
- Turing Complete: DApp have the capability to perform any action given the required resources;
- Isolated: they are executed in a virtual environment, ensuring that any bug or issue do not affect the normal functioning of the network.

Developing a decentralized application offers several advantages, including zero downtime, improved privacy, resistance to censorship, data security and integrity, and trustless computation.

Zero downtime is ensured since once a smart contract is deployed on the blockchain, the network will always be available to serve clients seeking to interact with the

contract. Furthermore, due to the consensus mechanisms and the peer-to-peer nature of the blockchain, malicious actors would find it arduous to successfully launch denial-of-service attacks.

Privacy is a key feature as users can interact with a decentralized application without the need to provide real-world identity. A virtual wallet is sufficient for access.

Resistance to censorship is inherent in DApp, as no single entity on the network possesses the ability to block users from submitting transactions, deploying DApp, or accessing data on the blockchain.

Data integrity is assured since no single entity can manipulate or alter the transactions and data on the blockchain, which promotes transparency and reliability. Malicious actors cannot forge transactions or other data that is already public.

Trustless computation is achieved through smart contracts, which can be analyzed and audited, offering predictability without reliance on a central authority.

Nevertheless, DApp also present certain drawbacks, including maintenance challenges, performance overhead, network congestion, and potential user experience issues [102].

Maintenance becomes more complex with DApp since modifying the code and data published to the blockchain is difficult. Updates to DApp, or the underlying data stored by them, become arduous tasks for developers even when bugs or security risks are identified in older versions.

Performance overhead is a considerable concern as achieving high levels of security, integrity, transparency, and reliability requires each node to run and store every transaction. Additionally, consensus mechanisms to ensure data integrity introduce delays in processing transactions.

Network congestion can occur when a particular DApp consumes excessive computational resources, causing the entire network to slow down and reduce performance. When the network receives more transaction requests than it can handle, it slows down, leading to an increase in the queue of unconfirmed transactions.

User experience may suffer as user-friendly interactions become harder to introduce, considering that the average end-user might lack the required knowledge, and setting up the tool stack to interact with the blockchain can be challenging.

Having clarified the essence of a decentralized application, our loyalty management platform aims to be fully decentralized while also addressing some of the drawbacks of a decentralized application. Our focus is on streamlining maintenance processes, minimizing the likelihood of network congestion, and, most importantly, providing a user-friendly experience accessible to users of all knowledge levels on our platform.

To address these challenges, we employ a paradigm that enables upgradable smart contracts, as discussed in section 2.1.4, to mitigate the maintenance problem. To reduce the risk of network congestion, we must select a blockchain capable of

handling a high number of transactions per second (Section 5.2). Additionally, for a seamless user experience, we must abstract the complexities of blockchain involvement.

5.2 Blockchain Selection

One of the most critical components within the proposed system is the choice of the underlying blockchain platform. Currently, there are numerous solutions with distinct characteristics and purposes. For the proposed loyalty management system, some key factors are essential, such as the number of transactions per second, cost per transaction, ability to support smart contracts, scalability, and a good source of information and documentation. Since this system is intended to support several loyalty programs and each customer can subscribe to several programs, high transaction volumes per second are essential to ensure good performance. Considering the expected number of transactions, the cost of each transaction must be minimal so that retailers are not penalized, and the platform becomes more affordable.

Since the purpose of the system is to support and manage different types of loyalty schemas and loyalty programs, smart contracts are mandatory. Therefore, the chosen blockchain must support them. Due to the need for the system to handle platform growth, scalability is required. Therefore, the blockchain network must permit scaling. The blockchain system must also have good documentation to ensure the best use of blockchain resources and capabilities.

The choice of a set of candidates blockchain networks took all the mentioned factors into account. The chosen networks were Avalanche, Ethereum, Hyperledger Fabric (Hyper), Polygon PoS, and Solana. These networks are currently the most popular and are distinct from each other. Table 5-1 summarizes their main characteristics for comparison. The information presented was obtained from the available documentation, except for the transaction price. Transaction prices for Polygon, Ethereum, and Avalanche have been determined utilizing the following specialized websites, which are designed for scanning transactions provided by the blockchains: PolygonScan [103], EthereumIo [104], and Avascan [105] for Avalanche. Solana's price is available on their official website [106]. It is important to note that these prices are subject to change, as they depend on several factors, such as the value of the cryptocurrency at any given time and the volume of transactions processed over the course of a day. Concerning Hyperledger, the transaction price is zero, as it is a private network. It's worth noting that the number of transactions per second (TPS) shown in the table was obtained in a controlled test environment, which may not accurately reflect reality.

Table 5-1 - Blockchain network comparison

	Avalanche	**Ethereum**	**Hyper**	**Polygon**	**Solana**
Environment execution	EVM [1]-AVM [2]	EVM [1]	Docker	EVM [1]	LLVM
Smart contract language	Solidity	Solidity	Java/GO	Solidity	C/Rust
Transaction per second	4500	15–27	3500	10,000	65,000
Permission type	Public/Private	Public	Private	Public	Public
Transaction price	<0.01 $	≈4 $	0 $	<0.01 $	0.00025 $
Layer	Layer 1	Layer 1	Layer 1	Layer 2	Layer 1

[1] EVM—Ethereum virtual machine; [2] AVM—Avalanche virtual machine.

Despite Ethereum being the most popular among the five solutions, it presents a weakness regarding the number of supported transactions per second. Additionally, the cost for each transaction is too high, which can lead to platform rejection. Hyperledger Fabric (designated Hyper in Table 5-1) is a private network with a smart contract environment and language distinct from the usual Ethereum Virtual Machine (EVM) and Solidity language, which can cause implementation problems. Also, some experts claim that private blockchains defeat the purpose of using blockchain technology, making the application more complex, less efficient, and more expensive than a centralized database [36]. As for the users, these permissioned blockchains will make it difficult for users to access information from the public internet.

Despite Solana's support for a higher number of transactions per second, resulting in lower costs, it faces similar issues as Hyperledger when it comes to smart contract implementation. Unlike the mentioned public chains, Solana is not EVM compatible, which means it cannot utilize Solidity as its smart contract language. This becomes a weakness since the Ethereum network is the leader in smart contract development and deployment. Moreover, developers who wish to switch to a different blockchain platform will face challenges with Solana's lack of EVM compatibility. They will be required to rewrite their smart contracts from scratch, adding complexity and effort to the migration process.

The Avalanche and Polygon networks are both compatible with smart contracts developed for the Ethereum environment since they use EVM. The main differences between these two solutions are the level of popularity, the supported volume of transactions per second, and the type of layer. Polygon is more popular and allows more transactions per second. It is also a sidechain/layer-two network, while Avalanche is a layer-one. This means Avalanche is a network that works independently, while Polygon works in parallel with a main network (i.e., a layer one

network such as Ethereum and Avalanche). In the Polygon case, the main network is Ethereum, but in the future, it will also be compatible with other main networks.

In conclusion, the preferred blockchain network for the proposed loyalty system is Polygon PoS, due to its increasing popularity and capability of providing a fair price and a good volume of transactions per second. In addition, Polygon is public and compatible with Ethereum Virtual Machine, which allows the transfer of smart contracts to Ethereum or other compatible networks at any time. By choosing the Polygon PoS network, we automatically minimize the chance of a network being congested due to its high transactions per second support.

5.3 System description and features

The proposed system is a self-managed loyalty platform focused on retailers of low magnitude who can create loyalty programs in which their customers can participate. The platform has two distinct users (customers and retailers) and the loyalty programs can include a variety of mechanisms. Despite the system requiring the use of blockchain technology, it abstracts and helps users in onboarding by providing each one with a wallet. These wallets are connected to the users themselves and are private. Furthermore, customers are not required to pay any transaction fee generated by the blockchain interaction, since the resulting costs are intended to be supported by the retailers.

This system includes two main components: a mobile application and smart contracts. Most existing blockchain applications are web applications, which limits interaction and provides an entry barrier for some less tech-savvy users. On the contrary, a mobile solution improves accessibility, taking advantage of the ubiquity of smartphones in everyday life. Despite having two types of users, the mobile application is the same for both types of users. The objectives of smart contracts are to manage loyalty programs, support different mechanisms, and ensure that the assets of the users are secure.

One of the most critical aspects of loyalty programs is the need for security and privacy measures, which is especially relevant today since security is a major global concern. With the objective of ensuring customer privacy, the proposed system does not store any personal information. On the other hand, it uses Web 2.0 authentication methods, which allow users to use their previously created accounts, thus preventing them from providing any personal information whenever they access the loyalty management system. Although the data stored on the used blockchain are publicly accessible and can be viewed by anyone, the underlying consensus protocols make it possible to validate transactions and identify any potential risk, which makes the blockchain a useful technology for ensuring data security. Furthermore, by implementing the appropriate security measures in smart contracts, the data is interconnected with the user address, making only the owner capable of changing it. Also, executions that are not intended or any malicious

changes detected during transactions can be immediately reverted, thus ensuring data integrity.

Despite the mobile application being unique, both types of users have different use cases. Customers have access to a catalog in which all the loyalty programs are within the scope of the application. For each loyalty program, a customer can view the information related to the business and a description of the loyalty program itself. Additionally, the customer can enroll in loyalty programs to start participating and earning rewards. Once the customers subscribe to a program, they can begin to spend and earn rewards. The system helps customers track all the assets related to the various loyalty programs so that they can check and be aware of their progress.

Retailers have different use cases, as their goal is to attract more customers, thus increasing their business. Retailers can create a loyalty program by providing their business information and by choosing a loyalty mechanism along with a brief description. Additionally, during this process, the retailer can upload images to give a better perception of the loyalty program or business. Since the main goal is to increase brand reach, the retailer can access the analysis and statistics data of the loyalty program. As retailers must pay transaction fees, they can also buy the required cryptocurrency using the app.

The mobile application provides a transaction system based on a QR code approach for trading assets with retailers/customers, thereby enhancing the overall user experience. All transactions require the presence of two individuals, and the rewards are defined by the retailers during the process. The two individuals must be one retailer and one customer since customer-to-customer and retailer-to-retailer transactions are not possible in the system.

Figure 5.1 gives an overview of the proposed loyalty management system, illustrating the main components and interactions. Customers and retailers, which are the only actors in the system, use their personal smartphones running the mobile application to access the system.

A customer accesses the system to interact and discover new retailers, while a retailer must create his or her loyalty program and exchange rewards with customers. The mobile application employs Web 2.0 credentials for user login. To enable customer interaction with the underlying blockchain network at no financial cost, each transaction originated by a customer first passes through a relayer, which then redirects it to the respective smart contract. On the contrary, a retailer interacts directly with smart contracts. For the execution of asset transfers or rewards, the receiving entity displays the QR code while the other agent reads it. Upon completion of the QR code-reading process, the transfer of assets from the QR code reader to the receiver is triggered through a transaction sent to the smart contract. Depending on whether the transfer is being performed by the retailer or the customer, it will go directly to the smart contract or through the relayer, respectively.

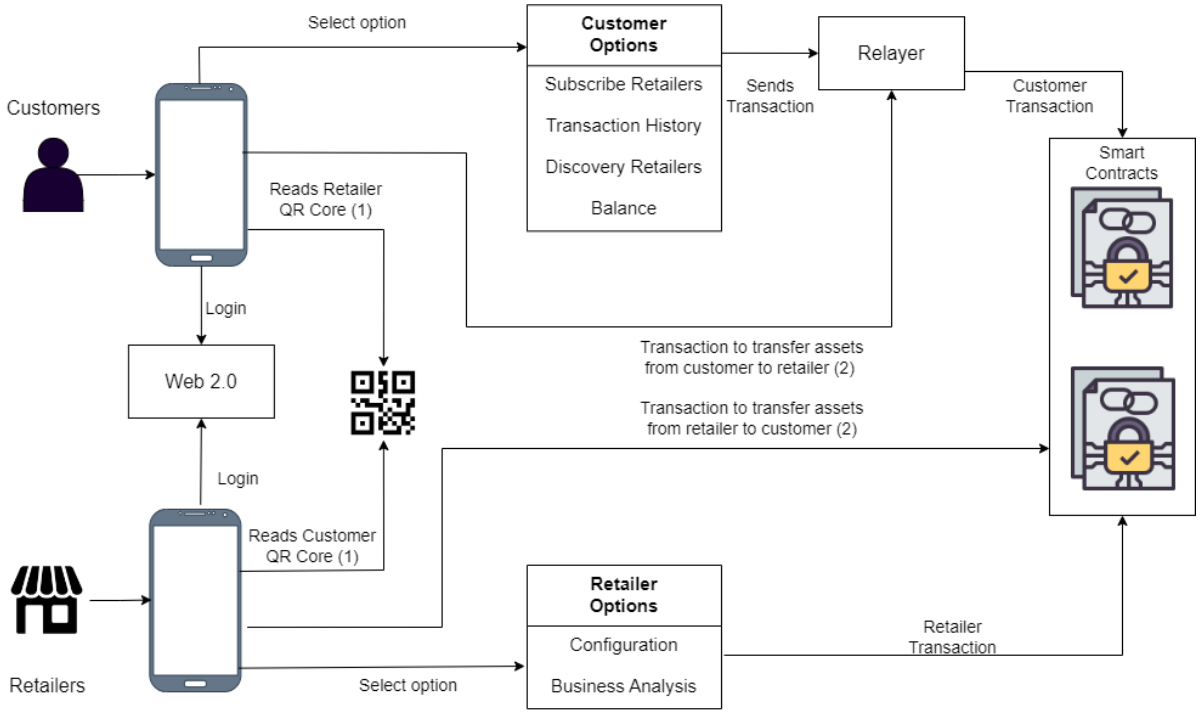


Figure 5.1 - Proposed loyalty management system overview

Despite its apparent straightforwardness, the system has certain restrictions and non-functional requirements that are essential for achieving satisfactory performance and providing good user experience. Table 5-2 shows the requirements at the system level.

Following an agile methodology, Tables 5-3, 5-4, 5-5 summarize the system user stories. To define the user stories, they were written following the agile structure and guidelines. An example of a fully detailed user story can be seen in Figure 5.2.

In summary, the system's goal is to manage multiple loyalty programs and types, primarily targeting smaller retailers like cafes, grocery stores, hairdressers, barber shops, and more. The main objectives, in addition to managing loyalty programs, include enhancing the user onboarding process by eliminating entry barriers at both blockchain and loyalty program levels, utilizing the widespread availability of mobile platforms, and incorporating all business logic into smart contracts without the need for centralized servers or databases.

Table 5-2 - System requirements

System Requirement	Name	Description
SR-1	Performance	Capable to perform several transactions per second
SR-2	Minimal cost per transaction	The cost for each transaction has to be minimal
SR-3	Business logic	All business logic must be contained in smart contracts
SR-4	Decentralized application	The system must operate independently without relying on any database or backend server
SR-5	Scalability	The system must be architecturally designed to allow new types of loyalty programs
SR-6	Data storage	Storage the minimum information. Only the essential and non-private information can be stored in blockchain.
SR-7	Meta Transactions	System must employ meta transactions to remove the customer payment responsibility
SR-8	Login	The system cannot have any registration process to create an account, leveraging the existing user accounts.

5.4 UI Design

The mobile application, being one of the core components of the system, requires an excellent user interface not only to enhance its appeal but also to elevate the overall user experience. For the UI design, two versions were created:

1. The initial version consisted of mockups elaborated using personal experience and based on existing apps in the market, since the WIT designers were unavailable at the time;
2. The second version was an improved iteration crafted by WIT designers after the Business Analyst, Ane Polito, reviewed and redirected the initial designs for enhancement.

This process of developing the mockups is crucial for understanding critical viewpoints and user flows, ensuring smooth navigation throughout the application, and refining the requirements. It is worth noting that Ane was involved during the definition of both the requirements specification and the design development to ensure the appropriate outcome. Figures 5.3 and 5.4 demonstrate an example of both versions of the same screen. The remaining screens are exhibit in Annex B.

Table 5-3 - Common user stories

User Story	Description	Name
US-1	As a user, I need to login, so that I can use the app	Login
US-2	As a user, I need to have a QR code identifier, so that I can receive transactions	Display QR code
US-3	As a user, I need to scan QR codes, so that I can send transactions	Read QR code
US-4	As a user, I need to perform transactions, So that I can send/receive rewards.	Perform Transactions
US-5	As a user, I need a Home Page, So that I can easily access information.	Home Page
US-6	As a user, I need to a transaction history, So that I can track my rewards.	Transaction History

Table 5-4 - Customer user stories

User Story	Description	Name
US-7	As a customer, I need to check my loyalty balance, so that I can be aware of my progress	Check loyalty balance
US-8	As a customer, I need to access a map, so that I can find new nearby business	Discovery Map
US-9	As a customer, I need a discovery page, so that I find new businesses to subscribe	Discovery page
US-10	As a customer, I need to subscribe to a retailer, so that I can start earning rewards	Subscribe retailers

Table 5-5 - Retailer user stories

User Story	Description	Name
US-11	As a retailer, I need to create my loyalty program, so that I can expand my business	Loyalty Creation
US-12	As a retailer, I need to check my top customers, so that I can gain insight into my success	Top Customers
US-13	As a retailer, I need to exchange money into cryptocurrency, so that I can pay for blockchain interactions	Currency exchange

5.5 System Architecture

After taking into consideration all factors mentioned previously, it was essential to define a detailed architecture with all the different software components, the interaction with external services, and the communication protocols. The architecture is composed of three major modules: (i) the mobile application the users will interact with; (ii) external services for facilitating user integration with blockchain and providing a set of required features, namely InterPlanetary File System (IPFS), Torus Web3Auth, Ramp Exchange, and OpenZeppelin; and (iii) smart contracts that contain the system's business logic. Figure 5.5 represents the system architecture.

As the system allows for different loyalty mechanisms, to facilitate the entire loyalty program management process, each smart contract represents a specific loyalty mechanism. Since numerous retailers with a specific loyalty mechanism can exist, each smart contract inherits the ERC-1155 interface [29] through the contract available through OpenZeppelin. ERC-1155 is a token standard interface that facilitates the smart contract representing multiple tokens simultaneously and includes management functions that efficiently help in reducing the fees. This interface, along with the already pre-built ERC-1155 contract, simplifies the whole management process and diminishes the effort needed to create smart contracts from scratch. Furthermore, since ERC-1155 allows multiple tokens to be managed within a single contract, the proposed loyalty management system can significantly reduce the number of smart contracts required, unlike current systems that require separate ERC-20 contracts for each token. Given the large number of tokens involved in our system, the use of ERC-1155 thus results in a more efficient and scalable solution that requires a minimum number of contracts: a single smart contract is required for delegating calls and storing data unrelated to loyalty programs, and a separate smart contract inheriting the ERC-1155 contract is required for each loyalty mechanism.

Module	Mobile App
Task	Login
User Story 1	Login
As a **I need** **So that**	User To be able to login my profile I can utilize the app

Description	The system provides three different ways to Log in such as:
	• Web 2.0 credentials
	  ○ Facebook
	  ○ Twitter
	  ○ Gmail
	  ○ iCloud
	• Email
	• Wallet Providers
	  ○ MetaMask
	When logging in, if the customer does not choose a wallet provider and it is the first time, the system should create a wallet.
	The customer moves to the Home screen if the Login is successful.
	The customer gets an error message if login failed.

Acceptance Criteria	**AC-1**: The user must choose one of the above options to log in so he can interact with the system. **AC-2**: The wallet information must be managed by an external entity. The system should not possess any information about the private keys. **AC-3**: The profile information such as name, address, number, profile picture, etc; must be saved in IPFS system with JSON file. **AC-4**: If the user fails to log in, the system should redisplay the login page and an informative message. **AC-5**: The system should redirect the customer to Home page, if the log in is successful.
User Flow	Log in Page >> Select Log in option with 2.0 credentials >> Host Log In page >> Insert credentials >> Log in succeeds >> Home Page Log in Page >> Select Log in option Wallet Provider >> Host Log In page >> Insert credentials >> Log in succeeds >> Home Page Log in Page >> Select Log in with e-mail >> Insert credentials >> Log in succeeds >> Home Page Log in Page >> Select any option to authenticate >> Log in fails >> Message error >> Log in Page
Priority	High

Figure 5.2 - Example of User story specification

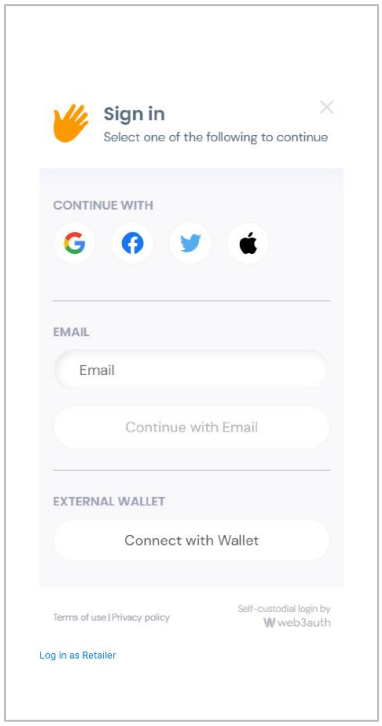


Figure 5.3 - Example of 1st version mockups

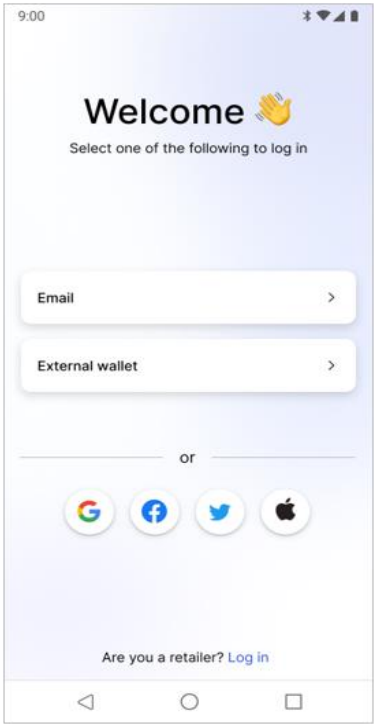


Figure 5.4 - Example of 2nd version mockups

The proposed loyalty system includes an intelligent contract manager to simplify blockchain calls, which is responsible for receiving transaction requests and delegating their execution to a contract running a specific loyalty mechanic. The manager is also responsible for storing all generic data, such as which retailers are registered in the system, customer subscriptions, and smart contract addresses for loyalty mechanics. Through this approach, calls to interact with and obtain information from the blockchain are simplified since the front-end application is only required to communicate with a single contract. Furthermore, in order to prevent any unauthorized data manipulation, all data are directly associated with the user's unique wallet address. This approach guarantees that only the users themselves can modify their data, thus preventing any malicious user from tampering with other users' information.

In order to streamline the development of smart contracts for each mechanism and store them in the loyalty manager component of the proposed system, a loyalty mechanism interface was created. This interface defines four methods that each mechanism contract must implement to properly define its behavior. The loyalty manager stores all the mechanism contracts in a map structure, upon which the keys are constants defined in an enumeration and the values are references to the loyalty mechanism contracts that are responsible for executing the received calls.

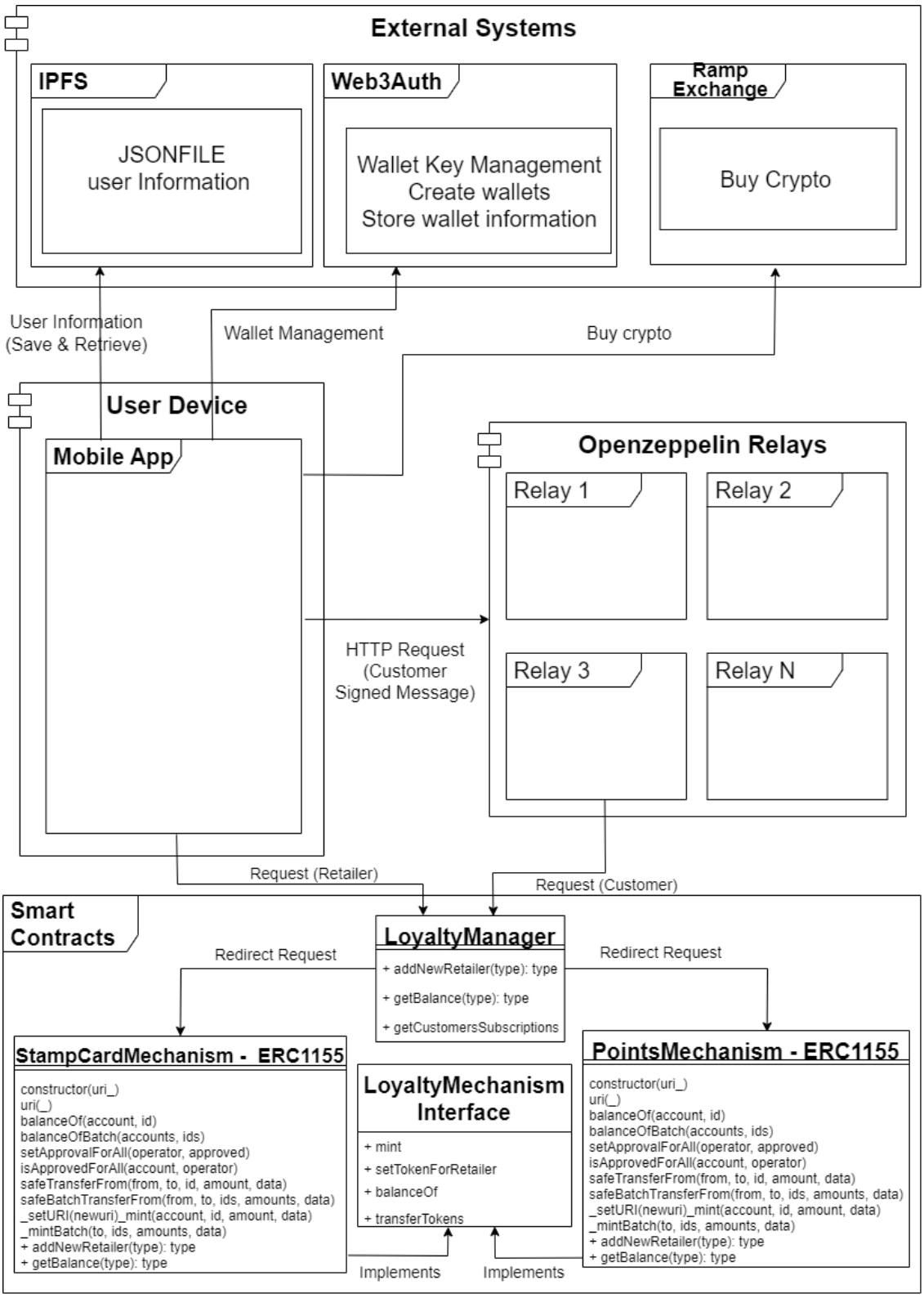


Figure 5.5 - System Architecture

Torus Web3Auth is a pluggable authentication infrastructure for Web3 wallets and applications. It provides seamless onboarding by providing web 2.0 login flows, such as Facebook, Gmail, and Twitter. Even though the users are abstracted from wallet

creation, they are always in control of ownership as it is directly associated with their credentials. Torus Web3Auth also manages the key infrastructure of the wallets (thus taking the responsibility away from the system) and can perform direct integration through a software development kit (SDK) for native Android, which provides a group of tools to enable the implementation of authentication. This authentication infrastructure allows for better user integration with blockchain, with all abstraction layers provided. Users can easily obtain a wallet through their Web 2.0 credentials without needing to create a new account and provide new information.

Despite the proposed loyalty management system is not intended to store personal information, retailers can publish images about their business and other business information. All these data are stored in the IPFS module in a JSON-like file, in which the file is also encrypted and assigned a hash to allow accessing it. The hash is a set of randomly generated characters that is created based on the content of the file. IPFS is a distributed system for storing and accessing files and a peer-to-peer hypermedia protocol which still guarantees the decentralization element. Since inserting information into the blockchain has an economic cost, non-relevant and sensitive information related to users and the system can be left off the chain. IPFS is appropriate for this scenario as it is also a peer-to-peer network. The only way of accessing the data is through the unique hash code, which will be stored on the blockchain and linked to the retailer's information.

Ramp is a service that allows exchange between cryptocurrency and physical money. The need for this service comes from the retailer use case, as retailers need to pay transaction fees to interact with the blockchain. So, to simplify this activity, the platform directly supports the purchase of the necessary token to remove the harsh process of retailers having to access an external platform and provide their wallet to buy the token, which could be a hassle for less tech-savvy users.

Meta transactions are essential to provide a more satisfactory onboarding process for any user, regardless of their degree of expertise. OpenZeppelin relays are employed to implement meta transactions, acting as intermediaries that receive transactions from customers, thus preventing them from paying fees. These transactions are sent via HTTP and signed by the client that created them. In the current system architecture, all requests are routed to the loyalty manager or, in the case of customer requests, to the relay, which are then routed to the loyalty manager contract. However, as the number of customer requests may increase in the future, it may be necessary to assign multiple relays to the loyalty manager to prevent any relayer from being overwhelmed with requests and causing system blockages, and to ensure a smoother flow of operations.

Summarizing the aforementioned set of external services, Web3Auth is used for wallet/key infrastructure management, IPFS is used to store any data in off-chain mode, Ramp is used by retailers to purchase cryptocurrency to pay fees, and OpenZeppelin relays are for meta transactions. The ERC-1155 contract standard provided by OpenZeppelin is also used.

The mobile application is the interaction center in which users execute their requests. In the case of retailers, the application communicates directly with the blockchain and external services. In the case of the customers, the application also communicates directly with the external service, but, whenever it is necessary to interact with the blockchain, an HTTP request is sent to a relayer, which later redirects it to the smart contract.

To interact with the public testnet of the Polygon PoS, we utilize a public RPC (remote Procedure Call) URL as the entry point for our application. However, this public URL is exposed by services that impose traffic limits. Consequently, if we exhaust all the allowed requests, we may need to switch to another RPC URL from a different provider. While hosting a full node could circumvent this issue, it requires a lot of computational power and resources, and for the nature of the project, hosting a full node is not optimal or required.

Figure 5.6 illustrates a temporal sequence diagram of a blockchain transaction in the proposed loyalty management system. Both types of users, i.e., customer and retailer, must first log in to interact with the system, as shown in the diagram. After completing the login process, the retailer queries the blockchain to retrieve his or her data, such as the IPFS hash. Once the data is returned, a second request is sent to IPFS to retrieve the remaining information associated with the hash. Then, if the user is a retailer, invoked transactions are directly sent to the blockchain, whereas, if it is a customer, transactions go through a relay to avoid fees. The transaction first contacts the loyalty manager contract, which is responsible for delegating all the transactions to the right contract loyalty type.

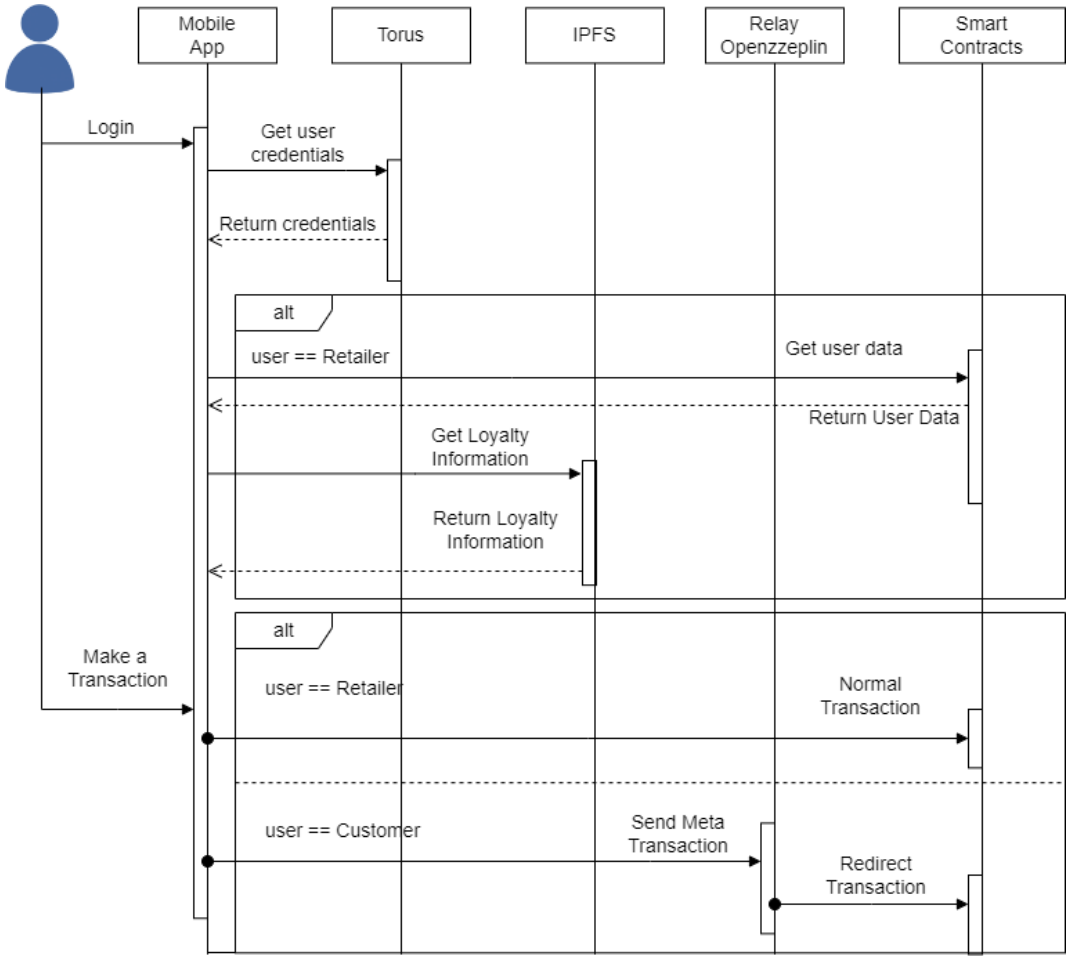


Figure 5.6 - Sequence diagram blockchain interaction

6 SYSTEM IMPLEMENTATION

Based on the system goals, functional and non-functional requirements, system architecture, and system execution environments defined in previous chapters, this chapter provides a comprehensive explanation of how the proposed loyalty management system was implemented and organized.

The platform, as it is a fully decentralized application, comprises two core modules: the mobile application and the smart contracts. Thus, to ensure clarity and coherence, this chapter is divided into two parts, starting with the smart contracts, as they implement the business logic, and following with the mobile application.

6.1 Smart contracts

Smart contracts form the core module responsible for the system's business logic, which makes them a crucial and fundamental aspect of the fully decentralized management system. Their stability and functionality are of utmost importance as they serve as the primary means of executing data-related operations.

In implementing the smart contracts for the designated system, it was imperative to ensure that our smart contract infrastructure supports meta-transactions, i.e., the capability to manage various loyalty mechanisms and programs, runtime addition of new mechanisms, and the possibility for maintenance and upgrades.

6.1.1 Smart contract architecture

Despite the system architecture (Figure 5.5) displays only four contracts, it is necessary to have additional contracts behind the scenes to provide a variety of features. As explained earlier, our system employs meta-transactions to enable users to interact with the blockchain without paying fees. Additionally, as stated in Section 5.1, one of the main challenges of a decentralized application is the maintenance problem of smart contracts, which can be overcome by using upgradable smart contracts. For these reasons, more contracts than the ones initially shown are required.

Figure 6.1 presents an overview of all the contracts used, illustrating their interactions in both upgradeable and non-upgradable scenarios. Different colors on arrows indicate the usage of the proxy pattern for upgradable contracts (blue arrows) versus non-upgradable contracts (black arrows). The green dots represent the contracts implemented in the context of the project described in this document, while the remaining ones are provided by OpenZeppelin with plugins or already implemented contracts.

The proxy contracts serve as intermediaries that store information for the attached contracts. For instance, the "Proxy Loyalty Manager" stores the information that

would otherwise be stored in the "Loyalty Manager". These proxy contracts help separate responsibilities, allowing the "Loyalty Manager" to focus solely on executing business logic while permitting changes to the contract logic without losing access to the data. Updates simply involve pointing the proxy to the newly deployed contract. To create these contracts, we used an OpenZeppelin plugin to do this automatically in the deployment process. Section 2.1.4 explained more in detail this design pattern and plugin. For both scenarios (upgradeable and non-upgradeable smart contracts), a different deployment process is necessary.

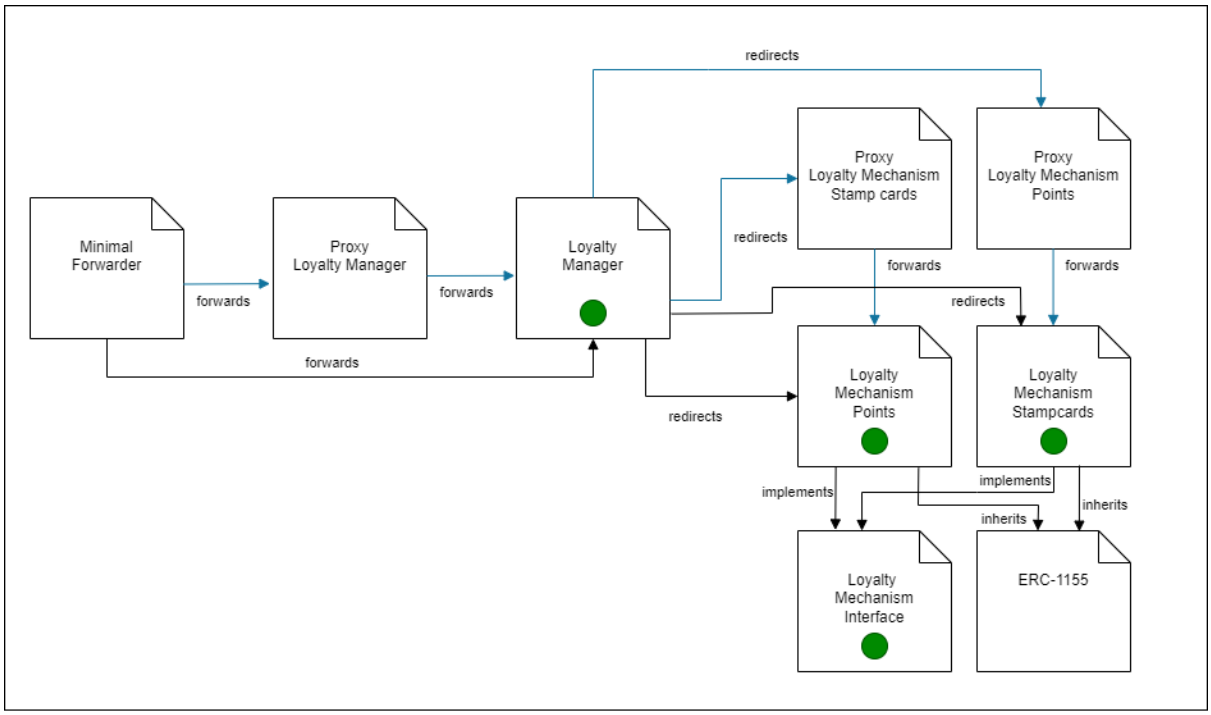


Figure 6.1 - Smart contracts architecture

To handle meta-transactions, we utilized the existing "Minimal Forwarder" contract from OpenZeppelin. This contract, as the name implies, only fulfills the requirement of receiving the request from the relayer and efficiently forwarding it to either the "Loyalty Manager" (non-upgradable scenario) or the "Proxy Loyalty Manager" (upgradable scenario).

To streamline the system and control entry points effectively, we adopted the approach of using the "Loyalty Manager" to receive all requests. This way, the system relies on a single address rather than on multiple ones, which would be impractical with multiple entry points and having to choose the right one for each interaction. By implementing this centralized point of contact, communication from the mobile app becomes more straightforward and minimizes the number of requests.

Moreover, the "Loyalty Manager" is responsible for storing the different contract mechanisms and communicate with them by redirecting the requests. To simplify

the communication process and storage of various loyalty mechanisms, we created a "Mechanism Interface" that each mechanism implements. This interface contains standard methods common to all mechanism contracts, enabling the "Loyalty Manager" to interact with mechanisms contracts without needing specific knowledge of each mechanism's details. Additionally, this design allows for future additions of new mechanisms without altering the source code - simply adding the new mechanism on deployment or at runtime to the "Loyalty Manager."

Furthermore, to manage loyalty programs efficiently, every mechanism contract inherits from the ERC-1155 contract. This decision was made because each mechanism is responsible for managing its loyalty programs, and ERC-1155 already provides capabilities for handling multiple tokens (both NFT and FT). Utilizing this existing standard from OpenZeppelin eliminates the need to build such functionality from scratch.

By adopting this architecture, we address several critical issues. First, we enhance the user experience through the implementation of meta-transactions, allowing users to interact with the system without worrying about transaction fees. Second, we tackle the maintenance problem associated with immutable smart contract source code on the network. Furthermore, our architecture enables scalability, as new mechanism contracts can be added seamlessly at any time. Additionally, managing multiple assets becomes more efficient with the use of ERC-1155.

It is important to highlight that offering both upgradable and non-upgradable scenarios effectively address concerns related to contract legitimacy. Some people may be apprehensive about upgradable contracts fearing potential changes that could disadvantage them.

6.1.2 Requirements

The requirements associated with smart contracts implementation are presented in Table 6-1.

6.1.3 Implemented smart contracts

To implement the system's smart contracts, we first created a project using Hardhat, which provides a development environment for Ethereum software compatibility. We opted to use the Remix IDE to write the source code. This IDE allowed us to directly write and test the code, providing features to detect syntax and compilation errors, along with debugging capabilities. The Remix IDE simulated an environment in our browser, granting us an interface to interact with every function defined in the smart contract. Utilizing this simulated environment proved more efficient than direct blockchain interaction, as it eliminated the need for deploying and implementing interaction scripts. The IDE's user-friendly interface enabled us to solely focus on writing and interacting with smart contracts.

Table 6-1 - Requirements in smart contracts

Requirements	Name	Description
R-1	Implement meta-transactions	The contract infrastructure must support meta-transactions.
R-2	Compatibility with several loyalty programs	The contracts must support multiple loyalty programs and types
R-3	Add new retailers and loyalty programs	Allow a retailer to create a loyalty program.
R-4	Retrieve a retailer info	Able to retrieve information associated with a retailer.
R-5	Retrieve all retailers presented in the system	Able to retrieve all information about the retailers registered in the system.
R-6	Customers subscribe retailers	Customers need to subscribe the retailer they want
R-7	Retrieve customers subscriptions	Get all the customer subscriptions
R-8	Mint tokens	Capacity to mint tokens for retailers reward the customers
R-9	Retrieve balance in a specific retailer	Get the balance in a loyalty program
R-10	Transfer tokens	Capacity to transfer tokens between retailers and customers
R-11	Get customers transactions counter in loyalty program	Obtain the customers number of transactions in the loyalty program

After ensuring the smart contracts were free of syntax and compilation errors, we conducted manual tests on each function. The manual testing was facilitated by the Remix IDE that provides an interface with all functions present in the smart contract that directly calls the smart contract. With this approach we manually tested some scenarios to guarantee that function works as intended. Subsequent to write and manually test the contract functions, we used Hardhat capabilities. This tool helped manage the necessary dependencies and provided tools for writing tests and deploying the contracts to a real blockchain network. Hardhat enabled us to run a local node, simulating the network and conducting unit tests for the contracts using JavaScript. Additionally, it facilitated cleaning the node and deploying the contracts whenever necessary. Once everything was ready, we deployed the smart contracts to the Polygon PoS testnet using a deploy script. While we chose not to deploy to the mainnet due to involving real money, the testnet served the same purpose, providing a testing and implementation environment without requiring actual funds.

In summary, our initial focus was on writing the source code in the Remix IDE and manually testing and interacting with it to obtain an initial impression and correct

the most obvious bugs. Subsequently, we utilized Hardhat for managing dependencies and providing tools for unit testing and deployment to the testnet.

6.1.3.1 Loyalty Mechanism interface

The Loyalty Mechanism interface serves to streamline the implementation process of mechanism contracts and establish a standard definition for the Loyalty Manager contract. This interface allows the Loyalty Manager to store multiple mechanisms and communicate with them without needing to know specific details about each contract. The Loyalty Mechanism interface comprises five methods that each mechanism must implement. Additionally, it defines two standard events and a data structure, which ensures standardization across all mechanisms. Figure 6.2 shows the Loyalty Mechanism interface definition.

By utilizing this interface, flexibility is granted to all mechanism contracts to tailor their implementation according to their individual needs. Simultaneously, it provides an abstract layer to the Loyalty Manager, promoting standardization across all implementations.

```
interface LoyaltyMechanism {
    function mint(address to, uint amount) external;
    function setTokenForRetailer(address retailer) external;
    function userBalanceOf(address account, address retailerAddress) external view returns(uint);
    function transferTokens(address from, address to, uint amount, bool isSenderRetailer) external;
    function getCustomersTransactionsCounter(address retailerAddress) external view returns(CustomerTransactionsCount


    event Mint(address indexed from, uint tokenId, uint amount);
    event Transfer(address indexed from, address indexed to, uint tokenId, uint amount, uint timestamp);

    struct CustomerTransactionsCounter {
        address customerAddress;
        uint numberOfTransactions;
    }
}
```

Figure 6.2 - Loyalty Mechanism interface

6.1.3.2 Mechanisms contracts

For the prototype of the system we propose, we implemented two loyalty mechanisms: points and stamp cards. These choices were made based on their distinct characteristics. The points mechanism was an obvious choice due to its widespread popularity. Additionally, utilizing tokens to represent points was a logical approach, thus we proceeded with its implementation. On the other hand, we opted for the stamp cards mechanism as it is commonly used by smaller retailers in their loyalty programs. Moreover, as mentioned earlier in Section 3.2, some companies are transitioning from physical cards to digital cards.

Regarding the business logic, the points program operates as follows: each token corresponds to one point in a 1:1 ratio. As for the stamp cards program, we adhered to its core principle. Whenever the customers transact with a retailer, the retailer

rewards them with one or multiple stamps. Once the customers accumulate ten stamps (a default value set by us), they receive a token. If the customer initiates the transaction, the system automatically sends one token to the retailer.

Starting with the loyalty points mechanism contract, it implements the Loyalty Mechanism interface as well as the ERC-1155 token standard. This contract is responsible for providing a unique *tokenId* (token identifier) for every retailer that registers in this type of loyalty program. The *tokenId* is used to manage the token, such as minting new tokens, transferring them, and checking the balance. Each retailer has its own *tokenId* and can only perform operations on its own token. The contract abstracts this process from users, so they do not need to worry about their *tokenId* when performing actions. The *tokenId* are stored in a mapping, which operates similarly to a traditional hashmap. By default, the Solidity programming language initializes each entry with the default value and, since a *tokenId* is a uint (unsigned integer) with positive values, including 0, it is initialized to zero. The mapping uses the retailer's address as the key and the *tokenId* as the value. This ensures that we can only retrieve the *tokenId* with the retailer's address, making token generation and management secure and controlled. Utilizing the ERC-1155 standard also enables us to manage multiple tokens without having to implement them separately, as this token standard already ensures best practices and safety measures for smooth execution.

To track the top customers of each loyalty program, we use another mapping where the keys are the retailers' addresses, and the values are arrays of a data structure defined in the Loyalty Mechanism interface (Figure 6.2). This approach ensures that each retailer can only access its own data since the key is the retailer's address. Furthermore, the system cannot access the data without a retailer address. The data structure stores information about the customer's address and the number of transactions performed at that specific retailer. Figure 6.3, Figure 6.4, and Figure 6.5 illustrate some functions of the points mechanism contract.

```
contract LoyaltyProgramPoints is ERC1155, LoyaltyMechanism {

    uint tokenIDIncrementor;
    mapping (address => uint) tokenOwners;
    mapping (address => CustomerTransactionsCounter[]) mappingCustomerTransactionsCounter;

    constructor() ERC1155("") {
        tokenIDIncrementor = 0;
    }

    function setTokenForRetailer(address retailer) external{
        uint tokenId = getRetailerTokenId(retailer);
        if (tokenId == 0) {
            tokenIDIncrementor = tokenIDIncrementor + 1;
            tokenOwners[retailer] = tokenIDIncrementor;
        }
    }
```

Figure 6.3 - Points Mechanism Contract

```
function mint(address to, uint amount) external {
    _mintInternal(to, amount);
}

function _mintInternal(address to, uint amount) internal {
    uint tokenId = getRetailerTokenId(to);
    require(ownerOf(tokenId, to), string.concat("Not the owner of the Token", Strings.toString(tokenId)));
    _mint(to, tokenId, amount, "");

    emit Mint(to, tokenId, amount);
}

function userBalanceOf(address account, address retailerAddress) external view returns(uint) {
    uint tokenId = getRetailerTokenId(retailerAddress);
    return balanceOf(account, tokenId);
}
```

Figure 6.4 - Points mechanism contract - functions userBalanceOf and mint

The function *setTokenForRetailer* is responsible for assigning the *tokenId* and storing it in the mapping along with the retailer's address.

The function *mint* had to be divided into two separate functions. First, the *mint* function is marked as external because it is defined in the interface. However, this posed a barrier to the business logic. Being external means only external sources can call the function, making it impossible to call it inside the contract. To address this issue, we created an internal function that is called by the *mint* function to create new tokens. This approach was necessary because the *transferTokens* function requires the mint capability, which could not be directly accessed due to the external restriction.

```
function transferTokens(address from, address to, uint amount, bool isSenderRetailer) external {
    uint tokenId;
    if (isSenderRetailer) {
        tokenId = getRetailerTokenId(from);
        if (amount > balanceOf(from, tokenId)) {
            _mintInternal(from, amount);
        }
        findCustomerTransactionsCounter(from, to);
    }
    else {
        tokenId = getRetailerTokenId(to);
        findCustomerTransactionsCounter(to, from);
    }

    _safeTransferFrom(from, to, tokenId, amount, "");

    emit Transfer(from, to, tokenId, amount, block.timestamp);
}
```

Figure 6.5 - Points mechanism contract - function transfer tokens

The *transferTokens* function is also marked as external because it is defined in the Loyalty Mechanism interface. As the name suggests, this function is responsible for transferring tokens from one user to another. The behavior of the function varies

depending on the sender. If the sender is the retailer, the function retrieves the *tokenId* associated with the retailer and checks if the retailer has sufficient funds. If the retailer has enough tokens, the transfer is executed immediately. Otherwise, new tokens must be minted before the transfer can take place. If it is a customer, it first finds the *tokenId* of the receiving retailer and then sends the funds. Regardless of whether the customer is the sender or not, the number of transactions for that customer with the specific retailer is incremented for each transfer. Upon completion of the transaction, an event is emitted. These events are stored in the transaction log of the blockchain and include the defined arguments. In this way we can store the data in blockchain without having it on our contract.

Regarding the stamp card loyalty mechanism contract, it shares many similarities with the points system in terms of definition. Among other similarities, this contract assigns a unique *tokenId* and divides the *mint* function into two functions just as the points mechanism contract does. The key differences are in how rewards work and how transfers are managed. As explained earlier in this section, the logic behind the stamp card mechanism involves rewarding a customer with one or more stamps when they interact with a retailer's business and, when they reach a certain condition, they receive a token. However, the contract does not directly have a variable to track the number of stamps. Instead, it employs the mechanism logic where a stamp is equivalent to a transaction. So, to determine the number of stamps a customer has, we can look at the customer number of transactions in that retailer. In other words, every time there is a transaction the contract increases the number of transactions and checks if the customer is eligible to receive a reward. To be eligible, the customer must have ten stamps, and the contract determines this by dividing the number of transactions by ten and checking if the remainder is zero. If it is zero, it means the user has collected ten stamps, and they receive a single token (compared to multiple tokens in the points system, where each point is equivalent to one token). Even if the user has, for example, fifty-five transactions, they will not be eligible to receive the token, as the logic dictates that fifty-five divided by ten results in a remainder of five. The next milestone for receiving the token would be at sixty transactions.

With this approach, we can continuously track whether the user has fulfilled the condition and increment the number of transactions without the need to create an additional variable. This streamlined logic allows for efficient management of the stamp card mechanism and ensures accurate rewards for customers. To ensure the best of this system we changed how the customer data is stored. Figure 6.6 and Figure 6.7 show the differences from the loyalty points mechanism, while the rest remains the same.

```
contract LoyaltyProgramStampcards is ERC1155, LoyaltyMechanism {

    uint public constant TRANSACTIONS_NUMBER_RULE = 10;
    uint public constant TOKEN_AMOUNT_RULE = 1;

    uint tokenIDIncrementor;
    mapping (address => uint) tokenOwners;
    mapping (address => CustomerTransactionsCounter[]) mappingCustomerTransactionsCounter;
    mapping (address => mapping (address => uint)) public indexOfCustomerInCustomerTransactionsCounterArray;

    constructor() ERC1155("") {
        tokenIDIncrementor = 0;
    }
```

Figure 6.6 - StampCards Mechanism contract

```
function transferTokens(address from, address to, uint amount, bool isSenderRetailer) external {
    if (isSenderRetailer) {
        processRetailerTransfer(from, to, amount);
    }
    else {
        processCustomerTransfer(from, to);
    }
}

function processCustomerTransfer(address from, address to) internal {
    uint tokenId = getRetailerTokenId(to);
    _safeTransferFrom(from, to, tokenId, TOKEN_AMOUNT_RULE, "");
    emit Transfer(from, to, tokenId, TOKEN_AMOUNT_RULE, block.timestamp);
}

function processRetailerTransfer(address from, address to, uint amount) internal {
    uint index = indexOfCustomerInCustomerTransactionsCounterArray[from][to];

    //Scenario where the customer hasn't made any transactions yet.
    //Initialize object
    if (index == 0) {
        mappingCustomerTransactionsCounter[from].push(CustomerTransactionsCounter(to, 0));
        indexOfCustomerInCustomerTransactionsCounterArray[from][to] = mappingCustomerTransactionsCounter[from].length;
    }
    index = indexOfCustomerInCustomerTransactionsCounterArray[from][to] - 1;

    CustomerTransactionsCounter storage customerTransactionsCounter =
        mappingCustomerTransactionsCounter[from][index];

    for (uint i = 0; i < amount; i++) {
        customerTransactionsCounter.numberOfTransactions = customerTransactionsCounter.numberOfTransactions + 1;
        uint restOfDivison = customerTransactionsCounter.numberOfTransactions % TRANSACTIONS_NUMBER_RULE;
        //check whether the user is apt to receive token or not
        if (restOfDivison == 0) {
            _mintInternal(from, TOKEN_AMOUNT_RULE);
            uint tokenId = getRetailerTokenId(from);
            _safeTransferFrom(from, to, tokenId, TOKEN_AMOUNT_RULE, "");
            emit Transfer(from, to, tokenId, TOKEN_AMOUNT_RULE, block.timestamp);
        }
    }
}
```

Figure 6.7 - StampCars Mechanism contract – transfer tokens

Although the contracts (points and stamp cards) share many similarities, the first key difference lies in how we store the customer transaction counter data. Like the points mechanism, we use a mapping to store the customers' transaction counters, where the key is the retailer's address. However, accessing this data can become cumbersome as we must iterate through the array until we find the customer we want. To avoid this potential increase in fees and ensure better performance, we

implemented a second mapping. For each retailer's address, this mapping associates an inner mapping where the key is the customer's address, and the value is the index position of the customer in the array of the retailer's customer transaction counter data.

With this approach, we can quickly retrieve the position of the customer in the specific retailer's customer transaction data array, resulting in faster access and improved performance. To achieve this, we first provide the retailer's address to obtain the inner mapping. Once we have this map, we then provide the customer's address to finally obtain the position of the customer in the specific retailer's customer transaction data array.

Despite the core of the transfers are different, the execution is similar to the points mechanism. Initially, we determine the sender type. If the sender is a retailer, we check the customer's data and verify if they have ever made a transaction. If no previous transaction exists, we create a *struct* object and save it to the array while retrieving the index position. As a retailer can send multiple stamps (representing multiple transactions, such as if the customer made two purchases or brought a friend for a haircut, resulting in two stamps), we need to check one by one if the customer is eligible to receive the token and increment his or her transaction counter accordingly. When the customer is eligible, the retailer mints the token and sends it to the customer's wallet. On the other hand, for the customer, since there is no logic associated with sending the rewards, the transfers are automatically carried out, transferring the token to the retailer's wallet.

Again, this contract approach is based on the logic behind the stamp cards, where a customer receives a stamp for each transaction. In this case, a retailer can send one or more stamps for each transaction, depending on the scenario. As for the customer logic, once we have a "card full" (the required number of stamps was reached), we can redeem the reward whenever we want, allowing us to send the token in exchange for something in the retailer's store.

6.1.3.3 Loyalty Manager contract

The Loyalty Manager contract serves as a centralized hub for handling all incoming requests within the smart contracts' ecosystem. Its purpose is to facilitate meta-transactions, enabling customers to transact without incurring transaction fees, and it acts as a mediator to interact with mechanisms contracts. The Loyalty Manager is entrusted with several key responsibilities, including managing the registered retailers in the system, associating each loyalty program retailer with their respective loyalty mechanism contract, storing and handling customer subscriptions, and, finally, maintaining a registry of mechanisms contracts and efficiently redirecting requests to the appropriate one. Figure 6.8 shows the Loyalty Manager contract, illustrating the defined variables along with the event that is emitted whenever a new retailer is registered in the system.

```
contract LoyaltyManager is ERC2771Context {

    uint retailersRegisterdCounter;

    //adress = retailer address
    mapping (address => uint) retailersKey;
    //uint reatailer Key
    mapping (uint => RetailerInfo) registeredRetailers;

    //adress = customer address
    mapping (address => RetailerInfo[]) customersSubscriptions;

    mapping (LoyaltyMechanismType => LoyaltyMechanism) public loyaltyMechanismContracts;

    enum LoyaltyMechanismType {POINTS, STAMP_CARDS}

    struct RetailerInfo {
        address retailerAddress;
        string ipfsHash;
        LoyaltyMechanismType loyaltyMechanismType;
    }

    /*
    ****EVENTS*****
    */
    event NewRetailer(address indexed retailerAddress, string ipfsHash, uint loyaltyType);
```

Figure 6.8 - Loyalty Manager contract

Each retailer in the system is represented by a *RetailerInfo* structure, consisting of the retailer's address, a string representing the IPFS hash of the retailer data, and the loyalty mechanism type associated with the retailer's loyalty program. The loyalty mechanism type is defined as an enumeration (*enum*), where each value corresponds to a specific mechanism. To store the mechanisms contracts, the Loyalty Manager utilizes a mapping where the keys are values from the enumeration *LoyaltyMechanismType*, and the corresponding values are references to contracts stored in the form of the Loyalty Mechanism interface. This approach leverages polymorphism and inheritance, aligning with the pattern described in Section 2.1.4, which covers storing rules and adding them at runtime. In Solidity, enumerations are converted into integers during compilation. For instance, if "Points" is the first enumeration value defined, it will be converted to zero, and subsequent values will follow the same pattern. By employing references to the loyalty mechanism interface, the Loyalty Manager contract only recognizes the four defined methods, thereby enhancing security and preventing access to other functions that may be defined in the contracts. This approach ensures the code remains dynamic, eliminating the need for nested "if" statements to determine the correct contract invocation and removing the necessity for the loyalty manager to have knowledge about implementation details. Consequently, an abstract way of communication is established, streamlining the contract's interactions.

Regarding the storage and management of retailers, we applied a similar approach to the one used for tokens identifiers in loyalty mechanism contracts. When a retailer registers in the system, the Loyalty Manager assigns a unique system identifier, which is then stored in a mapping with the retailer's address as the key. Additionally, we

created another mapping where the identifier serves as the key, and the corresponding value is an object of the *RetailerInfo* structure. This approach was chosen to determine if a retailer is new and to keep track of the total number of retailers in the system. The assigned identifier corresponds to the current number of retailers in the system plus one, ensuring that when a new retailer is added, their identifier matches the retailer counter (e.g., if there are ten retailers, the new retailer's identifier will be eleven). By accessing the *retailersKey* mapping with the retailer's address, we can check if a user is already registered based on the default value (zero) defined by Solidity for unsigned integer (*uint*) mappings. If the value obtained is different from the default value, it means the user is already registered. Once the retailer is assigned an identifier , we can effortlessly add an object representing their data. Furthermore, this approach becomes necessary because Solidity mappings do not offer features like obtaining all values or keys directly. Since the key of the second mapping is a *uint* and we know the total number of retailers in the system, we can conveniently iterate with a loop to obtain all registered retailers in the system.

As customers can have multiple subscriptions, we implemented a mapping to store this information. The mapping uses the customer's address as the key and stores an array of *RetailerInfo* structure objects as the value representing the subscribed retailers. We use the event related to the new retailer as a log entry. This log is stored in the blockchain log system, allowing us to retrieve it later or determine when the action was completed. It is important to note that blockchain transactions are not instantaneous since they require the consensus mechanism to approve them before being confirmed and recorded. Events serve as a solution to track and know when they have been executed.

To enable meta-transactions, the contract inherits from the abstract contract *ERC2771Context* (Figure 6.9), which requires a forwarder contract as a parameter. This abstract contract ensures that incoming requests are from our designated forward contract and retrieve the transaction information sent by the forwarder. As stated in our smart contract architecture, we utilized the *MinimalForwarder* contract, which is provided by OpenZeppelin, and similarly, the *ERC2771Context* is also provided by OpenZeppelin. Figure 6.9 illustrates the constructor of the Loyalty Manager, where we define the forwarder contract. The constructor of the Loyalty Manager takes two parameters: the forward contract and a loyalty mechanism contract. The forward contract is stored using the *ERC2771Context*, while the loyalty mechanism contract is stored in a mapping. It is worth noting that the loyalty mechanism sent is always the points mechanism, which is why the key used in the mapping corresponds to the *POINTS* value of enumeration *LoyaltyMechanismType*.

```
constructor(MinimalForwarder forwarder, LoyaltyMechanism loyaltyProgramPoints) // I
    ERC2771Context(address(forwarder))
{
    retailersRegisterdCounter = 0;
    loyaltyMechanismContracts[LoyaltyMechanismType.POINTS] = loyaltyProgramPoints;
}
```

Figure 6.9 - Loyalty Manger contract – constructor

As already mentioned, the Loyalty Manager is responsible for two main tasks: 1) storing and managing information about retailers and customers; and 2) redirecting requests to the appropriate loyalty mechanisms contracts. With these objectives in mind, transactions related to loyalty programs and loyalty mechanisms follow a consistent two-step process. The functions related are the ones defined in the Loyalty Mechanism interface: mint, transfer tokens, balance, obtain the customers transaction counters, and set the retailer token identifier. The first step involves obtaining the relevant loyalty mechanism, while ensuring that the caller is a registered retailer. If the caller is not a recognized retailer in the system, the transaction is halted and reverted. Once we successfully retrieve the loyalty mechanism, we access the mapping containing the loyalty mechanism contracts and fetch the associated contract. The second step entails calling the function corresponding to the specific request. For instance, if a user initiates a transaction to *mintTokens*, the Loyalty Manager calls the *mint* function of the respective contract. By following this approach, the Loyalty Manager effectively manages, and redirects transactions related to loyalty mechanisms and loyalty programs in straightforward manner. Figure 6.10 shows the *mint* function.

```
function mintRetailersToken(uint amount) external {
    LoyaltyMechanismType loyaltyMechanismType = getLoyaltyMechanismType(_msgSender());
    loyaltyMechanismContracts[loyaltyMechanismType].mint(_msgSender(), amount);
}
```

Figure 6.10 - Loyalty Manger - mint function

All functions of the Loyalty Manager contract, which are briefly described in Annex C are not loyalty mechanism related, and they were implemented to fulfill the responsibility of managing and storing the retailers and customers.

6.1.4 Unit Tests

To ensure the contracts' optimal behavior and validate the successful implementation of requirements, comprehensive unit tests were conducted. Unit tests are essential components of the testing process, aiming to verify the correctness and reliability of isolated and independent code segments. Their primary purpose is

to identify potential problems in specific points of the code and examine how the code behaves under various scenarios. During unit testing, each code unit, such as functions or methods, is tested independently of the rest of the codebase. By doing so, developers can ensure that individual units perform as expected and meet the specified requirements. This approach helps detect bugs and issues early in the development process, leading to more robust and maintainable code. Unit tests are designed to cover different scenarios and edge cases, which might include valid inputs, invalid inputs, boundary conditions, and exceptional scenarios. Through testing, developers gain confidence in the correctness of their code and reduce the likelihood of unexpected failures or errors in the final product. Additionally, unit tests serve as a form of documentation, providing clear examples of how each code unit should behave and function in various situations.

Table 6-2 presents the tests that were performed including the name, the contract target and result. Despite most of these tests succeeding on the first try, we had conducted prior manual tests on various scenarios during the code writing process. The stampcards contract, being the most recent one, exhibited some bugs, particularly in the token transfer and the incrementing of customers' transaction counter. This led to a modification in our approach to store the customers' transaction counter.

6.2 Mobile application

Although the mobile application is not directly responsible for handling the business logic, it plays a crucial role in product marketing as it serves as the primary interaction point between users and the system. Therefore, it is essential to implement a well-designed mobile application that can effectively cater to users' needs and offer a seamless user experience.

As the developed loyalty management system prototype is a decentralized system, the application interface becomes one of its two core modules (i.e., its *frontend*). Ensuring a user-friendly and intuitive interface is paramount to enable users to interact with the decentralized system efficiently and access its functionalities effortlessly. A well-designed mobile application enhances user engagement, fosters adoption, and contributes to the overall success of the decentralized system.

Table 6-2 - Unit tests for contracts

Test	Name	Contract	Result
1	Assigning a retailer token	Stampcards	Success
2	Mint tokens	Stampcards	Success
3	Mint must emit an event	Stampcards	Success
4	Transfer tokens	Stampcards	Success
5	Number of customer transactions increases	Stampcards	Success
6	Transfer 2 stamps when the user has 9, should receive a token	Stampcards	Success
7	Transfer 2 stamps when the user has 8, should receive a token	Stampcards	Success
8	Customer transfers tokens to a retailer	Stampcards	Success
9	Assigning retailer token	Points	Success
10	Mint tokens	Points	Success
11	Mint must emit an event	Points	Success
12	Transfer tokens	Points	Success
13	Retailer should mint when transferring a bigger amount than balance	Points	Success
14	Number of customers transactions increases	Points	Success
15	Customer transfers tokens to a retailer	Points	Success
16	Add new loyalty mechanism at runtime	LoyaltyManager	Success
17	Add new retailers	LoyaltyManager	Success
18	Add a retailer already registered should revert	LoyaltyManager	Success
19	Retrieve information of non-registered retailer should revert	LoyaltyManager	Success
20	Retrieve all retailers	LoyaltyManager	Success
21	Subscribe to a retailer	LoyaltyManager	Success
22	Subscribe twice to a retailer should revert	LoyaltyManager	Success
23	Subscribe to a non-retailer should revert	LoyaltyManager	Success
24	Retrieve all retailers excluding the customer subscriptions	LoyaltyManager	Success
25	Retrieve customer subscriptions	LoyaltyManager	Success
26	Redirect mint request	LoyaltyManager	Success
27	Mint request from a non-retailer should revert	LoyaltyManager	Success
28	Transfer tokens	LoyaltyManager	Success
29	Transfer between two customers should revert	LoyaltyManager	Success
30	Transfer tokens with a customer not subscribed to the retailer should revert	LoyaltyManager	Success

6.2.1 Mobile application architecture

The mobile application, being one of the core modules of the system, was meticulously designed to adhere to the principles of object-oriented programming and follow Android's guidelines for efficiency [107]. With this in mind, the architecture created for the mobile application is illustrated in Figure 6.11.

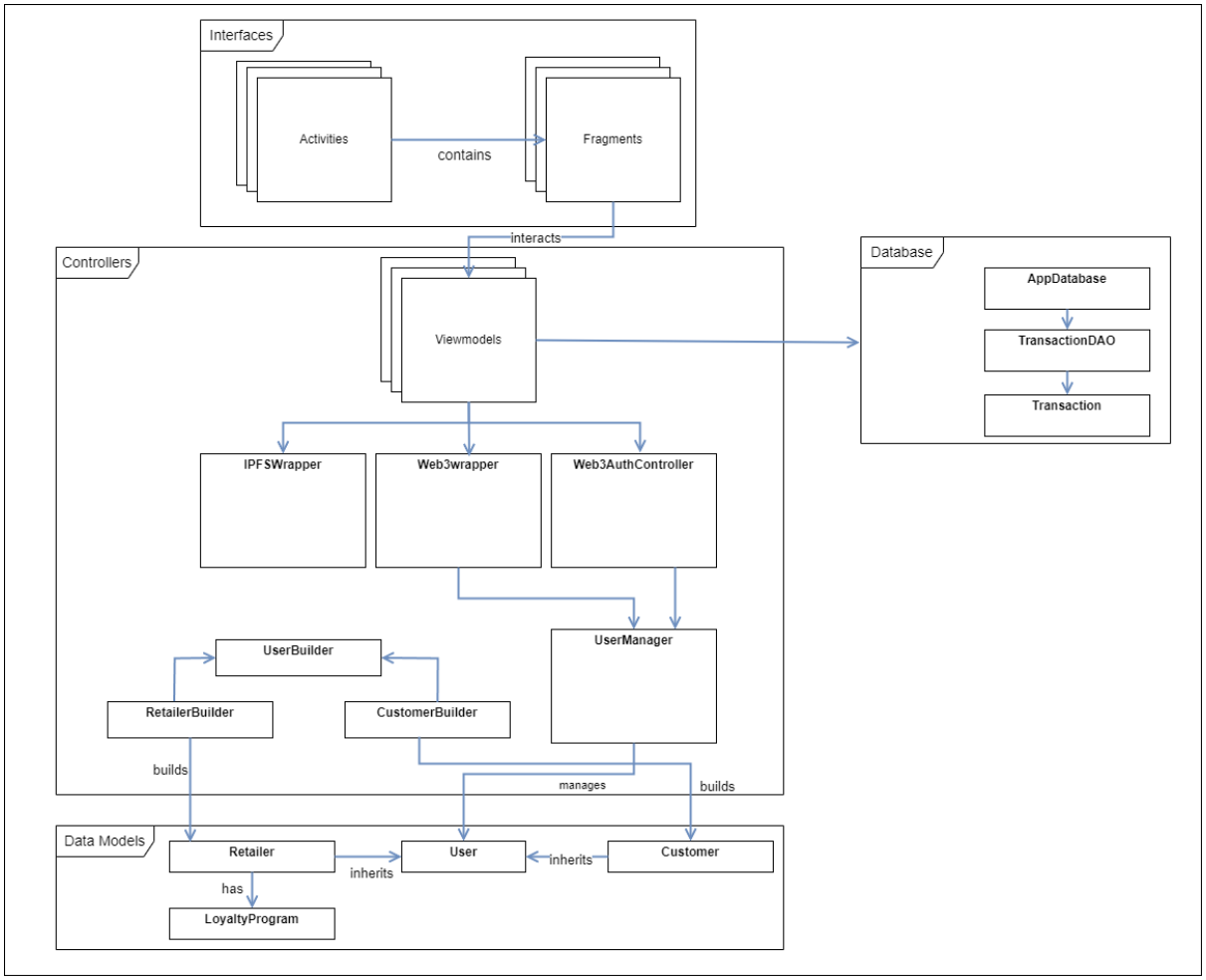


Figure 6.11 - Mobile application architecture

The architecture of the mobile application is thoughtfully divided into four key segments to ensure efficiency and adherence to object-oriented programming and Android guidelines. The first segment comprises interfaces responsible for displaying content on the screen, creating a user-friendly and intuitive experience. The second segment consists of controllers, each dedicated to handling specific tasks and functionalities efficiently. The third segment revolves around data models, which represent and manage the data within our system. Lastly, the fourth segment encompasses a local database responsible for storing transactions locally, optimizing data access and management. This well-structured architecture helps create a robust

and responsive mobile application, meeting the users' needs and providing an enhanced user experience.

Following the Android guidelines for development, the interfaces in our application are structured using fragments within activities. Instead of launching a new activity every time a new screen is required, the activity switches the displayed fragment, offering better performance and efficiency. This approach minimizes resource usage compared to creating new activities. Adhering to object-oriented programming principles, we implemented view models to separate the logic from the views, ensuring that views solely manage the screen presentation. View models act as a bridge between the screens and the data, abstracting requests, and dividing responsibilities effectively. This layer facilitates abstraction, enhancing the overall design and functionality of the mobile application.

Due to various services utilized in our application, such as Web3Auth, IPFS, and blockchain interaction, distinct controllers have been implemented to handle specific tasks. The IPFSWrapper controller focuses solely on interactions with the IPFS network, facilitating data retrieval and uploads. On the other hand, the Web3AuthController is responsible for Web3Auth service interactions, enabling login through various web 2.0 credentials and obtaining user wallet and information. Lastly, the Web3Wrapper controller manages interactions with the blockchain. To streamline implementation, all controllers are designed using the singleton pattern, ensuring only one instance of each is created during the application's lifecycle, as a single service of each type suffices due to their generic nature.

For user management, we have developed a user manager controller responsible for creating and storing a user. The user manager also follows the singleton pattern but incorporates a factory pattern. This factory pattern becomes necessary due to the two types of possible users in the application. Additionally, for user creation during login, a builder pattern is used to create users with limited data, ensuring efficient and secure handling of user information. The Builder pattern is a creational design pattern that aids in constructing an object gradually, step by step. It becomes essential in situations where not all information is immediately available. As a result, the pattern constructs the object with the currently available information, and upon obtaining the remaining information, it finalizes the creation process. Depending on the user type, the factory returns an object of a specific builder for that type and is responsible to construct the user.

In the application, we have four data models. The LoyaltyProgram model represents a loyalty program, which a Retailer model possesses. Both the retailer and customer data models are representations of a user since they inherit user capabilities. However, it is not possible to instantiate an object of the user type directly; it must always be either a customer or a retailer.

Although not mandatory, the local database became essential due to the nature of the blockchain. Storing the transactions a user has performed in the local database

allows us to achieve optimal performance and reduce loading times. It is worth noting that this local database exclusively focuses on storing transactions data.

6.2.2 Requirements

The requirements associated with android application implementation are presented in Table 6-3.

Table 6-3 - Mobile app requirements

Requirements	Name	Description
R-1	Support meta-transactions	The mobile application must allow a customer to create a meta-transaction.
R-2	Login	The mobile application must have a login system with Web 2.0 credentials.
R-3	Create a loyalty program	A retailer must create and configure his or her loyalty program, on the first-time logging in.
R-4	Home page	The user needs a Home page so he can quickly access important features and information.
R-5	Display QR Code	The app has to create a QR Code for a user to display receive transfers.
R-6	Read QR Code	The app must be able to read a QR Code, to perform transfers.
R-7	Transfers tokens	The mobile application must have a mechanism to allow the tokens transfer.
R-8	Transaction history	The app has to display the transaction history so a user can quickly track and view his or her transactions.
R-9	Discover new retailers	The app must show a catalog of the retailers not yet subscribed.

6.2.3 Implementation

The mobile application (app) was developed with a focus on two main concepts: interfaces and business logic. The app comprises five activities, each dedicated to presenting different types of information. According to the architecture explanation,

these activities contain a set of fragments to display the UI, leading to smoother and more efficient screen transitions.

Regarding the business logic, the application incorporates controllers to handle various external services. These controllers are implemented as singletons to ensure that only one instance is active during the application's lifecycle. This approach prevents data concurrency issues, reducing the occurrence of execution errors.

The implementation of the mobile application will be detailed in the following order: interfaces, controllers, and local database. It is important to mention that the system was developed and tested on a physical device rather than an emulator, ensuring real-world usability and performance assessment.

Table 6-3 Continued - Mobile app requirements

R-10	Discovery map	The app must show a map with the stores location not yet subscribed.
R-11	View a loyalty program	The customer can click on retailer and view the business and loyalty information.
R-12	Subscribe a loyalty program	The customer subscribes a retailer.
R-13	View subscriptions	The customer has a list of his or her current retailers' subscriptions.
R-14	Buy cryptocurrency	The retailer needs an option to buy cryptocurrency inside the app, facilitating the process.
R-15	View cryptocurrency balance	The retailer must see his or her current balance of cryptocurrency.
R-16	View loyalty balance	The customer must see his or her current balance in a loyalty program.
R-17	Menu navigation	The user must have bottom menu navigation to navigate through the app features.

6.2.3.1 Interfaces: Activities and fragments

In mobile development, screens are typically defined as activities. An activity serves as a component that provides a window for the application to draw the user interface (UI). Generally, each activity corresponds to one screen, representing a single view. However, the arrangement and management of these views can be customized, and this is where fragments come into play. Fragments are reusable portions of UI components, each with its own lifecycle and methods for management. They need to be hosted by an activity, which means that fragments depend on activities. We

can think of the activity as a blank canvas, and the fragments are the components that form the UI on that canvas. Notably, a single fragment can occupy the entire viewport of the activity screen, allowing for flexible and adaptable user interface designs.

Keeping this in mind, we have defined five activities, each representing different screens with distinct features. The first one is the login activity, which serves as the screen where users interact to perform the login process. Secondly, we have the main activity, also known as the Home activity, which incorporates most of the application's features and includes multiple fragments. The Configuration activity is responsible for guiding retailers through the creation and configuration process of their loyalty program, which is launched when a retailer logs in for the first time and does not yet have a loyalty program. The CurrencyExchange activity is designed to offer users an interface for buying cryptocurrencies. However, this activity primarily acts as a host for the SDK provided by the Ramp Service, which contains its own interface and features for crypto purchases. In this case, we simply integrate and host the SDK, without implementing any actions or screens, as all necessary functionalities are already present in the provided SDK. Lastly, we have the TransferTokens activity, which provides users with a window to perform token transfers. This activity enables seamless and efficient token transfer operations within the application.

Regarding fragments, only two activities, namely the Home and Configuration activities, incorporate them. This choice is based on the fact that the other activities have simple interfaces and do not require navigation as they remain static throughout. In contrast, the Home and Configuration activities entail numerous steps and features, necessitating a more sophisticated implementation and user experience. Hence, these activities are constructed using fragments. All the fragments created occupy the entire viewport, implying that each fragment constitutes a complete view. As a result, only one fragment is visible at a time, ensuring a seamless and focused user interface. Figure 6.12 presents a visual representation of the fragments within activities.

As depicted in Figure 6.12, the Login, CurrencyExchange, and TransferTokens activities do not contain any fragments. The Login and TransferTokens interfaces are implemented by us, whereas the CurrencyExchange activity utilizes the SDK from Ramp, which already includes the required interfaces. The reason for Login and CurrencyExchange being standalone activities is due to their straightforward features and lack of navigation requirements. On the other hand, the Home and Configuration activities possess multiple features and require navigation, making fragments the ideal approach for implementation. The Configuration activity comprises three fragments, each occupying the entire viewport. The first fragment, BusinessConfiguration, serves as a screen for retailers to provide business-related information. The second fragment, BusinessLogoConfiguration, enables users to select a photo from the gallery and use it as their business logo. Lastly, the LoyaltyConfiguration fragment presents a form for users to fill out loyalty

descriptions and select the intended loyalty type. Additionally, it offers actions to pick photos from the gallery to be displayed when customers view the loyalty program.

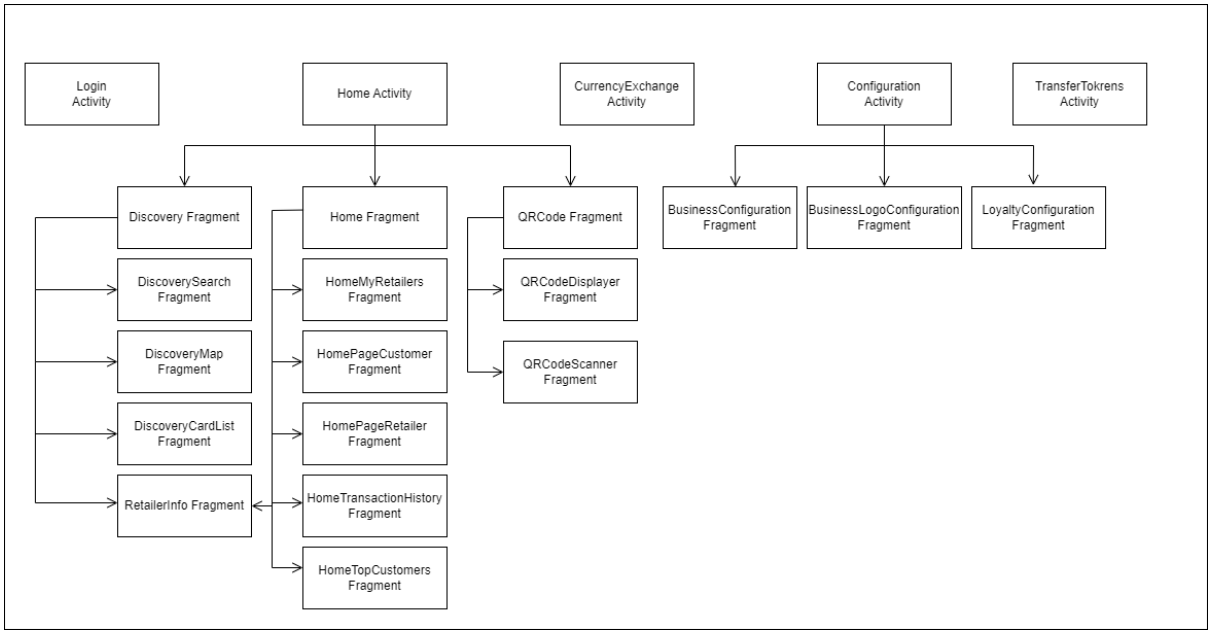


Figure 6.12 - Mobile app activities and fragments

The Home activity serves as the main activity, encompassing a wide array of features, making it appropriate to divide them into several fragments. The Home activity is grouped into three main sections: Discovery, Home, and QR Code. These primary groups house various other fragments and can be conveniently switched through a navigation menu.

The Discovery group comprises three fragments: DiscoveryCardList, which displays a list of retailers not subscribed in a card format, DiscoverySearch, which enables users to search for retailers by name in a tiled list format, and DiscoveryMap, which displays a map with marked locations of retailers not subscribed.

The Home group is hosted within the Home Fragment, which dynamically loads either the HomePageCustomer or HomePageRetailer fragments based on the logged-in user type. Both fragments contain a section displaying the last five transactions made. Additionally, HomePageRetailer includes a section for displaying the top customers, while HomePageCustomer displays the subscribed retailers. The Home Fragment also hosts HomeTransactionHistory, which lists all transactions in tile form from the most recent to the oldest, HomeMyRetailers, which lists all retailers a customer is subscribed to, and HomeTopCustomers, which lists all customers and their respective number of transactions.

The QR Code group is comprised of two fragments: QRCodeDisplayer, which displays the QR Code for reading, and QRCodeScanner, which utilizes the camera

view to scan QR Codes. These fragments enable seamless QR Code interactions within the Home activity, enhancing user experience and convenience.

Due to the varying requirements and features for different user types, customers and retailers have access to different fragments. For example, retailers do not access the discovery fragments, and, in the Home fragments, they do not have access to HomePageCustomer and HomeMyRetailers, as these interactions are not applicable to them. Conversely, customers have access to three out of the five activities. CurrencyExchange and Configuration activities are exclusive to retailers' features. Regarding the fragments, the only ones that customers cannot access are HomePageRetailer and HomeTopCustomers. This ensures that each user type has a tailored and streamlined experience, with access to only the relevant functionalities and information.

6.2.3.2 Controllers

As outlined in the mobile application architecture, the creation of controllers becomes essential to separate the responsibilities of interfaces from data retrieval and management. This adheres to the principles of object-oriented programming and decouples responsibilities, distributing them among different entities. This approach allows interfaces to solely focus on their presentation aspects, while the controllers are responsible for obtaining the necessary data. To further decouple responsibilities, multiple controllers were created, with each one handling a specific set of related tasks. All these controllers are implemented as singletons, as there is no need for multiple instances of these objects, which could potentially lead to concurrency issues. The controllers include ViewModels, IPFSWrapper, Web3Wrapper, and Web3AuthController. Each controller plays a distinct role in managing various aspects of the application, ensuring a well-structured and efficient design.

ViewModels serve as state holders for the UI, allowing interfaces to be relieved from the responsibility of directly managing data. Instead, ViewModels ensure data integrity and longevity throughout the lifecycle of the application. By acting as a bridge between the user interface and data, ViewModels facilitate seamless data interactions. Activities and fragments request data from the ViewModel, which, in turn, delegates call to other controllers based on the specific requirements. Since the other controllers are implemented as singletons, they can be accessed throughout the entire application without needing explicit references. This contributes to a smoother and lighter implementation process, reducing complexities and dependencies. By following this design approach, the mobile application can efficiently manage data and avoid being highly dependent.

IPFSWrapper is the dedicated controller responsible for all IPFS-related operations. Whether it is uploading or retrieving information from IPFS, this controller handles the interactions seamlessly. It encompasses four methods: *addFile*, *getFileData*, *uploadImage*, and *getImageFile*. To facilitate the interaction with IPFS, we utilized an open-source library called ipfs-api-kotlin [108]. This library enables easy integration

without the need to build all the requests from scratch, streamlining the implementation process and ensuring efficient communication with the IPFS network.

Web3AuthController is the controller responsible for facilitating user login using Web 2.0 credentials. This controller efficiently manages the login process, handling result callbacks, and user initiation. To enable this login system and provide users with wallets, we integrated the native SDK of the Web3Auth service. This seamless integration ensures a smooth login experience and offers a wallet to users of all levels of knowledge. During the login process, once the Web3Auth service returns a response with user information and wallet details, this controller, along with the UserManager, creates and initializes the user for the mobile application. Regardless of the user type, the login process results in the creation and instantiation of the user in the UserManager. The UserManager employs a built-in factory, which constructs the specific user type accordingly. Following the user creation, if the user is a customer, the login process concludes and proceeds to the home page. For retailers, the process continues with retrieving the retailer information from the blockchain. If this information is empty, the retailer is directed to configure the loyalty program. Conversely, if the retailer already has a loyalty program, they are directed to the HomePage. This streamlined process ensures a seamless and efficient user experience based on their respective roles.

The UserManager controller is implemented as a singleton with a straightforward responsibility: to create and store the user data throughout the entire app session. It incorporates a built-in factory to provide the appropriate builder for the user type. Upon login, the UserManager processes the login result and creates the user based on the received data. By utilizing the UserManager controller, the app can access the user information from any location without the need to store direct references to it. This design ensures a centralized and efficient way of managing user data, enhancing data integrity and accessibility throughout the application.

The Web3Wrapper controller is a vital singleton, developed using the web3j library [109], which allows precise interaction with Ethereum-compatible blockchains. It provides highly modular, reactive, type-safe Java, and Android features for seamless smart contract interactions. Given that the prototype is a decentralized application, the Web3Wrapper serves as the core of the app, because if the app fails to interact with the blockchain, the mobile application has no purpose. In this controller, we have defined all the necessary requests for blockchain interaction using the library's capabilities. These requests are based on the functions defined in the smart contract Loyalty Manager, which serves as the entry point to the system.

Additionally, Web3Wrapper includes a method to retrieve transaction events from the blockchain log. This approach allows us to access all transfers without the need to store them in the smart contract. However, it is important to note that this method can become inefficient as the blockchain grows, with an increasing number of blocks to search through for events. To address this issue, we have implemented an

optimization strategy. Instead of iterating through all blocks to find specific events, we start the search from the block where the contract was created. This assumption allows us to disregard the blocks before the contract's existence, improving performance. Furthermore, to avoid reiterating every time we search for new transactions, we store the already found transaction events locally and start from the block of the latest transaction. This approach minimizes the search period and enhances overall performance. Despite these optimizations, it is essential to consider that as the blockchain continues to grow exponentially, and the distance between the last searched block and the current block increases significantly, which may lead to potential congestion in the mobile application. Therefore, continuous monitoring and potential adjustments are necessary to ensure smooth functioning with the evolving blockchain data.

6.2.3.3 Local database

The purpose of this database is to significantly improve performance by efficiently storing transactions, eliminating the need to iterate through the entire blockchain to retrieve them. By saving already-found transactions locally, subsequent searches can start from where they left off, rather than from the beginning, resulting in faster and more efficient searches. For this local data storage, we utilized RoomDatabase [110], the recommended approach according to Android documentation [110] for running a lightweight local node (SQLite) on smartphones. Since our system does not rely on a database as a core component, the implementation is straightforward. The database comprises three main components: a RoomDatabase instance, a Data Access Object (DAO) that defines queries to interact with the database, and an entity representing each transaction and storing its relevant field values. It is important to note that this database is not essential to the decentralized system, so even if a user clears the database data, the system will continue to function as intended. The primary objective of implementing this database is to enhance user experience by reducing loading times and providing a more seamless app performance.

6.2.4 Unit Tests

Given the decentralized nature of the prototype's architecture, where we have only a mobile application and smart contracts on a network, it is crucial to ensure that the Web3Wrapper, which is responsible for blockchain interaction, is free from errors. To achieve this, we have implemented unit tests for this controller. Table 6-4 provides an overview of the unit tests conducted to validate the functionality and reliability of the Web3Wrapper. Despite not all tests succeeded on the first try, they contributed to enhancing the resilience of the functions and introducing exception handling when the results failed. Regarding the interface tests, they were manually performed during the development phase and also in the demos held at the weekly meetings The interface user experience was ensured by the WIT designers, who provided the refactored designs for the application.

6.3 Other implementation aspects

In order for the system to function properly, a local IPFS node was installed. This node connects with other peers in the network to share files. Once the node is operational, it provides an interact point with an API URL to upload/download data to/from the node. By default, this API URL points to the local environment (i.e., IP address 127.0.0.1). To make it accessible from other devices, we had to modify the configuration file to use the IP address of the machine in the network. However, it is important to note that this IP address is not publicly accessible, meaning users outside the local network cannot access it. The API of IPFS node is only accessible within the same network. If the node is running on one network and the mobile application is on a different network, it will not be able to access the IPFS node. While the API of the IPFS node is limited to functioning within the local network, the distribution of files within the IPFS network itself works flawlessly with external nodes. To enable access from outside the local network, one solution could be to open a port in the router and use the public IP of the machine as the new API IP address. However, for the purpose of the prototype and considering that it is not mandatory to have external access, this step was not implemented as it would not provide any significant benefits to the system.

Table 6-4 - Unit tests for Web3Wrapper controller

Test	Name	Result
1	Mint without being a relayer should throw exception	Success
2	Get retailer info	Success
3	Register new retailers	Success
4	Register a retailer that is already registered should throw exception	Success
5	Retrieve all retailers	Success
6	Get currency balance	Success
7	Get loyalty balance	Success
8	Mint tokens	Success
9	Assigning retailer token	Success
10	Mint tokens	Success
11	Subscribe a retailer	Success
12	Subscribe twice a retailer should throw exception	Success
13	Retrieve customer subscriptions	Success
14	Transfer tokens as retailer to a customer	Success
15	Transfer tokens as customer to a retailer	Success
16	Get customer transaction counter	Success
17	Get transaction history	Success

To enable meta-transactions using OpenZeppelin, we must create a script for the Autotask component in OpenZeppelin Defender. This Autotask will provide the webhook URI for the application to send requests signed by a customer. However, this Autotask requires a script to execute the redirection to the forwarder contract. In this script, we obtain the credentials of the Relay, which acts as a wallet, and then send the transaction to the forwarder contract. It is worth mentioning that we have access to the Relay's credentials because we linked the Relay with the Autotask during its creation. For the complete script source code, refer to Figure 6.13.

```
const ethers = require('ethers');
const { DefenderRelaySigner, DefenderRelayProvider } = require('defender-relay-c

exports.handler = async function(event) {
  if (!event.request || !event.request.body) throw new Error(`Missing payload`);
  const { request, signature } = event.request.body;
  console.log(`Relaying`, request);

  const ForwarderAddress = "0x5F3857658d828C7c2dfa6450EA6076dB0CE58F73";
  const ForwarderAbi = [{"inputs":[],"stateMutability":"nonpayable","type":"cons

  // Initialize Relayer provider and signer, and forwarder contract
  const credentials = { ... event };
  const provider = new DefenderRelayProvider(credentials);
  const signer = new DefenderRelaySigner(credentials, provider, { speed: 'fast'
  const forwarder = new ethers.Contract(ForwarderAddress, ForwarderAbi, signer);

  // Relay transaction!
  const tx = await forwarder.execute(request, signature);
  console.log(`Sent meta-tx: ${tx.hash}`);
  return { txHash: tx.hash };
}

```

Figure 6.13 - Script for the Autotask

7 DISCUSSION

The proposed system aims to manage various loyalty programs with different mechanisms. Even though the blockchain solves significant problems, it is impossible to address concerns related to rewards and personal preferences due to inherent human characteristics.

The proposed loyalty management system offers opportunities and can reshape how traditional loyalty systems are implemented. It provides new paradigms and possibilities to users with any level of knowledge about technology to create and interact with the blockchain without having the inconvenience of learning it. This way, users can easily participate and enjoy the loyalty programs of their favorite brands.

The system works on the Polygon PoS chain, which implies the use of Solidity as the smart contract programming language. Polygon also includes a network for testing purposes (which is useful while developing the system) and offers ways to track transactions.

A mobile application is available to interact with the blockchain. This application abstracts its users by setting up all the requirements for them. Additionally, the way smart contracts are designed allows for a more manageable infrastructure of loyalty programs.

With the ERC-1155 token interface standard, each retailer can mint its own tokens and be responsible for their generation, thus gaining better control and ownership. Whenever a customer purchases something, the retailer mints the desired number of tokens and gives them to the client. In order for the customer to be able to redeem rewards from a retailer, he/she needs to have a certain number of tokens from that retailer.

The designated external services are intended to produce a better user experience. Web3Auth is used to grant the users the ability to join the system with any credentials they have from Web 2.0, while managing the wallet and keys for user interaction. Ramp service is used to accommodate users when they need to buy cryptocurrencies. It offers an easy access point for currency exchange without leaving the application. OpenZeppelin provides relays that are used to diminish entry barriers for customers, as blockchain requires mastery, experience, and cryptocurrency to interact with. Since customers do not adhere to applications for which constant payment is required, the system acceptance would fail without the use of meta-transactions.

With the proposed loyalty management system, the users apply their previous Web 2.0 credentials and do not have to enter new information. Furthermore, the system does not include any database, which means that personal data will not be stored.

7.1 Comparison with other systems

There are several advantages that can be highlighted when comparing the proposed loyalty system with the related works discussed in Section 3.1. The universal blockchain system [42] supports multiple loyalty programs and partnerships, which our system can incorporate. However, this system limits the loyalty mechanism to points and uses a backend server for the users' application to communicate. Additionally, it stores user data in a database. In contrast, the proposed system eliminates the need to store personal information. Moreover, it does not rely on any backend system and achieves a more decentralized approach.

The system proposed in paper [50] has similarities with the universal blockchain system, since both employ the points mechanism using tokens. However, the proposed system incorporates additional types of loyalty mechanics and abstracts the users from the concept of tokens, enabling them to interact with points in a manner similar to non-blockchain-based platforms. Another distinguishing factor is that the author in [50] employs the ERC 20 interface for token creation, which requires that each token exists as a smart contract, whereas the proposed system uses ERC-1155 for managing multiple tokens. Additionally, the expectation that users possess knowledge of Ethereum wallets in [50] may present challenges for novice users.

The loyalty program that utilizes the Waves blockchain and a mobile application [61] employs a token creator that generates a unique token for each retailer. However, this system also stores personal information and potentially creates a token contract for each retailer, similar to the aforementioned system [50].

The reward system in [64] uses tokens that can be converted into points, and the platform is specifically designed for the FMCG industry, limiting its scope to this domain. However, this system resembles our transaction process, in that it employs QR codes to initiate the transfer of assets.

Although the systems proposed in [65], [66] do not directly aim to manage loyalty programs, they improve these programs by facilitating the exchange of assets between different loyalty programs. Currently, our loyalty management system lacks features that allow for partnerships between retailers.

Since the existing loyalty systems discussed in Section 3.1 do not specify their authentication process, it is assumed that they have created one and currently manage it. This results in an additional step for users onboarding, as they have to enter new information to create a new account. In contrast, the proposed system employs Web3Auth, which uses Web 2.0 credentials to streamline the authentication process. Additionally, a significant advantage of the proposed system is the use of meta transactions, which provide a superior user experience by eliminating the need for users to pay blockchain fees.

When comparing our prototype with other implementations in the market as described in section 3.2.2, we find similarities with the Joyn [84] and Loopy Loyalty [79] systems. These similarities stem from the shared focus on retailers with smaller operations. Additionally, all three systems offer easy, quick, and instant ready-to-go loyalty program creation, while capitalizing on mobile environments. However, our system stands ahead due to its capability of accommodating multiple loyalty mechanisms, unlike Joyn and Loopy Loyalty, which are limited to just one mechanism. Joyn does not utilize blockchain technology, suggesting a centralized approach. As for Loopy Loyalty, the available information about blockchain implementation is limited to using it for storing transactions. Additionally, these systems require a registration process, which leads to users giving personal information.

Regarding the other systems in the market (Qiibee, Loyyal, and Appsolutely), each system has its distinct scope. While Qiibee, Loyyal, and Appsolutely aim to accommodate retailers of all sizes, Loyyal specifically targets higher-level retailers. These systems share the common goal of creating a unique ecosystem for multiple loyalty programs. However, they also offer additional features, such as creating application interfaces for loyalty programs, allowing token exchanges for real money or other cryptocurrencies, and providing customers the freedom to exchange between them different tokens. One key difference between our system and the others lies in the transactions. Currently, our system requires a presence of retailer for transactions, which limits its flexibility compared to the other systems. Additionally, each system utilizes different blockchains and leverages them in unique ways. Unlike our system, Qiibee, Loyyal, and Appsolutely utilize private blockchains to enhance security and transaction rates. However, this approach may introduce concerns similar to centralized environments, where control is consolidated within a single entity or group. For instance, while Qiibee uses the Ethereum network for hosting their token's smart contract, the business logic remains within their private blockchain.

Regarding meta-transactions, there is no available information for the other systems. Furthermore, we can assume that these systems do not utilize the ERC-1155 standard for managing multiple tokens. This is evident as Loyyal uses Hyperledger, which is not compatible with ERC-1155 due to its development in Solidity. Similarly, Appsolutely follows a similar approach with the NEM blockchain. While Qiibee's private blockchain is EVM compatible, the standard used for creating retailer tokens could be ERC-1155, ERC-20, or ERC-721. Additionally, all these systems employ registration processes that require additional personal information, which is not necessary in our application. It is worth noting that the system we propose is currently a prototype and had limited development time.

7.2 System benefits

The designed and implemented loyalty management system offers countless benefits for loyalty program ecosystems and helps to reshape the traditional approach for a more modern, intuitive, and secure model.

The main benefits of the system are the following:

- The platform fully embraces decentralization and leverages the power of the ERC-1155 token to efficiently manage tokens and reduce the number of contracts required;
- The system provides near real-time transaction speed, allowing users to redeem their rewards faster without having to wait too long;
- Due to the behavior of smart contracts, third-party tools and intermediaries are no longer necessary;
- The system helps to reduce costs. As intermediaries and third-parties are abolished, retailers can save money. Furthermore, the infrastructure needed to man-age the loyalty program is transferred to the blockchain;
- Blockchain offers a secure infrastructure;
- A single digital wallet can be used to store all retailers' tokens, eliminating the problem of managing rewards from multiple brands. It also gives complete control to the user;
- The platform accommodates all distinct levels of user knowledge, allowing more inexperienced people to have a smoother first contact with it;
- The platform can manage several types of loyalty mechanisms, increasing the diversity of loyalty programs. In addition, it welcomes any retailer. This makes it easier for new retailers to join and customers with more options to choose from;
- It provides a catalog with all available loyalty programs. On the one hand, the marketplace helps retailers gain greater exposure to attract more customers. On the other hand, customers have a better access point to all available loyalty programs, meaning they are able to discover new businesses;
- The system takes advantage of the fact that smartphones are increasingly present in people's lives, thus enhancing the overall experience.
- The system is capable of introducing new loyalty mechanisms at runtime. Moreover, it presents both upgradable and non-upgradable approaches, giving the entity responsible for the system the flexibility to choose the one that best suits system needs and preferences;

7.3 System limitations

The proposed system has great advantages. However, it also has some limitations:

- Due to human nature, it is almost impossible to solve concerns about rewards and personal preferences;
- Even though the system does not require any personal information, some information is required at the retailer's business level;
- The blockchain is a new subject, and this may affect outcomes. Because it is so new, it may be subject to impactful changes at any time, which can affect the system as it is currently designed, making it unsuitable for the currently intended purpose;
- Exchanging of tokens between two customers is not allowed. Furthermore, the tokens do not have a real market value;
- Retailers cannot form any partnerships, which limits the possibility of trading retailers' assets;
- The transaction process requires the physical presence of the retailer and the customer. The application does not have an item or service available for purchase;
- Transactions can only be executed in person between a retailer and a customer;
- Remote transactions are not currently possible within this system;
- The performance of the system is dependent on the blockchain performance. If the blockchain suffers congestion, the system also suffers from it;
- The system focuses on retailers of lower magnitude.

8 CONCLUSIONS AND FUTURE WORK

The proposed loyalty management system described in this document had the objective of developing a prototype to test how the blockchain technology could improve the loyalty marketing area. It aims to manage different types of loyalty programs and mechanisms, and includes several challenges in the technical domain, since blockchain is a recent technology and introduces new paradigms.

The system aims to overhaul the way in which traditional loyalty programs operate and attempts to eliminate several entry barriers and infrastructure issues. It aims to replace conventional physical cards, instead adopting the strategy of using the smartphone, an increasingly ubiquitous object in everyday life. The system also includes a universal mechanism that aggregates loyalty programs, thereby simplifying the user experience.

The usability of this system allows users to have better interaction with the brands that they appreciate the most, alongside true ownership over their rewards, as the system cannot remove them from their wallet. Additionally, the system provides a marketplace with all available loyalty programs in a single space, removing the need to have multiple applications and allowing management of all loyalty schemes at once. The system allows retailers to customize their own loyalty program without restrictions on the type of mechanism to implement. Furthermore, it exposes brands to new audiences, increasing the accessibility and reachability of the business.

The proposed loyalty management system reduces the problem of managing the loyalty program. It removes the responsibilities from the retailers' side, giving them more freedom to focus on satisfying customers and improving their loyalty program and business. It also removes concerns about the privacy of personal information since the system does not require it. From the customers' perspective, it offers a new way to explore and know new businesses without the burden of having to download new apps and repeatedly introduce their data. It offers an opportunity to have a first encounter with the blockchain without worrying about learning and understanding the process. Overall, the platform prioritizes the user experience more.

Although the system solves significant problems, it has also some limitations. It cannot control human aspects, such as preferences for rewards. The blockchain is still new and is constantly changing, which can impact the system. The system also presents limitations about partnerships among retailers and the transaction process. The transaction process only considers trades between a retailer and a customer and needs both to be present in the same room.

In general, the use of the proposed system is intuitive and simple. Its primary objective is to design a platform capable of managing different loyalty programs with distinct mechanisms. It presents a scalable solution, such that it is possible to add and expand its functionalities.

Overall, all the internship goals set for the loyalty management system prototype were successfully achieved, and we even published a scientific paper on this topic [7]. Additionally, during the development process, we expanded the number of tasks beyond the initial plan, thanks to our efficient implementation and planning. In summary, the internship was executed successfully, with all goals satisfactorily accomplished. However, the implemented prototype still requires further improvements to become a fully decentralized application ready for the market.

In terms of future work, there are several areas that require attention and improvement. First, we need to continue polishing and refining the application, as well as refactoring certain aspects to enhance its overall performance. Furthermore, certain features had to be limited during the development phase due to time constraints. For instance, transfers were restricted to require the presence of one retailer and one customer, and as a result, transactions between two customers or two retailers are not supported. These two remaining transactions behaviors should be fully implemented to provide a more comprehensive user experience.

A significant aspect of future work involves implementing the remaining loyalty mechanisms. Currently, only a subset of mechanisms is supported (Points, Stamp cards), and expanding this range will offer users more flexibility and choice in their loyalty programs. It is also crucial to devise a strategy to attribute real value to the tokens used within the system. While some potential solutions have been explored in this document, further research and experimentation are required to establish a robust valuation mechanism.

Continuing to explore new blockchain networks and technologies is another important step for improvement. Different networks may offer distinct advantages, and staying up to date with emerging technologies could lead to more efficient and scalable solutions. In terms of system management, while meta-transactions are currently supported, there may be challenges in the future. At present, the system covers all customers' transaction costs, but this could become burdensome as the app attracts a larger customer base. To address this, considering the implementation of a monthly subscription model for retailers could help offset costs and maintain the system's sustainability.

Lastly, developing a plug-and-play SDK would be a significant advancement. By providing other applications with easy integration access to our infrastructure, creating new loyalty programs would become faster and more streamlined, benefiting both the developers and end-users.

REFERENCES

[1] WIT Software, "WIT Software," *WIT Software*, 2023. https://www.wit-software.com/ (accessed Jul. 04, 2023).

[2] ISEC, "ISEC," *ISEC*, 2023. https://www.isec.pt/ (accessed Jul. 04, 2023).

[3] WIT Software, "WIT Software Expertise," *WIT Software Expertise*, 2023.

[4] "SAFE agile." https://scaledagile.com/what-is-safe/ (accessed Jul. 17, 2023).

[5] "Scrum." https://www.scrum.org/resources/what-scrum-module (accessed Jul. 17, 2023).

[6] "Kanban." https://kanbanize.com/kanban-resources/getting-started/what-is-kanban (accessed Jul. 17, 2023).

[7] A. F. Santos, J. Marinho, and J. Bernardino, "Blockchain-Based Loyalty Management System," *Future Internet*, vol. 15, no. 5, 2023, doi: 10.3390/fi15050161.

[8] M. Krichen, M. Ammi, A. Mihoub, and M. Almutiq, "Blockchain for Modern Applications: A Survey," *Sensors*, vol. 22, no. 14, 2022, doi: 10.3390/s22145274.

[9] "AWS - Amazon," *What is Decentralization in Blockchain?* https://aws.amazon.com/blockchain/decentralization-in-blockchain/ (accessed Oct. 30, 2022).

[10] "IBM," *What are smart contracts on blockchain?* https://www.ibm.com/topics/smart-contracts (accessed Oct. 30, 2022).

[11] "GeeksforGeeks," *Features of Blockchain*. https://www.geeksforgeeks.org/features-of-blockchain/ (accessed Dec. 07, 2022).

[12] D. Geneiatakis, Y. Soupionis, G. Steri, I. Kounelis, R. Neisse, and I. Nai Fovino, "Blockchain Performance Analysis for Supporting Cross-Border E-Government Services," *IEEE Trans Eng Manag*, vol. PP, pp. 1–13, Jul. 2020, doi: 10.1109/TEM.2020.2979325.

[13] P. Aithal, P. Saavedra, S. Aithal, and S. Ghoash, "Blockchain Technology and its Types-A Short Review," *International Journal of Applied Science and Engineering*, vol. 9, no. 2, pp. 189–200, Dec. 2021.

[14] J. Shyamli, "The Complete Guide for Types of Blockchain!," *The Complete Guide for Types of Blockchain!*, May 31, 2023.

https://www.simplilearn.com/tutorials/blockchain-tutorial/types-of-blockchain#types_of_blockchain (accessed Jul. 12, 2023).

[15] "Bitcoin." https://bitcoin.org/en/ (accessed Jul. 17, 2023).

[16] "Ethereum Network," *Ethereum Network*, 2015. https://ethereum.org/ (accessed Jan. 15, 2023).

[17] "Hyperledger Blockchains." https://www.hyperledger.org/ (accessed Jul. 17, 2023).

[18] "Corda Blockchain." https://corda.net/ (accessed Jul. 17, 2023).

[19] Binance Academy, "What Is Layer 1 in Blockchain?," *Binance Academy - Articles*, Dec. 28, 2022. https://academy.binance.com/en/articles/what-is-layer-1-in-blockchain (accessed Jul. 12, 2023).

[20] "What is layer 2?," *Ethereum*. https://ethereum.org/en/layer-2/#what-is-layer-2 (accessed Jul. 12, 2023).

[21] "SIDECHAINS," *Ethreum*, May 08, 2023. https://ethereum.org/en/developers/docs/scaling/sidechains/ (accessed Jul. 12, 2023).

[22] R. Shrestha, R. Bajracharya, A. P. Shrestha, and S. Y. Nam, "A new type of blockchain for secure message exchange in VANET," *Digital Communications and Networks*, vol. 6, no. 2, pp. 177–186, 2020, doi: https://doi.org/10.1016/j.dcan.2019.04.003.

[23] "CONSENSUS MECHANISMS," *Ethereum*, Jan. 13, 2023.

[24] S. Suratkar, M. Shirole, and S. Bhirud, "Cryptocurrency Wallet: A Review," in *2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 2020, pp. 1–7. doi: 10.1109/ICCCSP49186.2020.9315193.

[25] D. Yalalov, "Fungible vs Non-Fungible Tokens (NFT): What is the Difference?," *Metaverse Post*, Sep. 14, 2022. https://mpost.io/fungible-vs-non-fungible-tokens-nft-what-is-the-difference/ (accessed Jul. 13, 2023).

[26] "GAS AND FEES," *Ethereum*, Jun. 13, 2023. https://ethereum.org/en/developers/docs/gas/ (accessed Jul. 13, 2023).

[27] T. Górski, "Reconfigurable Smart Contracts for Renewable Energy Exchange with Re-Use of Verification Rules," *Applied Sciences*, vol. 12, no. 11, 2022, doi: 10.3390/app12115339.

[28] K. Kim, J. Ryu, H. Lee, Y. Lee, and D. Won, "Distributed and Federated Authentication Schemes Based on Updatable Smart Contracts," *Electronics (Basel)*, vol. 12, no. 5, 2023, doi: 10.3390/electronics12051217.

[29] OpenZeppelin, "Proxy Upgrade Pattern," *OpenZeppelin Docs*, 2023. https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies#upgrading-via-the-proxy-pattern (accessed Apr. 18, 2023).

[30] R. Sandford *et al.*, "ERC-2771: Secure Protocol for Native Meta Transactions," *Ethereum Improvement Proposals*, Jul. 01, 2020. https://eips.ethereum.org/EIPS/eip-2771 (accessed Jul. 13, 2023).

[31] F. Blum, B. Severin, M. Hettmer, P. Hückinghaus, and V. Gruhn, "Building Hybrid DApps using Blockchain Tactics -The Meta-Transaction Example," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–5. doi: 10.1109/ICBC48266.2020.9169423.

[32] Y. Lu, "The blockchain: State-of-the-art and research challenges," *J Ind Inf Integr*, vol. 15, pp. 80–90, 2019, doi: https://doi.org/10.1016/j.jii.2019.04.002.

[33] Bobby Chernev, "53+ Customer Loyalty Statistics to Keep in Mind in 2023," *Review 42*, May 20, 2023. https://review42.com/resources/customer-loyalty-statistics/ (accessed Jul. 06, 2023).

[34] Fortune Business Insights, "Loyalty Management Market Size, Share and COVID-19 Impact Analysis, By Deployment (On-Premise and Cloud), By Enterprise Type (Large Enterprises and Small & Medium Enterprises),By End Use (BFSI, IT and Telecommunications, Transportation, Retail, Hospitality, Manufacturing, Media & Entertainment, and Others), and Regional Forecast, 2023-2030," May 2023. Accessed: Jul. 06, 2023. [Online]. Available: https://www.fortunebusinessinsights.com/industry-reports/loyalty-management-market-101166

[35] Bond Brand Loyalty, "Pathways to Growth Guide," 2022. Accessed: Jul. 06, 2023. [Online]. Available: https://352767.fs1.hubspotusercontent-na1.net/hubfs/352767/Bond%20Pathways%20to%20Growth%20Guide%20powerd%20by%20The%20Loyalty%20Report%202022.pdf

[36] D. Agrawal, N. Jureczek, G. Gopalakrishnan, M. N. Guzman, M. Mcdonald, and H. Kim, "Loyalty Points on the Blockchain," *Business and Management Studies*, vol. 4, pp. 80–92, 2018, doi: 10.2139/ssrn.3246395.

[37] Huhn Jessica, "8 Types of Loyalty Programs: Which Is Right for You?," *ReferralRock: CUSTOMER LOYALTY*, Dec. 20, 2022. https://referralrock.com/blog/types-of-loyalty-programs (accessed Jul. 07, 2023).

[38] Wolfer Paul, "7 Types of Loyalty Programs: Which is Right for Your Brand? [With Examples]," *Ebbo:Ultimate Guide to Loyalty Programs*. https://www.ebbo.com/insights/blog/7-types-of-loyalty-programs-which-is-right-for-your-brand/ (accessed Jul. 07, 2023).

[39] Antavo, "Gamified Loyalty Programs," *Gamified Loyalty Programs*. https://antavo.com/loyalty-program-types/gamified-loyalty-programs/ (accessed Jul. 07, 2023).

[40] J. Hwang and L. Choi, "Having fun while receiving rewards?: Exploration of gamification in loyalty programs for consumer loyalty," *J Bus Res*, vol. 106, pp. 365–376, Jan. 2020, doi: 10.1016/J.JBUSRES.2019.01.031.

[41] "BEYOND REWARDS: RAISING THE BAR ON CUSTOMER LOYALTY," Aug. 2019. Accessed: Jul. 07, 2023. [Online]. Available: https://www.sessionm.com/wp-content/uploads/2020/10/2019_Harvard_Business_Review_Report_Beyond_Rewards_Raising_the_Bar_on_Customer_Loya-2.pdf

[42] M. Agrawal, D. Amin, H. Dalvi, and R. Gala, "Blockchain-based Universal Loyalty Platform," in *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*, 2019, pp. 1–6. doi: 10.1109/ICAC347590.2019.9036772.

[43] M. Hader, A. Elmhamedi, and A. Abouabdellah, "Blockchain technology in supply chain management and loyalty programs: toward blockchain implementation in retail market," in *2020 IEEE 13th International Colloquium of Logistics and Supply Chain Management (LOGISTIQUA)*, 2020, pp. 1–6. doi: 10.1109/LOGISTIQUA49782.2020.9353879.

[44] I. Antoniadis, S. Kontsas, and K. Spinthiropoulos, "Blockchain and Brand Loyalty Programs: A Short Review of Applications and Challenges," *Procedia of Economics and Business Administration*, vol. 5, no. 1, pp. 8–16, Dec. 2019, doi: 10.26458/v5.i1.1.

[45] Mosley Megan, "Loyalty and Referral Programs: What's Best for Your Business?," *ReferralRock: REFERRAL MARKETING, CUSTOMER LOYALTY*, May 11, 2023. https://referralrock.com/blog/referral-vs-loyalty-programs/ (accessed Jul. 08, 2023).

[46] "Dropbox referral program." https://www.dropbox.com/refer (accessed Jul. 18, 2023).

[47] Hang, "Loyalty vs. Membership — What's the Difference?," *Loyalty vs. Membership — What's the Difference?*, Dec. 06, 2022. https://medium.com/hang-xyz/loyalty-vs-membership-whats-the-difference-7819ddf7a2d9 (accessed Jul. 08, 2023).

[48] "Amazon Prime." https://www.amazon.com/amazonprime (accessed Jul. 08, 2023).

[49] "Run a Loyalty Program," *The how-to issue*. Bloomberg Businessweek, May 24, 2021. Accessed: Jul. 08, 2023. [Online]. Available: https://www.bloomberg.com/features/2021-how-to#start-loyalty-program

[50] O. Sönmeztürk, T. Ayav, and Y. M. Erten, "Loyalty Program using Blockchain," in *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020, pp. 509–516. doi: 10.1109/Blockchain50366.2020.00074.

[51] Ethereum Org, "Rinkeby testesnet." https://ethereum.org/en/developers/docs/networks/#rinkeby (accessed Jul. 18, 2023).

[52] "EIP-20," *EIP-20 Token Standard*, 2015. https://eips.ethereum.org/EIPS/eip-20 (accessed Jan. 15, 2023).

[53] "React." https://react.dev/ (accessed Jul. 18, 2023).

[54] "Web3 js." https://web3js.readthedocs.io/en/v1.10.0/ (accessed Jul. 18, 2023).

[55] "Reactive Native." https://reactnative.dev/ (accessed Jul. 18, 2023).

[56] "Stellar." https://stellar.org/ (accessed Jul. 18, 2023).

[57] L. Wang, X. Luo, Y. Hua, and J. Wang, "Exploring how blockchain impacts loyalty program participation behaviors: An exploratory case study," *52nd Hawaii International Conference on System Sciences*, pp. 1–10, 2019, doi: 10.24251/HICSS.2019.553.

[58] L. Wang, X. (Robert) Luo, and F. Lee, "Unveiling the interplay between blockchain and loyalty program participation: A qualitative approach based on Bubichain," *Int J Inf Manage*, vol. 49, pp. 397–410, 2019, doi: https://doi.org/10.1016/j.ijinfomgt.2019.08.001.

[59] R. M. Ryan, E. L. Deci, and others, "Overview of self-determination theory: An organismic dialectical perspective," *Handbook of self-determination research*, vol. 2, pp. 3–33, 2002.

[60] "BubiChain." https://github.com/bubichain/bubichain (accessed Jul. 18, 2023).

[61] L. J. Dominguez Perez, L. Ibarra, G. F. Alejandro, A. Rumayor, and C. Lara-Alvarez, "A loyalty program based on Waves blockchain and mobile phone interactions," *Knowledge Engineering Review*, vol. 35, no. E12, 2020, doi: 10.1017/S0269888920000181.

[62] "Waves blockchain." https://waves.tech/ (accessed Jul. 18, 2023).

[63] S. Udegbe, "Impact of blockchain technology in enhancing customer loyalty programs in airline business," *Int J Innov Res Adv Stud*, vol. 4, no. 6, pp. 257–263, 2017.

[64] Ş. Bülbül and G. İnce, "Blockchain-based Framework for Customer Loyalty Program," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, 2018, pp. 342–346. doi: 10.1109/UBMK.2018.8566642.

[65] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, H.-A. Pham, N. H. Tuong, and E. Dutkiewicz, "Blockchain-based Secure Platform for Coalition Loyalty Program Management," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, 2021, pp. 1–6. doi: 10.1109/WCNC49053.2021.9417501.

[66] S.-F. Tu, C.-S. Hsu, and Y.-T. Wu, "A Loyalty System Incorporated with Blockchain and Call Auction," *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 17, no. 3, pp. 1107–1123, 2022, doi: 10.3390/jtaer17030056.

[67] M. Utz, S. Johanning, T. Roth, T. Bruckner, and J. Strüker, "From ambivalence to trust: Using blockchain in customer loyalty programs," *Int J Inf Manage*, vol. 68, p. 102496, 2023, doi: https://doi.org/10.1016/j.ijinfomgt.2022.102496.

[68] "KrisFlyer." https://www.singaporeair.com/en_UK/sg/ppsclub-krisflyer/krisflyer/ (accessed Jul. 17, 2023).

[69] Z. Wolfie, "Singapore Airlines to Launch Blockchain-Based Loyalty Wallet," Feb. 05, 2018. https://www.coindesk.com/markets/2018/02/05/singapore-airlines-to-launch-blockchain-based-loyalty-wallet/ (accessed Jul. 17, 2023).

[70] "Starbucks." https://www.starbucks.pt/ (accessed Jul. 17, 2023).

[71] "Starbucks Rewards." https://www.starbucks.pt/rewards (accessed Jul. 17, 2023).

[72] "Starbucks Odyssey." https://odyssey.starbucks.com/ (accessed Jul. 17, 2023).

[73] "The Starbucks Odyssey Begins," Dec. 08, 2022. https://stories.starbucks.com/stories/2022/the-starbucks-odyssey-begins/ (accessed Jul. 17, 2023).

[74] M. Legrand, "Luxury Brands Are Still Embracing Blockchain For Loyalty and Product Authentication Programs," *En Mode Magazine*, Jan. 04, 2023. https://www.enmodemagazine.com/luxury-brands-are-still-embracing-blockchain-for-loyalty-and-product-authentication-programs/ (accessed Jul. 17, 2023).

[75] "Loyyal." https://loyyal.com/ (accessed Jul. 14, 2023).

[76] Bitcoin Exchange Guide News Team, "Loyyal – Smart Contract Blockchain Internet Loyalty Rewards?," *Bitcoin Exchange Guide*, Aug. 30, 2017.

[77] "Qiibee." https://www.qiibee.com/ (accessed Jul. 14, 2023).

[78] "White Paper 2.7," Jan. 2019. Accessed: Jul. 14, 2023. [Online]. Available: https://static.qiibee.com/qiibee-White-Paper.pdf

[79] "Loopy Loyalty." https://loopyloyalty.com/ (accessed Jul. 14, 2023).

[80] Loopy Loyalty, "Introducing the new Loopy Loyalty," *The Customer Loyalty Marketing Blog - Loopy Loyalty*, Aug. 16, 2017. https://blog.loopyloyalty.com/introducing-the-new-loopy-loyalty-65c4f820a345#97ca (accessed Jul. 14, 2023).

[81] "Appsolutely." https://www.appsolutely.ph/ (accessed Jul. 14, 2023).

[82] CryptoNinjas, "Appsolutely to use NEM blockchain to scale transaction rate for LoyalPlatform," Nov. 14, 2017. https://www.cryptoninjas.net/2017/11/14/appsolutely-use-nem-blockchain-scale-transaction-rate-loyalplatform/ (accessed Jul. 14, 2023).

[83] "NEM documentation." https://nemproject.github.io/nem-docs/pages/ (accessed Jul. 14, 2023).

[84] "Joyn." https://joyn.eu/en/ (accessed Jul. 14, 2023).

[85] "Android." https://www.android.com/intl/pt_pt/what-is-android/ (accessed Jul. 17, 2023).

[86] "Kotlin." https://kotlinlang.org/ (accessed Jul. 17, 2023).

[87] "Solidity." https://soliditylang.org/ (accessed Jul. 17, 2023).

[88] Mozila, "JavaScript." https://developer.mozilla.org/en-US/docs/Web/JavaScript (accessed Jul. 17, 2023).

[89] "Polygon PoS." https://polygon.technology/polygon-pos (accessed Jul. 19, 2023).

[90] "Android's Kotlin-first approach." https://developer.android.com/kotlin/first (accessed Jul. 17, 2023).

[91] Polygon, "Architecture Overview," *Polygon PoS*. https://wiki.polygon.technology/docs/pos/polygon-architecture (accessed Jul. 19, 2023).

[92] Polygon, "Heimdall," *Polygon PoS documentation*. https://wiki.polygon.technology/docs/category/heimdall (accessed Jul. 19, 2023).

[93] Polygon, "Bor," *Polygon PoS documentation*. https://wiki.polygon.technology/docs/pos/design/bor/ (accessed Jul. 19, 2023).

[94] Finematics, "Polygon PoS Chain - A Commit Chain? DeFi Explained." https://www.youtube.com/watch?v=f7F67ZP9fsE (accessed Jul. 19, 2023).

[95] "Android Studio." https://developer.android.com/studio/intro (accessed Jul. 17, 2023).

[96] "Remix IDE." https://remix-project.org/ (accessed Jul. 17, 2023).

[97] "OpenZeppelin." https://www.openzeppelin.com/ (accessed Jul. 17, 2023).

[98] "IPFS." https://ipfs.tech/ (accessed Jul. 17, 2023).

[99] "Jira Software." https://www.atlassian.com/software/jira (accessed Jul. 17, 2023).

[100] "Web3Auth." https://web3auth.io/ (accessed Jul. 17, 2023).

[101] "Hardhat." https://hardhat.org/ (accessed Jul. 25, 2023).

[102] Ethereum, "Introduction to DApps," Jul. 11, 2023. https://ethereum.org/en/developers/docs/dapps/ (accessed Jul. 21, 2023).

[103] "PolygonScan," *PolygonScan*, 2023. https://polygonscan.com/ (accessed Apr. 08, 2023).

[104] "EthereumIo," *EthereumIo*, 2023. https://etherscan.io/ (accessed Apr. 08, 2023).

[105] "Avascan," *AvalancheScan*, 2023. https://avascan.info/ (accessed Apr. 08, 2023).

[106] Solana, "Solana Website," 2023. https://solana.com/ (accessed Apr. 08, 2023).

[107] Android Developers, "Recommendations for Android architecture." https://developer.android.com/topic/architecture/recommendations (accessed Jul. 30, 2023).

[108] "Github ipfs-api-kotlin." https://github.com/komputing/ipfs-api-kotlin (accessed Jul. 25, 2023).

[109] "Web3J." https://docs.web3j.io/4.10.0/ (accessed Jul. 25, 2023).

[110] Android developers, "Room Database." https://developer.android.com/training/data-storage/room (accessed Jul. 25, 2023).

ANNEX A: INTERNSHIP PROPOSAL

Instituto Superior de Engenharia
Politécnico de Coimbra

Mestrado em Engenharia Informática
Proposta de Estágio / Projeto / Dissertação

Apagar o que não interessar

Ano Letivo de 2022/2023

Título	Mobile App prototype for a Blockchain loyalty platform
Aluno	
Orientador	
Empresa	WIT Software
Supervisor	Raul Fonseca Director of Software Engineering raul.fonseca@wit-software.com
Local de trabalho	Neste momento, a generalidades dos colaboradores da empresa encontram-se em teletrabalho podendo o estágio decorrer nestes mesmos moldes. Deixamos, contudo, a indicação de que se a situação se alterar, o local de trabalho poderá ser o escritório da WIT Software, em Taveiro, no Porto, em Leiria ou em Aveiro, se assim for do interesse do aluno

SUMÁRIO

Este estágio consiste na prototipagem de uma aplicação de programas de fidelização, recorrendo a tecnologia Blockchain. Essa aplicação deverá permitir que pequenos comerciantes locais criem facilmente a criação de programas de fidelização junto dos seus clientes, gerindo todos os aspetos do programa desde a atribuição de pontos até o cliente redimir a recompensa.

1. ÂMBITO

Os programas de fidelização já deram provas de serem uma das técnicas mais eficazes parar aumentar negócio através de retenção de clientes. As marcas sabem que é mais caro conquistar um novo cliente do que manter satisfeito um cliente existente. Cerca de 84% dos consumidores manifestam preferência por manter-se cliente de uma marca que disponibilize um programa de fidelização. E cerca de 66% dos consumidores indicaram que a possibilidade de ganhar recompensas com um programa de fidelização, influenciou o seu comportamento como consumidor.

No entanto, a proliferação destes sistemas de fidelização, origina um labirinto de sistemas de pontos, formas de redimir recompensas e suporte para inúmeros parceiros.

A tecnologia Blockchain poderá ser uma solução para esta problemática. Consumidores podem usar esta tecnologia por exemplo para redimir recompensas de forma imediata, gerir e trocar pontos de diferentes programas de fidelização numa única plataforma, tendo apenas uma única carteira de pontos.

E é neste contexto que lançamos este desafio. Este estágio consiste no estudo da aplicabilidade de tecnologia Blockchain para a implementação de programas de fidelização, seguindo-se a implementação de um protótipo de uma aplicação para dispositivos iOS com objetivo de validar as mecânicas identificadas, tanto na vertente do comerciante como do utilizador/cliente final.

1/4

Instituto Superior
de Engenharia
Politécnico de Coimbra

A vertente do comerciante deverá permitir aos donos de comércio local o lançamento de programas de fidelização junto dos seus clientes, assim como o acompanhamento das métricas de sucesso de cada programa (clientes aderentes, pontos atribuídos, evolução de adesão, custos com recompensas, etc).

A vertente do utilizador final deverá incluir um localizador de negócios locais com programa de fidelização ativos, consulta de detalhes de cada programa, adesão a programas de fidelização, consulta de estado e opções para redimir recompensas.

O desenho da aplicação deverá assentar numa arquitetura totalmente distribuída e não depender de um servidor aplicacional único. Dessa forma, a aplicação deverá integrar diretamente com uma Blockchain a definir. A lógica das mecânicas de fidelização poderá incluir o desenvolvimento de smart contracts.

2. OBJETIVOS

O objetivo do estágio consiste no desenvolvimento de um protótipo de aplicação móvel para dispositivos iOS de gestão de programas de fidelização por cima de tecnologia blockchain, tanto do ponto de vista de comerciantes que lançam programas de fidelização como do ponto de vista dos clientes que aderem a cada programa, vão acumulando pontos e por fim pretendem redimir uma recompensa.

O presente estágio pretende atingir os seguintes objetivos genéricos:

1. Aquisição de conhecimento detalhado sobre funcionamento de blockchains e desenvolvimento de dApps com smart contracts;
2. Aquisição de conhecimento detalhado sobre mecânicas de programas de fidelização.
3. Definição de requisitos para o protótipo a implementar
4. Implementação de protótipo de aplicação móvel iOS com vista de comerciante para configuração e monitorização de campanhas de fidelização para os clientes do negócio local;
5. Implementação de protótipo de aplicação móvel iOS com vista do cliente final, para descoberta de negócios com programas ativos, adesão, consulta de progresso e opções para redimir recompensas;
6. Implementação das mecânicas de fidelização em smart contracts;
7. Demonstração de casos de uso completos de programas de fidelização.

3. PLANO DE TRABALHOS

Nota: O trabalho total consiste em cerca de 1170 horas: 390 horas no 1º semestre e 780 horas no 2º semestre.

O estágio consistirá nas seguintes actividades e respectivas tarefas:

- T1 – Estado da arte – programas de fidelização com e sem integração com Blockchain, em que medida a tecnologia Blockchain pode enriquecer as mecânicas de fidelização;
- T2 – Recolha de requisitos – Estudar requisitos funcionais e especificação de requisitos técnicos.
- T3 – Desenho da solução – Desenho dos módulos funcionais a prototipar e a arquitetura dos mesmos.
- T4 – Planeamento – Elaboração do plano de desenvolvimento.

Instituto Superior
de Engenharia
Politécnico de Coimbra

- T5 – Desenvolvimento – Prototipagem das funcionalidades da vista do comerciante, incluindo smart contracts.
- T6 – Desenvolvimento – Prototipagem das funcionalidades da vista do cliente, incluindo smart contracts.
- T7 – Testes – Testes funcionais e benchmarking do protótipo.
- T8 – Documentação – Documentação de projeto e de final de estágio.

A calendarização prevista para os trabalhos e entregas será a seguinte:

	Set		Out		Nov		Dez		Jan		Fev		Mar		Abr		Mai		Jun		Jul	
Tarefas																						
T1																						
T2																						
T3																						
T4																						
T5																						
T6																						
T7																						
Entregas																						
E1							X															
E2									X													
E3													X									
E4																					X	

4. METODOLOGIA DE TRABALHO

Serão utilizadas metodologias ágeis. O acompanhamento do estágio será feito não só pelo orientador, mas também por um tutor técnico que dará ao aluno todo o apoio necessário. O Orientador define os requisitos do estágio, define as prioridades do Backlog e acompanha os resultados parciais do projecto. O Tutor dá todo o suporte técnico necessário, garante o cumprimento das tarefas e promove as reuniõe de acompanhamento do cumprimento dos objectivos.

5. CARATERIZAÇÃO DO ESTÁGIO

- O horário será a tempo completo.
- Durante o estágio o aluno terá ao seu dispor todos os equipamentos necessários para desempenhar as suas tarefas.
- O estágio é remunerado. Se o desempenho do aluno ao longo do mês for positivo, terá direito a receber uma bolsa mensal.
- Terá ainda acesso às formações da WIT Academy.
- No final do estágio a WIT tem todo o interesse em que os alunos continuem na empresa. Será feita uma avaliação do desempenho durante o estágio, dos conhecimentos adquiridos e dos resultados obtidos. Se esta avaliação for positiva será feita uma proposta ao aluno para que possa vir a fazer parte da equipa de desenvolvimento.

3/4

Instituto Superior
de Engenharia
Politécnico de Coimbra

Sobre a Empresa:
A WIT tem 20 anos de experiência na área das telecomunicações e é uma empresa francamente exportadora. Prova disso é o facto do software que desenvolve já ter sido exportado para 46 países. Todos estes projectos foram desenvolvidos com uma metodologia muito forte de software, sempre com requisitos de alta-disponibilidade, segurança, performance, escalabilidade e especial cuidado com a user-experience. A empresa conta com os seguintes clientes: Grupo Vodafone, Deutsche Telekom, NTT DoCoMo, KDDI, Softbank, AT&T, Verizon, Safaricom, Vodacom, Unitel, entre outros. Na WIT somos Groundbreakers e procuramos, todos os dias, desafiar o status quo fazendo mais e diferente, sempre com o objectivo de sermos cada vez melhores. Descobre aqui se também tu podes ser um groundbreaker:
https://www.youtube.com/watch?v=Mt9sCqvJx9U

4/4

ANNEX B: MOBILE APP UI

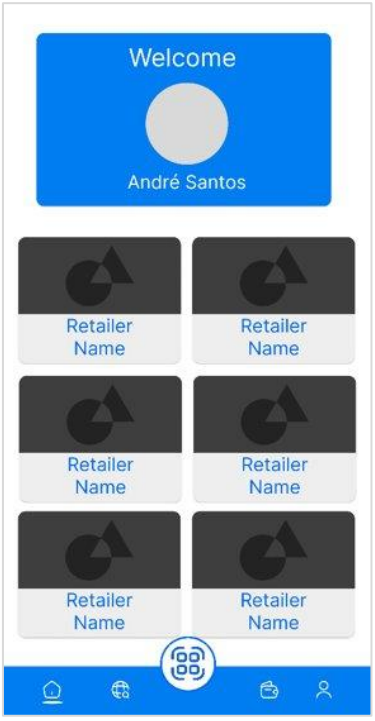


Figure B.1 - 1st version Customer Home Page

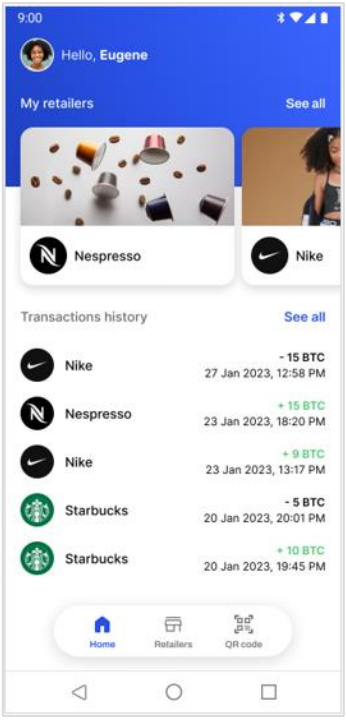


Figure B.2 - 2nd version Customer Home Page

Figure B.3 - 1st version Transaction History

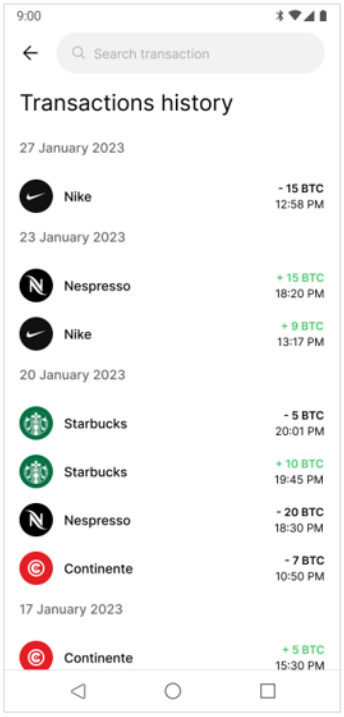


Figure B.4 - 2nd version Transaction History

Figure B.5 - 1st version Retailer Info

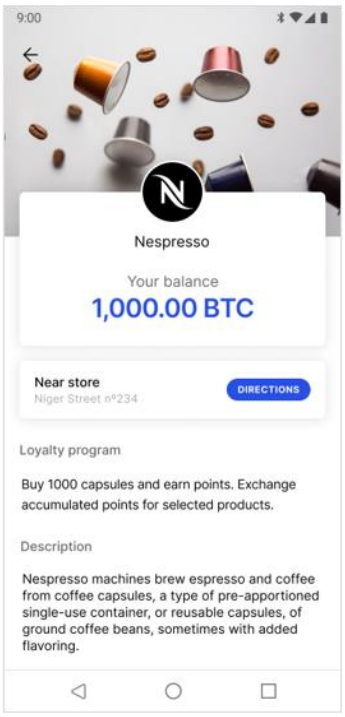


Figure B.6 - 2nd version Retailer Info

Figure B.7 - 1st version QR Code Display

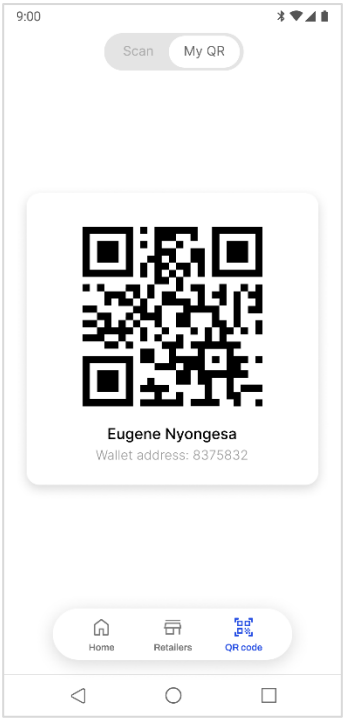


Figure B.8 - 2nd version QR Code Display

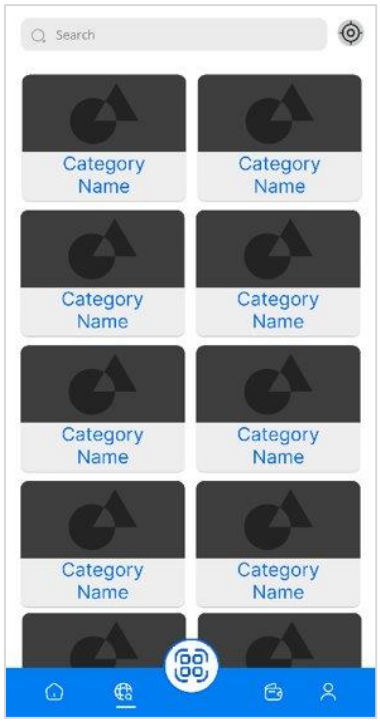


Figure B.9 - 1st version Retailers Discovery

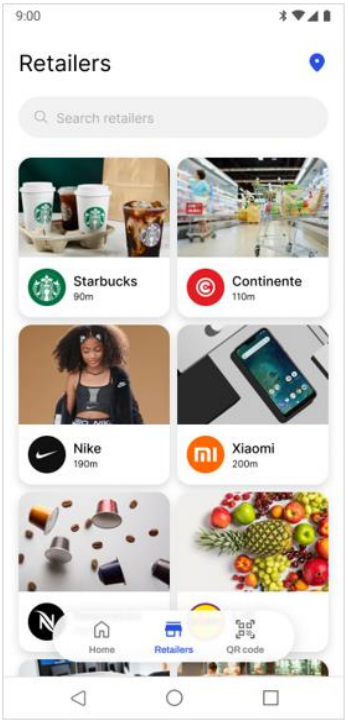


Figure B.10 - 2nd version Retailers Discovery

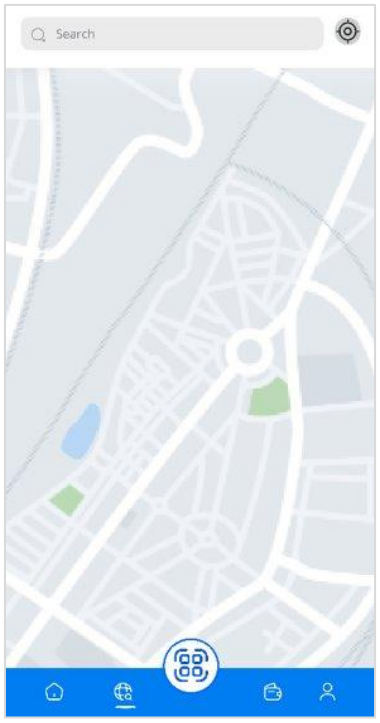


Figure B.11 - 1st version Map Discovery

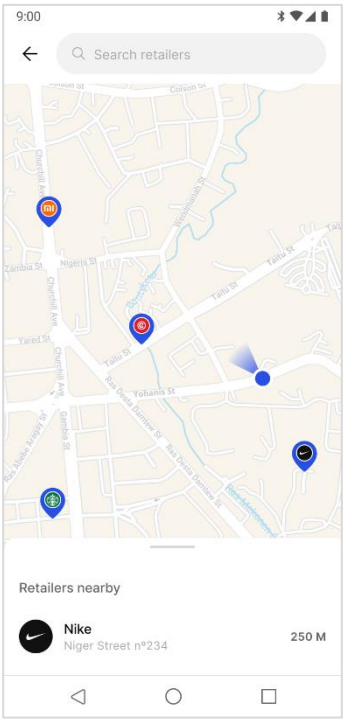


Figure B.12 - 2nd version Map Discovery

Figure B.13 - 1st version Subscribe Retailer

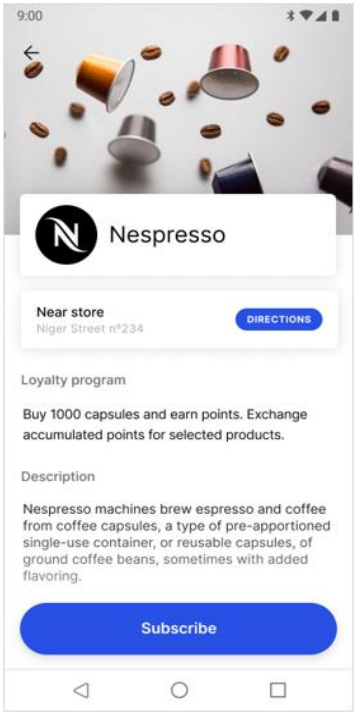


Figure B.14 - 2nd version Subscribe Retailer

Figure B.15 - 1st version Business Information

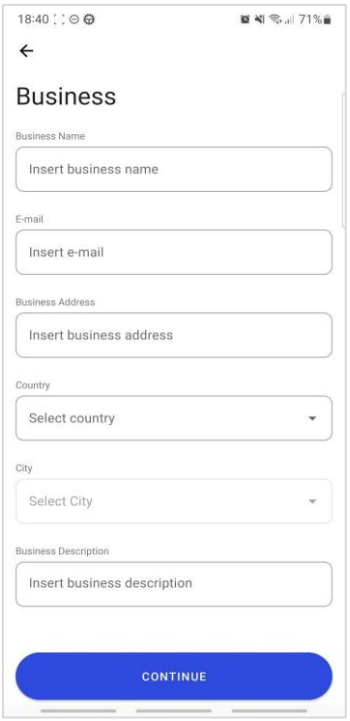


Figure B.16 - 2nd version Business Information

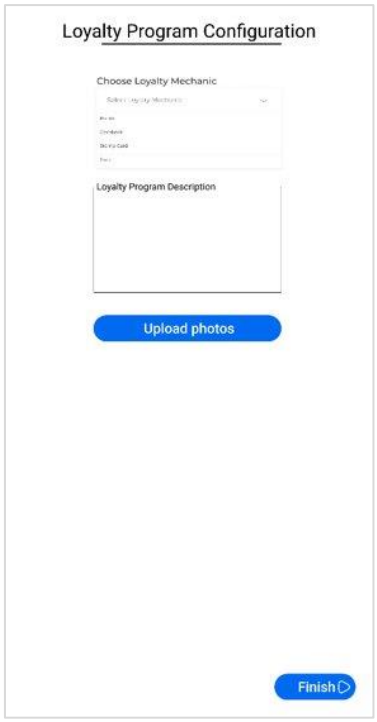


Figure B.17 - 1st version Loyalty Config

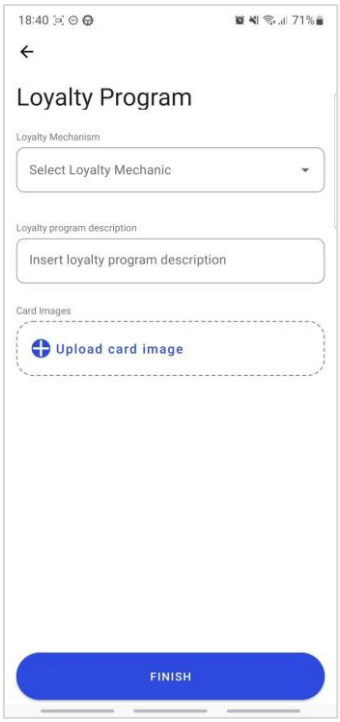


Figure B.18 - 2nd version Loyalty Config

Figure B.19 - 1st version Retailers Home Page

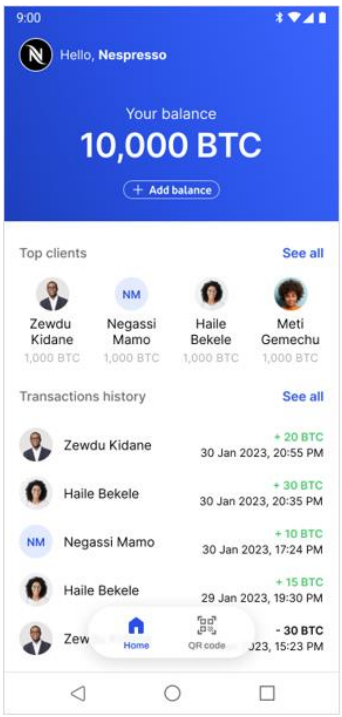
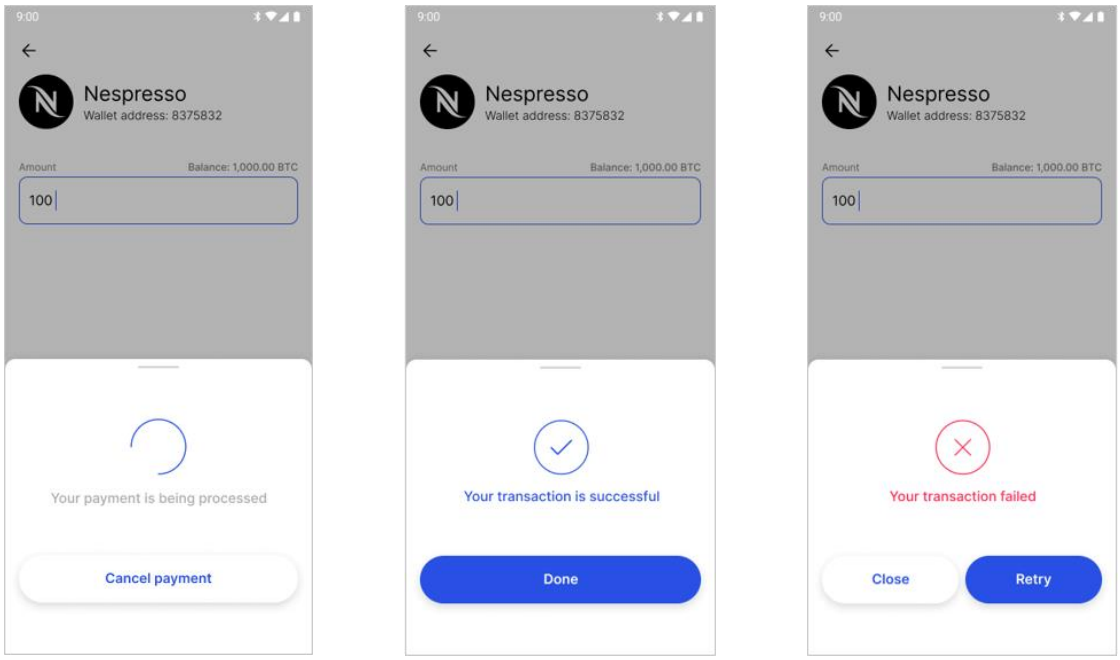


Figure B.20 - 2nd version Retailers Home Page

Figure B.21 - QR Code Reading

Figure B.22 - Transaction Process Screen



a) b) c)

Figure B.23 - Transaction Process Status Screens: a) Loading screen; b) Transaction Successful; c) Transaction Failed

ANNEX C: LOYALTY MANAGER FUNCTIONS

Table C-1 - Loyalty Manager functions

Method Name	Description
addLoyaltyMechanismContract	This method enables the dynamic addition of a loyalty mechanism contract during runtime. The function takes two parameters: the *LoyaltyMechanismType* and the contract reference in the form of the Loyalty Mechanism interface. Since *LoyaltyMechanismType* values are converted to *uint*, the function can also receive a *uint* to represent a value that is not explicitly defined in the *enum*. For example, if the *enum* includes "points" (0) and "stampcards" (1), we can call the function and pass the *uint* value 2 along with the contract reference to add the new loyalty mechanism. We don't need the value defined in the *enum* previously
addRetailerRegistry	This method adds a new retailer registration to the system. The function takes two arguments: an IPFS hash (a string) and the *LoyaltyMechanismType* chosen by the retailer for their loyalty program. To begin, the function obtains the retailer's address and accesses a mapping to retrieve the retailer ID. This approach ensures that the retailer is not already registered. If the retrieved value is different from the default (0), it indicates that the user is already in the system, and the transaction is reverted. Once it's verified that the retailer is new, a unique ID is assigned to them based on the current retailer counter plus one. Subsequently, an object of the *RetailerInfo* structure is created to represent the retailer's data. To conclude the process, the Loyalty Manager calls the setTokenForRetailer function of the specific mechanism contract that the retailer used for their loyalty program. This function is defined in the Loyalty Mechanism interface. Afterward, the Loyalty Manager emits the event "newRetailer" to signal the successful addition of the retailer.
getRetailerInfo	This method retrieves an object of the *RetailerInfo* structure based on the address provided as an argument. It accesses a mapping using the given address to obtain the retailer's ID. If the ID is equal to 0, it signifies that the address does not belong to a registered retailer, and the transaction is reverted. However, if the ID is different from 0, the method accesses the second mapping using the unique ID to retrieve the *RetailerInfo* object representing the retailer and returns it.

getLoylatyMechanismType	This method retrieves the loyalty mechanism type of the retailer's loyalty program. It utilizes the getRetailerInfo function, inheriting its safety measures to identify if the provided address belongs to a retailer or not. After receiving the result from the getRetailerInfo method, this function returns the loyalty mechanism type property stored in the *RetailerInfo* object.
retrieveAllRetailersRegistry	This method retrieves all retailers registered in the system while excluding those that the caller is already subscribed to. For example, if a user (customer or retailer) calls this function, it first obtains their subscriptions. Since retailers do not subscribe to other retailers, the subscriptions will be empty, resulting in all retailers being returned for retailers' calls. However, for customers, as they have subscriptions, the retrieved retailers will not include the ones they are already subscribed to. The process involves two steps: 1) First, we get the retailers that the user is subscribed to; 2) Then, we iterate over all the retailers in the system, adding them to an array while excluding the retailers the user is subscribed to. This iteration is facilitated by the approach mentioned previously, which involves the mapping with *uints* as keys and the retailer counter. With the retailer counter, we can easily access all the mapping keys within the range of [1, retailerCounter] to retrieve the retailers' currently in the system.
addCustomerSubscription	This method enables a customer to add a new subscription. To do so, the customer needs to provide the address of the retailer as an argument. The function first ensures that the customer is not already subscribed. If the customer has already subscribed to the given retailer's address, the transaction is reverted. However, if the customer is not yet subscribed to the provided retailer's address, the method proceeds by calling the getRetailerInfo function to obtain the retailer to subscribe. By calling this method, the subscription process inherits the safety measures that ensure the provided address is indeed associated with a retailer.
retrieveCustomerSubscriptions	This method allows retrieving all customer subscriptions. The function obtains the caller's address and accesses the mapping of customer subscriptions using the caller's address as the key. The function then returns the result obtained from the mapping access, providing the list of subscriptions associated with the caller's address.
checkIfAddressIsRetailer	This straightforward function checks whether the given address corresponds to a retailer or not. The return result is a Boolean type, with "true" indicating that the address belongs to a retailer, and "false" denoting that it is not a retailer.

checkIfCustomerIsSubscribedToRetailer	Checks if the customer is subscribed to a specific retailer. The function has two arguments that are the retailer and customer addresses. If the user is subscribed it returns true, otherwise returns false.
mintRetailersToken	This function is responsible for redirecting the mint tokens requests from a retailer to the loyalty mechanism used in their loyalty program. By calling getLoyaltyMechanismType, the function inherits safety measures to verify if the caller is indeed a retailer. If the caller is not a retailer, the function will revert. However, if the caller is a retailer, the function will proceed to call the "mint" function of the associated mechanism contract, effectively redirecting the request.
userBalanceOf	This simple function facilitates retrieving the balance in a specific loyalty program. The function takes the address of the retailer's program as a parameter to check the respective balance. Similar to the mint function, the loyalty mechanism contract is obtained through the provided retailer address. The Loyalty Manager then calls the "userBalanceOf" function of the respective mechanism to fetch the balance. The function returns this result to the Loyalty Manager, which, in turn, returns the obtained result.
transferTokens	The transfer tokens function enables the Loyalty Manager to perform token transfers by calling the appropriate mechanism contracts. This method takes the receiver address and the transfer amount as arguments. Moreover, the function obtains the sender address by accessing the caller address. To begin, the function ensures that at least one of the parties involved is a retailer since prototype customer-to-customer and retailer-to-retailer transactions are not allowed. If both parties are customers, the transaction is canceled and reverted. Once it verifies the presence of a retailer, it then checks if the customer is subscribed to the retailer. If not, the transaction is reverted. Once both criteria are met, the Loyalty Manager redirects the function calls to the retailer's loyalty mechanism-associated contract, allowing the token transfer to be executed.
getRetailerCustomerTransactionCounter	This straightforward function retrieves the customer transaction counters in a specific retailer's loyalty program. The retailer address is obtained from the caller address, following a simple process. It first gets the loyalty mechanism type associated with the retailer, then fetches the associated contract and calls the respective function to obtain the transaction counters for the customer in that specific retailer's loyalty program.

Instituto Superior de Engenharia
Politécnico de Coimbra