Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR'S DEGREE THESIS

THESIS TITLE: Blockchain based application for circular economy

DEGREE: Bachelor's Degree in Telecommunication's Systems Engineering

AUTHOR: Gabriel González Campos

ADVISORS: Toni Oller Arcas

DATE: 8th February 2022

Título: Aplicación basada en blockchain para economía circular

Autor: Gabriel González Campos

Director: Toni Oller Arcas

Data: 8 de febrero de 2022

Resumen

Durante los últimos años los servicios sobre internet han evolucionado enormemente, desde arcaicas webs con solo texto dónde solo el web máster puede subir y actualizar contenidos, hasta las más modernas aplicaciones como redes sociales con videollamadas y transmisiones de video en tiempo real donde todos los usuarios interactúan y crean contenido nuevo.

Todos estos avances en la manera que se construyen los servicios y como interactuamos con ellos han sido posibles gracias a grandes mejoras y aportaciones tecnológicas, como la cada vez mayor velocidad de las redes de acceso, la evolución de los ordenadores de sobremesa a ordenadores portátiles, la aparición de dispositivos de bolsillo como los Smartphones y el desarrollo de cada vez más potentes redes de acceso inalámbricas, como la evolución de los estándares de telefonía móvil.

Tras toda esta evolución la tecnología no se ha estancado y sigue evolucionando para proveernos de nuevas maneras de construir servicios e interactuar con mundos virtuales.

En los últimos años, han aparecido nuevas tecnologías que han producido todo un cambio de paradigma en la relación que tendremos con los servicios basados en internet, incluso hay compañías que hablan del inicio de un metaverso completamente inmersivo.

En esta tesis analizaremos algunas de estas tecnologías, como son las redes blockchain, su posible implicación social y desarrollaremos una aplicación basada en esta tecnología siguiendo un marco de trabajo de economía circular.

Títol: Aplicació basada en blockchain per economia circular

Autor: Gabriel González Campos

Director: Toni Oller Arcas

Data: 8 de Febrer de 2022

Resum

Durant els últims anys, els serveis sobre internet han patit una forta evolució, des de pàgines web on només es podia consumir contingut en format de text i on només el web màster podia penjar i actualitzar continguts, fins a les últimes aplicacions com les xarxes socials amb videotrucades i emissions en temps real en les quals tots els usuaris interactuen i creen nous continguts.

Tots aquests avenços en la manera de construir serveis i com interactuem amb ells han estat possibles gràcies a importants millores i aportacions tecnològiques, com la creixent velocitat de les xarxes d'accés, l'evolució dels ordinadors d'escriptori a ordinadors portàtils, l'aparició de dispositius de butxaca com els Smartphones i el desenvolupament de cada cop més potents xarxes d'accés sense fils, com l'evolució dels estàndards de telèfon mòbil.

Després de tota aquesta evolució, la tecnologia no s'ha estancat i continua millorant per tal de proporcionar-nos noves formes de construir serveis i de relacionar-nos amb mons virtuals.

En els darrers anys han sorgit noves tecnologies que han produït tot un canvi de paradigma en la relació que tindrem amb els serveis basats en internet, fins i tot algunes empreses estan parlant de l'inici d'una metavers totalment immersiu.

En aquesta tesi analitzarem algunes d'aquestes tecnologies, com les xarxes blockchain, la seva possible implicació social i desenvoluparem una aplicació basada en aquesta tecnologia en un marc d'economia circular.

Title: Blockchain based application for circular economy

Author: Gabriel González Campos

Director: Toni Oller Arcas

Date: 8th February 2022

Overview

Over the last few years, web services have evolved enormously, from archaic text-only websites where only the web master can upload and update content, to the most modern applications such as social networks with video calls and real-time video streaming where all users interact and create new content.

All these advances in the way services are built and how we interact with them have been made possible by major technological improvements and contributions, such as the increasing speed of access networks, the evolution from desktop to laptop computers, the emergence of pocket devices such as smartphones and the development of even more powerful wireless access networks, such as the evolution of mobile telephony standards.

Despite all this evolution, technology has not stagnated and continues to evolve to provide us new ways to build services and interact with virtual worlds.

In recent years, new technologies have appeared, and these technologies have produced a paradigm shift in the relationship we will have with web-based services, and some companies are even talking about the beginning of a fully immersive metaverse.

In this thesis we will analyze some of these technologies, such as blockchain networks, their possible social implications and we will develop an application based on this technology following a circular economy framework.

Table of contents

LIST OF FIGURES

INTRODUCTION

Last decades there has been a lot of growing interest in applications and web services. Every web-based solution generation improves several limitations of previous generations.

Starting with initial web projects, these projects were read-only sites where only the web master could upload new content to the site. For this, this initial scenario of web projects is known as read-only web. In this first web generation, the web was used by general users to consume almost-static content. In this initial web navigators, as Netscape, and initial e-commerce sites, as Amazon, were created.

Some years later, during the second half of 2000s decade, and profiting the benefits of faster residential networks, there appears some revolutionary services where users are not only content consumers, but users also are content producers, then the web social era started, with projects as Facebook, MySpace or Wikipedia users were able to create its own content and share it with other users, creating a social network on top of these projects.

After the first web revolution that leads from web1, the read-only web, to web2, the social web, there came a second web revolution, but this time comes lead by a hardware evolution, smartphones, as iPhone, and the upswing of wireless access networks as UMTS (Universal Mobile Telecommunication System). During this transition also appeared communication applications such as WhatsApp. At this time users are not only connected when they are using static devices as a computer, but they are permanently connected to different web applications such as social applications, such as Instagram, or content consuming applications, such as Kindle or Netflix.

Then, since last years, around 2018, a lot of new technologies have appeared that aim to change how web solutions are built and consumed. These technologies are blockchain technologies, augmented reality, virtual reality and artificial intelligence algorithms. These technologies aim to build a more fair, transparent and democratic services, where the user owns its data, with an enhanced and more natural interaction between the user and the service and removing human interaction for tedious tasks.

In this project, we will analyze current blockchain technologies and multi-platform user interfaces alternatives. Then we will develop an application based on these technologies and designed considering a circular economy approach.

To do so, we will structure this thesis in 4 main blocks:

1. Circular economy block: We will expose the circular economy concepts, how these new technologies fit under SDG umbrella and an initial overview to develop our solution following these guidelines (Chapter 1).

2. Analysis block: We will analyze the technologies that will be used to develop our application. Firstly, we will analyze the most popular

blockchain technologies to choose the one that better fits with SDG guidelines to develop our project (Chapter 2). Then, we will analyze the main multi-platform development alternatives and we will choose the one that could provide our solution from a better user experience (Chapter 3).

3. Development block: We will describe how the development process we had to perform to develop our solution was, focusing on the architectural proposal and the design of the final user interface (Chapter 4).

4. Conclusion block: We will retrospectively review how the process of building a blockchain-based application following a circular economy approach was, and we will propose future steps for upgrading our solution as a continuous improvement process will do in a product life cycle (Chapter 5).

CHAPTER 1. PROJECT PROPOSAL

In this chapter we will start introducing the main concepts of circular economy and its implications to future developments, then we will review the technology evolution of the last decades and finally, once the main implications of these two powerful concepts are explained, we will define the main objectives of this thesis.

1.1. Basics of circular economy

In the last decade there have been some growing paradigms, some of them in a technical environment, but others growing proposals from a sociological point of view.

From the first ones, the technical ones, there are some technologies that became feasible in the last decade and promise to solve several issues of the current society, an example of these technologies are blockchain technologies, that allow to have more democratic processing and storage of data, and new techniques of artificial intelligence ad machine learning and deep learning.

The technical ones can’t solve any society issue without robust, democratic and fair guidelines to build technologic solutions on top of, but in 2015, the United Nations General Assembly (UN-GA) developed a framework to guide the development of new millennium projects.

This new framework is the Sustainable Development Goals [27, 28] (SDGs), a set of 17 goals to create a road for future developments to build a more fair, democratic and sustainable world. These goals are no poverty; zero hunger; good health and well-being; quality education; gender equality; clean water and sanitation; affordable and clean energy; decent work and economic growth; industry, innovation and infrastructure; reduced inequalities; sustainable cities and communities; responsible consumption and production; climate action; life below water; life on land; peace, justice and strong institutions and partnerships for the goals, summarized at Fig.11.

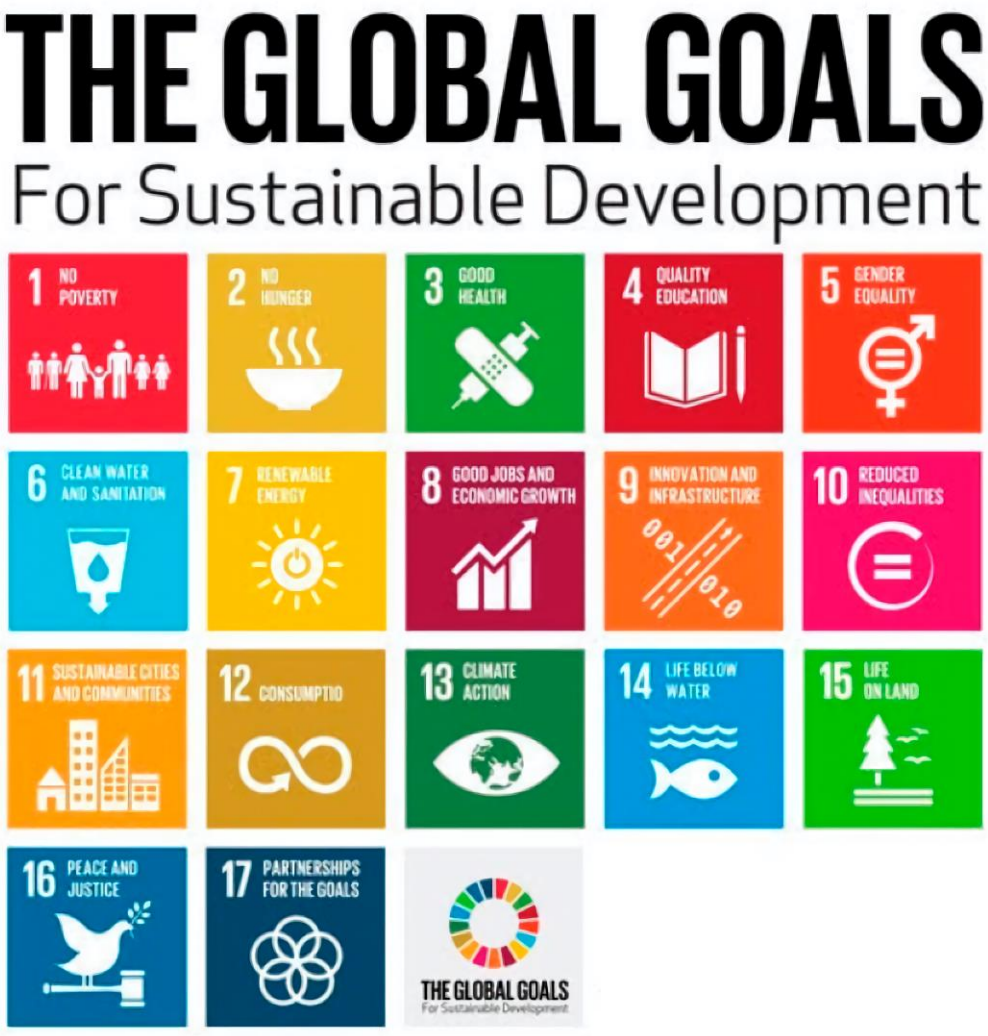

Fig. 1.1 The 17 SDGs, from [28]

In this project we will follow a as much as possible this SDGs framework, but we will focus on the following goals:

- Industry, innovation and infrastructure: We will propose a solution based 0n new technologies such as blockchain that allows us to build more resilient and democratic infrastructure. In this kind of technology everybody has a place as people can use it or power it (discussed in chapter 2).

- Reduced inequalities: These new technologies as the one mentioned previously, the blockchain technology, allows us to build more transparent and fair services and to revoke non-honest behaviors using the power of democracy. Based on that, we will bring these technologies closer to non-technical people.

- Responsible consumption and production: In this project we will strongly consider the resources consumption of each possible solution, as the hardware required or the power consumption.

1.2. Evolution to disruptive technologies

In the initial steps of the web there were small sites whose content was text-based content and where only the web master could upload new content, the web 1.0. In this period, users' relation with web content was in a unidirectional way, from the web master to users passing through its website.

Some years later, and with some technical improvements such as faster residential access networks, an improved kind of web services appeared, the web 2.0. These new web services provide users the ability of uploading its own content and building social networks on top of these services, the social networks' era has begun. This new era allowed users to relate with web content in a bidirectional way.

After some years where a lot of web 2.0 projects were founded and the power of big-tech companies increased, there started to appear a new web paradigm, the web 3.0. This new web paradigm aims to transfer the focus from enterprises to users, allowing users to be the owners of their data and use intelligent agents to make the web a more democratic place. Web 3.0 is supported in some revolutionary technologies such as blockchain, artificial intelligence, virtual reality and internet of things. A large developer's community aims to use these technologies to build the dreamed metaverse.

1.3. Project Goals

Nowadays, all social media is talking about technical concepts as cryptocurrencies, blockchain, web3 and NFT, but usually these concepts are not related with an average citizen's everyday life.

This project came with different main goals, each one designed to impact people's life in various ways.

The main goals of this project are:

- Analyze different blockchains in order to build a map of the current blockchain alternatives. Then, find a good blockchain, from a technical and sustainability point of view, to develop a decentralized APP (DApp) on top of that blockchain.

- Develop a friendly DApp that could be used by the average population. This DApp will be focused on rewarding the good practices of the citizens, giving them some tokens when a good action is performed.

- In order to get a friendly DApp, we will develop a multi-platform application with a modern user interface (UIs) and an improved user experience (UX).

- Provide teaching documentation and examples to develop solutions over blockchains.

CHAPTER 2. BLOCKCHAIN TECHNOLOGIES

Blockchain is a technology that has been growing in popularity in the last decade, but the idea behind this technology is simple and elegant, it consists in a data structure formed by a set of blocks, these blocks contain the useful data and some metadata used to refer unequivocally to another block in that structure, as can be seen in the Fig. 2.1., here's where the name of this technology comes from.

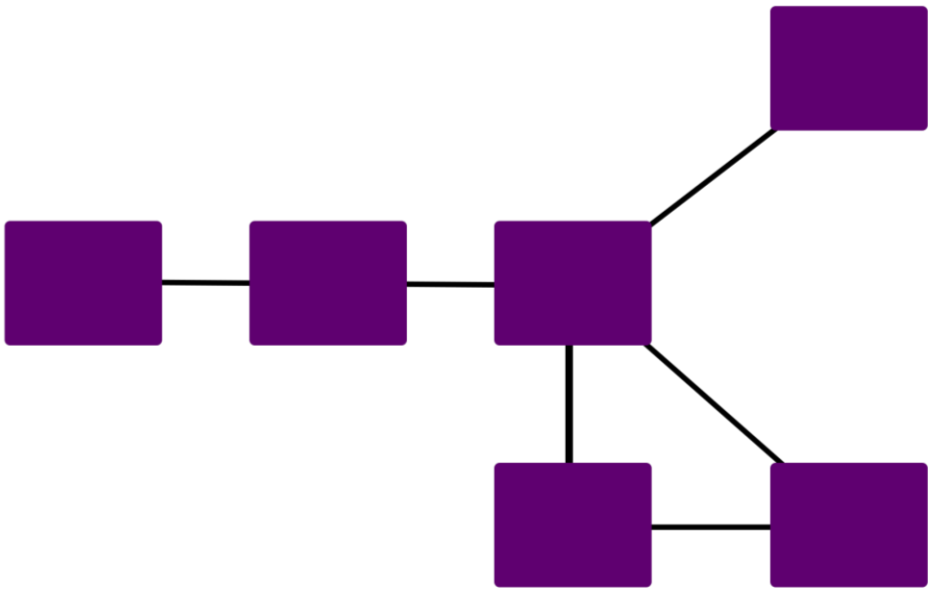

Fig. 2.1 Representation of Blockchain data structure

This idea of a set of blocks related from one to another makes this data structure robust against modifications, as if an attacker wants to modify a block he must modify the related block, and to modify the related block he must modify its related block and so on, then, to modify a specific block the attacker must modify all the related blocks chain (typically all the previous or following blocks).

2.1 Consensus mechanisms

Another key point in blockchain technology is the consensus mechanism. This mechanism is a set of rules that a new block must pass in order to be included in that data structure. The consensus mechanism is a very important point in

terms of decentralizing this data structure, as it defines a specific way to validate the new blocks and agree all nodes in a network.

There are different consensus mechanisms, each one being a trade-off between decentralization, security and scalability. The most popular consensus mechanisms are proof of work, proof of stake and in latest blockchain solutions as Solana, proof of history and proof of space.

2.1.1 Proof of Work (PoW)

Proof of Work was the first consensus mechanism widely used in blockchain networks. It was previously used to mitigate denial of service (DoS) attacks in online resources as the incoming spam into a mail server, as the proposal of Hashcash [2, 3]. Later, a similar approach was proposed by Satoshi Nakamoto as the consensus mechanism used in Bitcoin [4].

This mechanism is based in a computational effort and the most representative PoW mechanisms in blockchain ecosystem are the following:

- Reusable Proof of Work (RPoW): This mechanism is the one derived from Hashcash and proposed as the consensus mechanism of Bitcoin [5]. It consists of looking for a number when hashed, with a specific hash algorithm, results in a binary number starting with a specific number of zeros. In timestamped PoW based blockchains, as Bitcoin, there a nonce is used to provide this timestamp without using a centralized timestamp server, see Fig. 2.2. When a node computes a hash, the nonce value will be incremented, and this action will be repeated until the correct hash is found. This both features combined provides the PoW algorithm of Bitcoin an exponentially increasing needed effort over the time. Alternatively to the hash algorithm used by Bitcoin, SHA-256d, different blockchain can use different algorithms such as the Scrypt used at Litecoin [5, 6] or Ethash used at Ethereum [7]. Each algorithm has its own features as ASIC resistance, power consumption and memory or CPU requirements [8].

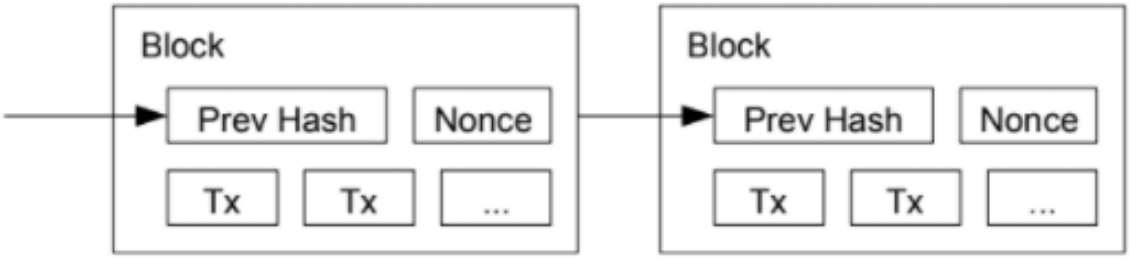

Fig. 2.2 Chaining flow of Bitcoin PoW, from [4]

- Proof of Useful Work: During the last year the power consumption of PoW blockchains has been a matter of discussion and a big hindrance for

blockchain usage expansion. The unused compute power added to this power consumption made researchers propose several solutions. An interesting solution is the use of Proof of Useful Work, this means to use at least a part of the PoW algorithm required power in order to compute results usable in other areas, such as training AI systems [9, 10] in small steps or performing scientific and medical computes. These proposals typically solve the problem of wasting compute power to only compute a specific hash operation.

A common consensus rule in Proof of Work blockchains, and later applied to different consensus mechanism blockchains, is the longest chain rule, represented at Fig 2.3. This rule means that in a certain moment the largest block chain is the valid one, as is the one with the major effort, this rule makes the blockchain robust against malignant greedy nodes, but these blockchains still vulnerable against 51% attacks, where a malignant group of nodes can accept non-valid transactions, as double spending tokens, as long as this set of nodes controls the 51% of compute power of the network. The non-largest chain blocks are called orphan blocks.

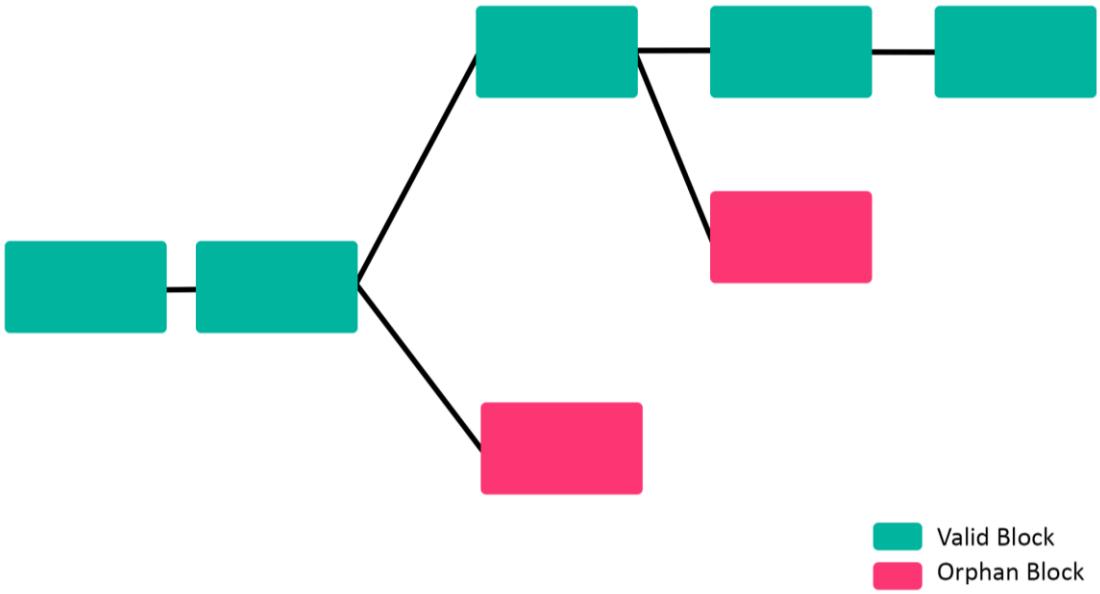

Fig. 2.3 Longest chain rule representation

In general, Proof of Work systems are really good at decentralizing the approval of new blocks without having issues of accepting invalid transactions. This mechanism also promotes the honesty of network nodes as they are rewarded by their effort in adding new blocks to the blockchain.

These systems also provide a high protection against data mutation attacks, but have not a high scalability, having a hard restricted number of transactions per second and exponential power consumption as the network grows.

2.1.2 Proof of Stake (PoS)

Proof of Stake was proposed as an alternative for other consensus mechanisms [11, 12] to avoid their large power consumption (as in PoW) or expensive physical resources (as PoW and proof of space) [13].

Proof of Stake is based on a leader election process that appoints a validator node as the leader of the following block. Once the leader is designed, it will have the authority to validate the following block. The leader election process is usually a random process that considers the stake of each validator, this stake typically refers to the number of tokens staked, the time the validator staked the tokens or a combination of both.

In a Proof of Stake based blockchain the key point is to use a secure and fair leader election process, as the competitors of being the next leader or the current leader mustn't be able to affect in the leader election process. These blockchains usually have punishment mechanisms in order to dissuade validators to perform some attacks possible at Proof of Stake blockchains. These possible attacks are:

- Grinding attack: This attack consists in the current leader node with a tiny stake but with considerable computation power can compute modification in block headers in order to enforce the leader election process to choose him as the next validator. This attack can be achieved by consulting previous blocks assigned to the stake of this validator and using this information to modify the block metadata accordingly. There are several ways to mitigate this attack, such as using secret sharing to generate the next leader election value in a distributed way or using non-modifiable and verifiable data to generate that value.

- Nothing-at-stake attack: This attack consists of that if an attacker will aim to perform a double spend, the attacker can spend the tokens and create a fork from the block before the spend transaction, then the attacker will decide to stake only for that new fork. Then, as a validator takes the same profit staking in a single fork or in both and the attacker will only stake in the recently created fork, in the main chain there will be the rest of the network staking and in the new branch will be the same amount of stake plus the attacker stake, the malicious fork will eventually become the main chain as it will become longer. At this moment, there are several researchers figuring how to prevent this kind of attacks, as the Ethereum team developing Casper, the PoS implementation for the Ethereum blockchain, and Versus working group claimed to be able to solve this problem combining PoW, PoS and Proof of Delayed Work. It can be better understood by looking at Fig 2.4 diagram.

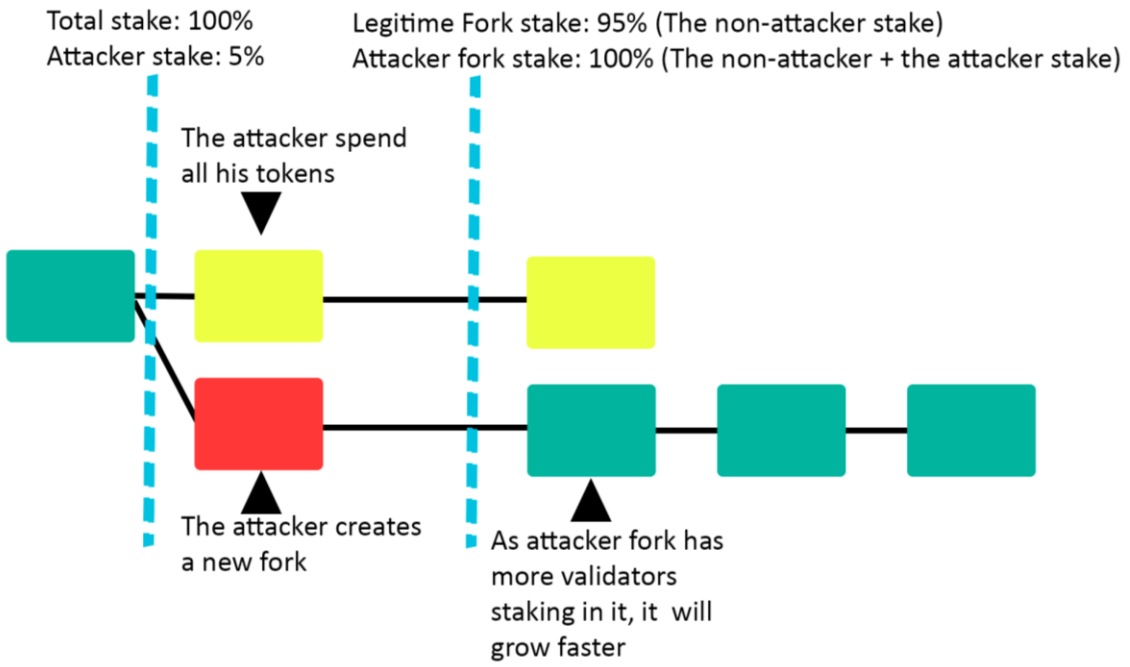

Fig. 2.4 Representation of a Nothing-at-stake attack

- Long Range Attack: This attack consists of an attacker trying to change an already processed and validated block. As PoS does not have any demanding compute operation, then, if the attacker has enough compute power it is possible to modify a past block and then change all the related blocks in the chain creating a fork that differs from the real fork in the modified block. The aim of creating a PoS algorithm able to prevent this attack was one of the goals of creating Ouroboros [13], the PoS algorithm used at Cardano blockchain.

In addition to previous possible attacks, there is one major concern about basic PoS implementations, the Weak Subjectivity issue. This issue consists in the necessity from a new node or a reconnected node of getting the current state of the blockchain. In proof of work there is a simple solution, ask another node for the state and if the new node gets different responses, the largest chain with a higher compute effort will be the active state of the blockchain, in this case, the network is objective, but in PoS blockchains an attacker can take advantage of this situation to spread its malicious fork, at this situation, if the malicious fork has the same length as the real fork, the new node will not be able to distinguish which fork is the correct one. In PoS blockchains, as they are weak subjectivity blockchains, the new node can distinguish the active fork after some blocks were added to the chains. As with other attacks and vulnerabilities of basic PoS implementations, there are working groups, such as Versus, that have proposed algorithms to prevent the network from these issues and make it more objective.

Despite all these issues and vulnerabilities of a basic implementation of PoS consensus mechanism there are a lot of effort to build robust blockchains using PoS as it provides a lot of advantages respectively to PoW:

- Faster transactions: As the consensus mechanism is not dependent on a very complex and time-consuming effort, the time for each block to be added to the chain is lower than in PoW blockchains.

- Power efficient: PoS blockchains consume less electricity than PoW ones. This is also due to the lack of a power-consuming algorithm as a key element of the operation of PoS blockchains.

Proof of Stake has some advantages against Proof of Work, but also has some drawbacks such as the threat of being more centralized Work and not as secure as Proof of Work.

In real blockchains PoS is usually combined with some other methods like PoW in order to make a system more secure and more tolerant to Byzantine faults. These faults lead to having some different consensus in the network, such as consensus failure, block validation or data validation failures and communication failures.

2.1.3 Delegated Proof of Stake (DPoS)

Delegated Proof of Stake was firstly proposed to enhance PoS benefits as the transaction velocity it also was designed to be a consensus mechanism with Byzantine fault tolerance (BFT).

This consensus mechanism is like PoS, but with some differences. In DPoS blockchains validators vote for a delegate validator to add a block by pointing their stakes into the delegate validator’s staking pool, see Fig 2.5, once the voting process is performed, the delegate validators (or witnesses) must agree to add the new block.

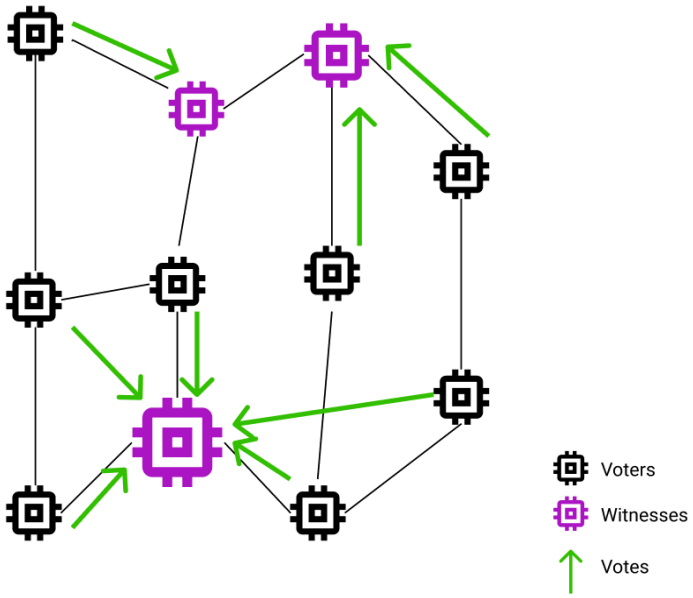

Fig. 2.5 DPoS voting process

This delegated version of PoS aims to be faster and more secure than PoS, but this consensus mechanism can lead to centralized networks as most of the stake can vote to a single validator, this validator can take advantage of this situation to modify the block in his own profit or to not accept the block. To mitigate this situation, DPoS blockchains usually have a concept of stolen block, a block that has been assigned to a delegated validator to be approved, but that validator has skipped it, in this case this block is called a stolen block, the validator will not receive the reward of that block and the stolen block and its reward will be assigned to the next delegated validator in order to not lose transactions.

2.1.4 Proof of Elapsed Time (PoET)

Proof of Elapsed Time is a consensus mechanism proposed by Intel [14] to be used in enterprise environments to automate and validate error prone tasks, an example is the Hyperledger Sawtooth platform.

This consensus mechanism is based on the fair lottery principle and using a trusted function running in a trusted environment, for example Intel SGX. The validators will ask the trusted function for a wait time and each validator will sleep until the wait time assigned to them finishes, the first validator that awakes will be the validator that will process the block.

The key point of this consensus mechanism is that the trusted function must distribute the leader election in a uniform way in order to provide the system the desired fairness.

Another distinctive point is the necessity of having a centralized element, the trusted function. This centralization is not desired in a lot of environments where blockchains are used, but in an enterprise environment PoET network can be

useful as ledger or even as infrastructure in an enterprise resource planning (ERP) system.

2.1.5 Proof of History (PoH)

Proof of History is a consensus mechanism based on timestamps. This consensus mechanism can place all blocks and transactions in the timeline; this is achieved by using Verifiable Delay Functions (VDF), a function that must be run in sequence in a single core and must be finished to get the output. The output of this function will be its input for the following iteration [15]. Thanks to the usage of VDF and the sequence of its outputs we can build a list of timestamps in order, see Fig 2.6.

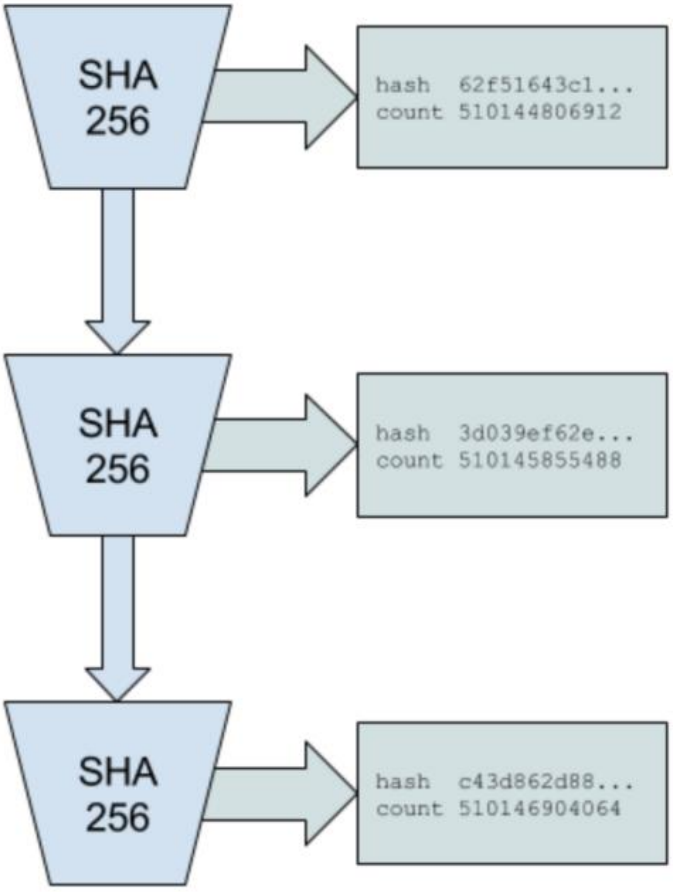

Fig. 2.6 PoH VDF operation, from [15]

Using this method and inserting some data, PoH blockchains can build a sequence of events and it can place in time when each event has performed. To do so, the event data and the previous hash can be combined using a collision resistant function and the result is used as the input of the next VDF iteration, represented at Fig. 2.7. Then, a specific output hash can represent the timestamp of a specific event because that hash could not be computed if the event is not inserted before the new has is computed, and it keeps following the

timestamp order as the previous VDF output is also needed to compute that output hash.

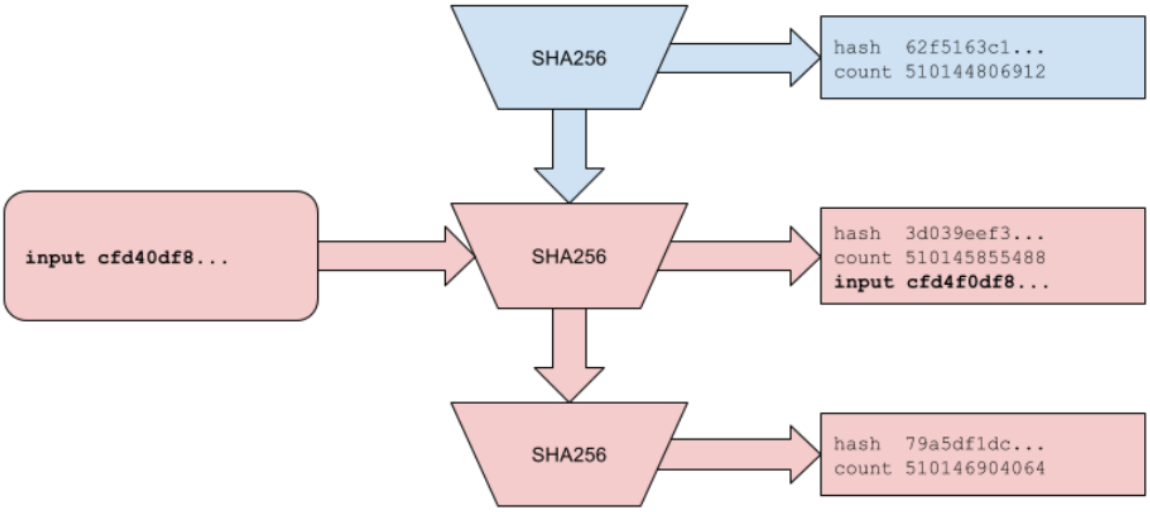

Fig. 2.7 Diagram of how to insert data in a Proof of History system, from [15]

Once this chain is created, the verification can be performed in significantly less time that it took to generate. As all the output hashes have been already computed, the validation can be performed in parallel using multiple compute threads, example of validating PoH ticks in a multi-core processor in Fig. 2.8.

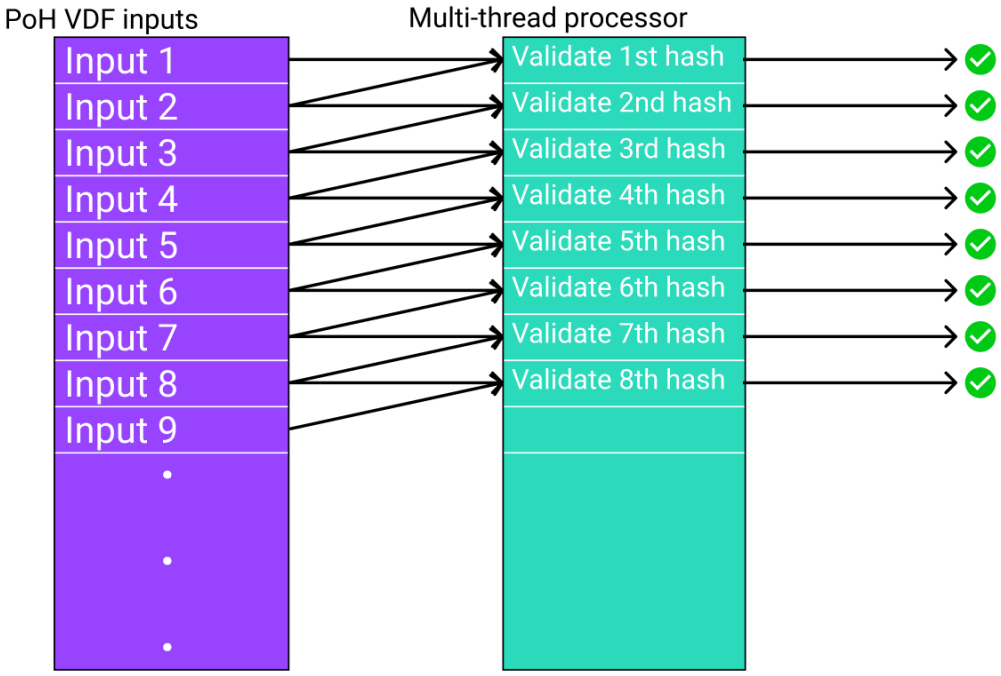

Fig. 2.8 Validation of Proof of History hash sequence

With the previous operation, proof of history is still vulnerable to reordering attacks as the attacker can reorder the events of a sub chain as long as this attacker has access to all the events in the sub chain at once or if the attacker can produce the sub chain faster than the current PoH generator. To prevent that vulnerability, the data of an event combined to compute the next hash must be signed by the current PoH generator.

2.1.6 Proof of Space (PoSpace)

Proof of Space is a consensus mechanism based on an interested node storing some data in order to prove its real interest in participating in a blockchain operation.

This consensus mechanism can operate in three ways:

- Proof of Capacity: It is a consensus mechanism based on compute a PoW function and store it in disk space.

- Proof of Storage: Similar to the previous one but storing useful data instead of a PoW function output, this kind of consensus mechanism is typically used in blockchains dedicated to store files or act like a database, as Arweave and Storej.

- Proof of Space-Time: this version of proof of space considers the amount of data, but also considers the time this node stores the data.

2.1.7 Proof of Replication (PoRep)

Proof of Replication is usually used in proof of space networks to retrieve the data stored by each node [16].

This consensus mechanism works by sending to each node some data to store and asking them to retrieve the data some time later.

This mechanism is usually used in blockchains dedicated to store data in order to provide high availability of the stored resources, as the same data can be assigned to different nodes.

Another advantage of PoRep is that this mechanism provides an easy way to calculate the cost of storing a specific piece of data [15].

Although all these advantages of PoRep mechanisms used in blockchains dedicated to store data there is a main issue when using this mechanism, when a new node is added to the network, this node is assigned to a replication pool, then this node must download all the data from that replication pool, consuming a lot of bandwidth of these nodes.

2.2 Blockchain types

Blockchains have different features depending on the consensus mechanism each one use, but these blockchains can also be categorized depending on the ownership of the nodes forming the blockchain peer to peer network

2.2.1 Public blockchains

Public blockchains are the ones that anyone can become a validator and send transactions if they have an internet connection. These blockchains usually provide an economic reward to encourage new nodes to become a part of this network.

These kinds of network blockchains are usually used as an infrastructure to cryptocurrencies and smart contracts execution and are based on a consensus mechanism that provides trust between all unfamiliar nodes.

The main advantage of these networks is the transparency as the state of the network is shared between all nodes and can be consulted by outsiders by using tools as blocks inspector.

The most important drawback of public blockchains is that the network is as fast and as secure as the consensus mechanism permits. For example, Bitcoin is a public Blockchain able to currently process a block each 10 minutes, having an average block size of 1.18MB [17] and an average transaction size of 454.48 bytes [17] we get an average transaction throughput of 4.33 transactions per second.

$$Transactions\ per\ block = \frac{Block\ Average\ Size\ [B]}{Transaction\ average\ Size\ [B]}$$

$$Transactions\ per\ block = \frac{1.18\ MB}{454.48\ B} = 2596.37\ transactions\ per\ block$$

$$Transactions\ per\ second = \frac{Transactions\ per\ block}{Block\ time}$$

$$Transactions\ per\ second = \frac{2596.37}{10\ \text{min} * 60\ sec/min} = 4.33\ transactions\ per\ second$$

2.2.2 Private blockchains

Private blockchains are blockchain networks that require a granted access to use it or to become a node, private blockchains also are blockchains that operate in a restricted environment, as in an enterprise network.

These blockchains are used by many companies as a ledger, to certificate internal documentation, etc.

As private blockchains typically operate in restricted environments, there is no need to use a heavy consensus mechanism as PoW or PoS, this leads to have lightweight blockchains with some advantages:

- High performance: as the consensus mechanism can converge faster and there are limited nodes in the network, the transaction rate can be higher compared to public blockchains.

- Efficient networks: usually private blockchains are not based in computing demandant efforts as PoW.

- Scalable: as usually consensus mechanisms of private blockchain are not power demanding nor time demanding, the blockchain can grow to provide a high availability network without increasing drastically the CAPEX and OPEX costs.

2.2.3 Hybrid blockchains

Hybrid blockchains are usually blockchain networks built in a controlled environment but able to serve anybody with an internet connection. In this blockchains a user can access and use the blockchain features but is not able to join as a validator.

These blockchains are designed to get the benefits from public and private blockchains as everyone can send transactions to the blockchain. But as the validator nodes need a granted access to the network an attacker can’t access to the network as a validator, preventing several types of attacks such as 51% attack.

Hybrid blockchains also have the benefits of private blockchain networks as having high performance, being power efficient and being scalable.

Although in hybrid blockchains the whole transaction history is not made public, the transactions still are verifiable and the validator nodes can’t modify the transactions, but these nodes can determine what transactions will be public.

A key advantage of hybrid blockchains is that the validation rules can be modified by the organization owning the validator nodes in order to modify the behavior of a service built on top of that blockchain.

2.2.4 Consortium blockchains

Consortium blockchains are similar to hybrid blockchains, but the validator nodes are not owned by a single organization.

These blockchains are formed by validator nodes (in some implementations, only one validator node is present, acting more like an orchestrator), able to create, receive and validate transactions and some member nodes able to create and receive transactions.

2.3 Ethereum

Ethereum is a blockchain created by Vitalik Buterin [19] based on the colored coins protocol [20] and Satoshi Nakamoto's original Bitcoin paper [4].

The Ethereum project was first conceived to build a blockchain with more powerful programming capabilities than the Bitcoin scripting, this project aims to build a blockchain with a Turing-complete programming language in order to create a blockchain infrastructure to build projects and decentralized applications on top of it.

Ethereum uses a PoW consensus mechanism whose work function is Ethash and its main internal crypto-fuel is Ether.

2.3.1 Accounts

To build this blockchain, Vitalik first aimed to be able to define a language able to define a state, the transition between states, rules of ownership and transaction formats. To do so there is a need to define a base object to build a state, this object is the account. In Ethereum, accounts are used to build states and are composed by a 20-byte address and 4 fields:

- Nonce: a counter used to track and manage state transitions, in order to build a historic of transitions and to prevent running a transition twice.
- Ether balance: a field used to track the balance of an account, this balance can be modified by state transitions.
- Contract code: The smart contract code that will define the state transitions, rules of ownership, state structure and transaction format
- Account storage: the data state of the account whose structure will be defined by the contract code and modifications and modified by the state transitions of the contract code.

Ethereum blockchain also provides two kinds of accounts:

- Externally owned accounts: these accounts do not contain contract code and can send transactions by creating and signing them.

- Contract accounts: these accounts are the ones that have a contract code. Every time this account receives a message the contract code will be executed. This code performs CRUD operations into the account storage, sends new messages or creates another account.

2.3.2 Transactions

Once the storage of states, balances and contracts is defined, the next step is to define the basic communication unit, transactions. Transactions are signed messages formed by 6 fields:

- Signature: a signature from the sender to identify it.

- Recipient: the recipient account address.

- Amount of ether: the amount of ether to be transferred from the sender to the receiver

- Data: the data to be used by the smart contract of the recipient account if it exists.

- Startgas: a representation of the number of computational steps allowed to this transaction.

- Gasprice: the value per computational step that the sender will pay as a fee.

Comparing an Ethereum transaction with a traditional bank transaction, the first three fields are easy comprehensible, as the Ethereum blockchain needs to know the destination of the transaction, the sender that will send the money and the amount of money to send, but the next three fields, data, startgas and gasprice are related to the Ethereum blockchain, as the data will be used by Ethereum smart contract during its execution and the startgas and gasprice are anti-denial preventions of Ethereum. The anti-denial prevention allows Ethereum blockchain to be robust against:

- Infinite loops in smart contracts.

- Denial of Service (DoS) attacks: if an attacker performs a DoS attack, he must pay a proportional fee as the magnitude of the attack, there is an additional 5 gas units fee per each byte in the transaction data.

2.3.3 Messages

In Ethereum, smart contracts can also send data; to do so, the Ethereum project defines Messages. These messages are only logical representations used to send data from a contract to another contract. Messages are similar to transactions and formed by 5 fields:

- Sender
- Recipient
- Amount of ether
- Data
- Startgas

As messages are produced by smart contracts and smart contracts are triggered by transactions, messages do not need a gasprice field as the gasprice is fixed by the transaction that triggered the smart contract In the first instance.

2.3.4 Ethereum Virtual Machine (EVM)

Ethereum Virtual Machine is the environment used to execute the contract code of an account [21, 22]. EVM reads smart contracts as EVM code. This code is bytecode represented as a stream of bytes that must be read sequentially until the end of that byte stream.

EVM executes the smart contracts' byte code using two main components:

- The Stack: A LIFO data structure where each entry is an operation. Cleaned after each computation ends
- Memory: the memory needed to run the contract, an infinitely expandable byte array. Cleaned after each computation ends
- Long-term storage: Storage that is not cleaned after each computation ends. Represented as a key-value data structure.

This revolutionary way to define an execution environment goes beyond the traditional cloud computing implementations. EVM defines a single entity made from all the computers running an Ethereum client. This distributed computing environment is essentially a distributed state machine able to execute arbitrary code and modify a global state by each execution.

The approach of having a distributed state machine is an evolution from the Bitcoin distributed ledger where the distributed data structure is used to store

the evolution of a global state over the time. This global state contains the account balances, but also each account state for a given time, representation of the EVM state machine and its volatile and persistent components at Fig. 2.9.

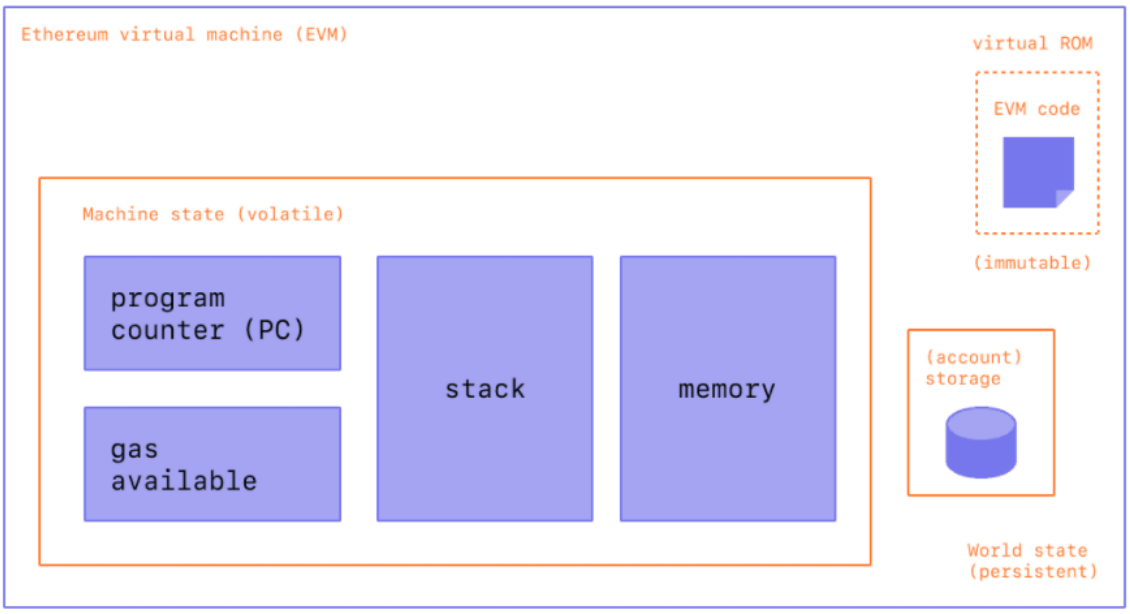

Fig. 2.9 EVM overview, from [22]

There are many EVM implementations [22] that are used in Ethereum execution clients (or ETH1 Clients) [23].

2.3.5 Ethereum summary

To get an overview of Ethereum in order to compare this blockchain proposal with other solutions we can consider several aspects:

- Consensus mechanism: As in Bitcoin, Ethereum blockchain uses PoW as a consensus mechanism. This approach leads to a really decentralized and secure blockchain network but has poor energy efficiency and big issues in terms of scalability and improving the number of transactions per second, currently around 13 transactions per second.

- Programming resources: Ethereum adds a key and interesting feature to blockchain networks, a Turing-complete state machine able to run code from accounts to modify the current state. This addition causes a revolution in blockchain technologies as it permits to build complex applications on top of a distributed network able to compute and store data with high availability and securely.

- Programming language: As Ethereum provides a way to execute arbitrary code in the EVM, there is a need for a way to develop this code

and add it to an account. Here comes the smart contract programming languages such as Solidity and Vyper. The most popular programming language in EVM based blockchains is Solidity thanks to its enhanced features. Solidity is a high-level object-oriented language.

- Contract-data relation: In Ethereum and other EVM blockchains a contract resides in an account and the data that this contract can modify also resides in the account. This fact leads to a high coupling between the logic and the stored data (state) of a decentralized application (DApp).

Ethereum Foundation is currently developing a new version of the Ethereum Blockchain, Ethereum 2.0. This new version of Ethereum network aims to reduce the power consumption of the network, make it more scalable and increase the number of transactions per second.

2.4 Solana

Solana is a new generation blockchain created by Solana Foundation [15].

This blockchain technology was conceived to be a high-performance blockchain network able to handle hundreds of transactions per second, its aim is to provide the features of a centralized database to a decentralized blockchain network, combining high-performance with high availability in a public blockchain, while keeping transaction fees low.

As Ether in Ethereum blockchain, Solana has a native token, the SOL token, and fractional value of that token, lamports, a lamport has a value of 10^{-9} SOL.

2.4.1 Accounts

Like Ethereum, in Solana there is also a need to save a state between some transactions, to do so Solana uses Accounts. An account is a data structure formed by [24, 25]:

- Key (sometimes called address): Typically, the public key of a ed25519 key pair or its hash as a 32 characters string, but for accounts related to a program (a Solana's smart contract) it can be a 32B program-derived account address.

- Owner: A reference of the program that owns this account, the program id, only the owner program can modify the account. By default, the owner program is the System program (whose program id is "11111111111111111111111111111111"), the native program that is able to create accounts, transfer native tokens, pay transaction fees, allocate account data and assign accounts to owning programs.

- Data: A byte stream of data typically serialized using borsh serializer. Data is modifiable by the owner program

- Lamports: The number of lamports an account owns, native tokens are always stored as lamports in order to ease transactions and program executions.

- Executable: A Boolean flag that marks if an account holds the bytecode of a program in its data field (an executable account) or if the data field is data that can be used by the owner program (non-executable accounts).

- Rent_epoch: A field representing the epoch when the account must pay rent.

In Solana, accounts need to pay a rent in order to keep alive in the blockchain. This rent is paid every several epochs, but if the account holds at least 2 years of rent, this account is marked as rent-exempt. When an account is marked as rent-exempt the holded lamports of this account will not decrease by the effect of paying rents, but the rent-exempt state will be checked every time the account lamports amount is reduced.

2.4.2 Transactions

In order to modify the state of an account in Solana blockchain, there is need to trigger a program execution, it can be a native program, as the system program (ex: to send lamports to another account), or a custom program. Here is when the transaction concept comes up, when a transaction reaches a cluster, the Solana Runtime of this cluster starts the execution of a program.

Transactions are received in binary format, containing:

- Signatures: A compact-array of ed25519 digital signatures. These signatures are verified by the Solana Runtime using the same index element in the account address array of the message. Solana Runtime also verifies that the number of signatures is equal to the fist 8-bit unsigned integer in the message header.

- Message: The message can be seen as the payload of a first layer protocol, as an IP packet packed inside an ethernet frame. This “payload” has the following parts:

    - Message header: Three 8-bit unsigned integers. The first one is the number of required signatures, the second one the number of addresses of read-only accounts and the third one is the number of read-only addresses that does not require a signature.

    - Account addresses: A compact-array of account addresses. First are placed the ones that require signature, then the addresses that do not require signatures. In each block, there are first placed

the read-write account addresses and then the read-only account addresses.

- Recent blockhash: This is used to place the transaction in the PoH chain. The sender will put the recent blockhash from when he last observed the ledger, and the Solana Runtime will reject the transaction if the recent blockhash is too old.

- Instructions: A compact-array of instructions.

The structure of an instruction is the following:

- Program id index: 8-bit unsigned integer pointing to the program account address in the account addresses array.

- Account addresses indexes: a compact-array of 8-bit unsigned integers, each entry pointing to an account in the account addresses array.

- Data: In Solana instructions data is sent as an opaque 8-bit compact-array.

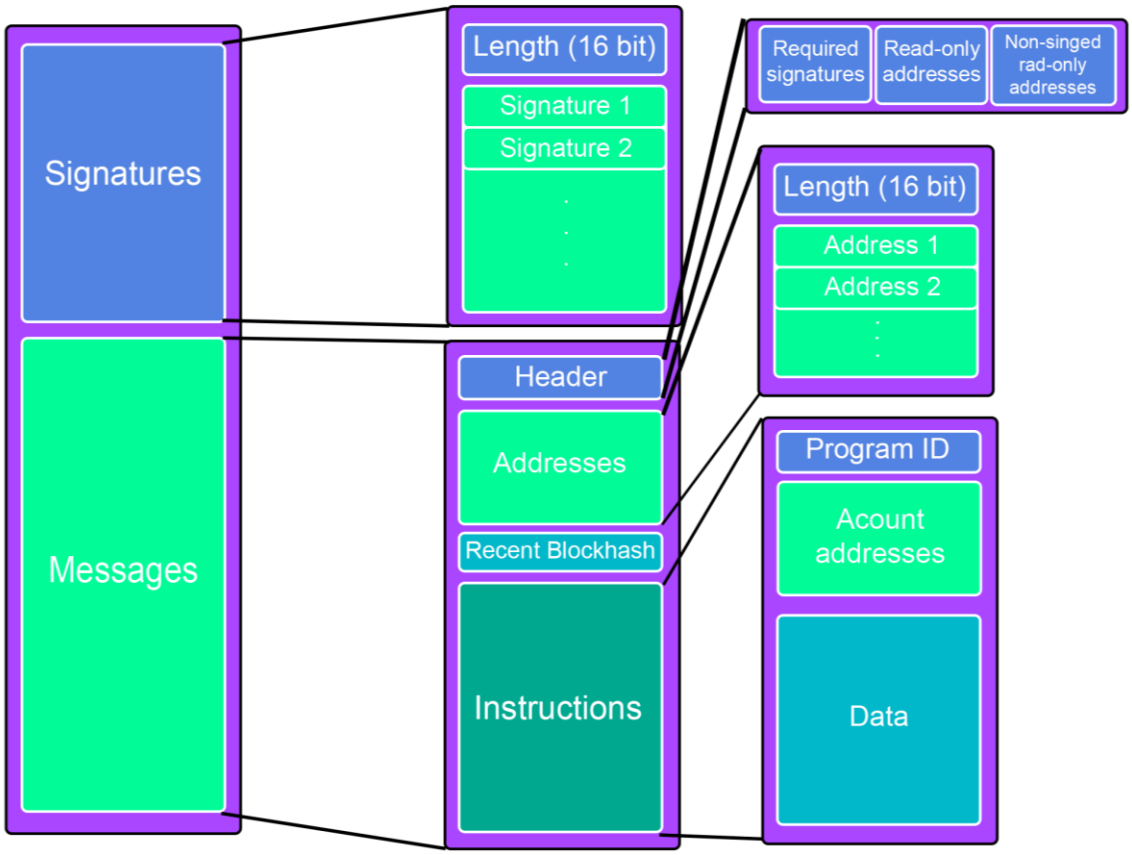

Fig. 2.10 Solana transaction structure

When a transaction is received by the Solana Runtime, it will execute the transaction atomically, and in order.

2.4.3 Sealevel (Solana Runtime)

As the EVM was an incredible evolution from the Bitcoin ledger, Solana tries to push this concept of a Turing-complete processor to a multiprocessor in order to compute in parallel several smart contracts.

Solana blockchain proposes the Sealevel [26], the Solana Runtime capable of running smart contracts (or programs in the Solana environment) in parallel and verifying that this program can run an instruction for a given account (or set of accounts).

Sealevel provides the capability to increase the number of transactions per second this blockchain can handle.

Sealevel verifies that the program can run an instruction for a given account following these policies:

- Only the owner can change the account owner field and if follow some rules:
    - The account is writable.
    - The account is non-executable (is not a program).
    - The account data is empty.
- Only the owner of the account can modify its balance.
- The balance of read-only and executable accounts must not change.
- Only the system program can resize the data and only of accounts this program owns.
- Only the owner can change the account data and following some rules:
    - Only for writable accounts.
    - If the account is non-executable (is not a program).
- Only non-executable accounts can become executable and not in the other way.
- Only the owner can make an account executable.
- The program can’t modify the rent_epoch field, only Solana Runtime can do it

2.4.4 Solana's "triforce"

Solana has 3 interesting mechanisms in order to provide these enhanced features.

The first one, and one of the key proposals of Solana blockchain is the usage of PoH in order to build the chain and get a robust ledger. This mechanism provides a way to create a timeline made of related timestamps (section 2.1.5). This approach of using a VDF to build a timestamp chain allows the system to combine an event with the input of the next VDF iteration (the output of the previous iteration), in order to ensure that a certain event has been produced before a specific VDF iteration. PoH also allows the Solana blockchain to handle a high number of blocks per second and it is not limited to producing a block every 10 minutes as in bitcoin or 15 seconds as in Ethereum, that use PoW in order to build the ledger.

The second powerful mechanism is related to how data is stored, shared and retrieved. For all these key points, Solana implements the usage of PoRep (Section 2.1.7) an evolved mechanism from PoSpace (Section 2.1.6). This mechanism was firstly proposed to provide the Solana network a mechanism to measure the space taken for a given account and for the whole blockchain state [15].

The usage of PoRep also provides Solana a mechanism to ensure a high-availability of stored data (ex: accounts) while keeping restrained the amount of resources needed, such as storage and bandwidth.

The last key mechanism that gives these enhanced features to Solana blockchain is its consensus mechanism. Its consensus mechanism is a PoS based solution (section 2.1.2). In Solana's PoS implementation, the PoS mechanism is used to select the next leader validator; that leader validator will also be the next PoH generator, and the rest of the validators must vote if this block is correct in a given time period. If this block is accepted, it will be added to the ledger.

The leader role is rotating continuously following a leader schedule, this schedule defines a leader validator per each slot of an epoch. The leader schedule for epoch N is computed at the start of epoch N-1 using the last PoH tick and ledger state of the start of epoch N-1. The leader schedule generation follows these steps:

1. Using a seed, the last PoH tick, each validator runs the same stable pseudo-random algorithm.

2. Each validator consults the balance of each stacking account that points to a validator that has voted in the last cluster-configured number of ticks (typically last epoch). This set is called the active set.

3. Each validator sorts the active set by the balance of the stacking account.

4. Using the output of the stable pseudo-random algorithm and the sorted active set, each validator will build its own leader schedule.

Using the previous leader schedule, each validator only can vote as valid the confirmed block of one validator per slot (the leader validator computed by itself for a given slot). If a confirmed block doesn't reach a minimum amount of votes in a given time, the block is discarded.

2.4.5 Solana programs

Another remarkable point of Solana blockchain is its programming model. To enhance Solana's network capabilities and resource optimization, Solana proposes a programming model based on a low-level programming language compiled into Berkeley Packet Filter (BPF) bytecode [29].

To do so, Solana allows to develop programs to its blockchain using Rust and C programming languages, but Rust is the one recommended by Solana Foundation, as it is a modern low-level programming language that provides thread-safety through all data and code and better control of memory usage, due to these advantages, Rust as programming language for Solana blockchain has more documentation from its community.

2.4.6 Solana summary

Once at that point, with the key points and revolutions of Solana been reviewed, we can analyze the strengths and weaknesses of this new-generation blockchain:

- Consensus mechanism: Solana blockchain, unlike Bitcoin or Ethereum, use PoS as a consensus mechanism and PoH as a mechanism to build the blockchain. This combination of mechanisms provides Solana blockchain a fast and robust blockchain without sacrificing scalability and power efficiency.

- Programming resources: Solana proposal goes beyond the idea of a blockchain based Turing-complete state machine and proposes the usage of a blockchain based "multiprocessor" as it can process multiple transactions simultaneously.

- Programming language: To enhance the number of transactions per second in Solana, low-level programming languages are used to develop on top of this blockchain. This decision builds a considerable entry barrier as these languages are not usually known by traditional application developers. It is also much harder if we consider that the recommended programming language for Solana programs is Rust, a

modern low-level programming language in which there is not yet a large number of developers.

- Contract-data relation: Unlike in EVM based blockchains, in Solana network the relation between a contract and the data it can modify is not fixed to belonging to the same account. In Solana, if a user wants to use a program, this user only needs to create an account whose owner is the program he wants to use. Another remarkable point is that if a user doesn't want to use that program anymore, he can delete its account by sending the lamports used to maintain this account to his wallet.

Solana has been criticized by arguing that its blockchain is a centralized network. These accusations came from some facts:

- At the moment, a great part of Solana's main cluster (Mainnet) are Solana Foundation's validators.

- Currently, Solana blockchain has some features that are accepted by Solana Foundation, but not implemented yet. This fact added that the code is only created by a single organization and feeds these accusations.

- Nowadays, there is only one way to set up a Solana validator, and this way is using the Solana Foundation validator code.

These three points can be interpreted as Solana Foundation is actually building a centralized blockchain, but as this project and its blockchain is not as mature as other blockchain solutions and Solana Foundation is continuously encouraging the usage and development of its blockchain, we can hope that these restrictions can be solver in the near future.

2.5 Ethereum and Solana Comparison

To decide at which blockchain develop our solution we will compare both blockchains in the following points:

- Blockchain trilemma: A trade-off between scalability, security and decentralization.

- Capabilities: Runtime capabilities of each blockchain, the amount of transactions per second each blockchain is able to manage, the delay of transaction in each blockchain and gas fees of each blockchain.

- Development: The amount of documentation and community support for developing in each blockchain.

- User data control: The control a user has on his data.

We will discuss each point of this comparison in following sections

2.5.1 Blockchain trilemma

The blockchain trilemma consists in a trade-off between three key points of a blockchain network: decentralization, security and scalability.

Firstly, we will compare decentralization properties of each network. Starting with Ethereum, it uses PoW as a consensus mechanism, which leads to a very decentralized network, as PoW is also used to build the data chain as a distributed timestamp server.

On the other hand, we have Solana, whose blockchain proposal consists in the usage of PoS as the consensus mechanism and PoH as a distributed timestamp server to build the data chain. This choice of this mechanism provides Solana blockchain a decentralized way to operate, but, as mentioned in section 2.4.6, the software-development of this network is done in a centralized way, as it only exists a client to build a validator and only one organization controls its development.

Secondly, we will compare the security of each network. Starting with Ethereum, as it uses PoW, it builds a very resilient ledger, but it is also vulnerable to 51% attacks, this 51% attacks becomes more feasible to produce for a big company as nowadays there are ASIC (application-specific integrated circuit) Ethereum miners.

Following Solana, as this blockchain uses PoH to produce the data-chain, it protects the blockchain against some PoS possible attacks such as long-range attacks and reversal attacks. On the PoS part, Solana has an accepted version of slashing, the solution to nothing at stake attack, but, currently, it is not implemented yet.

Thirdly, we will compare the scalability of each blockchain technology. On the Ethereum hand, it has a serious issue with scalability, this blockchain has a low number of transactions per second, around 13, and has an exorbitant power consumption, a fact that collides with the SDG goal of responsible consumption and production, discussed at chapter 1.

On the Solana hand, it has a very high number of transactions per second, around 3000. This solution also has a reduced power consumption compared with the ones.

Once all these three key points have been discussed, we can decide what blockchain proposal is better in each category and the reason of this chose, this election is summarized in the following table:

Table 2.1. Blockchain trilemma comparison between Solana and Ethereum

	Winner	Reason
Decentralization	Ethereum	Nowadays, with the latest Solana version being the 1.8.14 in Mainnet, we cannot affirm that Solana is a fully decentralized blockchain proposal.
Security	Ethereum	Although Solana's whitepaper proposes a really robust blockchain, the implementation is not complete yet, currently having Mainnet release 1.8.18. Current implementation of Solana does not support slashing, a basic mechanism to punish non-honest nodes in PoS blockchains.
Scalability	Solana	Solana Is by far more scalable than Ethereum, decoupling consensus mechanism and data-chain building provide Solana the capability of having a robust and scalable way to build the ledger and achieve a consensus between all its nodes.

2.5.2 Blockchain capabilities

As in the previous section we did an analysis from the blockchain trilemma point of view, in this section we will analyze the capabilities of each blockchain, and which one has better user experience.

On one hand, there is Ethereum, a revolution compared to previous blockchains. It enhances the power of first blockchains, such as Bitcoin, by providing a Turing-complete state machine to the blockchain. This new feature allows developers to build complex applications using Ethereum blockchain as infrastructure. Although all these improvements from previous blockchains,

Ethereum presents several problems when using complex applications built in top of this proposal:

- Large gas fees: Ethereum blockchain has an issue with gas fees as they can grow faster, as having gas fees up to 95$ in some popular decentralized finances (DeFi) projects as SushiSwap or Crypto.com

- Low transactions throughput: Ethereum, currently, has a low number of transactions per second and very scattered delays, going from 30 seconds to 16 minutes.

These two limitations of Ethereum blockchain are wanted to be solved by the Ethereum community at the Ethereum 2.0 release.

On the other hand, Solana goes beyond a single Turing-complete state machine to propose a runtime capable of processing several transactions at the same time. This design of Solana has several benefits from the enhanced capabilities point of view:

- Low gas fees: Solana solution has a contained power consumption and reduced fees, in dollars at Solana's maximum value at the moment of 259$ = 1SOL there is a fee around 0.00015$.

- High transaction throughput: Solana blockchain has a very- large number of transactions per second compared to EVM based blockchains and delays around 400ms.

Once the capabilities of each blockchain technology are reviewed, we can conclude that Solana blockchain provides higher capabilities than current Ethereum blockchain.

2.5.3 Development

Once we had analyzed the blockchain trilemma solution of each proposal and their capabilities, we will analyze the development facilities of each proposal.

Starting with Ethereum, its most used programming language is Solidity, a friendly object-oriented high-level language whose syntax is similar to JavaScript. Ethereum also has a big advantage, as it has a big community, there are a lot of documentation, tutorials and examples online. There are also more formal courses about developing DApps over Ethereum blockchain.

Following by Solana, its more supported programming language is Rust, a modern low-level programming language. Another withdraw of developing in Solana is the usage of Borsh serializer to send and store transactions and account data. Borsh serializer has some libraries in some languages to ease the development of systems using this serialization mechanism, but some of them are not fully-featured respective the Borsh specification [30]. As Solana

project has started growing recently, there are not a huge number of complex programs examples, tutorials or mere formal courses.

Seeing all these facts of the development on each platform, we can conclude that, currently Ethereum has a smaller entry barrier to start developing on top of that blockchain.

2.5.4 User data control

As we want to develop a fairer blockchain solution that the traditional application development would provide, we will analyze the control that a user can take in solutions based on these blockchains.

From the Ethereum side, the user data is typically stored in contract's account data, hard-coupling user data and the smart contract, or in third party servers, where the data owner is the DApp organization.

From the Solana side, user data is saved in accounts holded by the users but owned by the program that can modify this data. This relation decouples data holding from the program. This solution allows the user to remove all its data by transferring the lamports from his account related to the DApp program to his wallet.

On this point, Solana provides a most fair solution for storing data, allowing users to be owners of their data and not giving all their data to a huge corporation.

2.5.5 Comparison Results

Following the previous mentioned results, we have built two tables, one summarizing all technical aspects of each blockchain and another including all the conclusions of the previous sections and choosing one as the one to be used in this project.

Table 2.2. Comparison between Solana and Ethereum main features

	Solana	**Ethereum**
Current transactions per second	3,000	12
Theoretical maximum transactions per second	65,000	17
Transaction fees	Around 0.00015$	Around 15$ (Up to 95$)
Transaction Delay	400ms	30s – 16min
Consensus Mechanism	PoS (+ PoH)	PoW
Development languages	Rust, C	Solidity, Vyper
Multi-threading	Yes	No

From the previous table, we can conclude that Solana is a faster and cheaper blockchain to use, but Ethereum has a more tested consensus mechanism and higher-level languages respect the Solana ones.

Table 2.3. Comparison between Solana and Ethereum results

	Solana	**Ethereum**
Blockchain Trilemma	1/3	2/3
Capabilities	1	0
Development	0	1
User data control	1	0

Seeing both tables, we can conclude that Solana is a more interesting platform to develop this project as it provides almost all the Ethereum features but with some improvements. Respective to the points that Ethereum currently beats Solana, there are accepted solutions and developments to provide these features to Solana [15, 31].

CHAPTER 3. MULTIPLATFORM DEVELOPMENT

In this chapter we will expose the initial architecture proposal of our proof of concept of an application built to reward the civic actions of a citizen and, aligned with SDGs principles, reduce the barrier from new technologies solutions to more underprivileged citizens.

Once the use case and our initial architecture proposal has been exposed, we will face the technical difficulties of native developments and we will follow by analyzing the most popular alternatives to build a multi-platform solution.

3.1. Use case

Based on the SDGs and blockchain technologies, we built a service where a citizen with an internet connection could registry its civic actions and earn rewards with these civic actions.

To do a project that aims to be accessible to everyone we need to build a friendly UI with an easy UX, and in order to ease the usage of this application to people with difficulties with common smartphones' applications we will focus our user interface design in having clean pages and having an auto explicative layout.

Another key point in the success of this project is the aim to hide all blockchain complexity below the user application.

For the technical part, we will deploy a backend and database equivalents in the blockchain environment and enhance the fairness of the solution by adding an autonomous event validator, architecture in Fig. 3.1.

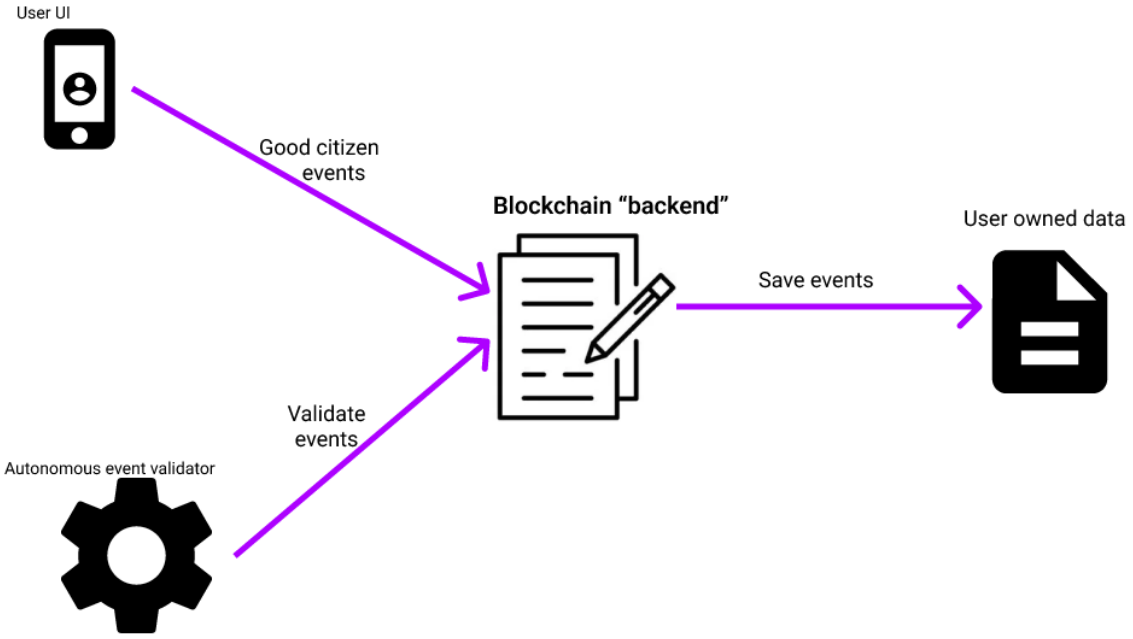

Fig. 3.1 First approach of our solution's architecture

3.2. Native development

Once we have chosen a blockchain to build our application on top, we need to choose a way to develop our application.

As we aim to build a multiplatform application, the intuitive way to face this development is to build an application for each of the most popular official app stores: Play Store (Android devices) and App Store (iOS devices). This decision would allow us to develop an application in Kotlin, a language developed by JetBrains that runs over the Java virtual machine; and an application using Swift, a language developed by Apple, based in Objective-C and used to develop applications in Apple's operating systems (OS).

Both languages follow a multi-paradigm approach focused on providing the features of an imperative, object-oriented and strongly-typed language.

This traditional approach has some advantages in execution time, but several drawbacks in development time.

On one hand, the main advantage of this approach is that developing an application in the native language of each platform produces very efficient applications, with great performance even if the application uses complex animations and media resources. These benefits lead to a better user experience (UX) in each platform.

On the other hand, there are several drawbacks of developing an application in different languages for different devices, but they can be resumed in two main points:

- Version inconsistency: having an application in each platform, implies to have a project per device. It increases the possibility of having different bugs in different devices, increasing the effort needed by the development team. This approach also creates the possibility of having some features in the application of a device, but have not implemented these features in other devices yet. These two drawbacks can be mitigated by using application testing frameworks as Katalon, a framework to test mobile applications, web applications and APIs, but it increases even more the developing efforts.

- Developing times: Due to having multiple projects in different languages, the developing team needs to have developers with knowledge of each framework, increasing the costs of developing the application, and to split its efforts in different projects, increasing development times and impacting final application costs.

In order to avoid these drawbacks but without scarifying the overall user experience, we will analyze some multiplatform development frameworks and choose the one that better fits for this project.

3.3 Angular

We will start with one of the most used frameworks in the last 5 years, Angular. This Google maintained framework provides an easy way to develop web based multi-platform applications. AngularJS was initially published in 2010, but Angular2 (currently just Angular) was published in 2016.

Angular uses a model-view-controller (MVC) to build single-page applications (SPA), angular applications usually have good user experience as the approach of building a SPA combined with the angular-material package of Angular allows the development team to build fluid applications based on material-design guidelines.

Angular framework is based on the concept of component. An Angular component is the basic building block of Angular framework. Each component has the following structure:

- A selector for which the component can be used in another component's template.

- An HTML template used to declare what is going to be rendered in this component. In this HTML template can also be used some structural directives (such as ngIf or ngSwitch) to dynamically render parts of a

component. In the HTML template of a component, there are also specified the connections between components.

- A TypeScript class used to define the behavior of a component and handle the connections between components, called as data binding in Angular framework.

- A CSS (or SCSS) file to define the component's style and animations.

Angular has the big advantage of having only one project to produce a multi-platform application, but it has also some drawbacks:

- Angular natively does not support state-based application architecture, such as following the Redux architectural pattern, but it can use third party libraries such as NgRx and JsRx to build this state-based architecture.

- While the usage of Angular with the Ionic framework works well to develop applications with an improved UI design and enhanced UX. These frameworks do not have good performance in mobile devices, having performance issues such as frame-rate drops and lag in animations due to using JavaScript Bridges to run these applications.

Aiming to solve the previously mentioned performance issues of Angular applications in mobile devices, such as Android or iOS devices, the JavaScript and TypeScript community developed NativeScript, an open-source framework to build native applications for iOS and Android devices. Although NativeScript solves a lot of performance issues of running Angular applications using JavaScript Bridges, this solution requires higher expertise of the developer in using Android and iOS native APIs.

3.4 React

Once Angular has been analyzed, we will follow with one of the most used libraries to UI development, React. This library is a JavaScript library developed by Facebook and initially published in 2013. React, as Angular, is used to build SPA but, without an inherited architecture, as the MVC used by Angular, but the typical React architecture is based in Redux

React is also based in components, but React components are developed in a declarative programming paradigm where the developer can specify how the component should behave without writing explicitly the steps to get this behavior. React also uses JSX, a syntax extension of JavaScript that allows embedding UI templates in JavaScript code.

React is typically used in combination with Cordova to build mobile applications, but this combination has similar limitations to the Angular and Ionic one, having a huge impact on its performance due to the usage of JavaScript Bridges.

Similar to NativeScript in Angular, in React there is React Native, this framework allows React applications to translate the HTML template into native templates, but the core logic of a React application is still running in a background JavaScript thread. This approach enhances the performance of React applications, but it does not reach the performance of native applications.

3.5 Flutter

As we have reviewed the most popular web-based frameworks to build multi-platform applications, we will continue by analyzing a more modern multi-platform applications SDK, Flutter. This framework is developed by Google and published its first alpha it 2017 and its 1.0.0 in 2018.

Flutter is an SDK and framework to build multi-platform applications in Dart, an open-source programming language developed by Google. Dart was originally designed to build client-side applications. This new programming language has a functional object-oriented paradigm and modern features as an efficient garbage collector.

Dart has three main targets to run its code [32]:

- Dart Virtual Machine: Dart VM is mostly used to develop dart code in desktop environments, but can also be used to run non-demandant applications

- JavaScript compiled: Dart allows to build web applications, to run these applications, Dart SDK provides two compilers to translate Dart code to JavaScript, dartdevc compiler for development purposes and dart2js compiler to build production applications.

- Native code compiled: Dart SDK also brings compilers to build applications for Android, iOS, ARM32 devices, ARM 64 devices and x86 devices. These compilers allow Dart developers to build applications even for embedded devices.

Dart also provides another interesting feature, hot reload, this feature allows Dart developers to just-in-time compile new changes in Dart code, accelerating development and debugging.

From the Flutter SDK and framework part, we get a declarative paradigm to build UIs in a framework based on providers that has support of several development patterns as model-view-controller (MVC), model-view-view model (MVVM) or Redux.

Flutter also provides an approach to build fluent animations using Skia as a graphic library. This graphic library developed by Googles is able to profit from GPU-accelerated render technologies such as Vulkan, Metal or OpenGL ES.

Flutter is based on widgets, as Angular or React in components, but Flutter widgets came with a more generic way to define these building blocks. In Flutter, even the style of application widgets is a widget. All widgets are immutable, but based on if a widget must change with some user actions there are two main categories:

- Stateless widget: These widgets have no stat, once created they will not be refreshed by user actions.

- Stateful widget: These widgets have an instance of a state. This state is mutable and contains the mutable data needed to rebuild the widget and the widget's build method.

Flutter allows developers to build high performance multi-platform applications in a fast and declarative paradigm, reducing the development costs of Flutter applications.

3.6 Comparison

Once analyzed the pros and cons of each technology, we are able to choose the one to develop our multi-platform application on top.

To get a conclusion we will compare the previous analyzed technologies in 4 dimensions:

- Performance in multiple devices: In this section we will compare the performance of each technology in mobile platforms.

- Development: In this section we will compare some basic development concerns as how fast is to develop in each technology and the online resources available for each technology.

3.6.1 Performance

From a performance point of view, currently, there is nothing better than native applications, but due to its development costs there are more interesting approaches to build an application.

Angular with NativeScript has good performance compared to more traditional approaches of this framework, such as using Angular with Ionic, but it has some performance issues as the size of an application is quite high and it has performance issues as frame-rate drops.

React, since React Native was released, has grown in popularity thanks to its improved performance compared with other multi-platform approaches. However, React Native has issues rendering animations as these animations

cannot be rendered at 60 FPS and also have frame-rate drops and variable time between frames when rendering complex animations. Although React Native is able to profit better from an isolated JavaScript thread, it also uses JavaScript bridges that have a performance impact.

Flutter has not as big community as the previous approaches, but its community has been growing during the last few years thanks to the performance of flutter applications in mobile platforms. Another remarkable achievement of Flutter is the ability of rendering animations at 60 FPS with minimal frame-rate drops and a constant time between frames. This enhanced performance from Flutter applications is thanks to dart-to-native code compilation and the usage of a graphic library able to profit from modern graphic APIs and graphic dedicated processors' power.

In spite of all the improvements from first multi-platform frameworks to latest ones, from the performance point of view, it is still better to develop an application in a native language than in a multi-platform framework.

3.6.2 Development

Once we have analyzed each approach by the performance point of view, we will continue by analyzing the development costs and online material of each approach.

Starting by native application development, mobile platform programming languages such as Java, Kotlin, Swift or Objective-C have a lot of documentation online, but these languages follow an imperative paradigm and their development times are quite long. In addition, if we want to be present in each of the most popular official app markets, we will need to develop at least two applications, one for Android and one for iOS, greatly increasing the development times and costs.

Following with Angular, this framework has a lot of online documentation and a lot of community supporting this framework, providing tutorials, examples and even more complete courses. The approach of Angular to build a UI based on components allows to speed up the development as, once the UX designer builds the application mockups, the development team can break the views of the application into components in order to reuse as maximum as possible each building block. Once a building block is developed and tested it can be used in many locations of an application speeding up its development.

Once Angular is reviewed, we can continue with React, this approach has similar advantages as Angular, it has a big community that provides a lot of tutorials, examples and professional courses. As Reacts is also based in small building blocks, it is able to build some tested components and reuse them to speed up the development of an application, reducing its development costs. Respectively to Angular, React has the advantage of being designed to be used in a declarative way, easing and accelerating even more the development of applications. React also has a drawback compared to Angular, using React in

large applications can easily lead to a non-clean code due to not having the UI template, its logic and its style split by the framework in different files.

Finishing with Flutter, this SDK doesn't have as huge community as Angular or React, but its popularity is growing through the past years. Fortunately, Flutter has a very detailed documentation where each widget has a page with examples and with a video-tutorial of how to use the widget and use-cases. The Flutter community also has a lot of tutorials, examples and professional courses. In terms of development times, Flutter has an incredibly fast development workflow, with a good analysis of the application's mockups a developer can face the development by building some reusable widgets and placing them in the widget tree of the application in a very fast way, reducing the development costs.

3.6.3 Comparison Results

After analyzing each approach from the performance and development costs points of view, we can conclude that Flutter is the most adequate solution to build a friendly application with a good UX.

We have chosen Flutter thanks to its enhanced performance and its development advantages, such as the speed of development and the ease to develop applications following the material design or Cupertino guidelines.

CHAPTER 4. DECENTRALIZED APPLICATIONS

In this chapter we will use the technologies chosen from previous chapters and we will build some architectural proposals using these technologies, Solana and Flutter.

Firstly, we will expose the main benefits of decentralized applications.

Then, we will evolve the basic architecture of almost all current applications to a blockchain-based generic architecture.

After this, we will expose two applications that we have developed to get into developing DApps and splitting the main features of our final proof of concept into two smaller problems.

Once we have exposed our previous developments, we will propose the final architecture of our proof of concept.

Last but not least, we will create the mockups of our application and its basic features.

4.1 Decentralized applications main benefits

Traditional service developments had expensive CAPEX, but also important OPEX to maintain the infrastructure of the service running and up-to-date.

These costs have been reduced during recent years, in major part thanks to the introduction of cloud computing with infrastructure as a service (IaaS) platforms, such as Microsoft Azure and Amazon web services. These platforms allow service providers to build services without having a huge and complex data center to serve its software solutions.

IaaS platforms are based in provide the infrastructure of project in a fast and programmatically way, giving providers the ability to only pay for what they are using at a given time, ex: if a provider has a load balancer with two instances of a given service, it can set up a third instance when a peak of demand occurs without having this third instance permanently up.

Companies using these cloud computing solutions usually need a DevOps team, this team is responsible for developing software able to use IaaS providers APIs and manage the resources needed to maintain the solution they are serving.

As cloud computing came to reduce the huge amounts of investment a small company must spend to set up and maintain a new service, small companies and startups are moving to creating DApps, this new approach of creating services allows companies to profit from the benefits of blockchain technologies.

4.2 From Traditional Apps to DApps

DApps are a growing alternative to build services, as the company providing the service can forget about investing in renting and managing its IaaS needs.

Generally, DApp solutions also have some OPEX as these solutions lead the service provider to the need of getting a wallet in the chosen blockchain and at least an account to deploy its smart contract and store its data. This account usually needs some tokens to be maintained by the blockchain.

As we chose Solana in section 2.5.5, we will discuss the transition from a traditional application proposal to a Solana based DApp, this approach will change depending on the chosen blockchain, as EVM based blockchains will have a different approach.

In a traditional application-based service architecture, we have a frontend (the application itself), a backend to manage all the data and ask for data to the database, represented in Fig. 4.1.

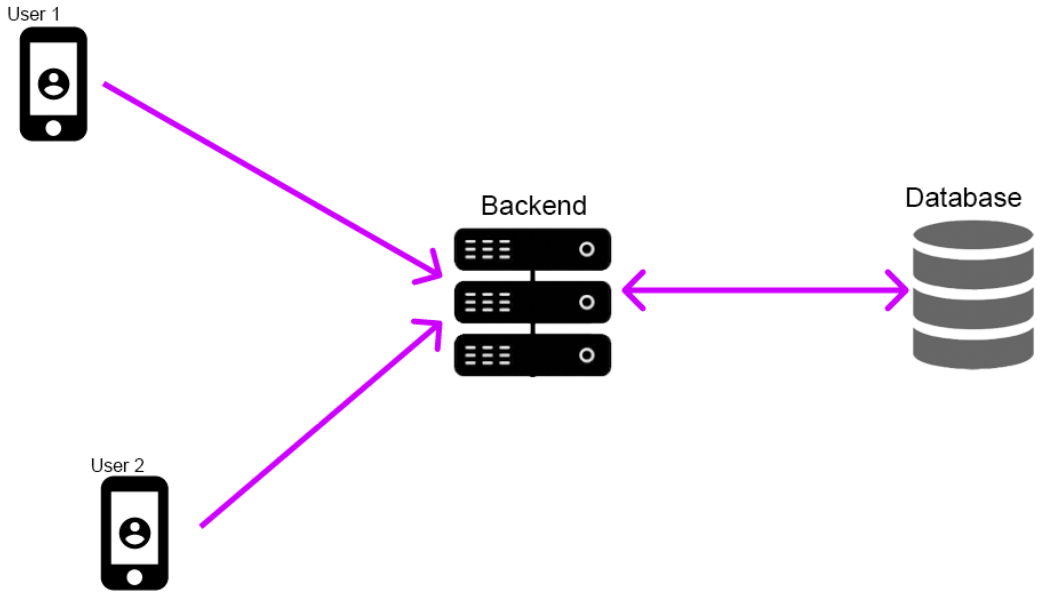

Fig. 4.1 Representation of traditional App-based solutions architecture

Another architecture proposal can be a frontend that directly consumes a service like Firebase, a cloud computing-based platform that provides a way to develop authentication mechanisms, file transfer, message solutions and database storage as services, this approach allows to develop application-based solutions that does not have a backend properly speaking, instead, this architecture is based in an all-in-one service solution, represented in Fig. 4.2.

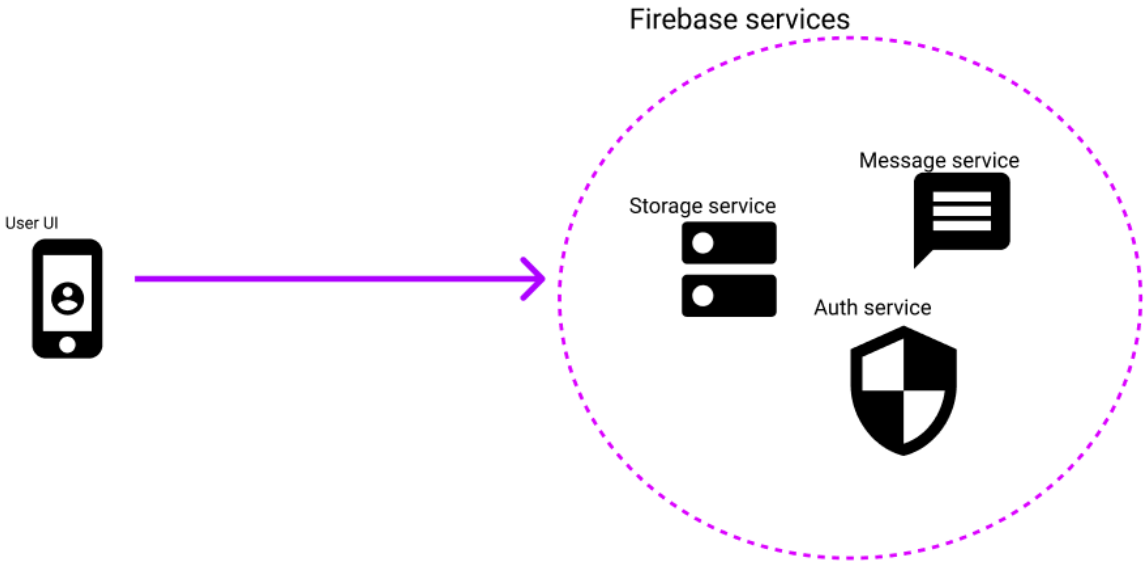

Fig. 4.2 Representation of Firebase-based solutions architecture

DApps architecture has a mixture of both previous architectural solutions, having the control of having a dedicated data processor entity, a decoupled data storage entity but without having to trouble with traditional backend solutions such as Spring or Express.js.

Having as base the traditional App-based solutions architecture, we can change the components of this architecture without performing a big modification of the architecture itself.

Starting with the backend component, its main features in a traditional application architecture are acting as an authentication server, as a mediator between the application and the database and as a mediator between different users. All these features can be implemented in a smart contract, or program in Solana. A program can get the request of a user and send the needed data to the desired account, it can also verify that only some public keys with its signatures can perform some actions.

Following the database component in the traditional App-based solutions architecture, in this basic DApp architecture we can substitute this storage component, typically a traditional database, by blockchain accounts. In a Solana-based architecture each entity, ex: each user and the application provider, can have one or more accounts to store their data and once this data is not needed anymore, ex: a user doesn't want to use the application, the account storing this data can be removed by sending all its tokens to the user wallet. This approach makes DApp solutions more transparent to users than traditional application solutions.

The overview architecture can be seen in Fig. 4.3.

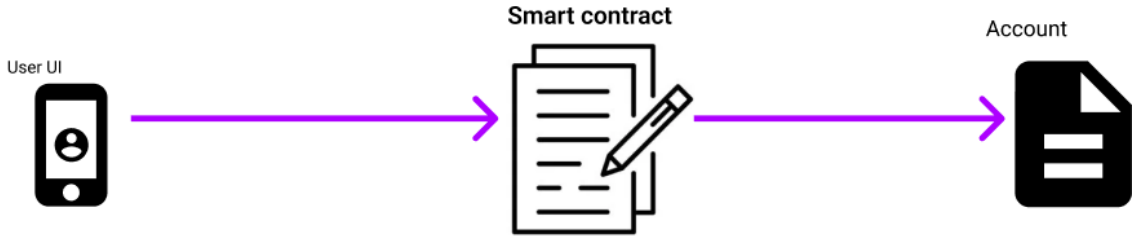

Fig. 4.3 Overview of a basic DApp solution architecture

Although the previously described architecture is a basic architecture to DApp solutions, as traditional application-based solution architecture, DApp solutions can have a more complex architecture, the equivalent of a microservice architecture for a traditional application. This architecture can be achieved by splitting the features of the main program to several programs (ex: one to manage the authorization, authentication and accounting (AAA) features, another to manage how the data must be stored, etc.). Solana provides a complete documented feature called cross-program invocation (CPI) to send transactions between programs and build a “microservice” blockchain-based architecture, represented in Fig. 4.4.

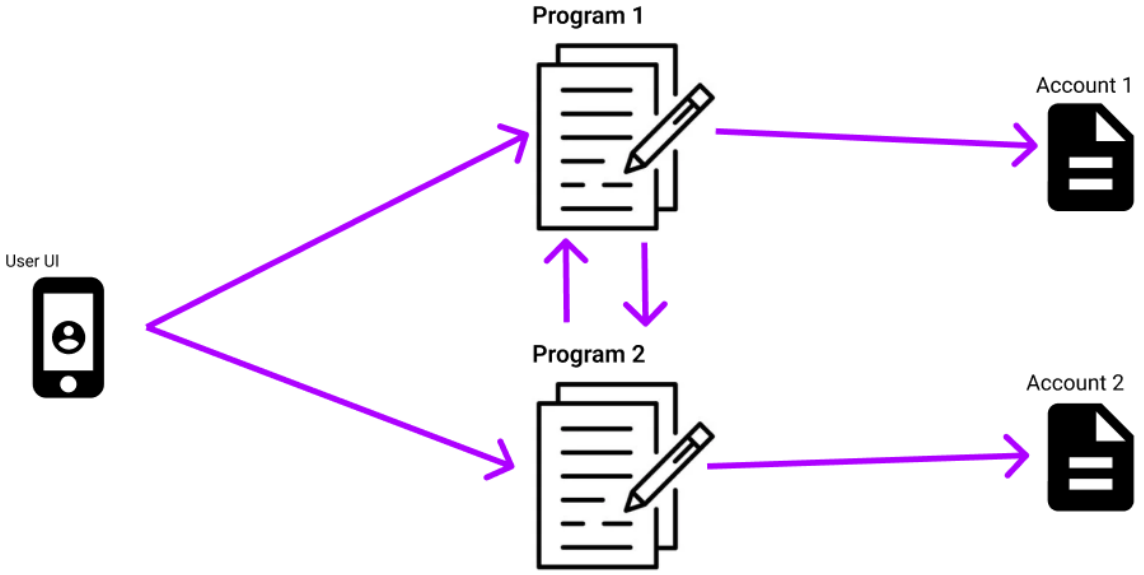

Fig. 4.4 Overview of a “microservice” DApp solution architecture

4.3 Divide and Rule

We can start analyzing the features our application should have. We are going to develop an application whose main objective is to reward the good actions of citizens, such as recycling thrash or warning that there is a flaw on a public

road. We must build an application where a user could upload an action, this action must be processed by the program and stored into the user's account, once the action is accepted to have a reward the program must be able to send lamports from the acceptor entity to the user that upload the action.

To provide our application from the previous features we have developed two smaller applications to face each one a part of the solution:

- Transactions application [33]: An application where a user can send tokens from its wallet to another wallet and review all the transactions its wallet has performed. Main view of the application in Fig 4.5.

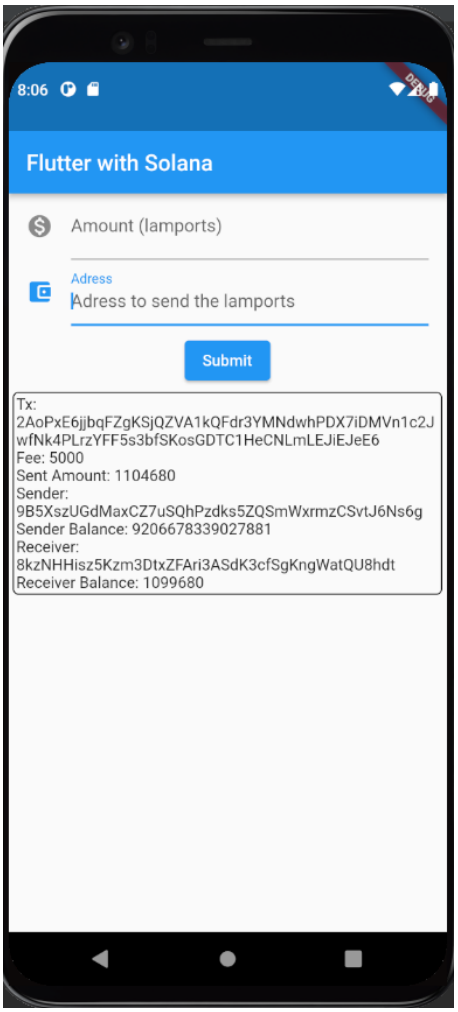

Fig. 4.5 Main view of the transaction's application.

- Chat application [34]: An application where a user can send data (messages) to another user's account and retrieve the messages sent to its account. Application screenshots at Fig 4.6.

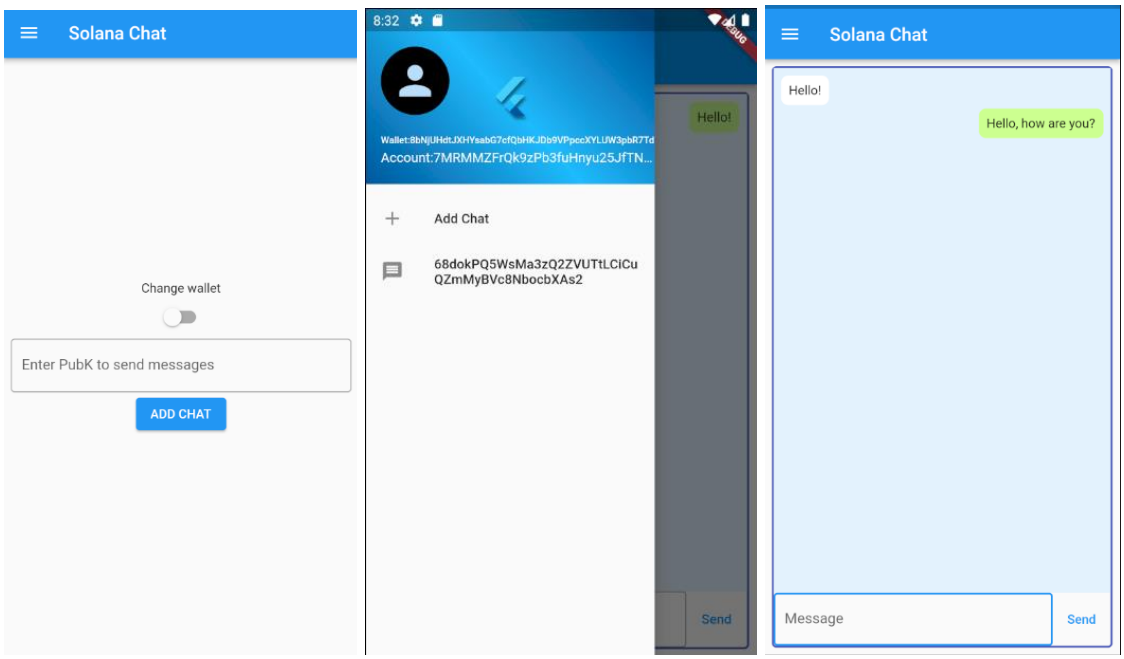

Fig. 4.6 Views of the chat application.

Based on these two applications we can start with our application final proposal.

4.4 Architecture

Based on the pre-study made at section 1.2, the architectures proposed at section 4.1, the small applications developed at section 4.2 and the choices of Solana blockchain at chapter 2 and Flutter as multi-platform development framework led us to propose an architecture based on five main components: a frontend application, a Solana program, some Solana accounts and an autonomous event validator. See Fig. 4.7 with the overview of our architecture.

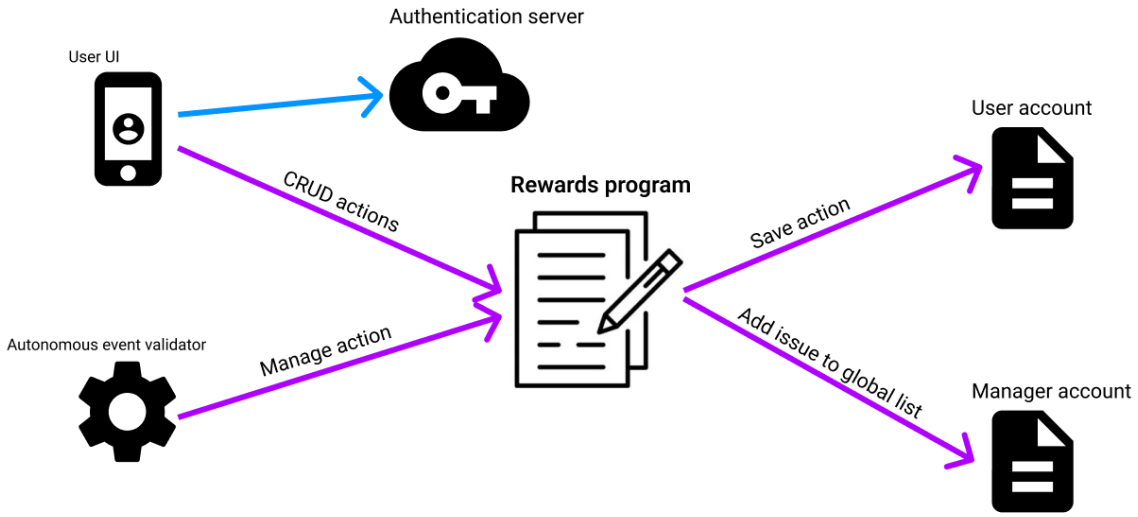

Fig. 4.7 Overview of our solution's architecture.

The final UI specification will be described at section 4.4, a part of the rest of the components, as it is the entry point of a new user to our system and should have a good UX.

4.4.1 Solana program

Firstly, we will expose the Solana program [35], the core of our solution. The Solana program is developed in Rust and has two main functions:

- Manage the actions uploaded by users and store them.
- Transfer a reward in tokens when an action is accepted.

To do so we had to build multiple processors (similar to traditional API endpoints) in our Solana program, to route an incoming transaction to the desired processor we use the Rust match command in the transaction payload. All the incoming transactions' payload are byte arrays serialized following Borsh specification so we must implement the data models of our program and some of their serialization mechanisms.

4.4.2 Solana accounts

Secondly, and related to the Solana program, we will use the data models created in the program to format the stored data of our users' accounts. Each user will save its actions on its own account, this account is created using the Flutter application. The holder of the account will be the user but the owner will be our Solana program so our program can modify the data of the account, but

the user can decide to delete the account without needing the intervention of the service provider.

4.4.3 Autonomous event validator

Thirdly, and related to the reward process, we will discuss the autonomous event validator. This component is designed to prevent non-honest users from upload non-valid actions and get a reward, non-valid actions can be sending several times an action that in the past got a reward, sending random transactions to get rewards, etc.

To accomplish this goal of preventing reward non-valid actions, there can be some strategies, but we had resumed in two main strategies:

- Human validator: In this strategy we should develop a second UI and get some human validators that could use this new UI to accept or decline incoming actions.

- Autonomous validator: In this strategy we should develop an algorithm capable of distinguishing valid actions from non-valid actions. To do so, and based on the success of machine learning techniques in fields such as recognizing spam emails or fraudulent mobile calls, we could develop a machine learning-based agent to perform this validation in an autonomous way.

As the goals of this project were to develop a blockchain based application following the circular economy guidelines, we had only developed a basic bot-like algorithm to accept or decline user created actions, but we have made a proposal of some considerations to develop the machine learning-based autonomous validator that follows the SDG guidelines:

- No bias: Some machine learning studies have realized that if the data used to train the model is biased, these biases will reach the final model and its performance will not be the desired one, lowering the fairness of the overall system.

- Open source: Following the working line of this project, as each component is going to be open-source, the autonomous validator should also be open-source. This approach enhances the transparency of the project and opens several working lines to upgrade this solution.

4.4.4 Authenticator server

In order to permit users to migrate from a device to another device, we use an authentication server.

This authentication server is used to securely store users' wallets, to do so, we encrypt wallets using AES GCM and the key will be encrypted using a key derivation function from user credentials.

4.5 User Interface (UI)

Once the system behavior and all the non-user related components are described, we will define the UI specifications.

The UI of this project must be an intuitive and user-friendly UI with an improved UX that hides all the blockchain complexity under a reactive interface.

From the blockchain related logic point of view, we use the solana_dart package to hide part of the blockchain complexity. This package is in development by Cryptoplease, even if we use this package to face a part oof the blockchain complexity we also had to develop our services and serializers as the ones provided by solana_dart are not stable and some features don't work as expected.

From the application architectural point of view, we chose to use a combination of Redux and MVVM paradigms to provide our application a good responsive experience.

Last but not least, we have designed a set of mockups, see Fig. 4.8, to get a friendly UI with a material design approach.

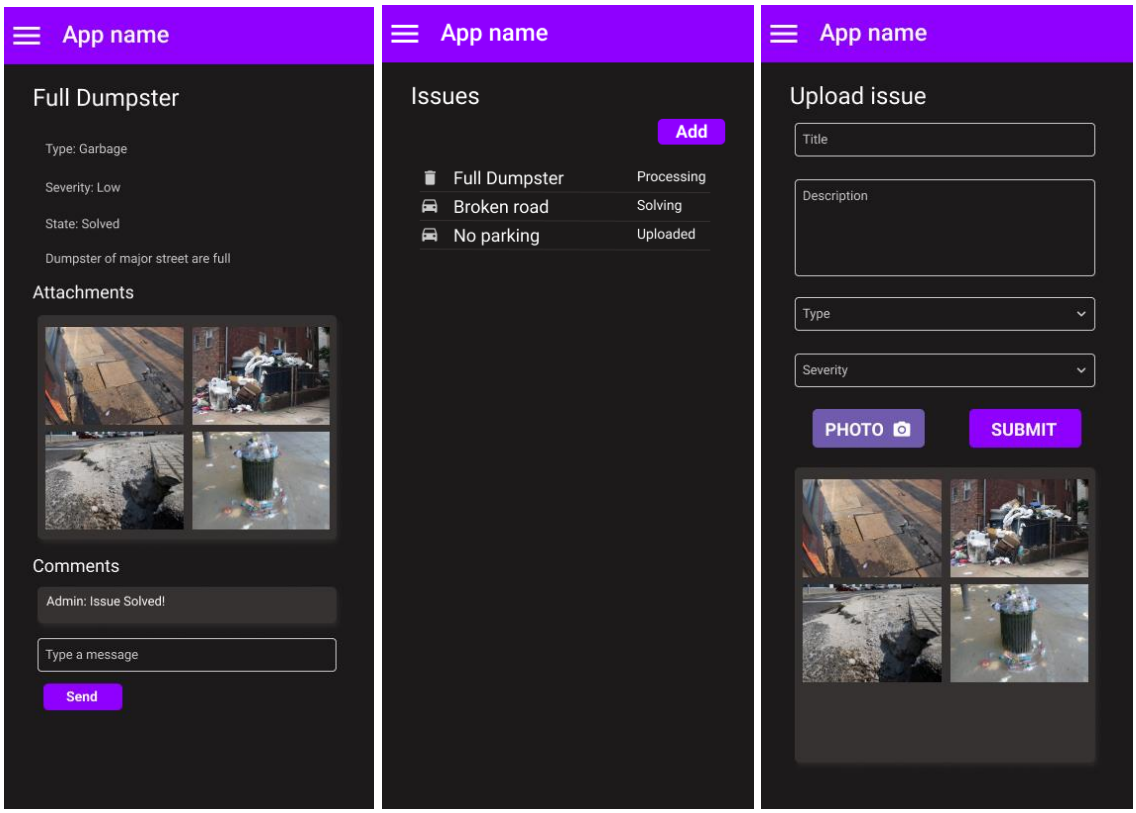

Fig. 4.8 Final application mockups.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

During the last twenty years, there has been big development in web services and applications, since first applications where the web master was the only one that could produce content, passing through current social web applications where everyone can upload its own content to these huge platforms and this in coming new generation of application where users doesn't only can produce its own content, but also users own the content produced by them. This new generation comes with several technical improvements and revolutions as blockchain technologies and artificial intelligence. This new generation comes with a lot of opportunities to build a better world.

Another improvement of last years, but not in technical aspects but in social aspects, are the guidelines and the direction of several institutions, companies and individuals to build a more democratic and fairer world, as SDG proposals.

In the development of this project, we have learned a lot about these new technologies analyzed in this thesis, but also about how to use these technologies to bring their benefits to everybody, and as a result, we have built an application based on these new technologies that provides an efficient way to reward good citizens for their actions.

5.1. Conclusions

After all the efforts put into this project, we can conclude that blockchain doesn't only provide technical and economic benefits as reducing the OPEX of a web service and providing high availability to projects built on top of a blockchain with really low compared costs. Blockchain technologies also come with a lot of new features that allow service providers to build applications more aligned with social-economic guidelines as SDG.

Blockchain technologies also have their drawbacks. The main drawbacks of these technologies come for the part of being new technologies:

- There are not many official courses to develop using these new technologies.

- Although the number of tools, as frameworks and libraries, to build applications using these new technologies are growing, there is a lack of some tools as stable serialization libraries.

- Even if the basic architecture of an application based in blockchain is quite similar to the architecture of a traditional application. It is difficult to adapt an existing solution to the blockchain environment, it can be easier or harder depending on the chosen blockchain technology.

About the development using a new generation of multi-platform development frameworks, there has been a huge evolution from previous frameworks. Using Flutter, we realized that having good mockups and a technology to easily build a UI based on these mockups, the development of a mobile application can be incredibly speeded up relative to previous framework or native development.

The overall conclusion can be achieved by analyzing the state of our initial goals:

- Blockchain analysis: We have been able to analyze EVM-based blockchains with new generation blockchain such as Solana and get some comparison results between them.

- DApp development: We have developed two small applications based in the blockchain technology and a proof of concept of our good citizen application following SDG guidelines with a material design user interface and an improved user experience.

- Teaching documentation: We have documented several key points of different blockchain technologies, we have also compared these technologies and proposed several architecture alternatives starting from simpler to more complex architectures.

5.2. Future work

As we have exposed in section 4.3.3, this project can be upgraded by using a machine learning-based autonomous validator instead of a bot-like autonomous validator.

Another detected need of development is in building libraries to ease the management of the blockchain complexity in frontend application, such as dialoging with blockchain APIs and serializing. In our development with Dart and Solana we detected that there are some Dart and JavaScript Borsh serializer libraries, but there is not a really complete and stable Borsh serializer library, the majority of them have problems serializing complex data structures as object arrays or objects inside objects.

5.3. Environmental considerations

As this thesis's main objective was to align new technologies, such as blockchain, with the circular economy, we have considered environmental effects of every choice we had made.

Firstly, about power consumption, we have compared two blockchain alternatives and, once compared, we have realized that Ethereum blockchain has a huge power consumption due to the usage of PoW as consensus mechanism, around 2400 times higher that PoS based blockchains [36].

Secondly, about electronic devices waste, Ethereum validators usually buy a large number of GPUs and ASIC devices and use them 24h per day 7 days a week at a high percentage of their capabilities, reducing considerably the useful life of these devices.

On the other hand, Solana validators use significantly less power consumption than Ethereum ones, but the hardware requirements to run a full Solana validator are high [37], the good point is that Solana Validators does not run at so high loads than the Ethereum ones, having a poor impact in the useful life of the equipment.

BIBLIOGRAPHY

[1] Sun, J., Yan, J., Zhang, K.Z.K., “Blockchain-based sharing services: What blockchain technology can contribute to smart cities.” Financ Innov 2, 26, 2016.
[2] Back, A., “Hashcash” 1997.
[3] Back, A., “Hashcash - A Denial of Service Counter-Measure” 2002.
[4] Nakamoto, S., “Bitcoin: A Peer-to-Peer Electronic Cash System” 2008.
[5] Kaliski, B., “Password-Based Cryptography Specification Version 2.0” 2000.
[6] Percival, C., "Stronger Key Derivation Via Sequential Memory-Hard Functions", 2009.
[7] Buterin, V., “Ethereum Whitepaper”, 2013
[8] Dillak, R., Suchendra, D., & Hendriyanto, R., Agung, A., “Proof of work: Energy inefficiency and profitability. Journal of Theoretical and Applied Information Technology”, 97, 2019
[9] Lihu, A., Du, J., Barjaktarevic, I., Gerzanics, P., Harvilla, M., “A Proof of Useful Work for Artificial Intelligence on the Blockchain”, 2020
[10] Baldominos, A:, Saez, Y., “Coin.AI: A Proof-of-Useful-Work Scheme for Blockchain-Based Distributed Deep Learning”, 2019
[11] Saleh, F., “Blockchain without Waste: Proof-of-Stake”, The Review of Financial Studies, Volume 34, Issue 3, March 2021, Pages 1156–1190
[12] Bentov I., Gabizon A., Mizrahi A. “Cryptocurrencies Without Proof of Work”, Lecture Notes in Computer Science, vol. 9604, 2016
[13] Kiayias A., Russell A., David B., Oliynykov R., “Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol”, Lecture Notes in Computer Science, vol 10401, 2017
[14] Intel Corporation, “PoET 1.0 Specification”, [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html [Accessed 20 01 2022]
[15] Solana, “Solana: A new architecture for a high performance blockchain v0.8.13“, [Online]. Available: https://solana.com/solana-whitepaper.pdf [Accessed 25 01 2022]
[16] Fisch, B., “PoReps: Proofs of Space on Useful Data” IACR Cryptol. ePrint Arch. 2018, p.678.
[17] TradeBlock, “Bitcoin Historical data“, [Online]. Available: https://tradeblock.com/bitcoin/historical/1h-f-tsize_per_avg-01101 [Accessed 26 01 2022]
[18] Chaindebrief, “Introduction to Solana”, [Online]. Available: https://chaindebrief.com/introduction-to-solana/ [Accessed 26 01 2022]
[19] Buterin, V., “Ethereum Whitepaper”, [Online]. Available: https://ethereum.org/en/whitepaper/ [Accessed 26 01 2022]
[20] Assia, Y., Buterin, V., Hakim, L., Rosenfeld, M., Lev, R., “The Colored Coins Protocol”, [Online]. Available: https://github.com/Colored-Coins/Colored-Coins-Protocol-Specification [Accessed 26 01 2022]

[21] Ethereum Organization, “Community guides and resources”, [Online]. Available: https://ethereum.org/en/learn/ [Accessed 26 01 2022]
[22] Ethereum Organization, “Ethereum Virtual Machine (EVM)”, [Online]. Available: https://ethereum.org/en/developers/docs/evm/ [Accessed 26 01 2022]

[23] Ethereum Organization, “Nodes and Clients”, [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/ [Accessed 27 01 2022]
[24] Solana Foundation, “solana_program”, [Online]. Available: https://docs.rs/solana-program/latest/solana_program/ [Accessed 27 01 2022]
[25] Solana Foundation, “Accounts”, [Online]. Available: https://docs.solana.com/es/developing/programming-model/accounts [Accessed 27 01 2022]
[26] Solana Foundation, “Runtime” https://docs.solana.com/es/developing/programming-model/runtime [Accessed 28 01 2022]
[27] United Nations, “THE 17 GOALS”, [Online]. Available: https://sdgs.un.org/goals [Accessed 30 01 2022]
[28] United Nations, “Sustainable Development Goals”, [Online]. Available: https://www.un.org/sustainabledevelopment/ [Accessed 02 02 2022]
[29] The kernel development community, “BPF Documentation”, [Online]. Available: https://www.kernel.org/doc/html/latest/bpf/index.html [Accessed 02 02 2022]
[30] NEAR, “Borsh, binary serializer for security-critical projects”, [Online]. Available: https://borsh.io/ [Accessed 03 02 2022]
[31] Solana Foundation, “Accepted Design Proposals”, [Online]. Available: https://docs.solana.com/proposals/accepted-design-proposals [Accessed 04 02 2022]
[32] Google, “Dart Overview”, [Online]. Available: https://dart.dev/overview [Accessed 05 02 2022]
[33] González Campos, G., “solana_getting_started”, [Online]. Available: https://github.com/gabrielgonzalezcampos/solana_getting_started [Accessed 06 02 2022]
[34] González Campos, G., “solana_chat”, [Online]. Available: https://github.com/gabrielgonzalezcampos/solana_chat [Accessed 06 02 2022]
[35] González Campos, G., “solana_issues_manager_program”, [Online]. Available: https://github.com/gabrielgonzalezcampos/solana_issues_manager_program [Accessed 06 02 2022]
[36] Ethereum Organization, “Ethereum energy consumption”, [Online]. Available: https://ethereum.org/en/energy-consumption/ [Accessed 05 02 2022]

[37] Solana Foundation, “Validator Requirements”, [Online]. Available: https://docs.solana.com/es/running-validator/validator-reqs [Accessed 05 02 2022]

GLOSSARY

UMTS	Universal Mobile Telecommunication System
UN-GA	United Nations General Assembly
SDGs	Sustainable Development Goals
NFT	Non-Fungible Token
APP	Application
DApp	Decentralized Application
UI	User Interface
UX	User Experience
PoW	Proof of Work
DoS	Denial of Service
RPoW	Reusable Proof of Work
PoS	Proof of Stake
DPoS	Delegated Proof of Stake
BFT	Byzantine fault tolerance
PoET	Proof of Elapsed Time
ERP	Enterprise Resource Planning
VDF	Verifiable Delay Functions
PoSpace	Proof of Space
PoRep	Proof of Replication
CAPEX	Capital Expenditure
OPEX	Operating Expenses
CRUD	Create, Read, Update, Delete
DoS	Denial of Service
EVM	Ethereum Virtual Machine
LIFO	Last-in-first-out
ASIC	Application-Specific Integrated Circuit
DeFi	Decentralized Finances
OS	Operating System
MVC	Model-View-Controller
SPA	Single-Page Applications
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SCSS	Syntactically Awesome Stylesheets Syntax
SDK	Software Development Kit
VM	Virtual Machine
MVVM	Model-View-Model View
FPS	Frames per Second
IaaS	Infrastructure as a Service
aaS	As a Service
AAA	Authorization, Authentication and Accounting
CPI	Cross-Program Invocation