FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

An Ethereum-based point of sale system for retail stores

Maria Betânia Benita Afonso

FOR JURY EVALUATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Mário Amorim Lopes

External Supervisor: Dinis Freitas

July 26, 2021

Resumo

Blockchain, uma tecnologia de registro distribuído concebida para a utilização em criptomoedas, tem vindo a ganhar reconhecimento nos últimos anos e a demonstrar que tem potencial em várias áreas, incluindo a automatização do sistema de pagamentos de retalho, o que consequentemente melhora o funcionamento do sector bancário e impulsiona o crescimento económico [1].

O objectivo desta dissertação é analisar a plataforma Ethereum, enquanto método de pagamento, e explorar a sua aplicação em sistemas de pontos de venda (POS) para futura incorporação em lojas de retalho. Baseado em *Blockchain* e na tecnologia de *smart contract*, foi feita a análise e implementada uma DApp para determinar as características subjacentes à Ethereum, que permitirá uma maior adopção por parte dos retalhistas.

O desenvolvimento da dissertação centrou-se em dois temas principais: o primeiro relacionou-se com os sistemas POS, com o objectivo de proporcionar uma introdução aos seus processos e características de funcionamento; o segundo centrou-se em *Blockchain* e Ethereum, de forma a ter uma melhor compreensão de como esta tecnologia inovadora funciona e interage.

Esta abordagem é validada pelo desenvolvimento de um sistema de *smart contract* através de uma aplicação descentralizada que estabelece a comunicação entre um sistema POS que recolhe os dados correspondentes à compra e à *blockchain*. Além disso, o estudo aqui apresentado destaca as vantagens destas tecnologias quando aplicadas aos pagamentos e verifica a viabilidade desta abordagem, acabando por fornecer uma visão das capacidades, características e limites que um sistema desta natureza pode possuir.

Palavras-chave: Blockchain, Ethereum, Point of sale, Pagamentos, Smart Contract

Abstract

Blockchain, a decentralized ledger technology (DLT) designed for use in cryptocurrencies, has gained attention in recent years and showing evidence that it has potential in several areas including automating the retail payment system which consequently improves the functioning of the banking sector and boosts economic growth [1].

The purpose of this dissertation is to analyze Ethereum, as a payment method, and to explore its applicability in point of sale (POS) systems for future adoption in retail stores. Based on Blockchain and smart contract technology, we analyze and implement a DApp to determine the underlying characteristics of Ethereum that will allow for greater adoption or rejection by retailers.

The study focused on two major themes. The first was related to POS systems, for the purpose to provided an introduction to their functioning processes and features. The second was focused on Blockchain and Ethereum, in a way to have a better understanding of how this cutting-edge technology functions and interacts.

This approach is validated by the development of a smart contract system via a decentralized application that bridges the communication between a POS system that collects the data corresponding to the purchase and a blockchain network. The study provided here highlights the advantages of these technologies when applied to payments and verifies the viability of this approach, eventually providing insight into the capabilities, characteristics, and limits that such a system may have.

Keywords: Blockchain, Ethereum, Point of sale, Payments, Smart Contract

Acknowledgments

The completion of a master's degree dissertation is without a doubt a lengthy road filled with commitment, obstacles, pleasures, sorrows, questions, and uncertainties, but it is only possible with the direct or indirect support and encouragement of a few individuals to whom I am eternally grateful.

To my supervisor, Professor Mário Amorim Lopes for their continued support and availability in carrying out this subject, for their comments and criticisms that shaped the study, and for their full participation in addressing questions and uncertainties that arose along the way.

To all the people in the whole Retail Consult team, who were always available to assist and guide me in whatever was necessary. In particular to Tomás Sotomaior and Pedro Ferreira, who were the ones I interacted with the most and who helped me directly in the development of the project I was able to accomplish.

A sincere vote of gratitude and appreciation to my parents and sister, Paula Afonso, José Fernandes, and Mariana Afonso, for the education instilled in me through the years that have shaped me into the person I am today, and for all the support they have given me during these years.

To my boyfriend, Afonso Brito, who has always been there for me during this huge and unknown challenge, encouraging me through the more tough times.

In general, I would like to express my gratitude to everyone who helped me in any way in the completion of this project.

Maria Betânia Afonso

Contents

List of Figures

Abbreviations and Symbols

B2B	Business-to-business
CRM	Customer relationship management
DApp	Decentralized Application
DLT	Decentralized Ledger Technology
IT	Information Technology
PG	Payment Gateway
POS	Point of Sale

Chapter 1

Introduction

1.1 Context and Motivation

In a highly competitive and fast-paced business like retail, there is a clear need to define and implement strategies to meet market needs in an optimal way, in order to reduce time and inherent costs. Moreover, with the continuous growth of online stores and mobile applications, the omnichannel concept has become really important today. The possibility to allow customers to buy through any channel and have goods or services available for collection or delivery can result in an important differentiator and increased profit.

Besides that, the payment methods available to the customers also present a big advantage in their competitiveness and is an element that has evolved greatly over the past few years. Most of the purchases made by individuals are done via conventional means of payment, as cash and credit card payment. However, today's industry is witnessing a rise of alternative payment methods like electronic payments through electronic wallets or online payment solutions. Those new payment solutions are faster, more mobile and more flexible to make purchases.

In this sense, and with the need to keep up with new technologies and to resolve the challenges faced by omnichannel retailers, a new payment method emerged, connecting two of today's big trends: blockchain and cryptocurrencies.

Blockchain is a distributed ledger, a shared database, where multiple partners can access the immutable information and registers that exist in the database. Besides that, blockchain is the technology that enables the existence of cryptocurrency, which represents the beginning of a new phase of technology-driven markets that have the potential to disrupt conventional market strategies, since in few years, cryptocurrency, and Bitcoin, in particular, has proven its worth, now boasting 14 million Bitcoins in circulation.

This technology promises to facilitate fast, secure, low-cost international payment processing services (and other transactions) through the use of encrypted distributed ledgers that provide trusted real-time verification of transactions without the need for intermediaries. This way, consumers can access a global payment system anywhere, anytime, in which participation is restricted

only by access to technology, rather than by factors such as having a credit history or a bank account.

1.2 Objectives

As mentioned in the previous section, the use of cryptocurrencies and their use on retail has evolved a lot throughout these last few years. Thus, the tendency for a progressive increase in the demand for methods of using these currencies is to be expected. As such, the main objective of this dissertation is to study and implement a strategy capable of making payments in Point of Sale (POS) systems using cryptocurrencies to keep up with the demand of the public. In this proof of concept, Ethereum technology was used, which employs the Ether cryptocurrency.

There are a variety of features and processes inherent in a transaction between a merchant and a consumer, so this thesis will try to find out which technologies and requirements are necessary to make the payment possible through a POS system available in a business. In order to propose an implementation strategy that can successfully target the goal, it is mainly necessary to research the POS systems available in the market and the blockchain payment methods currently used, in a way to be able to link these two topics.

1.3 Methodologies

This work project will be led by a theoretical and practical approach to Blockchain technology and its implementation in POS systems with Ethereum payments.

The first step was to research work already developed in this area, gathering some results and conclusions. The research was based on 2 main themes: POS systems and blockchain. Within each of them, several issues were studied that are related to their features and applicability as is the case of cryptocurrencies. In this way, the search included seeking in different databases, articles, papers, websites related to the issues to be discussed. With this reading, some research became indispensable to understand some background concepts and some bases to understand the connection between cryptocurrency payments and POS systems.

The next phase was to discuss possible implementation ideas and research the technology needed to make them a reality so that a plan for the dissertation's outcome could be created.

After this step, following several trials and market studies, a final proposal for the dissertation's objective was discussed and implemented.

Chapter 2

Background

2.1 POS System

A point of sale system, or POS, is the place where the customers make a payment of products or services at a store. In today's competitive era, it has become crucial for everyone to embed the POS solution in the businesses, allowing them to implement the payment services, inventory process, and customer services. This way, POS system is not a independent machine or process, but a constellation of things that together enable retailers to process customer-facing transactions efficiently and streamline business processes connected with these transactions.

2.1.1 Key Components

Every POS system is a conjunction of software and hardware components that makes the everyday tasks of a business simpler and quicker [3].

Hardware

The first hardware component of a POS system is the **cash register**, where, as the name suggests transaction details are registered. It can be a touchscreen PC monitor or mobile device with a POS app.

The next component is the **Cash drawer** which is used to store the daily takings and cash float along with cheques, vouchers, receipts, and slips relevant to accounting.

The third component is the **Receipt printer**, which is used to print receipts for customers for the payments done by them.

The next component is the **Barcode Scanner**. A barcode scanner is typically used in retail environments to scan the unique barcode on the products and automatically update product count based on the items sold.

Another important hardware component is the **Credit card reader** which allows the customers to securely pay by credit card in-store, whether that is through a contactless payment like Apple Pay, a chip card, or a magstripe (magnetic stripe) card.

Software

The POS Software helps a retailer to manage his business and to control all transactions [3]. All modern POS systems have a frontend interface for the point of sale and a backend side for behind-the-scenes analytics and management functions. The transactions proceded by the staff or other person are made by the frontend interface, normally on a touchscreen monitor or a tablet, while the backend is accessed separately in a browser or an application window either on the same device or a separate computer.

Regardless of the type of POS software [3], the frontend interface and backend are always connected and synced. Therefore, there are two ways by which data can be stored in a POS system, it can either be onsite or cloud-based.

An onSite POS, also known as “legacy” or “traditional” POS, stores data on local servers and runs on a closed internal network, making it impossible to access out of the network. On the other hand, a cloud-based POS software is hosted online, i.e. data is stored on the POS provider’s internet servers, enabling the retailer to access it from any computer browser. One of the key advantages of cloud-based systems is that they tend to provide more options to integrate with other software programs.

2.1.2 Key Features

The point of sales system emerged from the year’s old hand till machine and cash registers. The functions of the cash registers, however, were limited to storing cash and few others if it was a recent model. However, today’s POS system goes beyond processing sales, offering a multitude of features to help the retailers to increase efficiency and provide insights to help them to improve their productivity.

So, let’s look at the important backend functions that a POS system performs [4]:

1. **Payment processing**

   Payment processing is one of the core functions of a POS system. Each time a customer buys an item, your POS system processes the transaction. There are several different payment types a POS system might accept:

   - Cash;
   - Chip cards, which are credit cards with an embedded chip;
   - Magstripe credit cards, which are read by swiping past a magnetic reading head;
   - Contactless payments, which might include a contactless card that customers tap, a mobile wallet (e.g., Google Pay or Apple Pay) or a QR code (Quick Response code);
   - Card-not-present transaction, which happens when a customer and their credit card are not actually present and the information need to be manually entered by the employee. This also occurs when a customer enters their payment details while checking out online.

2. **Inventory and stock management**

   One of the main functions of a POS system, inventory management, is used to keep track of all products in a business. This way is possible to know when it is time to order/or not order specific products and provide complete visibility and accountability of the store level at all time. The system should provide a view of lot-wise inventory, SKU transaction history about in, out, and within movements of any SKU, and inbound and outbound inventory.

   It can also exist an automated inventory software that can connect with the business sales data and be able to make stock adjustments, increasing or decreasing stock while viewing inventory and capture the reason for stock adjustments.

3. **Sales monitoring and reporting**

   A detailed analysis on sales outcomes must be generated by the point of sale system. It should be able to provide winnings and outgoings on an hourly, daily, weekly, monthly, and annual basis so that merchants can easily assess their overall success.

   Seasonal product demand, forecasting based on sales patterns, insights into needless product overspend, and stock management information are just a few of the capabilities of a robust POS reporting module.

4. **Cross channel returns management**

   A POS returns management module's features include accepting cross-channel returns and allowing refunds and replacements from any retail location. The system should make it simple to create numerous returns for the same sales order at various times. It should include information such as the reason for the return, the salesperson's name, and any comments.

5. **Customer relationship management (CRM)**

   By preserving customer data and purchasing history, a CRM solution may assist the store in attracting consumers. This may be utilized to give customers a more personalized experience and assist businesses in customizing their communications, marketing, and customer care.

6. **Employee management**

   A POS system's employee management module may assist in managing staffing numbers, employee hours, and sales performance, as well as tracking employee productivity. As a result, it will be able to track employee activity by connecting it to each transaction, allowing managers to distinguish between successful and unsuccessful performers and take necessary steps to enhance their productivity.

7. **Order Management**

   Accepting client orders from all sales channels, storing the requested products for pickup by the customer, and even sending the goods to the customer's home or another shop so that

the customer may pick up the goods at the chosen location are all part of the everyday job of store operations.In this way, modern POS systems may accept orders from many channels, such as e-commerce, and provide various customer journeys, including home delivery, pickup, shipping to the shop, or shipping for pickup.

2.1.3 Payment Process

A POS transaction is simply any transaction that occurs within a business, whether it is the sale of a food item, retail product, or service. A point-of-sale transaction occurs when money is exchanged for one product or service. However, the process inherent in selling and paying for a product is much more complex than the perspective that the customer experiences. As a business owner or backend viewer [5], it is important and beneficial to understand the payment process starting with the primary players since the moment a customer's card has been swiped, keyed into a terminal, or entered into a web page, up to the moment the funds are transferred into your bank account.

In this way, before we go any further into the payment process, we must first identify the main players:

- **Merchant**: a company or individual who sells product or service to customers;
- **Customer**: Also called a cardholder, who wants to access the products or services that the merchant is selling, and initiates the transaction;
- **Issuer**: The entity or bank that grants credit to a customer and issues a credit card. The Issuer typically partners with Visa or Mastercard. These institutions also issue payment to the merchant bank (also called the acquiring bank) on behalf of its customers, the buyers in any given transaction.
- **Acquirer:** Also known as the acquiring bank, the Acquirer is the financial institution that maintains the merchant's bank account (known as the merchant's account). The acquiring bank passes the merchant's transactions to the issuing bank to receive payment
- **Card Network:** Also called Card Brands, this network facilitates the entire process of payment authorization and settlement, acting as an intermediary that connects the merchant with the financial institution that issued the card to process and approve credit card transactions. Visa and Mastercard work as intermediaries and they work to maintain the connections between Acquirers and issuers.

In addition to the main players mentioned in the payment process, there are other participants who provide a lot of "extras" and others features to merchants [6]. Theoretically, a merchant might be able to accept electronic payments without these additional organizations by communicating directly with Acquirers. In practice, however, given the complexity of the payment processing schemes and the enormous amount of different payment cards and methods, such implementation would be almost impossible without the involvement of payment processors and gateways.

- **Payment Processor:** Is the company that handles the credit card and debit card transactions for a business, acting as a mediator between the bank and the merchant. They also maintain the merchant accounts where merchants actually receive their money paid by the cardholders for their goods or services. Payment processors' pricing structures and fees vary by the amount and value of the transactions the merchant process and the model they choose. Generally, payment processors charge a percentage of each transaction, often adding a small per-transaction fee as well, and a few other fees, such as a monthly statement fee and a monthly minimum fee. If the business wants to accept credit card and debit card payments from the customer online, over the phone, or at the point of sale, it is necessary to partner with a payment processor.

- **Payment Gateway:** A merchant's POS payment system communicates directly to the payment processor. Sometimes, however, in-between the merchant and payment processor there is another intermediate called Payment Gateway (or Payment Switch).Most businesses that sell goods or services online should integrate a payment gateway to make the purchasing process easier for the customers. It is a technology that creates a secure connection between the merchant business's website or browser and the credit card processing company. This secure connection is used to encrypt credit card payment data for every transaction, verifying the authenticity of a transaction and keeping sensitive information secure.

After showing the players included in the payment process, we can now see how a payment and its authorization are made. From the entire payment system perspective, there are two main stages of payment processing [6]: authorization and settlement.

Authorization

Authorization is the very first stage in the payment process. It is required to verify the cardholder's credit or, in the case of a debit card, to ensure that the cardholder's bank account has sufficient money to complete the transaction. The authorization flow is shown in Figure 2.1.

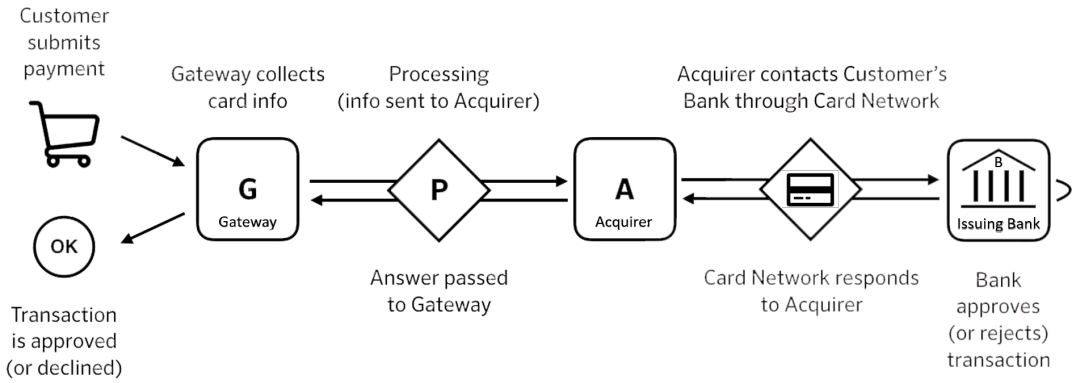


Figure 2.1: Authorization flow

After the customer has chosen a payment option and provided the merchant with the relevant information, the payment transaction proceeds to the next "station," which is either a payment gateway or a payment processor. The transaction data is forwarded to the appropriate Acquirer, who subsequently communicates with the issuer via the card network to get permission.The issuer is the one that keeps track of the cardholder's account information and checks to see if the customer has enough credit to cover the whole payment transaction amount. During the authorization process, the answer of the Issuing Bank can be one of the following: Approval, Decline, or Referral (basically a request for additional information). Regardless of the Issuing's response, the response will be sent all the way back to the POS, where it will show a suitable message along with the process's result.

From an information security perspective, the approval stage is critical since it requires transferring all accessible verification information from the POS throughout the whole system to the Acquirer.

Settlement

When the authorization is received and the transaction is completed by the POS system, the payment must be settled, which means the payment must be reconciled between the merchant, its acquirer, and the issuer, i.e. the actual transfer of money from the customer's bank account to the merchant's bank account must take place.

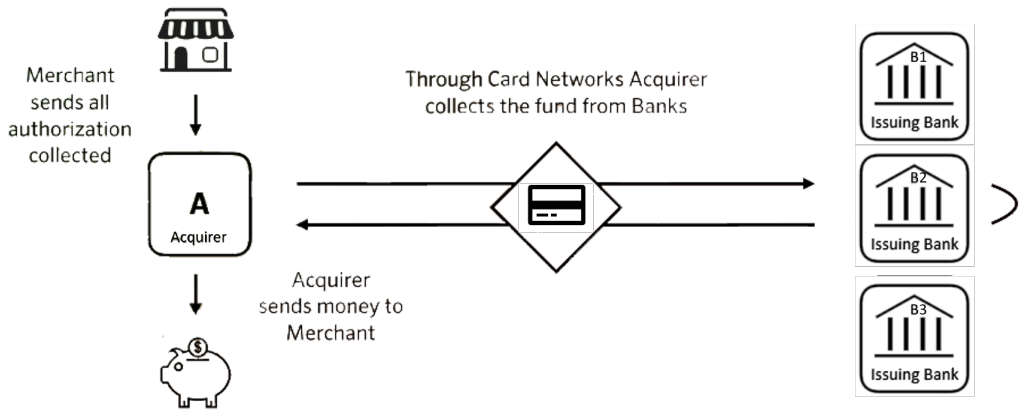


Figure 2.2: Settlement flow

In this stage, the merchant (or more precisely, its payment system) collects all the authorizations received for the various transactions over the period and sends them as a batch to the processor which forwards it to the Acquirer. Next, the Acquirer or the processor credits the merchant's account and sends the data through the Card Network to the issuer, who posts the transaction to the cardholder's account.

It is only during this Settlement process that the merchant receives the money and that the customer's bank account is debited.

2.2 Blockchain Technology

After gaining a better knowledge of what is included in a POS system, we can focus on Blockchain technology and learn more about its properties and potential in the payment process.

2.2.1 Blockchain Features

Since the first implementation of the Blockchain technology, with the Bitcoin 2008, by Nakamoto, the interest in developing this innovation has been growing spontaneously. Actually, Blockchain becomes one of the most interesting technologies in the world which witnessed explosive growth during the past few years [7].

Blockchain can be described as a set of blocks that are linked to each other creating a chain. Each block is identified by a hash, generated using the SHA256 cryptographic hash algorithm [8] on the header of the block and each one of them references to a previous block, known as the parent block, through the "previous block hash" field in the block header. These concepts are illustrated in Figure 2.3.

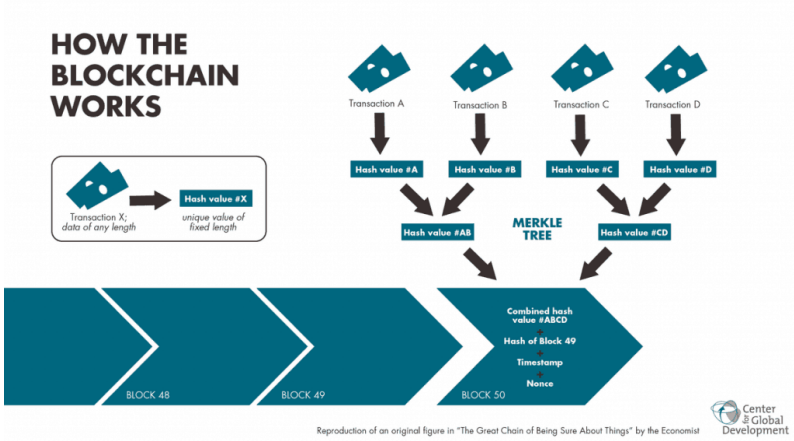


Figure 2.3: Representation of a blockchain's structure

The blockchain is a peer-to-peer network that holds a decentralized encrypted database [9]. Is a network that continuously saves and updates a chronological chain of blocks, where each block saves the information relating to it and the previous ones. Besides that, the collection of blocks is available for all participants to see and check its validity, so at any given time, can see all the history and origin of past transactions. Because of this, there is no need for a central authority to control and secure its use and it´s a network recognized as transparent.

Since each block is connected to the one before it, when a transaction is put into the database and the information is modified, the records cannot be changed. If one block is altered, the hash on the next block will no longer match the hashes of the other blocks. If we modified something on the first block, we would have to modify the hash on the subsequent blocks to match. Consequently, the hash on the subsequent block would have to be changed to match, and so on. This means that the blockchain is immutable [9] since no one block can be changed without affecting the others.

The information stored in the Blockchain is protected and validated by a mechanism called consensus [9]. Consists of a group decision-making process in which group members develop and agree to reach a decision and maintain the consistency of shared data. Each type of mechanism is what defines how a transaction can be added to the chain itself and aim at ensuring that all participants dispose of identical copies of the distributed database files. Most blockchain projects use one of the three currently most common consensus algorithms are Proof of Work (PoW) and Proof of Stake (PoS). Each type of consensus mechanism has its pros and cons regarding its use. For example, the mechanism used by Bitcoin [10], "Proof of Work", if on the one hand it is known to be a very difficult mechanism to find a solution, but extremely easy to verify that an answer solves the problem, on the other hand, it is a mechanism that requires a lot of energy and computer energy to reach a consensus.

2.2.2 Ethereum Smart Contract

Smart contracts are self-executing computer programs based on blockchain technology that perform functions automatically in response to a triggering event [11]. These contracts are only available in digital form and can involve two or more parties. Smart contracts can not change their properties once they have been deployed since the blockchain infrastructure does not allow it. This means that the conditions of any such contract are linear and will be carried out automatically once has been agreed upon [12]. Nick Szabo, an American computer and legal scientist, is credited with developing the theoretical framework for smart contracts. In a white paper published in 1994, he first mentioned the concept, which is defined as follows:

"A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries."

According to Szabo [13], such transactions might be as simple as buy/sell transactions or can involve more detailed instructions. Smart contracts are similar to regular contracts in that they involve an agreement between two or more intermediaries to do or not do something in exchange for something else [12]. Unlike traditional contracts, which need each party to trust the other to fulfill their duties, smart contracts do not require this level of confidence between participants [14]. This is due to the fact that any smart contract's program code will be performed automatically and without deliberation.

All smart contracts are essentially built on the "if-this-then-that" principle, which states that if specific criteria are met, the computer will respond automatically to that trigger [11]. For total operability, Smart Contract technology relies on clear, unambiguous logic as well complete and accurate information for complete operability. So because the program is always performed exactly as it was programmed in advance, there is no space for interpretation. After the programming step, since smart contracts are recorded and ultimately processed on the blockchain, human intervention is no longer essential, if not impossible [12].

A frequent example used to illustrate this concept is a simple vending machine [13]. A vending machine, like a program code, behaves algorithmically, which means that the same set of instructions will be followed in all scenarios in the same way. The item is delivered as soon as money is deposited into the machine and an item is selected. As long as the machine is in good functioning order, it has neither the ability nor the capability of breaking the contract's terms. Likewise, a smart contract's binary design binds it to run the predetermined code. As a result, rather than current legislation, the program's code determines the rights and duties flowing from the contract.

2.2.3 The Cost of Ethereum Blockchains

Smart contracts have nearly endless potential, limited only by one factor: computational power, which is the only resource used to maintain a blockchain. This resource can become extraordinarily expensive on Ethereum blockchains, owing to efficiency difficulties with the Ethereum blockchain's overall design and EVM execution methodology [15].

Each transaction or smart contract on the Ethereum blockchain requires the consumption of gas, i.e. each line of code in Solidity requires a certain amount of gas to execute, whereby each transaction requires at least 21,000 gas [15] but varies depending on the current demand for gas and the size and speed of the contract being executed. This gas is used by the contract to reward the miners who secure that transaction on the network.

The transaction must be paid with a sufficient amount of gas, so if the amount of gas is too low, the transaction will not take place since miners will not be adequately compensated and will abandon the job. Developers may ensure that their smart contracts function properly over the network by using gas low and high limitations. This is accomplished by providing adequate remuneration to the miners while avoiding the loss of the extra gas they pay for the transaction, which would otherwise be wasted to the miners after the transaction is completed.

Gas in itself is an extremely important innovation in the blockchain community.In its implementation as a Proof of Work blockchain system, it not only enhances the effectiveness of Proof of Work, but also produces a more fair and transparent mining environment.

2.2.4 Blockchain in Payment Sector

Blockchain can change the way that people and associations interact, the way that organizations team up with each other, the transparency of processes and data, and, eventually, the productivity and sustainability of our economy. Therefore, as this technology evolves and new use-cases emerge, Blockchain technology has been used in a variety of sectors such as banking, financial services, insurance, government, healthcare, retail and automotive.

As the retail industry is being driven by data, Merchants are looking to focus more on customized retailing in order to grow their consumer level and enhance customer service. Blockchain technology will serve as an enabler, allowing retailers to accomplish their objectives more quickly [16].

Just like the blockchain, cryptocurrencies have become a keyword to refer to a wide range of technological developments that use a technique better known as cryptography. In this sense, cryptocurrency is a digital currency that is electronic-only and does not have a physical form.

Cryptocurrencies are produced, tracked, and managed through blockchain, and that´s why this technology combines a distributed ledger that may establish confidence among participants with quicker, low-cost, secure payment services [17].

Although blockchain originated as a support platform for digital currencies [17], it has now been integrated into a wide range of industries, including payments, and so it is possible to see a growing number of businesses embracing this new era.

2.2.4.1 Issues of Payment Sector

In retail sector, the blockchain offers a lot of promise to help merchants in improving their current business operations, resulting in more income, despite the reality that many companies are apprehensive about blockchain or any distributed technology in the payments industry. However, in order to see the benefits that this implementation can bring to the businesses, it is essential to first look at the issues that are related to the payments sector [17]:

- **Payment Frauds and Chargebacks:** At the present, the rise in money theft and fraud is extremely worrying. Indeed, as the e-commerce industry grows, the possibility of unlawful use of customer information or card thefts is becoming more common. Furthermore, chargebacks are expensive and can harm a company's image. The issue is that chargebacks frequently do not occur for a valid cause. As a result, fraudulent actions build-up, making it harder for a firm to survive in the long run.

- **Delayed cross-border transactions:** Cross-border payments can be slow, inefficient, and costly, although they are vital to global commerce. Even if the merchant does not conduct the transactions through a banking channel, it might take up to 6 days to complete. In fact, this might have a negative impact on the corporate sector, making the company process inefficient.

- **Low card data security:** In order to take any debit or credit cards, all merchants must be certified by the Payment Card Industry Data Security Standards. However, merchants frequently fail to meet all of the requirements of this certification, and they frequently dismiss the fact that card data security and the protection of their customers' personal information should be given top importance.

- **Lack of multi-currency and payment methods:** International or global e-commerce needs to accept a variety of currencies and payment methods. Electronic payments, such as e-wallets, mobile payments, and credit/debit cards, allows online retailers to compete in foreign markets by allowing customers to pay in their own currency. Multi-currency cross-border transactions might need new bank accounts, corporate entities, and legal obstacles in

each national market for merchants, and as a result, some firms refuse to support particular local currencies, limiting a user's ability to buy from anywhere.

- **High Processing Fees:** Processing fees rise year after year, and the variances in processing fees might be extremely complicated in some situations. As a result, keeping track of the processing fees and how much they will cost in the long term becomes difficult.

2.2.4.2 Benefits of Blockchain Payments

Now that the payment industry's problems have been exposed, it is time to consider how blockchain may benefit it. Far from being a flawless technology, it can yet provide a number of benefits to retailers:

Less intermediaries

With the current payments system, to make a payment, it must go through many intermediaries and authorizations, such as the payment gateway, exchange mode, issuer, and so on. These intermediaries maintain track of who sent money to and under what conditions, ensuring the integrity of the value transaction [18]. Moreover, each of the middlemen charges a fee for their services, which prolongs the transaction and reduces its security and reliability.

Payment systems based on the blockchain do not require intermediaries due to the blockchain's intrinsic transparency, which may be regarded as a source of "truth" [18]. As a result, the buyer and seller can conduct a direct cash transfer, or peer-to-peer payment, and the system will accurately and authentically preserve the transaction data.

Transparency and Security

As previously said, one of the most major advantages of blockchain technology is the high level of transparency it provides. As a result, blockchain provides a more secure and irreversible platform for transaction operations, opening up new options for both consumers and businesses. Since every block is chronologically connected since it contains its hash as well as the preceding block's hash. Hence, no one can manipulate the blockchain records since any changes will be evident. As a result, blockchain payment systems provide high levels of security, safety, and transparency to ensure that payments are genuine and secure, while also protecting the network from both external and inter-network threats [19].

High-speed Payments

Within a country, cash transactions are quick, requiring only one or two minutes. For the private sector (purchases, payments, transfers), and enterprise, this speed is more than enough. Cross-Border payments, on the other hand, which occur when the payee and the payer live in separate

countries, can encounter a number of obstacles, taking anywhere from one to five days to execute [18].

The fact that money travels via many middlemen slows down international transactions. However, as we saw in the first point, blockchain payments can eliminate the need for middlemen in the payment process, reducing payment processing time from days to hours [19].

Lower Processing Fees

Because the blockchain-based payment system minimizes the number of intermediaries, the number of commissions and fees paid by participants may be subtracted from the transaction cost. Additionally, the transition to the blockchain lowers transaction costs, which lowers transaction costs even further [18].

If a cryptocurrency is used for money transfers, the cost of the transaction will be even lower, not to mention the cost of converting cryptocurrency to fiat, which can range from 0% to 10% depending on the fiat currency, cryptocurrency, and exchanger chosen, resulting in a slight increase in the overall cost. However, as the market for digital assets develops, the size of these expenses will decrease.

Aside from that, the blockchain fee is a cryptocurrency transaction cost that consumers are paid when they conduct crypto transactions. The charge is collected in order to complete the transaction on the network and paying the blockchain fee is required for cryptocurrency transfers to arrive on time. The blockchain fee is one of the most common methods for speeding up crypto transactions, which are frequently slowed by the blockchain network's high congestion. The lower your transaction's priority on the blockchain network, the cheaper the blockchain cost.

Automation with Smart Contracts

Smart contracts enable automation, which is a huge benefit for business owners and executives. The seller can profit from Smart Contracts in a number of ways, including reducing payment time, assisting in the implementation of immediate payments, and automating payment flows [19]. Smart contracts also provide unbreakable security due to their decentralized nature. Unlike in the real world, where contracts are sanctioned by central parties such as banks and governments, users retain complete control of their assets and no institution may manipulate their smart contracts.

2.2.4.3 Blockchain Payments Process

With blockchain and thousands of cryptocurrencies available on the market, using this technology to purchase goods or services in a retail store is not as simple as with more traditional means of payment. For this reason, only a few methods are accepted by retailers as a form of payment.

There are two major differences between traditional payment processing and cryptocurrency payment processing [20]. First, a client pays via a digital wallet, not a credit card, as usual and secondly, he/she pays in cryptocurrency, not in physical currencies like USD, AUD, etc.

In the same way, digital wallets are digital versions of traditional wallets that hold debit, credit, cash, and other cards or coupons. It is an online service that allows clients to collect or keep money in the same way that money is held in a bank account. A user must first create an account with a digital wallet service provider, following which funds are sent to the digital wallet account through debit, credit, or online bank transaction [21]. Crypto wallets are divided into two categories:

- Single currency wallets: These wallets link to a specific currency, like Bitcoin. They fit best for individuals or inventors who work with a particular currency.
- Multi-currency wallets: Business owners prefer multi-currency crypto wallets as they offer a more comprehensive range of payment options and allows them to store variables types of cryptocurrencies.

From a software point of view, a cryptocurrency payment through the blockchain follows the following scheme:

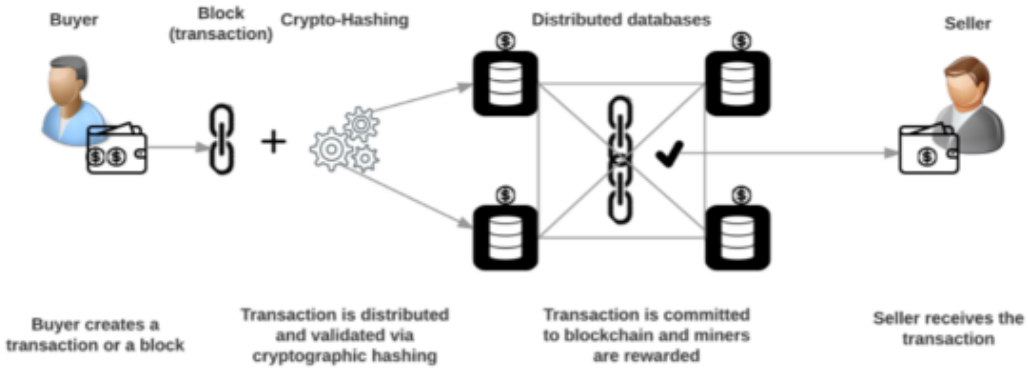


Figure 2.4: Blockchain payment flow

Initially, when a transfer takes place [22], a set of data from the players is required: keys of the sender and the receiver or their wallet address and the amount of money to be transferred in the transaction. Any new record or transaction within the blockchain implies the building of a new block. The transferred keys are then encrypted with a hash function, rendering the content unchangeable, and a copy of this new node is distributed to all other nodes in the blockchain system. After that, each node verifies the block and ensures that the data it contains is accurate. If everything checks out, the block is uploaded to each node's own blockchain. Although minors require time to ratify the purchase, it is considered nearly automatic. The transaction is irrevocable and cannot be falsified after it is verified, and the seller receives confirmation of the transaction.

This technology demonstrates that with digital wallets is possible to enable transactions to be initiated by a mobile device at POS, online or in-app, where the payment can be deducted automatically through the crypto wallets of the customer when they place an order and transferred to the wallet of the business owner.

Although most cryptocurrency payments are made online, some merchants like to offer a face-to-face cryptocurrency payment platform. Scanning a QR code displayed on the merchant's POS system is the most frequent way of in-store transactions [23]. The consumer scans a unique code on the screen of the POS terminals that are integrated with the merchant's public key using the QR reader on their mobile crypto wallet of choice. The address field of the merchant's public key (and the amount if it is provided in the QR code) will be supplied after scanning is complete. The consumer then provides the information to produce the payment, which is subsequently broadcasted via the mobile wallet. Online payments function similarly, the customer just scans a QR code on the screen to transmit funds for a transaction.

Alternatively, NFC-capable POS machines that take payments via mobile devices are being developed. If the payment provider permits it, the retailer can change the payment into domestic currency immediately to avoid exposure to market volatility or add it to their cryptocurrency balance. A payment confirmation is then issued instantly. Unlike traditional card payment acceptance, using these methods there is no intermediary entity involved.

Figure 2.5 represents a draft of a diagram that describes the message flow for creating and processing a transaction in a POS system. In this process are involved four main players:

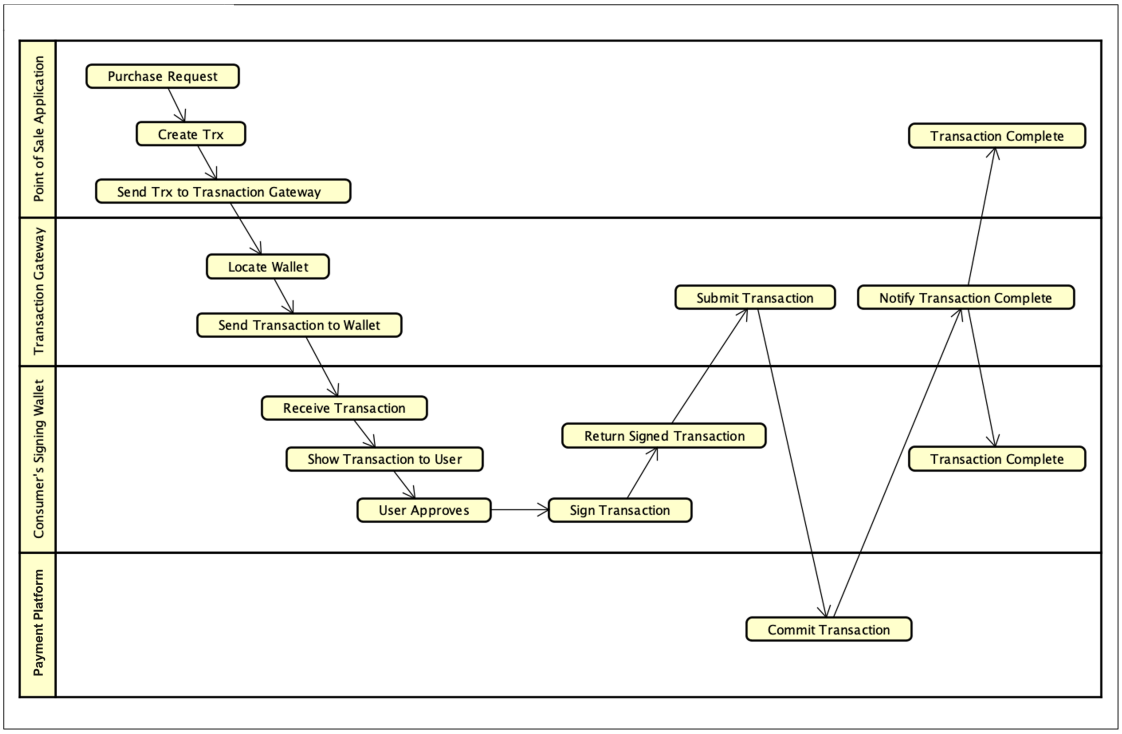


Figure 2.5: POS activity diagram

1. **The point of sale application:** For each purchase, the point-of-sale application creates a transaction and submits it to the transaction Gateway, which manages the signing process and notifies the point-of-sale application when the transaction has been successfully committed to the blockchain.

2. **Transaction gateway:** The Transaction Gateway coordinates the interaction between the business application and the payment platform. The point-of-sale app sends unsigned transactions to the Transaction Gateway, which then forwards the transactions to the appropriate wallets for signing. After signing the transaction, the wallet sends the signed transaction back to the Transaction Gateway, which commits it to the payment platform. The Transaction Gateway notifies the point-of-sale application when the payment is complete.

3. **Signing wallet:** The Signing Wallet is a client application used by consumers and to sign transactions received from the Transaction Gateway. The Signing Wallet can be a mobile app, web app, or desktop application. When the Signing Wallet receives a transaction, it notifies the user and displays the details to the user, including the receipt stored off-chain, and then prompts the user to either accept or reject the transaction. If the user agrees, then the wallet will sign the transaction and return the signed transaction to the Transaction Gateway.

4. **Payment Platform:** Provides a fast and efficient payment intermediate station. It features support for assets allowing support for local currencies and all the processes included in the payment process.

2.3 Successful Applications

Cryptocurrencies have been around for more than a decade, but it was not until the price of bitcoin reached nearly $20,000 in 2017 that they were widely recognized. We can envision a near future in which cryptocurrencies will achieve widespread acceptance if we follow the links between the dematerialization of payments and the development of cryptocurrencies. Cryptocurrencies have been around for more than a decade, but it was not until the price of bitcoin reached nearly $20,000 in 2017 that they were widely recognized. We can see a near future in which cryptocurrencies will achieve widespread acceptance if we follow the links between the dematerialization of payments and the development of cryptocurrencies [2].

The present adoption rates are subject to change, and will most likely do so. Currently, several big corporations and enterprises, such as the Chinese government, Google, Amazon, Facebook, and Apple, have successfully overcome some of the challenges to cryptocurrencies, allowing them to become progressively prevalent. This will speed up their acceptance and allow them to eventually replace cash.

The Figure 2.6 illustrates the outcome of this evolution, comparing the pace of adoption of blockchain wallets to the rate of adoption of the Internet. After correcting for scale, the curves are equivalent for the time being. By 2030, there might be 200 million blockchain wallet users if present trends continue.

Indeed, throughout the study on this state of the art, it became obvious that papers, active developer groups, and a few innovative companies are the most up-to-date and finest sources of information on the progress of this technology.

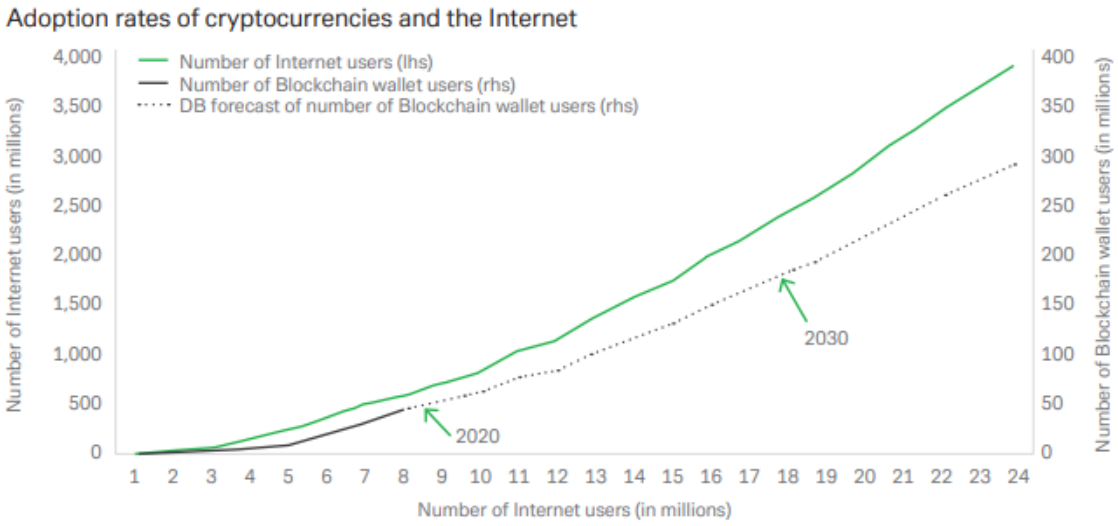


Figure 2.6: Graph of adoption rates of cryptocurrencies and the Internet. From [2]

Starting with the existing solutions for cryptocurrency payments on the market, we can state that certain businesses have already begun to accept them. Microsoft, one of the world's top software firms, has welcomed Bitcoin for use in their online Xbox Store since 2014 [24]. Due to the volatility, they temporarily stopped taking it, but are now accepting it for Xbox store credits exclusively.

Another example is Overstock, a major retailer that accepts multiple forms of cryptocurrencies as payment [25]. Customers may pay using Ethereum, Litecoin, Dash, and Bitcoin Cash, among other cryptocurrencies. Besides that, due to ShapeShift [25], a service used by this company that allows users to swap crypto assets, Overstock consumers may not only make purchases with cryptocurrencies, but also convert digital currencies between different coin types. This means that if an Overstock customer owns Ethereum coins, they will be able to convert them to bitcoin during the checkout process utilizing ShapeShift's technology.

Like these two examples, many other companies like Etsy, Starbucks, IKEA, Tesla have bet on this new technology to change their business model, whether in the Payment field or others such as Supply chain and logistics monitoring, Financial Services, Personal Identity Protection, Secure sharing of medical data, among others.

Besides implementation in businesses, other studies and articles have addressed the use of blockchain and payment gateways:

A smart contract system for security of payment of construction contracts

A new smart contract payment security system named SMTSEC for eliminating or reducing payment issues in the construction sector was suggested by Salar Ahmadisheykhsarmast and Rifat Sonmez in 2020 [26]. The smart contract-based system was created to ensure the safety of payments for works in progress and to provide a platform for the secure, efficient, timely, and transparent payment of construction projects.

The proposal was based on five main stages:

1. Coding the conditions in smart contract;
2. Deploy the contract on the blockchain;
3. Blocking the amount of the payment related to the work in the smart contract;
4. Informing the smart contract by the employer that determined tasks and conditions were fulfilled by the contractor;
5. Contract enforcement and transaction of the blocked amount to the contract´s, subscontractor´s and supplier´s wallet addresses automatically.

SMTSEC revealed the potential of smart contracts to protect main contractors, subcontractors and suppliers against the insolvency of the principal or late payments. Furthermore, it ensures the availability of funds for a progress payment period by blocking the projected progress payment amount at the start of the period, and it allows direct payment from the employer's wallet to the subcontractors and suppliers wallets, thereby improving cash flow and reducing payment issues.

E-commerce payment model using blockchain

In 2020, Shee-Ihn Kim and Seung-Hee Kim wrote a paper proposing a simple payment model that incorporates fundamental cryptocurrency characteristics like a public key, private key, and digital signature to eliminate the requirement for transaction intermediates like public key certificates and payment gateway (PG) [27]. This was created since existing e-commerce payment systems for credit cards or checks require a PG, which imposes PG fees, raising the cost of participating in e-commerce. The proposed payment system for blockchain e-commerce authenticates the payment in three components: the merchant, the customer's smartphone application, and the blockchain system.

A message including the merchant's address, amount, transaction time, and transaction ID is configured by the merchant subsystem. The message is signed using the merchant's private key to produce a digital signature, which is displayed as a QR code on the merchant's online shopping mall screen, along with the four parameters. The customer receives the message and the merchant's digital signature by scanning the QR code (digitalsignature1). To produce the customer's digital signature, the acquired digitalsignature1 is signed using the customer's private key (digitalsignature2).Then the consumer sends the blockchain system the message (which includes the merchant's address, amount, transaction time, and transaction ID), as well as digitalsignature1 and digitalsignature2.

Finally, the blockchain system receives the message (containing the merchant's address, amount, transaction time, and transaction ID), digitalsignature1, and digitalsignature2 from the customer.The public key of the customer is then used to validate digitalsignature1 and digitalsignature2. When the findings match, it is established that digitalsignature1 is genuine and that the client is the only

one who could have supplied it. The merchant's public key is then used to validate digitalsignature1 and the message (which includes the merchant's address, transaction time, and transaction ID). When the findings match, the message's integrity is established, and the merchant is the only one who could have sent it. Through this verification procedure, the payment transaction's integrity and nonrepudiation are ensured.

Chapter 3

System Implementation

After introducing the principles and areas to implement Blockchain and a POS system, the purpose of this chapter is to present the DAPP design and implementation of the system designed in this dissertation's work. Therefore, we started with a concise overview of the project case study, as well as the development tools and languages. Following that, the broad architecture of the smart contract system and the resulting APIs will be shown, with the individual components being discussed in better detail.

3.1 Application Scenario

This project creates a Decentralized Application (DApp) that is integrated into an existing POS system and allows customers to pay with Ethereum at retail establishments. The POS used is the Oracle Retail Xstore Point-of-service provided by Retail Consult, the company with which this dissertation was performed.

Owning and operating a retail store can be both a rewarding and exciting career path and a great challenge. Employees and business owners often have difficulty collecting money efficiently and properly, meaning that they spend a significant amount of time each month generating, distributing and reviewing bills, and errors can result in late payments.

In this sense, this application acts on top of a number of frequent billing errors, many of which can be reduced, if not eliminated, by using smart contracts and the blockchain. In Ethereum, a DApp is supported by a smart contract, which contains the DApp's logic. Because the code is duplicated across many peers in the network, a dapp operating on the Ethereum blockchain might constitute a significant improvement over current practices.

This new Ethereum payment process can be described by 4 general steps represented in Figure 3.1 which includes the two intermediates of the process, the retailer and the customer.

Figure 3.1: Ethereum payment process

The first step is executed by an employee at the POS, who registers the items purchased by the customer and, after selecting the "Ethereum" payment method, a Smart Contract is automatically created that will store the data related to this purchase. After this, the QR Code is displayed on the POS system either directly from the store's tablet or from the credit card reader.

At this point, it is the customer who intervenes and has to read the QR Code with his wallet application and confirms the payment request. Following that, according to the confirmation or rejection of the payment by the customer, the connected POS responds with an update on the status of the sale.

3.2 Ethereum Framework

For the implementation of the blockchain-based POS payment system described in this thesis, the Ethereum blockchain technology was ultimately chosen. There were a few factors that led to this decision:

- Ethereum is at the vanguard of blockchain technology, with a wide range of developer tools, a rich documentation, and a vibrant community.
- Support for the most complex smart contract technologies currently available, which enable a wide range of procedures to be executed.
- Support for developing blockchain-based applications that allow regular customers to access data and engage with smart contracts through more user-friendly interfaces.

The EVM is used by Ethereum to construct Solidity smart contracts and interface with them in Ethereum blockchains. Solidity is a computer language that was built specifically for the development of Ethereum smart contracts. As a result, while the language's syntax is straightforward, it has a number of unique features, most of which come from the fact that smart contracts are stored and evaluated in a decentralized, account-based system.

The following Ethereum development tools and libraries were utilized in the creation and implementation of our smart contract system:

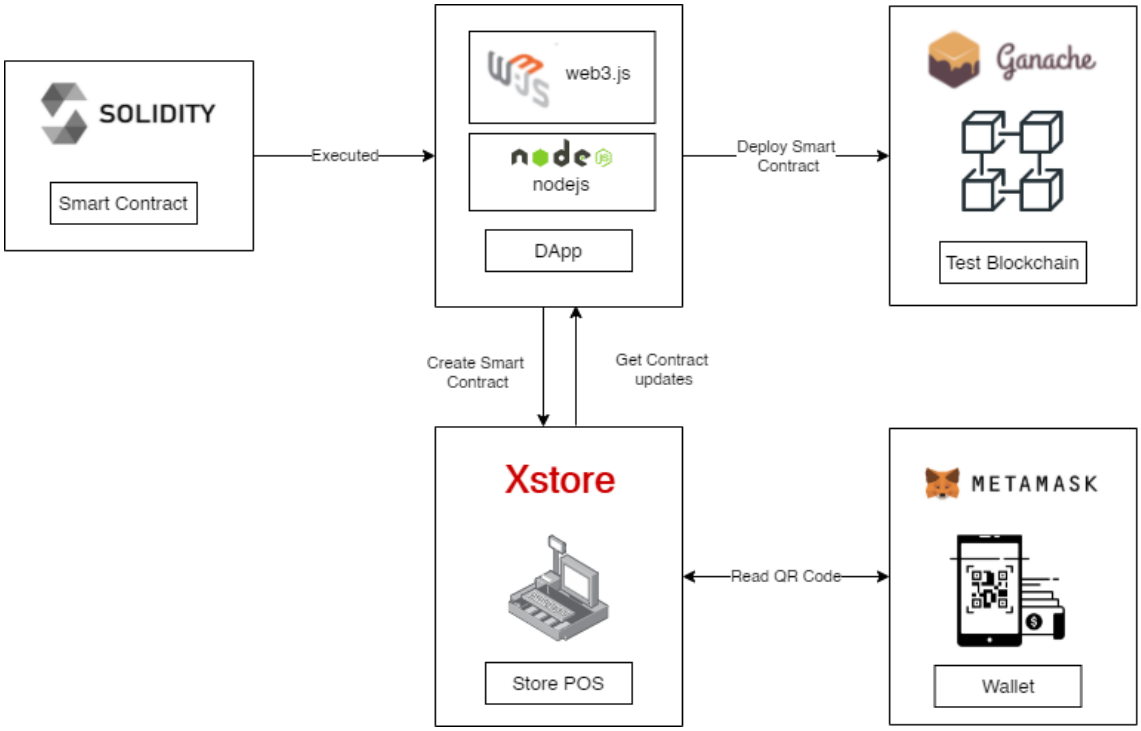


Figure 3.2: Ethereum framework

The DApp's smart contract was developed in Solidity, a high-level object-oriented language for developing smart contracts on blockchain systems. Our DApp operated locally on Ganache, with which we communicate via a Node.js application that uses the web3.js library, a JavaScript framework that offers an API for interacting with the Ethereum blockchain. Besides that, the Dapp was connected to Metamask in order to simulate the client and retailer wallet and allows to make the payment via QRCode.

Thus, the toolss seen in Figure 3.2, as well as an IDE (Integrated Development Environment), were used to execute the DAapp, which included executing the Solidity smart contract, deploying it on a custom blockchain, and running some Javascript tests. These tests were used to verify that the smart contract was executed correctly and to assess the system's overall performance.

3.3 Tools

An overview of the tools used to implement the solution is presented on the following pages.

Solidity

Ethereum uses its own language called Solidity (Figure 3.3) so all of the business logic will be coded into the Smart Contracts using Solidity [28]. Ethereum was the first decentralized app platform, and as a result, it is the Blockchain platform over which most

Figure 3.3: Solidity logo

developers are creating real-world apps. Aside from that, since the creation of Ethereum, several developers have been working on a variety of tools (Truffle, Infura, Web3.js) to make it easier for other developers to construct DApps. For all of these factors, as well as the availability and ease of access to knowledge, tools, and the technology's well-known success, Ethereum was chosen for the construction of this DApp.

Web3.js

Figure 3.4: Web3.js logo

Web3.js is a set of libraries that enables users to communicate with an Ethereum node, either locally or remotely, through an HTTP or IPC connection. The web3 JavaScript library interfaces with the Ethereum blockchain, allowing users to access user accounts, send transactions and engage with smart contracts, among other things [29].

Node.js

Figure 3.5: Node.js logo

Node.js (Figure 3.5) is a cross-platform open-source runtime environment for executing JavaScript code outside of a browser. Node.js is used to create back-end services (also known as APIs). These are the application that enables apps to run in a browser or on a smartphone, such as a web app or a mobile app. These client apps are the ones that the user interacts with and sees. They must connect with services on the server or in the cloud underneath the interface in order to save data, send emails or push alerts, start operations, and so on.

Node.js is suitable for building data-intensive, real-time back-end services for apps.

But why select node.js over other tools?

- Node.js is easy to pick up and use, making it perfect for quick prototyping and development.
- It is ideal for services that need to be quick and scalable. PayPal, Uber, Netflix, and Walmart are just a few of the organizations that utilize it.
- Node.js has the biggest open source libraries available to the general public.

For the development of this system, the Node Package Manager from node.js will be used. The major package manager for the JavaScript runtime environment Node.js is Node Package Manager (npm). It is a package manager for the programming language JavaScript. It includes an online database of public and paid private packages, known as the npm repository, and a command line client, also known as npm registry. The client is used to access the registry, while the npm website is used to browse and search for available packages. The package manager and the registry are managed by npm [30].

Ganache

Figure 3.6: Ganache logo

Ganache is an Ethereum development personal Blockchain that can be used to deploy contracts, build apps, and conduct tests. It comes in two flavors: a desktop program and a command-line tool (formerly known as the TestRPC). Ganache is a cross-platform application that runs on Windows, Mac OS X, and Linux. Our Blockchain Network will be based on this program. [31].

Metamask

Figure 3.7: Metamask logo

MetaMask is a browser plugin that allows users to conduct Ethereum or other network transactions directly from their browsers, eliminating the necessity for specialized Ethereum or other network user interfaces.

A user, for example, purchased a television from a retailer. He receives the payment information for that transaction, but he prefers to pay using Ether. To do so, he would have to write the transaction into the Ethereum network, then broadcast it throughout the

network using an Ethereum node. The retailer would then examine the Blockchain to see if the amount had been transferred successfully to its account.

Regular web browsers are unable to connect to an Ethereum node to read and write to the Blockchain, needing the use of a dedicated browser like Mist. This is where MetaMask comes in, by injecting a JavaScript package, web3.js, into the namespace of each website the browser opens, allowing normal browsers to write such transactions to the Blockchain.

MetaMask will be utilized on this system to make the process of confirming and putting transactions into the Ganache Ethereum easier.

3.4 Smart Contract Structure

In this section, we describe some relevant details of the implementation of smart contract in Solidity, which by convention have the extension ".sol".

The developed solution resulted in the integration of the customer's purchase invoice into an intelligent contract named "StoreInvoice.sol" that gathers and executes payment terms using programmed logic in a secure way. Therefore, the problem was divided into two parts: the business that is providing the service called "ServiceProvider" and the client who is consuming the service set as "ServiceConsumer".

The smart contract is formed, as previously mentioned when the employee registers the sale at the POS and within the SmartInvoice.sol file (listing 3.1) is written the contents refer to the purchase. The "pragma" keyword (line 1) is the first item written in code, and it specifies which version of the Solidity compiler should be used to compile the code. Thus we construct an instance of a contract and call it SmartInvoice, and within this, we define all of the state variables that make up the content of the business-client agreement, as well as the functions that will carry out the conditions of the agreement.

Regarding the variables, as we want both participants to be able to view the invoice's due date and the amount owed, we establish two state variables, "dueDate" and "invoiceAmount" (line 4,5), with public visibility so that an interface for this smart contract may easily obtain these values. Besides that, we also defined a third state variable, "serviceProvider" (line 6), which will store the business's blockchain address and a bool named "isStopped" (line 7) that is a state variable carrying the information if the contract is currently stopped or not. This variable is checked by the modifier "stoppedInEmergency" (line 9) which only works when the variable "isStopped" is false. The modifier function is used to change the behavior of functions declaratively, so in this case, it is useful to disable the contract, i.e., only when the variable is false it is possible to interact with and execute its functions, and only when the variable is true it is possible to stop the contract and close the sale.

Once we have established the state variables that will be used throughout our smart contract, we defined the contract's constructor method to initialize them. The constructor

function is executed only once at the time when the smart contract is created. Since the business will have different invoice amounts per client, the constructor was written to anticipate the "_invoiceAmount" argument (line 13) upon the contract creation.

Once the smart contract has been created with the invoice's due date and the amount that is due, the business owner can make it available to their client to scan and initiate payment. Therefore, a function to receive the money from the client and save it in the smart contract is written in Solidity to make the payment through a payable modifier to signal that this function can accept Ether. Inside we use a require function that validates that the value delivered in the transaction is equal to the "invoiceAmount" provided when the contract was created. If this condition is true, the function is executed, but in case it is false, the transaction is reversed and the error is delivered in the response along with a custom message that we defined.

It was also defined the "getContractBalance" function (line 25), which is a custom getter function that allows all participants in the contract to view the current balance. This function can run without paying for the gas required by the Ethereum network to process its execution since it is specified with the "view" keyword, which prevents the function from changing the state of the contract.

The next function "withdraw" (line 28) is the one that allows the company owner to remove the payment from the smart contract once the client has paid the billed amount. Within the body of this function, the condition requires (line 29-32) that the person trying to withdraw the contract's payment has to be the service provider. If the condition is met, we can execute the withdraw function. The msg.sender was used to retrieve the address from which the function was called. Then, using Solidity's "transfer" mechanism (line 33) we can send Ether, in wei, to the retailer address. In this scenario, the amount transferred is the balance of the smart contract, which can be accessed through "address(this).balance" (line 33).

However, before the Service Provider performs the withdraw, the contract can perform one more function, the "return_invoice" (line 35) which is the return of part or the totality of the payment made by the customer. In this case, the retailer only has to mention the amount to be returned and the address to which the amount will be sent.

That said, only one function remains to be mentioned. The "StopContract" function, as already mentioned at the beginning of this section, allows the manager to close and end the sale, disabling the contract from performing the functions that are restricted by the "stoppedInEmergency" modifier. Inside this the value of the variable "IsStoped" is changed to true, contradicting the restriction imposed by the modifier on line 9.

```
pragma solidity ^0.6.4;

contract SmartInvoice {
  uint public dueDate;
  uint public invoiceAmount;
  address serviceProvider;
  bool public isStopped;

      modifier stoppedInEmergency {
            require(isStopped == false);
            _;
      }
       constructor(uint _invoiceAmount) public {
            dueDate = block.timestamp + 120;
            invoiceAmount = _invoiceAmount;
            serviceProvider = msg.sender;
            isStopped=false;
      }
      fallback () payable external stoppedInEmergency {
            require(
              msg.value == invoiceAmount,
              'Payment should be the invoiced amount.'
            );
      }
      function getContractBalance() public view returns(uint)  {
            return address(this).balance;
      }
      function withdraw() public  stoppedInEmergency{
            require(
              msg.sender == serviceProvider,
              'Only the service provider can withdraw the payment.'
            );
            msg.sender.transfer(address(this).balance);
      }
      function return_invoice(uint return_value) public stoppedInEmergency  {
          msg.sender.transfer(return_value);
      }
        function stopContract() public returns (bool)  {
            isStopped = true;
            return isStopped;
        }

}
```

Listing 3.1: SmartInvoice.sol

3.5 DApp Structure

In order to be able to connect the smart contract presented in section 3.4 with the POS system, a DApp was developed that communicates with the Xstore through a REST API.

REST stands for Representational State Transfer, which means that there is no state between the client and the server, simply data that is processed and transferred. All that is required is to put some logic on a certain URL that processes the data and returns it in JSON format.

The Dapp was implemented in a javascript file called "dapp.js" that communicates with the Xstore. As so we start the code by importing the libraries used:

- Hapi framework, an open-source and rich Node.js framework for building applications and services and that provides more flexibility than other Node.js frameworks.
- Web3.js, a JavaScript library that provides an API to interact with an Ethereum blockchain.
- fs, the Node.js file system that enables to interact and read files

```
1  //Framework hapi.js
2  const Hapi = require('hapi');
3  //Web3.js
4  const Web3 = require('web3');
5  //define fs
6  const fs = require('fs');
```

Listing 3.2: Dapp.js libraries import

That done, we create a new "Hapi.server()" with connection information, including a port number for listening and the host address. It then launches the server and logs the fact that it is up and running.

```
1  // hapi server
2  const server = new Hapi.Server({
3    "host": "localhost",
4    "port": 3000
5  });
6
7  // start server
8  server.start(error => {
9    if (error) {
10     throw error;
11   }
12   console.log("Listening at " + server.info.uri);
13 });
```

Listing 3.3: Dapp.js Hapi Server Creation

To connect the ganache with the system we have to load the web3 package containing the web3.js library. As so, it is defined the "provider" (line 4 of listing 3.4) variable to reference a new instance of the package's main class which contains all things related to Ethereum. The "getAccounts" function (line 7 of listing 3.4) supplied by web3 provides the accounts that we established, indicating that the connection was successful.

To define the variables, we start by using an asynchronous function "getServiceConsumer" to assign the service Consumer with one of the addresses retrieved by the ganache. With the evolution of the project, the Service Consumer address was acquired by connecting with Metamask and therefore this function has been made ineffective. Besides that, the "serviceProvider", "contractAmount" and "serviceConsumer" variables (line 16-18 of listing 3.4) are here initialized.

```
1  //connect to ganache
2
3  //gui
4  const provider = new Web3.providers.HttpProvider('http://127.0.0.1:7545');
5
6  //get available accounts
7  web3.eth.getAccounts()
8    .then(console.log);
9
10 //definition and variable initialization
11 async function getServiceConsumer() {
12   const acc = await web3.eth.getAccounts();
13   setServiceConsumer(acc[0]);
14 }
15
16 var serviceProvider = null;
17 var contractAmount = null;
18 var serviceConsumer = null;
19
20 getServiceConsumer();
```

Listing 3.4: Dapp.js ganache connection and variable initialization

To deploy the contract, we use the deploy method that is available to a Contract object so we first need to create a new instance of that object. A jsonInterface object must be given as a parameter to the Contract constructor. Therefore, a new variable named abi was created (line 2 of listing 3.5), which reads the "StoreInvoice_sol_SmartInvoice.abi" file into the Node.js server and assigns the object returned to the new variable. After we provide it to the "contract" constructor to use (line 5 of listing 3.5).

Aside from that, the deploy method requires a data parameter that contains the smart contract's bytecode in string format. Thus, we utilized the .bin file that held our smart contract's bytecode format, and we created a new variable called bytecode (line 8 of

listing 3.5) that read the "StoreInvoice_sol_SmartInvoice.bin" file into our Node.js server and convert the returned buffer to a string.

```
1    //ABI description as JSON structure
2    var abi = JSON.parse(fs.readFileSync('StoreInvoice_sol_SmartInvoice.abi'));
3
4    //create contract proxy class
5    var contract = new web3.eth.Contract(abi);
6
7     // smart contract EVM bytecode
8    var bytecode = fs.readFileSync('StoreInvoice_sol_SmartInvoice.bin').toString();
```

Listing 3.5: Dapp.js JSON structure and bytecode

Two functions were now built up to assign values to the contract variables. The first one is called "setContractInfo" (line 2 of listing 3.6) and the attributes are obtained via the Xstore whereby the "sProvider" is the address recorded at the POS and the "amt" the total when the employee registers the sale and the products.

The other function refers to the assignment of the client's address, "setServiceConsumer" which is obtained through the connection made with Metamask once the payment QRCode is scanned, a process that will be explained in section 4.1.

```
1    //set serviceProvider
2    function setContractInfo(sProvider, amt) {
3      serviceProvider = sProvider;
4      contractAmount = amt;
5    }
6
7    //set serviceConsumer
8    function setServiceConsumer(sConsumer) {
9      serviceConsumer = sConsumer;
10   }
```

Listing 3.6: Dapp.js variables allocation

Afterward, IT is finally possible to pass to the deploy method with the stringified bytecode and the amount argument. This is inserted into an asynchronous function "contractDeploy" (line 2 of listing 3.7) since is used the keyword "await". This way, when there is an await for a promise, the function is paused in a non-blocking manner until the promise is completed. If the promise satisfies, the value is taken back, otherwise, the rejected value is discarded.

Additionally, the send method of the transaction object is called to deploy the smart contract to Ethereum. This method requires a "from" parameter containing the address the transaction should be sent from, the "gasPrice" argument that defines the gas price in wei to use for this transaction, and the "gas" argument to define the maximum units of gas that we are willing to expend on this transaction, i.e. the transaction gas limit. If

the smart contract is successfully deployed, the send method will return an address that resolves with the newly deployed contract.

```
1  //function to deploy contract
2  async function contractDeploy(serviceProvider, amount) {
3    //deploy contract
4     console.log('Deploying contract');
5
6     const contractInstance = await contract.deploy({
7          data: bytecode,
8          arguments: [amount]
9        }).send({
10         from: serviceProvider,
11         gasPrice: web3.utils.toWei('0.0000000128', 'ether'),
12         gas: 1500000
13       }).then((deployedContract) => {
14         contract.options.address = deployedContract.options.address;
15         console.log('Finish DEPLOY');
16       });
17
18    return contract.options.address;
19 }
```

Listing 3.7: Dapp.js contract deployment

In order to receive the values and addresses from the POS, four different services were defined according to the type of information that was being passed on.

Initially, we write a service that receives the serviceProvider address and sends the smart contract address. It fetches the contract information from the patch containing the URL with the necessary arguments. In this case, it accesses the "contractInfo" (line 4 of listing 3.8) and then separates this data, extracting from it the serviceProv and the amount. The deployment of the contract is then completed by invoking the contractDeploy function described above in the listing 3.7.

```
1  //Receive serviceProvider address and send contract address
2  server.route({
3    method: 'GET',
4    path: '/dapp/contractInfo/{contractInfo*2}',
5    handler: async (request, h) => {
6      const contractInfoAux = request.params.contractInfo.split('/')
7
8      const serviceProviderAux = contractInfoAux[0];
9      const amountInfoAux = Number(contractInfoAux[1]).toFixed(18);
10     console.log(amountInfoAux);
11     const amountAux = web3.utils.toWei(amountInfoAux, 'ether');
12     setContractInfo(serviceProviderAux, amountAux);
13     const result_contractDeploy = await contractDeploy(serviceProviderAux,
   amountAux);
14
```

```
15        return contract.options.address;
16      }
17    });
```

Listing 3.8: Dapp.js service that receives serviceProvider and sends contract address

The service shown in Listing 3.9 was established so that the balance could be accessed using the getContractBalance function defined in the smart contract code. This way, the value is stored in the variable "cBalance" so that it can later be accessed by other services and perform actions according to the amount present.

```
1   //check the contract balance
2   server.route({
3     method: 'GET',
4     path: '/dapp/checkBalance',
5     handler: async (request, h) => {
6       const cBalance = await contract.methods.getContractBalance().call();
7       console.log("balance" + cBalance);
8       return cBalance;
9     }
10  });
```

Listing 3.9: Dapp.js service that checks the contract balance

Using the withdraw and stopContract functions defined in Listing 3.1, this service aims to withdraw the amount entered in the smart contract corresponding to the payment for the purchase and close the smart contract, making it unable to perform its functions. Both functions take an address as a parameter, which in this case is the address relative to the service provider.

```
1   //make withdraw and close contract
2   server.route({
3     method: 'GET',
4     path: '/dapp/makeTransaction',
5     handler: async (request, h) => {
6       contract.methods.withdraw().send({ from: serviceProvider });
7       contract.methods.stopContract().send({ from: serviceProvider });
8       return true;
9     }
10  });
```

Listing 3.10: Dapp.js service that makes withdraw and close the contract

In some cases, the client may request a refund, which this service 3.11 provide by using the return_invoice method of the .sol file. This resource is used at the end of the order process when a request is made to confirm the purchase's conclusion. If the contract balance is greater than zero (line 8 of 3.11), the return is executed and then the contract is closed. Otherwise, only contract closing will occur (line 12 of 3.11), since there is no amount entered in the contract.

```
// return invoice and close contract
server.route({
  method: 'GET',
  path: '/dapp/returnInvoice',
  handler: async (request, h) => {
    console.log("contractAmount" + contractAmount);
    const cBalance = await contract.methods.getContractBalance().call();
    if (cBalance > 0) {
      contract.methods.return_invoice(contractAmount).send({ from:
serviceProvider });
      contract.methods.stopContract().send({ from: serviceProvider });
    } else {
      contract.methods.stopContract().send({ from: serviceProvider });
    }
    return contractAmount;
  }

});
```

Listing 3.11: Dapp.js service that returns invoice and closes the contract

3.6 POS Structure

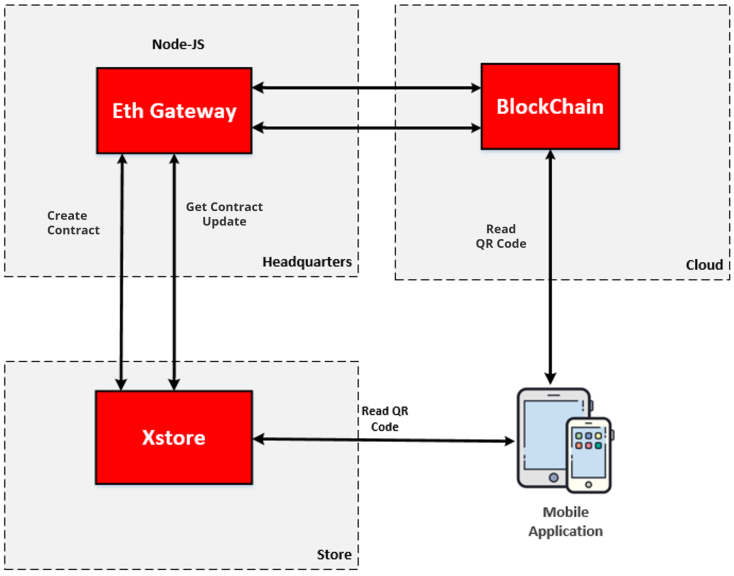


Figure 3.8: System architecture diagram

Regarding the POS system, we used one already developed by Oracle, the Oracle Retail Xstore Point-of-service. This was adapted in order to allow the implementation with the DApp developed described above in the section 3.5. The general architecture diagram of the POS and the remaining components is represented in the Figure 3.8.

Here, the services are established in the same way as they are in a Dapp, so two distinct services were formed: one related to the contract details, such as quantity and service provider. And another that is for the conversion of fiat currency to ether.

The first service begins by defining a route to the patch that is utilized in the DApp. This string is constituted with the variables localhost, serviceProvider, and amount (line 2 of listing 3.12). On lines 10 and 11 of the Listing 3.5 is created the "requestHttp" which subsequently creates the builder with the serviceProvider and a response is acquired by executing the call on line 15.

As a result, the "responsebody" is stored in the value key "FTD_CONTRACT_ADDRESS" (line 19 of listing 3.12) in order to record the contract information and then used to create the QRCode for the purchase payment.

```
1    //URLs
2    String url_serviceProvider = "http://localhost:3000/dapp/contractInfo/" +
     serviceProvider + "/" + amount;
3
4    //restStrings
5    StringBuilder restString_sProvider = new StringBuilder("/serviceProvider/" +
     serviceProvider).append("\r\n}");
6
7   //-----------------------------Service---------------------------------------//
8
9    //GET REQUEST service_provider and amount RESPONSE contract_address
10   RequestBody body = RequestBody.create(mediaType, restString_sProvider.toString
     ());
11   Request requestHttp = new Request.Builder().url(url_serviceProvider).get().
     build();
12
13   OkHttpClient client = builder.build();
14   _logger.debug("Beginning communication with Rest service");
15   Response response = client.newCall(requestHttp).execute();
16   responseBody_getContractAddress = response.body().string();
17   _logger.debug("Rest response: " + responseBody_getContractAddress);
18
19   _transactionScope.setsValue(FtdQRValueKeys.FTD_CONTRACT_ADDRESS,
20     "ethereum:" + responseBody_getContractAddress);
21
22   return this.HELPER.completeResponse();
23   }
```

Listing 3.12: POS service that establishes smart contract information

We use the CoinMarketCap API to construct another service that allows the merchant to convert the amount to be paid from fiat money to ether. It was necessary to create an account on the site in order to use the API, which created an API-Key for us to use (line 2 of listing 3.13) .

Following that, the function used was Price Conversion, which enables for the use of the most recent market rate for each currency. The arguments required to pass were the amount, the currency symbol of the base fiat to convert from, in our case is euro, and the cryptocurrency symbol to convert the source amount to, which is Ethereum (line 5 of listing 3.13).

```
1    //CoinMarketCap API
2   String apiKey = "a561ec32-eb8b-48d3-b9c0-a8c2e1154cfc";
3
4    String url = "https://pro-api.coinmarketcap.com/v1/tools/price-conversion?
     CMC_PRO_API_KEY=" + apiKey
5        + "&amount=" + eur_amount + "&symbol=EUR&convert=ETH";
```

Listing 3.13: POS service that connects with CoinMarketCap API

Chapter 4

Demonstration

In this chapter, the purpose is to demonstrate the system functioning to handle the cases it was made for. To achieve this, we divide the chapter into 2 sections, one concerning the initialization of the applications used in the project and another in which the different use cases considered in the payment process in a retail store are exposed.

4.1 Application Startup

To execute our project we used different programs, so this section describes how these were launched in order to be able to create a complete and successful case study.

4.1.1 Ganache

Ganache is an Ethereum developer tool that allows users to simulate a blockchain environment locally and test deployed smart contracts. As so, we use Ganache throughout the development process, allowing us to create, deploy, and test our dApps in a secure and predictable environment.

The Ganache app is displayed in Figure 4.1. This program creates an Ethereum-based Blockchain with ten accounts/wallets initialized each with 100 ETH for the user to test the system.

To allow the connection between the ganache and our developed program it is necessary to connect it to the default localhost represented by the network, http://127.0.0.1:7545.

4.1.2 Metamask

The metamask extension was used to simulate the digital wallets of both the retailer and the customer. It is through this program that intermediaries can check their balance, and that will later be used to scan the QR Code at the time of payment.

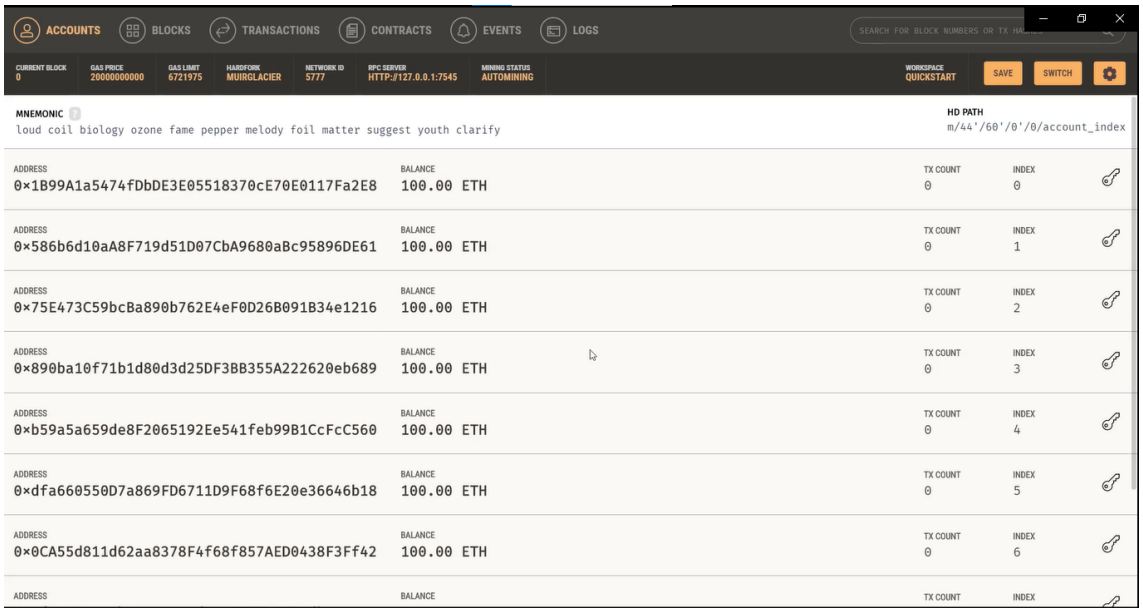


Figure 4.1: Ganache interface

A password that was established when we first initialized MetaMask is required to access it the next time. MetaMask has the ability to connect to numerous networks as one of its features (Figure 4.2). As a result, in order to connect to the business network, the network's URL must be entered (Figure 4.3). In this example, the URL inserted is localhost with the port supplied by ganache, which is where the Blockchain is, as previously mentioned.

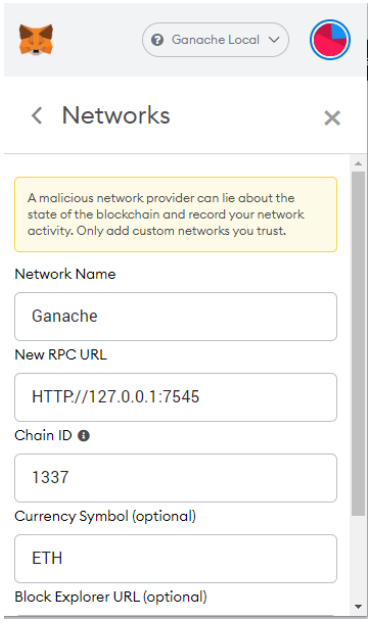


Figure 4.2: MetaMask connecting to the right network

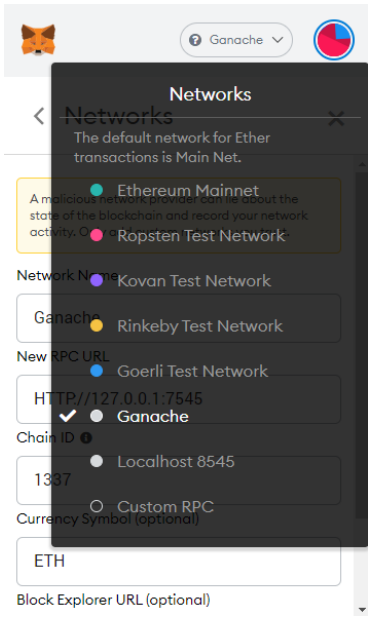


Figure 4.3: MetaMask choosing the right network

We must link to a specific wallet after successfully connecting to the network. In this example, the first account is designated as the client, and we must enter its private key supplied in ganache in order to utilize it with MetaMask (Figure 4.4).

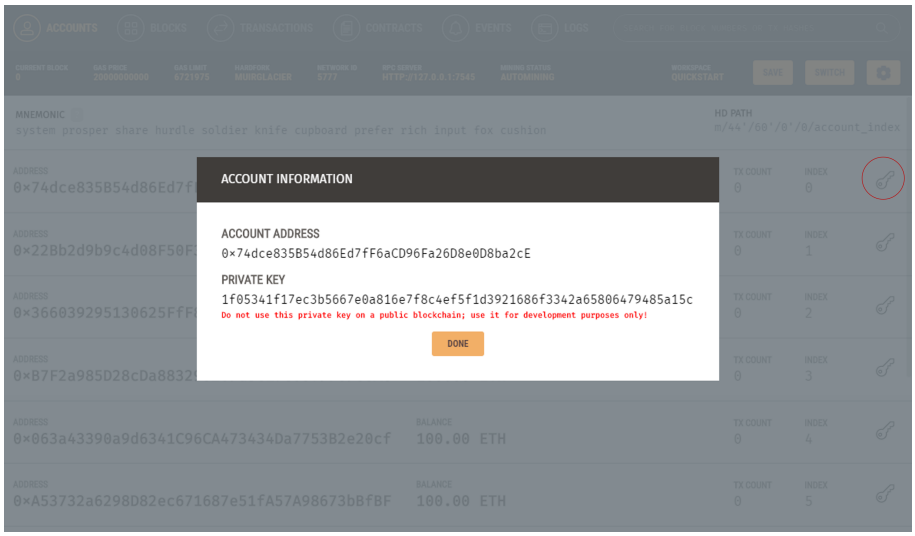


Figure 4.4: Ganache interface

After obtaining the private key for a specific wallet from ganache, the key is entered into metamask to establish an account through the "import account" option and begin using that wallet (Figure 4.5). We associate two accounts, one that corresponds to the retailer and the other to the customer so we can check the balance of their wallets directly. (Figure 4.6).

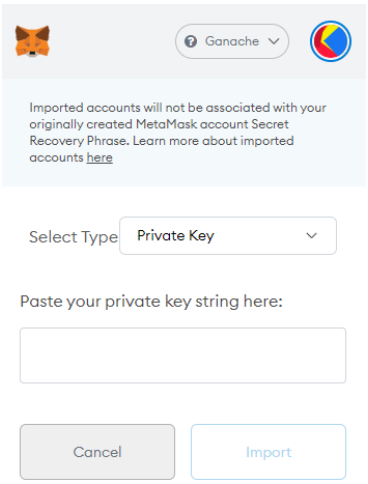


Figure 4.5: MetaMask import account

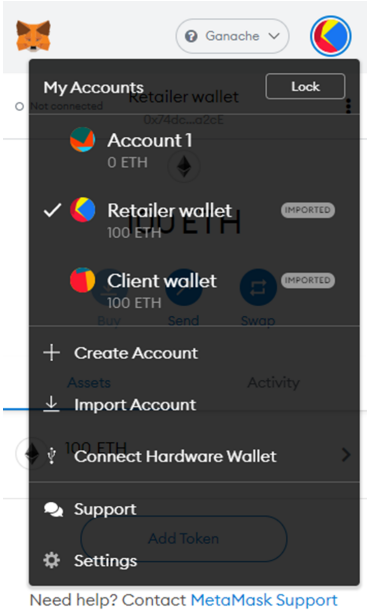


Figure 4.6: MetaMask accounts available

4.1.3 POS

The POS is set up using the Dapp terminal, which involves launching a local Node.js server and connecting one of the ganache accounts to the system's POS. After that, as a result of the successful connection, the available accounts in ganache are presented on the terminal.

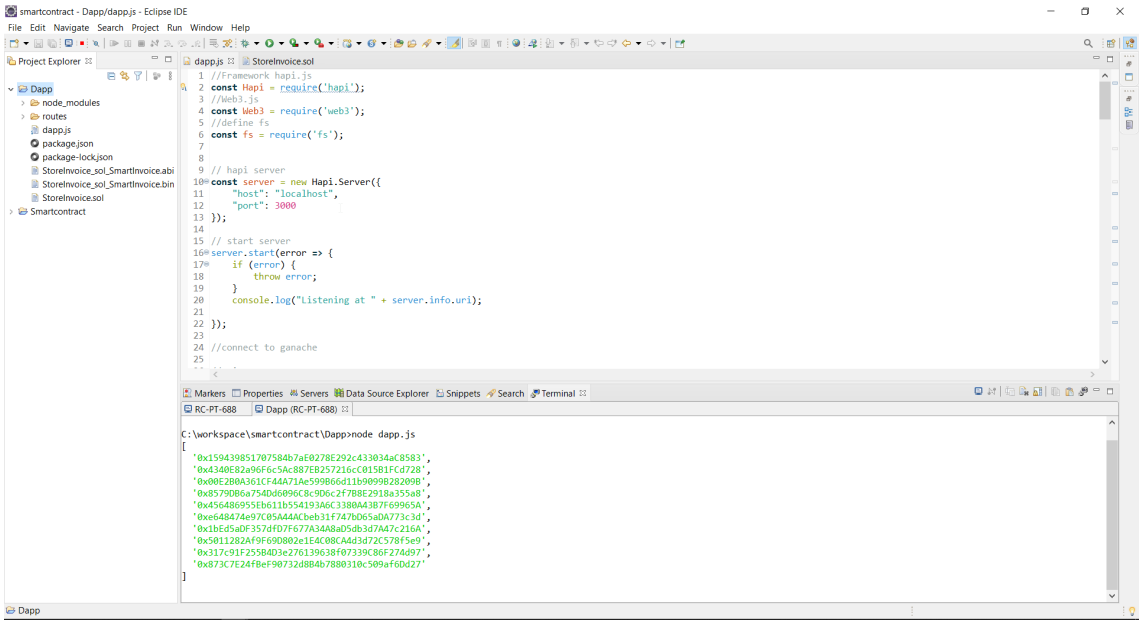


Figure 4.7: POS successful connection

4.2 Use Cases

In this section, we will go over the four use cases investigated in our decentralized app.

4.2.1 Transaction completed

According to the POS version used, the employee initially needs to be logged in with his credentials to make the sale and associate it with one of the customers present on the store's list. After that, as shown in the figure 4.8 he can register the products selected by the customer and choose the payment method he wants.

At this point the worker confirms the amount to be paid in Ether and the smart contract is created and deployed. As a result, the QR Code is displayed for the client to scan as well as the value in Ether corresponding to the amount and the address of the contract to which the value will be sent (figure 4.9). To allow the consumer to accept the payment, a two-minute wait time was established while the POS stands in a constant update of its balance.

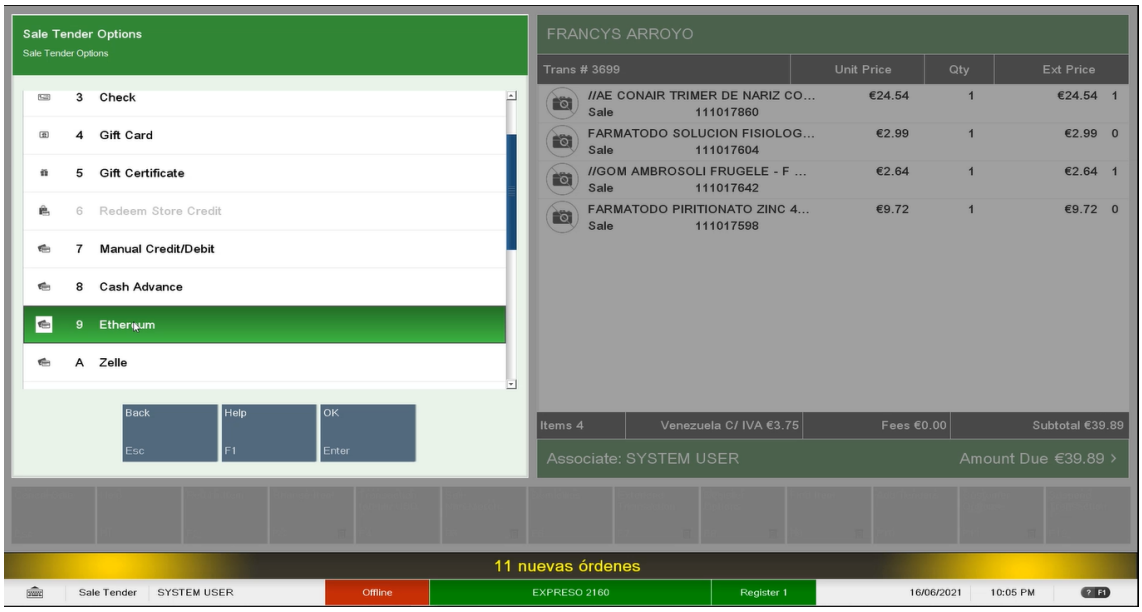
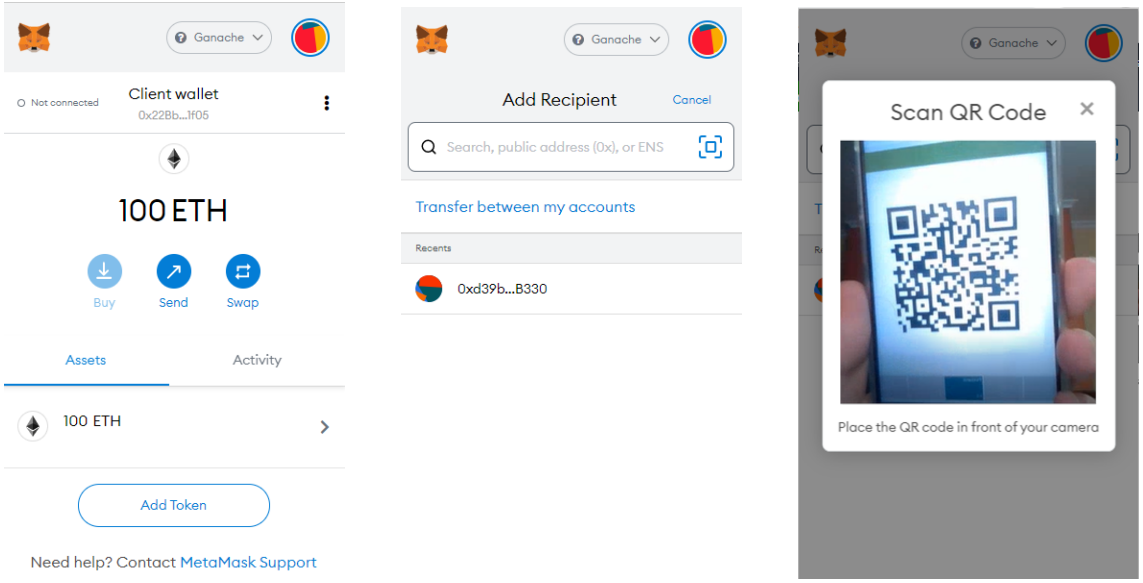


Figure 4.8: POS products register and payment method selection

Figure 4.9: POS QR Code display

In order to simulate the client's role, the metamask is then opened in the client's wallet (figure (a) 4.10) and the send option is selected as well as the scan QR Code option (figure (b) 4.10) via the computer's webcam (figure (c) 4.10).

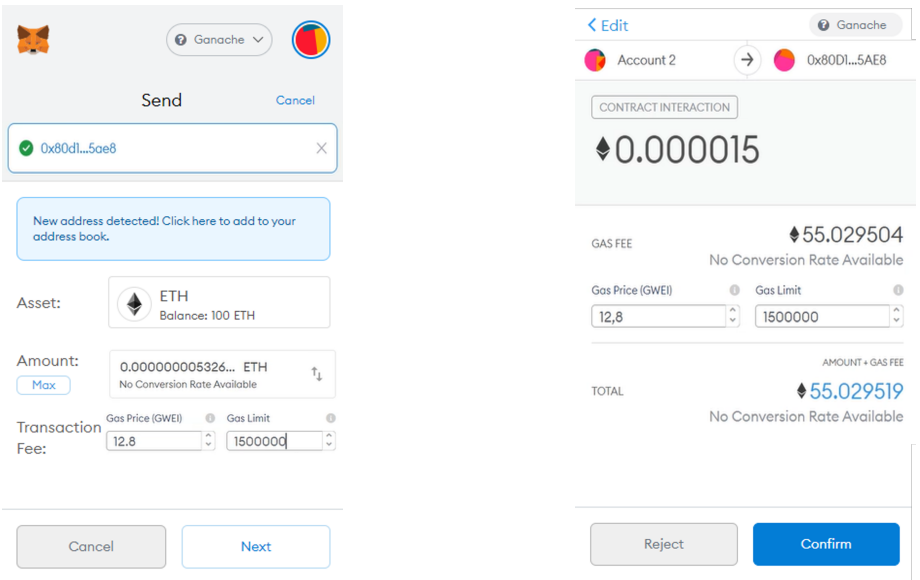
Subsequently, the address of the smart contract is automatically read from the QRCode and the request for payment for the purchase can be accepted. It should be mentioned that in this step there is a difference between what will occur in reality and how will happen in our simulation. The idea is that the amount to be paid will be inherently included in the



(a) (b) (c)

Figure 4.10: Metamask QR Code scan

QR Code, and the client will just have to approve or reject the payment request. However, Metamask has a limitation since the QR Code must be in the format "ethereum: + address" and so the amount must be entered manually as the transaction fees (figure 4.11).



(a) Enter payment details

(b) Accept request

Figure 4.11: Confirm payment transaction

Following confirmation, an update is received on the POS side indicating that the contract balance has been altered, and therefore a confirmation of the successful transaction is displayed (figure 4.12).

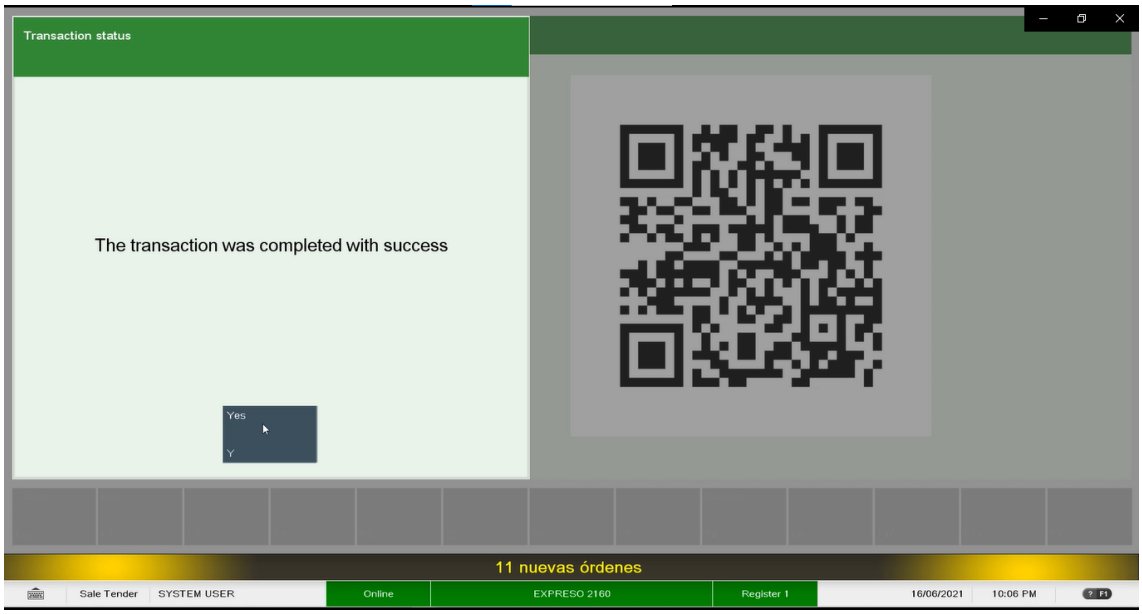


Figure 4.12: POS successful transaction

The sale close request is the final step in the process, which corresponds to the contract closure (figure 4.13). If the answer is "Yes", the amount will be withdrawn and the contract methods will be deactivated. We will look at the "No" choice in the following section 4.2.2.

Figure 4.13: POS sale close

The sale is then concluded and we can verify the existence of four transactions in ganache corresponding to:

1. Creation of the smart contract related to the sale

2. Sending ether from the client´s wallet to the smart contract
3. Performance of the withdraw by the service provider
4. Closing of the smart contract

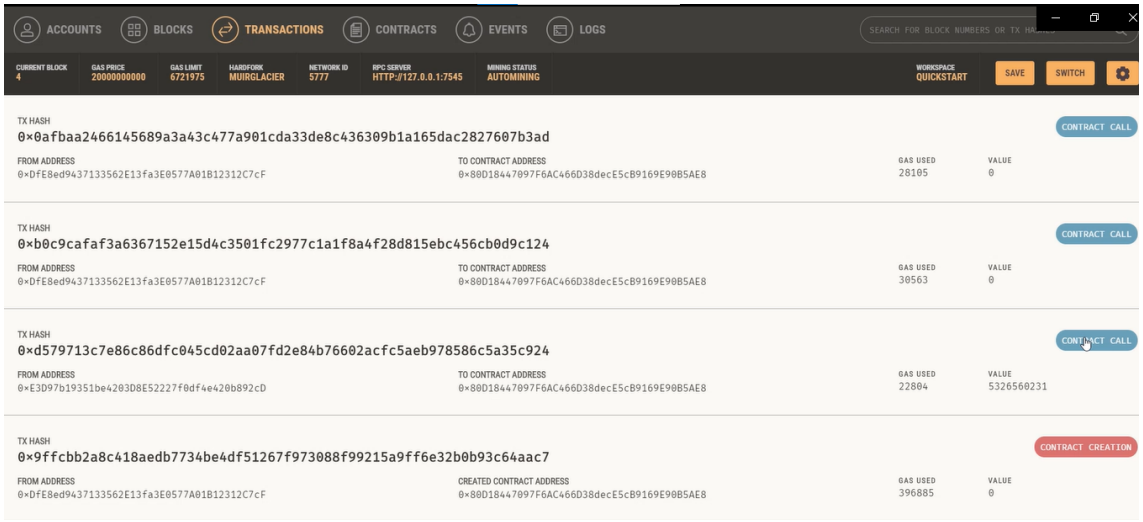


Figure 4.14: Ganache transactions report

4.2.2 Transaction canceled

As mentioned in the previous section, the customer has the option to request a refund of his payment. This will only be done before the smart contract amount is withdrawn, otherwise, if the consumer requests it after, a direct transfer from the retailer's wallet to the customer's wallet will be required.

This is done by selecting the "No" option when asking for confirmation of the closing of the purchase as shown in the figure 4.13. As a result, the message "The transaction was canceled" is displayed (figure 4.15) and the return service is called executing the returne_invoice() and stopContract() functions.

4.2.3 Transaction completed with different payment methods

Another use case considered was when the customer wants to pay the total bill using two payment methods. In this situation, the employee must indicate at the POS the amount to be paid in each currency. In figure 4.16 part of the amount is introduced to be paid in ethereum, and a QR Code is subsequently created with the amount selected to allow payment. After this payment is completed, the amount paid is discounted from the total purchase, and the rest of the payment is normally made through another payment method selected by the customer.

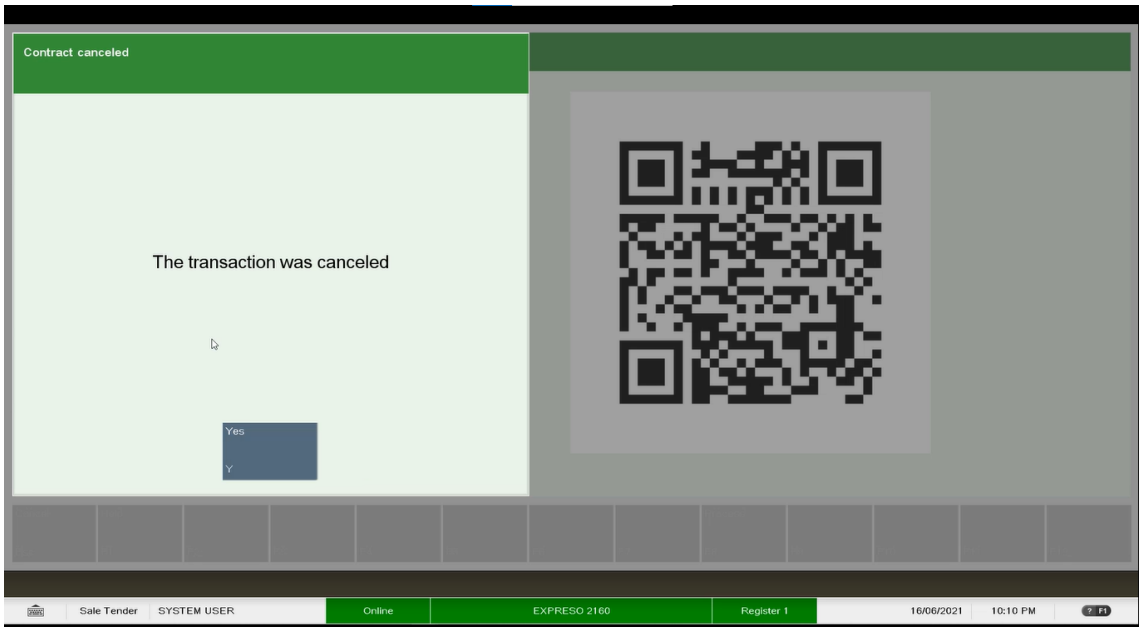


Figure 4.15: POS sale canceled

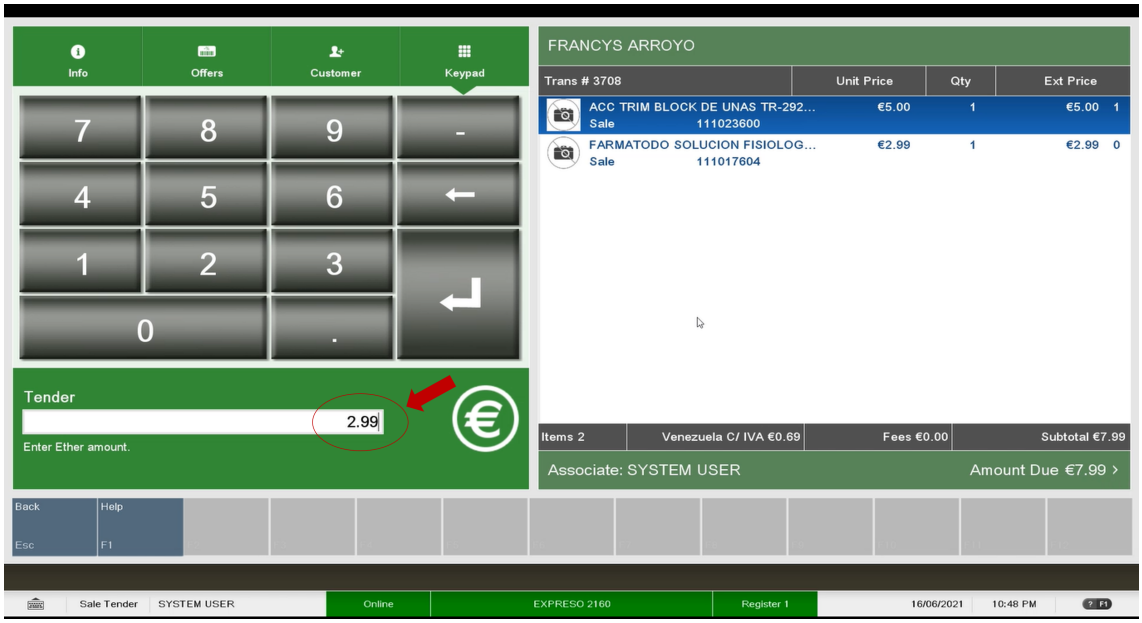


Figure 4.16: POS two payment methods

4.2.4 Transaction canceled due to time limit exceeded

The last scenario considered refers to when the customer does not scan the QR Code within the 2 minute time limit determined by the POS. In this case, only the contract is closed and a new purchase is initiated, if necessary.

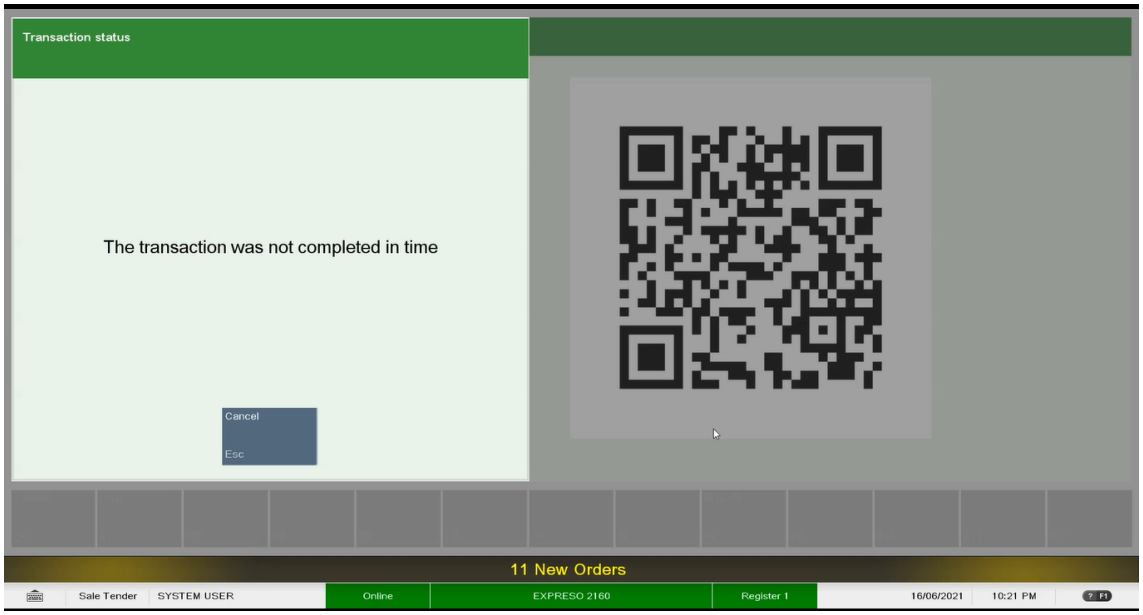


Figure 4.17: POS canceled by timeout

Chapter 5

Conclusion

5.1 Achieved Objectives

This dissertation presented the design and implementation of a blockchain smart contract system to be integrated into POS in order to make payments with Ethereum. Several distinct functionalities have been effectively incorporated in a decentralized blockchain system to this aim, including:

1. A Ethereum POS system, implemented with smart contract technologies;
2. A decentralized registry for tracking and tracing data of payments in retail stores;
3. Automated debugging and quality control feautures that take into account in the purchase billing process;
4. A web application that serves as a communication bridge between the POS system and the blockchain smart contract system.

The solution described here showcases how blockchain technology may be used to solve some of the difficulties that the payments industry faces while also utilizing the decentralized nature of these systems. The tracking data is made available to all retail companies and customers, culminating in transparency. Aside from that, because the system is decentralized, no intermediaries are in charge, and data integrity is maintained on a secure, shared database. We choose Ethereum as the smart-contract enabled blockchain because it has a higher level of decentralization than the alternatives and a large development community.

As stated at the beginning of this document, the trend toward the usage of cryptocurrencies is expected to grow and diversify, and therefore it can only be expected that the resulting decentralized application will drive crucial technical developments in retail payment systems and ensure competitiveness against other markets.

5.2 Future Work

For future work, the idea would be to implement this case study in a real store environment. To do this, the blockchain test network used through ganache would have to be replaced and metamask would be substituted by the wallets of the players. Besides that, we would like to obtain more statistics and tests on payment processes. These could be used to make a more thorough cost analysis of our system and eventually find improvements to it that would better fit the retailer's needs.

Another factor to consider is how the retailer's Ether wallet balance will be converted to fiat cash. This process would need to be thoroughly investigated, and a market analysis should be performed to examine which applications and players would be the most suitable.

Other proposal is to apply Blockchain and Smart Contracts to other fields. As mentioned in the section, the POS has more functionalities beyond the basic payment process. Extending the use of blockchain would be very useful and beneficial to the retailer, such as in inventory and stock management, sales monitoring and reporting, customer relationship management, employee and order management, among others.

References

[1] Emmi Martikainen, Heiko Schmiedel, and Tuomas Takalo. Convergence of European retail payments. *Journal of Banking and Finance*, 50:81–91, jan 2015. doi:10.1016/j.jbankfin.2014.09.021.

[2] Marion Laboure and Jim Reid. The Future of Payments Part III . Digital Currencies : the Ultimate Hard Power Tool. (January 2020), 2020.

[3] Arnab Dey. New Method of POS based on Artificial Intelligence and Cloud Computing. page 6, 2019.

[4] Kira Deutch. What Is a Point-of-Sale System And How Does It Work?, oct 2020. URL: https://squareup.com/us/en/townsquare/what-pos-system.

[5] Emerchantpay. What is a Payment Gateway and How Does It Work?, jul 2019. URL: https://www.emerchantpay.com/insights/what-is-a-payment-gateway-and-how-does-it-work/.

[6] Slava Gomzin. *Hacking Point of Sale : Payment Application Secrets, Threats, and Solutions*. John Wiley & Sons, Inc, 2014.

[7] Matteo Monti and Steen Rasmussen. RAIN: A Bio-Inspired Communication and Data Storage Infrastructure. 2017. doi:10.1162/ARTL_a_00247.

[8] Andreas Antonopoulos. 7. The Blockchain - Mastering Bitcoin [Book]. chapter 7. 2014. URL: https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch07.html.

[9] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, pages 557–564. Institute of Electrical and Electronics Engineers Inc., sep 2017. doi:10.1109/BigDataCongress.2017.85.

[10] L. M. Bach, B. Mihaljevic, and M. Zagar. Comparative analysis of blockchain consensus algorithms. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, pages 1545–1550. Institute of Electrical and Electronics Engineers Inc., jun 2018. doi:10.23919/MIPRO.2018.8400278.

[11] Juri Mattila. THE BLOCKCHAIN PHENOMENON The Disruptive Potential of Distributed Consensus Architectures. Technical report, 2016. URL: http://brie.berkeley.edu/BRIE/.

[12] O'Reilly Media and Sebastopol. *Blueprint for a New Economy*. 2015.

[13] Nick Szabo. The Idea of Smart Contracts. Technical report, 1997. URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/L...

[14] Capgemini Consulting. Smart Contracts in Financial Services: Getting from Hype to Reality. Technical report, 2016.

[15] Lodovica Marchesi, Michele Marchesi, Giuseppe Destefanis, Giulio Barabino, and Danilo Tigano. Design Patterns for Gas Optimization in Ethereum. In *IWBOSE 2020 - Proceedings of the 2020 IEEE 3rd International Workshop on Blockchain Oriented Software Engineering*, pages 9–15. Institute of Electrical and Electronics Engineers Inc., feb 2020. doi:10.1109/IWBOSE50093.2020.9050163.

[16] Arijit Chakrabarti and Ashesh Kumar Chaudhuri. Blockchain and its Scope in Retail. *International Research Journal of Engineering and Technology*, 2017. URL: www.irjet.net.

[17] Hasib Anwar. Blockchain in Payment: Accelerating Payment Services, dec 2020. URL: https://101blockchains.com/blockchain-in-payment/.

[18] Yuri Musienko. 5 Benefits of the Blockchain Payment System. *Merehead*, dec 2019. URL: https://merehead.com/blog/5-benefits-blockchain-payment-system/.

[19] Blockchain in Payments- Transforming the payments industry. URL: https://www.leewayhertz.com/blockchain-in-payments/.

[20] Karmila Sari Sukarno. The Use of Cryptocurrency as a Payment Instrument. 130(Iclave 2019):366–370, 2020.

[21] Akash Vijayan, Akeel Mohammed Ashique, Mathew Koshy Karunattu, Ansamma John, and Manu J. Pillai. Digital Payments: Blockchain based Security Concerns and Future. In *Proceedings - International Conference on Smart Electronics and Communication, ICOSEC 2020*, pages 429–435. Institute of Electrical and Electronics Engineers Inc., sep 2020. doi:10.1109/ICOSEC49089.2020.9215431.

[22] Lastovetska Anastasiia. Blockchain Architecture Explained: How It Works & How to Build, jan 2021. URL: https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture.

[23] Payments Forum. Mobile and Digital Wallets : U . S . Landscape and Strategic Considerations for Merchants and Financial Institutions About the Mobile and Contactless Payments. (January):1–50, 2018.

[24] CNNMoney. Microsoft begins accepting Bitcoin | Hartford Business Journal. *Hartford business*, dec 2014. URL: https://www.hartfordbusiness.com/article/microsoft-begins-accepting-bitcoin.

[25] Overstock.com Now Accepts All Cryptocurrencies | PYMNTS.com, aug 2017. URL: https://www.pymnts.com/news/retail/2017/overstock-now-accepts-all-cryptocurrencies/.

[26] Salar Ahmadisheykhsarmast and Rifat Sonmez. A smart contract system for security of payment of construction contracts. *Automation in Construction*, 120(August):103401, 2020. URL: https://doi.org/10.1016/j.autcon.2020.103401, doi:10.1016/j.autcon.2020.103401.

[27] Shee Ihn Kim and Seung Hee Kim. E-commerce payment model using blockchain. *Journal of Ambient Intelligence and Humanized Computing*, (2008), 2020. URL: https://doi.org/10.1007/s12652-020-02519-5, doi:10.1007/s12652-020-02519-5.

[28] Solidity Documentation Release 0.8.5 Ethereum. Technical report, 2021.

[29] web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation. URL: https://web3js.readthedocs.io/en/v1.3.4/.

[30] About | Node.js. URL: https://nodejs.org/en/about/.

[31] Ganache | Overview | Documentation | Truffle Suite. URL: https://www.trufflesuite.com/docs/ganache/overview.