



**João Pedro Silva
Almeida Quintanilha**

**Sistema de recompensas multi-agente sobre
blockchain
Multi-agent loyalty program over blockchain**



**João Pedro Silva
Almeida Quintanilha**

**Sistema de recompensas multi-agente sobre
blockchain
Multi-agent loyalty program over blockchain**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Prof. Doutor João Paulo Silva Barraca, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Prof. Doutor Hélder José Rodrigues Gomes, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro.

o júri / the jury

presidente / president

André Ventura da Cruz Marnoto Zúquete

Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

João Paulo Silva Barraca

Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Filipe Alexandre Pais de Figueiredo Correia

Professor auxiliar da Faculdade de Engenharia da Universidade do Porto

agradecimentos/ acknowledgments

Quero agradecer aos meus orientadores de tese, João Barraca e Hélder Gomes, pela sua compreensão, disponibilidade e apoio durante esta dissertação. Agradeço a todos os meus amigos, especialmente ao Bruno Henriques, Bernardo Ferreira, Pedro Marques, Bernardo Amorim e à minha namorada Jéssica Caldas, que me ajudaram a construir maravilhosas memórias durante o meu percurso académico. Um agradecimento muito especial ao Pedro Diogo, da Ubiwhere, que nunca desistiu de mim durante este caminho e me apoiou não só profissionalmente como também pessoalmente. Por último, mas não menos importante, um agradecimento muito especial à minha família, em particular à minha mãe Manuela, por me apoiar durante os altos e baixos do meu percurso académico. Às minhas irmãs, Mariana e Márcia, por partilharem a sua sabedoria comigo quando solicitada ou não. Ao meu irmão Rafael, por acordar a "criança dentro de mim" quando eu mais precisava, e ao meu padrasto, Alcides, a prova viva de que amor de pai não é apenas solicitado pelo nosso sangue. Finalmente, dedico todo o meu esforço e o meu sentimento de realização ao meu pai, pelo qual lamento não poder estar mais entre nós.

resumo

O crescimento contínuo de estratégias para aumentar os hábitos de consumo de clientes levou ao desenvolvimento de inúmeros estudos, onde investigadores estudam novas abordagens de marketing e tecnologias para alcançar soluções de sucesso e tentadoras aos olhos do público alvo. Esta dissertação reflete um estudo rigoroso sobre os diferentes tipos de sistemas de recompensa, bem como algumas das tecnologias de blockchain existentes, na esperança de convergir ambos os tópicos da maneira mais adequada e benéfica. Este estudo resulta numa solução que utiliza a blockchain de Ethereum para introduzir uma vertente de descentralização, removendo a necessidade de envolver entidades intermediárias. Simultaneamente, a solução oferece a diferentes tipos de entidades a possibilidade de produzir e consumir vouchers eletrónicos não falsificáveis, não repudiáveis e exclusivos. Para além de promover um ecossistema de aplicações descentralizadas, esta solução oferece também novas possibilidades para sistemas de recompensa. Vouchers eletrónicos rastreáveis, auditáveis, seguros e autênticos são o futuro do comércio eletrónico e do marketing digital.

abstract

The continuous growth of strategies to raise customers activity has opened room to numerous studies. It led researchers to seek for suitable marketing and technology approaches to achieve successful and tempting solutions by providing benefits for every involved participant. Therefore, this dissertation focuses on a study of different loyalty program types, as well as blockchain technologies, hoping to merge the two topics the most suitable and beneficial way. This study results on a solution that makes use of Ethereum blockchain to leverage cost reduction, friction and decentralization, removing any necessity for trusted mediators. Concurrently, the solution offers different stakeholders the possibility to produce and consume non-falsifiable, non-repudiable and unique vouchers, while fostering an ecosystem of decentralized applications by opening new possibilities for rewarding systems. Verifiable, traceable, auditable, secure and authentic vouchers are the future of electronic commerce and digital marketing.

Contents

Contents	i
List of Figures	iii
List of Tables	v
List of Snippets	vii
List of Acronyms	ix
1 Introduction	1
1.1 Motivation	2
1.2 Objectives	3
1.3 Contributions	3
1.4 Dissertation outline	3
2 State of the art	5
2.1 Blockchain	5
2.1.1 Blocks and Consensus Algorithms	8
2.1.2 Digital Tokens, Identification and Transactions	11
2.1.3 Ethereum	13
2.1.4 Hyperledger	17
2.2 Loyalty programs	23
2.2.1 How loyalty programs work	23
2.2.2 How loyalty programs add value	24
2.2.3 Types of loyalty programs	24

2.3	Related work	27
2.3.1	Non-electronic loyalty programs	27
2.3.2	Electronic loyalty programs	28
2.3.3	Blockchain-based loyalty programs	31
3	Multi-agent loyalty program	35
3.1	Problems statement	36
3.2	Solution overview and objectives	36
3.3	Stakeholders	39
3.4	Vouchers	40
3.5	Use cases	42
3.6	Case scenario	46
3.7	Requirements	50
4	Implementation	53
4.1	Technology choices	53
4.1.1	Blockchain choice	56
4.1.2	Frameworks choice	57
4.2	Assets definition	58
4.2.1	Voucher properties	59
4.2.2	Voucher deployment	60
4.2.3	Voucher functionalities	61
4.3	Stakeholders representation	65
4.3.1	Access control	66
4.4	Voucher factory	67
5	Result Analysis	69
5.1	Functionality tests	69
5.2	Demonstration	73
5.3	Requirements validation	76
5.4	Problems addressed	78
6	Conclusion	83
6.1	Future work	84
7	Appendix A: Voucher factory	87
	Bibliography	91

List of Figures

2.1	Representation of a standard blockchain.	6
2.2	Public/permissionless vs private/permissioned blockchain.	8
2.3	A standard example of a Block.	9
2.4	Transaction process on a PoW blockchain.	11
2.5	Example of a Smart contract application.	14
2.6	Metamask Google chrome extension.	17
2.7	Key concepts of HIF components.	19
2.8	Example of three peers holding chaincode and ledger.	20
2.9	Standard relationship between HIF nodes.	21
2.10	Hyperledger composer elements.	22
2.11	Part of Amazon UK billing page that offers clients a code string input. . .	29
2.12	Part of Mlovers application screen.	30
2.13	Some of the many offers provided by Benfica’s membership card.	31
3.1	Concept overview.	37
3.2	Checkout state from www.eurocarparts.com	38
3.3	Voucher definition.	40
3.4	Voucher state transition diagram.	41
3.5	System solution use case.	42
3.6	Issuing voucher and setting rules flows.	43
3.7	Making a sale and allowing clients to acquire a vouchers.	43
3.8	Flow chart from a receiving company perspective.	44
3.9	Flow chart from a client perspective.	45
4.1	Chain-based graph that helps deciding the need for a blockchain.	54
4.2	Voucher factory behavior.	67

5.1	Ganache framework.	70
5.2	Successful voucher issuance.	72
5.3	Different unit tests to analyze voucher's contract.	73
5.4	Infura connection to kovan remote node.	73
5.5	Ethereum network and solution platform connection.	74
5.6	Voucher issuance demonstration.	75
5.7	Voucher acquired.	75
5.8	Current balance of 0xec5259F5840C6be0528Dd177ae8A65Fa3a528a01.	76
5.9	Last state for voucher's properties.	76
5.10	Performance test for scalability.	77
6.1	Integration on other services.	85

List of Tables

2.1	Fraction relation between Ether currency [15].	16
3.1	Status after voucher issuance on 01/01/2000.	47
3.2	Status after some acquisitions on 01/01/2000.	48
3.3	Status after redemption on 10/01/2000 and expiration on 15/01/2000.	49
3.4	Status after redemptions on 17/01/2000.	50

List of Snippets

4.1	Voucher properties definition.	59
4.2	Voucher constructor definition.	60
4.3	Caller function that retrieves voucher value.	62
4.4	Caller function generated on compile-time that returns validity value.	62
4.5	Transaction function that changes ownership of the voucher.	63
4.6	Transaction function that verify and update voucher's validity state.	64
4.7	Transaction function that allows clients to redeem their vouchers.	64
4.8	Contract execution of <i>redeem</i> function from a 'wallet.privKey' identification.	66
4.9	Modifier that ensures that the caller of a function is the owner of the voucher.	66
5.10	Truffle project directory tree.	70
5.11	Example configuration for truffle-config.js.	71
5.12	Two examples of voucher's functionalities unit tests.	72
5.13	Web3 connection to kovan test network.	74
7.14	Part 1 of voucher factory contract.	87
7.15	Part 2 of voucher factory contract.	88
7.16	Part 3 of voucher factory contract.	89
7.17	Part 4 of voucher factory contract.	90

List of Acronyms

ABI	Application Binary Interface	IoT	Internet of Things
API	Application Programming Interface	IPFS	Interplanetary File System
CPU	Central Processing Unit	MAS	Multi-Agent System
CRUD	Create Read Update Delete	P2P	Peer-to-Peer
DAG	Directed Acyclic Graph	privKey	Private key
dApp	Decentralized Application	pubKey	Public key
DDoS	Denial of Service	PoS	Proof-of-Stake
DLT	Distributed Ledger Technology	PoW	Proof-of-Work
EIP	Ethereum Improvement Proposal	REST	Representational State Transfer
EVM	Ethereum Virtual Machine	RPC	Remote Procedure Call
GDPR	General Data Protection Regulation	SHA	Secure Hash Algorithm
HIC	Hyperledger Composer	SME	Small or Medium-sized Enterprise
HIF	Hyperledger Fabric	UI	User-interfaces
HTTP	Hyper Text Transfer Protocol	UID	Unique Identification
ICP	Independent Connection Provider	VIP	Very Important People
ICO	Initial Coin Offer		

CHAPTER 1

Introduction

The continuous evolution of incentive programs for customers activity as caught the attention of many-sized enterprises. This programs, also named Loyalty programs, are one of the two main topics of this document. Loyalty programs consist of structured marketing strategies designed by marketing specialists, and implemented by merchants, to encourage customers activity. These incentives, usually membership-based, take place in order to customers continue to shop or use the services of businesses associated with each program. Loyalty programs exist and cover most types of commerce, each one with different strategies and rewarding schemes. These programs have a long-time history and can be traced back to 1896 when S&H introduced Green Stamps[1]. By the 1960s, S&H Green Stamps was the largest purchaser of consumer goods in the world, however, with its popularity decline, newer generations of loyalty programs have gained widespread appeal, like American Airlines after introducing their flyer program in 1981[2].

Despite the lack of research concerning the fairness of loyalty programs in terms of their value proposition, discrimination and personal information exploitation, these programs keep taking place for many different use cases[1]. From 2000 to 2006, more than half of all adults in the United States have joined in at least one loyalty program, and studies estimated that more than 1.5 billions of the total worldwide population adults had joined loyalty programs[3]. Current examples of loyalty programs are, for instance, point, voucher, game and tiered programs.

One common aspect of every loyalty program is that they are usually set in a centralized way. Namely, an individual company, who proposes the loyalty program, is represented as a central authority responsible for managing this program and therefore their customers' data, allowing it to take control on the behaviour of the program. Nevertheless, loyalty programs are also frequently used within a set of partners. However, different companies who enrol the same loyalty program are commonly within the same business group, like Sonae¹. The lack of loyalty programs shared between independent and distinct companies relies mostly on the lack of trust between the participants. Thus, trust is one of the many concepts approached throughout this dissertation, which brings us to the second main topic of this document: Distributed Ledger Technology (DLT).

This recent technologies are currently very coveted by companies and have proved their value solving numerous problems presented in current distributed systems shared by multiple parties. DLTs keep growing at a fast pace, having helped the growth of many different industries like supply-chain management and finance[4]. These technologies have committed outstanding results and have had continuous room for improvement. Therefore, DLTs are a possible solution for the problems stated throughout this document, since they may offer several useful features and tools to provide trust between different parties. Namely, save time and avoid conflict by removing the third-party entity that is generally necessary on any transaction activity. It is still a recent technology acting on a daily basis, but it is undeniably more secure than traditional systems.

1.1 Motivation

A through study about DLT encouraged the search for new use cases to endorse. This whole study showed that DLTs could be applied to any use case, although, they are most suitable for systems that involve asset exchanges and ownership change. Companies, no matter the size, keep struggling and competing between each other to reach successfulness. Loyalty programs may bring individual benefits when centralized over themselves, which feeds the need to compete to achieve profitability. To sum up, the big motivation behind this dissertation is based on challenging the natural behaviour of the existing loyalty programs, making distinct companies achieve compliance and mutual profitability while, at the same time, provide customers with beneficial results.

¹Multinational business group (<https://www.sonae.pt/>)

1.2 Objectives

Has previously said, traditional loyalty programs are built on top of centralized architectures, which exclusively raise customer engagement. On the other hand, considering a distributed approach, a set of companies would potentially be able to endorse the same loyalty program, having the ability to trust each other. Thus, the objectives for this dissertation consist of developing a decentralized system based on one type of loyalty program, being able to plug-in existent e-commerce platforms. Consequently, this system should allow different independent companies to participate in the same loyalty program, not having to concern about fraudulent entities. This solution should be seen to its participants as a trust protocol and must bring transparency, auditability and financial independence to all its users.

1.3 Contributions

The proposed solution contributes to the evolution of decentralized applications within the space of loyalty programs. Essentially, it provides a new vision for a decentralized loyalty program that offers a trust protocol to its stakeholders. This work was initially proposed by Casa da Moeda, although later on diverged somewhat from its initial proposal. Both the document and the solution have had the full support of Ubiwhere², a Portuguese Small or Medium-sized Enterprise (SME) focused on innovative solutions for smart cities.

1.4 Dissertation outline

This document is organized in 6 Chapters, being the first Chapter the already presented introduction. The next Chapters of this dissertation are organized in the following sequence:

- **chapter 2:** background information of DLTs, more specifically, Blockchain technology. Introduction to different blockchain components and characteristics as well as some blockchain-based projects, such as Ethereum and Hyperledger. Also, introduces the necessary data about the existence of different types of loyalty programs, relevant existing solutions and related work;
- **chapter 3:** refers to a multi-agent loyalty program where it proposes an extensive solution description for the problems stated throughout this document;
- **chapter 4:** presents the implemented prototype, based on the designed solution previously depicted. Shortly, the stakeholders, assets architecture and its consequent functionalities are defined;

²<https://www.ubiwhere.com/>

- **chapter 5:** provides functionality tests to the prototype and shows a demonstration of it. Also, describes an overview of the obtained results, as well as a discussion for potential system limitations;
- **chapter 6:** sums up the outcomes of this dissertation, along with relevant conclusions and future work that may continue to enrich this work.

Concluding, it is now introduced the State of the art. This topic holds a high relevance for this dissertation, as it presents the overall study that later supports the solution endorsement.

State of the art

This Chapter grants an overview of the essential concepts of Blockchain technologies in section 2.1. Later on, section 2.2 introduces the major distinct types of loyalty programs as well as why they are relevant. Finally, section 2.3 puts together some existing solutions and related work, which may be seen as valuable studying resources to achieve the most suitable choice of technologies for the solution implementation.

2.1 Blockchain

DLTs are a record system that is replicated and kept in-sync across multiple participants of a network. All those network participants, mostly known as nodes, are in charge to hold an updated record of its ledger. On most traditional record systems, like databases, a master copy of the ledger is kept safe on a central server. DLTs remove the idea of a master copy of a ledger, and instead, host a copy of it between different nodes. This technology enables the maintenance of a global data structure by a set of mutually untrusted participants in a distributed environment[5]. DLTs are still considered a short lifetime technology at the current date. Consequently, these are still in a constant evolution state and are massively exposed to changes and functionality updates. It has already proved its value with the currently existing projects,

such like Cryptocurrencies¹(mostly famous for Bitcoin²) where its most notable features are immutability, resistance to censorship, decentralized maintenance, and elimination of the need for a centralized trusted third party. In other words, there is no need for an entity to be in charge of conflict resolution and upkeep of a global truth that is trusted by all untrusted stakeholders. DLTs are suitable for tracking the ownership of digital assets, and it holds promise beyond mere cryptocurrency transfer since an entry in the ledger may be generalized to hold arbitrary data. However, before being applied on a global scale, DLTs need to solve several issues that they are currently facing[6]. Two notable examples of DLTs are Blockchain and Directed Acyclic Graph (DAG). This dissertation focuses on blockchain technology as it is at the current date, one of the most trending topics and present better support, stability and documentation on development.

Blockchain was first conceptualized on the treatise "Bitcoin: A peer-to-peer electronic cash system" by Satoshi Nakamoto in 2008 [7]. It is a data structure that represents a digital book of records more known for a Digital Ledger. It keeps track of asset transactions over a particular trust protocol where each one of them is digitally signed, guaranteeing its authenticity and ensuring that no one adulterates the ecosystem. Comparing blockchain to traditional databases, these usually keep their ledger-state in secured infrastructures, however, even on the safest place, a document or file can be susceptible to tamper or to be modified with bad intentions. While this is a significant risk for transaction systems, blockchain offers a great solution by making this ledger shared across a Peer-to-Peer (P2P) network of multiple machines.

There are various blockchains but as the name itself says, all of them are composed by a chain of blocks, that are further explained in detail on Section 2.1.1. Fundamentally, blocks hold several transactions that are represented through hashe function outputs. Each blockchain has its different type of blocks to fulfil their purposes, although, with these set of norms, a blockchain continuously grows as blocks are added to it, always containing new sets of transaction records. Blocks are chronologically incremented to the blockchain as a stack, and any computer connected to this P2P network has the task of validating the transactions that comprise those blocks[8].

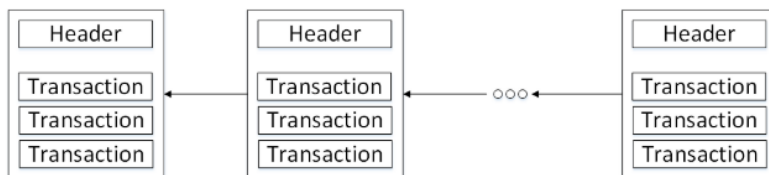


Figure 2.1: Representation of a standard blockchain.

¹A digital currency operating independently of a central bank.

²<https://bitcoin.org/en/>

Figure 2.1 illustrates a shallow blockchain where *header* represents a unique identifier for each block[6], *transaction* represents any data transaction depending on the specific blockchain, and the arrows represent the link between each block. In other words, every new block points to the previous one, and as blocks are added to the chain, they become deep-seated and immutable[9].

Block's identification depend on its content, which is further detailed in Section 2.1.1. Although, for now, it is essential to understand that a block containing a link to the previous one means, indirectly, that it is linked to the previous block's content too. Being said, changing the content of a block would break all the subsequent links, having the need to verify every subsequent block again. The difficulty of making those verifications to the whole subsequent chain of blocks is how blockchain guarantees the block's data integrity, not being susceptible to changes nor deletions[9]. This mechanism provides a secure way of recording data transactions or anything that needs to be recorded and verified as having taken place.

Types of Blockchain

Blockchains are categorized as Public/Permissionless or Private/Permissioned. The distinction between a Public and Private blockchain relies mostly on visibility access — who can audit records. While Permissionless and Permissioned blockchains distinguish themselves in terms of access control — who can do what. Both fulfill different business requirements, which allow business models to expand through a wide variety of solutions.

Public vs Private Blockchains

A public blockchain is completely open, and anyone who wishes to, can read data from it. Therefore, to actively participate in the network, public blockchains also needs to be permissionless. On the other hand, to be part or to see the records of a private blockchain, it is required an invitation or an authorization. Those must be validated through specific rules set by a network founder or anyone who has permissions to do it. Typically, a public blockchain has a mechanism that encourages more nodes to join the network, like solving cryptography problems in exchange for value, also called consensus, further detailed in Section 2.1.1. In contrast, a private blockchain keeps its nodes centralized and only able to act when accepted by other nodes. Public blockchains are open and expose a considerable amount of data publicly, which might come with little if no privacy and lacks security in terms of data transactions[10]. Differently, private blockchains do not expose data to anyone who has no read access to it.

Permissionless vs Permissioned Blockchains

Permissioned blockchain participants need to be identified. They require an invitation and must be validated by either the network founder or by someone authorised to do it. This mechanism dictates who

is allowed to participate, such as practising consensus protocols and maintain the shared ledger[11]. To those participants, permissions are assigned for different interactions with the blockchain[12].

Organizations who build their business models over a permissioned blockchain can be more engaged over its content since they can specify the speed of the transactions, create different assets and minimize security methods such as cryptographic signatures. Commonly, public blockchains are also permissionless, while private blockchains are also permissioned. Thus, for the sake of simplicity, Figure 2.2 demonstrates some of the differences between public/permissionless and private/permissioned blockchains[13].

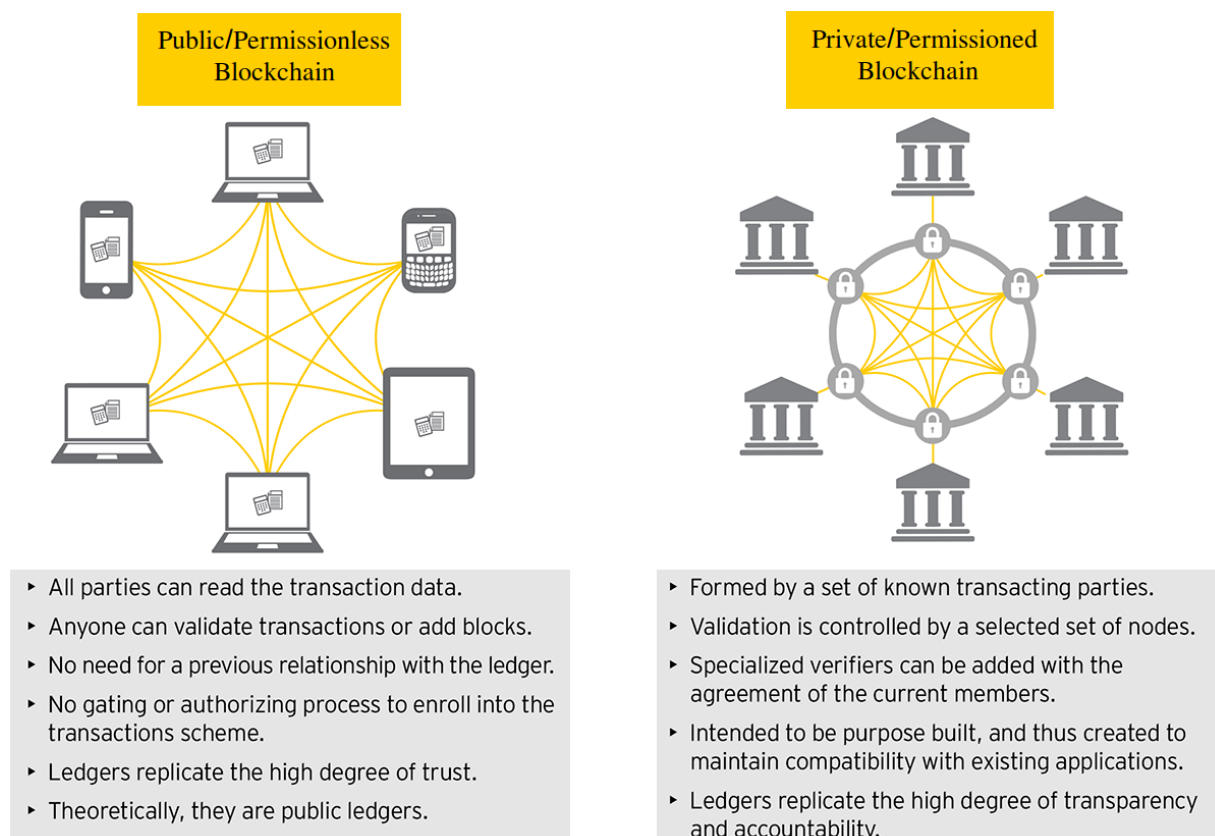


Figure 2.2: Public/permissionless vs private/permissioned blockchain.

2.1.1 Blocks and Consensus Algorithms

A block can be thought of as a page in a ledger, and each represents a data structure composed of several components. Depending on the different blockchain nature, a block can contain different types of data, although, some common properties take place no matter which blockchain is considered, for instance, an identification and a timestamp. Timestamp works as a stamp that dictates the time when the block is added to the chain. On the other hand, the identification hash works as a block ID, and it is generated

using cryptography algorithms, such as Secure Hash Algorithm (SHA), over each block's content. This identification process allows the block-linking process, where every newly attached block points to the ID of the previously added one, creating a full chain back to the original one, also called genesis block. Figure 2.3 illustrates an example of a Block, where CH2 represents ID, and M2 represents Timestamp. All the other variables might differ for different blockchains[14].

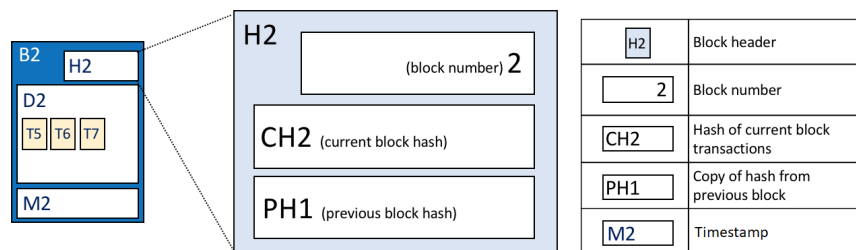


Figure 2.3: A standard example of a Block.

According to Figure 2.3, a block link (PH1) is secured by the way its hash is generated. Considering, for example, Ethereum blockchain, which is further detailed in Section 2.1.3, blocks are a collection of essential pieces of information together with the comprised transactions (T5, T6 and T7)[15]. The current block hash (CH2) is generated using M2, D2, PH1 and block number, and this is how a blockchain resists to any mutation or attempt to make a change on its blocks. Since CH2 is created according to its content, changing a block's content would also change its CH2, breaking the PH1 link and resulting in a mismatch among all the following blocks. Breaking the PH1 link would bring the need of recalculating every block ID after the changed one, which would be seen as a fraud because, as said before, the ledger is shared across all the peers on the network[15].

Consensus algorithms are the key concept that makes every blockchain run. Since blockchains are distributed systems, every network node needs to agree on adding various blocks to the chain. For this to happen, there must be a consensus protocol able to solve general byzantine problems [16]. Different nodes provide data through peer-to-peer communications, and consequently, some might be malicious and have the intention to damage the system. The rest of the nodes are then responsible for distinguishing the information that has been tampered and filter it from the consistent results. This process is named Consensus Algorithm[17]. Different blockchains practice different types of Consensus Algorithms where among the dozens of them, the most famous is Proof-of-Work, after its success at the current date of Bitcoin blockchain[16]. On Sections beneath, it is introduced what we believe to be the most important Consensus Algorithms for this dissertation.

Proof-of-Work

Proof-of-Work (PoW) is a computation problem that is meant to be difficult to perform but whose result should be simple to verify [18]. It is a protocol, and a requirement for a node to prove that they have performed a costly action to protect the network from attacks, such as spam or Denial of Service (DDoS). To secure the blockchain against tamper attempts, the network requires PoW to be performed on blocks. An example of consensus algorithm is Bitcoin's blockchain PoW that requires a node to find a value that begins with a predefined number of zero bits when hashed, and the computational cost grows exponentially with the number of zero bits required [7]. This kind of solution-verification algorithm is known as partial hash inversion PoW and is used to mine blocks.

Mining

Mining is the term used to the process of adding new blocks to the blockchain. Mining a block requires a validation of its content and is commonly referred to blockchains that perform over a PoW consensus. It refers to the necessary process of generation, transmission and validation of transactions. It ensures stability, secure and safe propagation of the asset from the payer to payee[7].

Miners are network nodes in charge to make these validations among the PoW algorithm and they do it by reaching consensus through solving mathematical puzzles. In exchange for completing those puzzles, miners get rewarded with a value which, for the case of Bitcoin blockchain is the Bitcoin cryptocurrency[7]. This mechanism prevents miners from generating new bitcoins out of nowhere, keeping the fairness and integrity of the system[8]. Miners make use of computational power to solve the PoW problem, brute-forcing an arbitrary number, also called nonce, until one of them creates an expected result. It is massively expensive to find the right nonce, but at the same time, it is straightforward to verify it[17]. The miner who finds the right nonce to mine the block gets rewarded, obliging miners to compete between each other for the prize. When a miner solves the PoW problem, it broadcasts the block to all the network nodes, which consequently re-verify the transactions in the block[7].

It has already been proved that this mechanism holds a couple of flaws. It is highly expensive and needs too many energy resources to be maintained[19]. Also, the biggest issue at the current date is named "51% attack"[19]. This attack takes place when there are two conflicting blocks vying for addition to the blockchain. In case an attacker would have the mining power majority, it could mine his block into the chain, where this block could contain fraudulent data[19].

Proof-of-Stake

Proof-of-Stake (PoS) aims to replace the way of achieving consensus in a blockchain. It proposes that a node would validate and add a block to the chain by providing proof that it has access to a certain amount

of coins before the block being accepted by the network[20]. On a PoS consensus, the digital currency has the concept of coin age. Coin age is the coin value multiplied by the time period after it was created. The node who longer holds the coins, the more rights it can get in the network.

Furthermore, holders of the coins from a PoS consensus also receive a particular reward according to the coin age. With the concept of coin age, the blockchain is no longer entirely relying on PoW which effectively solves the resources waste problem in PoW. The security of the blockchain using PoS improves with the increasing value in the blockchain[17]. To have a successful attack on PoS blockchain, the attackers would need to accumulate a large number of coins and at the same time, hold them long enough.

2.1.2 Digital Tokens, Identification and Transactions

Tokens are not new, and its concept exist since long before the emergence of blockchain. Traditionally, tokens can represent any form of economic value. For instance, worldwide currencies such as Euro, US dollar and Renminbi, even stocks or poker chips can be considered tokens. However, the concept of the digital token has become most famous due to the introduction of blockchain. Most tokens have inbuilt anti-counterfeiting measures to prevent people from cheating the system, which may be more or less secure depending on the way these tokens are built. For this study case, the tokens use blockchain and consensus mechanisms, such as the ones described in the Sections above, to prevent participants from adulterating the system. For these tokens to be valuable, just like any other asset, they need to follow the natural laws of supply-demand, and therefore they need to be able to transact, i.e. to change ownership. This process is called transaction and generally follows the steps illustrated by Figure 2.4.

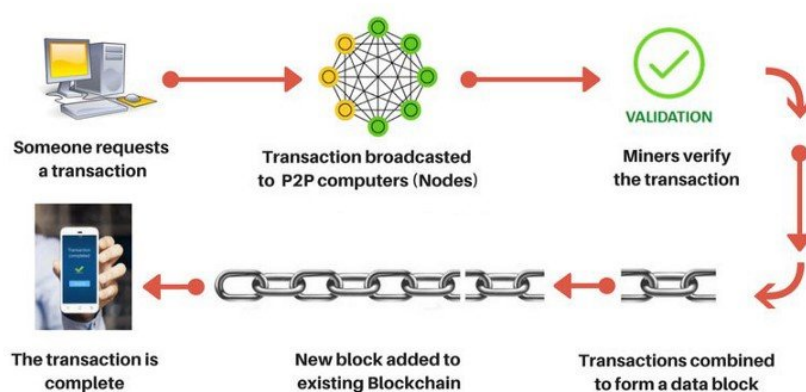


Figure 2.4: Transaction process on a PoW blockchain.

Transactions may be seen as a way of altering the overall blockchain ledger state, consequently this

change must be processed through the consensus algorithm. For Figure 2.4, the transaction request must be broadcasted along with a fee to pay miners for the transaction validation. To perform the first step of the image, i.e. to request a transaction, token owners need to sign a hash of the previous transaction digitally. Thus, assuming A wants to send tokens to B, these entities have to be identified uniquely to perform within the network, leading blockchain to introduce another important concept of digital identification.

Major blockchains like Ethereum and Bitcoin have their digital identification based over asymmetrical key encryption[7][15]. Asymmetric cryptography refers to a type of cryptography whereby the key that is used to encrypt the data, also named Public key (pubKey), is different from the key that decrypts it, named Private key (privKey). A privKey is fundamentally a number between 1 and 2^{256} , however, the key concept is to find a secure source of entropy to randomly generate it[21]. Later, the generation of the pubKey can be achieved with the outcomes of complex asymmetric encryption algorithms, such as Elliptic curve cryptography, El Gamal and Diffie-Hellman[22]. Both pubKey and privKey are large hexadecimal numbers, and as the name goes, the privKey should be kept private to its owner, while the pubKey can be publicly known[21].

In a blockchain context, a key pair composed of a pubKey and a privKey is the integral identity of a user. Users can make use of their privKey to sign transactions of assets they own, whereas the pubKey is used to verify these transactions and therefore validate them or not[21]. A third concept named address is derivated from the pubKey and is the unique public network identifier of a user, e.g. an Ethereum address is formed by the first 20 bytes of a SHA-3 hashed pubKey[15][21]. From a blockchain point of view, these key pairs can also be recognized as account identifiers[21], which in turn, depending on the blockchain, comprises different fields, such as a *balance*. In other words, it is possible to publicly audit the balance of a particular account using its address. Although, it is only possible to make use of that balance when proving its ownership by making use of the privKey[21].

Different pieces of software were developed to manage these accounts, named digital wallets. At the current date, there are many different digital wallets on the market, like Blockchain³ and Electrum⁴, and these are important for participants as they offer a way-into the blockchain ecosystem. Digital wallets are ideologically similar to ordinary physical wallets having the particularity of just be unlocked using the owner's privKey. Furthermore, wallets can create different accounts or interpret privKeys to provide access to its correspondent funds. These should not store privKeys online, but instead, offer users the possibility to save them locally on their devices.

³<https://www.blockchain.com>

⁴<https://electrum.org>

2.1.3 Ethereum

The Ethereum project consists of a platform that is built for developers and application users to deploy and consume distributed applications[15]. These applications, also called Decentralized Applications (dApp), run with no possibility of downtime, censorship, fraud or third-party interferences[15]. Ethereum monitors the state of every account, as well as all the blockchain transactions with its PoW consensus algorithm. In other words, Ethereum may be characterized as a decentralized and cryptographically secure, transaction-based state machine[15]. An essential concept of Ethereum the Ethereum Virtual Machine (EVM). EVM is a system interpreter, just like a common Central Processing Unit (CPU), but instead of acting locally on a machine, it redistributes its outcomes to other nodes on the network[21]. Within the Ethereum context, EVM is responsible for interpreting the original source code of the dApps and consequently deploy it throughout the network.

In this context, each user of a deployed dApp provides the application to other users. In exchange, the consumers of the application use Ether⁵ to interact with it[21]. Additionally, the developers who intend to deploy their dApp have to pay Ether to have it deployed in the platform, paying for the mining process explained previously. Therefore, Ether is a cryptocurrency generated by the Ethereum platform and used to compensate mining nodes for producing consensus algorithms. Each Ethereum account has an Ether balance and it may be also transferred from one account to another[15]. As Ether is openly owned and transferred, at the date, numerous marketplaces have been developed to trade it against Fiat⁶ currencies. This led Ether to fall under the supply and demand economic model that determines the Ether price in a worldwide market. At the date, as most cryptocurrencies, Ether market is extremely volatile, which causes its market price to vary significantly over time. This price variation can occurring for different numerous reasons, from which stand out: the fact that cryptocurrencies have smaller market sizes as compared to Fiat currencies, meaning that even small movements of Ether can have a pronounced affect on its price[23]; media influences people's perception of this technology and can lead to spikes on the price charts[23]; the abstention of government regulations and other legal requirements is delaying adoption as there must be regulations for this new type of technology[23].

Apart from the economic side of Ethereum, according to the state of dApps⁷, a leading web application on dApp exploration, more than 1500 dApps are running on top of the Ethereum platform and, every day more enterprises are adopting this technology to fulfil their needs. Many different dApps are already deployed and successfully running over the Ethereum Network. This dApps are evolving models in market segments such as finance, social media, gaming and music with remarkable results. They are pushing

⁵the underlying token powering the Ethereum blockchain

⁶legal tender whose value is backed by the government that issued it, e.g. Euro or U.S. dollar

⁷<https://www.stateofthedapps.com/>

the blockchain industry forward towards an era that is less about speculation and more about results, adoption, and making a substantive improvement in users lives.

Smart Contracts

Smart contracts term was first introduced by Nick Szabo[24], and are held as the most essential feature of blockchain technology[25]. Smart contracts are autonomous agents that can replicate code functionalities of legal contracts amongst other advantages[26]. They can be seen as cryptographic "boxes" containing code that has been written with a specific purpose of doing a particular task, so, this code only triggers if certain conditions are met. Smart Contracts can be built on top of different platforms, such as Ethereum, that shows itself vastly more potent than Bitcoin scripting, because of a better turing-completeness, value-awareness, blockchain-awareness and state[25].

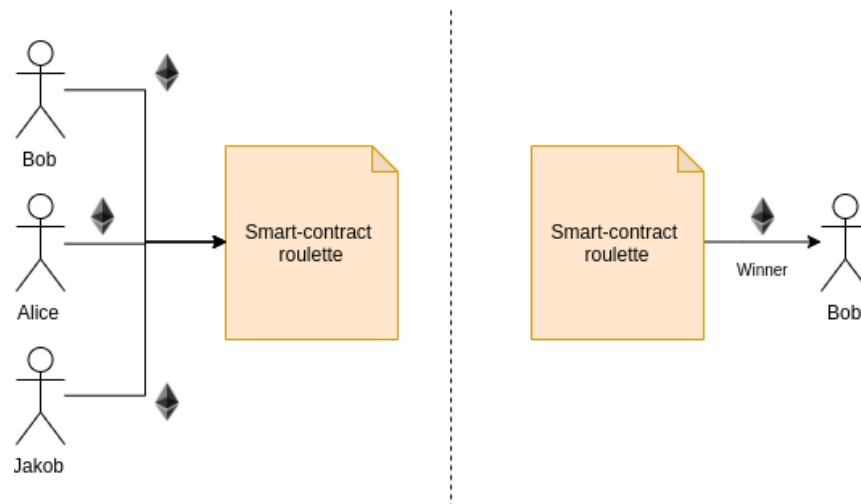


Figure 2.5: Example of a Smart contract application.

Smart contracts can be used for any purpose, but its most famous usage is among digital assets transactions according to arbitrary pre-specified rules. A more specific example of a smart contract is illustrated by Figure 2.5, where a casino roulette is developed in the shape of a smart contract. The whole living purpose of this smart contract is to receive Ether and select a winner randomly among the participants. In this particular example, it is possible to see Bob, Alice and Jakob depositing a certain amount of Ether on the smart contract roulette. Consequently, the smart contract selects one of the three depositors based on the algorithm set on the development of the contract. After having a final selection, the smart contract automatically clears and settles the agreement, sending the winner, Bob, the previously deposited amounts. The logical extension of smart contracts is the decentralized autonomous organizations

— long-term smart contracts that contain the assets and encode the by-laws of an entire organization or enterprise[25].

Smart Contracts development and deployment

Different applications and frameworks were already created to ease the development of Ethereum smart contracts. Ethereum smart contracts were initially coded in its contract language Serpent and Mutan (both deprecated) and moved forward to the most recent Vyper and Solidity[27]. Solidity is the focus of this Section as it is better documented and better supported in terms of development.

Solidity provides a big set of features that ease the execution of a smart contract. When a smart contract is written in Solidity, it's not the actual code that is executed into the Ethereum network. Instead, the contract definition built on the machine is fed into a Solidity compiler. This compiler has then two separate outcomes: Bytecode and Application Binary Interface (ABI). The bytecode is the actual code deployed into the Ethereum network, containing the meaning of the smart contract. Bytecode is not meant to be understood by a non-machine entity. On the other hand, ABI is the key for writing applications that can interact with deployed smart contracts, i.e. it is a layer of translation that provides a back-end application with the ability to understand Bytecode that was previously deployed. Unlike bytecode, ABI is legible. Being human-readable is essential because it gives a user an understanding of how exactly he or she can interact with a contract.

Alternatively, as Ethereum issues rely mostly on its compiler bugs, EVM bugs and attacks on the blockchain network[28], other ways of creating smart contracts have been developed, like Lisk⁸. Lisk project provides a development kit that offers the ability to develop smart contracts or dApps using JavaScript, one of the most popular programming languages in the world. Familiar underpinnings reduce the entry barrier to practically nothing, making it a good starting point for developers who wish to dive into blockchain dApps development.

Ethereum Transactions and its parameters

An Ethereum transaction is a single cryptographically-signed instruction constructed by an actor out of the Ethereum scope. These transactions can be seen as the creation of smart contracts or an interaction with an existing one. To create a transaction, each requires a set of parameters explained after table 2.1. To encourage network computation power, there needs to be an agreed method for value transmission. To address this issue, Ethereum has its currency called Ether, as explained in Section 2.1.3. This currency can be fractioned into many levels among the smallest, Wei. Every transaction value of the currency is

⁸<https://lisk.io/>

calculated over Wei, since it can represent minimal fees, but yet not free of charge[15]. Table 2.1 reveals the relation between Ether fractions.

Multiplier	Name
10^{18}	Wei
10^{15}	Szabo
10^{12}	Finney
1	Ether

Table 2.1: Fraction relation between Ether currency [15].

Being explained, a new concept is introduced: Gas. Gas is a common word that refers to a unit on the Ethereum community for its internal price representation for running a transaction over Ethereum. When setting up an Ether transaction, some input parameters are required, such as: **gasPrice**, a Wei quantity to be paid per gas unit throughout all costs wasted on computation power as a result of the transaction execution[25]; **gasLimit**, the maximum amount of Gas units someone is willing to spend on a particular transaction. The payment for Gas is made in Ether, and must take place before the transaction, which means it must take place before any computation is done and may not be increased any time later after broadcast[25].

Meta-mask and Test nets

Concluding from Section Ethereum Transactions and its parameters, it would be challenging for a developer to practice any work over the main Ethereum network since transactions broadcast require Ether to be mined. Ether on the main network is worth actual Fiat currency, as explained in section 2.1.3. That problem is overcome after the creation of test-nets. At the date, three Ethereum test-nets offer developers an opportunity to test dApps, having to use Ether within those networks. Each behaves similarly to the main Ethereum network where the real Ether tokens reside, although, test-net Ether tokens were created for development purposes, meaning that those are test-worth only, and have no Fiat price attached to them. Developers may have a preferred test-net, and projects typically are developed only in one of them, from which stands out: **Ropsten** - a proof-of-work blockchain that most closely resembles Ethereum; **Kovan** - a proof-of-authority blockchain, started by the Parity team⁹. Ether to work on this test-net can't be mined, and so it has to be requested through online faucets; **Rinkeby** - a

⁹<https://www.parity.io/>

proof-of-authority blockchain. Ether to work on this test-net can't be mined it has to be requested through online faucets.

Each test-net can generate an address and keys for a user account, and those work the same way as the main Ethereum network apart that there is no financial value involved. Figure 2.6 depicts a Google Chrome extension named Metamask¹⁰ that offers an easy and secure way to interact with all three Ethereum test-nets described above.

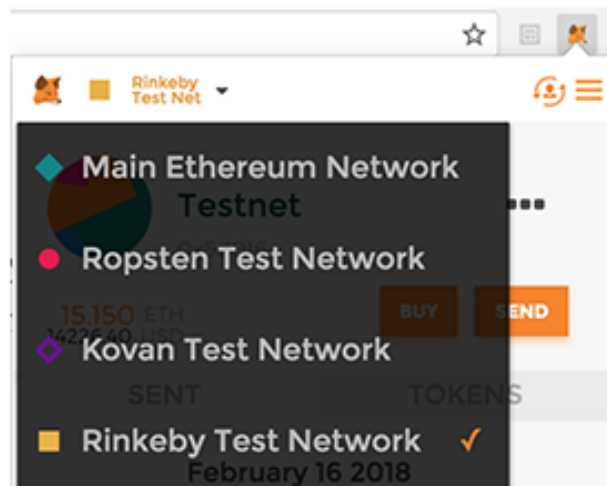


Figure 2.6: Metamask Google chrome extension.

Metamask is an extremely developer-friendly tool that allows developers to test their dApps without being part of the Ethereum network as an Ethereum Node. Not only, but Metamask also manages privKeys and pubKeys as a wallet, on behalf of a user, allowing them to send and receive Ether.

2.1.4 Hyperledger

Hyperledger is an open-source collaborative effort made to enhance cross-industry blockchain technologies. It is a hosted and fully supported by The Linux Foundation with intentions to improve Internet of Things (IoT), supply chain management, manufacturing, among other fields[29]. Open-source software products from Hyperledger are freely available for anyone to see, use, copy, and change it. It is a project designed to develop Permissioned blockchain technologies, to use between enterprise networks, and technical decisions after taking place by a group of selected enterprises[29].

¹⁰<https://metamask.io/>

Hyperledger Fabric

Hyperledger Fabric (HIF) is the largest sub-project under Hyperledger and is an example of a permissioned blockchain framework that abstracts the cryptocurrency complexity. It has been designed ground up to target enterprise requirements, like digital identification, decentralized infrastructures and supply-chain management, by offering significant benefits to those enterprises[14]. Digital identity, in particular, is fundamental for most industry use cases, from handling supply chain challenges to facilitating security-rich patient-provider data exchanges in healthcare. Hyperledger tackles digital identities problems by allowing a permissioned system, for example, only the entities participating in particular actions would have knowledge and access to their history of acts, bringing other entities no need or interest to these.

Permissioned blockchains also unlock other orders of greater magnitude such as scalability in terms of transactional throughput and level of trust by being built on a modular architecture, separating transaction processes into three phases: distributed logic processing and the agreement also called chaincode; transaction ordering; transaction validation and commitment. HIF supports pluggable blockchain consensus protocols to ensure that transactions are validated according to the policy selected by the designated business network participants.

Ledger, Chaincode and Nodes

A ledger in Fabric consists of a database defined by a World State and a blockchain together[14], comprising every participant and asset state on the world state while following blockchain methodologies. Assets are items of property owned by a person or company (participants), regarded as having value and available to meet debts, commitments, or legacies. Some examples of assets can be defined as a car, an aircraft, a contract, building maintenance record, a vote or a promotion.

The world state represents the current asset's value on the ledger in the form of key/value pairs, e.g. "Car - BMW", whereby Car is the Key while BMW represents the value. This state is advantageous because applications mainly need the current value of a ledger state, and that is always available, excluding the need to traverse the entire blockchain to calculate a current value of any ledger state[14]. Figure 2.7 illustrates the relationship between all components of HIF[14].

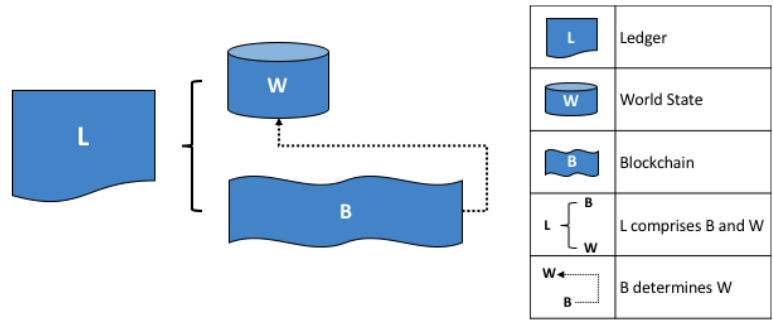


Figure 2.7: Key concepts of HIF components.

HIF manages the ledger through a type of smart contracts, which HIF calls: Chaincode. Just like smart contracts, chaincode does not only implements business logic. It also contains asset definitions and states, as well as business intellect to change them. The mechanism used to interact with the chaincode consists of transactions that can be called by applicants, depending on their permissions to do it so. These transactions should not be seen as common transactions but, instead, they should be thought of Create Read Update Delete (CRUD) operations on any resource, such as participants or assets. Participants may or may not call different transactions to, depending on the business logic, update asset parameters that they own, or retrieve queried data[14]. In a business network, many chaincodes can be dynamically implemented and therefore communicate between each other, so together update the world state of the assets. Lastly, nodes are the communication entities of HIF, and multiple nodes can run on the same host server. Thus, what matters the most is how they get grouped within the same trust domain and how they are associated with logical entities[14]. There are three types of nodes explained beneath, amongst them: Peer Nodes, Orderer and Client Nodes.

Peer Nodes

Peer nodes are a fundamental element of HIF network since they host ledgers and chaincodes and, they are responsible for committing transactions into the ledger. Peers can be created, started, stopped, re-configured, and even deleted. They hold an Application Programming Interface (API) that expose the services they provide to other applications, leaving the chance to interact with it if permitted[14]. Figure 2.8 illustrates the relationship between peers, ledgers and chaincodes[14].

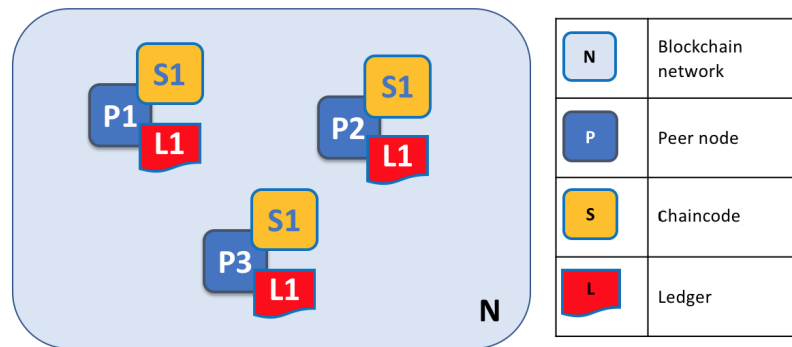


Figure 2.8: Example of three peers holding chaincode and ledger.

Peer nodes may additionally take up two particular roles called: **endorsing peer** and **anchor peer**. Endorsing peers validate the certificate used to submit a transaction according to a set of endorsing policies, and if valid, it simulates the chaincode. Important to mention that it does not execute it, meaning that it does not save the state of the ledger[14]. Anchor peers act as the middle entity between Orderer and Client Nodes.

Orderer and Client Nodes

Client nodes represent entities that act on behalf of an end-user. They must connect to a peer node to communicate with the blockchain. Client nodes submit transaction-involutions to the endorser peers, explained in Peer Nodes, and broadcasts transaction-proposals to the ordering service. These ordering services are formed by Orderer nodes, i.e. a fabric communication that provides delivery guarantees. In other words, ordering service provides a shared communication channel between client and peer nodes, offering a broadcast service for messages containing transactions. Clients connect to the channel and may broadcast messages on the channel which are then delivered to all peers by the anchor peer[14]. As an example, Figure 2.9 explicitly illustrates a simple relationship between all existing nodes[14].

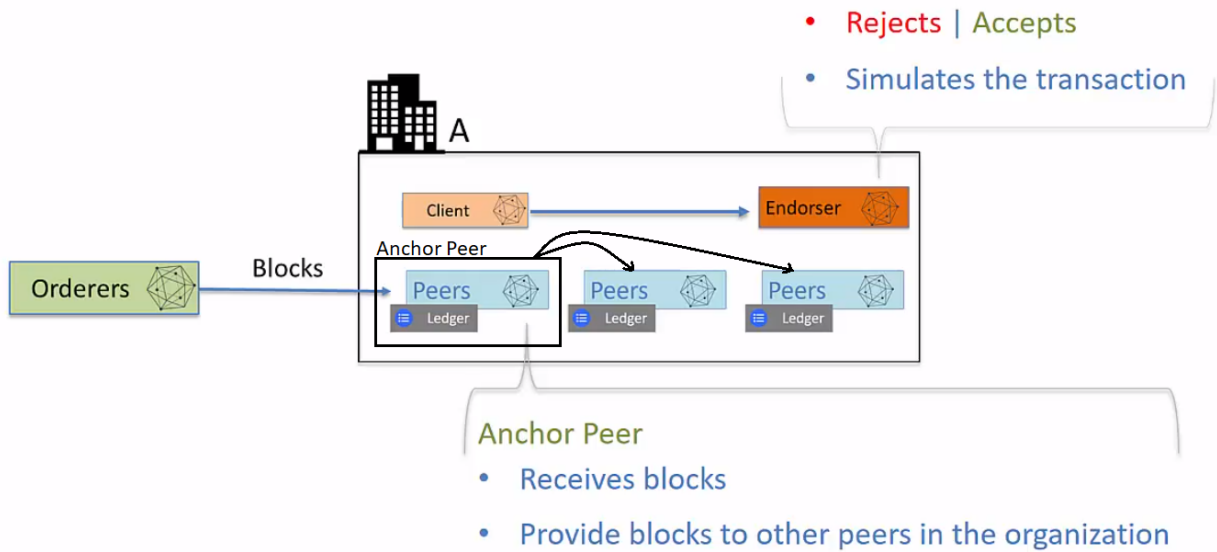


Figure 2.9: Standard relationship between HIF nodes.

Hyperledger Composer

Hyperledger Composer (HIC) is an extensive, open-source framework built over HIF that offers a valuable set of tools for development, making blockchain applications easier to progress[30]. HIC priority is to step-up into a new level of time-wasting and efficiency by simultaneously easing the integration of blockchain applications with existing business models and systems. HIC can be used to develop use cases and deploy a blockchain solution in a short period, as it offers a set of tools already configured. It allows developers to model a business network and integrate existing systems and data within their blockchain applications[30]. HIC is most relevant due to its compatibility with the existing HIF blockchain infrastructure and runtime. Applications can consume the data from business networks, providing end-users with simple and controlled access points. Thus, HIC can quickly model and test a business network, containing all the resources (assets, participants and transactions) related to it. Assets, as said previously in different words, are tangible or intangible goods, services, or property while transactions represent functions to update data[30].

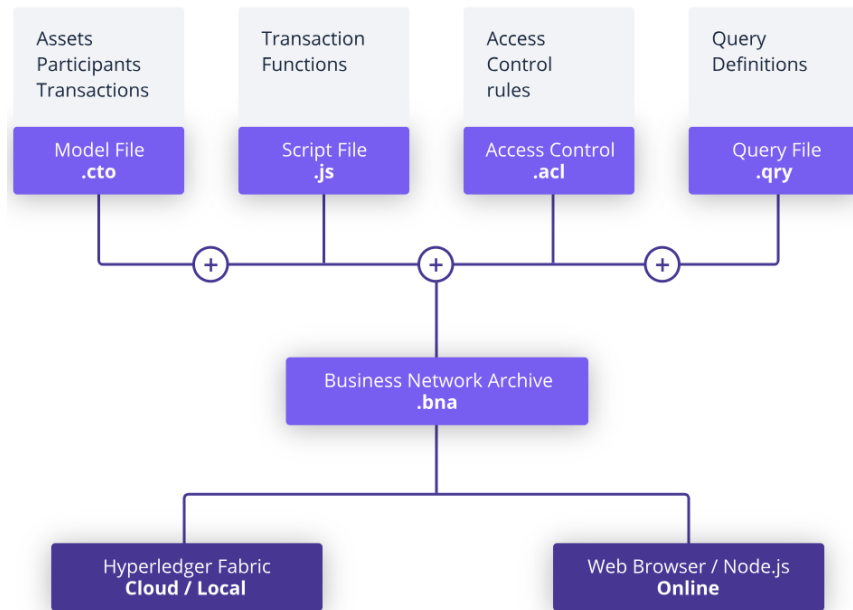


Figure 2.10: Hyperledger composer elements.

According to Figure 2.10, Composer can be used to create a business network definition, comprised of:

- **model .cto file** - Contains the structure definition regarding every resource such as asset, participant, transactions and other types of data.
- **script .js file** - Contains the functionality of every transaction stated in the model .cto file.
- **access control .acl file** - Contains declarative access control over the elements of the domain model.
- **query .qry file** - Contains queries to filter results and returned them using different criteria.

Once all the files are set up, HIC packages them up and exports it as a Business Network Application .bna file. This file can then be deployed into HIF and be used by multiple users within the network[30]. Another valuable component of HIC is a Representational State Transfer (REST) server called composer-rest-server. It can be used to generate a REST API from a deployed blockchain business network that can be easily consumed by Hyper Text Transfer Protocol (HTTP) or REST clients[30]. The REST server can be configured using different environment variables, such as: *composer multiuser*, that once set, activates the multi-user mode, enabling the business network to distinguish different clients of the REST server[30]. Many of these environment variables should be taken into consideration before being enabled or not, according to the business network model and specifications.

Apart from these tools presented above, the HIC also has the Hyperledger Composer Playground, a web application to develop and test business network applications using web-browser's cache for database

memory[30]. It is undeniably a useful feature for developers to start understanding blockchain, and most importantly, HIF concepts. It provides an excellent user interface with useful tools to configure, deploy and test brand new business networks[30]. Advanced Playground features give the users the chance to manage the security of their business networks as well as to create, delete or update participants and assets to their business networks, issue or revoke different identities and connect to multiple blockchain business networks[30].

2.2 Loyalty programs

Loyalty programs bring a different idea of customer-vendor relationship and play a crucial role in the sales of any vendor. A broad spectrum of vertical consumer markets make use of these programs, such as hotels, airlines, car rental companies, entertainment firms and even e-commerce, e.g. Amazon¹¹ which is an American multinational technology company focused on e-commerce, or standard retailing industry, e.g. Lidl¹², a German supermarket chain. Loyalty programs are marketing strategies designed by merchants according to their business logic. They are meant to encourage customers to return to stores where they frequently make purchases. Incentives provided to customers may differ from the strategy used, although they commonly include advanced access to new products or services, additional discounts or sometimes even free merchandise. Even though this concept may seem simple in its surface, it is supported by different complex marketing strategies, which took years to improve by constant research and analysis[31]. After understanding the complex general background of these loyalty programs, later on in this document, different types are described and explained.

2.2.1 How loyalty programs work

As said previously, in other words, customers loyalty programs are coordinated marketing strategies. These programs are part of an emphasis on defensive marketing, i.e. activities which focus on retaining existing customers and getting more activity from them rather than focusing on acquiring new customers. While other marketing activities, e.g. advertising or service quality raise, may have a positive impact on repeat-purchase loyalty, any increase is majorly accidental rather than being a primary objective[31]. At the date, most loyalty programs offer privileged or even free activities in a membership-based system. In other words, customers typically register their personal information under sellers predefined terms and conditions. Consequently, it is offered to the customer a unique identifier, such as a numerical ID or membership card. This ID classifies each customer uniquely within the loyalty program and is consequently

¹¹<https://www.amazon.com>

¹²<https://www.lidl.com>

mandatory on every purchase, as it is responsible for granting access to any leverages[1]. The natural world evolution has inevitably increase society's demand for their life-styles quality raise. Rewarding customers activity could potentially be used as a motivation to satisfy these demands since it is always an advantage to get rewarded with something rather than nothing. Although, at the date, sharing personal data with businesses is currently a very delicate topic after Cambridge Analytica data scandal in early 2018, where millions of companies were not General Data Protection Regulation (GDPR) compliant[32], which led to client's mistrust. However, accessing customers data is fundamental to vendors as it brings the most value out of these loyalty programs, by filtering customers activity to accomplish their marketing strategies. Therefore vendors must disclose their practices and inform customers how their data is being collected, used and shared[1].

2.2.2 How loyalty programs add value

It is not the purpose of this dissertation to explain the marketing strategies of individual firms, but rather to examine in a generalized way, the value added by these strategies. From a vendor point of view, in practice, granting a price discount over a service or product is a non-profit operation. Although, studies prove that by adopting loyalty programs, vendors attract more customers who might conclude in a more beneficial and profitable business[31], based on a simple principle: spend more to get more. In membership-based loyalty programs, the ultimate marketing objective is their use of data-gathering platforms that can help improve the efficiency and effectiveness of a firm's marketing initiatives. Often loyalty programs allow marketers to capture detailed customer's information and preferences, allowing them to filter customer's interests and consequently make adequate suggestions to its customers. Therefore, the most significant benefit acquired from loyalty programs resides in the data and knowledge gathering that firms may use to develop statistical models to improve customer loyalty[1].

To customers, these loyalty programs are somewhat misleading. Firstly, the value added to a customer always depend on the way the marketers develop their loyalty program. Second, loyalty programs are built to favour sale charts, and even though customers may feel advantaged, most of the time, they are pursued to buy a service or product that they did not need. However, they still buy it because there was a discount applied to it. Last but not least, a downside of loyalty programs is that customers may feel slighted when their interactions with the vendor are inconsistent with what they believe it should be.

2.2.3 Types of loyalty programs

Now that is clear the meaning and the value of loyalty programs, different types are further described throughout this Section. In order to analyze the best type of loyalty program to implement the solution,

some problems are stated later on this document together with the proposed solution to fix most of them.

Point Program

Point programs are the most famous loyalty program[33]. In this program, recurrent customers earn points from each purchase, which may then be exchanged for some value, depending on the offer. The most common scenarios at the date are exchanging points for discount, gifts, or exceptional customer treatment. This type of loyalty program is most suitable for businesses to encourage short-term purchases[33]. Point programs represent its assets as untraceable points, i.e. points do not have identification and therefore can not be tracked.

Voucher Program

Voucher programs consist of the issuance of gift vouchers that contain value and validity date. Typically, the seller who issues the vouchers is their owner and is in charge of define and distribute those vouchers among their customers. Upon acquiring this gift vouchers, buyers are encouraged to return to the vendor who has issued them and spend the amount held by these vouchers[33]. The voucher program is one of the most flexible programs, as many variables can be added to this type of program. For instance, the vendor may define a minimum purchase amount, forcing customers to purchase a specific amount of goods to use their vouchers, resulting in increased goods sales[33].

Tiered Program

Tiered programs usually have what it is named "loyalty ladder". It is a strategic way that offers small rewards as a base reward for customer activity. This program encourages customers to repeat purchases by increasing the value of the reward according to their position on the loyalty ladder. The more customers consume, the higher they climb the loyalty ladder. A significant difference between point programs and tiered systems is that customers get long-term value instead of short-term[33]. Tiered programs may have a better outcome on high commitment businesses, like airlines, hospitality businesses, or insurance companies[33].

Paid Program

In paid programs, customers pay an upfront monthly or annual fee to join a Very Important People (VIP) club with access to specialized services, discounts or unique opportunities. This type of programs must include benefits that are exclusive to members, otherwise, it would lose its value. Paid programs

are most suitable to customers who want to be relieved of inconveniences that could impede their future purchases[33].

Store Card Program

Store card programs are similar to paid programs, with the alternative that customers provide their data instead of paying a monthly or annual fee. As the trend goes, "data is the new currency", and big enterprises may take advantage of user's data to strict analytic research on their consumption habits. Store cards are often free and offered to attract consumers by giving away promotional prices, frequent shopper points and coupons for future purchases[34]. Store card programs are typically used by financially-large companies, which then restrict their access to itself or its enterprise group.

Partnered Program

Partnered programs are strategic partnerships between distinct vendors. This programs can be extremely effective regarding customer retention, as it is offered more variety of rewards. Currently, this strategy may help the growth of a business by building new business partnerships. Partnered programs differ from other types of programs by providing customers with value that not only one business can offer, but as many as partners[33]. In other words, a customer would be rewarded with every value from every partner of the program. This may show customers carefulness regarding their needs.

An interesting trust behaviour of partnered programs is when a vendor offers a particular value to a customer, and it then takes advantage of this value in another vendor. This process creates an imbalance between both partners, and as a result, a debt assertion needs to create equity for them. Briefly, and further on detailed, a debt assertion is the process where a vendor pays its debt to another vendor after a value transition.

Hybrid Program

Hybrid loyalty programs are a combination of two or more types of loyalty programs. Merging two different programs may be the most suitable solution for businesses with a particular set of requirements. For example, a point-based program together with a tier program. This combination may make use of points to ease the customer's placement throughout the loyalty ladder. This visual context can potentially trigger customer's eagerness to pursue next loyalty levels and thus, more purchases.

2.3 Related work

Stated previously in Section 2.2, companies are forced to compete for profitability, sometimes offering customers incentives to get their loyalty. Loyalty programs are not merely about offering assets, instead, it is a channel to strengthen customer relationships. This Section separates examples of related work in the areas of non-electronic, electronic and blockchain-based loyalty programs, referring to the traditional use of paper and mobile and desktop applications. It is undeniable the fact that loyalty programs are taking over many commercial businesses and for this matter, it is essential to know some of the cases applied in real use cases. Thus, a survey of existing work takes place, ensuing results on the Sections beneath.

2.3.1 Non-electronic loyalty programs

Every day on we witness the conversion of old fashioned physical and non-electronic loyalty programs to the electronic systems. Despite needing to overcome some issues, electronic control over loyalty program's assets provide customers with many benefits like ease-of-use, flexible tariffs, monthly billing, reduced time consumption and sales predictions[35]. As stated in Section 2.2, businesses are starting to adopt electronic loyalty programs. Although, some of them might not be financially strong enough to afford one, or even have no interest to adopt an electronic system. For these, there is still much use of non-electronic loyalty programs. Taking the opportunity to go on the field, we visited a few small marketers in the areas of Leiria, which are making use of different loyalty program strategies. A few local examples are, for instance:

Lidia's Health Food, a small restaurant based in Leiria's old town. This restaurant makes use of a loyalty stamp card program. Loyalty stamp cards follow the typology of a Point Program. With slight differences, these cards are papers held by buyers that get stamped by the vendor after products or service purchases. For this particular case, one meal offers the client one point, which is in the shape of one stamp. After a limited number of stamps, ten for Lidia's Restaurant case, it entitles the customer the right to have a free meal. This method is commonly used by small marketers, as it is cheap and easy to perform. Although, this program can be easily adulterated, for example, making a copy of the stamper. Even though some other layers of security can be added to this types of loyalty programs, such as signing the card over every newly added stamp, it is still possible and most importantly, simple to fake.

Another example is Galerias do Lis car parking. This car parking is based outside a small shopping mall named Galerias do Lis, and makes use of a Partnered Program typology. Together with the Hairdresser Louro Moreno, based inside Galerias do Lis, both companies take advantage of their relationship by offering significant price discounts to each-others customers. In other words, hairdresser's customers get a discount at the parking site and vice versa. Even though we took the chance to speak with both seller's coordinators, they stated that the relationship was purely a "spoken contract", and all their resources were provided

through signed parking tickets. More specifically, if a car parking user would present a valid car parking ticket after a haircut, they would receive an immediate discount on the purchase and vice-versa. Both vendors also mentioned that the debt settlement was executed, face-to-face, at the end of the month so that no party would be losing out of this program.

Apart from those smaller merchants we visited, there are numerous other examples. As the last example, let's take into consideration a more prominent marketer, Continente¹³. Continente's suppliers make use of a Voucher Program to enhance their sales, while Continente takes advantage of its business size to endorse a central way of distributing those vouchers, making use of a Store Card Program. In other words, to have access to those vouchers, a client must have a Continente Card, which it can get by providing its data, and later on, identify him or herself as a customer. Inside any Portugal-based Continente, a customer who owns a Continente card can go to a machine that prints a paper voucher with a bar-code, holding a specific discount based on the user's consumption profile. This methodology is functional, although this system which issues the vouchers is centralized within Sonae's group and may not be used to other companies.

Excluding Continente's case, even though those antiquated systems are used, ideally, every reward system should be electronic. Due to the fact that not only it is impractical to carry coupons and parking receipts every day, also it would be a significant advantage to avoid the use of natural resources, like trees. Non-electronic loyalty programs also carry a significant risk of fraud as papers are easily printable or changeable. However, depending on the case scenario and the targeted achievements, there are also advantages of using non-electronic loyalty programs — for instance, the ability to have higher anonymity and ease to exchange value between users. In case the business strategy wants to ignore the identification of its assets as well as the way they are transacted between end users, then the use of non-electronics loyalty programs is most adequate. As technology evolved, loyalty programs started to convert to electronic systems, unlocking several features that may not be acquired by non-electronic systems.

2.3.2 Electronic loyalty programs

Electronic loyalty programs are getting more frequent and adopted on the daily-basis, having already been adopted by many international and national enterprises, such as airline companies, restaurant and market chains and even small businesses like some mini-markets and hairdressers[35]. It is an undeniable change that the world of loyalty programs is taking by adopting systems that prevent resource waste and minimizes the probability of fraud in many aspects. The use of electronic systems came along with new benefits for adopting commerces, which non-electronic loyalty programs are unable to offer. For this case, some examples are Amazon, which makes use of gift cards or coupon codes in the shape of a Voucher

¹³retail chain that belongs to Sonae group

Program. This company, like many others, make use of strings to validate if a user is entitled to a specific discount or not, as shown by Figure 2.11.

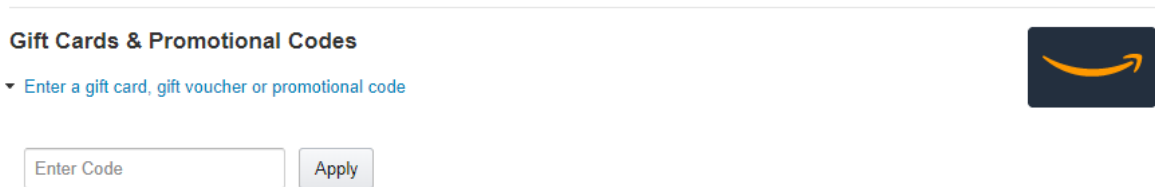


Figure 2.11: Part of Amazon UK billing page that offers clients a code string input.

Amazon issues these string codes and distributes them among customers the way the business wants, selling or giving away. Whoever provides them back to it is entitled to a direct discount, always depending on the issuer policies. These policies can encompass any clauses the issuer desires, for instance, the issuer can imply the voucher use to specific items, require a minimum purchase amount or demand that the code is redeemed during a specific time window. This mechanism is an excellent method to have an available promotion on a time frame set by the seller or even to draw customers' attention to specific products.

Another example is Mlovers¹⁴. Owned by the multinational company McDonald's, Mlovers is a mobile application that offers customers the possibility of creating accounts, having to identify themselves by their phone number. After making any purchase on any Portuguese McDonald's restaurant, customers may opt to identify themselves with their phone number and be rewarded with points according to their order, following a Point Program typology. Figure 2.12 shows one of several screens of Mlovers, displaying the number of points the customer possesses on the top-left corner and the available offers on the center-bottom of the screen.

¹⁴<https://www.mcdonalds.pt/mlovers>

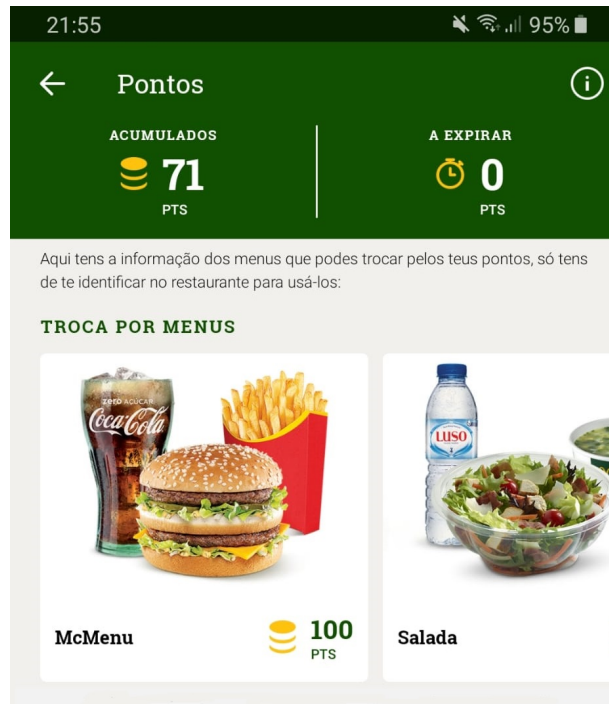


Figure 2.12: Part of Mlovers application screen.

McDonald's is the entity responsible for issuing promotions where they offer food in exchange for points. Just like McDonald's, this technique is approached by many other multinational enterprises, as it encourages users to buy more in order to achieve a certain amount of points to spend. As a strategy to force customers to raise their consumption, Mlovers does not allow the exchange of points between different users and also defines an expiration date for these. Although this technique is widely known across different multinational enterprises, it is important to refer that this program is centralized around one entity only, McDonald's for this case. In other words, this entity is in charge of the issuance of points, allowing limitless market cap for Mlovers points. This fact comes with a significant flaw regarding points traceability, i.e. points can not be traced by their instantiation. Instead, they have to be tracked by the user. On the other hand, as points are not uniquely identified nor controlled, they can be generated and elimination with no meaningful consequences.

The last example refers to Benfica's membership card¹⁵. Benfica is a professional sports club based in Lisbon, and unlike the other commerces described above, it puts together a Hybrid Program composed by a Partnered Program and a Paid Program. This membership card has the shape of a standard credit card and presents a monthly cost between 3€ and 50€ depending on the type of subscription. Figure 2.13 shows some of the many existing offers provided by a valid Benfica membership card.

¹⁵<https://www.slbenfica.pt/en-us/loja/socios>



Figure 2.13: Some of the many offers provided by Benfica’s membership card.

When subscribing to Benfica’s loyalty program, customers unlock every offer provided by Benfica’s partners. As an example, users that acquire first-class tickets from Comboios de Portugal¹⁶ have a 15% direct discount on the final billing price. Not only, membership card owners also get privileged games seats, custom services among several other perks. This loyalty program approach may have the biggest impact to customers as they do not need to worry about validity of assets. Although, it is also the most intrusive from all the examples because it requires a substantial amount of capital to support a monthly subscription. A monthly billing system is one of the main reason to repel customers from loyalty programs because it makes customers wondering if a monthly payment is worth it.

Like the three exemplified cases above in this Section, many other companies are struggling to offer their customers the most exclusive loyalty programs. The need to reach a perfect balance between customers retention, partnership and profitability led to studies like this dissertation, which seek a reliable and trustworthy solution. The growth of digital loyalty programs with its many formats is phenomenal, and this type of market still has space to improve its disadvantages. Some of those disadvantages can be entirely solved or partially improved through the use of new and innovative technologies like blockchain. For this reason, the next Section is dedicated to expose existing blockchain-based projects within loyalty programs space in order to filter good practices to us, as well as to try to avoid existing difficulties and discard possible bad technology practices.

2.3.3 Blockchain-based loyalty programs

The short life span of blockchain technology is reflected on the lack of projects that hold together loyalty programs with blockchain technology. However, within the vast world of loyalty programs, some projects already took one step ahead by implementing solutions over blockchain technology, intending to take advantage of its features. After a strict research, it can be summarized to the two most relevant examples: The Rogue Project and the 5miles Tokenized Loyalty Program[36][37].

¹⁶state-owned company which operates passenger trains in Portugal

The Rogue Project

The rogue project wants to shape a standardized and decentralized protocol for managing electronic assets like e-tickets or e-coupons. It implemented a protocol named Rogue Network that makes use of a set of Smart Contracts to achieve uniformity and standardization of those assets. As explained in Section 2.1.2, Rogue created their own digital token - Rouge Token (RGE), in order to make use of its smart-contracts functionalities. This project tokens and smart-contracts run over the Ethereum network and was initially created to reduce costs, friction and the need for trusted middle-men on the procedure of creating non-falsifiable, non-repudiable and unique electronic vouchers[36]. These vouchers may contain a specific value and can be exchanged between the participants of the network.

Rogue's business model workflow is based on three different actors: an issuer - the entity responsible for setting the value and other properties of issuing vouchers; a bearer - an entity responsible for carrying an issued voucher, usually representing a customer; a distributor - someone to which issuers pay commissions to distribute vouchers[36]. Rogue aggregates vouchers through campaigns that have a global validation date[36], which for this dissertation study can be seen as a failure for limiting merchants' allowance to issue new vouchers. In other words, depending on the goal, an issuer should be able to issue vouchers every time it desires, having no need to create campaigns. Also, distributors are entities that should be able to be replaced by autonomous smart-contracts, i.e. the relation between stakeholders should be strictly between issuers and bearers by developing smart-contracts that are able to replicate distributors functionalities. Overall, Rogue's approach is adequate for this dissertation study case as it presents robust documentation and describes in detail the different roles of each entity and asset.

5miles Tokenized Loyalty Program

5miles is a free peer-to-peer local marketplace application, one of the fastest-growing online shopping ventures in the United States that launched an Initial Coin Offer (ICO)¹⁷ named Cybermiles¹⁸. 5miles want an innovative way to offer unique loyalty and referral benefits as well as promotions from sellers to buyers within the marketplace[37]. They intend to offer several money-earning opportunities to users, such as performing marketplace tasks and referring friends. A wallet for these tokens is incorporated on 5miles application, giving users the ability to spend them on in-application purchases[37]. Although this is a project hosted by a single company, bringing benefits only to itself and its customers, it is a role model project since it intends to benefit every customer of the Cybermiles blockchain. 5miles makes use of the Cybermiles blockchain smart contracts in order to unlock several different features to its users, such as the ability to make purchases by sending payments to smart contracts that hold product listings, upcoming

¹⁷Type of funding using cryptocurrencies

¹⁸<https://www.cybermiles.io/>

decentralized escrow payments and offering tokens for different task completion, e.g. reviewing sellers and referring friends. Word of mouth marketing is one of the most effective marketing tactics and, by using the networking power of social media, many iconic companies, like Dropbox and Airbnb, achieved hyperbolic growth with referral campaigns. 5miles believes that adding the power of blockchain to this already proven marketing strategy takes referral programs to the next level — when incentive rewards are paid in a limited release blockchain-based token, it unlocks the network ownership effect for the users, a double economic incentive[37].

Multi-agent loyalty program

As it is intended to develop a solution that brings multiples parties together working over the same loyalty program, it is necessary to first to contextualize this Chapter. A Multi-Agent System (MAS) is a system operated by a group of entities that pursue together to achieve a wide variety of goals, such like building constructions, supply chain management[38], and for our particular study case, to operate over the same loyalty program. According to the study done in State of the art, it is reasonable to develop a solution based on a hybrid loyalty program, which blends an electronic-voucher program typology with a partnered program. These programs are chosen because, as this solution intends to bring together several selling entities, it makes perfect sense to follow a partnered loyalty program. As for electronic-voucher typology, the choice relies mostly on the fact that vouchers can be traceable rather than points, as explained in sections above, and also involve no subscriptions. With these two choices into consideration, this Chapter starts by stating the current daily problems (section 3.1) that come together with the choice previously taken. Then it is introduced an overview as well as the objectives for the solution as a proof of concept (section 3.2), followed by stakeholders and asset definition (sections 3.3 and 3.4). Later on, use cases are defined (section 3.5) and some user stories are exemplified, based on a real life environment (section 3.6). Finally, to ensure that all system needs are addressed and taken into account, a survey of necessary capabilities and requirements to satisfy every stakeholder (section 3.7).

3.1 Problems statement

Mobile broadband, together with the internet, have empowered people on a global scale to have open access to useful data and a wide range of lower-cost and more convenient services and products. This connectivity may be proven useful and lead to a significant positive impact on society when used correctly. To achieve such impact, the solution presented in this document makes use of different technologies in order to solve daily problems presented by generic loyalty programs. This solution intends to solve not only problems faced by average buyers, but also problems faced by global sellers, namely:

- P.1 Double redemption of vouchers.** This problem is faced often in traditional systems when vouchers are redeemed more than once, creating an economic imbalance, leading to financial deficit. Vouchers must never be redeemed successfully more than once.
- P.2 Multiple customers cards.** For traditional systems, each commercial business has its loyalty program, usually involving a loyalty point card, a mobile application or another way to identify the users. The solution must prevent customers from having multiple cards or applications to different companies, making always use of the same identity to different sellers.
- P.3 Voucher data transparency.** Today's systems do not openly expose data. For instance, voucher's issuance and expiration dates, their respective user's ownership and other significant data. The solution's data and business logic must be transparent, being publicly auditable by any interested party, guaranteeing full transparency between stakeholders.
- P.4 Fake ownership of vouchers.** Nowadays, it is rather simple to fake the ownership of a voucher and make use of it improperly. As a consequence, the solution must prevent vouchers from being redeemed by anyone except their respective owners.
- P.5 Voucher authenticity.** Traditional paper vouchers are easily printed and faked as well as electronic ones can be print-screened and modified. For this reason, the solution must assure that every voucher is authentic and is issued and redeemed under specific contract definitions.

After analyzing all the problems stated above, it is undeniable that these exist in our modern world. Thus, they are targeted by the solution to be either eliminated or less problematic. Finally, after understanding the issues, it is time to start describing the solution.

3.2 Solution overview and objectives

This solution proposes a multi-agent loyalty program solution that consists of a general digital payment processor with a built-in discount system based on vouchers, which may be openly used by different

commercial businesses and customers. For the sake of simplicity and a better interpretation throughout the document, commercial entities and customers are henceforth addressed as **Companies** and **Clients** respectively.

Companies may promote clients activity by rewarding them with vouchers upon their purchases. Consequently, these vouchers hold a specific value established by the company who issued them, which can then be redeemed on any other company who provides the same sort of payment processors. Since the solution is meant to operate between different independent companies, one of the biggest challenges is to provide a decentralized trust protocol that grants transparency among them all. To enhance this trust between the different participants the solution focus does not target the development of a new payment processor, but instead, a plug-in extension that interacts with existing ones (fixing problem **P.2**) providing then an auditable and open system.

Using this solution, companies are free to set their own rules according to their own marketing strategies, meaning each company is responsible for setting its voucher’s value, as well as how/when they are acquired by clients. Additionally, when buying goods or services from a company, clients are entitled to acquire or redeem vouchers depending on the rules previously set by the companies. Figure 3.1 briefly illustrates the concept of the solution, starting with a company X issuing several vouchers identified from 1 to N. Consequently, clients 1 and 2 acquire vouchers 1 and 3 respectively, and finally they redeem their vouchers at companies Y and Z.

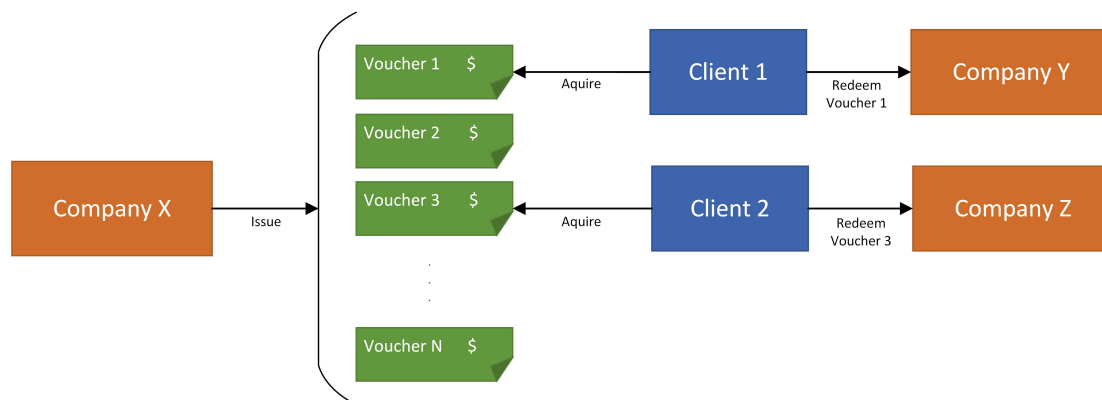


Figure 3.1: Concept overview.

Upon a successful workflow of Figure 3.1 company Y and company Z would reach a sale deficit, as they offered the voucher’s value to client 1 and 2. Here is where a debt assertion needs to take place. In traditional partnered programs, as described in section 2.2.3, the issuing company would personally have to assert its debt. So, to avoid any conflict between parties, this solution takes one step ahead by forcing issuing companies to allocate the value of voucher’s issuance into the solution platform. This strategy can be seen as a disadvantage because companies have to provide the voucher’s value in advance. However,

this makes possible to immediately reimburse the company that receives the redeemed voucher. Also, this strategy restricts the execution of transactions to the solution’s functionalities, not allowing the assets value to be withdrawn under non-contractual specifications. For instance, in case a voucher expires before being redeemed, its value must return to the company who issued it. Being said, it is conclusive that this solution must behave as a settlement mechanism between the different stakeholders. It must ensure that all the debt assertions take place automatically, removing the need for a trusted middle-man that commonly takes care of this process.

To offer an objective awareness of how this solution is meant to work, Figure 3.2 exposes a real example case of a traditional digital voucher code (1) that has a value of 26.80€ (2) over the items to be purchased. This example e-commerce accepts the most common payment processors to the date (3), like Visa, MasterCard and Paypal.

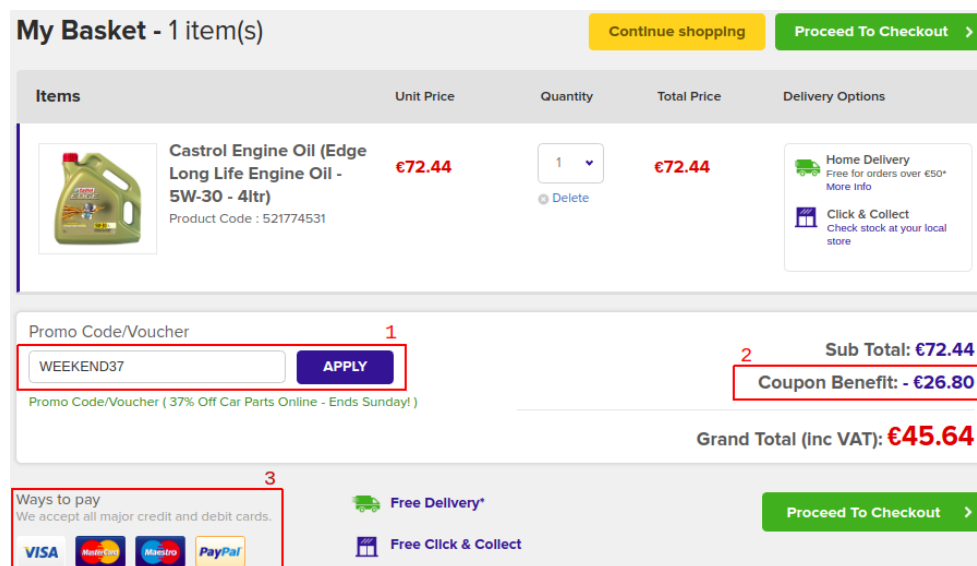


Figure 3.2: Checkout state from www.eurocarparts.com.

Hence, the solution may not oblige any change in e-commerce’s User-interfaces (UI). Instead, the functionality behind the system input (1) must be connected to our solution. Consequently, the input is henceforth defined according to details described further in this Chapter.

The solution platform can enable the use of notifications, providing ways to alert clients of important events, for instance, notifying clients when their vouchers are about to be expired. This solution offers companies not only an opportunity to help themselves mutually in a larger scaled loyalty program, but later on, may also be flexible enough to implement other types of incentives. Such as encouraging customers environmental activities, e.g. offering vouchers to clients who shop to struggle against environmental issues, like rewarding public transportation consumers.

It is a complex solution with different use cases regarding different stakeholders activity. For a better understanding, these are now sectioned, starting with the stakeholders' definition.

3.3 Stakeholders

Starting the solution description, it is fundamental to identify any possible stakeholder. A solution successfulness depends directly and exclusively on its stakeholders, whether they are developers, sellers, customers or any other entity that has an impact or is directly or indirectly involved in the solution. For this solution, two potential stakeholders were identified, namely:

- **Companies** - Any individual or commercial business that sells goods or services publicly. For the sake of the solution, it is important to separate companies into two major stakeholder roles:
 - **Issuing Companies** - Any company that issues vouchers for existing or potential clients that offer a marketing privilege.
 - **Receiving Companies** - Any company that accepts the redemption of vouchers from their existing clients in exchange for economic benefits.
- **Clients** - A large section of general public. Persons who acquire vouchers when purchasing products and redeem those vouchers when acquiring other products, both operations in the context of a loyalty program.

As this solution is characterized by an opened MAS, having one of its stakeholders defined as the general worldwide public, it is reasonable to classify it as a public network, meaning that any individual or business can and may take part in it. However, as stakeholders are independent, they all have to reach a consensus and perform over the same terms and conditions. As natural criteria to distinguish identities, every stakeholder is identified by an Unique Identification (UID), enabling the ability to identify participants in any operation/activity that takes place within the solution's network. For a well-functioning ecosystem, each of the different stakeholders carries different roles, responsibilities and benefits regarding this solution.

In one hand, assuming clients commonly shop, these take full advantage from the implemented solution because they get rewarded for their natural need to buy goods or services, more specifically, they earn extra value in the form of vouchers. Furthermore, this solution must give clients legit guarantees about the real value and terms of their vouchers, for instance, to acknowledge clients when vouchers are out of date. On the other hand, companies may be distinguished by Issuing Companies and Receiving Companies. These two categories distinguish the companies who are issuers of vouchers and those that are receivers of the

same. It is relevant to mention that issuing companies can also be receiving companies at the same time. In other words, a company can receive vouchers when actively issuing others. Moreover, issuing companies can also receive vouchers issued by themselves, for example, considering Figure 3.1 in Section 3.2, client 1 could redeem voucher 1 to company X if desired.

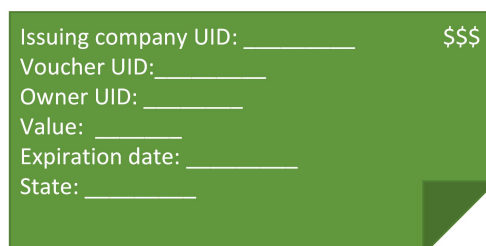
The benefits for companies to join this network remains not only, but mostly by raising their products or services visibility and awareness. The public is naturally attracted to great deals who influence their buying decisions, therefore making use of vouchers can have a significant impact within a company's economy if followed the right marketing strategy. In short, the sales of each company can increase after opening a voucher gateway, naturally increasing demand and profit.

Hereupon for a better understanding of the clients and companies relationship, it is necessary to describe the concept of vouchers deeply. Consequently, the next Section is committed to defining the interaction layer between the existing stakeholders.

3.4 Vouchers

Vouchers are digital coupons and represent the only asset of this solution. Vouchers share similar characteristics with currencies, even if they are distinct concepts, they hold value, must have an owner, and must not be counterfeited. Thus for this solution, stakeholders identified in Section 3.3 are distinctly the owners of these assets.

As briefly mentioned before, a voucher is issued by an *issuing company* and hold a particular *value* set by its issuer. Those variables may be seen as voucher properties, as they represent core characteristics of the voucher instance. Apart from those two properties, vouchers also contain: an *Expiration date* — a previously determined date after which the voucher should no longer be redeemed; an *Owner UID* — an identification of the owner who, at the time of issuance, should be the same as the identification of the issuing company; a *State* — the state of the voucher's instance. Figure 3.3 demonstrates how a voucher would look like in a traditional paper form. Furthermore, vouchers are uniquely and identified by a *Voucher UID*. Figure 3.3 demonstrates how a voucher would look like in a traditional paper form.



Issuing company UID: _____ \$\$\$
Voucher UID: _____
Owner UID: _____
Value: _____
Expiration date: _____
State: _____

Figure 3.3: Voucher definition.

Vouchers' structure, together with the chosen technology are the key concepts to solve some of the problems stated in Problems statement. Granting mandatory properties such as *owner* and *voucher id* can later prove useful to fix problems **P.4** and **P.1**. In addition, *Value* and *Expiration date* do not fix any particular problem but are necessary properties that carry an important role. *Value* contains the worthiness of a voucher and *Expiration date* holds a specific date after which the voucher should no longer be redeemed. Finally, *issuing company id* may help solving problem **P.5** as voucher's issuance requires its issuer authentication.

Moreover, to enhance problem **P.1**, a *State* property takes place to track the conditions of a current voucher. Specifically, during a voucher life-cycle, vouchers go through the following four states:

- **Free** - The voucher has been issued but has not yet been acquired by any client, i.e. its owner is still the company who issued it.
- **Acquired** - The voucher is linked to a client, potentially associated with an identity.
- **Expired** - The voucher is out of date and is no longer redeemable.
- **Redeemed** - After the right represented by the voucher has been exercised by a client in a receiving company and this receiving company is reimbursed with the voucher value.

A voucher may only take the state transitions behaviour of Free \rightarrow Acquired/Expired and Acquired \rightarrow Redeemed/Expired. This is defined so it becomes impossible to rewind states, preventing clients of making use of a voucher multiple times. Figure 3.4 demonstrates the voucher's state diagram with every available state transition offered by the solution platform.

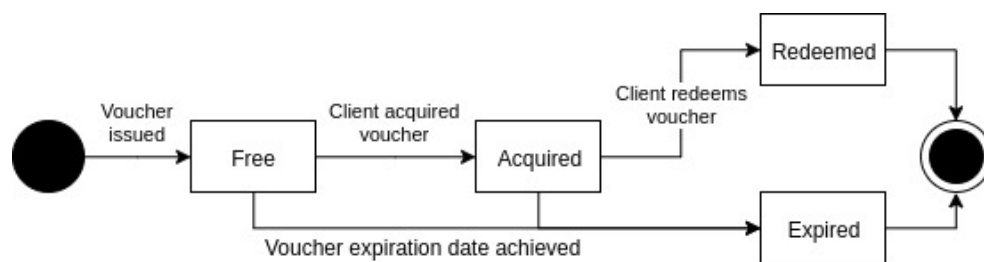


Figure 3.4: Voucher state transition diagram.

Understanding how vouchers are structured, how their life-cycle is defined and who are the solution stakeholders, the next Sections focus on the interaction behaviours between vouchers-stakeholders relationship.

3.5 Use cases

Based on stakeholder’s survey previously done in Section 3.3, it is now defined real use cases for both client and company stakeholders. There can be some possible outcomes of each different use cases, e.g. after a voucher is expired, its state must automatically change without any stakeholder intervention. Although, this Section only covers the main objective of the solution, consequently showing the use cases that are triggered by stakeholders intervention, hoping to give the best understanding of the overall solution’s functionalities. Assuming companies for this particular use case already plugged-in a functional solution to their commerce and started issuing and accepting vouchers, Figure 3.5 specifies stakeholders actions which are potentially supported by the system solution.

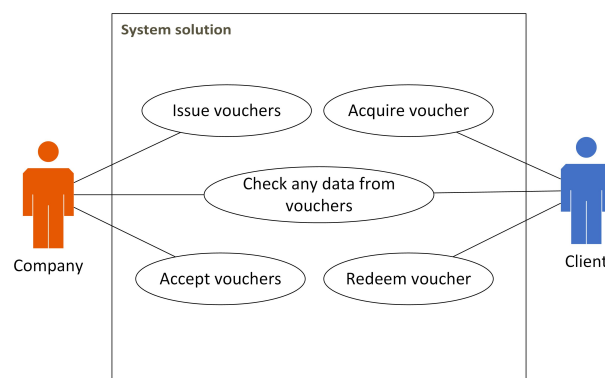


Figure 3.5: System solution use case.

Figure 3.5 demonstrates a shallow level of actions that stakeholders may take regarding the solution. It is noticeable that both actors share the right to audit and check data from the solution if necessary or wanted, fixing problem **P.3**, i.e. any stakeholder is free of consulting the solution’s data at any time. Nevertheless, for the rest of the actions - issue, acquire, accept and redeem vouchers - it is now necessary to deepen their logic and demonstrate how could they possibly progress.

To begin with, considering an issuing company, it may issue vouchers whenever it intends to if it has enough capital to support them. Issuing a voucher can be seen as an escrow, i.e. the issuing company locks a certain amount of funds into an automated escrow mechanism (our solution platform) that dictates the future of those funds according to predefined rules. In other words, issuing companies only get access back to their funds in case the vouchers get expired. Otherwise, the client who acquired the voucher can redeem it to any receiving company. Meaning that whenever a voucher is issued, its predefined set of rules are what controls the transaction of the escrowed funds, and consequently these can only be triggered by its owner.

Figure 3.6 and 3.7 demonstrates three flow chart diagrams that represent the workflow process from an issuing company perspective. Important to note that all these processes are events that can take place

multiple times. For an issuing company, the process of issuing vouchers and allowing clients to acquire them can be distinguished in two processes. First, issuing companies are allowed to issue vouchers whenever they intend to. To do so, as Figure 3.6 shows, they must immediately lock funds into the solution and set the validity of the voucher. Apart from these parameters, all the other voucher parameters must be generated automatically. For instance, its issuing and owner ID must be defined according to the issuing entity; its state must be set to free; its ID must be randomly calculated. After issuing vouchers, issuing companies must set their rules on their commerce, answering to "when does a client is allowed to acquire a voucher?". This process should not be addressed on the side of the solution platform for two reasons. First, different commerces may have different requirements and want to personalize their marketing strategies too detailed and most importantly, confidentially. Secondly, as specified previously, this solution should solely be a plug-and-play loyalty system, abstracting from every marketing strategy concept. An example of a non-technical rule is "a client is entitled to a voucher of X value if he or she buys a product that costs three times X or higher".

After rules are set and vouchers are issued, Figure 3.7 demonstrates that whenever an issuing company completes a market sale, it must validate if the sale was executed under the previously set rules' policies and, according to that validation, offering the voucher's ownership to the client or not. If the client acquires the voucher, the issuing company is no longer the owner of the voucher, having no power over its value nor functionalities. The client is henceforth entitled to redeem the voucher to a receiving company.

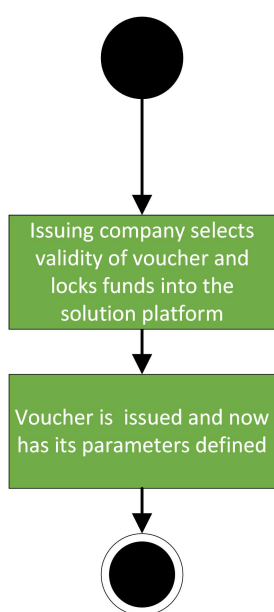


Figure 3.6: Issuing voucher and setting rules flows.

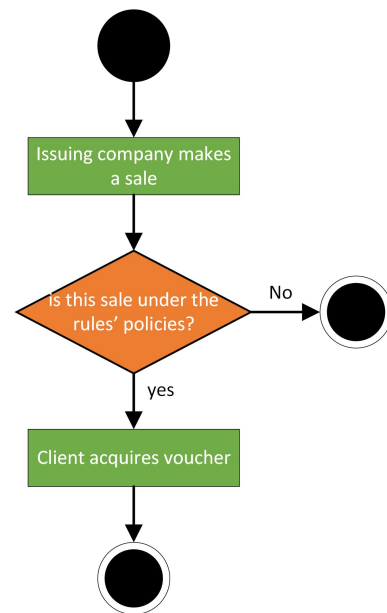


Figure 3.7: Making a sale and allowing clients to acquire a vouchers.

On the other hand, a receiving company has to implement our solution's embed input to provide

clients with the ability to redeem their vouchers. Figure 3.8 demonstrates a workflow from a receiving company point of view and, accordingly, when clients are shopping, they are prompt during the billing process to input the identification of the voucher they want to redeem. After clients input, the solution platform must validate the voucher's authenticity and the legitimacy of its owner. If valid, the receiving company must display the billing price with the voucher value deducted in it. Finally, if the client finishes the payment successfully, the solution platform sends the value of the redeemed voucher to the receiving company, achieving dept assertion automatically. In case clients do not provide a voucher identification, the company will no longer act as a receiving company, but can act as an issuing company instead.

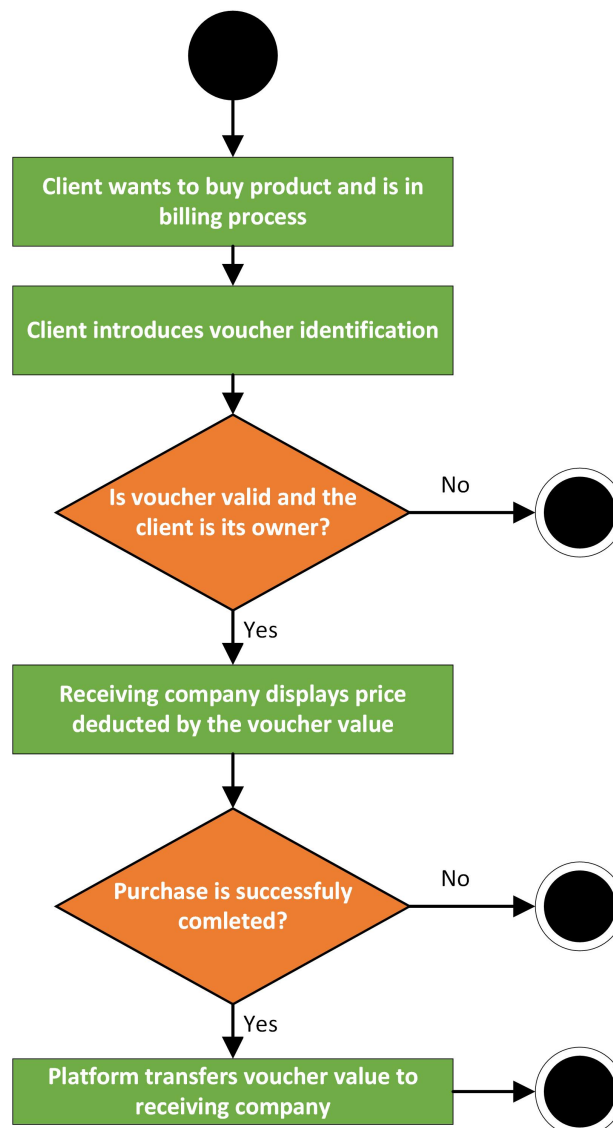


Figure 3.8: Flow chart from a receiving company perspective.

It is important to highlight the last phrase to understand that a company acts as an issuing or receiving company depending on the clients input, i.e. if a client inputs a valid voucher, the company must operate as a receiving company, otherwise as an issuing company. Furthermore this can easily be noticeable in Figure 3.9, which demonstrates the workflow of the solution — similar to a conjunction of figures 3.7 and 3.8 — although from a client perspective.

When shopping, companies offer clients the opportunity to input voucher identification during the billing process. Depending on this input, the company acts as an issuing company or a receiving, taking then the normal behaviour previously illustrated in Figures 3.7 and 3.8 respectively.

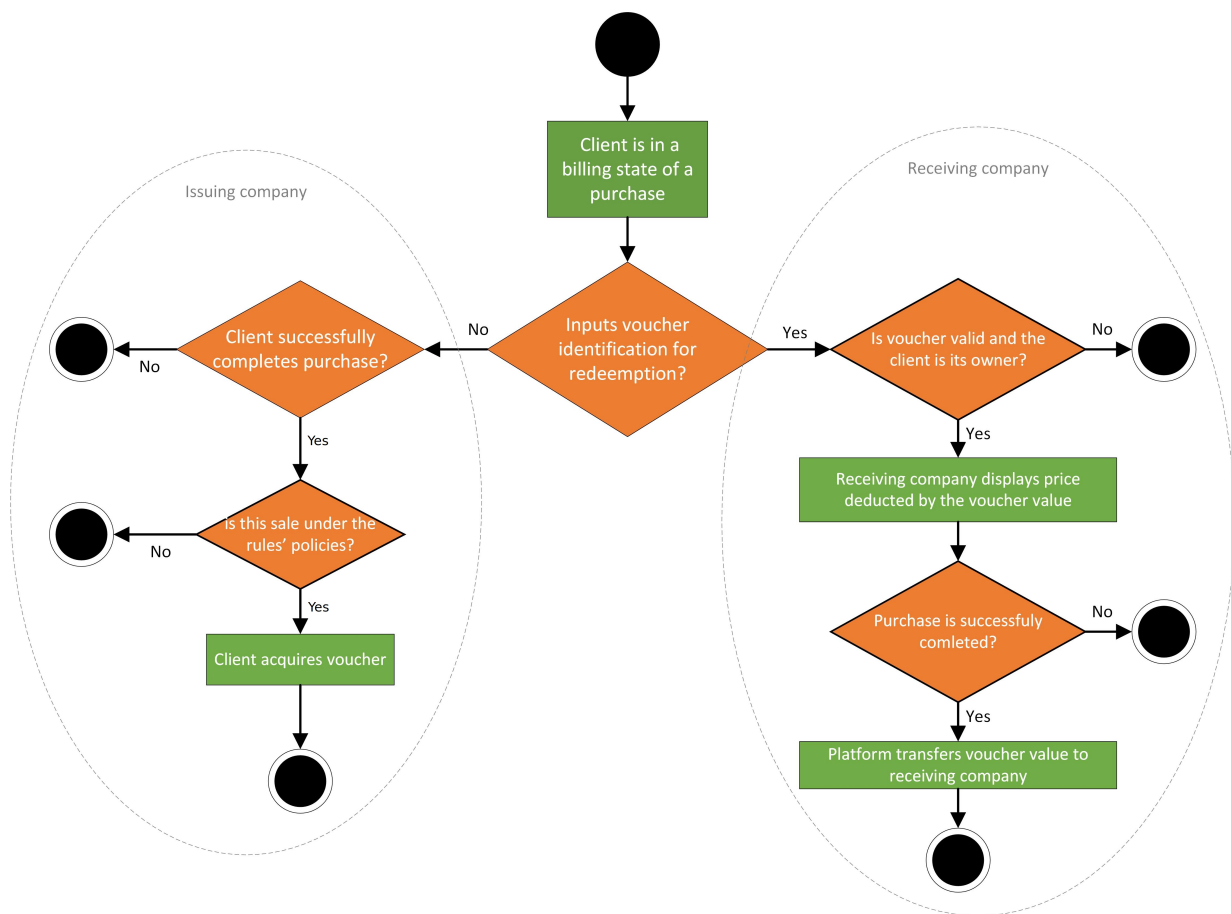


Figure 3.9: Flow chart from a client perspective.

To have a better insight of the overall behaviour of the solution platform, the next Section focuses on real life examples and possible user stories, hoping to bring the best understanding into the solution's routine.

3.6 Case scenario

In this Section are exemplified some solution's user stories. To do so, for the sake of the example it is considered three distinct companies from different business fields: Amazon (e-commerce), TAP (airline), CinemaCity (cinema operator) and three particular clients: Elias, Jakob and Sarah. All participants behave according to the solution's overview concepts, stakeholders and assets definition, and use cases described in the Sections above. To ease the thinking process, let's assume that:

- User stories defined beneath are presented chronologically.
- Companies have already integrated the solution plug-in to their payment processors.
- The number of issued vouchers, as well as their respective value, are arbitrary numbers.
- For the sake of simplicity, the value of vouchers is presented in €, and dates are represented in DD/MM/YYYY.

Companies want to promote their services, therefore let's arbitrarily set that each issue two vouchers. Also, every voucher is issued on the same date — 1st of January 2000, and each contains different value, hoping to lure potential clients differently. Important to note that, at this point, issuing companies already locked the value of the voucher into the solution platform, being the owners of the voucher but no longer having direct access to those funds. Table 3.1 contains the initial status for every stakeholder and asset list. Along with the user stories is presented the same table with different results, bring a summary of the participant's and asset's state upon procedure results.

Client		Company	
User	Vouchers (ID)	User	Vouchers (ID)
Elias	{}	Amazon	{001, 002}
Jakob	{}	TAP	{003, 004}
Sarah	{}	CinemaCity	{005, 006}

Voucher				
ID	Owner	Value	Expiration date	State
001	Amazon	1	01/02/2000	Free
002	Amazon	1	01/02/2000	Free
003	TAP	2	20/01/2000	Free
004	TAP	2	20/01/2000	Free
005	CinemaCity	1	15/01/2000	Free
006	CinemaCity	1	15/01/2000	Free

Table 3.1: Status after voucher issuance on 01/01/2000.

Once all three clients get notified on their mobile applications about the issuance of new vouchers, they tend to be receptive regarding a rewarding opportunity. Jakob immediately wants to buy a book at Amazon, knowing he is entitled with a 1€ voucher (001) upon the purchase. While Sarah and Elias take the opportunity to book two TAP flights, receiving then two distinct vouchers (003 and 004). Later this day Jakob also wanted to book a TAP flight but quickly realized by auditing the system, that all TAP vouchers have been assign. Since he wanted to get more vouchers, he booked a cinema ticket at CinemaCity and therefore acquired a second voucher (005). At this point, table 3.2 demonstrates changes presented on the ownership and state of some vouchers, consequently changing the owned vouchers by each stakeholder.

Client	
User	Vouchers (ID)
Elias	{004}
Jakob	{001,005}
Sarah	{003}

Company	
User	Vouchers (ID)
Amazon	{002}
TAP	{}
CinemaCity	{006}

Voucher				
ID	Owner	Value	Expiration date	State
001	Jakob	1	01/02/2000	Acquired
002	Amazon	1	01/02/2000	Free
003	Sarah	2	20/01/2000	Acquired
004	Elias	2	20/01/2000	Acquired
005	Jakob	1	15/01/2000	Acquired
006	CinemaCity	1	15/01/2000	Free

Table 3.2: Status after some acquisitions on 01/01/2000.

On the 10th January of 2000, Elias wanted to buy a skateboard at Amazon. When he was processing his bill, he requested the redemption of his voucher. By mistake, he inputs the wrong voucher ID, accidentally trying to redeem Jakob's 001 voucher. The system failed the validation due to the wrong ownership of the input voucher, leading Elias back to the billing process. Realizing his mistake, he corrected the voucher ID to its own 004, proceeding automatically to the deduction of 2€ in the total price. Satisfied with the result, Elias concluded his purchase. After the purchase is concluded, a transaction from the solution platform to Amazon took place with the value of 2€, concluding the life-cycle of voucher 004.

After six days, on 16th January, with the series of occurred events, table 3.3 demonstrates the status of the user stories.

Client		Company	
User	Vouchers (ID)	User	Vouchers (ID)
Elias	{}	Amazon	{002}
Jakob	{001}	TAP	{}
Sarah	{003}	CinemaCity	{}

Voucher				
ID	Owner	Value	Expiration date	State
001	Jakob	1	01/02/2000	Acquired
002	Amazon	1	01/02/2000	Free
003	Sarah	2	20/01/2000	Acquired
004	-	-	20/01/2000	Redeemed
005	-	-	15/01/2000	Expired
006	-	-	15/01/2000	Expired

Table 3.3: Status after redemption on 10/01/2000 and expiration on 15/01/2000.

Since it is already the 16th of January 2000, voucher 005 from Jakob expired, notifying him that he just lost a 1€ voucher. With this event, the solution platform must execute a transaction of the voucher's value back to its issuer, has it had no client to redeem it, finishing life-cycle of voucher 005. Also, voucher 006 from CinemaCity expired, returning the value it locked during the issuance back to its possession, finishing life-cycle of voucher 006.

Being notified, Jakob realized he still owns one voucher in his right to redeem. Therefore, he booked a movie ticket at CinemaCity, making use of his 001 voucher and consequently getting a total price reduction of 1€. Again, after this voucher redemption, the solution platform transacted 1€ to CinemaCity, finishing the life-cycle of voucher 001. Coincidentally, on the same day, Sarah bought a ring at Amazon and made use of her voucher 003, leading the solution platform to transfer 2€ to Amazon, finishing life-cycle of voucher 003. After all these events, the status table would look like 3.4. Needless to say that after the 1st of February 2000, voucher 002 would turn expired if not acquired and redeemed in time.

Client	
User	Vouchers (ID)
Elias	{}
Jakob	{}
Sarah	{}

Company	
User	Vouchers (ID)
Amazon	{002}
TAP	{}
CinemaCity	{}

Voucher				
ID	Owner	Value	Expiration date	State
001	-	-	01/02/2000	Redeemed
002	Amazon	1	01/02/2000	Free
003	-	-	20/01/2000	Redeemed
004	-	-	20/01/2000	Redeemed
005	-	-	15/01/2000	Expired
006	-	-	15/01/2000	Expired

Table 3.4: Status after redemptions on 17/01/2000.

Using some real companies together with fictional users as an example, the user stories described above attempted to demonstrate all the use cases established in Section 3.5. Hoping that the solution’s functionalities, stakeholders and asset definitions are clear at this point, it is now crucial to analyze and state the solution requirements in order to adjust the development process.

3.7 Requirements

The solution proposed must be designed to integrate a real-world scenario. After presenting the scope of this solution, the requirements defined beneath must converge with the solution’s objectives, stakeholders and asset definitions. It also must conform with the use cases, and functionalities described previously along with this Chapter. Summing up, the following list of requirements have to be accomplished, to achieve a secure and functional solution:

1. **Decentralization** - to begin with, this solution must operate between independent companies who have no trust in each other. Because it must not have a third-party entity managing the solution assets, it is self-evident that the solution must be decentralized. Decentralized solutions are usually more complex as data is distributed through network nodes, and their performance depends on the number of nodes maintaining the solution. However, on the other hand, a failure of a node must not

affect the whole system, as well as the information it is storing, which results in a smaller impact in case of a successful attack;

2. **Audit trail** - the need for a decentralized solution brings the necessity for an auditable system. The solution must offer traceability of actions, events and data for audits, investigations and compliance. The ability to follow asset's records back to their origin provides numerous benefits to the stakeholders, such as offering a reliable way to track their assets data and a way to defend their records for compliance. As an example, every stakeholder must be aware of the voucher's issuance within the system, also the ability to freely verify the ownership of any voucher. The implementation process must take into account that any stakeholder may have read access to every type of data;
3. **Data replication** - the solution must have its data replicated throughout the network, to guarantee no losses of data. This requirement can be accomplished by choosing the right set of development technologies. Regardless of the technology used, the solution needs to store asset's and stakeholder's data, which for a decentralized system, should be done in different network access points. Regarding possible failures of some access points, the system must keep track of the stored data, for instance, stakeholders must never lose access to their assets;
4. **Data integrity** - the data stored in the system solution should maintain its accuracy and consistency over its life-cycle. This point is a critical aspect to the solution implementation as it needs to store, process and retrieve data. For example, if an integer value represents the price of a voucher, the solution must prevent issuing companies from setting other this parameter with other variable types;
5. **Interoperability** - with regard for the heterogeneous environment of decentralized infrastructures, as well as the lack of standards, the solution must provide interface communications for different communication protocols. In other words, any entity who intends to interact with the solution must be able to do so through existing communication protocols, such as, HTTP or Web Sockets.
6. **Fault tolerance** - to intensify the purpose of data replication, in case of any failure or loss of resources, the solution must keep operative, always having to be able to store data, as well as replying to data requests. Fault tolerance can be achieved when having data replicated through different infrastructures, meaning that, in case of some infrastructure fails, others fulfill its tasks;
7. **Test environment** - to accelerate system deployment and decrease the loss of resources, functionalities must be tested against a test environment. Test environments must be developed as similar to production environment as possible, helping to produce a real scenario case. For example, tests must

cover stakeholders functionalities, like issuance, acquisition and redemption of vouchers, against a test network.

8. **Real-time data** - the solution must provide a secure data stream, as a result of the real-time requirements on most of the use cases. For instance, stakeholders must be able to inspect a voucher's information, like the expiration date and value of vouchers. To achieve real-time data, it must be exposed at all time through the different access points;
9. **Scalability** - the solution must be capable of exponential growth, keeping to operate smoothly regarding transaction and data gathering. For this dissertation case, assuming the solution becomes hugely successful and trendy, it must keep on keeping track of every vouchers ownership transaction. On some use cases, the methods used to execute CRUD operations are the key to achieve a scalable solution. Therefore, the solution should make use of technologies that bring a scalable environment.
10. **User-friendly integrations** - looking to the solution as a final product, it is not supposed to have any UI, as it is meant to integrate already existing commerce. Thus, the solution needs to be easily plugged-in into already deployed payment processors. It must be documented and therefore provide an accessible and easy way to be integrated.

Implementation

This Chapter explains the implementation of the multi-agent loyalty program described in Section 3. It firstly introduces Section 4.1 where some technologies and consequent frameworks are presented and debated to use on the solution implementation. Later in Section 4.2 it is vastly detailed the multi-agent loyalty program solution from a technological point of view, always taking into account the functionalities and requirements gathered in the previous chapter 3. Next, Section 4.3 defines how stakeholders are represented within the solution, from a technical perspective. Lastly, this chapter finishes by introducing a *factory* concept, challenging the solution to gather all data in one decentralized point.

4.1 Technology choices

Previously explained in Section 2.1, at the date, blockchain technologies are distinguished in two main types: public/permissionless and private/permissioned. To help choosing between the two types, Figure 4.1 executes a series of questions in a chain-form, leading to the most suitable technology for the solution implementation — from the two types of blockchain to a traditional database[39].

- ***Does more than one participant need to be able to update data?***

According to chapter 3, the solution must have different participants, companies and clients, that consequently are responsible to manage vouchers. Those participants' interaction with the solution is reflected through the particularity of creating vouchers and updating their state, consequently changing those assets ownership. Moreover, there can be different clients and companies that may eventually handle distinct vouchers. Therefore, different participants must be able to update data.

- ***Do you and all those updaters trust one another?***

Mentioned in Section 3.3, companies are distinct commercial businesses that rationally do not trust one another. Apart from the natural business distinction between companies, client-company relationship is also not trustful, as both parties share distinct roles on the solution and asset transaction may take place between them. Therefore, solution participants do not trust one another.

- ***Would all the participants trust a third party?***

Including a third party to the solution involves extra costs, which stakeholders naturally try to avoid. Also, for the sake of this solution, trusting a third party is not as trustworthy as trusting ourselves as participants, assuming a fully decentralized system. Nevertheless, trusting a third party entity could potentially be feasible, although, this is not on the solution's requirement scope, as it intends to move towards a decentralized system.

- ***Does the data need to be kept private?***

According to 3.7, the solution must have an audit trail. Meaning no data should be private, and any participant must be able to audit the system.

- ***Do you need to control who can make changes to the blockchain software?***

The solution requirements state that it requires a decentralized solution, meaning it must be open and accessible to any entity who wishes, to take part in it. There are action restrictions to each stakeholder role, nevertheless, those restrictions do only take place at a role level. In other words every entity may take a role on the solution, and only after, it has different ways of changing the blockchain state. In conclusion, there is no need to restrict control over who can make changes to the blockchain, as all participants can change the blockchain's state in different ways.

After the support of the previously completed survey, it is conclusive the necessity to employ a public blockchain, in order to achieve our solution requirements, and complying with our solution definition.

4.1.1 Blockchain choice

To this date, there is not an accurate way of counting the existing public blockchain projects. Although it may be roughly estimated by the number of existent cryptocurrencies shown in coinmarketcap¹, a leading website for tracking capitalization of different cryptocurrencies. Roughly 2941 cryptocurrencies are listed as of today and are likely running their independent public blockchain, despite the ones that represent tokens within a public blockchain. Based on the vast amount of public blockchains projects, the criteria used to select one took place based on five domains:

- **Smart contract compliance** - The selected public blockchain must be able to deploy and make use of smart contract functionalities. This is a major domain to consider, as smart contracts allow the execution of credible asset transactions according to predefined and unchangeable virtual logic, removing the need to have a third parties executing that logic. Smart contracts are the key concept to execute the solution's functionalities and therefore, its compatibility with the chosen blockchain must be considered.
- **Maintenance** - Blockchain is typically an open-source technology and consequently it is maintained by a global community. The selected public blockchain must have an active community that can fix current problems, implement improvement proposals and have a continuous growth of key performance indicators.
- **Libraries** - Blockchains need frameworks to interact with it. Each blockchain usually has its different frameworks and libraries. Newer blockchains typically lack the development of those libraries, leaving some developers no choice but not to select them. The select public blockchain should have a rather wide choice of frameworks and tools in order to facilitate the solution development.
- **Documentation** - Every blockchain dispose their nodes, frameworks and other implementations, despite the programming language used. A well-documented technology is halfway towards a successful implementation and therefore must be this domain must be considered in order to do not lack the documentation for the used tools.
- **Community** - An active community is essential to have other developers answering inevitable questions or even to query previous uncertainties from other developers. This domain is taken into consideration by reviewing the different communities through social or development platforms, like Github, Gitter and Twitter, hoping to gather

Being the domains set, some projects stand out for their achieved goals and roadmaps. Some of those blockchain are EOS, Bitcoin and Ethereum. Bitcoin blockchain has the largest community, support

¹www.coinmarketcap.com

and maintenance among the three options, although, it lacks the development environment. All three blockchains run over scripting language, making use of opcodes to operate functionalities, i.e. low level programming language. For bitcoin, this makes development extremely difficult due to scarcity of libraries to manage those opcodes. On the other and, EOS follows a PoS algorithm and its governance is designed to have only 21 block producers at a given time, and the overall running of the network is dependent on voting by stakeholders. The extremely small number of block producers as well as the possibility of low voter turnout lead to solution malfunction. Ethereum stands out being, to this date, the world-leading blockchain-based distributed computing platform, featuring smart contract's functionalities. Ethereum community is actively supportive in StackExchange² and also on the project's GitHub. Also, it is vigorously maintained by the community, trying to execute the best Ethereum Improvement Proposal (EIP) and fixing the exposed issues. For this reason, we decided that making use of the Ethereum blockchain would be the most suitable option for our solution implementation.

4.1.2 Frameworks choice

According to the previously specified domains in 4.1.1, like any other Public blockchain, Ethereum also makes use of open-source libraries to interact with its nodes. It holds some of the most powerful libraries and frameworks, like:

- **Web3**³ - Collection of libraries that allow developers to interact with a local or remote Ethereum node, using HTTP connection or Independent Connection Provider (ICP).
- **Truffle**⁴ - Development environment, testing framework and asset pipeline that uses EVM, aiming to ease the smart contracts development and deployment.
- **Ganache**⁵ - Framework capable of simulate a local test network blockchain for Ethereum development. Developers can deploy contracts, develop applications and run tests while abstracting all complexity involving key encryption.
- **Remix**⁶ - a powerful open-source tool that helps writing smart contracts with Solidity or Vyper programming language straight from a web application that runs on conventional web browsers. Remix has a built-in mechanism for compilers, storing accounts, debugging and deployment.

²<https://ethereum.stackexchange.com/>

³<https://web3js.readthedocs.io/>

⁴<https://www.trufflesuite.com/>

⁵<https://www.trufflesuite.com/ganache>

⁶<https://remix.ethereum.org/>

- **Infura**⁷ - a hosted Ethereum node cluster that lets users run their dApps without requiring to set up their own Ethereum node. It provides scalable API access to the Ethereum and IPFS networks.

Also, Ethereum is compliant to Metamask chrome extension, which is an important development advantage as it may be used as a ICP in Remix. This chain of choices leads to a decision of making use of Ethereum blockchain and smart contracts. For those contracts definition, the Solidity programming language was selected instead of Vyper, mostly for the faster growth it had last years and better work on developing new versions of compilers. As for the itemized frameworks — web3 was selected instead executing direct HTTP calls to Ethereum nodes because it abstracts the complexity behind Ethereum nodes APIs. Moreover, it eases the implementation of CRUD operations within any API resource or collection; ganache and infura are similar, but were selected for different purposes; ganache avoids having the development and testing process on a main or test network, creating a local network that runs over only one node. This process means that the consensus algorithm is rather faster, as consensus is reached by that single node. In other words, Ganache is suitable to test the solution’s functionalities because its state transition is rather instantaneous; infura, on the other hand, represent a remote node to a blockchain on a real test network, therefore, the consensus is not reached instantly among the several network nodes. Making use of Infura is a way of achieving a closer representation of the main network, being the most efficient way of testing performance against a real case scenario; remix and truffle are similar and may achieve the same goal. Although, the use of both tools is important for the process of development. Remix is a web application editor and is the simplest way to start the developing, deploying and manually testing the smart contracts, whereas truffle does the same and, at the same time, can be included later in projects as a build dependency. The overall use of these tools is reflected from the need to interact with the blockchain to implement our solution.

4.2 Assets definition

Following chapter’s 3 ideology, requirements and definitions, it is conclusive that the assets of type voucher must be designed digitally in conformation to the description. Important to state that the implementation of this voucher-typed solution could be designed in many different ways, different technologies and frameworks. Although, below it is presented what we believe to be the most suitable set of practices and mechanisms to cover all specifications studied throughout the previous chapters.

⁷<https://infura.io/>

4.2.1 Voucher properties

The first action to take after having a robust structure for the solution's asset is to define these vouchers on a smart contract environment. Starting by declaring its properties according to the previously done study-case of chapter Multi-agent loyalty program.

```
pragma solidity ^0.5.11;
contract Voucher{

    /// Storage
    address payable public issuer;
    address public owner;
    State public state;
    uint public validity;

    enum State {
        FREE,
        ACQUIRED,
        EXPIRED,
        REDEEMED
    }
    ...
}
```

Snippet 4.1: Voucher properties definition.

Solidity requires the first line of code to be the definition for the solidity compiler version, as specified on the first line of snippet 4.1. *Issuer* and *owner* are defined as *addresses* that henceforth represent identities on the Ethereum network. These two properties are entitled to represent the ID of the voucher's owner and its correspondent issuer. Apart from these properties, a voucher definition also has a *state* that is represented by a set of enumerated states conforming to the voucher definition in Section 3.4 : *FREE*, *ACQUIRED*, *EXPIRED* and *REDEEMED*.

Important to notice that every variable is declared as *public*. In solidity language, public variables automatically have a getter function⁸ created on compilation-time. These practices intend to solve problem **P.3** while achieving audit traceability. Thus, having public properties allow every user to be able to verify each property of every voucher instance. Last but not least, issuer is declared as a *payable*. A *address* declared as *payable* allows the smart contract to operate Ether transactions to or from that *address*.

⁸A function that returns the current variable value.

4.2.2 Voucher deployment

According to Section 3.4, every property was defined in the previous chapter apart from the voucher's value and ID. Solidity smart contracts offer beneficial functionalities that help to represent these two properties. One relevant fact that is important to be aware is that smart contracts are represented by unique addresses, just like any other identity on the Ethereum network[21]. This identification is an excellent way to distinguish vouchers from one another. In other words, once a smart voucher contract is deployed on the Ethereum network, it is generated a unique address to it on the block mining process. Thus, the address of each voucher smart contract is the identification of itself. Later on, it is explained the mechanism to keep track of every voucher address, mapped to its issuer and owners.

As for the voucher value, being represented via address, it can hold Ether just like any other address[21], allowing a useful way to structure an escrow mechanism. In other words, the voucher smart contract definition behaves as a middle-man responsible for holding and releasing Ether according to the contract definition. Hence, every voucher must be initialized in conformity to the *constructor* represented in snippet 4.2.

```
...
constructor(uint _validity) public payable{
    require(msg.value > 0);
    require(_validity > now + uint256(1 days), "Validity of function must be at least 1 day!");
    owner = msg.sender;
    issuer = msg.sender;
    state = State.FREE;
    validity = _validity;
}
...
```

Snippet 4.2: Voucher constructor definition.

Snippet 4.2 shows the initialization of every voucher instance. Following the structure, once creating a voucher, the constructor must be provided with a validity timestamp which is consequently subject of a validation process, asserting the validity of this voucher is bigger than one day. Also, the created voucher must inevitably be linked to a *owner* and an *issuer*. The voucher-owner link is essential for the smart contract to later guarantee that the owner of the voucher is who ultimately can access functionalities restricted to him or her self. For this particular case, when a voucher is issued, its *owner* is also its *issuer*, which makes sense as issuing companies issue vouchers before having clients to acquire them. Additionally, as settled in 3.4, vouchers issuance demand its state set to *free*. Last but not least, it is import to notice this constructor is *payable*. Payable constructors assert that when issuing a voucher by instantiating a

smart-contract of type voucher, the issuer must send Ether to it. At the same time, the constructor requires an amount of sent Ether bigger than zero to this contract because of the line `require(msg.value > 0)`. After a voucher's issuance, the smart contract is responsible for handling its value according to its functionalities. This mechanism, as stated throughout chapter 3 may be seen as a disadvantage, as issuing companies have to provide Ether in advance. Although, it has a major impact on the debt assertion on voucher's post redemption, as the Ether is immediately available to the receiving company. Therefore, apart from properties, voucher's smart contracts also are characterized by powerful functionalities that allow themselves to execute those operations, such as voucher acquisition, redemption and debt assertion.

4.2.3 Voucher functionalities

Smart contract functionalities may be seen as routines from any other programming language, such as JavaScripts and Python, therefore declared as *functions*[21]. Within the solidity and Ethereum environment, there are two main types of functions.

The first type of functions are called transactions functions. Transactions are functionalities that change the blockchain state or make any verification process, i.e. referring to Section Ethereum Transactions and its parameters, these functions require computation power to mine a new block that holds the blockchain state modification. Consequently, for this function to be executed, the function caller must pay for necessary gas. Gas can be calculated according to the function operations[15], e.g. an addition operand costs 3 gas, and a multiplication operand costs 5 gas. Depending on the current network gasPrice, the transaction may be more or less expensive[15].

The other type of functions are functionalities which do not change the ledger state, i.e. functions that do not change any property value or generate any outcome from a specific input. This type of functions do not need to pay for gas, therefore are instant and free. These are called caller functions and are similar to getter functions, where its only purpose is to return property values.

Caller functions

After explaining the differences between the two types of functions, according to the requirements and definitions proposed in chapter 3, there are two main *caller* functions to be defined.

```

...
function voucherValue() public view returns (uint256){
    return address(this).balance;
}
...

```

Snippet 4.3: Caller function that retrieves voucher value.

Snippet 4.3 defines the voucher’s caller function *voucherValue()* tagged as *view*, which means this function may not execute any operation that requires gas. Concluding, it returns to the caller, the current balance of the voucher instance. Any stakeholder who has access to this voucher’s address can also verify its value by freely calling this caller function.

As mentioned in Section 4.2.1, every *public* property automatically generates a caller function for itself in compile-time. In other words, when compiling a contract, the compiler searches for public variable and creates a caller function for the ones found. As an example, snippet 4.4 demonstrates the caller function that is automatically created to retrieve the current validity value of the voucher’s instance.

```

...
uint public validity;
...
/// This function is generated automatically
function validity() public view returns (uint){
    return validity;
}
...

```

Snippet 4.4: Caller function generated on compile-time that returns validity value.

The voucher’s creation displays and offers stakeholders five caller functions: *issuer()* - returns address of the voucher’s issuer; *owner()* - returns address of the voucher’s owner; *state()* - returns current state of the voucher’s instance; *validity()* - returns timestamp referring to the date which the voucher’s instance should no longer be redeemed or acquired; *voucherValue()* - returns the value held by the voucher’s instance, in Wei units.

Transaction functions

After identifying and developing caller functions, it is necessary to implement transaction functions to change the blockchain state by maneuvering voucher’s states and ownership. To begin with, now that it is

clear the differences between a caller and transaction function, it is conclusive that the constructor function defined in 4.2.2 on snippet 4.2 is, in fact, a transaction function. Due to the properties initialization and declaration of its values, the constructor function changes the blockchain state by adding a new voucher contract instance.

Apart from the constructor function, according to our requirements and as settled in Section 3.4, vouchers need to go through four states: Free \rightarrow Acquired/Expired and Acquired \rightarrow Redeemed/Expired, which means it is necessary to have at least four transaction functions to go through this states. Constructor defined in Section 4.2.2 already sets the voucher's initial state to *free*. Thus, the next step is to define the second transaction function that allows issuing companies to offer the voucher's acquisition to its clients.

```
...
function acquire(address _client) public isIssuer(msg.sender) notExpired notAcquired notRedeemed {
    if(isValid()){
        owner = _client;
        state = State.ACQUIRED;
    }
}
...
```

Snippet 4.5: Transaction function that changes ownership of the voucher.

Snippet 4.5 demonstrates a possible way to let clients acquire the ownership of a certain voucher. This function has restricted access when making use of *isIssuer(msg.sender)* as well as execution-ability according to the voucher current state. In solidity, restrictions are declared after the key word that represents the function visibility, which is *public* for this particular case. Restrictions like *isIssuer*, *notExpired*, *notAcquired* and *notRedeemed* are called modifiers and will be explained in further detail through Section 4.3.1. Function *acquire()* is responsible to change the voucher's ownership to the client, consequently setting the voucher's state to *acquired*. Although, because a voucher may only be acquired if not expired and, additionally, it also must update its own state automatically, *acquire()* function need to validate its own state. This brings up to the third transaction function: *isValid()*.

```

...
function isValid() internal returns(bool){
    if(now > validity) {
        issuer.transfer(voucherValue());
        state = State.EXPIRED;
        return false;
    }
    else return true;
}
...

```

Snippet 4.6: Transaction function that verify and update voucher's validity state.

Snippet 4.6 has a particular different visibility set to *internal*. This key restricts the access of this functionality solely to itself, i.e. the only identity capable of triggering this function is the voucher instance itself. Although this is not a mandatory procedure, tagging this functionality as *internal* may result in saving unnecessary expenses. Since vouchers already execute this validation, users do not necessarily need to execute this functionality. That being explained, *isValid()* is responsible for comparing the timestamp of the execution time with the validity property timestamp. In case the execution timestamp is bigger than the validity timestamp, it means that the voucher is expired. Whenever a voucher expires, it retrieves its value to its issuing company, therefore not concluding on value losses. After this transaction, voucher's state is set to *expired* and it is no longer valuable nor usable. On the other hand, if the voucher's validity is bigger than the execution time, *isValid()* returns the value *true*, allowing the rest of the transaction functions to proceed with its execution.

Last but not least, the forth transaction function: *redeem()*, which allow voucher's state transition from *acquired* to *redeemed*.

```

...
function redeem(address payable _company) public isOwner(msg.sender) notFree notExpired notRedeemed{
    if(isValid()){
        _company.transfer(voucherValue());
        state = State.REDEEMED;
    }
}
...

```

Snippet 4.7: Transaction function that allows clients to redeem their vouchers.

Snippet 4.7 describes a *redeem* function which purpose relies on redeeming the voucher's value to a receiving company. Once again, execution restrictions like *isOwner*, *notFree*, *notExpire* and *notRedeemed* are further detailed in Section 4.3.1. Although for now, it is important to notice that function *redeem* must only be accessed by its owner client. In other words, *redeem()* function must not execute in case any other identity tries to execute it. Just like *acquire()* function, *redeem()* also verifies the validation of the voucher's instance, following the same procedure explained in the previous paragraph in order to update the property *validity*. In case the voucher is validated, this function executes a transfer to the receiving company with the total amount of its value and consequently, set its state to *redeemed*. This function is meant to be executed at the moment of purchase, in order to balance the debt assertion. In other words, having this procedure to occur successfully, the voucher's value is sent to the receiving company when the purchase is complete — concluding on moving Ether successfully from issuing company to voucher and from voucher to receiving company with no need to involve any third party payment certifier. After understanding the functionalities of our asset vouchers, it is necessary to recognize how they are connected to its stakeholders.

4.3 Stakeholders representation

Just as in other public blockchains, every identity is represented by a key pair composed of a *privKey* and a *pubKey*, which are then indexed by an address as explained in Section 2.1.2. Self-sovereign identity has been studied for a long time, and it refers to the concept of storing identities on local devices, allowing users to provide it efficiently to those who need the validation of it, without relying on a central repository of identity data. For the particular dissertation case, user's identification is imperative to authenticate and authorize blockchain's state change requests. In other words, every time an identity wants to make use of any the transaction functions described above, it needs to identify him or herself.

One of the essential characteristics of blockchains is that they are fundamentally powered by a collection of cryptographic primitives. Consequently, everyone can quickly generate key pairs and use them to interact with the solution. Therefore, it is conclusive that both stakeholders are identified for this solution through the same mechanism: key pairs. This mechanism matches perfectly with the decentralization requirement, as any identity can participate in the solution if they hold a valid key pair. When adopting this identification mechanism, it is conclusive that the stakeholder filtration must be accomplished within the smart contract. In other words, because this solution is entirely open, any key pair can execute the solution's smart contracts, consequently going through an authentication validation inside smart contracts.

```

...
txn = contract.functions.redeem(COMPANY_RECEIVER).buildTransaction({
    'nonce': web3.eth.getTransactionCount(wallet.address)
})

txn_signed = web3.eth.account.signTransaction(txn, wallet.privKey)
txn_hash = web3.eth.sendRawTransaction(txn_signed.rawTransaction)
...

```

Snippet 4.8: Contract execution of *redeem* function from a 'wallet.privKey' identification.

Snippet 4.8 shows one of the many ways of executing *redeem* functionality from a certain voucher contract. As it is illustrated, this transaction function is being executed from an identity labelled as *wallet.privKey*. Just like this, any other private key would be able to execute this functionality. Consequently, there is a need to develop an access control layer to restrict functionalities access to different types of identities. As demonstrated by snippet 4.8, this identification must be passed along with every action.

4.3.1 Access control

As it has been mentioned in the previous Section and also throughout the last Sections, asset's functionalities must be restricted according to the rules defined in chapter 3. For instance, the client who owns a voucher must be the only entity able to redeem it, or a voucher must not be valid after its validity expires. For this functionalities' access control, solidity offers a convenient tool called modifiers. Modifiers are rather small-sized functions that run before an actual routine. Usually, they make assertions for specific requirements, i.e. whenever a function is tagged with a modifier, solidity executes the modifier's functionality, and only afterwards it runs the function if the modifier's validation is completed successfully.

```

...
modifier isOwner(address _caller){
    require(_caller == owner, "This is not the owner of the voucher!");
    -;
}
...

```

Snippet 4.9: Modifier that ensures that the caller of a function is the owner of the voucher.

Consider modifier from snippet 4.9. If we look back to snippet 4.7, it is conclusive that the function *redeem()* may only be executed if its caller is the owner of the voucher contract. Just like *isOwner()*

modifier, there can be as many modifiers as necessary, where, for our case study there are modifiers to verify each state of the voucher, allowing, or not, the execution of each voucher functionality regarding its state value.

4.4 Voucher factory

Now that a voucher contract and its functionalities are defined, it is possible to deploy it to one of the Ethereum test-networks explained in Section Meta-mask and Test nets. Despite the tool used to deploy the contract, there is always at least one output from the deployment process: an address. This address refers to the voucher contract instance and must be tracked somehow, so companies and clients are able to access its functionalities. It would be extremely inconvenient for a decentralized solution to need a centralized platform or database to keep track of all vouchers, as this would go against our decentralization requirement. For this reason, it is necessary to develop a decentralized middleware that can monitor and deploy instances of voucher contracts. Because of that, a new concept is now introduced: voucher factory.

Solidity language works somewhat close to other object-oriented programming languages, such as Java and Python. It has a class-based behaviour where contract definitions can also be seen as a class. Therefore, it is conclusive that it is possible to have multiple contracts to interact with each other and most important, a contract can deploy another contract. As a result of this mechanism, it is possible to think of a voucher factory as another contract that is in charge of managing every different instance of a voucher, eventually getting issued by companies and acquired and redeemed by clients.

Figure 4.2 offers a low-level architecture of how this mechanism would be represented and how stakeholders would interact with it.

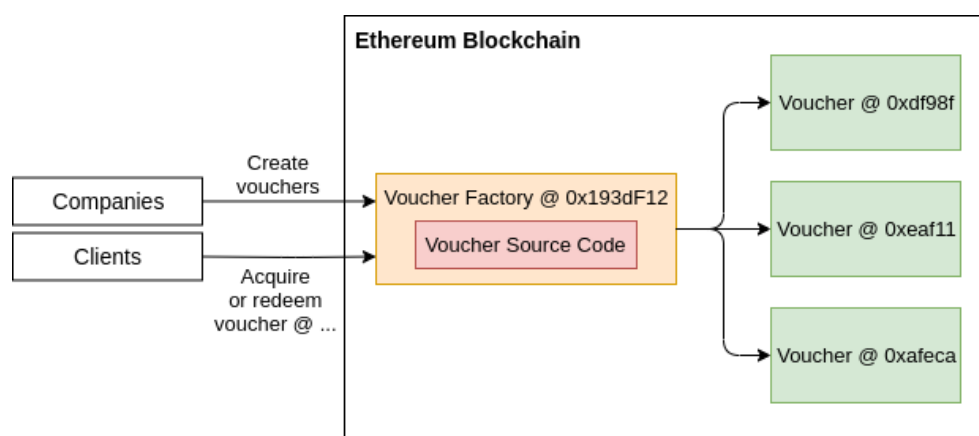


Figure 4.2: Voucher factory behavior.

A voucher factory is a one-time deployed contract that has direct interaction with each voucher in-

stance. To do so, it is used to store a list of all the different vouchers that have been deployed, i.e. this voucher factory is where every stakeholder can have one central location that gives them a canonical address list of all the deployed vouchers. The use of this voucher factory contract is a beneficial mechanism because not only it serves as the access point for every user, but it also keeps a record of every voucher instance that has been issued. As shown in Figure 4.2, the factory contract itself contains the source code to an individual voucher. Consequently, once the factory is deployed to the Ethereum network, it is forever immutable and no identity can re-write the contract or modify it in any way, shape or form. The first version of voucher factory is exemplified in Appendix A: Voucher factory.

Result Analysis

Having developed a functional solution, it is now time to evaluate if it fulfills the objectives defined throughout the document. This Chapter focuses on gathering post results of the implemented solution and along it are discussed the requirements as well as the problems the solution addressed, concluding how those requirements behave during different real life environments. As stated in Section 1.2, this dissertation had its goal focused on devising a decentralized loyalty program capable of supporting multiple companies. Being employed on a public blockchain, the solution presented throughout this document offers a reliable and secure way for multiple companies to issue valuable vouchers, offering value to its clients in exchange for customers activity, which are tested and consequently demonstrated in the next Sections.

5.1 Functionality tests

Having concluded the implementation phase, our assets are defined and the stakeholders interactions hold a robust structure. At this stage, the solution can be classified as a proof of concept that is ready to be tested. Thus, this Section focus on making use of the frameworks chosen in Section 4.1.2 to test every voucher functionality with the use of different stakeholders.

To test the whole solution system, a connection to an Ethereum network must be set. In order

to accomplish this connection, different approaches may be considered, for instance, to set up a local Ethereum node or, set up a connection to a remote Ethereum node or, test our application using a private local Ethereum network. For the sake of the solution tests, the ganache-cli framework provides a complete set of tools that lets developers connect to a private network hosted locally and consequently to have multiple identities with predefined Ether.

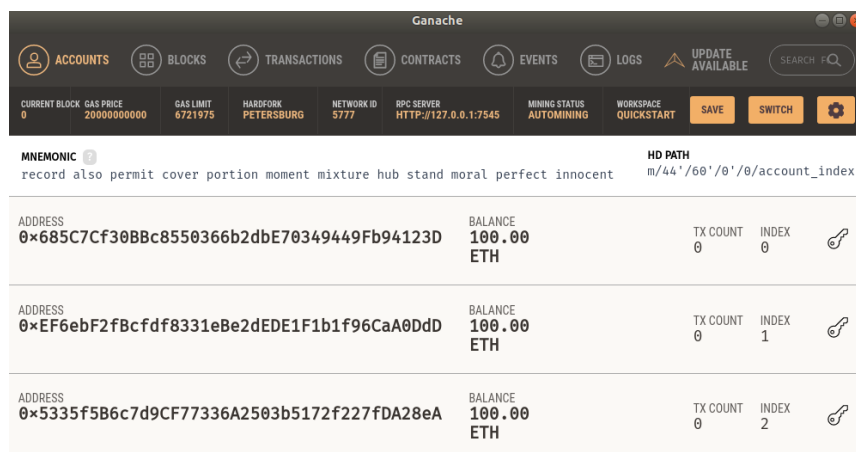


Figure 5.1: Ganache framework.

Figure 5.1 shows a project created on ganache framework that, by default, generates a private network on <http://127.0.0.1:7545> and also offers several accounts with Ether previously deposited to ease dApps testing. Private test blockchain networks are able to simulate an entire network environment, although as they local hosted, they are no exposed to congested transaction-flows. Because of that, it is not accurate to simulate an overflow of throughput in order to study the behaviour of the network. However, after setting up an Ethereum local network, truffle framework offers a set of tools to compile and deploy smart contracts to a predefined network. Truffle projects are generally structured as snippet 5.10.

```
truffle project
├── build
├── contracts
├── test
└── truffle-config.js
```

Snippet 5.10: Truffle project directory tree.

Contracts definitions are held in the *contracts* directory while *truffle-config.js* is where is configured all the data to establish a node connection together with the necessary data to compile contracts. To compile voucher's definition set throughout Section 4.2, it is mandatory to specify the solidity compiler

version, together with the Ethereum network to where to deploy the contract. Hence, one of the possible configurations would look like snippet 5.11

```
...
networks: {
  local: {
    port: 7545,           // Custom port
    network_id: 5777,    // Custom network
    gas: 8500000,        // Gas sent with each transaction (default: ~6700000)
    gasPrice: 20000000000, // 20 gwei (in wei) (default: 100 gwei)
    from: 0x685C7Cf30BBc8550366b2dbE70349449Fb94123D, // Account to send txs from
  },

  compilers: {
    solc: {
      version: "0.5.11" // Fetch exact version from solc-bin (default: truffle's version)
    }
  }
}
...
```

Snippet 5.11: Example configuration for truffle-config.js.

As contracts are written in solidity, all files containing contract definitions have a file extension of *.sol*. Running `$ truffle compile` from the project's root directory triggers solidity compiler to compile every contract inside the contracts directory. Doing so, as explained in Section Smart Contracts development and deployment, the compiler generates a *.json* file for each compiled contract containing its correspondent ABI and bytecode. This outcome is stored inside the build directory and can then be used to deploy and interact with each contract.

After having our voucher contract compiled, it is time to deploy it to the previously set local private network. This process can be achieved by running `$ truffle migrate` from the project's root directory. Truffle fetches every *.json* file inside build directory and deploy each on the previously set network on the truffle-config.js file. Figure 5.2 shows a successful issuance of a voucher contract.

```

2_initial_Voucher.js
=====
Replacing 'Voucher'
-----
> transaction hash: 0xca439ef32b7b838809cd3f3d9effaba1dd14f073d8ac48fa9a244037284855bf
> Blocks: 0
> contract address: 0x3607E64C8aFbE497Ddde2E82c357E22121A2427F
> block number: 3
> block timestamp: 1571000116
> account: 0x685C7Cf308Bc8550366b2dbE70349449Fb94123D
> balance: 99.97754029
> gas used: 817221
> gas price: 20 gwei
> value sent: 0.00000001 ETH
> total cost: 0.01634443 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.01634443 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.01634443 ETH

```

Figure 5.2: Successful voucher issuance.

As Figure 5.2 demonstrates, the voucher contract instance was deployed to the address `0x3607E64C8aFbE497Ddde2E82c357E22121A2427F` and can now be accessed publicly. Thus, it is now possible to develop different unit tests to endorse our contract's functionalities. This can be achieved by instantiating the voucher at address `0x3607E64C8aFbE497Ddde2E82c357E22121A2427F` and calling its respective functionalities. Snippet 5.12 shows two of the several implemented tests, asserting that there is indeed a voucher contract at the set address and later asserting its value is bigger than zero.

```

const Voucher = artifacts.require(0x3607E64C8aFbE497Ddde2E82c357E22121A2427F);

contract("Voucher", () => {
  it("Voucher issued", () => {
    Voucher.deployed().then(v => { assert(v) })
  })
  it("Voucher issued with value bigger than 0", () => {
    Voucher.deployed()
      .then(instance => instance.voucherValue.call())
      .then(balance => { assert.notEqual(
        balance.valueOf(), 0, "Voucher must be initialized with a value");
      })
  })
});
...

```

Snippet 5.12: Two examples of voucher's functionalities unit tests.

After developing other relevant tests, it is possible to run `$ truffle test`. This command uses the

mocha framework to execute different describe-it blocks having the result exposed in Figure 5.3.

```
Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Contract: Voucher
✓ Voucher issued
✓ Voucher issued with value bigger than 0
✓ Voucher is acquired by client
✓ Voucher can't be acquired by the issuer
✓ Voucher is expired
✓ voucher is redeemed
✓ Voucher can't redeemed by lowner
✓ Voucher cannot be issued with 0 Eth
✓ Voucher returns Eth to company issuer after expiration
✓ Voucher state set to expired after acquisition out of date
```

Figure 5.3: Different unit tests to analyze voucher's contract.

5.2 Demonstration

Having the voucher's functionalities tested on the previous Section, it is time to deploy a voucher factory to one of the Ethereum test networks and test its functionalities against different identities. Just like on the previous Section, to deploy a contract to the Ethereum test network, a connection to it must be set. The difference between this deployment and the one executed in the previous Section is that this time, it is necessary to set up a local or remote Ethereum node from an existing real test network. For the sake of the solution demonstration, infura provides an excellent service by offering free remote node connection to all three test networks described in Section Meta-mask and Test nets plus the main network. Although the connection is free, there is an API call limit of 100 000 daily requests, which for the sake of testing, suits perfectly. Moreover, it is essential to notice that there are other different ways to accomplish this process, like using Remix. Although, this is a rather straightforward approach that achieves the same goal. Figure 5.4 shows an example of a created endpoint towards the Ethereum Kovan test network and Figure 5.5 illustrates a component diagram that shows how both components are connected.



Figure 5.4: Infura connection to kovan remote node.

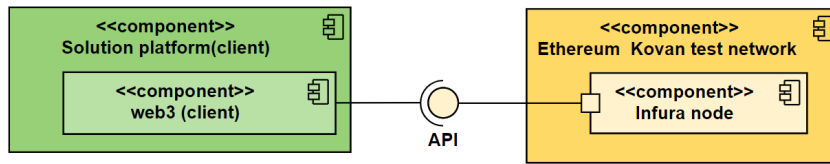


Figure 5.5: Ethereum network and solution platform connection.

Having an endpoint to a remote node from Kovan test network, it is now possible to use web3 library to connect to it and later on execute Remote Procedure Call (RPC)s, just as snippet 5.13 demonstrates.

```

from web3 import Web3

infura_url = 'http://kovan.infura.io/v3/3e3326511666419484f6225ae39a541d'
web3 = Web3(Web3.HTTPProvider(infura_url))
  
```

Snippet 5.13: Web3 connection to kovan test network.

As a consequence, web3 can now be instantiated to deploy the factory using its compiled outcome. Truffle can help to do the deployment by changing its configuration to the current web3 connection. After migrating a voucher factory, the resulted address is `0x7FFD3DEE8860b22a992936d8905455F7279c481D`, which can also be verified in <https://kovan.etherscan.io/>, a blockchain explorer to the Kovan network. After having this factory deployed, Remix becomes handy to execute a couple of manual tests to issue, acquire and redeem the voucher from different accounts. The next sequence of images demonstrate a thriving flow of a voucher state transition from an issuing company, to a client and finally to a receiving company. Figure 5.6 shows the remix tool, where our deployed voucher factory is shown in as 2 of Figure 5.6. The factory functionalities are listed as 3, where blue functions represent calls, orange represents transactions and red show payable functions. Finally, the factory functionalities are called from the account shown as 1, which is correspondent to the Metamask current account.

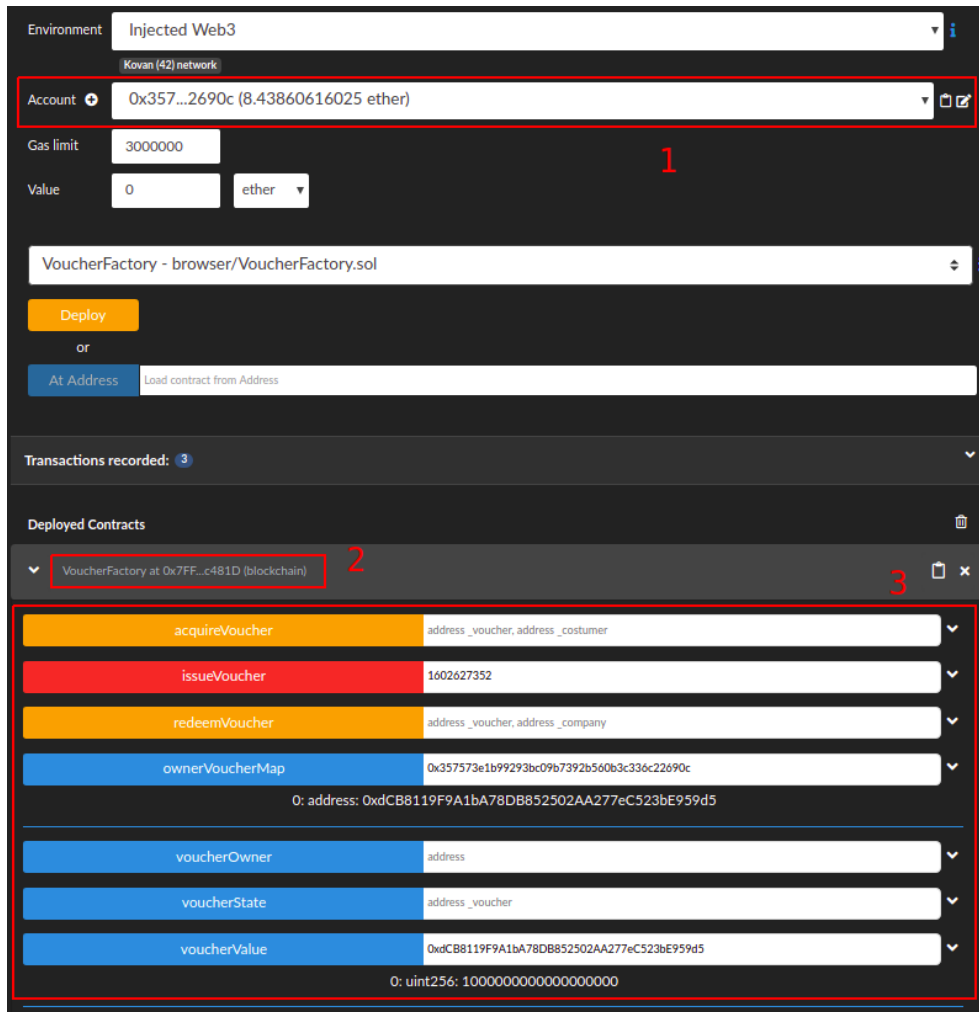


Figure 5.6: Voucher issuance demonstration.

Figure 5.6 shows a voucher issued by 0x357...2690c with a value of 1 Ether to the address 0xdCB8119F9A1bA78DB852502AA277eC523bE959d5. This means another account representing a client can acquire this voucher. After executing the acquire functionality from a different account, note that the owner of the voucher changed as well as its state, just like Figure 5.7 demonstrates. Important to refer that states are converted to integers, having 0 - Free; 1 - Acquired; 2 - Expired; 3 - Redeemed.

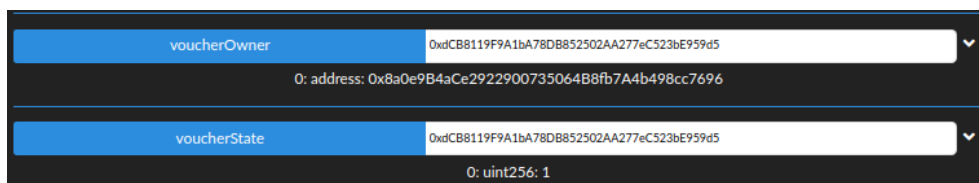


Figure 5.7: Voucher acquired.

Now lets assume that `0xec5259F5840C6be0528Dd177ae8A65Fa3a528a01` is a receiving company and has 0 Ether on its balance. When the client owner of the voucher redeems it to the receiving company, its balance increase by the voucher's value, as it is confirmed by Figure 5.8, and all of the vouchers properties are set according to its state transition. In other words, as Figure 5.9 proves, the voucher instance no longer has an owner, nor value and its state have been set to Redeemed.

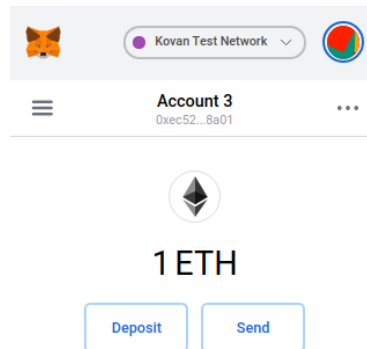


Figure 5.8: Current balance of `0xec5259F5840C6be0528Dd177ae8A65Fa3a528a01`.

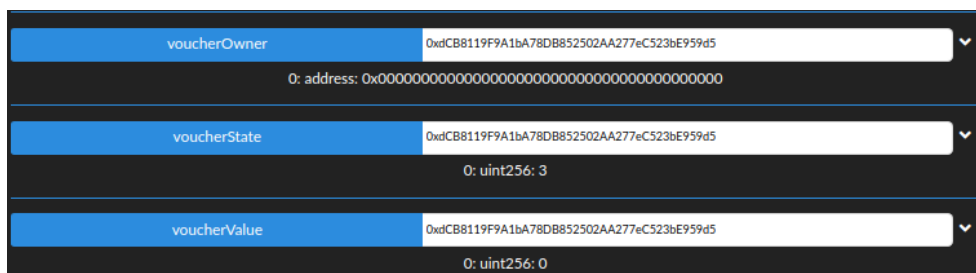


Figure 5.9: Last state for voucher's properties.

5.3 Requirements validation

Regarding requirements analysis, the solution implementation, together with the choice of technologies, have made it accessible to meet almost every requirement defined previously. Taking into consideration Section 3.7, decentralization has been fully achieved due to the use of Ethereum public blockchain. Ethereum allows this solution to run and be maintained by its nodes, inheriting data immutability and replication from Ethereum blockchain's nature. Audit traceability is achieved through the voucher factory, as it offers any entity to read vouchers data on a solution level openly. Moreover, to trace and review any user activity, like asset transactions, blockchain explorers like etherscan dispose of several APIs to filter user transactions. The voucher factory also eases the consumption of vouchers interfaces by abstracting developers from the voucher's source code functionalities. In other words, interoperability becomes a reality,

as developers can integrate this solution by only using the voucher factory ABI together with its address. The use of these two variables, together with some Ether allows any authentic pair of keys to be able to act as an issuing company. In addition, when deploying the voucher factory to the Ethereum network, this contract gets replicated through every node in the network. Thus, in case one of the nodes fail, the rest of the network nodes keep track of incoming or out-coming data, contributing to a fault-tolerant solution. Those requirements are successfully achieved due to the practices applied among vouchers definition and their factory.

On the other hand, some requirements could not be accomplished entirely. The solution was successfully tested both locally and on a test network, although, it was not possible to test it against a real scenario (main network) where the network transaction flux can be overloaded. Working locally and on Kovan test network, allowed the solution to process transactions rather instantly, ignoring the fact that, on the Ethereum main network, transactions could potentially take minutes to occur. Additionally, scalability still remains one of the most prominent downside aspects of blockchain technology in general. The solution does not present a scalable behaviour, because, with the increase of users and vouchers, the solution's access point (voucher factory) remains the same. The increase of CRUD operations requests may lead to network congestion, resulting in slower responses and more expensive transactions. The reason to not test this solution over the Ethereum main network is because it involves significant monetary costs. Nevertheless, Figure 5.10 is one of the test examples that proves the lack of scalability of the solution. The test makes use of Kovan test network and scripts to deploy 1 voucher and later on deploy 100, using single threaded processes in the same environment.

```
PS C:\Users\Tester\Desktop\tese\implementation> truffle test --network kovan
Using network 'kovan'.

Compiling your contracts...
=====
> Compiling .\contracts\Voucher.sol

Contract: Voucher
  ✓ Deploy 1 voucher (532ms)
  ✓ Deploy 100 vouchers (6521ms)
```

Figure 5.10: Performance test for scalability.

Assume a linear approach where x axis represent the number of vouchers deployed and y axis represent the time to conclude the deployment process. According to Figure 5.10, it is possible to map two points $p_1 \rightarrow (1,532)$ and $p_2 \rightarrow (100,6521)$. These dots connected generate an equation of type $y = mx + b$, with a positive m . This means that, even though, the solution may process a large amount of requests, it can

end up taking several seconds, or even minutes, to deploy a large amount of vouchers. The performance relationship would be $\approx(6521/532):100$, meaning it would be $\approx 0,12$ times slower for every incremented voucher to deploy. Nevertheless, this approach is only reliable if the Ethereum congestion is ignored, which suits for this use case, as the solution was not tested on the Ethereum main network. The reason for taking a linear approach is because, even if testing the solution on a test network, a single average computer is not capable of generating enough requests so that the test network is completely congested.

Last but not least, transactions and smart-contract write, update and delete operations need validation through consensus algorithms and, consequently, these need time to process. As stated in Section 2.1.2, changing the ledger state demands transaction time and this fact results on the delay between the operation execution and completeness. In other words, when executing transaction functions, data is not immediately available as it is firstly subject of validation. Apart from those operations, read operations remain instant.

5.4 Problems addressed

After some analysis on the final solution together with the collected difficulties during the development, it is conclusive that it solves every problem stated in Section 3.1. Although, it comes with great costs, bringing other problems that must be addressed before the solution is able to be in production and, therefore, live on the Ethereum main network. Before detailing the new problems, it is important to explain how the previous problems addressed in Section 3.1 are solved with the dissertation work.

- **P.1** is solved by stakeholders nonce when executing transaction functions. More detailed, Ethereum has two references to nonce[15], proof of work nonce, which is explained in Section 2.1.1 and account nonce. An account nonce is simply a transaction counter in each account, as displayed in snippet 4.8. Taking this snippet as an example, the account redeeming the voucher has to provide its nonce to execute the function and broadcast it to the network. This mechanism prevents the account from redeeming the voucher more than once, as the next transaction function can not have the same nonce as the previous one. Another detailed example would be a situation where A redeems a voucher, broadcasting this transaction function to the network. Once the transaction is sent to be mined, A can try to redeem the same voucher paying a higher gas price, which would lead to a faster mineration. Doing so, the first voucher redemption would be mined later than the second, by getting a higher priority in the pending transaction queue. Since Ethereum considers nonce, this is not allowed as the nonce of the later transaction is higher than the previous one.
- **P.2** is solved by the way each stakeholder is represented. As explained in 4.3, stakeholders are entirely identified by their account address, indexing its privKey and pubKey, which leads to a mapping of *wallet address* \Rightarrow *voucher address*. Therefore, having the perfect scenario where many

different companies would integrate this solution on their payment processors, clients would be identified the same way through all the different companies. In other words, clients would not need different mobile applications nor cards for each different company, as they would be identified by their wallet address among the different companies.

- **P.3** is tackled by the factory development in Section 4.4. The whole purpose of the voucher factory contract is to keep track of every deployed voucher activity. The factory holds tracking mechanisms that able full traceability of voucher's life-cycle, i.e. from the moment of a voucher's issuance to its redemption or expiration, the voucher factory keeps it as an auditable record. This factory is a valuable resource of this solution due to its immutability as a smart contract, which means its source code can not be modified after being deployed to the Ethereum blockchain. In addition, every state transition of the solution would be freely auditable using etherscan, helping to keep track of voucher's data, value and state.
- **P.4** is solved by Section 4.3.1. During this Section it is explained an internal way of authentication. Every transaction function executed on voucher's functionalities go through modifiers process, guaranteeing to the smart contract who really is executing the function. From an ownership point of view, it is simple for an owner to prove its identity when signing the transaction function with its privKey. Using modifiers mechanism prevents non-owners to execute voucher's functions, as they can not provide a privKey that they are not aware of.
- **P.5** is fixed when linking the vouchers identification do its own Ethereum address. Since every Ethereum address is unique and generated during deployment, this makes a perfect association between its authenticity and identity. Concluding, the authenticity of all vouchers rely entirely on the Ethereum network, as it is responsible to keep the ledger of record. In other words, whenever the Ethereum blockchain states a voucher of address A with X Ether, it means it is authentic to the network and therefore exists on the blockchain state.

Needless to say that this solution is entirely backed by the Ethereum public blockchain. An eventual attack on this network could and would potential have its impact mirrored on our solution. 51% attacks are a real possible threat to every PoW-based blockchain, as they would dictate the state of the system, leading to an eventual Ether drain from the solution's vouchers. Although, this is a general blockchain problem that is not directly related nor dependent on the solution presented in this document.

That being stated, it is now presented the problems produced by the solution, starting by the natural singular issuance of a voucher. So far, it has been presented a solution that allows an entity to issue a voucher by sending Ether to it together with an expiration date. This procedure creates a single smart contract that consequently represents a single voucher. As explained in Section 4.2.3, the issuance of a

voucher requires the payment for necessary gas to execute the constructor of a voucher instance. As a result, this issuance methods becomes exceptionally inconvenient to an issuing company in case it would prefer to issue multiple vouchers. Issuing a voucher requires time for the transaction to be mined (as shown in Figure 5.10), which therefore leads to an aggressive raise of waiting time to issue numerous vouchers.

Another issue of Ethereum smart contracts relies on its inability to trigger itself automatically based on specific rules. In other words, by nature, a smart contract is unable to update its state automatically, for example, based on time. For our solution this would be important to achieve, as a voucher instance would require to be set to expired according to its validation date, i.e. after comparing the validation date to the current date, the smart contract should be able to update its state. Unfortunately, smart contracts execute based on human or service interaction, falling under a "must call to execute" paradigm. Even though this functionality is not within smart contract's nature, the Ethereum community already started developing alternatives to this paradigm, called Oracles. Oracles are third-party information sources that have the sole function of supplying data to the blockchain. They can be used to break the paradigm, allowing the solution to trigger events when necessary. However, Oracles do not make part of the blockchain consensus mechanism, therefore need to be trusted. In other words, stakeholders would not have guarantees that the source of information coming from Oracles would be legit, whether it comes from a website or a sensor. Besides, contract's functionalities require the use of Ether to be triggered, which would lead to additional costs. For instance, Ethereum alarm clock¹, an oracle that allows scheduling transactions to be executed at a later time. Concluding, Oracles could potentially break the "must call to execute" paradigm, even though this approach would bring new trust and cost issues.

With significant decentralization also comes numerous challenges and difficulties, such as version control. As explained throughout the document, once a smart contract is deployed to a blockchain, its source code is forever immutable. While this feature may be seen as a general application benefit, it is also a big issue when it comes to features development. Considering an initial smart contract deployed on the blockchain, any change to its source code would require another smart contract deployment. This process is extremely inconvenient as developers would have to deploy different smart contracts to the blockchain regularly. For the solution case, assuming the voucher's source code would have to suffer any change, this would require a new instance of a voucher factory because as explained in 4.4, it would contain an outdated version of the voucher's source code.

Decentralization also brings another issue regarding stakeholders' privacy. When using this solution platform, as intended, every stakeholder operation is susceptible to be audited, as well as their Ether balances. All parameters involved in this solution are openly exposed, such as Ether spent on vouchers issuance, Ether returned from expired vouchers, and Ether redeemed using vouchers. While overall de-

¹<https://www.ethereum-alarm-clock.com/>

centralization can offer transparency to those who have no data to protect, it may also expose data from those who do not wish to. In other words, what for some stakeholders is a significant advantage, for others can be a big downside aspect.

Another problem disclosed by our solution is that, even though there is a canonical differentiation between the two types of companies, issuing and receiving companies, there is not yet a fully reliable mechanism that would prevent a company to always act as a receiving company. This issue means that a company is allowed to take advantage of this solution by only receiving vouchers' value, attracting customers attention to its business and never issuing any voucher in return to the network. To avoid this behaviour, a receiving company should be limited to its activity as an issuing company. In other words, there should be a mechanism that keeps the balance between issuing and redeeming activity, preventing less beneficial behaviours for all companies participating in this network.

Last but not least, even though this dissertation focus does not cover economic aspects to achieve a balanced economic ecosystem, it is inevitable that the solution is running over Ethereum, leading to the use of Ether. Referenced in Section Ethereum, Ether holds value against Fiat currencies, and transactions need Ether to be confirmed. Billing price deductions are entirely backed by the average worldwide market price of Ether. That said, assuming an environment where gas price raises along with network congestion and Ether gets cheaper, clients may end up spending more Ether value to redeem their vouchers rather than what the voucher is worth. Taking an unrealistic, yet possible scenario where a voucher is worth 0.01 Ether, the worst case would be the need of a value higher than 0.01 Ether to execute the transaction. This would require more value to redeem the voucher than the value it represents. This problem is unavoidable because we must use the Ethereum platform to operate this solution.

Despite the stated issues that need to be addressed and tackled, the solution proposed by this dissertation fixes many modern daily problems, therefore adding a value proposition as a proof of concept to standard loyalty programs.

CHAPTER 6

Conclusion

This dissertation aimed to develop a decentralized application capable of acting as a loyalty program for its user. Thus, the objectives were achieved, making use of the Ethereum platform, as the research revealed that a public blockchain would be the most suitable technology to fulfil all requirements.

Following a severe and extensive analysis of existing solutions, both based or not in DLT, as well as different types of blockchain technology, it is conclusive that true decentralization would offer various parties the trust they do not hold between each other. During the development, it was unquestionable that creating a dApp brings new types of concerns that a common centralized application would not bear, for instance keeping vouchers value within the loyalty program. Having a fully decentralized application means there can be externally executed functionalities, opening worldwide access to a system that is intended to act fully opened but also closed enough to keep its value.

During the execution of this dissertation we strongly enriched our knowledge regarding blockchain technology, as well as about the way this technology is shaping marketing and management practices around the concept of loyalty programs and rewarding systems. Many of the challenges encountered during this study were not technical in nature, although they could not easily be separated from the technology point of view, due to integration decisions. In other words, shaping small functionalities could fundamentally change the whole behaviour of the solution. Blockchain is a maturing technology, and its

complexity and immutability make it even more important to consider long-term effects of architecture decisions.

Other approaches were taken into consideration, such as the use of permissioned blockchain technologies. To test some of the functionalities, it was developed a prototype based on the same requirements and fundamentals as the current solution presented. According to the study of Chapter 2, it was used HIF and HIC to achieve the same requirements stated for the solution presented during this document and, therefore achieve an improvement of the same exposed problems. Regardless, the study for the use of Hyperledger turned out not to be adequate, as it could not achieve full decentralization by needing a central authority to provide access to different users. The use of Hyperledger would require the solution to be accessed solely by invitation or acceptance, which comes as a great inconvenient from a client point of view. Besides, auditability would have to be restricted to its participants, not offering an open ledger to, at least, read operations. This type of approach would undoubtedly offer more exclusivity to its users, although that was never the focused target of the solution.

6.1 Future work

Although every problem was addressed and adequately fixed, as stated throughout Chapter 5, the presented solution still demonstrates certain functionalities that may be improved and others that can be integrated. Being said, for future work, it is proposed the challenge of creating a mechanism to avoid single-issued vouchers, offering issuing companies the ability to issue several vouchers aggregated to any sort of campaign. Every value-added service to this solution does not require to be based on a decentralized platform, although it should, which means this mechanism could eventually be another smart contract. Also, to bring full automation to the voucher's state transition, as future work, it should be designed an external service that would map every voucher instance to its expiration date, consequently triggering the voucher to be set to expired.

Despite the versioning issue discussed in Chapter 5 involves a substantial amount of effort to fix, it can be achieved. To do so, every feature developed over the contracts source code should implement a mapping mechanism. This map should contain the address of the previously deployed version of the factory, creating a chain link of deployed voucher factory versions, offering APIs the possibility to consume every data from the first deployed factory to the last. Also, it is suggested a study for a staking mechanism that would create a balance between issuing companies and receiving companies — guaranteeing that no company would be allowed to consequently act as a receiving company without ever giving any benefit to the network.

Last but not least, as Figure 6.1 demonstrates, client-based applications, such as electronic commerce

or any other type of commerce that intends to make use of the solution, may implement or modify their APIs to interact with the contracts functionalities.

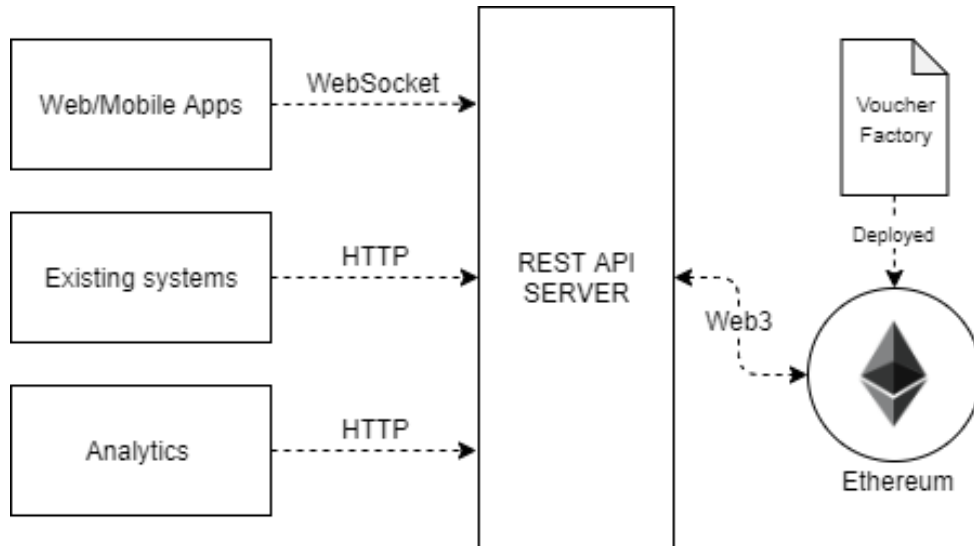


Figure 6.1: Integration on other services.

Finally, for testing purposes, any client based application may reference the voucher factory address at `0x7FFD3DEE8860b22a992936d8905455F7279c481D` with the respective proofed ABI shown in kovan's etherscan.

Appendix A: Voucher factory

```
pragma solidity ^0.5.11;

/// @title VoucherFactory - Contract that manages vouchers
/// @author João Quintanilha - <jpquintanilha@ua.pt>

import "./Voucher.sol";

contract VoucherFactory {

    /*
     * Storage
     */
    mapping(address => address[]) public ownerVoucherMap; /// owner => voucher[]
    mapping(address => address) public voucherOwner; /// voucher => owner
    uint constant EXPIRED_STATE = 2;
```

Snippet 7.14: Part 1 of voucher factory contract.

```

/*
 * Events
 */
event newVoucher(address v);
event voucherAcquired(address v, address costumer);
event voucherExpired(address v);
event voucherRedeemed(address v);

/*
 * Modifiers
 */
modifier isVoucherOwner(address _caller, address _voucher){
    require(voucherOwner[_voucher] == _caller, "Caller is not the owner of the voucher!");
    _;
}
/*
 * Public functions
 */
/// @dev function to issue a voucher contract and keep track of its mapping issuer and owner
/// @param _validity - timestamp holding datetime for voucher's validity
function issueVoucher(uint _validity) public payable{
    require(msg.value > 0, "Ether sent to voucher must be bigger than 0!");
    Voucher v = (new Voucher).value(msg.value)(_validity, msg.sender);
    ownerVoucherMap[v.owner()].push(address(v));
    voucherOwner[address(v)] = v.owner();

    emit newVoucher(address(v));
}
/// @dev function that allows voucher acquisition
/// @param _voucher - Voucher contract address
/// @param _costumer - address of the costumer that will acquire the voucher
function acquireVoucher(address _voucher, address _costumer) public
isVoucherOwner(msg.sender, _voucher) {
    Voucher(_voucher).acquire(_costumer, msg.sender);
}

```

Snippet 7.15: Part 2 of voucher factory contract.

```

if(uint(Voucher(_voucher).state()) == EXPIRED_STATE){
    delete voucherOwner[_voucher];
    for (uint i = 0; i < ownerVoucherMap[_costumer].length; i++) {
        if(ownerVoucherMap[_costumer][i] == _voucher){
            delete ownerVoucherMap[_costumer][i];
            break;
        }
    }
    emit voucherExpired(_voucher);
}else{
    ownerVoucherMap[_costumer].push(_voucher);
    voucherOwner[_voucher] = Voucher(_voucher).owner();
    emit voucherAcquired(_voucher, _costumer);}
}

/// @dev function that allows voucher redemption
/// @param _voucher - Voucher contract address
/// @param _company - adress of the costumer that will aqcquire the voucher
function redeemVoucher(address _voucher, address payable _company) public
isVoucherOwner(msg.sender, _voucher) {
    Voucher(_voucher).redeem(_company, msg.sender);

    if(uint(Voucher(_voucher).state()) == EXPIRED_STATE){
        delete voucherOwner[_voucher];
        for (uint i = 0; i < ownerVoucherMap[_company].length; i++) {
            if(ownerVoucherMap[_company][i] == _voucher){
                delete ownerVoucherMap[_company][i];
                break;
            }
        }
    }
    emit voucherExpired(_voucher);
}

```

Snippet 7.16: Part 3 of voucher factory contract.

```

else{
    delete voucherOwner[_voucher];
    for (uint i = 0; i < ownerVoucherMap[_company].length; i++) {
        if(ownerVoucherMap[_company][i] == _voucher){
            delete ownerVoucherMap[_company][i];
            break;
        }
    }
    emit voucherRedeemed(_voucher);
}

}

/// @dev function that retrieves voucher value
/// @param _voucher - Voucher contract address
function voucherValue(address _voucher) public view returns(uint){
    return address(_voucher).balance;
}

/// @dev function that retrieves voucher state
/// @param _voucher - Voucher contract address
function voucherState(address _voucher) public view returns(uint){
    return uint(Voucher(_voucher).state());
}
}

```

Snippet 7.17: Part 4 of voucher factory contract.

Bibliography

- [1] Russell Lacey and Julie Z. Sneath. Customer loyalty programs: Are they fair to consumers? *Journal of Consumer Marketing*, pages 458–464, 2006.
- [2] Laurens de Rooij and Zsoka Koczan. Frequent flyer programs. *Encyclopedia of Economics and Society*, January 2015.
- [3] Ning Feng, Ming Li Zhang, and Sai Li Tang. Impact of loyalty programs on relationship benefits and customer loyalty: A customer perspective. *2010 International Conference on Management Science and Engineering (ICMSE)*, pages 533–538, 2010.
- [4] Advait Deshpande, Katherine Stewart, Louise Lepetit, and Salil Gunashekar. Distributed Ledger Technologies/Blockchain: Challenges, opportunities and the prospects for standards. *British Standards Institution (BSI)*, pages 1–34, May 2017.
- [5] Mauro Isaja and John Soldatos. Distributed ledger technology for decentralization of manufacturing processes. *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, pages 696–701, 2018.
- [6] Federico Matteo Benčić and Ivana Podnar Žarko. Distributed Ledger Technology: Blockchain Compared to Directed Acyclic Graph. *IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, January 2018.
- [7] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". <https://bitcoin.org/bitcoin.pdf>. Accessed: July 1, 2018.

- [8] Hari Krishnan, Sai Saketh, and Venkata Tej. Cryptocurrency Mining – Transition to Cloud. *International Journal of Advanced Computer Science and Applications*, 6(9):115–124, 2015.
- [9] Dejan Vuji, Dijana Jagodi, and Siniša Ran. Blockchain Technology , Bitcoin , and Ethereum : A Brief Overview. *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 21–23, March 2018.
- [10] Harry Halpin and Marta Piekarska. Introduction to security and privacy on the blockchain. *Proceedings - 2nd IEEE European Symposium on Security and Privacy Workshops, EuroS and PW 2017*, pages 1–3, 2017.
- [11] "IBM Blockchain Platform Technical Overview". (2018, March 30). Retrieved from <http://www.smallake.kr/wp-content/uploads/2018/06/ibm-blockchain-platform-technical-overview.pdf>.
- [12] Tatsuya Sato and Yosuke Himura. Smart-Contract Based System Operations for Permissioned Blockchain. *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–6, 2018.
- [13] "Blockchain in DevOps: Implementing transparentcontinuous delivery". (n.d.). Retrieved from [http://www.ey.com/Publication/vwLUAssetsPI/EY-Blockchain-in-DevOps/\\$FILE/EY-Blockchain-in-DevOps.pdf](http://www.ey.com/Publication/vwLUAssetsPI/EY-Blockchain-in-DevOps/$FILE/EY-Blockchain-in-DevOps.pdf).
- [14] "Hyperledger-Fabricdocs Documentation Release Master". (2019, June 2019). Retrived from <https://media.readthedocs.org/pdf/hyperledger-fabric/release-1.2/hyperledger-fabric.pdf>.
- [15] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, pages 1–32, 2014.
- [16] Nutthakorn Chalaemwongwan. State of the Art and Challenges Facing Consensus Protocols on Blockchain. pages 957–962, 2018.
- [17] Du Mingxiao, Ma Xiaofeng, Zhang Zhe, Wang Xiangwei, and Chen Qijun. A review on consensus algorithm of blockchain. *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, 2017-Janua:2567–2572, 2017.
- [18] Joshua a Kroll, Ian C Davey, and Edward W Felten. The Economics of Bitcoin Mining, or Bitcoin in the Presence of Adversaries. *The Twelfth Workshop on the Economics of Information Security (WEIS 2013)*, (Weis):1–21, 2013.
- [19] Danny Bradbury. The problem with Bitcoin. *Computer Fraud & Security*, pages 5–8, November 2013.

- [20] Pavel Vasin. BlackCoin’s Proof-of-Stake Protocol v2 Pavel. Retrieved from <https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf>. page 2. Accessed: February 11, 2019.
- [21] A.M. Antonopoulos and G. Wood. *Mastering Ethereum: Building Smart Contracts and DApps*. O’Reilly Media, Incorporated, 2018.
- [22] Pavel Innokentievich Tutubalin and Vladimir Vasilevich Mokshin. The Evaluation of the cryptographic strength of asymmetric encryption algorithms. *RPC 2017 - Proceedings of the 2nd Russian-Pacific Conference on Computer Technology and Applications*, 2017-Decem:180–183, 2017.
- [23] Financial Analysis. Factors Influencing Cryptocurrency Prices: Evidence from Bitcoin, Ethereum, Dash, Litecoin, and Monero. *Journal of Economics and Financial Analysis*, 2(2):1–27, 2018.
- [24] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.
- [25] Vitalik Buterin. A next-generation smart contract and decentralized application platform. Retrieved from http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf. pages 1–36. Accessed: January 15, 2019.
- [26] Nida Khan, Abdelkader Lahmadi, Jerome Francois, and Radu State. Towards a Management Plane for Smart Contracts : Ethereum Case Study. *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2018.
- [27] Santiago Bragagnolo, Henrique Rocha, Marcus Denker, and Stephane Ducasse. SmartInspect: solidity smart contract inspector. *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 9–18, 2018.
- [28] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on Ethereum smart contracts (SoK). *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 164–186, July 2017.
- [29] ”Hyperledger Architecture , Volume 1”. Retrieved from https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf. page 15. Accessed: July 11, 2018.
- [30] Hyperledger composer documentation. <https://hyperledger.github.io/composer/v0.19/index.html>. Accessed: July 10, 2018.
- [31] Byron Sharp and Anne Sharp. Loyalty programs and their impact on repeat purchase loyalty patterns. *International Journal of Research in Marketing*, pages 473–486, December 1997.

- [32] Felipe González, Yihan Yu, Andrea Figueroa, Claudia López, and Cecilia Aragon. Global reactions to the Cambridge analytica scandal: A cross-language social media study. *The Web Conference 2019 - Companion of the World Wide Web Conference*, 2:799–806, 2019.
- [33] Sima Ghaleb Magatef Elham Fakhri Tomalieh. The Impact of Customer Loyalty Programs on Customer Retention. *International Journal of Business and Social Science*, 6(81):78–93, 2015.
- [34] McIlroy Andrea. Building customer relationships: do discount cards work? *Managing Service Quality: An International Journal*, 10(6):347–355, January 2000.
- [35] Samuel Brack, Stefan Dietzel, and Bjorn Scheuermann. ANONUS: Anonymous Bonus Point System with Fraud Detection. *Proceedings - Conference on Local Computer Networks, LCN*, 2017-Octob:356–364, 2017.
- [36] "White Paper v1.2". (2018, June). Retrieved from <https://rouge.network/whitepaper-rge.pdf>. Accessed: February 10, 2019.
- [37] "CyberMiles : Empowering the Decentralization of Online Marketplaces". (2017, September). Retrieved from <https://www.bitcv.com/storage/pdf/whitePaper/aatjufPPe1EIj1Bx5aTvjPMiRfMCbSLodsiSoG8Q.pdf>. Accessed: March 29, 2019.
- [38] Patrick Pascheka and Michael During. Advanced cooperative decentralized decision making using a cooperative reward system. *INISTA 2015 - 2015 International Symposium on Innovations in Intelligent Systems and Applications, Proceedings*, 2015.
- [39] Morgen Peck. Blockchain world - do you need a blockchain? *IEEE Spectrum*, 54:38–60, 10 2017.