Universidade de Aveiro
2023

Daniel Baptista Andrade

Transações de Energia baseadas em Distributed Ledger Technologies para Comunidades de Energia Renovável

Energy Transactions based on Distributed Ledger Technologies for Renewable Energy Communities

Universidade de Aveiro
2023

Daniel Baptista Andrade

Transações de Energia baseadas em Distributed Ledger Technologies para Comunidades de Energia Renovável

Energy Transactions based on Distributed Ledger Technologies for Renewable Energy Communities

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Cibersegurança, realizada sob a orientação científica do Doutor Paulo Jorge de Campos Bartolomeu, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, do Doutor Joaquim José de Castro Ferreira, Professor Coordenador com Agregação da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro, e com a colaboração do Doutor João Miguel Pereira de Almeida, Investigador do Instituto de Telecomunicações.

Este trabalho é financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER), através do Programa Operacional Regional de Lisboa (POR LISBOA 2020) e do Programa Operacional Regional do Centro (CENTRO 2020) do Portugal 2020 [Projeto COMSOLVE com o nº 047019 (CENTRO-01-0247-FEDER-047019)].

Dedico este trabalho aos meus pais, Fernando Andrade e Fernanda Andrade, que me acompanharam ao longo de toda a minha formação, ajudando-me prontamente a ultrapassar quaisquer dificuldades que me fossem surgindo pelo caminho.

agradecimentos / acknowledgements

Agradeço toda a ajuda prestada pelos meus amigos, assim como aos docentes que me acompanharam ao longo de todo este processo.

Palavras Chave

Distributed Ledger Technology, Comunidade de Energia Renovável, Criptomoeda, Descentralização

Resumo

O surgimento de comunidades de energia renováveis em combinação com a transição de combustíveis fósseis para fontes sustentáveis de energia potenciou o surgimento de um novo paradigma no qual a energia é trocada de forma descentralizada entre membros da comunidade, os chamados prosumidores. Aliado a esta ideia de descentralização, surge também o conceito de *Distributed Ledger Technology*, uma tecnologia emergente, cujo propósito é descentralizar a informação por uma rede de dispositivos, permitindo assim verificar a veracidade do seu conteúdo. Esta tecnologia, além de armazenar informação, permite também trocas de moeda digital, designadas por criptomoedas. Este trabalho propõe aplicar estes novos conceitos a uma comunidade de energia renovável, através de transações diretas ou por via de *smart contracts*, entre os membros de uma comunidade que vendam o excesso de produção energética, sem a necessidade de intervenção por parte de um operador de rede ou de uma entidade bancária. Pretende-se que, recorrendo a Distributed Ledger Technologies, seja possível criar um mecanismo capaz de criar um histórico de acontecimentos dentro da comunidade, ao mesmo tempo que se habilitem pagamentos entre os seus membros a preços mais competitivos que os do mercado tradicional.

Keywords Distributed Ledger Technology, Renewable Energy Community, Cryptocurrency, Decentralization.

Abstract

The emergence of renewable energy communities, in combination with the transition from fossil fuels to sustainable energy sources, has empowered the emergence of a new paradigm in which energy is exchanged in a decentralized manner among community members, known as prosumers. Alongside this idea of decentralization, the concept of *Distributed Ledger Technology*, an emerging technology designed to decentralize information across a network of devices that allows for verifying the truthfulness of its content, also arises. In addition to storing information, this technology also enables digital currency exchanges, known as cryptocurrencies. This work proposes to apply these new concepts to a renewable energy community through direct transactions or via *smart contracts* among community members who sell excess energy production, without the need for intervention from a network operator or a banking entity. It is intended that by using Distributed Ledger Technologies, a mechanism capable of creating a history of events within the community can be established while enabling payments among its members at more competitive prices than those in the traditional market.

Contents

List of Figures

List of Tables

Glossary

DAG	Directed Acyclic Graph
DLT	Distributed Ledger Technology
P2P	Peer-to-Peer
PoW	Proof-of-Work
PoS	Proof-of-Stake
PoB	Proof-of-Burn
PoA	Proof-of-Authority
BFT	Byzantine Fault Tolerance
EV	Electric Vehicle
REC	Renewable Energy Community
SP	Solar Panel
USDT	Tether
XAUT	Tether Gold
TPS	Transaction per Second
EVM	Ethereum Virtual Machine
IBC	Inter-Blockchain Communication Protocol
dApps	Decentralized Applications
GDPR	General Data Protection Regulation
PoC	Proof-of-Cooperation
PYLNT	Pylon Network
TOTP	Time-Based One-Time Password
API	Application Programming Interface
GDPR	General Data Protection Regulation
RPC	Remote Procedure Call
HTTP	Hypertext Transfer Protocol
SSL	Secure Sockets Layer

CHAPTER 1

Introduction

The Energy Systems model has undergone significant changes, coming from a hierarchical, top-down model in which large companies were responsible for producing all the energy needed to maintain the public grid and supply their clients, to a more decentralized model focused on the consumer. Consumers are now also responsible for producing, storing, and selling their own energy, primarily from renewable sources such as Solar Panels (SPs) or wind turbines. This type of consumer is called “prosumer”, since it no longer consumes energy from the public grid, but also injects its surplus into the grid. This new approach to energy production and consumption requires new methods and structures for electricity markets, which are currently undergoing major changes to accommodate these new possibilities.

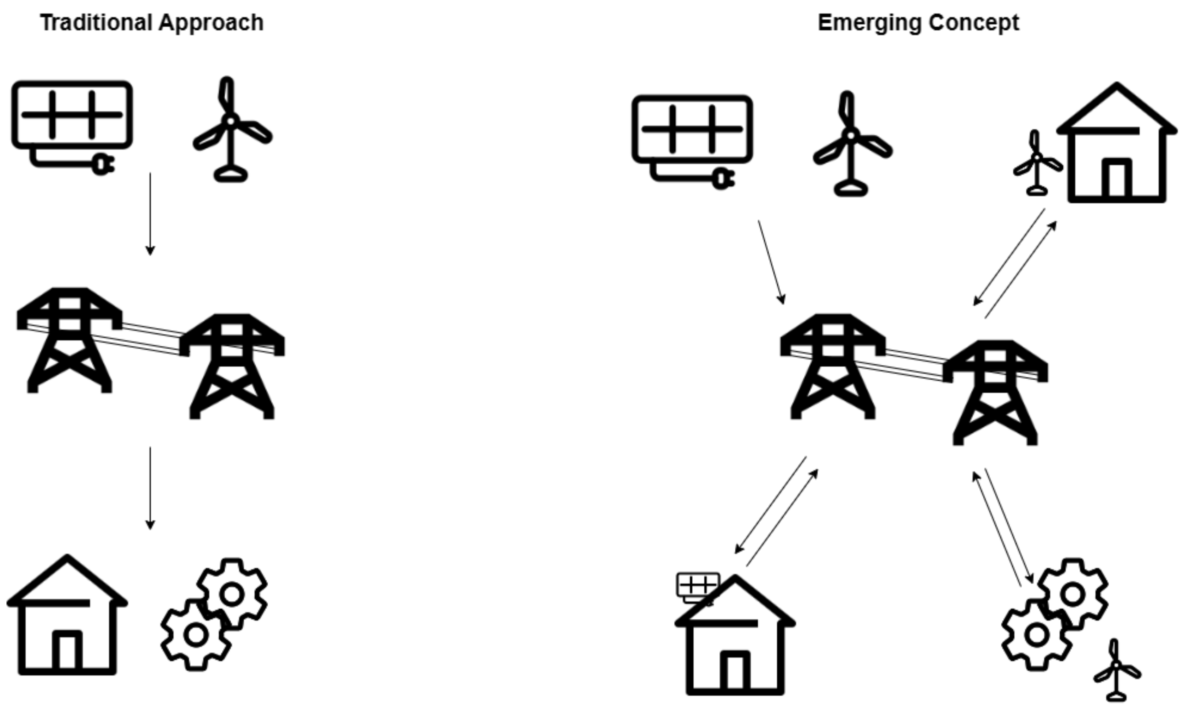


Figure 1.1: Energy model.

The public grid, which is responsible for transporting electricity, is no longer used for one-way transportation, but rather operates as an intermediary platform between multiple delivery and exit points, even when a single peer may have both behaviors. This model is recent when compared to the oldest one and it arises due to concerns about the sustainability of our planet. With the increase of carbon dioxide emissions and other greenhouse gases to the atmosphere, it was necessary to think about a viable alternative to fossil fuels, the principal form of energy production, but also the principal cause of gas pollution on Earth.

Addressing this long-term problem, large producer companies started to adopt sustainable energy sources like the use of hydroelectric dams, wind turbines, and SPs. With this, it became visible that a new form of energy production on a smaller scale was also possible, fostering the concept of the prosumer to fill that gap. However, a smaller-scale energy market solution was also needed, making Renewable Energy Communities (RECs) the first time to be talked about. A REC is a group of consumers and prosumers, either a small neighborhood or an entire region, that executes some sort of "micro energy exchanges" among each other.

The second highest contributor to gas emissions is fossil fuel-powered vehicles. Being aware of this, car companies have begun to shift their focus from developing polluting cars to adopting new techniques such as the use of batteries to power their creations, making Electric Vehicles (EVs) a new, innovative, and more friendly option to the environment.

Hereupon, it is necessary to develop solutions for managing RECs with EVs integration and energy storage systems based on end-of-life batteries. By doing so, the consumption of fossil fuels will gradually be reduced in favor of more sustainable options, and using batteries that no longer have their full capacity for their conceptualized objective can retain surplus energy for later consumption without being immediately considered waste. Also, EVs can be used with this same goal of storing or injecting energy into the public grid according to the needs of the community.

1.1 Contextualization

Given the introductory explanation, it is necessary to develop a mechanism that can accommodate these new rising opportunities and stakeholders, while at the same time being reliable in its form of operation. In this community ecosystem, members may issue buy and sell transactions directly between themselves, in a secure and distributed form. These transactions are going to make use of consumption patterns, such as buy and sell prices as well as the energy source, and preferences of its various users.

The management entity of the REC also has the role of optimizing the energy balance of the community by evaluating, in real-time, the energy buy and sell price from the public grid, and making decisions regarding its storage, buy for future consumption or even the sell the energy surplus. Although the management entity could be administrated manually by a group of individuals who are responsible for the community behavior, it is also possible to develop mechanisms that make decisions automatically with little to no supervision. To achieve it, a reliable option is the usage of Distributed Ledger Technologies (DLTs), a powerful technology that could be responsible for a completely decentralized automation of decisions and a convenient way of issuing payments for micro energy exchanges. With the use of a DLT, this document aims to bring several benefits, namely:

- The coordination of the various devices to perform critical decisions in terms of energy costs. This includes transactions in real-time, device updates, and analysis of when and who should the peer buy or sell its energy to;
- The immutability of recorded monetary and energy transactions. DLTs can be seen as a reliable source of information (e.g., a database) with the particularity being publicly available to everyone that participates in that network, while maintaining user confidentiality and anonymity by using hashes and obfuscation mechanisms. A key aspect accounted for is the compliance with the General Data Protection Regulation (GDPR) rules implemented in the target country, Portugal;
- Hindering malicious behavior from stakeholders. By using a DLT, the data stored in the ledger is tamperproof and, since a large quantity of nonbiased network peers host the same information, it is unfeasible to modify its content without notice.

1.2 Motivation

DLTs are an emerging topic and their usage in financial environments shows that they can provide a way of securing and guaranteeing data anonymity and immutability without the need for third-party infrastructure to validate data authenticity. A DLT can be seen as a group of devices connected via an untrusted network where each one contains a partial or integral copy of the database with records of every past action, as well as a program where they manage their actions in that network [1]. Not only these networks could provide all of the above, but in this specific scenario, there is a lot more this technology can offer.

Nowadays, energy management across the energy grid suffers from energy losses and malfunctions either due to commercial mistakes or technical faults [2], [3], [4].

When considering using DLTs on a project involving microtransactions, such as the implementation of a REC, the taxes associated with the operation of exchanging small quantities of money play an important role in defining the viability of the solution. This is also something that must taken into account by trying to reduce this cost as much as possible in order to not be impactful on the involved peers' savings.

This document aims to describe a proof of concept for a DLT implementation. The implementation targets decentralized Peer-to-Peer (P2P) energy transactions in a REC. The goal is to use a DLT as a reliable source for implementing new energy model concepts, being those on small or bigger scales. Additionally, the implementation aims to provide an efficient method to bypass energy grid faults and malfunctions [5].

1.3 Objectives

The present research is a component of a major project, called COMSOLVE, whose objective is to create a REC and management platforms that enable energy surplus exchanges between peers. To do so, the objective was to attach devices to the electricity meters of the residences of the community members that were capable of communicating on the network for sending measurements in real-time and receiving due payments related to the energy consumed that came from the surplus of another community member. These payments are then issued using a DLT alternative and those records should be stored on the REC management system as well as the DLT for auditability by regulatory agencies.

The document focuses on the DLT part of the solution, which are:

- The implementation of an automatic mechanism of processing payments between two community peers.
- The implementation of energy exchange forms and payments between more than two peers.
- The implementation mechanisms of transactions obfuscation.
- Allowing community information stored on a DLT to be auditable by regulatory agencies.
- Processing payments from different periods of the day with minimal to no loss on money conversion.
- Using a DLT to store useful additional data despite transactions.

These objectives assume that some aspects, such as peer matching and buy and sell price selection have already been addressed since it is not part of the current plan for this dissertation.

1.4 Document Organization

This document is organized into six chapters.

The first chapter, Introduction, provides an overview of the problem, the question that the dissertation tries to clarify, the contribution that it can provide to the area, and the objectives of the present study.

The second chapter, entitled “Fundamental Concepts and State-of-the-Art”, gives an explanation of relevant concepts needed to understand the document while providing technologies and related projects where these concepts are implemented. It is divided into three main sections: “Distributed Ledger Technologies”, describing relevant technologies for the study; “Renewable Energy Communities”, where it is explained briefly how they operate and possible structures of RECs; and “DLT Applications in Energy Systems”, projects and other related implementations where the DLT usage was one of the core aspects of those solutions.

Architecture, the third chapter, has the objective of elaborating the details accounted for before the implementation itself, describing the needs of every community component as well as the overview concept of the project where this document is inserted into.

The Implementation chapter, which is the fourth chapter, gives an elaborated insight on every component developed, by describing every technology used in it and how and why some implementation decisions have been made the way they were.

Next, the fifth chapter entitled “Results”, compares every exchange alternative implemented from the previous part by using network and DLT usage costs metrics. It also has a section where these alternatives are put in discussion with the goal of finding the most reasonable alternative, based on the results obtained.

Finally, the conclusion chapter summarizes the results obtained and aims to show how the achieved on the document could contribute in diverse aspects, being technological advances as well as academic and professional contributions. It is also in this chapter that the document explains some of the limitations that happened throughout the elaboration of the application and gives some foundation for future work.

CHAPTER 2

Fundamental Concepts and State-of-the-Art

This section presents the technologies developed or in development that were considered relevant to elaborate this document alongside a brief description. The first section entitled Distributed Ledger Technologies has the goal of elucidating to the reader the main aspects of a DLT, encompassing the different architectures, the mechanisms of reaching a consensus, and giving an overview of different DLT built-in functionalities. The following section aims at consolidating some architectural aspects of a REC and describes key related terms. Finally, the last section presents a review of related projects where DLTs play a central role in the energy transactions.

2.1 Distributed Ledger Technologies

The term “distributed ledger” goes back to the Roman Empire. The objective of a distributed ledger is to bypass a problem known as “Byzantine Generals’ Problem” [6]. This problem arises when separated armies commanded by different generals around an enemy city need to execute a synchronous and simultaneous attack. Otherwise, they will lose the battle and get killed. Among them could be one or more general traitors who may try to confuse and corrupt the coordinated attack. Therefore, the messengers should give a correct message to the other generals for them to reach an agreement. The final objective is to find an algorithm in which the loyal generals execute a successful attack.

This can be transposed to the problem that a DLTs tries to mitigate, which is to find a way of assuring its data is immutable and correct across the network.

DLTs are database infrastructures hosted across the Internet by either personal or corporate devices whose objective is to store and share information with other peers. Nevertheless, these devices, commonly called “nodes”, communicate with each other via P2P connections to update themselves and to guarantee the integrity of their data.

Although this jargon was recently popularized after the exponential growth of Bitcoin, initially theorized as a whitepaper by Satoshi Nakamoto, in 2008 [7], it was in 1991 that Stuart Haber and Scott Stornetta wrote a paper about tamper-proofing a digital document [8]. Later, another paper was published trying to debate and solve the problem of building a trusted system across untrusted servers [9].

Nowadays, there are two main data structures for DLT implementation, Blockchain and Directed Acyclic Graph (DAG), which can be permissioned or permissionless. In the former case, access to the network is restricted to a number of actors. In the latter, there are no rules on who can access the data structure.

Before introducing these topics more in-depth, it is necessary to clarify how the "Byzantine Generals' Problem" [6] was solved.

2.1.1 Consensus Algorithm

A consensus algorithm is intrinsic to all types of DLTs since its objective is to reach an agreement on how to proceed given a certain objective. However, there are multiple ways of achieving that validation and, because of that, the objective reaching form has an impact on various aspects of the DLT itself, from the approval speed of the block (group of transactions on a blockchain) on the network to its carbon footprint. Provided that the idea is to integrate a DLT as a complementary mechanism of a Renewable Energy Community (REC), it is necessary to search for the most used algorithms and study their viability in this type of implementation.

Practical Byzantine Fault Tolerance

Byzantine Fault Tolerance (BFT) is a practical Byzantine state replica that can work even with malicious nodes in the system. In this model, all nodes are ordered sequentially, with one being the primary (leader) and the others being secondary (backup) nodes. For the model to work, the number of malicious nodes must be less than one-third of the total number of nodes. The model works on the basis that the client sends a request to the primary node and, after that, the primary node spreads it to the secondary nodes, which execute the request and send the client a response. The client waits for responses and the end result is that all non-malicious nodes agree to accept or reject the order of the record. The majority of the secondary nodes can decide if the primary node is malicious and, if so, remove it [10].

Proof-Of-Work

Proof-of-Work (PoW) is the first used type of consensus algorithm, being the one implemented in the Bitcoin blockchain. The idea is to find a hash that meets a specific rule. In the case of Bitcoin, this "rule" is achieving an SHA-256 hash that has a certain quantity of zero bits in it [7]. To accomplish this, nodes make use of the Nonce parameter, a variable used to reach that rule, and increment it until the final value fulfills the desired condition. The next step is to broadcast the block to the network. In this phase, every node linked counts as a unique voter aiming to approve or reject the block. If the majority of the nodes reach an agreement, then the block is appended to the chain. Finally, there is a monetary reward for the node that mined the block. Although this seemed an easy and effective approach to the problem, when it comes to the amount of energy spent in the computing process, concerns began to emerge about the sustainability of a worldwide blockchain using this method. Two of the most known blockchains in the world that make use of PoW, Bitcoin and Ethereum consumed, at their peak, an estimated value of 204.5 TWh and 93.97 TWh in May 2022 [11] [12], respectively. Comparing those numbers with the highest peak of annual energy consumption in Portugal, 50.505 TWh in 2010 [13], a single month of these blockchain carbon footprints can almost double, in Ethereum's case, and even quadruplicate the carbon footprint of an entire nation in a year. These values are not tolerable if aiming to use blockchain as a daily-based technology. When talking about Ethereum, its operation no longer resorts to the use of PoW. Instead, since 2022 [14] it uses Proof-of-Stake (PoS).

Proof-of-Stake

The main alternative to the previous algorithm is PoS. PoS came to play after the concern about the sustainability and the environmental issues that the previous mechanism could originate. The idea is that each party, someone who is willing to mine the block, stakes an amount of money in the blockchain. Next, for each block, there is a leader election in which the party elected gets to mine the next block. So, the more money is at stake, the higher the probability of an election. With this, the mining process is focused only on a small group of nodes instead of all the network trying to achieve a verified block. This substantially reduces the amount of energy consumed per block approved while, at the same time, rewarding the block issuers like PoW [15].

Proof-of-Burn

Proof-of-Burn (PoB) consists in destroying or “burning” a volume of cryptocurrency that a miner has in its possession to have the right to mine the next block in the chain. In this case, since nothing in the blockchain can be erased, the idea is to have an address to where the user sends an amount which irrevocably “stuck” and, therefore, can be considered destroyed because no one has access to this wallet and this money can not be transferred in the future [16]. Since a person burns an amount of money to have the power to mine a block, the computational power needed to mine is reduced to a single miner, and only the verification is shared across the network by other peers. Compared to the PoW which tends to have a large network of devices mining each block, the idea is to change from using enormous amounts of consumed energy by those devices into a more virtual approach of “destroying virtual equipment”, the cryptocurrency of the blockchain in question, that can also be equally expensive, without compromising indirect sources, like the environment [17].

Proof of Authority

Proof-of-Authority (PoA) is a consensus algorithm mostly found among “permissioned” blockchains, due to its nature. In networks using PoA, the authority to validate and verify transactions is given to a group of nodes, typically called “authorization nodes”. Therefore, all the other members of the chain trust these nodes, breaking the idea of other types of algorithms that operate above “untrusted networks” [10]. This is a more centralized approach when compared to other types of consensus. Hence, nodes must be known inside the network. In case of misbehavior, malicious nodes can be instantly removed. This method was designed to be more efficient and secure than other decentralized options because all the critical part of the process, the block creation, is made on already verified entities. This type of algorithm is found essentially on enterprise blockchains because they can assure the trust factor of these authority nodes [18].

Overview

This subsection serves as a comparison and wrap-up of the discussed algorithms showing their key advantages and disadvantages. It is important to note that this comparison uniquely features widely used algorithms. To perform an adequate evaluation, some key performance indicators are necessary. Throughput, to compare the number of transactions issued per second; Time until approval, a metric to understand how much time is required until the transactions are issued; Fairness, an indicator concerning the mining fairness of the DLT [19]; Sustainability, to have a perception on how the algorithm has environmental consequences and Scalability, a form of evaluating on how the network reacts when put on “stress”.

	Throughput	Time until approval	Fairness	Sustainability	Scalability
PoW	3-5Tx/s	High	High	Low	Low
PoS	875Tx/s	Medium	Medium	High	Medium
PoB	Low	High	Medium	Low	Low
PoA	N/A	N/A	N/A	N/A	N/A

Table 2.1: Comparison between different consensus algorithms.

In Table 2.1, the characteristics were rated using “Low”, “Medium”, and “High” as qualitative terms since, in some cases, it is only possible to speculate on how the algorithm operates for a given parameter. If any quantitative evidence is found, its quantitative value is shown, as seen in the throughput column in PoW and PoS cells [20]. For parameters that it was not possible to find or infer a qualitative or quantitative value, a “N/A” was defined [21], [22], [23]. Since the target DLT must issue transactions in a short period of time, we focus on algorithms where the time of approval between transactions is short. Note that the peers that are going to join a DLT based REC could have little to no knowledge of how this system works. Hence, delays between transaction issuing and its approval may create alarm.

Another important factor is the currency exchange. If the payment values were calculated using a currency like the Euro, the delay elapsing between the cryptocurrency conversion and the corresponding transaction approval must be as short as possible to prevent overpayments or underpayments. The suitable candidate for this parameter is the PoS algorithm, due to its lowest “time until approved”. Sustainability is another important factor. It is contradictory to implement an energy-intensive consensus algorithm on a REC whose purpose is to increase efficiency and reduce the carbon footprint. Again the PoS seems to be the the best option. The Fairness option is only relevant if talking about small-scale DLT systems. Since the document aims to opt for a large DLT, the fairness characteristic is not too relevant, because the chosen DLT must have a large network of peers.

Scalability is another metric that must be well-evaluated. In terms of connected devices to the network, they are a small number. However, if multiple large regions implement a REC mechanism using the same DLT alternative, scalability can be conditioned either by large verification times or higher fees.

A key metric is throughput, the amount of transactions per second. The highest number is always preferred since, to guarantee anonymity and obfuscate transaction patterns, bigger transactions could be divided into small ones at different periods of the day. Even in this metric, PoS achieves the highest score. With this, when talking about common consensus algorithms, PoS seems to be the most reliable option. Consequently, derivatives of this method can also be a potential alternative.

2.1.2 Blockchain

The Blockchain model is the most common form of a DLT. For this reason, most of the time, DLTs are simply referred to as "blockchains". Its structure was defined in the Bitcoin whitepaper [7] by Satoshi Nakamoto. The structure of a blockchain encompasses three main components: Transactions, Blocks, and Nodes. Transactions are the atomic part of the chain. Since the objective of a blockchain is to host a decentralized database across the network that stores money transfers, a transaction is a record of that process and is composed of a reference of the wallet where the money comes from, the number of coins transferred, and the destination wallet, that are hashed like Figure 2.1 suggest. This is the core idea of how it should work, despite the different code implementations of this structure.

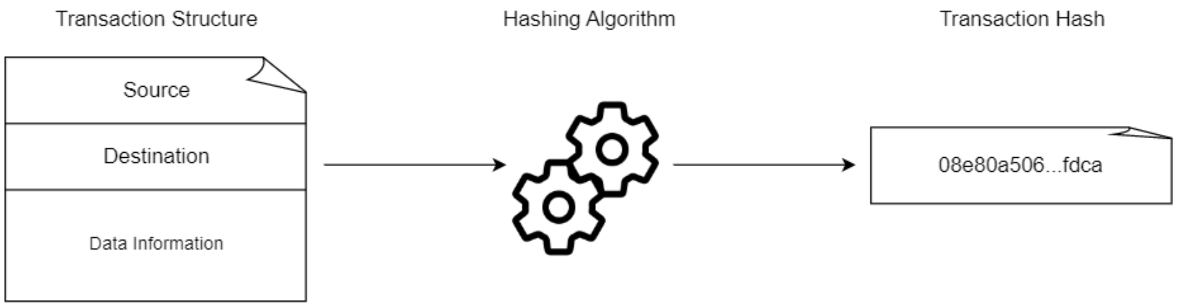


Figure 2.1: Transaction structure and hash.

This transaction is then hashed and, alongside other transactions that are waiting to be verified and inserted into the blockchain, forming a block. A Block consists of a Block Header and the group of transactions that are part of it. The block header has the hash of the previous block of the chain, Previous Hash; a Root Hash, that results from hashing the transactions in a Merkle Tree [24], and a Nonce, a value that is going to be found via a consensus algorithm, a topic explained later, to make the block verifiable and valid according to a specific rule, as shown on Figure 2.2.

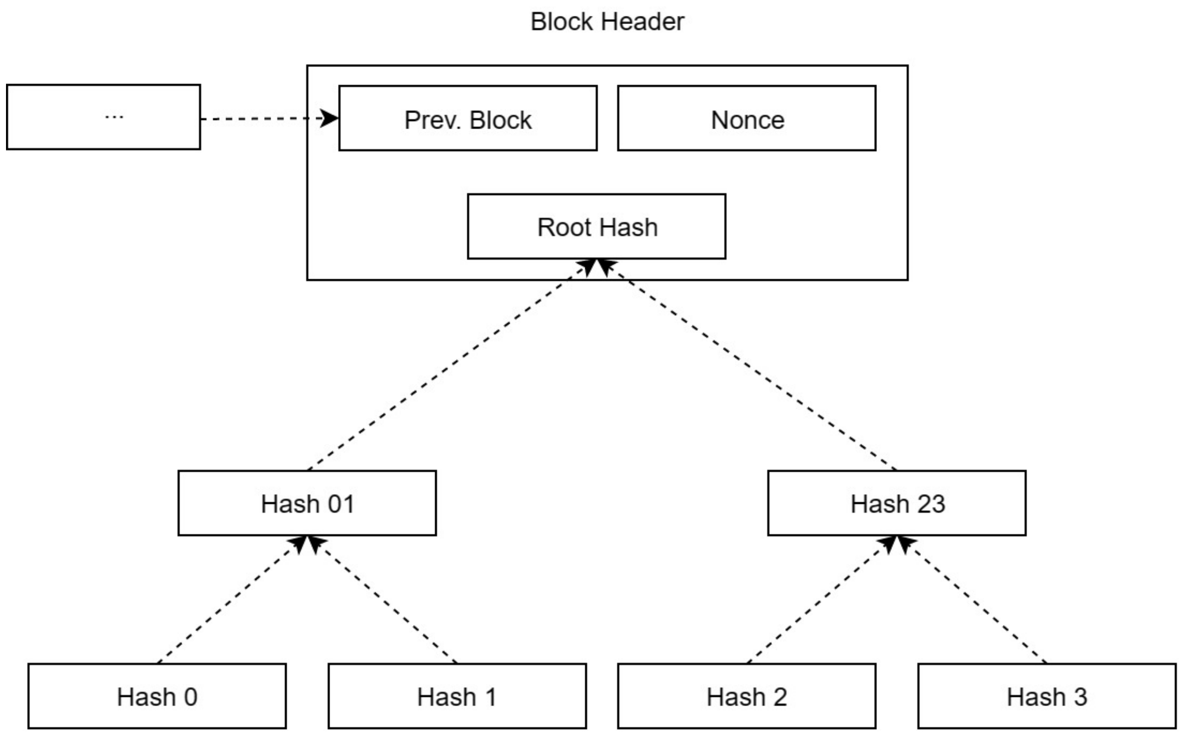


Figure 2.2: Transactions hashed in a Merkle Tree and stored in Block Header.

A block is broadcasted to the network to be verified by other peers that are executing the same mechanism, the Nodes. Nodes are devices across the network whose objective is to hash the blocks, verify pending ones, and insert them into the blockchain. Figure 2.3 represents that process involving four nodes and showing the acceptance of the block by the majority of them. Another particularity of a blockchain is that there are no ramifications of the chain. The valid one is always the longest chain of blocks that exists. If at an exact moment, there are two or more valid blocks, they stay in a pending state until a more recent and valid block is "chained" to one of those pending blocks. When this happens, the other blocks are discarded and the valid blockchain becomes the one with the longest chain.

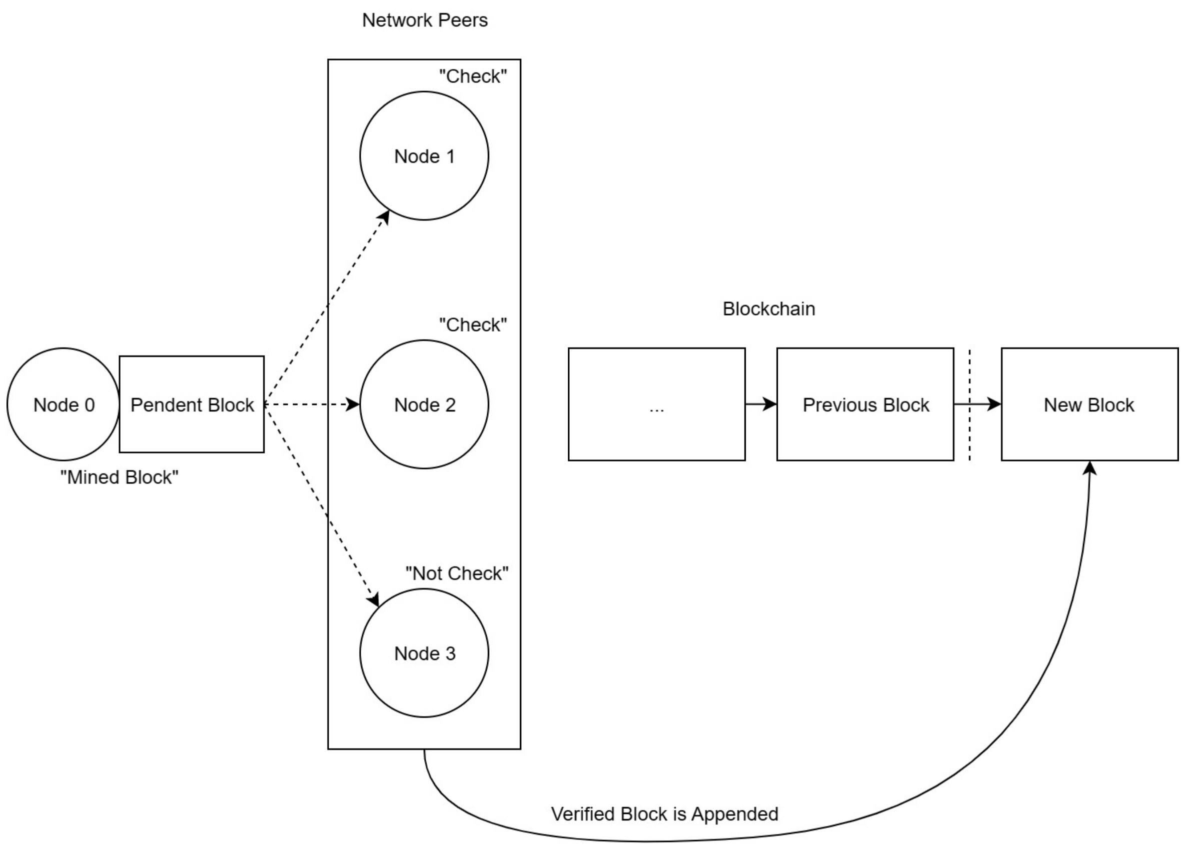


Figure 2.3: Process of accepting and recently mined block to the chain.

Provided their relevance and broad market adoption, some blockchain technologies are described in the following text. Although in the previous chapter2 some concepts have already been explained, we opted for addressing additional technologies here to make the document easier to follow.

Bitcoin

As mentioned before, Bitcoin was the pioneer DLT, and its first official document dates from 2008, written by "Satoshi Nakamoto". As a matter of curiosity, its author remains a mystery and till today nobody really knows who he/she is or who they are. Bitcoin came as an alternative form of issuing transactions from two different actors without having to resort to financial institutions while using an untrusted network to achieve it. Moving on to technical terms, each peer, an actor in this system, possesses a wallet where the funds are stored. Each wallet exists by generating three parameters: the public key, responsible for verifying a transaction's authenticity by other peers; a Bitcoin address, a digital signature derived from the public key and finally the private key, a personal key responsible for signing transactions. For security reasons, the private key cannot be inferred from the public key, but a public key can verify a transaction signed by the private key. The Bitcoin address can be seen as the "account owner's name", the public key as an "account identifier" and the private key as the "CVV", when compared to a credit card. Essentially, every other aspect of

Bitcoin implementation relies on the general blockchain architecture exposed above, 2.1.2, due to Bitcoin being the first and the foundation for other blockchain modifications [7].

Ethereum

Ethereum is the second most known permissionless blockchain in the world and it was first enunciated in 2014 in a whitepaper written by Vitalik Buterin [25]. Ethereum emerges in the blockchain world as an improvement of Bitcoin, by introducing the concept of smart contract, a piece of code responsible for issuing some action, and Decentralized Applications (dApps), applications that have as foundation code residing inside a blockchain. In terms of accounts, Ethereum has two types: external accounts, the ones controlled by Ethereum users, and contract accounts, accounts that are generated by smart contracts and are activated every time that piece of code is executed. Another innovation brought by Ethereum is the use of "Messages". Messages have a similar structure as transactions have, with the particularities of being able to be created by contracts. They can contain data and have the possibility of returning a response from the receiver of that specific message [25]. This blockchain has also introduced the token concept, a way of creating subcurrencies and associating their value to any desired asset, as explained in a later section. With this, its use not only contemplates financial applications, like Bitcoin was supposed to do, but also provides alternatives to this usage, like applications where money transactions are involved. Another important aspect of Ethereum is its consensus algorithm. Although Vitalik wrote in its document [25] that Ethereum uses PoW as the mechanism of validating transactions in the chain, nowadays it is not true anymore. Since September 15, 2022, Ethereum changed from being PoW blockchain to a PoS one. It is also claimed on their online page that the energy consumption of the blockchain was reduced by almost 99.95%. The process was called "The Merge", because, before changing the algorithm, a new blockchain was created, the Beacon Chain, to serve as a comparison and to see if it was beneficial to make these transactions due to concerns about scalability, security, and sustainability.

Ethereum Beacon Chain

Launched on December 1, 2020, the Beacon Chain was the name given to the blockchain using PoS. Since Ethereum could not go through critical changes without a huge foundation of evidence that the update was viable, Beacon Chain was created and ran alongside Ethereum to serve as a comparison and test instrument to major changes that were upcoming, the change from PoW to PoS. A notable difference between this chain and Ethereum was that by using PoS, the chain became even more secure than its current counterpart. In this case, miners are called "Validators" and, by staking a quantity of ETH, they are able to validate the transactions. To ensure these validators are trusted if they misbehave, the validator loses part of its staked amount. The Beacon Chain also served as a setup for a future scalability improvement called "sharding" [26]. This concept should come in 2023 and consists of separating the database in order to spread the load between those small blockchains. Since there is no real usage of this concept yet, its possibilities are only theoretical and any progress of this concept could be evaluated in the near future [27]. As stated previously, the current state of the Beacon Chain was merged with the Mainnet of Ethereum to make it the Ethereum blockchain known today [28].

Binance SmartChain

Binance, an online cryptocurrency exchange, also enrolled in the search for a desired blockchain and developed their own, Binance SmartChain, also called BNB Chain. According to their whitepaper [29], its design should have some particularities. One of them is the execution times should be shorter than the Ethereum network and its consensus algorithm is something called Proof of Staked Authority. This proposed algorithm combines Delegated PoS, a variant in which cryptoasset holders choose a group of nodes to make decisions on their behalf [30], and PoA, achieving three principal goals, which are: the fact that a block is produced by a limited set of validators; validators rotate to produce blocks like PoA; and those validators are chosen according to their stake. BNB Chain developers designed this chain to be compatible with Ethereum. Therefore, deployment of smart contracts is another feature. To operate, the BNB Chain has a native token called BNB. It also supports Cross-Chain Communication, a way of communicating between blockchains. This type of transfer is only possible between their blockchains, because, like Ethereum, they possess a Beacon Chain, and this "SmartChain" emerges due to the community necessity of using new features such as the smart contracts.

2.1.3 Directed Acyclic Graph

DAGs are another way of implementing a DLT [31]. In this case, instead of "blocks" the edges of the graph are called "nodes" and each node is composed of a single transaction [32]. Be aware not to confuse the node designation from the blockchain terminology with the dag one. Deconstructing the terminology, "Directed" means that all the links of the graph link earlier transactions to later ones, "Acyclic" means that there are no loops in this type of graph. So, an earlier transaction could not be pointed by a later transaction that itself is pointing to. Finally, the "graph" indicates a data structure represented by nodes linked with each other following a specific topology. Since this is an abstract concept, it is used as a foundation for specific DAG implementations by enterprises whose goal is to develop a DLT for their own purposes. That is the case, for example, of IOTA's Tangle [33] and Hashgraph by Hedera [34], topics described and compared in the following subsections. A representation of a typical DAG is shown in Figure 2.4, where the vertices are shown as circles and link to newer vertices.

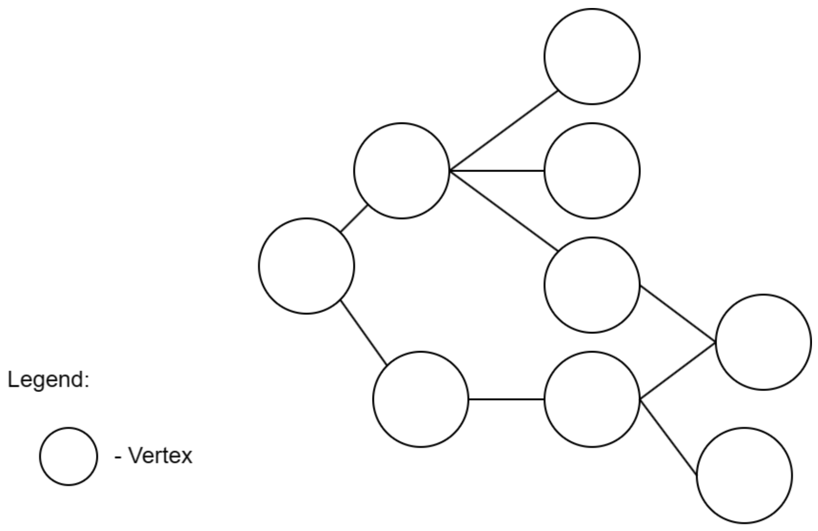


Figure 2.4: DAG construction example.

Tangle IOTA

Instead of having a chain of blocks, Tangle generates a DAG. In this graph, a vertex can point to multiple vertices and vice versa. Looking at Figure 2.4, the first vertex on the left is pointing to two newer vertices, while, on the right side of the figure, two older vertices are pointing to a new one. This linking occurs because Tangle is based on a primary rule: in order to validate a new transaction (called a “tip”), two older transactions must be validated first. This rule also ensures that the transaction log is always complete. As a result, the scalability of the Tangle network depends on the number of active users [35]. IOTA also uses a PoW algorithm similar to the one used by Bitcoin, but it has been slightly modified to use less energy and take less time to validate a “tip.” In the early stages of IOTA, there were not enough diverse miners to prevent 51% attacks, so the IOTA foundation created a node called the “Coordinator”. This node is considered a trusted node, and part of the validation process relies on it. However, IOTA has stated that it plans to remove the “Coordinator” in future versions once a sufficient number of diverse nodes has been reached. One of the main applications of IOTA is microtransactions, as there are no fees applied to them [33]. It is also worth noting that the implementation of Tangle is open source.

Hedera Hashgraph

Hashgraph is a DLT that uses a unique and complex algorithm to achieve consensus on the order of events in the network. Differing from other DLTs, the initial implementation of Hashgraph was developed by a private company, Swirlds, but they have since released an open-source version of the software. Hedera Hashgraph is a public network built on top of the Hashgraph consensus algorithm, but it’s not the only one. Other organizations and companies can use it. In Hashgraph, each event is known by all participants, but the information about the event is cryptographically secure and tamper-proof. The process of information exchange in Hashgraph is achieved through a "gossip about gossip" protocol [36], where each node sends information about new events to a small number of other nodes, which then repeat the process, spreading the information throughout the network. This process allows the network to efficiently come to an agreement on the order of events and detect and resolve conflicts, without the use of explicit voting [34]. According to the online page of the technology, the Transaction per Second (TPS) is substantially higher when compared to alternatives like Bitcoin and Ethereum, and its confirmation time is very reduced. On top of that, the amount of energy consumed per transaction is residual when compared to those two blockchain alternatives, making it a viable solution [37].

Nano

In Nano, each account has its own blockchain, known as an "account-chain", which stores the account's balance and transaction history. This account chain can be thought of as a personal blockchain, and the nodes on the chain are represented by individual transactions. To make a successful transaction, the sender generates a "send" transaction and the receiver generates a "receive" transaction. When the send transaction is broadcast, the balance goes into a pending state until the receive transaction is received [38]. Nano uses a PoW algorithm to prevent flooding of transactions by attackers, and it has an Open Representative Voting system in place to resolve conflicts. To resolve conflicts, representatives, which are accounts that vote to resolve conflicts, are used. When a new account is created, it can associate with one of these representatives, and it can change its representative over time. These votes have more impact the more balance all the accounts that chose that representative have. Nano is highly scalable, as users are responsible for sorting their own transactions. Tests have shown that Nano can reach peak speeds of 306 TPS with an average of 105.75 TPS, although these values can be impacted by the hardware and network used by the user. Unlike some other DAG technologies, Nano does not have a leader election system for resolving conflicts. Instead, conflicts are delegated to a group of representatives, and the majority vote determines the outcome of the conflicted transaction [39].

2.1.4 Cryptoassets

It is important to distinguish between crypto coins and Tokens, although, most of the time, users call them just tokens. So, crypto coins or just coins are cryptocurrencies that are a primary cryptoasset associated with a blockchain. Inside the topic of coins, due to historical reasons, they subdivide into two categories: Bitcoin and Altcoins.

Bitcoin and Altcoins

Bitcoin was the first cryptocurrency that existed and came along with the Bitcoin Blockchain [7]. It has a special treatment due to the innovation it brought to this type of technology.

Altcoins are all the other coins that are intrinsic to other blockchains, like Ether for the Ethereum network, but came after the Bitcoin blockchain, either to correct some "flaws" that Bitcoin had, from the perspective of its developers, or to append new functionalities to a blockchain, such as smart contracts, a term proposed by Nick Szabo in 1990s and first implemented in Ethereum by Vitalik Buterin, as described in his Ethereum whitepaper [25].

Tokens and Stablecoins

On the other side, tokens are cryptoassets that do not own a blockchain but have a monetary value associated and coexist with each other. A Stablecoin is a cryptoasset that aims to mitigate the wide fluctuations of cryptocurrencies when comparing themselves to fiat currency, like the American dollar or Euro [40]. This fluctuation has proved dissuasive to users since these investors are not experienced enough in such a complex theme as cryptocurrencies. So, stablecoins are digital assets that, in order to mitigate this volatility, have their value attached to another asset, like fiat money. One great example of this approach is the Tether (USDT) whose value is equivalent to the American dollar. There are other approaches to this mechanism worth mentioning, such as, instead of having its value compared to a real currency, it can be issued against a physical asset, the most common, gold, therefore called "gold-backed cryptocurrencies". In these cases, there is some price oscillation. However, this price fluctuation is directly related to the physical asset which, in the real world, is also suffering some variance. Looking for one of the most popular gold-backed cryptocurrencies, on 15Th November, Tether Gold (XAUT) has its price rounding 1,760.54$ [41] while gold itself is about 1,769.20$ [42] at the moment of writing.

2.1.5 Smart Contracts

In P2P energy networks, due to DLTs, it is possible to establish smart contracts assuring that the financial transaction has been made. It is only possible because the code is immutable, therefore it cannot be erased nor modified. Because of this, smart contracts can be seen as a trustable form of information. A smart contract consists of a piece of code inside a DLT which is triggered automatically when some event occurs without the need for a centralized controller. The main objective of smart contracts is the reduction of third-party companies and human errors [43]. Typically, a smart contract is written in high-level languages, the most common one, Solidity, and finally compiled to bytecode. After that, it is deployed on the DLT. To communicate with this piece of code, a contact address is returned as part of the transaction receipt. Smart contracts can be used in Smart Grids to help distribute energy more efficiently than the traditional energy grids. These contracts can act as a way of automatically trading surplus energy between domestic users, they can also be used as a resource allocation manager allowing better energy distribution and reducing energy waste, or simply, be used to automatically process all the related transactions without the need of a third-party authority. Smart Contracts can also improve the auditability of a house's energy consumption due to blockchain transparency.

Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) is a virtual machine deployed inside the Ethereum blockchain where EVM code, which is bytecode at this stage, is deployed. Since the code hosted by EVMs is bytecode, there is no need for compilation processes, therefore the code is instantiated and not deployed and compiled every time someone wants to use that piece of data. [25]. It is necessary to state that, to run this code, it is necessary to have enough "Gas".

"Gas" is the term used by blockchain users when referring to taxes for performing certain contract actions. These taxes are applied when in-built smart contract functions require a state modification. On the other side, viewing public variables has no "Gas fees" applied [44]. There are alternatives to EVMs that rely on using a Docker container to deploy smart contracts, like the HyperLedger Fabric chaincode [45] [46].

Solidity

Diving into the most common high-level languages to deploy smart contracts, Solidity is the most used across all types of blockchain networks, due to a vast majority of them using EVMs in its core. Although Solidity is associated with Ethereum, the Ethereum developers used another language called Serpent [25] before migrating to Solidity. With this, Solidity was first developed by Gavin Wood in 2014 and later adopted by the Ethereum developers team and, only after this, it became the main language for written smart contracts. When talking about its functioning, Solidity is described as a language similar to JavaScript in its syntax, therefore, it is possible to create objects, it supports inheritance, and possesses many other functionalities like any other object-oriented programming language. A brief program is written below and it will be explained after for a better understanding of the language.

```
pragma solidity ^0.8.6;

contract Inbox{
    string public message;

    constructor(string memory intialMessage){
        message = intialMessage;
    }

    function setMessage(string memory newMessage) public {
        message = newMessage;
    }
}
```

Code 1: Code example written in Solidity.

First of all, there is a keyword called "pragma". This tag is used to specify which version of solidity the compiler should use. This part is useful because sometimes version updates implement features or adjust some minor aspects of the code and old pieces of code will not be compatible with the recent changes. So, the compiler, being aware of the version it needs to compile the code into, can make a correct interpretation and translate the high-level language into its bytecode equivalent. Next, it is possible to see the "contract" keyword. In solidity, this tag is equivalent to using "class" in JavaScript and it is used to encapsulate all the functions and variables related to "Inbox". The declaration of the class, or in this case, contract variables follow similar patterns of declaration, being the type of variable, its scope, public, private, or protected, and the variable name. The constructor follows the same approach as other object-oriented languages, but, it is possible to visualize a new type of keyword, "memory". "Memory" has another substitute which is "storage" and it is related to how the values should be stored. While "memory" can be seen as saving the data in RAM, “storage” is the same as stored in the hard drive. In terms of coding it could seem irrelevant, but using “storage” consumes a much higher gas fee than using “memory”. So, since in the code, “initialMessage” variable does not need to persist on the smart contract, there is no need to use the “storage” tag. It is relevant to remember that when changing any type of value inside a smart contract, a gas fee is applied, but, if a user accessing it wants to only see its content, there is no cost associated. Another particularity of this language is that the “getter” method is implicitly written without the need for its declaration. Just by declaring the variable “message”, it is accessible to everyone after the contract deployment and that is the reason behind its omission. Every line of code is delimited using a comma and every function and contract is written inside curly brackets.

DAML

DAML is a language that is not easily interpreted besides its representation. Although it could be seen as written in pseudocode or even similar to Python, DAML is a language that is independent of the smart contract language implemented in the DLTs. It is used in Canton, a framework detailed in another section, but, for context purposes, DAML is used to deploy smart contracts across different blockchains that were linked using the Canton framework.

```
module Iou where

template Iou
  with
    issuer : Party
    owner : Party
    amount : Int
    currency : Text
  where
    signatory issuer
    observer owner

    choice Transfer : ContractId Iou
      with newOwner : Party
      controller owner
      do create this with owner = newOwner

run = script do
    bank <- allocateParty "Bank"
    alice <- allocateParty "Alice"
    bob <- allocateParty "Bob"
    cid <- submit bank do
      createCmd Iou with
       issuer = bank
       owner = alice
       amount = 10
       currency = "USD"
    submit alice do
      exerciseCmd cid Transfer with
       newOwner = bob
```

Code 2: Code example written in DAML.

The keyword “module” has the same meaning as the “contract” or “class”, it creates the “Iou” module and all its content. Next, the “template” is equivalent to the constructor method of the class, and “with” declares its parameters: issuer, owner, amount, and currency. The amount and currency types are straightforward, but “Party” is DAML exclusive. It is a type related to associating the peer to the variable in order to execute transactions. Later, the tag “where” associates the variable issuer as being the signatory, the one creating the transaction, and the owner as the “observer” of the contract. Next, the script creates a sequence of transactions. First, the Bank notifies Alice that it is going to use “10 USD” of currency that Alice owns. Then, the Bank transfers that quantity to the “newOwner” which is Bob.

Common Languages

It is important to note that, despite the above languages being specific for smart contract development, there are libraries associated with all use languages such as Python, JavaScript, GoLang, and so on, that can also be helpful to write contracts. Every one of them has its own advantages and disadvantages when compared to each other, but, if allowed by the blockchain in use, they could serve the same purpose as Solidity and DAML.

2.1.6 Frameworks

This section has the objective of enlightening the reader about different frameworks that developers could use to carry on making a private DLT with its own inputs and settings, instead of using an already existing DLT that could be expensive when comparing to a personal solution. However, even with frameworks, achieving a “production state blockchain” could take years of work. So, it is important to understand if using an existent DLT could be a viable solution.

HyperLedger Project

Hosted by the Linux Foundation, Hyperledger is a set of open-source tools built by either government or individual developers to make possible the objective of building a permissioned network with a high level of security to mitigate existing weaknesses in public blockchains, like the number of TPS. According to them, Hyperledger Foundation aims to provide community-driven projects at the same time that they educate the common user about market opportunities for this type of technology [47]. Since Hyperledger Foundation develops frameworks like Hyperledger Fabric and Hyperledger Sawtooth, different approaches to create a blockchain network, a permissioned blockchain does not need to have the security concerns of running on an untrusted network, because all the nodes are known to each other. Therefore, these problems could be lightly approached while focusing on other important aspects. Being the most known framework of Hyperledger, Fabric was released in June 2016 and it has two main peer approaches: validating and non-validating peers. The first group of peers, the validating ones, are nodes responsible for maintaining the ledger by working like in any other blockchain, by validating transactions and running consensus algorithms. The non-validating peer is a type of node that serves as an intermediary for clients to connect to validating peers to issue transactions. Those nodes are not responsible for executing these transactions but have the possibility of verifying them. This framework also supports smart contract technology, which they call “chaincode” that uses Go as the high-level programming language to write them. Instead of using EVM, the chaincode runs inside Docker containers. Hyperledger Fabric has the particularity of using consensus algorithms in a pluggable way, which means that they can easily be replaced by another algorithm. Since consensus algorithms are actively being discussed by the community due to a variety of reasons, 2.1, Hyperledger Fabric opts to use a modular approach to test them and to open debate around it [48].

Canton

Canton is a distributed ledger interoperability protocol whose objective is to create a virtual ledger that can communicate with other blockchains or common storage services, like personal databases. To do that, Canton divides its topology into 3 main actors: party, participant, and domain. Domains are storage services like DLTs and databases; parties are the entities involved in all the processes, for example, normal users, banks, etc; and participants are nodes that host parties and expose 2 APIs, a Ledger API, where all the parties interact and execute their smart contracts which are going to execute procedures in the domains, and an Admin API, whose goal is to be accessed by administrators to manage the participant node. These smart contracts are written in the DAML language. However, developers also call the structure of DAML + Canton only by DAML. At the current state, DAML only supports interoperability between Postgres-based domain, Oracle-based domain, Hyperledger Fabric-based domain, Secure enclave-based domain, and Ethereum-based domain, although only in a demo phase yet [49].

Cosmos

Following a similar approach as Canton 2.1.6, Cosmos is a decentralized network to make, as they call it, an "Internet of Blockchains" [50] to solve the most common limitations of widely known DLTs. Scalability is one of the limitations to be solved, mainly due to low TPS. Usability is another limitation and Cosmos claims that to make EVMs an all-case possibility, sometimes it is necessary to decrease efficiency or code design to fit a desired application purpose, and the limited amount of programming languages supported can be discouraging, and finally, sovereignty.

To prove the last limitation, it is demonstrated that, if some bug occurs on the application of the blockchain, only the governance of it can make modifications. To surpass those problems, Cosmos implements the three layers required to build a blockchain. Networking and consensus layers are the responsibility of an open-source tool called Tendermint BFT, a tool that permits developers to abstract themselves from the core of the blockchain and focus on developing the application. Its engine communicates with the application via a socket protocol called Application Blockchain Interface, which can be used in any programming language. The application itself relies on using Cosmos SDK, a framework to create compatible blockchain applications with the Tendermint BFT. It has a modular approach, meaning that it is not necessary to write every program from scratch, because it is possible to import other existing modules to the project, making it possible to create more complex and customizable blockchain applications when compared to Ethereum, as an example [51].

Finally, this solution provides the possibility of connecting blockchains via a protocol called Inter-Blockchain Communication Protocol (IBC). This is achievable because each chain runs a light client of the other chains. If a money transfer has to occur between blockchains, the sender blockchain freezes the amount of money that is supposed to be transferred and sends proof of it to the receiver. This one verifies that proof validity and generates an equivalent amount of that token inside itself. A big advantage of this framework, when compared to the Canton one, is that it is possible to communicate between blockchains that do not rely on Tendermint to operate [50].

2.2 Renewable Energy Communities

RECs are communities that prioritize the use of renewable energy sources, either wind, solar, and/or hydroelectric power to reduce their dependency on fossil fuels. These communities may differ in size, ranging from a small neighborhood to a whole village. There are many ways that RECs can be structured and operated. Some may be organized around a central renewable energy facility, such as a wind farm or SP field, while others may rely on distributed production, where energy can be generated by each individual, commonly called "Prosumer", that belongs to the community. Some RECs may be self-contained and independent, while others may be integrated into the larger grid and sell excess energy back to the grid.

One of the main goals of RECs is to reduce greenhouse gas emissions and combat climate change by moving away from fossil fuel-based energy sources. In addition to environmental benefits, RECs can also provide economic benefits by creating jobs, reducing energy costs, and increasing energy security. Some of the challenges that RECs face include the high upfront costs of renewable energy technology, regulatory barriers, and the need for infrastructure and technical expertise. However, as the costs of renewable energy technology continue to decline and public support for renewable energy grows, more and more communities are looking to transition to renewable energy sources.

2.3 DLT Applications in Energy Systems

In this section, some projects with related concepts in this document are going to be described to serve as reference for later discussion.

2.3.1 PriWatt

PriWatt is a decentralized platform for buying and selling energy using smart contracts. It uses the Bitcoin blockchain and PoW as its consensus algorithm. All transactions are signed by the sender and verified by the receiver to ensure their authenticity. Smart contracts are used to prevent fraud and facilitate complex transactions through the use of multi-signature requirements. There are two main ways to trade energy on the PriWatt system. In the first method, the consumer pays for a set amount of energy upfront and may use or trade the energy once the deal is complete. In the second method, the consumer pays for energy on a per-kilowatt-hour (kWh) basis for a set period (usually 24 hours). At the end of the period, the buyer pays for the amount of energy received and may receive a refund if all the tokens are not used. If the buyer fails to respond, the seller can close the channel, receive payment, and send the completed transaction to the blockchain [52].

2.3.2 Brooklyn Microgrid

"Brooklyn MG" is a microgrid energy market in Brooklyn, New York. Due to severe weather conditions that tend to be more frequent in that zone, there have been grid faults and the electrical grid of the neighborhood is outdated. The project of Brooklyn Microgrid is an initiative that started in April 2016 and nowadays is one of the most significant implementations of blockchain regarding P2P decentralized energy trading and implementation of a local marketplace [53].

This first instance, the project registered the first-ever energy transaction using blockchain. This system is based on Ethereum smart contracts, due to its ability to facilitate renewable energy transactions between peers using a token-based transaction system. The surplus energy produced by SPs is converted to tokens by the smart meters installed in their houses and these tokens can be used to trade inside the energy market. The Brooklyn Microgrid can record the transactions either in energy units or tokens, depending on the user's requirements. Future improvements on the project consist of including the possibility to choose the required energy and the ability to buy a percentage of the energy from other users and the rest being filled by the public grid. To match the peers, an auction system takes place and the renewable energy will be sold to the highest bidder [54]. To interact with the platform, the project has a mobile application where users can do all the above.

2.3.3 Jouliette at De Ceuvel

In the Netherlands, Alliander is a group of companies related to energy activities. Alliander, in collaboration with Spectral Trading, another company aiming to reduce the impacts of climate change by investing in emissions-free energy supply, is developing a P2P energy sharing platform, Jouliette at De Ceuvel, which is implemented using a private and permissioned blockchain solution [10]. Joulliete is the token used to facilitate the transaction between peers. De Ceuvel is the community where the smart grid is implemented. A primary phase was launched to verify if sustainable urbanization was possible to achieve using a blockchain. Since the pilot project confirmed that blockchain technology could improve energy transactions, a second phase of this project took place in May 2018 and its main focus was to stimulate the use of the Jouliette token in daily activities inside the community [55].

2.3.4 OmegaGrid

By using a private blockchain ledger, OmegaGrid [10] can record communication, power delivery confirmations, and automated transactions all in one place. The uniqueness of this project consists of having a P2P energy platform focused on grid balancing. Being aware of the great use of energy by some DLTs projects, instead of using a PoW algorithm, they use the PoA, since it uses significantly less energy to validate transactions. A delay of 5 minutes per block validated is also claimed to be the average time until confirmation. The team state that, using a blockchain platform, prevents having only a single point of failure and helps change the actual grid design that they claim to be inefficient. Besides using a decentralized method, to set up a meter in an energy market, it is necessary to register the meter with whoever is responsible for the local grid.

2.3.5 Pylon Network

Another project that aims to improve the efficiency of energy trading is the Pylon Network. Pylon is developing a decentralized platform with the objective of trading energy between Renewable Energy Systems generators and consumers without third parties involved. It uses an internal token, which can be traded through the platform, as a reward for generating green energy [56]. The project was built using the Ethereum blockchain. At a later stage, a particular open-source blockchain was created. Therefore they have defined a consensus algorithm called Proof-of-Cooperation (PoC), which is not as intensive in energy consumption as PoW and, at the same time, allows to eliminate the competitiveness between miners and reduce the hardware costs [57]. For reference, PoC is a newer consensus algorithm that addresses the energy consumption issue of PoW. It is a more energy-efficient consensus algorithm that eliminates the competitiveness between miners and reduces hardware costs. Instead of using computational power to solve mathematical problems, PoC uses a different approach, which is based on the cooperation and reputation of the network participants. This approach allows the network to reach consensus more efficiently and with less energy consumption [58]. This blockchain does not have a gas fee per transaction, which differs from the majority of blockchains. Pylon Network (PYLNT) is also used to reward the work of validating and maintaining the information stored and this work is assured by nodes called Validating Nodes.

2.3.6 PONTON

PONTON developed a smart energy trading solution for regional markets via a blockchain called Gridchain. The platform was successfully used to trade energy between Yuso, a renewable energy platform located in Belgium, and Priogen, a market focused on short and medium-term power trading located in the Netherlands, in 2016. To achieve this, PONTON partnered with over 40 energy trading firms with the goal of achieving smart trading between those two [10]. The Gridchain was designed to achieve greater coordination between all the Transmission System Operators, aggregators, and Distributed System Operators to prevent energy congestion. Another goal of this innovative tool is to contribute to a European standardization of communication for smart grids [59].

2.3.7 Summary

In this section, a summary of the relevant aspects regarding the previously mentioned projects will be provided, by emphasizing their key features related to the usage of DLTs.

	PriWatt	Brooklyn MG	Jouliette at De Ceuvel	OmegaGrid	Pylon Network	PONTON
Scalability	Medium	Medium	Low	High	High	N/A
Decentralization	High	High	Low	Medium	High	Medium
Maturity	Deployed	Deployed	Deployed	Deployed	Deployed	Deployed
DLT	Bitcoin	Ethereum	Custom	Custom	Custom - Pylon Chain	Custom - Gridchain
Consensus Algorithm	PoW	PoS	N/A	PoA	PoC	BFT
Custom Token	No	Yes	Jouliette	Yes	PYLNT	Yes

Table 2.2: Comparison between related projects.

To better understand the above table, it is necessary to review how the project characteristics were evaluated. The scalability parameter, responsible for giving information about how the project could expand and how easy it is to happen considers the consensus algorithm, which has an important role when deciding the scalability of the system, and how the nodes are integrated into the network since there a significant penalty to escalate the network if node onboarding is not automatically processed. Decentralization was measured by analyzing if some nodes have differentiated importance on the network while verifying transactions. Another important aspect to consider is if projects operate with a database outside of the blockchain, then the decentralization is penalized. The DLT used for the projects could either be one of the major existing blockchain technologies, such as Bitcoin or Ethereum Blockchains, or they could be a custom implementation developed by the companies themselves. The consensus algorithm can be one of the most common algorithms implemented, which was explained earlier, or it can be another algorithm developed or merged by the developers of the project. The Custom Token is a parameter indicating if the project uses a custom token or not. Information that, at the moment of the elaboration of this table, could not be retrieved or evaluated was marked with an N/A.

According to what was reviewed in the consensus algorithm table 2.1, PoS achieves better metrics when compared to the PoW algorithm. Since no statistics were available for the PoA algorithm, it is preferable to use a known and tested algorithm, like the PoS one.

The Brooklyn Microgrid seemed to be a good example to follow. All the projects appear to have the same degree of maturity, which is a good metric to prove that the projects can achieve some longevity. Scalability can be somehow ignored due to the low amount of devices that are going to be introduced in the network. The interesting part comes in the DLT row. 4 out of 6 analyzed projects opted to develop a custom blockchain for its own purpose and usage, either by using Pylon Chain, in Pylon Network project [57], or Gridchain, in PONTON's case [10]. Jouliette at De Ceuvel and OmegaGrid implementations do not manifest what type of DLT they are using, meaning that possibly some custom-made solutions were considered. Only PriWatt and Brooklyn microgrid opted for universal alternatives like Bitcoin and Ethereum. PriWatt has the disadvantage of relying on Bitcoin. Using a high pollutant blockchain for a green energy incentive project seems contradictory, leaving Brooklyn Microgrid the most viable model of implementation in terms of DLT usage. Although it is claimed that Ethereum was the best-adopting choice for permissionless blockchain usage due to using PoS, this was not the case when Brooklyn Microgrid started back then [53]. As stated previously, Ethereum also used PoW as a consensus algorithm. In terms of custom tokens, 5 out of 6 alternatives use a proprietary token. The most plausible explanation is the usage of applications such as smart contracts to track the energy exchanges alongside their corresponding monetary value [10].

To adopt a permissionless blockchain it is necessary to keep in mind which consensus algorithm is behind the technology, like the usage of Ethereum on Brooklyn Microgrid. When developing a custom alternative, either by using a framework like the ones presented above, a more personalized approach could be adopted, because there is more control on every edge of the energy and monetary flow spectrum.

CHAPTER 3

Architecture

In the dynamic realm of sustainable energy solutions, the architecture of a system serves as the foundation for orchestrating diverse components toward a common goal. To do so, it is necessary to build a platform that enables communication and information processing of the various REC stakeholders. The platform is composed of an array of microservices, each dedicated to specific functions and responsibilities. By encapsulating discrete features within these microservices, the REC platform attains scalability, agility, and responsiveness, critical in addressing the evolving demands of renewable energy ecosystems. Throughout this chapter, the document provides a comprehensive exploration of the REC platform's architecture. The following subsections will guide that exploration:

- **3.1 REC Platform Architecture:** Section 3.1 presents to the reader the different components inherent to a REC platform. It is in this section that is shown an overall overview of its different constituents while explaining briefly why they exist as well as some important considerations throughout the document.
- **3.2 Backend Platform Services:** Section 3.2 guides the reader through the different analyzed approaches and explains why some of them were abandoned to the detriment of others. It is there that some of the core technical components start to be mentioned alongside the explanation of the information flow throughout the different architecture components.
- **3.3 Energy Transactions Use Cases:** Section 3.3 focuses on the different actors and their possible actions when interacting with the whole API. The document tries to provide an alternative to the already existent energy systems by resorting to DLTs, but while doing it, different forms are explored. So, it is important to provide a clear view of the different interactions available for each actor in this ecosystem.
- **3.4 REC Components Description:** Finally, section 3.4 describes the core operation of each different microservice and device related to the API, while briefly explaining the technologies present on each one. Every relevant component to the operation of the previous information flow has its subsection and there it will be explained its internal operation as well as some important particular considerations to it, when applicable.

Each subsection explores the REC platform's architecture and its various components, unraveling the synergistic collaboration of microservices that drive the evolution of RECs. By the culmination of this chapter, readers will have gained a profound understanding of how these microservices synergize, facilitating the creation of sustainable and interconnected energy communities, poised to tackle the existing challenges.

3.1 REC Platform Architecture

A REC platform contains many more components than just an API. Therefore, the diagram represents the whole concept used in this specific document. Figure 3.1 represents the overall architecture of the COMSOLVE project, the project in which this thesis was developed.

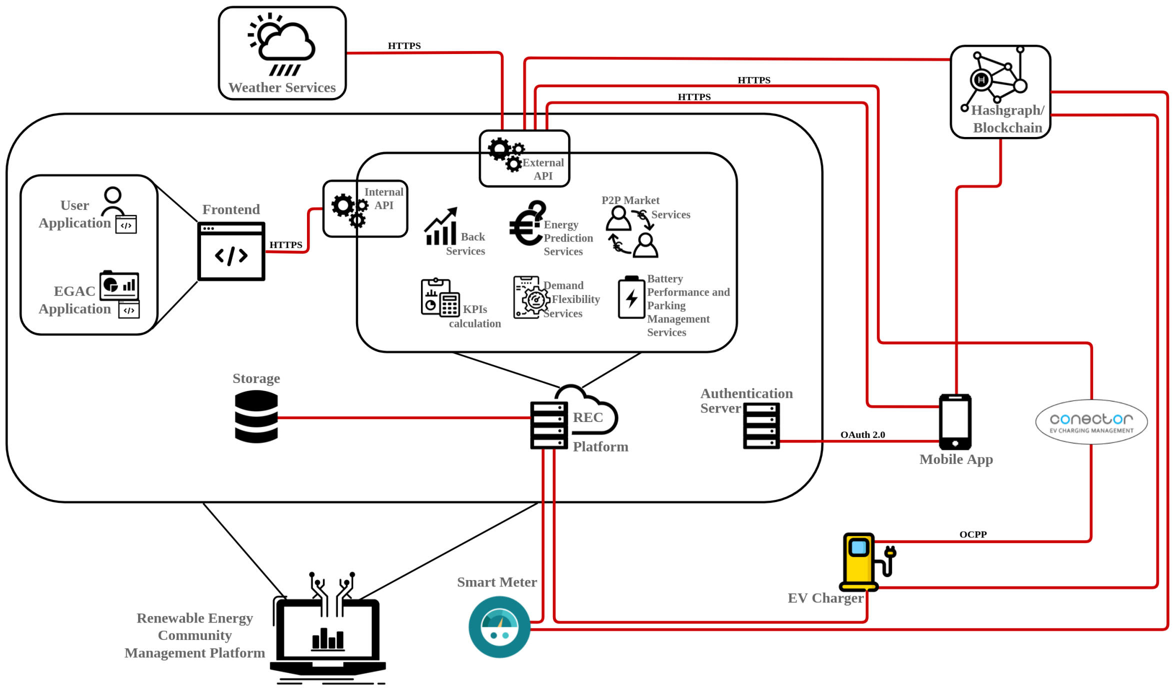


Figure 3.1: REC Platform overview.

As shown in the diagram, a core part of the design relies on the REC management platform, a component where both frontend and backend services are present. The frontend should contain at least two dashboards: one for a normal community member, either to show statistics or to define his/her energy prices (useful for the Market Services and indirectly for cryptocurrency conversions), and another for the community management itself. The backend contains various services, energy, and price prediction as an example, but the most relevant part of the document relies on the Back Services and P2P Market Services. There is also a database associated. This database is needed due to compliance with local country law and the General Data Protection Regulation (GDPR)[60]. Therefore, sensitive data should be private, secure, but also auditable. Using a DLT, those aspects could also be implemented using ciphers as an example. However, a person may want to have his/her information erased, as stated in GDPR, something not achievable on most known DLTs to date.

Outside the REC management platform, there are both smart meters and EV chargers. Although they serve different purposes, both of them could be represented as a single type of device because their interest in this document is simply to take measurements and issue transactions. Most of the time, only smart meters are referred to in the document, since those devices were the ones that were continuously developed and tested.

The used DLT technology was Hedera Hashgraph, due to its characteristics as described in the Fundamental Concepts, section 2.1.3. Hashgraph is linked almost to every component of the architecture due to the frequent need for information checks and payment. There is also the necessity of having a Weather dataset for the production and consumption prediction and energy management components. There is also the possibility of integrating a mobile application that may have the functionalities of the frontend and other use cases, for example, verifying EV Charger availability nearby.

3.2 Backend Platform Services

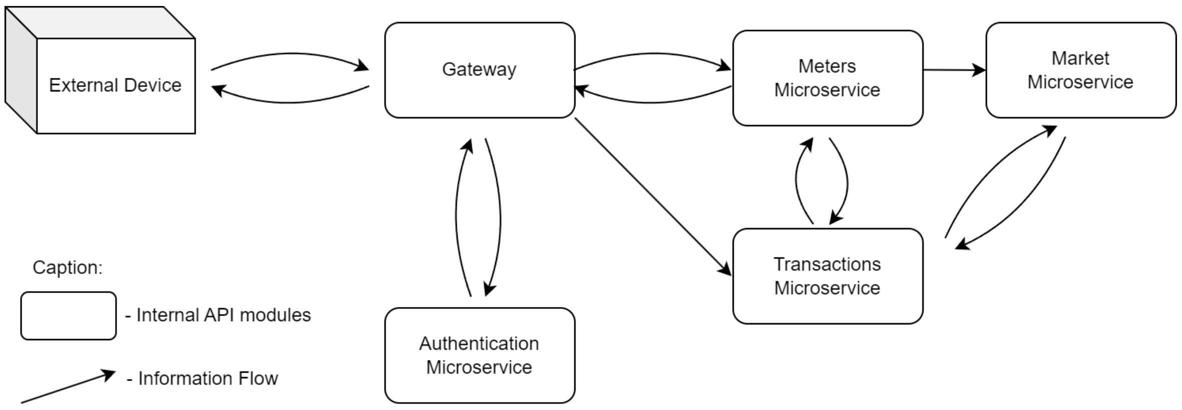


Figure 3.2: Internal API structure.

When conceiving the actual architecture, there were older architectures that were developed, but further refuted. The first idea of the API consisted of a single API where all the needed modules were present. So, instead of multiple individual parts that behave independently, everything had to be compiled and executed at the same time. As time went by, this approach was impractical due to diverse modules being developed and at different stages at each API deploy. So, each time something was changed, the whole architecture needed to be shut down, even when some of the services running did not have any implications on the newer updates that were being deployed. Another important aspect was the phase of debugging. When something malfunctioned, it was not easy to trace the error back to its root due to either similar error characteristics or error propagation between services. Lastly, the traffic amount comes in bulks of data, because the connections from every device happen in specific periods. This happens in accordance with the country law which states that the smallest timeframe between measurement registers is 15 minutes in Portugal, where everything that occurred inside this period is considered net metering. Net metering is a billing process where the energy consumed and produced by a building is deducted from each other, leaving only the remainder of energy to be accounted for (either surplus or deficit of energy). If there were devices sending requests when debugging, the whole process of finding the error on a mono repository turns out to be harder.

Given the previous explanation, the traffic generated by the API and its functions are not interdependent. For instance, registering measurements does not imply the issuing of payments. The API's structure was designed using a microservices architecture approach. Those microservices communicate between themselves using Remote Procedure Call (RPC). Explaining the components of the architecture following the communication flow, when something is sent to the API, the first microservice responsible for handling the incoming requests is called "Gateway". Its function relies on redirecting incoming requests to the microservice accountable for the processing of that information. Due to the requests being made via Hypertext Transfer Protocol (HTTP), this microservice is also responsible for converting them into the format required by the various components. Alongside the Gateway, there is a microservice called "Authentication", that is summoned each time an endpoint needs a validation of the requester. An example of that is the endpoint where the meters send measurements periodically, which needs to certify that the sender is the real smart meter and not some malicious actor on the system. So, to continue the information flow, the request continues to the microservice "Meters", whose responsibility is to process, store, and provide any information related to the different device measurements. These measurements are requested internally by another microservice, "Market", responsible for, via different trading algorithms, matching consumers and prosumers according to their energy surplus/deficit as well as their stipulated kWh selling/buying price. This microservice will run independently and asynchronous from any other process running internally on the API. When a measurement is processed by the Meters microservice, one last microservice is called, "Transaction", which returns the pending payments that the smart meter still has to issue. Transactions then request every entry on the Market microservice database that is still unpaid and convert them into a state coherent with Hashgraph objects. Finally, these objects are returned to the device that, via a locally preinstalled wallet, issues the transactions to the DLT for later verification and validation from the REC management platform, the API.

3.3 Energy Transactions Use Cases

After the conceptualization of the architecture defining use cases (section 3.2) was important to realize which actors are in play on this transaction flow and what should their actions be.

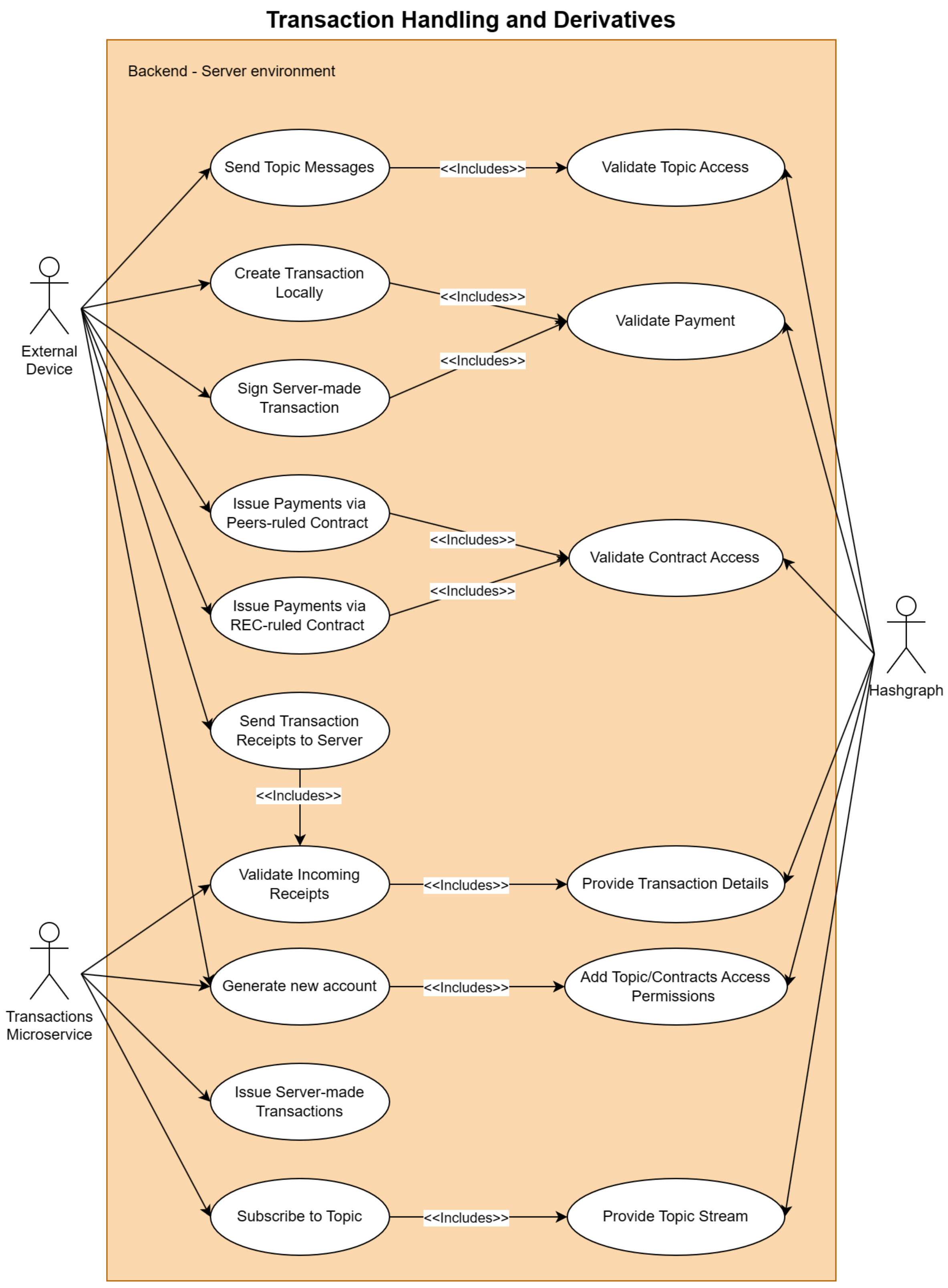

Transaction Handling and Derivatives
Backend - Server environment
External Device
Transactions Microservice
Hashgraph
Send Topic Messages
Create Transaction Locally
Sign Server-made Transaction
Issue Payments via Peers-ruled Contract
Issue Payments via REC-ruled Contract
Send Transaction Receipts to Server
Validate Incoming Receipts
Generate new account
Issue Server-made Transactions
Subscribe to Topic
Validate Topic Access
Validate Payment
Validate Contract Access
Provide Transaction Details
Add Topic/Contracts Access Permissions
Provide Topic Stream
<<Includes>>

Figure 3.3: Transaction handling and derivatives.

Figure 3.3 represents actions of the different stakeholders with the API described in figure 3.2.

To demonstrate the importance of the use cases, it is necessary to understand the operation flow between all the interveners. To give context, the *External Device* is the smart meter attached to the electricity meter. For simplicity of reading, “device” is going to be used from this part on.

The device is responsible for issuing payments on the ledger regarding energy exchanges. To do so, when booting for the first time, it must be able to generate an account wallet to interact with the DLT. This wallet must reside only on the device, but there is a need to notify the API of the new device in the community. Besides that, the *Transactions* must allow this new device to make part of every content related to the REC inside the DLT, like smart contracts and topics. Since topics were never discussed before, it consists of an address on Hashgraph where authorized accounts can write messages, similar to a “Community Inbox”. Smart contracts and topics are member-restricted only, which means that, in their development, there is a verification of the account interacting with them. If it matches an account that is related to the REC, they are allowed to perform actions, such as writing a new message. If not, their access is denied and no further action is required.

There are four main approaches to how transactions are made with slight variations on how to achieve them. That said, they are referred to later as:

1. “Direct Transactions”, where the response from the API server is a JSON object containing an array of pending payments that must be processed on the device and issued from it;
2. “Freeze Transactions”, where the server, specifically the Transactions microservice creates the transaction Object and “blocks” them, leaving only the possibility of signing it or not for the device;
3. “REC-Ruled Contract”, where the funds are sent to a smart contract wallet hosted by the REC Manager that periodically verifies them and acts accordingly, either by refunding them or marking them as paid;
4. “Peers-Ruled Contract”, similar to the above but this time the responsibility of verifying the correctness of those pending payments relies on the prosumer, and the REC manager just assumes the position of supervisor.

If everything was set up correctly, for every approach, Hashgraph just validates those payments and, in the case of smart contracts, checks if the incoming communication is from an allowed account.

The first two approaches have a secondary consequence after the transaction acceptance on the ledger. For the server to have knowledge of the already made payment, the external device sends back a receipt that contains the address of that transaction on the ledger. And here comes the slight variation. This could be achieved via Topic, the “Inbox” mentioned above, or via another endpoint back in the API.

If it is made via Topic, then the external device sends Topic messages and Hashgraph needs to check if the REC manager provided access for this device to write a message on.

If it is made via an endpoint, then the external device sends all the receipts to the server. The microservice queries Hashgraph for the transaction details and validates it with the payment that that device was supposed to make.

Moving to the Transactions actor, besides all of the actions above, it should have the capability of issuing the “Freeze Transaction” and sending it back to the device. Finally, it should also have the ability to subscribe to the Topic and listen for new events on that address, therefore, Hashgraph must provide Transactions with a Topic Stream from an already created Topic on early setup phases of the API.

3.3.1 REC-Ruled Contract

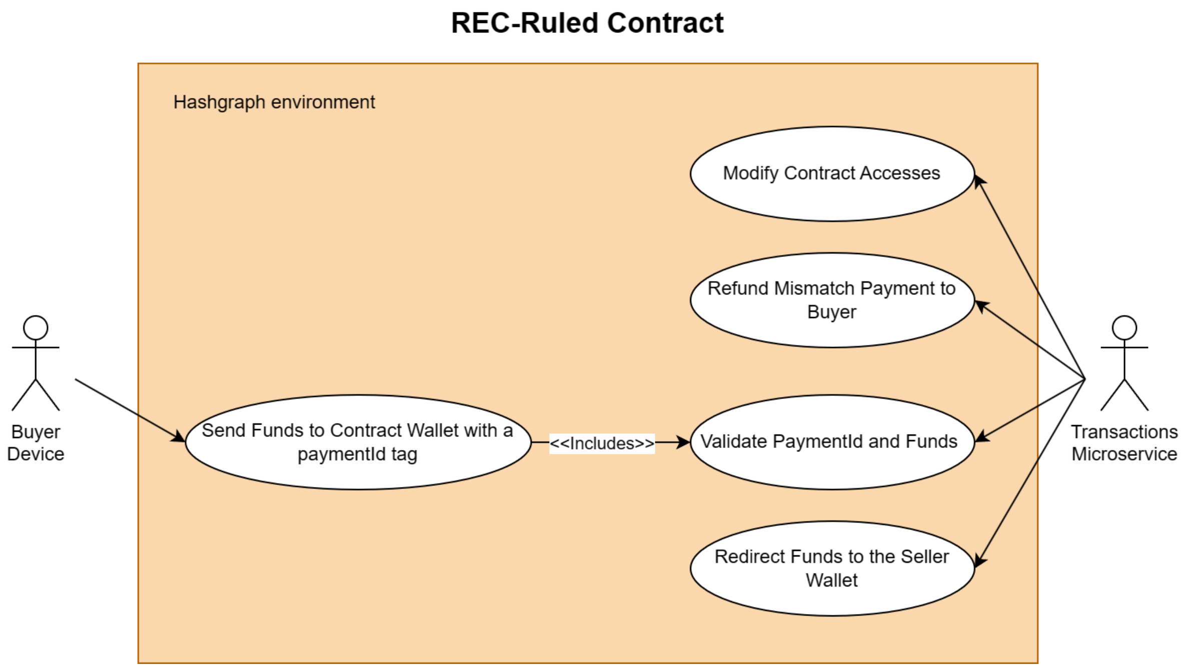
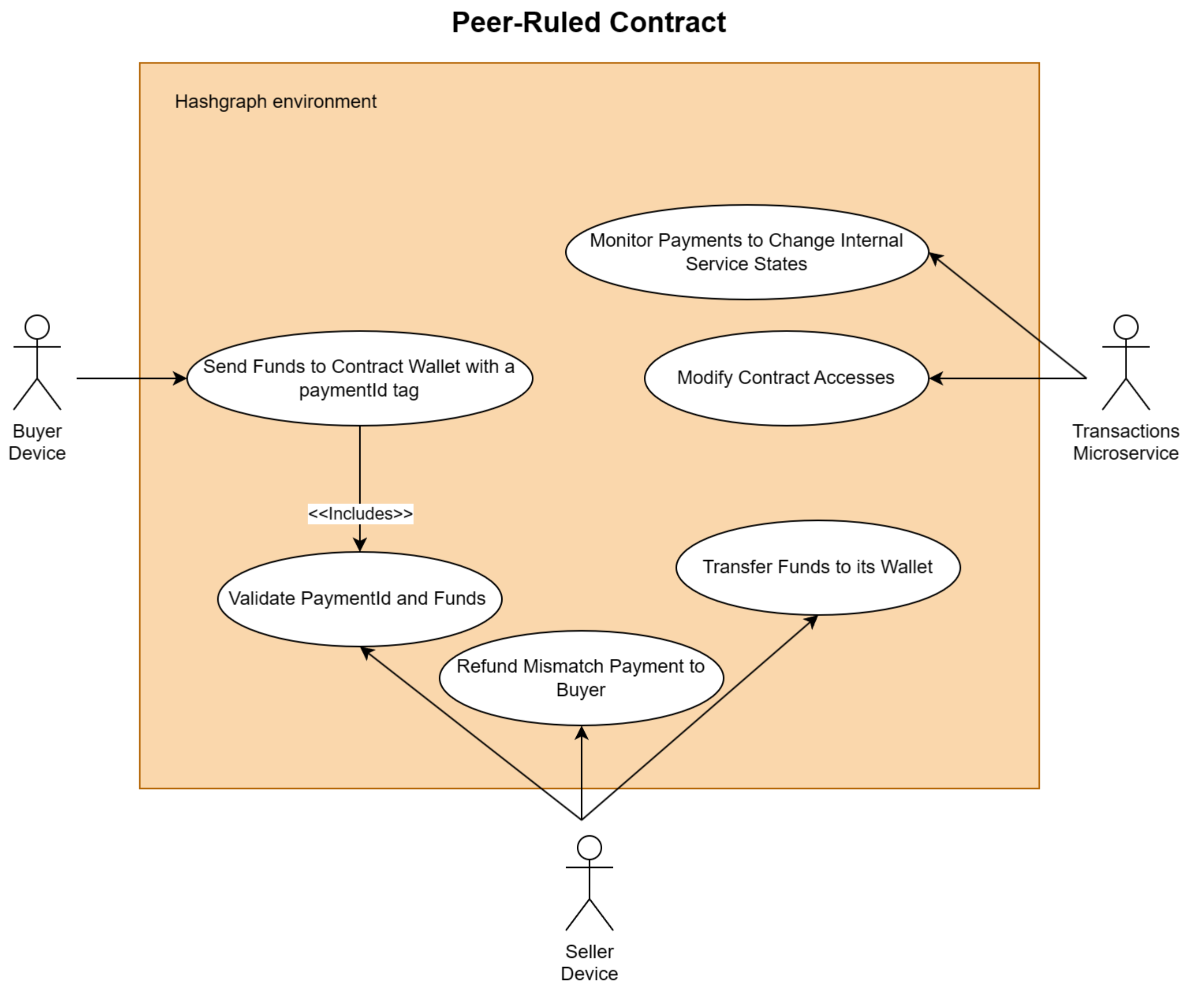

REC-Ruled Contract
Hashgraph environment
Modify Contract Accesses
Refund Mismatch Payment to Buyer
Buyer Device
Send Funds to Contract Wallet with a paymentId tag
<<Includes>>
Validate PaymentId and Funds
Transactions Microservice
Redirect Funds to the Seller Wallet

Figure 3.4: REC-Ruled use case.

Although Hashghraph is an actor in figure 3.3, it is also an environment for other use case appliances, shown in figure 3.4 and in figure 3.5. Talking about REC-Ruled contract, two actors come into play: the buyer device and the Transactions. The owner of the contract is the Transactions microservice, the part of the REC management entity that operates the transactions on the DLT. Therefore, it should be able to modify who can interact with the contract. The buyer just needs to send the pending payment funds with the payment identifier to the contract. Periodically, the Transactions verify the pending payments registered on the contract and validate them. Two actions can be taken: if the payment is correct, then the funds are transferred into the seller device wallet and, internally, the service marks the payment as concluded; if not, then the payment is refunded and must be resent correctly.

3.3.2 Peers-Ruled Contract



Peer-Ruled Contract
Hashgraph environment
Monitor Payments to Change Internal Service States
Send Funds to Contract Wallet with a paymentId tag
Modify Contract Accesses
Buyer Device
Transactions Microservice
<<Includes>>
Validate PaymentId and Funds
Transfer Funds to its Wallet
Refund Mismatch Payment to Buyer
Seller Device

Figure 3.5: Peer-Ruled use case.

Since the purpose of a DLT is to decentralize processes, this approach tries to achieve that. Distancing oneself from the above, section 3.3.1, the responsibility of verifying payments relies on the device that receives them. In this case, three actors interact with the contract: the buyer device, the seller device, and the Transactions. The function of the buyer remains the same as in the previous contract. However, Transactions now only monitors the flow inside the contract just to change its internal state, for example, if a payment was confirmed, it changes its state on the API. Its privileges of modifying contracts remain the same. Now, the seller device has the responsibility of validating the pending payments. If correct, the funds are transferred to its wallet and if not, then the payment is refunded back to the buyer.

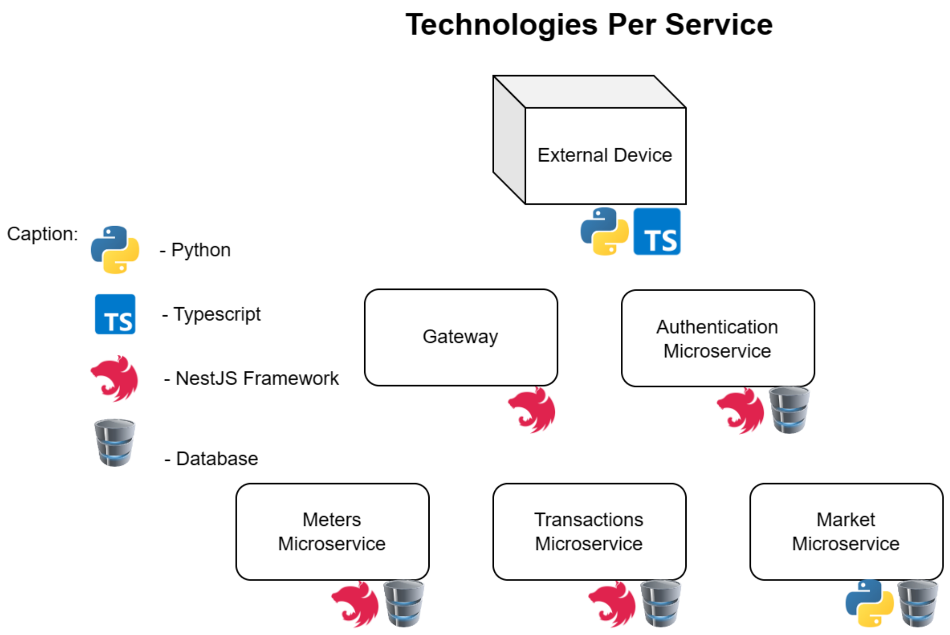


Figure 3.6: Technologies per service.

When developing and studying useful tools for the architecture above, the possibility of making it as generic as possible was taken into account. With minimal effort, microservices should be easily transferable between different server specifications with little or no need for modifications. Moreover, the smart meter should attach and detach from the electricity meter while autonomously recognizing the characteristics of the equipment it's connected to and adjusting its functionality accordingly.

3.4.1 Communication Layer

The DLT can not work without a core infrastructure specific to this type of project. So, it is necessary to develop a whole backend API for the smart meter devices to be able to send and receive data. With that said, the API has a microservice structure. Inside the API, all the communications are made using gRPC as a transport method. Since this work also has research and investigation purposes, using RPC is one of the research parts of it. The idea of using it was also influenced by the DLT, because it possesses direct communication with it via gRPC. This technology requires the usage of a file called "protobuf" where all the methods and arguments must be described. It gives every service that tries to communicate with that gRPC server a clear understanding of all the reachable methods and what is expected to return from them. Apart from it, the same "protobuf" file provides the ability to have separate groups of methods that could, for example, allow the usage of Secure Sockets Layer (SSL) certificates between microservices. This example is implemented between the Transaction and Market microservices and it is going to be detailed later in the document. The API has a main gateway that receives the requests either from the devices or by human-made requests. This is a necessity since gRPC uses HTTP/2 which is incompatible with the vast majority of browsers available. So, the gateway needs to receive proper HTTP/1.1 requests and transform them into gRPC-compatible content. Inside the API environment, five microservices are operating. The Authentication microservice is responsible for authenticating incoming messages and allowing or denying the progress of the requested process, mostly securing the endpoints that update or create database entries. Although Meters and Market Microservices are referred to, they are not relevant to explain their internal operation either due to the particularities of the research project developed alongside this document or because it is the research problem of another dissertation. The necessary part to retain from this is that the Meters request Transactions for pending payments and Market returns community matches that need to be paid or changed. Finally, the important microservice for the case study is the transaction microservice, responsible for handling incoming pendent payment transactions and other key features relevant to the blockchain infrastructure.

3.4.2 Smart Meter Device

To structure this device, there was the need to understand, “a priori”, its operation in every stage of the architecture flow, Figure 3.2.

The first stage is the extraction stage. This device needs to be connected to the electricity meter of the building where it is supposed to operate. This requires that the device in use should be capable of handling logic and be able to communicate with the electricity meter via its own communication protocol. Next, the results obtained are converted and filtered into a universal format that can be replicated into any device even if its model differs.

Following, the next stage is the communication stage. The device must be able to transfer the data gathered in the previous step to the server. To do that, it needs an internet connection. Not every building has available Ethernet or Wi-Fi communication near the location of the electricity meter. Therefore, the device must possess some sort of independent module that facilitates this communication.

Next, it is necessary to handle the response of this communication. Here, the response could be in different formats, and the device should be capable of temporarily storing them and handling them accordingly. A database should be used to contemplate this stage.

Another step that is the objective of this document is to handle payments on the DLT. The device must have tools capable of retrieving those database entries and processing them into currency transactions on the DLT without any human intervention.

Finally, the confirmation stage. From the previous stage, there should be any type of confirmation or record on the DLT. By itself, the server could not reach or guess which of the transactions correspond to a specific payment, therefore the device notifies the server of its occurrence.

3.4.3 Authentication Microservice

The authentication microservice is responsible for protecting every endpoint. For example, to successfully send a request to the meters counterpart, the message must be correctly authenticated. To achieve this, a One-Time Password mechanism was implemented, more specifically a Time-Based One-Time Password (TOTP). This mechanism could be seen as similar to the usage of the “Nonce” parameter when finding a valid block since it uses the message timestamp to dynamically change the encryption data. To achieve the TOTP, some steps should be made at the assembly phase of the device: an “API Key”, consisting of 16 Bytes, is generated by the server and stored on the device as well and the mac address of the WiFi interface of the device must be registered and stored on the server as well as the name of the device, that follows the rule: “es-sms-XX”, where “XX” is the number of the OpenVPN account identifier. Finally, a header called Authorization is sent alongside each request containing an SHA-512 hash generated by concatenating the parameters in this order: device identifier, timestamp of the current request, API key, and mac address. To verify on the server side, two extra headers are sent: deviceId and the timestamp. The microservice receives the headers and hashes all the parameters in the same way to produce the same hash. It is also important to specify a time window where a specific timestamp is valid. This is necessary because the communications are made via GSM and some of the locations where the devices are located may not be in the most indicated condition. Although this is made, the timeframe of validity is being stuck to the minimum time possible. It means that a request is valid if the timestamp has a small offset. For example, if the TOTP is issued at 7:00:00, it is valid until 7:00:10 to compensate for delays (those values are merely representative). Another important validity check is if the data present in the body corresponds to the header that is being sent to prevent users with malicious intentions. For example, headers could be copied from a valid message and implemented in another body message whose intention is to fake a measurement, for example.

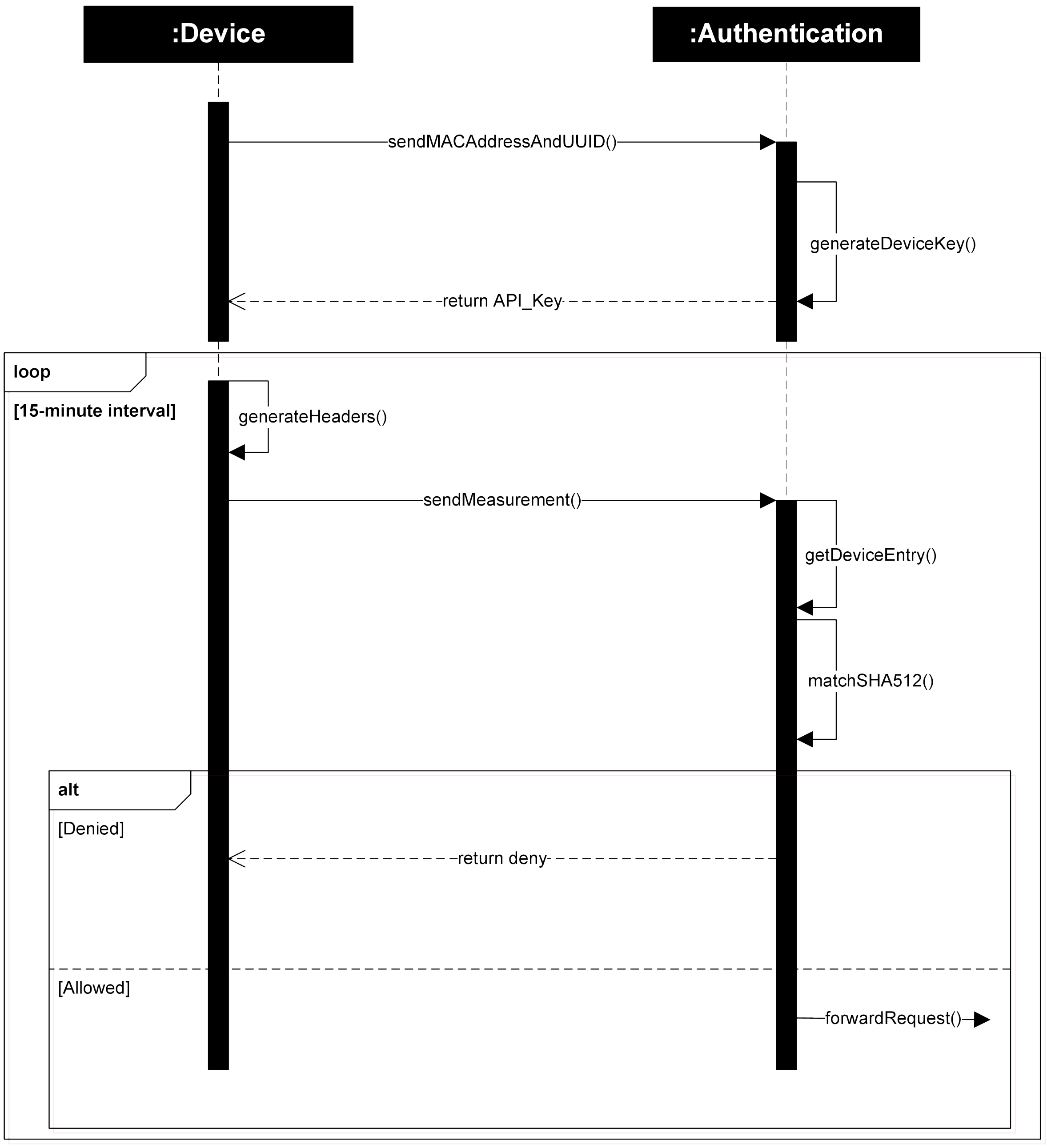


Figure 3.7: Internal configuration and communication.

To develop this microservice, NestJS[61], which is a Node.js framework using Typescript language, was used with a Docker image of PostgreSQL[62] database to store the authentication keys, mac addresses, and device identifiers. Every other microservice uses NestJS, besides the Market and the smart meter/external device. Other additional differences that other microservices may possess are stated in their own section.

3.4.4 Gateway Microservice

The *Gateway* is the front service that serves as a barrier between outside communication and the internal operation of the server. It is here that every HTTP request is translated into gRPC defined arguments and sent to the desired microservice.

```
/**...**/
export const grpcClientMetersOptions: ClientOptions = {
  transport: Transport.GRPC,
  options: {
    url: 'meters:5049',
    package: protobufPackage,
    protoPath: 'node_modules/grpc-protos/proto/meters.proto',
  },
};
```

Code 3: Normal Opening Connection

As shown in Code 3, every connection that is opened must state that its transport is carried via gRPC, as well as its destination “url:port” alongside the protobuf file already mentioned. This specific snippet shows an opening connection from the *Gateway* to the *Meters* Microservice hosted on port 5049 and imports its “.proto” file that contains every available method and argument. Connections to other services are opened the exact same way, on *Gateway* as well as on the other microservices, besides a specific connection between the *Transactions* and *Market* microservices. This specific case is going to be detailed later when talking about that connection.

Once every connection is defined (that does not mean it is created nor established), the endpoints defined on the Gateway shall then connect to the gRPC method that is mapped.

```
/**...**/
Controller("meters")
export class MetersController implements OnModuleInit {
  Client(grpcClientMetersOptions)
  private readonly meters: ClientGrpc;
  private metersService: MetersServiceClient;

  onModuleInit() {
    this.metersService =
      this.meters.getService<MetersServiceClient>(METERS_SERVICE_NAME);
  }

  Get("measurement")
  async retrieveMeasurement(Query() entry: QueryMeters): Promise<QueryResponse> {
    return await firstValueFrom(this.metersService.retrieveMeasurement(entry));
  }

  Post("measurement")
  UseGuards(AuthGuard)
  async addMeasurement(...): ... {
    //...
  }
}
```

Code 4: Endpoint Mapping

On Code 4, the mapping of endpoints starting with“/meters” (“@Controller("meters")”) is shown . The class “MetersController”, when the service starts, creates the connection to the Meters service via “onModuleInit()” function. This prevents errors like calling endpoints before the connection was set. Taking a closer look at the method “GET /meters/measurement”, it is possible to see that the arguments are well defined (“‘QueryMeters” and “Query Response”).

Those structures are the ones that come from the “protobuf” file, therefore they are stated there to prevent wrong structures to be passed through.

Another important statement is that gRPC connections support stream connections, that is, it is possible to maintain a flow of unidirectional or bidirectional information until one of the ends of the connection closes it. So every time a non-stream connection is made, it is necessary to call “firstValueFrom()” in order to just wait for a single response from the called microservice. It is only when “this.metersService.retrieveMeasurement(entry)” is called that the connection is then established and the handling of the incoming request is made.

Taking a closer look at the “POST” method, another decorator is present, “@UseGuards”. Using this, the microservice then knows that the endpoint is protected, thus redirecting the request to the already explained Authentication microservice to make its verifications (section 3.4.3).

Finally, this microservice is also responsible for mapping error between gRPC and HTTP since they are different. Here are some of them just to exemplify it:

HTTP	gRPC
Unprocessable Entity	Invalid Argument
Internal Server Error	Aborted
Service Unavailable	Unavailable
...	...

Table 3.1: Error equivalence.

3.4.5 Meters Microservice

Without entering into too much detail since this microservice serves as an intermediary between the smart meter device and the *Transactions*, *Meters* is built using NestJS and uses an “InfluxDB” Time-series database to store incoming measurements from all the smart meters. It is also possible to query measurements from it. After storing it, this service establishes a connection with *Transactions* where it sends the device identifier from the request and awaits pending payments that the device must issue into the DLT, returning them to the smart meter as a response.

3.4.6 Market Microservice

This service is the research component of another dissertation that was elaborated alongside the COMSOLVE project. Therefore, just to give an overview, it consists of a service that every 15 minutes queries for new measurements to the *Meters* microservice. After that, according to a specific matching algorithm, it creates peer pairs of consumers and prosumers according to their stipulated prices and energy surplus/deficit in that period. It can be seen as the core storage service of the API since it is here that resides in critical information that is important for auditing and data validation.

Due to the critical data present, it is necessary to have two groups of methods, one that just queries that information and another one, more secure, that can interact and modify that information. Then, there is the need for SSL certificates implementation. To protect those methods the server generated a certificate and a private key that only Market possesses. So, when starting the server, it loads this pair of files and just accepts communications from clients that have the certificate of the *Market.* Since the certificate is not broadcasted, only its holders can connect to the Market to make changes and this holder should only be the Transaction microservice.

3.4.7 Transactions Microservice

The core case study of the document relies on two services: the DLT used, Hedera Hashgraph, and the microservice API that uses it and processes different information regarding information data storage on the ledger as well as sends and verifies recently made payments across the peers enrolled on the community, the Transactions Microservice.

First, there is a specific need to join new members in the different parts of the backend. That said, when information regarding a new account creation arrives in the Transactions, it executes a request for a new account in the ledger. After that, subprocesses are started to enroll this account into the various components. Since different payment approaches need to be evaluated, the microservice starts by calling the ledger Topic that the community has and adds the new peer into it, allowing messages from this new account. Besides that, the smart contract must be called and this new account shall be added to the authorized participants on those block codes hosted on Hedera Hashgraph.

When the service starts, prior to anything, it should search for the available topic on the ledger (this topic should already exist) and start listening to the messages. According to the different types of messages that arrive there, the microservice shall react according to them. For example, if a peer writes a message stating that he paid someone, the microservice has to validate it. This microservice is not responsible for holding any kind of information regarding peer payment matches. Therefore, it has to establish a connection with someone who does, in this case, the Market.

Since Transactions can change these match states, either by changing them to "Paid" or "Wrongly Paid", as an example, those two services need to establish a secure way of communication, using the certificates previously mentioned. Established this connection, then Transactions can query for pending payments specific to a peer as well as validate its transaction values.

Transactions must be able to query the ledger at any time for more detailed data since peers only provide a "receipt" of the transaction they made. This receipt can be seen as the address inside the ledger where that transaction lies either by arriving through an endpoint or via a Topic message, depending on the approach being used at the moment. The cross-verification of all the different sources of information above is the deciding factor when evaluating these transactions. Only then, *Market* is notified of the current state of a specific payment identifier.

Besides those approaches, *Transactions* shall also have forms of communication with two different contracts, the REC-Ruled Contract and Peers-Ruled Contract. According to the particularities of each one, it must possess methods that add newer members to the contracts and provide means to supervise its activity.

3.4.8 Hedera Hashgraph Ledger

To support the whole architecture, a connection to a ledger node is needed. Looking at some of the common alternatives, Hedera Hashgraph hosts a network specifically designed for the development and testing of functionalities. Besides this alternative, there is also the possibility of hosting a local network just with this in mind. Having an environment completely configurable could also be a go-to approach due to the fact of constant debugging and improvements throughout the implementation phases.

That said, Hedera Hashgraph must be able to hold and load every scenery such as the smart contracts, the topics, and the Direct and Freeze Transactions at reliable speeds and costs.

All the methods are embedded in the network itself. Therefore, its usage should be straightforward to implement with minor differences to other already known blockchain networks, such as Ethereum. This assumption can be made because Hedera Hashgraph networks also support Solidity, the language used for developing smart contracts.

CHAPTER 4

Implementation

In this chapter, the document describes the practical realization of the innovative energy transaction system within the context of a REC. The focal point of this implementation lies in the cutting-edge REC platform, which operates as a transformative force in facilitating peer energy transactions. Leveraging the power of modern technology, the REC platform harnesses microservices built using gRPC framework, alongside a robust gateway that translates HTTP requests into gRPC communications.

While looking through the technical aspects of this implementation, the document explores the architectural choices that have been made to enable real-time energy trading. Notably, the adoption of the Hedera Hashgraph technology as the underlying DLT for energy transactions provides a hands-on perspective on distributed ledgers. Hashgraph, known for its scalability, security, and fairness, emerges as a natural choice to validate and record energy transfers within the renewable energy community.

This chapter serves as a comprehensive guide to understanding the deployment of microservices, the orchestration of communication via the gRPC protocol, and the pivotal role of the *Gateway* in ensuring interoperability between different communication paradigms. Furthermore, it offers insights into the integration of Hashgraph as the bedrock of trust and accountability in energy transactions.

Next, it presents the various components and technical implementation details.

4.1 Gateway Microservice

As stated before, the *Gateway* must serve as a conversion block between HTTP connections and gRPC ones. It is also here that the redirection of requests is made as well as the error handling and the data displayed that is going to be displayed for the user, on a browser, or for an IoT device, like the smart meters.

4.1.1 Available Endpoints

1. **GET /meters/measurement**

   This endpoint queries the Meters microservice to retrieve information according to various parameters:

```
{
    "startInterval": 15,
    "deviceId": "es-sms-XX",
    "skip": 0,
    "limit": 30,
}
```

Code 5: Optional Arguments on GET /meters/measurement

Every parameter is optional; by default, when none of them is provided, the response is an extensive list of the 1000 last entries on the database. If provided, "startInterval" is the number of minutes before the current timestamp that should be queried, in this example, it retrieves the last 15 minutes of entries; "deviceId" is the identifier of a specific smart meter; "skip" is an offset of entries, for example, if it is 4 it discards the first 4 retrieved entries; and finally, "limit" serves to limit the number of retrieved entries. Imagine that the query gives 100 entries, if the limit is set to 30, only the most recent 30 entries are shown.
The URL connection should be something like this:
"http://127.0.0.1/meters/measurement?deviceId=es-sms-XX"
Below are represented the possible responses to querying GET /meters/measurement:

a) ***Status Code 200 - The query was successful***

The output contains a list of objects following this example:

```
{
    "DeviceId": "es-sms-XX",
    "Field": "Active import",
    "Value": 5,
    "Timestamp": 1664211986.
}
```

Code 6: Desired Object Output

Value is in watt-hour (Wh) for Active import and Active export and in volt-ampere-hour (VArh) for Reactive QI(+Ri) and Reactive QIV(-Rc) field types.

b) ***Status Code 412 - Precondition Failed***

The reason behind this error relies on incorrect argument types or values above a certain limitation (either imposed by explicit API rules or derived from components that the API makes usage of). For the correct usage of the endpoint, the user should know the following: "limit" is an integer value between 0 and 10000 and its default value is 100; "skip" must be 0 or superior, "startInterval" could either be a date, example "2000-01-01T23:59:59.000Z", where every value from there until now is retrieved, or a positive integer number (corresponding to minutes) to return every entry from now until the current time minus the number of minutes required.

c) ***Status Code 503 - Service Unavailable***

An advantage of using gRPC and microservice architecture is precisely this error. A service could be in maintenance, offline, or with an error, and, besides that, all the other components of the architecture can be running normally without even detecting this microservice outage. So, to summarize, this error happens when the Gateway Microservice cannot make a connection to the Meters Microservice.

2. **POST /meters/measurement**

This endpoint is where devices connect every 15 minutes to send their measurements. Its desirable response is a status code 201 containing a body with at least one of three possibilities:

- Array with JSON objects containing pending payments.
- Byte arrays containing frozen transaction objects.
- A log message.

Below is a Python snippet code for endpoint interaction:

```python
""" ..."""
url = "https://127.0.0.1/meters/measurement"
meterId ='es-sms-30'
timestamp = str(int(time.time()))
mac = hex(uuid.getnode())
apikey = "eR2XCgnjmAxIfVjnMZ5v9A"
e = meterId+timestamp+apikey+mac
hash = hashlib.sha512(str.encode(e)).digest().hex()

measure_JSON = {
    "deviceId": meterId,
    "activeImport": 0.134,
    "activeExport": 0.00,
    "reactiveInductive": 0.00,
    "reactiveCapacitive": 0.00,
    "timestamp": timestamp
}
headers={'authorization': hash, 'deviceid': meterId, 'timestamp': timestamp}
requests.post(url,headers=headers,json=measure_JSON)
```

Code 7: Interaction with POST /meters/measurement using Python

Below are represented the possible responses for POST /meters/measurement:

a) ***Status Code 201 - The connection was successful***

The response could come in two different forms or every approach combined. That said, the response contains a field called "approach" that gives the methodology that the API expects to work on at a later stage. After that, it can have an array "json" where all the pending payments reside or an array "transactions" with the payments pre-processed and compiled into a form that the blockchain already expects them to be. Those two arrays are not mutually exclusive, therefore, they could coexist in the same response. It is important to pass to the reader the idea that inside this endpoint, there is a multi-stage process. At first glance, the new arriving measurements must be stored correctly. After that, the pending payments are retrieved and returned to the requester. If something else happens in the meantime that is not the intended behavior, alongside the normal fields, a message appears containing an explanation of what really happened and in which phase the request flow misbehaved.

```
{
  "approach": 0,
  "json": [
    {
      "timestamp": "2023-09-13T09:45:00Z",
      "buyerID": "0.0.1034",
      "sellerID": "0.0.1002",
      "energy": 0.3400000035762787,
      "price": 0.7339674830436707,
      "id": "650185cadedb229a217049ab",
      "createdAt": "2023-09-13T09:50:02Z",
      "transactionState": 0
    }

  ]
  "transactions": [
    {
      "transaction": "/***Byte array containing the object***/"
    }

  ]
}
```

Code 8: Response for POST /meters/measurement

On the "json" array, every object must contain, at least, the following fields: "timestamp", which has the date that the exchanged energy corresponds to; the "buyerID" and "sellerID" are the wallets involved in the transaction. Typically, "buyerID" is the device that receives the response, and "sellerID" is the wallet that which the buyer must transfer funds. "energy" is the volume of energy in kWh exchanged and its "price" in Hashbar (or HBAR), the currency unit of Hedera Hashgraph for the prior 15 minutes of the timestamp provided. The last relevant field is the "id" that corresponds to the identifier of the match in the internal API structure for later verifications. Any other field obtained via this method can serve as some type of validation or can simply be neglected.

b) ***Status Code 503 - Service Unavailable***

This error happens when the Gateway Microservice cannot make a connection to either the Authentication Service or the Meters Microservice.

3. **POST /transactions/account**
When a smart meter is deployed, there is no wallet associated with the device by default. That means the device must be independent when generating its account/wallet, therefore, when launching for the first time, it possesses the means to generate the needed Key Pair, the Public and Private Keys. Another aspect is the need to call

another wallet for this creation, because there are taxes associated with this account creation and the device does not possess any funds or means for its creation. Since there is a wallet managed by the Community Management, the device sends a pair containing the device identifier plus its newly generated public key, to the API. With this, internally, the API creates an account for that device without ever knowing its private key.

```
{
    "pubkey": "302d300706052b8104000a032200022...",
    "deviceId": "es-sms-XX",
}
```

Code 9: Arguments for POST /transactions/account

Below are the possible responses for POST /transactions/account:

a) ***Status Code 200 - The query was successful***

This endpoint does not provide any output, therefore, if nothing is returned, it worked as intended.

b) ***Status Code 412 - Precondition Failed***

This status code may occur when a new wallet is issued for an already registered device with an already existing wallet. This could be an alert to inform any malfunction or intrusion on the device.

c) ***Status Code 503 - Service Unavailable***

This error happens when the Gateway Microservice cannot make a connection to the Transactions Microservice.

4. **POST /transactions/receipts**

The last available endpoint for external interaction with the API is responsible for receiving incoming receipts of already paid payments via blockchain. Here, the API expects the arrival of a list containing the pair “txID”, and “paymentID” of those transactions. The “txID” parameter stands for the receipt that results from issuing a transaction on the blockchain, referring to the address location of the transaction, and “paymentID” was the identifier provided at an earlier stage, specifically when receiving the response of Code 7.

```
"receipts":[
    {
        "paymentID": "650185cadedb229a217049ab",
        "txID": "0.0.1034@1694599203.687195526" ,
    }
]
```

Code 10: Arguments for POST /transactions/receipts

Below are the possible responses for POST /transactions/receipts:

a) ***Status Code 200 - The query was successful***

Like /transactions/account, this endpoint does not provide any output, therefore, if nothing is returned, it worked as intended.

b) ***Status Code 503 - Service Unavailable***

This error happens when the Gateway Microservice cannot make a connection to the Transactions Microservice.

To wrap up all the information, this service is mainly responsible for redirecting requests to the appropriate services. Still, it is also an important component in translating and delivering a more user-friendly response for upcoming requests, providing a universal message structure readable for a variety of devices that may be attached to the architecture.

4.2 Authentication Microservice

In the realm of software implementation, the integration of an authentication microservice emerges as a pivotal requirement. This specific architecture is no outlier to the need for an authentication method. Conversely, since the system handles multiple sensitive data, its usage is a necessity. This component divides itself into two parts: the service, responsible for producing the logic for validation and verification of the incoming requests, and a database, to maintain a list of authorized devices on the endpoint required.

Starting with the database, it is a Docker image of a PostgreSQL with a single table containing three columns: the device identifier, like 'es-sms-XX'; the MAC address of the WiFi module of the Raspberry Pi serving as a smart meter; and an API key, a previously unique generated string with at least 20 bytes, that is already embedded on the device. The device identifier was needed because, throughout the communication, a public parameter must be known to identify the connection. Since this ID is used everywhere publicly, there was no apparent harm in using it. Second, the choice of that MAC. When setting up the Raspberry, the Wi-Fi module is intentionally deactivated and since this value is not broadcasted anywhere for the network, it could be seen as an internal and private element of the device. To add another layer of trust, the API key is generated by the server and embedded on the device. All of this plus the message-sending timestamp forms a secret that is not easily retrievable during the communication.

This database is completely isolated from the network and the only way to reach it is via the service itself, which is also not available publicly. It is a pattern in all the microservices in this architecture. Every database is only accessible via its specific microservice, and, those microservices are only available for the other microservices of the API with the exception of the *Gateway*, which possess endpoints reachable from the outside network.

Focusing now on the microservice itself. As stated, almost all the microservices, excluding the *Gateway* and *Market*, are developed using NestJS and communicate via gRPC. gRPC enforces developers to explicitly define every communication method, including its input and output message types. These methods can be considered analogous to endpoints in traditional RESTful architectures but with the added benefit of strong typing and automatic code generation. The structure of these messages is defined in the .proto file. For this specific case, only one method was declared.

```
service AuthService{
        rpc Validate(ValidateRequest) returns (ValidateResponse) {}
}
message ValidateRequest{
        string authorization = 1;
        string timestamp = 2;
        string deviceid = 3;
}
message ValidateResponse{
        int32 status = 1;
        repeated string error = 2;
}
```

Listing 4.1: Protocol Buffer for Authentication Microservice.

The idea is to extract every one of these headers that should come with the request itself: “authorization”, “timestamp”, and “deviceid”. First, the service queries the database to retrieve the MAC address and API key related with the “deviceid” header. If none is found, the request is automatically unauthorized and blocked. On the contrary, if an entry appears, it is necessary to validate the next header, the “timestamp” one.

Here, it is necessary to keep in mind that a delay between the sent request and its arrival may occur. So, the server side cannot expect to be, for example, 2023-06-13T09:50:02.123Z, and that the request arrives exactly at that timestamp without a delay, even if it is minimal. Therefore, it was given a slight margin of +3 seconds from that timestamp (e.g., until 2023-06-13T09:50:05.123Z, the request is valid). However, it is necessary to contemplate the inverse. For any reason, the timestamp that arrived could be before the one it currently is. To prevent any unintended usage, but still contemplate any timestamp offset, a margin of -2 seconds from the current timestamp is also given. To summarize, a request arriving at 2023-06-13T09:50:02.123Z is valid if it arrives with a timestamp between 2023-06-13T09:50:00.123Z and 2023-06-13T09:50:05.123Z. The idea is to prevent any brute force attack by trying to crack the “authorization” header. Following, if the timestamp does not match, the request is blocked and unauthorized.

If it still complies with the established rules, the “authorization” header is verified next. This header contains a SHA-512 hash that was generated by appending the parameters in the following order: “deviceid” header; “timestamp” header; API key; and MAC address. The server does the same hashing using this specific order. If the final hash matches the one that came in the header, the request is authorized and redirected to its specific microservice by the *Gateway*. If not, the request is unauthorized and blocked.

At this stage, any unauthorized attempt is logged for later analysis.

4.3 Meters Microservice

In the context of the project, the *Meters* component is a crucial part of our architecture, responsible for storing measurements from various devices. However, that is not the case within the context of this document. Hence, only a brief overview is provided to elucidate the reader of its core functionalities. An InfluxDB time-series database was selected to store this data efficiently.

In InfluxDB, data is organized into 'buckets,' and the structure is flexible, primarily relying on 'field' and 'value' pairs. The essential fields for each device, such as Active Import, Active Export, Reactive Inductive, and Reactive Capacitive, are consistently present in every measurement. Time intervals are managed in 15-minute windows, which aligns with the requirements of net metering, a billing concept that consolidates electricity usage within these intervals.

While the database handles the storage of measurements, the request flow passes through the *Meters* microservice before reaching the database. More details on the *Meters* component can be found in Appendix A.

It is important to retain that *Meters* microservice redirects the device identifier regarding the newly stored measurement to the *Transactions* Microservice and returns to the requester any type of information provided by that service. If, for some reason, it could not reach *Transactions* or it does not function properly, an explanatory message is returned containing only the necessary information to understand what went wrong.

4.4 Market Microservice

This service operation is out of the scope of this document. However, the service is necessary for the operation of other mechanisms like the transactions between devices, where buyers and sellers are matched to produce generic pending payments to later be used and manipulated by other services, namely, the *Transactions* Microservice.

This microservice is developed using Python and has attached a MongoDB database where all the community information is stored. Comparatively with the other services described throughout the document, this service implements two applications. One application is for querying information, insecure by default since it only is used for showing information with no changes to data. The other has methods with the capability of changing internal information, like the state of each payment. This second approach, due to the possible manipulation of critical system information, was opted to implement a secure channel, that being a connection with SSL certificates. In this case, since the communication is between two services in the same internal network, there is no apparent need for the usage of a certificate authority. To bypass this, the certificate and its key reside inside the Market itself, while the other copy of the SSL certificate is located inside the transactions microservice because they are the only needed components for this communication.

The service exposes to the internal network, the methods defined in Appendix B. There are a number of arguments that could be used to query this service. However, the most important concept is that the matches (or pending payments) have different states, identified in the Appendix by “States”. When Market generates matches, they all have the “Created” state by default. There is also another state, the “NotPaid”, that also mentions that no payment action has been taken regarding that specific match. Those two states, when present on the arguments of the “MatchesFilter” object, internally change the state of every returned entry to “Sent”. It is the only indirect entry modification that does not occur explicitly. It is useful for maintaining control of the already requested entries.

As a response, it returns objects with the following structure:

```
{
    "timestamp": "2023-09-16T19:27:00Z",
    "buyerdID": "es-sms-XX",
    "sellerID": "es-sms-YY",
    "energy": 0.0034,
    "price": 0.08,
    "id": "64f84eb150e852c156fde834",
    "createdAt": "2023-08-19T14:30:00Z"
}
```

Code 11: Model object returned by Market Microservice.

Those are the necessary parameters for the other services handling their necessities and work around.

There is one more endpoint, “UpdateMatch”, whose name is self-explanatory. It accepts an array of objects with the “MatchID” of the previous match returned and its new state, like “Paid” or “Error”. In case of an “Error” a message should be attached, if its transaction was successful, the transaction identifier of the payment is attached to the object and sent back to the *Market* for storage.

4.5 Transactions Microservice

Among other aspects, dealing with the *Transactions* Microservice was one of the most controversial implementation ideas. Following the logic of DLTs being decentralized databases, there was an effort to remove traditional database implementations on this part. However, it proved to not be so linear. One of the aspects was the fees attached to the use of the DLT in general. The other was related to the interoperability between the *Market* and the *Transactions*. The Protocol Buffer file of the *Market* microservice describes that for payment processing there are two parties involved: the prosumer, or seller; and the consumer, or buyer. However, there is no direct association between the device identifier, supposed to be the buyer or seller identifier, and their wallet counterpart.

It was also not correct to implement non-fundamental logic on a microservice that should be as generic as possible in terms of what is supposed to be received and returned. Microservices should only carry the logic that they need to function. The only part interested in matching the peers’ identifier with their account identifier (a more user-friendly ID representing the wallet) is the *Transactions* microservice. So, knowing that at least a match between a device ID and a wallet must be saved, the *Transactions* microservice encompasses an SQLite3 database with a simple table containing two columns: “deviceID” and “accountID”. This proved to be enough to make the DLT object preparation fast and efficient.

By looking into Appendix C, it is possible to notice three available methods for interaction with this microservice. The first is “AddAccount”. When a device is deployed for the first time, there is no wallet attached. Therefore, it cannot communicate with anything instantiated in the DLT. Besides, the database also does not have any row corresponding to this device. So, when the device is installed, it makes a POST request to the gateway that is redirected to this method. This request contains the public key of the wallet inside the device and its identifier.

Internally, the service passes through multiple iterations.

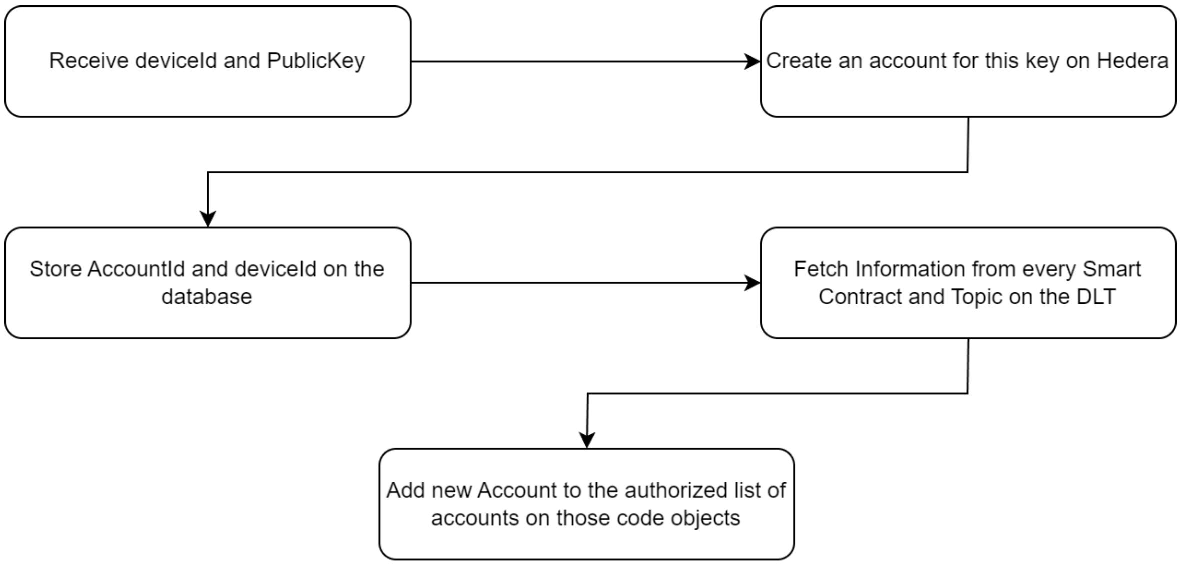

Receive deviceId and PublicKey
Create an account for this key on Hedera
Store AccountId and deviceId on the database
Fetch Information from every Smart Contract and Topic on the DLT
Add new Account to the authorized list of accounts on those code objects

Figure 4.1: Account addition iterations.

After receiving the public key and the device identifier, the service makes a request to the DLT to create an account, whose public key is the one provided in the request. By default, the mechanism of creating this account also adds a balance of 10 euros in Hashbar, the cryptocurrency of the Hedera Hashgraph. If everything goes accordingly, the result of this request is an “AccountId”, a shorter identifier of the wallet. To give context, Hashgraph has three important components for each wallet, a private key, which is a secret code for authorizing actions on the wallet; a public key, something similar to the IBAN of a traditional bank account; and the “AccountId”, a shorter representation of the object that is the wallet on the blockchain. This wallet identifier, alongside the device identifier, constitutes a row entry on the database.

Next, the service fetches the Topic hosted on the DLT and retrieves a list of every public key authorized to write on it. It adds the newly created account and updates the Topic. On the smart contracts, the process is similar. When built, inside the solidity code of the contract, an array of allowed devices on each function was created. Therefore, the service also gets this array and, by using an owner-specific implemented method, that is the account proprietary of the microservice itself, it can add or remove elements. In this case, it adds the new wallet into it, making the device able to interact with that piece of code on the blockchain finishing the iterations of this method. The process is equal in both the smart contract implementations.

"GetPayments" is the method referred to in Section 4.3, where the *Meters* Microservice redirects the device identifier to this service. When an identifier is received, the first action is to get every match on the *Market* database, where the device is a consumer, also referred to as the buyer, and where its internal state is "Created" or "NotPaid" (as mentioned in Section 4.4). The buyer and seller identifiers are then changed for their wallet counterpart and the price is then converted from Euros to Hashbar. After this, there are multiple approaches, the ones referred to in Section 3.3 If the approach being used is the "Direct Transaction" the JSON objects are just sent back as a response and every process of converting JSON into Transaction Objects readable by the DLT is made on the device side. Opting for "Freeze Transaction", the process is made also on this microservice, leaving those transactions pre-prepared just waiting to be executed by the device and not changeable from that point on. Finally, the smart contracts approaches are based on "Direct Transactions" but, instead of issuing a transaction object to the blockchain, the device must communicate to the contract in use and transfer the funds plus associated information.

"AddTxReceipts" is the last method that needs to be addressed. It appears from the necessity of notifying the API that some payment has been issued and needs to be checked by the service responsible for the payment verifications: *Transactions*. So, it accepts as arguments an array of objects containing the transaction identifier, that is the address to where the register of that transaction lies on the blockchain; and the payment identifier that it corresponds to. This endpoint could be entirely optional, depending on which approach is being used and if the community management administration wants to implement the reception of the identifiers via an endpoint or via the blockchain itself. Using the blockchain, the alternative is to use the topic.

The topic is a historical register of messages sent to a specific address reserved and monitored, in this case, by the wallet possessed by the microservice. Therefore, only the previously allowed devices, via the "AddAccount" method can submit a message to that address. Those messages follow a message structure: first comes the match identifiers followed by a "-" and the transaction address. By default, if this approach is being used when the *Transactions* service starts, it automatically creates a continuous channel to that address and reacts to any change happening in terms of incoming new messages (The sender is implied on the communication since its identification is embedded in the object Hashgraph creates).

Both of these methods then follow the same procedure. They query the *Market* service for that incoming match identifier and start to make a list of assertions. They check the following: if the transaction was successful and accepted by the DLT: if the buyer and seller wallets match the devices involved; and if the amount transferred equals the intended amount in Euros. Then, the transaction identifier is attached to that match identifier and sent back to the *Market* service acknowledging its authenticity.

Smart contracts differ from those previous steps. In this case, all verifications still apply but the way they come into play is slightly different. Here, the service periodically calls a smart contract function that holds the records of the payments that were issued but are still pending approval, either by the REC, if the contract used is the REC-Ruled Contract, Figure 3.4, or by the seller device (same as the prosumer), if using Peer-Ruled Contract, Figure 3.5.

4.6 Device Transaction Components

To face all possibilities of the architecture introduced in from Section 3.4.2, the device assembled is a Raspberry Pi 4 with a 5G Module attached running RasbpianOS. The scope of the document is focused on getting transactions executed. Therefore, the concepts of reading from the electricity meter and how to manipulate that information are out of scope. That leaves two main aspects worth contemplating and describing: the process of storing the incoming payments and all components responsible for interacting with the Hedera Hashgraph.

To ensure smooth operations, a clear demarcation was established between the measurement collection and payment handling processes. Despite receiving pending payments via HTTP POST requests for measurements, handling these incoming payments does not necessarily occur sequentially. Consequently, the device is not compelled to commit payments immediately upon receipt. This flexibility is essential, as it accommodates scenarios such as insufficient funds in the wallet.

When developing the device, the storage choice was MariaDB. MariaDB was not the only alternative, SQLite3 was also contemplated. Throughout the development, SQLite3 proved to be a hard choice to maintain, due to package incompatibilities from the different scripts running on the device. So, every device contains a MariaDB with a database “Payments”. This database contains two tables “json” and “tx”, the first one stores json entries while the other corresponds to the byte array ones.

Each row of the JSON table has the following structure:

- Id - A row identifier
- Identifier - The payment identifier of the pending payment
- Buyer - The device wallet that must transfer funds
- Seller - The wallet for the funds to be transferred to
- Energy - The quantity of energy that is being paid
- Price - The quantity of currency that must be transferred to the seller
- Timestamp - The interval to which the payment corresponds to

Since most of this information is present on the byte array, its table is composed by:

- Id - A row identifier
- Tx - The byte array of the transaction object
- Timestamp - The interval to which the payment corresponds to

Following a clean deployment, the device lacks an existing wallet. To create one, the device initiates a TypeScript method that utilizes the Hashgraph SDK to generate a Key Pair consisting of a Private and Public Key. The Public Key is then transmitted to the API, which, in turn, communicates with the *Transactions* microservice to execute the “Account Creation” process within the ledger. Upon successful execution, the API internally stores the pair (deviceID, AccountID), enabling the device to engage in payment transactions.

On the device side, the Private key is then saved on a “store.key” on the “/home/root/” directory after being encoded into Base-64. Saving the private key is the only requirement because it is possible to derive the public from the private one. After that, if it is necessary to retrieve the “AccountId”, a simple curl to a Hedera node API is enough to get it by using the public key.

An additional TypeScript program was created to query the database for pending payments. Using the Hashgraph SDK, this script is periodically called via a cronjob, fetches every row of the tables (one table at a time), and processes the transaction accordingly. Dealing with the JSON objects, there is a method responsible for manipulating the data and creating the object before sending it to the DLT. For already created objects, the device should only sign and issue them to Hedera Hashgraph.

The response to these actions yields a “receipt” containing transaction details and its identifier. The database, equipped with a table that correlates the payment identifier received via POST with the transaction identifier in the DLT, stores this new information.

In an asynchronous fashion, the device sends this list to the API for internal validation and subsequently empties the database table to optimize storage capacity. This method can be achieved via two alternatives. The first is a POST for the API, where the device sends a pair of “PaymentId” and “TransactionId” and the other is via topic. This topic serves as a “mailbox” for authorized wallets. So, the device sends a message to the topic and the server is able to extract new messages and process their content.

If smart contracts are in use, the first part of storing the incoming payments is similar to a Direct transaction, JSON objects are stored in the same table. However, its handling differs. Instead of creating a Transaction object readable by the DLT, it instead calls a script that interacts with the deployed smart contract and transfers its funds to it. At this point, the device does not have any other responsibility for this interaction, if it is purely a buyer. In the case of it being a producer and the Peers-Ruled Contract is the go-to methodology, two more steps come into place. Firstly, the device makes a call to the contract to receive the pending payments and associated values. After receiving them, a validation is made on the device end that then sends both an array of accepted and rejected identifiers back to the contract. The contract is then responsible for transferring the funds accordingly.

Talking about the other contract, REC-Ruled Contract differs from this only on the validation part. Instead of being the prosumer devices that execute this validation, it is the API itself that is responsible for those steps.

4.7 Hedera Hashgraph Components

This section provides a detailed overview of how various stakeholders, including device owners and administrators, can interact with the smart contracts deployed as well as the REC topic. Across all these components device management is a fundamental aspect, allowing the definition of authorized entities and access control and monitoring of the actions.

4.7.1 REC-Ruled Smart Contract Interaction

Authorized users, typically REC management administrators, can grant access to multiple devices at once using the “insertDevices” function. This function streamlines the process of allowing multiple devices to participate in the ecosystem. Alternatively, a device owner can grant access to a single device by utilizing the “insertDevice” function. Each authorized device is identified by its unique public key. The contract owner also has the capability to revoke access from previously authorized devices through the “removeDevice” function. This function enhances security and access control by allowing administrators to manage device permissions effectively.

The REC-Ruled smart contract facilitates the issuance of payments between authorized devices. Device owners can initiate payments by specifying the recipient, payment amount, and every other information available on the database table. Device owners can use the “issue-Payment” function to send payments to other authorized devices. Payments are characterized by the quantity of cryptocurrency transferred, the sender’s address, and the recipient’s address. The smart contract keeps track of these payments in a pending state, awaiting further action.

These pending payments are stored in the smart contract until they are confirmed or denied. This temporary storage is essential for ensuring the orderly execution of payments:

To manage pending payments, the owner can use the “retrievePending” function. This function retrieves the pending payment IDs, while also locking pending payments to prevent any conflicting actions. This is crucial for maintaining data consistency. In this specific contract, REC administrators play a pivotal role in confirming or denying pending payments, thus determining whether payments are executed or refunded. To do so, the owner of the contract can use the “confirmPayment” function to either confirm or deny specific pending payments. Confirming payments results in the transfer of cryptocurrency to the intended recipient, while denying payments triggers a refund to the sender’s address. This function helps maintain the integrity and security of the payment process.

Finally, the device can initiate fund withdrawals using the “withdrawFunds” function. This function allows them to access the funds they have accumulated within the contract, and the corresponding cryptocurrency quantity is transferred to their address.

Across all the smart contracts, there is a function available only for the owner itself, ensuring that the contract is completely controlled by the correct entity while there are functions that require authorized accounts to be accessed. In any case, non-authorized accounts cannot interact with any code of the contract besides reading it on the ledger.

4.7.2 Peers-Ruled Smart Contract Interaction

This contract is very similar to the previous explanation. The only major change relies on the interactions of the REC administrators and the seller devices. In this case, the contract owners have an over-watch position, just serving as a monitoring mechanism of the contract interactions. The responsibility for verification and authorization of the payments is a seller device function. When this contract is in place, the devices not only receive the payments that they are supposed to emit but also the ones that other devices must pay to them. With this information, the seller's device can fetch its corresponding pending payments and validate them on the contract. If valid, the funds are transferred to their account. If not, the payment is refunded to its original owner to be later reissued. At every stage, the REC API is constantly verifying the contract. So, it is always aware of these exchanges to update its internal state (paid or not paid).

4.7.3 Topic Interaction

The topic also possesses an access control in which only authorized wallets can write to this address. As stated, the topic could be seen as a "mailbox", but, since the interaction is made using devices, it is convenient to have a message structure. To facilitate, messages corresponding to payments are simple and intuitive for the server to analyze them, " 123456 - 0.0.829465@1695745800.195571337". The first part is the id of the issued payment followed by a separator. The second part is the transaction address corresponding to the payment. Topics also have the particularity of being able to be listened to. Therefore, the server is notified of changes and parses those values. This comes as an alternative to sending a request to the API directly while maintaining a previous history alongside other possibly useful details for auditing in the future.

4.7.4 Hashgraph Local Node

The Hedera Hashgraph Local Node is a critical tool that replicates the Hedera network locally, enabling developers to test and fine-tune applications and smart contracts in a controlled environment. It provides real-time data access and configuration options for simulating different network conditions. This tool is ideal for secure development, ensuring applications are thoroughly tested before deployment on the public Hedera network. It also supports private network deployment and integrates seamlessly with Hedera's software development kits, enhancing the development and testing process. The node can run on multiple configurations, either by replicating the mainnet or testnet behaviors, or simply by having a local mode where everything is speed up and throttling limits are removed. In this work, the node was deployed and configured as a replication of the Hedera Hashgraph's testnet, despite being sometimes unstable [63].

Every component was deployed on the official testnet and was replicated on the local node for various debugging effects and comparisons.

Another useful thing about using a local node is that, by default, there are docker images that help visualize node data, such as Graphana.

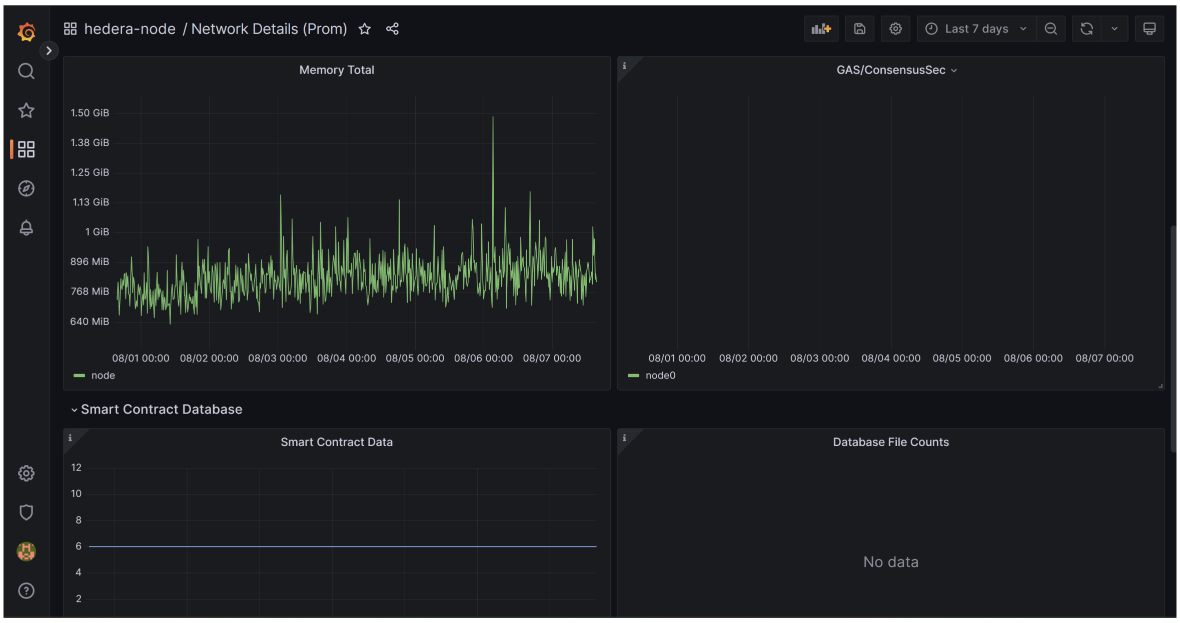


Figure 4.2: Node Statistics shown by Graphana.

In this figure, the node owner can visualize the memory its node is consuming on the Virtual Machine as well as the gas used and smart contract-related info. For example, the number of smart contracts on the node and the number of files residing there. These are some of the metrics since there are a lot more to be explored if needed.

CHAPTER 5

Results

This chapter is dedicated to evaluating a REC built upon Hedera Hashgraph, focusing on two integral components: transaction object alternatives and smart contracts. Those components were developed to provide automatic transactions in a decentralized form, like the Direct and Freeze transactions, while the smart contracts were developed in an attempt to find a more robust alternative to these transactions at the same time they contemplate other transaction aspects like automatic refunds and automated server notification of paid payments without increasing the costs of operation significantly. In the subsequent sections, a thorough analysis of the results obtained from the evaluation of every different currency exchange alternative is presented. These results encompass key metrics such as transaction processing times, reliability, and costs, providing a holistic view of the performance of the Hedera Hashgraph solution in the context of a REC.

This chapter aims to inform stakeholders, researchers, and industry professionals about the viability and implications of employing Hedera Hashgraph in building sustainable, efficient, and secure RECs.

5.1 Data Collection and Methodology

In this section, the goal is to elucidate the methods employed to gather and filter data pertinent to the evaluation of a REC built on Hedera Hashgraph. Therefore, the research is guided by the principles of precision, consistency, and transparency, aimed at ensuring the reliability and validity of the data acquired.

To get consistent metrics across the different strategy implementations, it is necessary to work with the same dataset on all of them. That said, since this document is a specific part of a global project, this data was needed and imported from another component, the *Market* Microservice 4.4. The service gathered a week of data exchange between two community members, one merely a consumer and the other with the capacity to produce energy surplus at different times of the day. Using an internal matching algorithm, the components of this work remain agnostic, which means they are not directly influenced or dependent on the matching algorithm used by the *Market*. The *Market* microservice provides a list of objects similar to Code 11.

The data used encompasses the period from 2023-09-19 18:15 until 2023-09-27 18:30, giving 198 intervals of 15 minutes in which an energy exchange between these two peers occurred.

5.2 Object-based Transactions Results

This category groups both Direct and Freeze transactions. They are very similar, in terms of what they can do and the fees associated with them. The difference resides on the processing side.

In Direct transaction mode, all the transaction processing and execution is done on the device side. It receives a JSON containing every previous pending payment to create the objects locally and execute them on the DLT. Freeze transactions differ from this approach because, instead of letting the device freely modify and generate the incoming transactions, it is the server that creates the object, blocks it, and sends it to the device, giving only the possibility of signing it and executing it on Hedera Hashgraph or discard it.

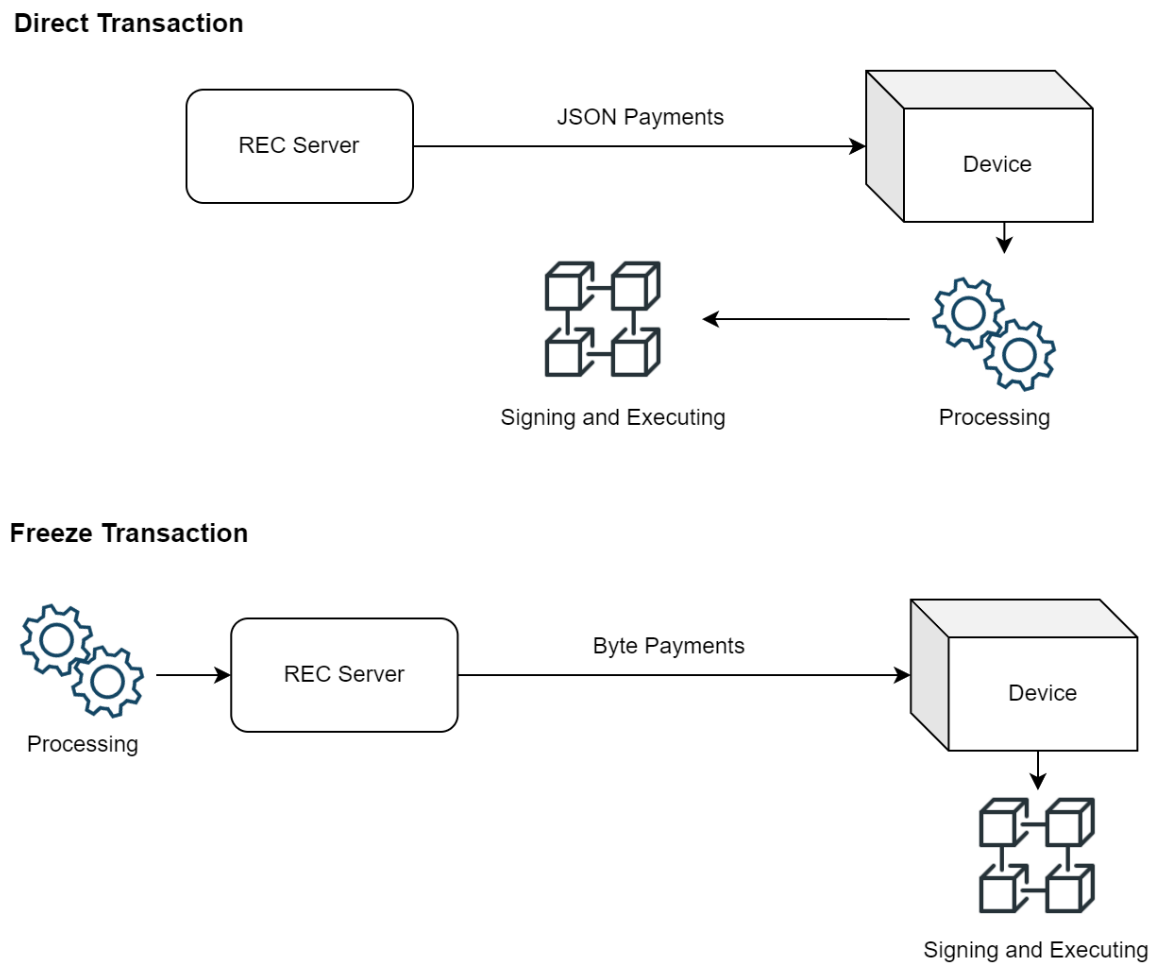


Figure 5.1: Direct and Freeze transactions flow.

The usage of Local Node was only used on these two approaches since they are equal in terms of objects exchanged between devices and nodes.

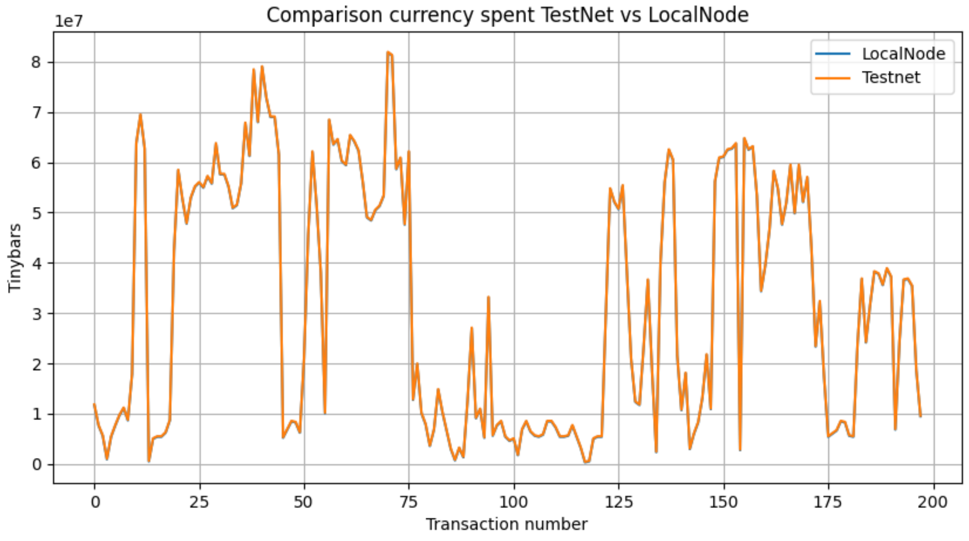


Figure 5.2: Comparison of currency spent on TestNet and LocalNode.

At first glance at Figure 5.2, the difference is imperceptible. According to assumptions, this should come as evident, since a possible variation should only come in the form of fees.

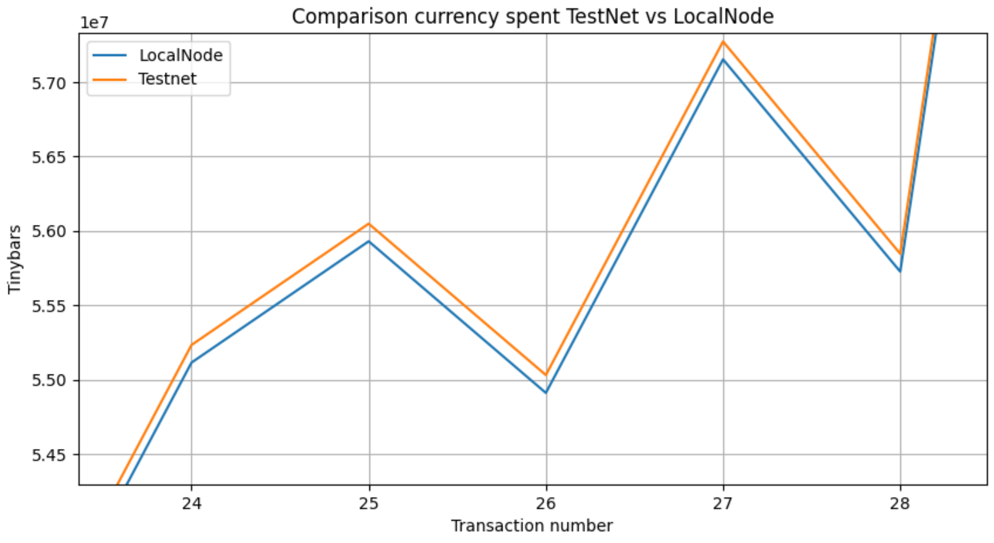


Figure 5.3: Comparison of currency spent on TestNet and LocalNode (zoomed).

A closer look at Figure 5.3 corroborates that assumption, despite those values having to be multiplied by 10^7. To make this clear, the base unit, Hbar, is equal to 100 million Tinybars, a value that, normally, stays between 0.04€ and 0.06€.

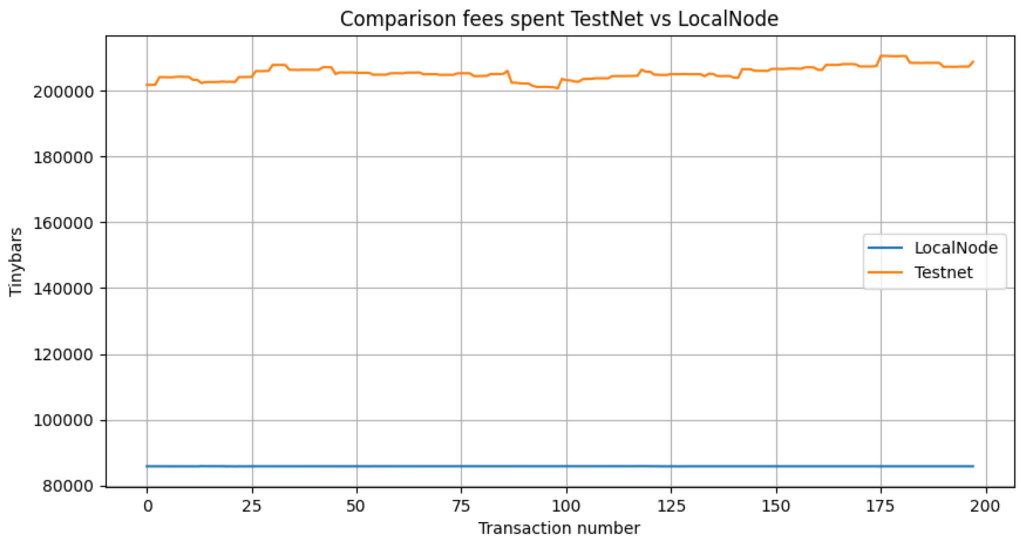


Figure 5.4: Comparison of currency spent on TestNet and LocalNode (zoomed).

The margin represented on this chart is not substantial, provided that the average fee on the local node is 85812 Tinybars compared to the average of 205417 Tinybars. Using the Euro as the fiat currency, this difference is equivalent to ≈ 0.006 cents while the highest value, the average fee of the TestNet is ≈ 0.01 cent. These values were obtained considering 1 Hbar equals 0.05 Euros.

5.2.1 Topic usage

When implementing the Topic as a form of notifying the server that a transaction has been processed, the base cost of the operation just increments with a static value of 208653 Tinybars, which corresponds approximately to 0.0001 EUR. At this point, since the fee variation was not significant throughout that week, the cost of writing a topic message was considered a fixed value, the 208653 Tinybars, and this cost is added on top of each transaction executed, like the ones in figure 5.2. The message used for this metric was the same that is represented in Section 4.7.3.

5.2.2 Comparison

To compare these approaches, it is not possible to simply look at the charts and opt for an alternative because the operation costs on the ledger for these two implementations are the same. It is necessary to look back to Figure 5.1. The real comparison in those two scenarios relies on the security and effectiveness of the approach.

Another quantitative metric can be considered: the amount of byte transmissions that occurred between the server and the device itself. While the Direct transactions are sent using JSON, the Freeze Transactions uses byte arrays for its transmission. The values provided in Table 5.1 use the exact object provided in Code 11.

	Direct Transaction	Freeze Transaction
Transmission Size (Bytes)	1461	11882

Table 5.1: Byte size comparison between objects.

Measuring the byte transmissions in those two approaches, it is noticeable the difference between the Direct and Freeze approach. The value provided by Direct Transaction is around 87% smaller than the Freeze Transaction.

It is necessary to also look for qualitative metrics. In terms of security, Direct Transactions let the device be completely autonomous in the object creation and its issuing on the ledger. This could lead to unintended behavior such as errors in payment processing or the device could be tampered with. For this reason, Freeze could be useful, because, in this case, the transaction is created internally by the API. Therefore, any errors that may occur and only dependent on the API operation and may be corrected at later stages.

5.3 Smart Contract Results

When looking at the behavior of these alternatives, other problems besides tampering arise, like the refund process and the handling of incorrect payments. For that reason, having a "third party" in this process can be useful: smart contracts.

Smart contracts imply usage costs and although the REC manager must overwatch the contract, it could be expensive. So, in the REC-Ruled Contract, the manager has an active role in accepting and denying payments as well as validating them in the API. The Peers-Ruled Contract assumes a more passive role of just fetching information from the contract and updating the API's internal state. For that reason, devices have to intervene more in this exchange, especially the seller device, because it is responsible for validating payments whose destination is its wallet.

To have a term of comparison, the same-week values from Section 5.2 were used in the smart contracts.

5.3.1 Peers-Ruled Contract

As previously stated, the Peers-Ruled Contract gives autonomy to the devices to manage the payments either by accepting or refusing them from an intermediary wallet held by the contract. Here, devices can perform various actions. The most relevant of them are issuing payments, retrieving the pending identifiers and their associated values, and validating them by transferring them to the recipient or refunding them.

Manager Costs

For this contract, the costs incurred by the manager vary. A value of 1188254730 Tinybars is necessary for the contract deployment ($\approx$ 0.60 Euros). Then, to make the device accounts able to interact with the contract, the manager must make a call to the contract which costs 14320000 Tinybars for each device that must be allowed on the contract. For this use case, only two devices were added. The only cost that is omitted in this part of the results is the contract polling costs. Since the manager plays the role of overwatch, the frequency with which he checks the status of the contract, can be even zero. This is because since both devices receive payments where they are the buyers and sellers and the information is provided by the server, the odds of manipulation or errors on both ends are low. Assuming the server issued a payment from "A" to "B" of 1 Hbar with the identifier "123", it is sent to both devices while maintaining a copy on an API database. Even if "A" issues a payment of 0.5 Hbar, "B" will not accept it, because it was expected to receive a 1 Hbar payment for the identifier "123".

	Contract Deployment	Device Addition (x2)	Total Cost
Manager (Tinybars)	1188254730	28640000	1216894730

Table 5.2: Fixed costs of running peers contract (manager).

Table 5.2 summarizes all the information. Note that, in this case, the poll costs of the contract are omitted as they may vary due to the size of the object retrieved (an array) and the number of operations performed in the contract.

Seller Costs

In this case, the seller, besides receiving the profit from his sold energy surplus, also has operation costs associated with the validation of the incoming payments. Those costs came from querying the contract and validating the payments, therefore these operation costs cannot be neglected and need to be evaluated.

There is no simple way of showing the temporal costs of interacting with the contract. Contrary to object-based transactions where the sellers do not have to make any call to the DLT, in this case, they have to validate their incoming funds.

For this test, the seller's wallet had 200 Hbar. It is also considered that the seller is expected to receive, throughout the week, $\approx$ 64 Hbar. It is also important to note that, in this week, between those two peers, only on 215 periods did an exchange occur for a total of 672 periods in a week.

Every period that the seller receives pending payments whose destination is its wallet, the contract must be called to check for those specific identifiers. This response may come as an empty array, meaning that no payment is yet waiting for validation, or it can have one or more identifiers. In this specific case, at every request, only one identifier was available at each moment. An average of 5728000 Hbar was the required amount for this calling.

If an identifier is provided, the seller then calls the corresponding objects from another call and validates them. These are both individual methods. Therefore, the seller must make two additional calls. Since those calls involved transferring funds from the contract to the seller, it is not possible to estimate the cost implied on this.

At the end of the test, the wallet balance of the seller was $\approx$ 215.67 Hbar, receiving only 24,48% of the supposed profit.

For statistical purposes, the seller itself issued 645 transactions consisting of interacting with the contract. However, there were other operations indirectly associated with the seller, such as the refund issued by the contract.

	Initial Balance	Expected Profit	Final Balance	Profit Loss	Calls
Seller	200 Hbar	64 Hbar	215.67 Hbar	-75,52%	645

Table 5.3: Seller's balance overview (Peers-Ruled Contract).

Buyer Costs

The most costly part of the contract is on the buyer's end. Besides the payments that the buyer needs to pay, it also has an increase in the operation cost, due to the necessity of interacting with the smart contract. Therefore, the total cost of the operation for the buyer consists of the payment issue plus the contract interaction costs. The buyer also does not interact with the contract periodically, it only calls the intended method when a new payment is received from the server.

At the start of the week, the buyer's wallet had 200 Hbar. Since it is supposed to pay $\approx$ 64 Hbar at the end of the week, it is estimated that the wallet's final balance should be around 136 Hbar, neglecting fees.

To this end, only one call was made per period with pending payment, summing up 215 calls to the contract.

At the end of the test, the buyer wallet had $\approx$ 88.85 Hbar, an increase of 47,15 Hbar in the total cost of the operation.

	Initial Balance	Expected Expense	Final Balance	Real Expense	Calls
Buyer	200 Hbar	64 Hbar	88,85 Hbar	111,15 Hbar	215

Table 5.4: Buyer's balance overview (Peers-Ruled Contract).

Overview

Figure 5.5 shows a temporal evolution of the wallet balance of both the seller and the buyer. As was explained, the seller had many transactions compared to the buyer. To make both lines represent the wallet state in a temporal form, the buyer's wallet value at a given time was expanded until both wallets started to deal with a new payment.

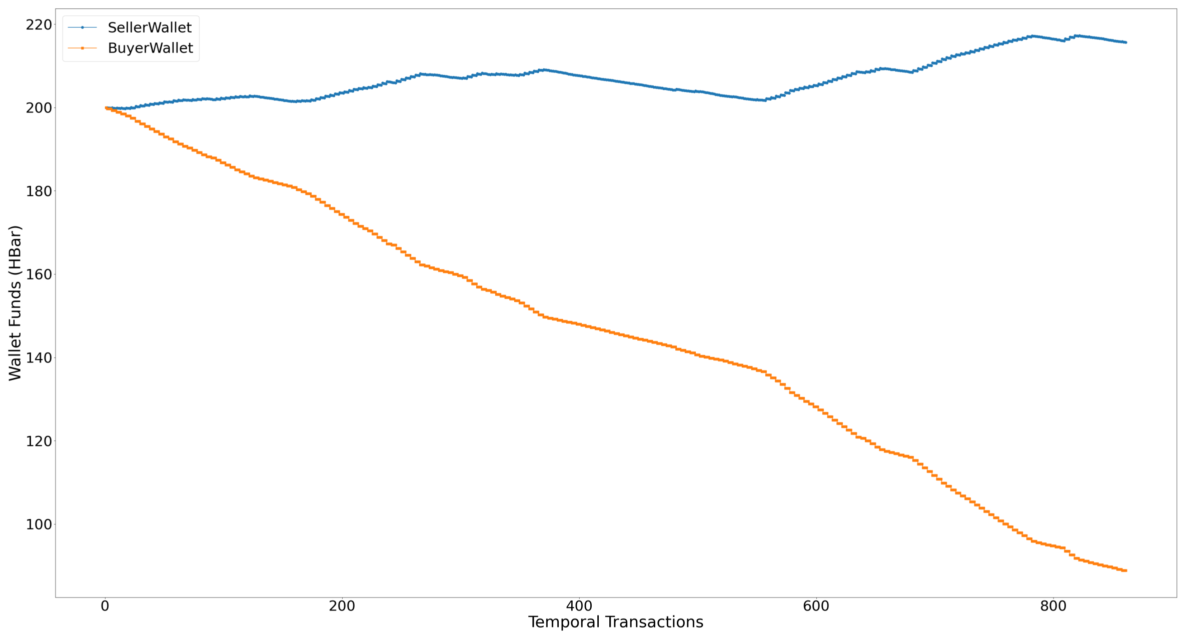


Figure 5.5: Wallets' balance overview (Peers-Ruled Contract).

Analyzing the figure 5.5 it already shows an increase in operation costs compared to the Direct and Freeze transactions, but it was expected since the smart contract has costs associated with its functioning. It is also clear that the seller has a cut in its profit, as shown in table 5.3. However, this solution does not need a third-element interaction in the validation of the payment, therefore, it provides a decentralized form of energy exchange and cryptocurrency transfer.

Besides monetary costs, since the project deals with IoT devices, metrics related to network traffic were also acquired throughout the operation.

	Download Bytes	Upload Bytes
Buyer	0.38 MB	0.50 MB
Seller	1.06 MB	1.32 MB

Table 5.5: Bytes statistics overview (Peers-Ruled Contract).

Table 5.5 demonstrates that, in 215 periods of energy exchanges in a week, the network traffic that occurred is not very intensive and, when compared to the processing and authenticity of the transactions that are made by the smart contract is a relatively minimal cost.

5.3.2 REC-Ruled Contract Results

On the REC-Ruled Contract, the buyer's duty remains the same. Its function in the whole process still consists of making the due payments into the contract using the same methodology as previously. However, the validation part of the process relies on the organization, in this case, the REC manager. This alternative is not so "decentralized" as the previous one, because it involves a "third" party to confirm the actions in course. Since this responsibility is passed to this entity, the seller does not owe any charge in this process.

Manager Costs

This time, the Manager accumulates two different costs. The first one is similar to the other contract, having the responsibility of deploying and authorizing the peers to interact with the contract, which has a total cost of $\approx 0,62$ Euros.

	Contract Deployment	Device Addition (x2)	Total Cost
Manager (Tinybars)	1218895545	28640000	1247535545

Table 5.6: Fixed costs of running REC contract (manager).

Table 5.6 presents very similar costs of deployments as table 5.2. It is expected since the contract possesses almost the same features and code length, being the only difference the user authorization with certain methods.

Despite these charges, the manager accumulates the charges of the seller. If, for Peers-Ruled Contract, the seller had an expected profit, in this case, everything that the manager does with the contract finishes in debt.

	Initial Balance	Final Balance	Costs	Calls
Manager	200 Hbar	150.48 Hbar	49.52 Hbar	645

Table 5.7: Manager's balance overview (REC-Ruled Contract).

So, in a week of energy exchange between two peers, for validation, the manager incurs a cost of $\approx 2,48$ Euros.

Buyer and Seller Costs

For the buyer, there are no differences in terms of contract costs or currency sent. Therefore, all the claims of the Peer-Ruled Contract remain valid. Every value used in Table 5.4 is the same in this case.

For the seller, as explained, since there is no cost of operation because it does not have to interact with anything, the expected profit and what is reflected on the wallet are the same. Hence, if it had to earn 64 Hbar from the energy surplus sold, all that quantity is available to it.

Overview

Figure 5.6 shows the evolution of the manager's and buyer's wallets throughout the week. Again, for visualization purposes, since the manager in this case had a larger amount of transactions, the buyers' wallet value was expanded.

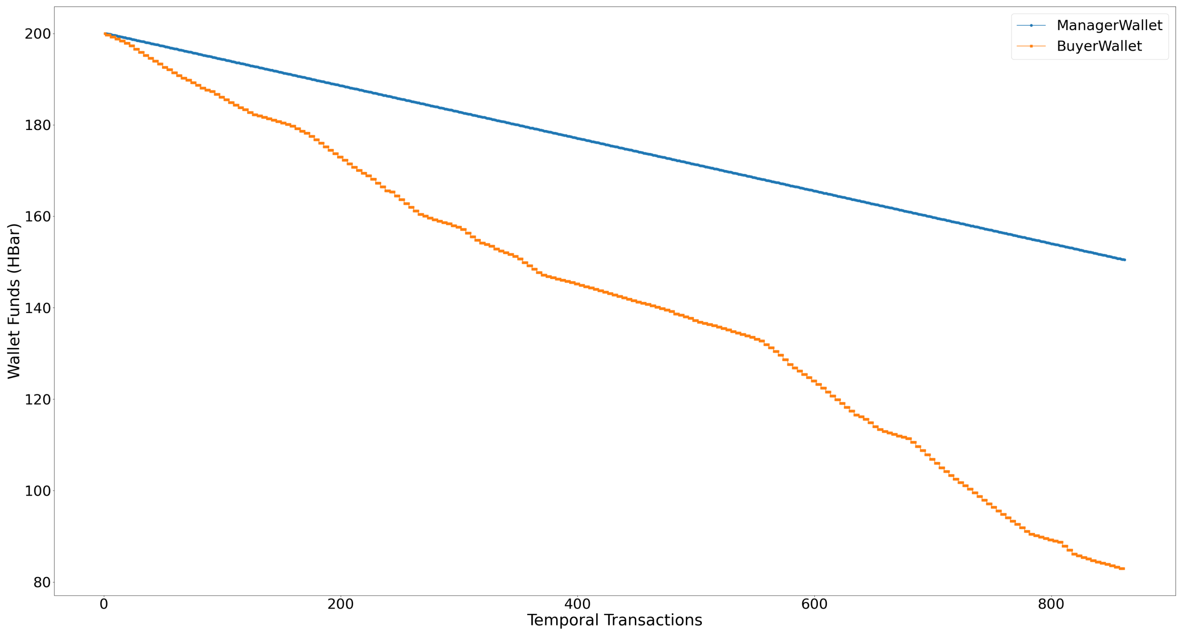


Figure 5.6: Wallets' balance overview (REC-Ruled Contract).

	Download Bytes	Upload Bytes
Buyer	0.38 MB	0.46 MB
Manager	1.07 MB	1.33 MB

Table 5.8: Bytes statistics overview (REC-Ruled Contract).

5.3.3 Byte Comparison between Alternatives

The smart contract tests allowed to separate the behavior of the buyer and the seller. However, it is necessary to keep in mind that it is possible that buildings are purely consumers. Furthermore, no building is purely a producer, therefore, although a distinction was made between buyers and sellers, the costs (either monetary or byte costs) mix up, increasing both costs for each peer. Table 5.9 represents the different approaches and their network traffic to contemplate the week.

	Buyer Upl.	Buyer Dwl.	Sellers Upl.	Sellers Dwl.
Direct Transaction	0.21 MB	0.11 MB	N/A	N/A
Freeze Transaction	0.21 MB	0.11 MB	N/A	N/A
Direct + Topic	0.51 MB	0.14 MB	N/A	N/A
Freeze + Topic	0.51 MB	0.14 MB	N/A	N/A
Peers Contract	0.50 MB	0.38 MB	1.32 MB	1.06 MB
Rec Contract	0.46 MB	0.38 MB	N/A	N/A

Table 5.9: Byte operation cost for every approach.

As shown in table 5.9, Direct and Freeze transaction methods, without topic, have the least network traffic of all the alternatives. However, when implementing the topic, their costs increase to values around the same cost as the smart contract approaches, 0.51MB to 0.50M for Peers contract and 0.46MB for REC contract, looking for the buyer upload bandwidth.

For the download bandwidth, the topic alternatives have half of the MB consumption when compared to the smart contracts. Another particularity is that only on Peers contract, the seller has bandwidth costs, due to the requirement of payment validation.

5.3.4 Cost Comparison of Alternatives

Similar to the byte comparison section, instead of network traffic metrics, Table 5.10 displays each of the alternatives alongside the costs incurred for each peer, the buyer, the seller, and, when available, the REC manager account. The amount of Hbars expected to be transferred between the buyer and the seller, in every alternative, is 64 Hbars.

	Buyer Expense	Seller Profit	REC Expense
Direct Transaction	64.42 Hbar	64 Hbar	N/A
Freeze Transaction	64.42 Hbar	64 Hbar	N/A
Direct + Topic	64.42 Hbar	64 Hbar	N/A
Freeze + Topic	64.42 Hbar	64 Hbar	N/A
Peers Contract	111.15 Hbar	15.67 Hbar	N/A
REC Contract	111.15 Hbar	64 Hbar	49.52 Hbar

Table 5.10: Costs of the alternatives.

5.4 Discussion

Analyzing the entire obtained results, an immediate statement can be made: if the idea is to provide real-time execution, smart contracts are costly. Comparing the Direct and Freeze transactions, both of them have the same currency and network traffic costs. The only difference relies on the security of the process, that is, in the connection between the API and the device. For that reason, although the devices are IoT devices with a possible isolated module for network communication, which by itself has costs associated with the SIM card plan, the difference is an increase of ≈ 1000 bytes and a baseline security against transaction tampering. For that reason, since everything is the same, it is preferable to rely on Freeze Transactions.

In terms of how should the API be notified, there were two variants, a normal POST request and the usage of a Topic. Despite Table 5.9 showing network statistics for the REC, they could be neglected. That is possible due to the API being a server and not an IoT device. Therefore, the network connection is assured via Ethernet. For this reason, opening a permanent channel of communication to the DLT is not problematic. The increase of nearly 0.30 Megabytes is not significant besides the cost shown in Section 5.2.1. Using the topic also guarantees an immutable temporal message registry, which is also beneficial for later auditing and comparing with the API databases. On the contrary, sending a POST request only assures its storage on the API. For this reason, merging both statements, when comparing object-based transactions, Freeze Transactions with the usage of the topic functionality from Hedera Hashgraph is preferable.

It is also important to not neglect why the smart contracts occurred in this scenario. Having a form of refunding and a form of notifying the API with authenticity is not easily achievable, therefore, using smart contracts can help because its operation, if correctly coded, is trustable and the records it originates are immutable. The first smart contract evaluated, Peers-Ruled, was the most convenient way of using contracts for this scenario. It provided a decentralized way of validation and auditing without the REC manager being directly involved. For this reason, it is preferable in comparison to the REC-Ruled one, where the manager must have an active role in maintaining the integrity of the exchanges. However, this maintenance is costly and, if the manager is responsible for it, a number of costs arise without any profits. This is the big difference between those two approaches. While in Peers-Ruled the costs are the sellers' responsibility and are deducted from the total profit of selling surplus energy, in REC-Ruled it is just a non-refundable expense for the manager entity. For this reason, Peers-Ruled is the most wise choice.

When putting side-by-side the Peers-Ruled contract and the Freeze transaction with the topic, both in bandwidth and operation costs, the expenses are higher for the sellers. They went from having nothing to pay or communicate, on freeze transactions, to having costs associated with this validation, on the Peers-Ruled contract. The idea is to have a real-time execution of both processing measurements and executing the money exchanges. This is where Peers-Ruled cannot compete with the Freeze Transactions. The contract could be a more appealing choice if, for instance, instead of calling the contract every 15-minute period, the seller queried it after a large period, for example, a week. With this, since more payments are going to be stored on the contract and its validation comes from only one call, the operation costs paid for these actions are greatly reduced, because each interaction is not one call for one payment, but becomes one call for an array of payments.

To summarize, the idea of the document is to produce a fully functional REC using a DLT as a foundation for currency exchanges associated with energy trading in a real-time situation, the Freeze Transactions with the usage of the topic is the best alternative for this use case.

CHAPTER 6

Conclusion

Throughout the document, microservices were described, the used technologies, how they communicate, and the limitations that occurred alongside those choices. All of this had the goal of achieving a fully functional REC with an innovative technology such as DLTs, that could be more than a concept and be used on a daily basis for a community.

This work proposes the use of a DLT as an underlying element of a REC by encouraging prosumers to produce their own energy and share it with a community. In this sense, the ledger comes as a form of maintaining a registry of everything that occurs between the exchanges of these community members.

This document shows a new alternative to the common usage of Ethereum and Blockchain, Hedera Hashgraph, with substantially more transaction throughput and lower associated costs. This enables the community to perform energy exchanges in real-time, allowing the members to know the origin of their energy and the status of their wallet without long delays. The document also makes use of specific features of Hedera Hashgraph, like the topic, to notify peers of a new event. This is particularly useful because there is no need to have auxiliary blockchain tools.

After analyzing the results, it is evident that, for achieving real-time execution in the context of a REC system, smart contracts can be costly. When comparing Direct and Freeze transactions, both have similar currency and network traffic costs, with the primary distinction being the security of the process.

Given that the devices involved are IoT devices, which may have isolated network communication modules with associated costs, despite the slight increase in transaction size, the added security against tampering makes Freeze transactions a preferable choice.

In terms of how the API should be notified, two options were considered: a normal POST request and the usage of a topic. Despite network statistics showing that the increased bandwidth of using the Topic is negligible since it provides the benefit of an immutable temporal message registry, which is valuable for auditing and comparing with the API databases. On the other hand, sending a POST request only ensures storage on the API. Therefore, for object-based transactions, Freeze Transactions with the use of the topic functionality from Hedera Hashgraph are the preferred approach.

Smart contracts were considered due to the need for refunding and authenticating emitted payments between devices and the API. The Peers-Ruled smart contract proved to be the most convenient option, offering decentralized validation and auditing without the direct involvement of the REC manager. This is in contrast to the REC-Ruled contract, which imposes additional costs on the manager without directly generating profits to minimize those costs. As shown in the discussion part, table 5.3 shows a loss of almost 76% of the seller's profit, but still managing to obtain 15.67 Hbar, while on the REC-Ruled contract, the manager, that is responsible for the payment validation, loses almost 50 Hbar, as shown on table 5.6.

However, when comparing the Peers-Ruled contract with Freeze Transactions using the Topic, the expenses are higher for sellers with the contract. Freeze Transactions excel in real-time execution, particularly for executing money exchanges. The contract may be more appealing if interactions are less frequent and occur over longer periods, thereby reducing operational costs.

In summary, for the purpose of creating a functional REC system using a DLT as the foundation for energy trading and currency exchanges in real-time, Freeze Transactions with the use of the Topic functionality are the most suitable alternative for this use case.

6.1 Limitations and Future work

Throughout this document, multiple bugs were also discovered in the tools that were used. A community contribution was made on several related topics as documented on Hedera Github [63]–[65].

Being auditable and GDPR compliant is not something that is easy, especially when using something like a blockchain where all the information, if not ciphered, is accessible to everyone. In the developing process of the community, the goal was to make something that was secure and auditable, neglecting a bit of privacy. Of course, this is something that must be revised and a mechanism like a TOTP could be used when deploying a topic message, as an example. Privacy is immediately acquired since only the buyer and the manager should possess the same secret.

Following the limitation of not having easy refunds and management via the methodology chosen in the discussion part, it is important to revisit smart contracts and the strategy used to process the community. In this case, the project opted for a real-time community, where everything that happens is what can be seen immediately. However, taking into account the effectiveness of the smart contracts in the payment validation process, it is worth considering a daily or weekly community processing of the community state.

APPENDIX A

Protocol Buffer for Meters Microservice

```
service MetersService{
    rpc AddMeasurement(MeterEntry) returns (MeterResponse);
    rpc RetrieveMeasurement(QueryMeters) returns (QueryResponse);

}

message MeterEntry{
    string deviceId = 1;
    int32 activeImport = 2;
    int32 activeExport = 3;
    int32 reactiveInductive = 4;
    int32 reactiveCapacitive = 5;
    string timestamp = 6;
}

message MeterResponse{
    int32 status = 1;
    optional PaymentsResponse payments = 2;
    optional string message = 3;

}

message QueryMeters{
    optional string startInterval = 1;
    optional string deviceId = 2;
    optional int32 skip = 3;
    optional int32 limit = 4;
}
```

```
message QueryResponse{
    repeated MeasurementResponse entries = 1;
    optional int32 status = 2;
    optional string error = 3;
}

message MeasurementResponse{
    string DeviceId = 1;
    string Field = 2;
    string Value = 3;
    string Date = 4;
}
```

Listing A.1: Protocol Buffer for Meters Microservice

APPENDIX B

Protocol Buffer for Authenticated Market Microservice

```
service AuthenticatedMarketService{
    rpc RetrieveMatches(MatchesFilter) returns (ListMatchResponse);
    rpc UpdateMatch(ListUpdateMatch) returns (Empty);
}

enum State{
    Created = 0;
    Sent = 1;
    Paid = 2;
    NotPaid = 3;
    Error = 4;
}

message Empty{}

message MatchesFilter{
    optional string buyerID = 1;
    optional string sellerID = 2;
    repeated State state = 3;
    optional string startTimestamp = 4;
    optional string endTimestamp = 5;
    optional string matchID = 6;
    optional int32 skip = 7;
    optional int32 limit = 8;
}

message MatchResponse{
    string timestamp = 1;
    string buyerID = 2;
    string sellerID = 3;
    float energy = 4;
    float price = 5;
    string id = 6;
    string createdAt = 7;
    optional string transactionID = 8;
    optional State transactionState = 9;
    optional string updatedAt = 10;
    optional string message = 11;
}

message ListMatchResponse{
    repeated MatchResponse matches = 1;
}

message UpdateMatch{
    string matchID = 1;
    State state = 3;
    optional string transactionID = 2;
    optional string message = 4;
}

message ListUpdateMatch{
    repeated UpdateMatch matches = 1;
}
```

Listing B.1: Protocol Buffer for Authenticated Market Service

APPENDIX C

Protocol Buffer for Transactions Microservice

```
import public "market.proto";

service TransactionsService{
    rpc AddTxReceipt(ListReceipts) returns (Empty);
    rpc GetPayments(IssuerId) returns (PaymentsResponse);
    rpc AddAccount(DeviceInfo) returns (Empty);

}

message DeviceInfo{
    string pubkey = 1 ;
    string deviceId = 2;
}

message PaymentsResponse{
    int32 approach = 1;
    repeated MatchResponse json = 2;
    repeated TransactionResponse transactions = 3;
}

message JsonReceipt{
    string paymentID = 1;
    string txID = 2;
}

message ListReceipts{
    repeated JsonReceipt receipts = 1;
}
```

```
message IssuerId{
    string id = 1;
}

message TransactionResponse{
    bytes transaction = 1;

}
```

Listing C.1: Protocol Buffer for Transactions Microservice

References

[1] M. Rauchs, A. Glidden, B. Gordon, *et al.*, "Distributed Ledger Technology systems: A conceptual framework," *Available at SSRN 3230013*, 2018. [Online]. Available: `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3230013`.

[2] R. Marsh, *Energy experts sound alarm about US electric grid: 'not designed to withstand the impacts of climate change'*, Jun. 2022. [Online]. Available: `https://edition.cnn.com/2022/05/31/us/power-outages-electric-grid-climate-change/index.html` (visited on 09/12/2023).

[3] R. Jiménez, T. Serebrisky, and J. Mercado, "Sizing electricity losses in transmission and distribution systems in Latin America and the Caribbean," *Published by Inter-American Development Bank*, 2014. [Online]. Available: `https://www.researchgate.net/publication/267750864_POWER_LOST_Sizing_Electricity_Losses_in_Transmission_and_Distribution_Systems_in_Latin_America_and_the_Caribbean`.

[4] J. Lesser, *America's Electricity Grid: Outdated Or Underrated?* Heritage Foundation, 2014. [Online]. Available: `https://www.heritage.org/environment/report/americas-electricity-grid-outdated-or-underrated`.

[5] Consensys, *Blockchain in the energy sector: Real World blockchain use cases*. [Online]. Available: `https://consensys.net/blockchain-use-cases/energy-and-sustainability/` (visited on 09/24/2023).

[6] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," in *Concurrency: The Works of Leslie Lamport*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 203–226, ISBN: 9781450372701. [Online]. Available: `https://doi.org/10.1145/3335772.3335936`.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21 260, 2008. [Online]. Available: `https://assets.pubpub.org/d8wct41f/31611263538139.pdf`.

[8] S. Haber and W. S. Stornetta, "How to Time-Stamp a Digital Document," in *Advances in Cryptology-CRYPTO' 90*, A. J. Menezes and S. A. Vanstone, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 437–455, ISBN: 978-3-540-38424-3. [Online]. Available: `https://link.springer.com/chapter/10.1007/3-540-38424-3_32`.

[9] J. Li, M. N. Krohn, D. Mazieres, and D. E. Shasha, "Secure Untrusted Data Repository (SUNDR).," in *Osdi*, vol. 4, 2004, pp. 9–9. [Online]. Available: `https://www.usenix.org/legacy/event/osdi04/tech/full_papers/li_j/li_j.pdf`.

[10] M. Andoni, V. Robu, D. Flynn, *et al.*, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renewable and Sustainable Energy Reviews*, vol. 100, pp. 143–174, 2019, ISSN: 1364-0321. DOI: `https://doi.org/10.1016/j.rser.2018.10.014`.

[11] Digiconomist, *Bitcoin energy consumption worldwide 2017-2022*, 2022. [Online]. Available: `https://www.statista.com/statistics/881472/worldwide-bitcoin-energy-consumption/` (visited on 09/12/2023).

[12] ——, *Ethereum Energy Consumption Worldwide 2017-2022*, 2022. [Online]. Available: `https://www.statista.com/statistics/1265897/worldwide-ethereum-energy-consumption/` (visited on 09/12/2023).

[13] P. DGEG/MAAC, *Consumo de Energia Eléctrica: Total E por tipo de consumo*. [Online]. Available: `https://www.pordata.pt/Portugal/Consumo+de+energia+electrica+total+e+por+tipo+de+consumo-1124` (visited on 09/12/2023).

[14] Ethereum, *Proof-of-stake (POS).* [Online]. Available: `https://ethereum.org/pt/developers/docs/consensus-mechanisms/pos/`.

[15] J. Siim, "Proof-of-stake," in *Research seminar in cryptography*, 2017.

[16] K. Karantias, A. Kiayias, and D. Zindros, "Proof-of-burn," in *International conference on financial cryptography and data security*, Springer, 2020, pp. 523–540. [Online]. Available: `https://link.springer.com/chapter/10.1007/978-3-030-51280-4_28`.

[17] A. Andrey and C. Petr, "Review of Existing Consensus Algorithms Blockchain," in *2019 International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT&QM&IS)*, 2019, pp. 124–127. DOI: `10.1109/ITQMIS.2019.8928323`.

[18] M. A. Manolache, S. Manolache, and N. Tapus, "Decision Making using the Blockchain Proof of Authority Consensus," *Procedia Computer Science*, vol. 199, pp. 580–588, 2022. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1877050922000710`.

[19] Y. Merrad, M. H. Habaebi, E. A. Elsheikh, *et al.*, "Blockchain: Consensus Algorithm Key Performance Indicators, Trade-Offs, Current Trends, Common Drawbacks, and Novel Solution Proposals," *Mathematics*, vol. 10, no. 15, p. 2754, 2022. [Online]. Available: `https://www.mdpi.com/2227-7390/10/15/2754`.

[20] B. S. Reddy and G. Sharma, "Optimal transaction throughput in proof-of-work based blockchain networks," *Multidisciplinary Digital Publishing Institute Proceedings*, vol. 28, no. 1, p. 6, 2019. [Online]. Available: `https://www.mdpi.com/2504-3900/28/1/6`.

[21] A. Corso, "Performance analysis of proof-of-elapsed-time (poet) consensus in the sawtooth blockchain framework," Ph.D. dissertation, University of Oregon, 2019. [Online]. Available: `https://www.cs.uoregon.edu/Reports/MS-201906-Corso.pdf`.

[22] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities," *IEEE Access*, vol. 7, pp. 85 727–85 745, 2019. DOI: `10.1109/ACCESS.2019.2925010`.

[23] M. Platt, J. Sedlmeir, D. Platt, *et al.*, "The Energy Footprint of Blockchain Consensus Mechanisms Beyond Proof-of-Work," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2021, pp. 1135–1144. DOI: `10.1109/QRS-C55045.2021.00168`.

[24] R. C. Merkle, "Protocols for Public Key Cryptosystems," in *1980 IEEE Symposium on Security and Privacy*, 1980, pp. 122–122. DOI: `10.1109/SP.1980.10006`.

[25] V. Buterin, "Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform," 2014. [Online]. Available: `https://github.com/ethereum/wiki/wiki/White-Paper` (visited on 09/15/2023).

[26] ethereum.org, *The Beacon Chain.* [Online]. Available: `https://ethereum.org/pt/upgrades/beacon-chain/` (visited on 09/17/2023).

[27] ethereum, *Sharding.* [Online]. Available: `https://ethereum.org/pt/upgrades/sharding/` (visited on 09/17/2023).

[28] ——, *The Merge.* [Online]. Available: `https://ethereum.org/pt/upgrades/merge/` (visited on 09/17/2023).

[29] Bnb-Chain, *Whitepaper/WHITEPAPER.md at master · BNB-Chain/Whitepaper.* [Online]. Available: `https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md` (visited on 09/17/2023).

[30] S. M. S. Saad and R. Z. R. M. Radzi, "Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos)," *International Journal of Innovative Computing*, vol. 10, no. 2, 2020. [Online]. Available: `https://www.researchgate.net/publication/347308252_Comparative_Review_of_the_Blockchain_Consensus_Algorithm_Between_Proof_of_Stake_POS_and_Delegated_Proof_of_Stake_DPOS`.

[31] N. Živić, E. Kadušić, and K. Kadušić, "Directed Acyclic Graph as Hashgraph: an Alternative DLT to Blockchains and Tangles," in *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2020, pp. 1–4. DOI: `10.1109/INFOTEH48170.2020.9066312`.

[32] I. Kotilevets, I. Ivanova, I. Romanov, S. Magomedov, V. Nikonov, and S. Pavelev, “Implementation of Directed Acyclic Graph in Blockchain Network to Improve Security and Speed of Transactions,” *IFAC-PapersOnLine*, vol. 51, no. 30, pp. 693–696, 2018, 18th IFAC Conference on Technology, Culture and International Stability TECIS 2018, ISSN: 2405-8963. DOI: `https://doi.org/10.1016/j.ifacol.2018.11.213`.

[33] W. F. Silvano and R. Marcelino, “Iota Tangle: A cryptocurrency to communicate Internet-of-Things data,” *Future Generation Computer Systems*, vol. 112, pp. 307–319, 2020. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167739X19329048`.

[34] L. Baird, M. Harmon, and P. Madsen, “Hedera: A public hashgraph network & governing council,” *White Paper*, vol. 1, 2019. [Online]. Available: `https://hedera.com/hh_whitepaper_v2.1-20200815.pdf`.

[35] S. Popov, “The Tangle,” *White paper*, vol. 1, no. 3, p. 30, 2018. [Online]. Available: `http://cryptoverze.s3.us-east-2.amazonaws.com/wp-content/uploads/2018/11/10012054/IOTA-MIOTA-Whitepaper.pdf`.

[36] H. Hashgraph, *Gossip about gossip protocol.* [Online]. Available: `https://hedera.com/learning/hedera-hashgraph/what-is-gossip-about-gossip` (visited on 10/02/2023).

[37] Hedera, *Hedera Hashgraph.* [Online]. Available: `https://hedera.com/` (visited on 09/26/2023).

[38] F. M. Benčić and I. P. Žarko, “Distributed ledger technology: Blockchain compared to directed acyclic graph,” in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2018, pp. 1569–1570.

[39] Nano, *Living Whitepaper - Nano Documentation.* [Online]. Available: `https://docs.nano.org/living-whitepaper/` (visited on 08/10/2023).

[40] U. W. Chohan, “Are Stable Coins Stable?” *SSRN Electronic Journal*, 2019. DOI: `10.2139/ssrn.3326823`.

[41] CoinMarketCap. “CoinMarketCap Tether Gold Price.” (2022), [Online]. Available: `https://coinmarketcap.com/currencies/tether-gold` (visited on 11/15/2022).

[42] M. Insider. “Insider Gold Price.” (2022), [Online]. Available: `https://markets.businessinsider.com/commodities/gold-price` (visited on 11/15/2022).

[43] C. Pop, T. Cioara, M. Antal, I. Anghel, I. Salomie, and M. Bertoncini, “Blockchain Based Decentralized Management of Demand Response Programs in Smart Energy Grids,” *Sensors*, vol. 18, no. 1, 2018, ISSN: 1424-8220. DOI: `10.3390/s18010162`. [Online]. Available: `https://www.mdpi.com/1424-8220/18/1/162`.

[44] C. Natoli, J. Yu, V. Gramoli, and P. Esteves-Verissimo, *Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure*, 2019. DOI: `10.48550/ARXIV.1908.08316`.

[45] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, “Performance analysis of hyperledger fabric platforms,” *Security and Communication Networks*, vol. 2018, 2018. [Online]. Available: `https://www.hindawi.com/journals/scn/2018/3976093/`.

[46] Hyperledger, *CHAINCODE tutorials.* [Online]. Available: `https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html` (visited on 08/21/2023).

[47] ——, *About Hyperledger.* [Online]. Available: `https://www.hyperledger.org/about` (visited on 09/14/2023).

[48] C. Cachin *et al.*, “Architecture of the hyperledger blockchain fabric,” in *Workshop on distributed cryptocurrencies and consensus ledgers*, Chicago, IL, vol. 310, 2016, pp. 1–4. [Online]. Available: `https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf`.

[49] D. A. C. Team, *Canton: A DAML based Ledger Interoperability Protocol*, Feb. 2020. [Online]. Available: `https://www.canton.io/publications/canton-whitepaper.pdf` (visited on 09/09/2023).

[50] Tendermint, *Cosmos - Internet of blockchains.* [Online]. Available: `https://v1.cosmos.network/intro` (visited on 09/10/2023).

[51] J. Kwon and E. Buchman, “Cosmos whitepaper,” *A Netw. Distrib. Ledgers*, 2019. [Online]. Available: `https://v1.cosmos.network/resources/whitepaper`.

[52] N. Z. Aitzhan and D. Svetinovic, “Security and Privacy in Decentralized Energy Trading Through Multi-Signatures, Blockchain and Anonymous Messaging Streams,” *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 840–852, 2018. DOI: 10.1109/TDSC.2016.2616861.

[53] T. Alladi, V. Chamola, J. J. Rodrigues, and S. A. Kozlov, “Blockchain in smart grids: A review on different use cases,” *Sensors*, vol. 19, no. 22, p. 4862, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/22/4862.

[54] M. F. Zia, M. Benbouzid, E. Elbouchikhi, S. M. Muyeen, K. Techato, and J. M. Guerrero, “Microgrid Transactive Energy: Review, Architectures, Distributed Ledger Technologies, and Market Analysis,” *IEEE Access*, vol. 8, pp. 19 410–19 432, 2020. DOI: 10.1109/ACCESS.2020.2968402.

[55] Spectral, *About Jouliette*. [Online]. Available: https://www.jouliette.net/about.html (visited on 09/18/2023).

[56] P. Network, *Pylon network. the energy blockchain platform.* [Online]. Available: https://pylon-network.org/wp-content/uploads/2019/02/WhitePaper_PYLON_v2_ENGLISH-1.pdf (visited on 04/15/2023).

[57] *Pylon network FAIRCOOP: Building a scalable blockchain consensus for the energy system*, Dec. 2017. [Online]. Available: https://pylon-network.org/pylon-network-faircoop-building-scalable-blockchain-consensus-energy-system.html (visited on 04/16/2023).

[58] T. König, E. Duran, N. Fessler, and R. Alton, “The Proof-of-Cooperation Blockchain FairCoin,” *White paper version*, vol. 1, 2018. [Online]. Available: https://fair-coin.org/sites/default/files/FairCoin2_whitepaper_V1.2.pdf.

[59] P. Picazo-Sanchez and M. Almgren, “Gridchain: an investigation of privacy for the future local distribution grid,” *International Journal of Information Security*, vol. 22, no. 1, pp. 29–46, Feb. 2023, ISSN: 1615-5270. DOI: 10.1007/s10207-022-00622-6.

[60] E. Union, *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016*, Apr. 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj (visited on 08/23/2023).

[61] K. Mysliwiec, *A progressive node.js framework*, 2017. [Online]. Available: https://nestjs.com/ (visited on 06/07/2023).

[62] POSTGRES, 1996. [Online]. Available: https://www.postgresql.org/about/ (visited on 08/30/2023).

[63] D. Andrade, *GRPC failure when Deploying Smart Contract and other misc bugs*, Jul. 2023. [Online]. Available: https://github.com/hashgraph/hedera-local-node/issues/358 (visited on 07/10/2023).

[64] ——, *Reducing empty arrays on contractfunctionparameters() causes error*, Oct. 2023. [Online]. Available: https://github.com/hashgraph/hedera-sdk-js/issues/1990 (visited on 10/19/2023).

[65] ——, *Uint256Array on contract function arguments expecting number[] instead of BigNumber[]*, Oct. 2023. [Online]. Available: https://github.com/hashgraph/hedera-sdk-js/issues/1976 (visited on 10/16/2023).