POLITECNICO
MILANO 1863

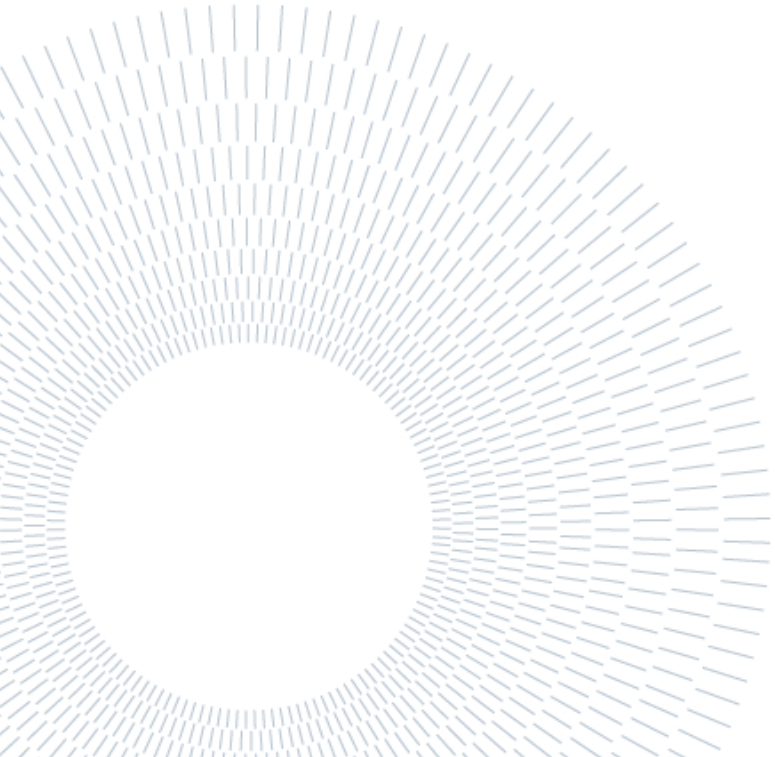
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DecentMarkt – decentralized marketplace for second-hand goods

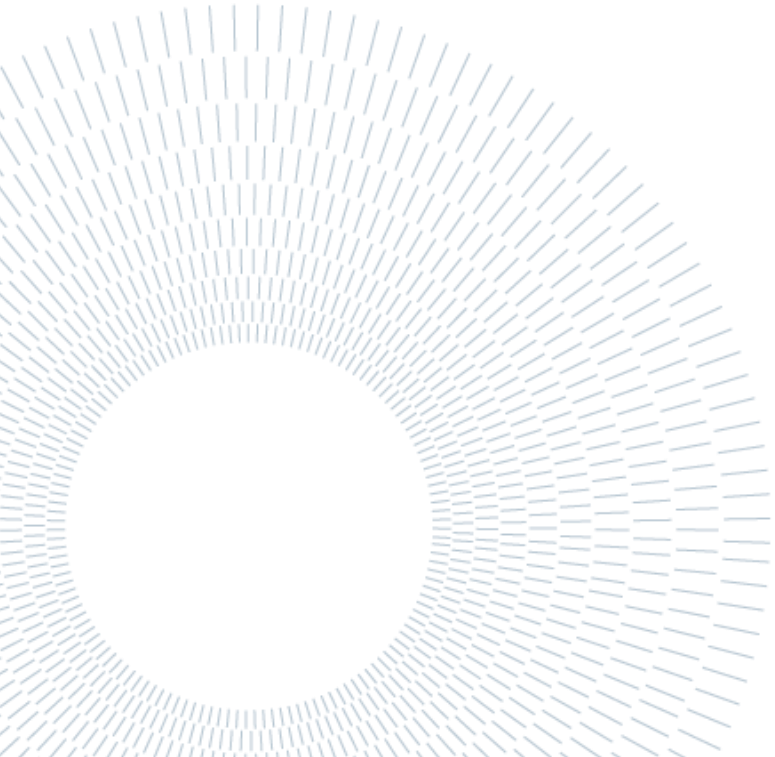
TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE ENGINEERING
INGEGNERIA INFORMATICA

Author: **David Drvar**

Student ID: 222890
Advisor: Francesco Bruschi
Academic Year: 2024-25



To my parents Marko and Dejana



Abstract

Online marketplaces from big tech companies such as Amazon, Facebook and eBay have significantly changed the way people trade and purchase goods and have become a major part in global e-commerce. However, these centralized marketplaces bring many issues - centralized authority, censorship, misuse of customer data, strict regulations, etc. The goal of this thesis is to explore how blockchain technology can solve these issues by offering a decentralized and transparent way of direct trading, without the need for centralized authority.

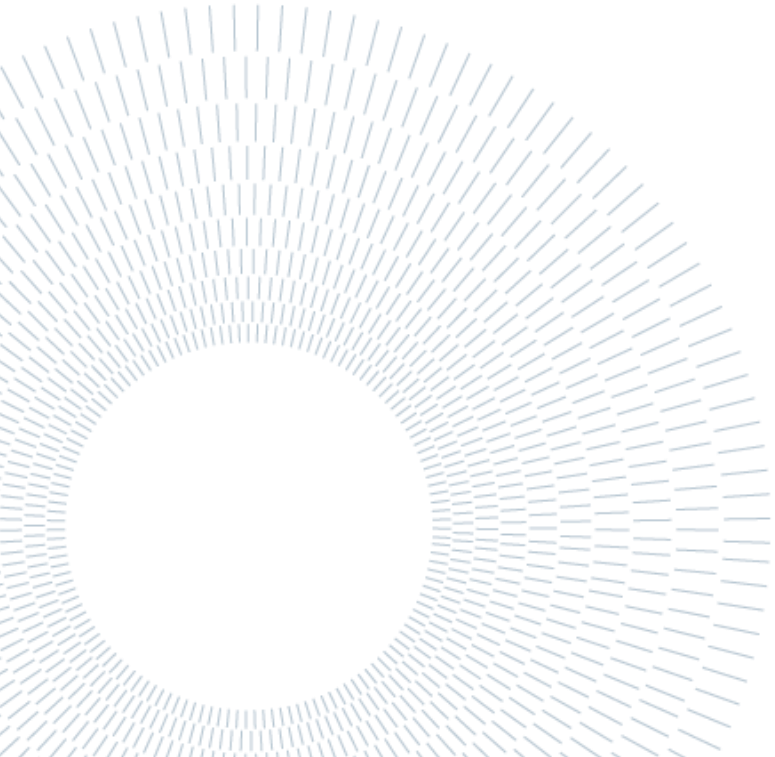
Some solutions have already been developed, and research has been conducted to assess their benefits, challenges they faced, and reasons behind the failure of some of them. In particular, OpenBazaar, Particl Marketplace and Origin Protocol have been analyzed. Within this context, the DecentMarkt project is implemented and presented as a decentralized marketplace for trading second-hand goods. Inspired by OpenBazaar, the most well-known decentralized marketplace launched in 2016 and shut down in 2021, DecentMarkt aims to address the issues of scalability and limitations of usability that its predecessors have faced.

DecentMarkt infrastructure combines centralized and decentralized components to balance transparency with user experience while ensuring scalability and cost effectiveness. To ensure fair trade the escrow system has been implemented. It temporarily holds funds until both the buyer and seller confirm successful order completion. In case of a dispute, a chosen optional moderator intermediates and decides on the fair distribution of the funds, while collecting a fee for their service. Additionally, the reputation system enhances trust within the marketplace, along with the support of ERC-20 tokens in the form of stablecoin USDC which helps users mitigate volatility risks while trading. Firebase, a centralized database, has been utilized to store sensitive user data while IPFS and Pinata services store images to optimize smart contract storage costs. Lastly, after extensive cost and throughput analysis, Polygon blockchain network has been chosen as a layer 2 solution for its low gas fees and quick transaction confirmation.

Through the DecentMarkt project, this thesis demonstrates the potential of blockchain in the scope of e-commerce, offering a viable alternative to traditional marketplaces. DecentMarkt serves both as a proof of concept and a

foundation for further research in the development of decentralized e-commerce.

Keywords: blockchain, smart contracts, decentralized marketplace, dApps, web3 technology



Abstract in lingua italiana

I marketplace online delle grandi aziende tecnologiche, come Amazon, Facebook ed eBay, hanno trasformato radicalmente il modo in cui le persone commerciano e acquistano beni, diventando una componente essenziale dell'e-commerce globale. Tuttavia, questi marketplace centralizzati presentano numerose problematiche, tra cui l'autorità centralizzata, la censura, l'uso improprio dei dati dei clienti e normative stringenti. L'obiettivo di questa tesi è esplorare come la tecnologia blockchain possa risolvere queste criticità offrendo un modello di scambio decentralizzato e trasparente, eliminando la necessità di un'autorità centrale.

Alcune soluzioni sono già state sviluppate e diverse ricerche sono state condotte per valutarne i benefici, le sfide affrontate e le ragioni che hanno portato al fallimento di alcune di esse. In particolare, sono stati analizzati OpenBazaar, Particl Marketplace e Origin Protocol. In questo contesto, è stato sviluppato e presentato il progetto DecentMarkt, un marketplace decentralizzato per il commercio di beni di seconda mano. Ispirato a OpenBazaar, il più noto marketplace decentralizzato lanciato nel 2016 e chiuso nel 2021, DecentMarkt si propone di affrontare i problemi di scalabilità e le limitazioni di usabilità riscontrate dai suoi predecessori.

L'infrastruttura di DecentMarkt combina componenti centralizzate e decentralizzate per bilanciare trasparenza ed esperienza utente, garantendo al contempo scalabilità ed efficienza dei costi. Per assicurare scambi equi, è stato implementato un sistema di escrow che trattiene temporaneamente i fondi fino alla conferma della corretta esecuzione dell'ordine da parte dell'acquirente e del venditore. In caso di controversia, un moderatore opzionale scelto dagli utenti interviene come intermediario e decide una distribuzione equa dei fondi, trattenendo una commissione per il servizio. Inoltre, un sistema di reputazione rafforza la fiducia all'interno del marketplace, mentre il supporto ai token ERC-20 sotto forma di stablecoin USDC aiuta gli utenti a mitigare i rischi di volatilità durante le transazioni. Per ottimizzare i costi di archiviazione degli smart contract, i dati sensibili degli utenti vengono archiviati su Firebase, un database centralizzato, mentre le immagini vengono memorizzate tramite i servizi IPFS e Pinata. Infine, dopo

un'analisi approfondita dei costi e delle prestazioni, è stata scelta la rete blockchain Polygon come soluzione di livello 2 per le sue basse commissioni di transazione e i tempi di conferma rapidi.

Attraverso il progetto DecentMarkt, questa tesi dimostra il potenziale della blockchain nel settore dell'e-commerce, offrendo un'alternativa valida ai marketplace tradizionali. DecentMarkt rappresenta sia una prova di concetto sia una base per ulteriori ricerche nello sviluppo del commercio elettronico decentralizzato.

Parole chiave: blockchain, smart contracts, marketplace decentralizzato, dApps, tecnologia web3

Contents

Introduction

Online marketplaces have revolutionized the way in which people buy and sell goods. Tech companies such as Amazon and Facebook developed marketplaces which became essential in global e-commerce and are used daily around the globe. However, these centralized platforms face many issues such as lack of transparency, centralized governance, misuse of user data, and many others. With advancements in blockchain technology, many projects have been developed to provide a more transparent, alternative way of trading, without the need of a centralized intermediary while still providing security, and wide accessibility.

The objective of this thesis is to create a decentralized marketplace platform where users from around the world can buy and sell goods by relying on blockchain to offer fast and secure order processing without centralized authority. The project, called DecentMarkt, utilizes web3 technologies to provide a decentralized, transparent and censorship-free marketplace running on blockchain. It enables users to list, buy and sell their goods on the platform in a secure way with moderators which intermediate the order making sure that the funds are released only when all sides of the trade are satisfied. The platform empowers users to trade while still retaining control over their data and its processing, all while keeping their identity pseudo anonymous.

The project addresses the challenges with the traditional online marketplaces and explores solutions to overcoming not only the technical limits of blockchain, such as lower scalability compared to traditional solutions, but also user adoption problems such as people's reservations about blockchain and its currency volatility. To combat the latter problem, DecentMarkt supports USDC stablecoin in the form of ERC-20 token. This stablecoin USDC is pegged to corresponding fiat currency (USD in this case) and can always be redeemed for it, enabling similar experience to the traditional platforms. Furthermore, the project explores solutions for storing sensitive user data, scalability and possible future upgradeability of the platform.

Initially envisioned as a marketplace only for selling second-hand clothes inspired by the application Vinted, the project expanded to all categories. It provides a glimpse into the experience of using blockchain technology to transform e-commerce, exploring different caveats of the technology and its limitations, which will be discussed. Throughout the design and development

process, it was important to take into consideration that decentralization is a spectrum, and that the most appropriate solution would integrate the benefits of both ends of it.

1 Blockchain technology

Blockchain presents a data structure of connected blocks each of which hold data. It is designed to be a secure digital ledger that is distributed across the network of computers running it with no central authority. The first blockchain was drafted in 2008 when the manifest Bitcoin: A Peer-to-Peer Electronic Cash System was published by Satoshi Nakamoto, an individual or group that kept their identity anonymous. In the published white paper, Satoshi Nakamoto elaborated an algorithm for creation of a decentralized digital ledger called bitcoin. The entity behind the white paper created the first block on January 3rd, 2009, marking the beginning of a new internet era called web3.

The following quote from the white paper presents the main idea and motivation behind the blockchain technology, emphasizing trading backed by mathematical proofs instead of centralized authorities:

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party [1].

The proposed blockchain design had many characteristics that will be discussed in this chapter and that made it a disruptive technology aimed at revolutionizing the way people trade assets without the need for a centralized financial intermediary:

- **Immutability** - information is held in a block which is linked to the previous block forming a chain. As a result, changing any data in any block requires altering all its subsequent blocks, which would make tampering evident thanks to cryptographic proofs. Additionally, decentralized consensus protocols make it computationally infeasible to invalidate all subsequent blocks and append fraudulent ones.
- **Solution to the double-spending problem** - double spending means spending the same asset twice, and as cryptocurrency is a digital asset it could be easily misused. Blockchain solved this problem by allowing only one user to create a new block at a time. When a block is generated and broadcasted, each node verifies all transactions with their local ledger and accepts the block if it is valid. Once a block is accepted, it

becomes extremely hard to change information, hence one coin can only be spent once.

- **Decentralization** - there is no central authority among participants in the network that governs it. It is important here to distinguish between the terms distributed and decentralized. Distributed means that there can be a central authority, but information processing is spread across multiple nodes in the network, whereas decentralization implies that no single entity has control, and any decision-making is shared across all participants.
- **Transparency** - the basis of blockchain relies on transparency and trustlessness. As blockchain is a public ledger, all information kept in it is public and easily verifiable by anyone, ensuring its integrity. Additionally, the underlying code powering the network is open-sourced.

1.1. Blockchain fundamentals

Before elaborating each characteristic that made blockchain a disruptive technology, it is important to explain the fundamental principles of the technology. Different blockchains use different ways to generate a new block and verify it, but in this chapter all principles will be explained relating to Bitcoin since it presents the first widespread blockchain network which is still the most popular and widely used today. The differences between blockchains will be pointed out in the following chapters.

The key elements of any blockchain are blocks and transactions. Transactions, as in any traditional system, present a way to document the movement of assets, from sender to receiver. By collecting these transactions, we can create a new block holding all this data and append it to our chain. Each block is timestamped and linked to the previous one creating a blockchain, represented in Figure 1.1.

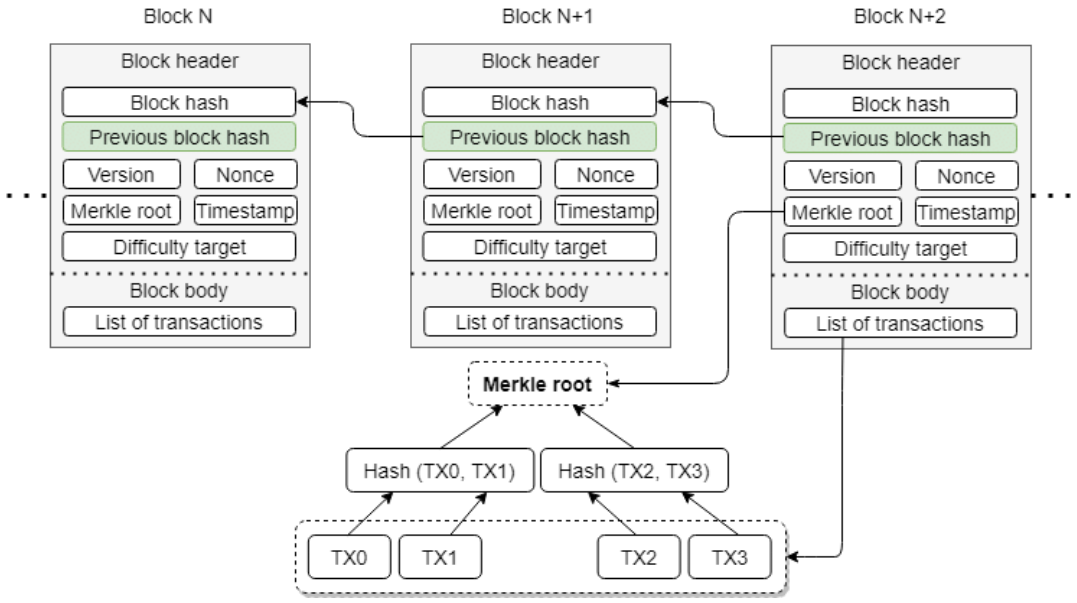


Figure 1.1: Blockchain representation

The way in which blocks are linked is through hash. Each block contains a hash of the block preceding it. There are numerous ways to generate a hash, but in bitcoin's example, a hash is generated by combining data from the header and running it through hash function SHA-256 twice. The reason the hash function is executed twice is to reduce the risk of exploitability. Hash function is a simple, yet essential piece for blockchain systems to function properly. It has numerous characteristics that make it ideal for usage in blockchain context:

- Deterministic - for the same input, the function produces the same output. This property ensures easy and fast verification of data.
- Preimage resistance - given an output, it is computationally impossible to reverse engineer what is the input to it. It is vital to ensure the security of data by not being able to determine their original value based on the output.
- Collision resistance - It is impossible to find two different inputs that produce the same output. This property prevents attackers from substituting original data with fake one that would produce the same hash.
- Small input changes result in completely different output - It is not possible to predict how close one is to the solution as the output appears to be completely random
- Fixed output size no matter the size of the input
- Fast computation

The way block hash is computed and linked with the previous block varies between the blockchains, but it is calculated by hashing the block header. This has multiple purposes, such as linking the blocks, ensuring data integrity and taking part in consensus mechanisms, which will be discussed in the next chapter.

Block header carries information about the current block and its metadata. It contains the Merkle root which is a summarized hash of all transactions in the block. Merkle root is easy to compute and any change in any of the transactions in the block would change the Merkle root and violate data integrity. Other data included in the header are timestamp, version of the consensus protocol, previous block hash, difficulty target, and nonce. Two last data fields play a vital role in bitcoin's consensus protocol.

After the block is created, i.e. mined, it is broadcasted across all nodes in the network. The reason blockchain is called a distributed digital ledger is because every node in the network contains a full or partial copy of all data. For a node accepting a new block, it is easy to verify if the block is valid by taking the incoming input (block header), hashing it and comparing the output with the proposed one by a new block. If outputs match and a set difficulty target is achieved, the local ledger copy is updated with a new block.

1.2. Consensus protocols

After the short introduction to the simplified basics of blockchain technology, it is important to explain the mechanisms which enable collective decision-making on such a large, distributed scale where actors cannot be trusted. In blockchain context, these protocols decide which block is going to be appended to the ledger, all while ensuring that it is secure, and transparent, protecting the network from malicious actors and attacks. There are many kinds of consensus protocols, but in this chapter the two main ones will be discussed - proof of work (PoW) used by Bitcoin, and proof of stake (PoS), used by the second most popular blockchain called Ethereum which is going to be discussed in the next chapter.

It is important to note that the problem of collective decision-making did not come with the discovery and rise of blockchain technology, but has troubled mathematicians, economists and computer scientists for centuries. The challenge of it comes with the fact that in a large collective with independent actors some of them can be faulty or act malicious consciously, nevertheless all nodes must agree on a single decision despite possible misinformation. The now famous problem is called the Byzantine Generals Problem and was formally introduced in 1982 by works of Leslie Lamport, Robert Shostak and Marshall Pease [2].

Researchers developed various algorithms for different use cases, such as the Paxos algorithm published in 1989 by Leslie Lamport that was tackling the problem of reaching agreement in distributed databases where some delay or failure might be introduced. Another protocol, Practical Byzantine Fault Tolerance protocol (PBFT) developed in late 1990s tried to improve reliability of systems that require high fault tolerance. All these previous works helped Satoshi Nakamoto to further develop these protocols by using cryptographic proofs and rewards to make Bitcoin's powerful consensus protocol that has never been broken since its first block was mined in 2009.

1.2.1. Proof of Work

Bitcoin uses proof of work protocol, abbreviated PoW. Although it was first successfully implemented by Satoshi Nakamoto, the protocol was first conceptualized by Cynthia Dwork and Moni Naor in 1993 in their paper called Pricing via Processing and Combatting Junk Mail. In their paper the authors introduced the idea of discouraging spam emails by requiring each sender to submit a proof of a small computational task (proof of work) which would be easy to perform for regular users, but quite costly for spammers that do it on a large scale [3]. Satoshi Nakamoto took core components of the protocol and applied it to blockchain to secure it and ensure its decentralized nature.

The fundamental principle of PoW is block mining. The process is associated with the metaphor of miners who perform hard work to extract precious materials. In the same way, Bitcoin miners are required to compute complex tasks in order to mine a new block, show proof of their work, and receive an award in return in the form of cryptocurrency.

The block hash plays a vital role in this protocol. The input to block hash is as previously mentioned block data and nonce. While block data is immutable, nonce is a number that can be adjusted by miners to solve the puzzle. In Bitcoin, the puzzle is to have a block hash start with the number of zeros set by the difficulty target. To solve the puzzle, miners need to keep changing the nonce until they find the right one that results in the hash output to be in a required format, thus spending computational work and receiving a prize in return.

Although it is easy to compute a single hash, the difficulty increases exponentially when it has to start with a certain number of zeros. The more zeros it has to start with, the computational power needed to find it increases. In the Bitcoin network, the block must be mined every 10 minutes, thus the difficulty target which presents number of zeros the hash has to start with is adjusted accordingly every 2016 blocks. If there are more miners in the network, there is more computational power, and with that the difficulty

target is higher, thus the miners must spend more power to compute the hash. In the periods when miners are scarce, the difficulty target will automatically be lowered so that the miners will be able to compute the hash and create a block in the 10-minute mark.

A miner who first successfully finds the right hash, creates a block and broadcasts it to the whole network. Other nodes are able to easily verify the block by taking input block data, newfound nonce and checking if the output hash matches the broadcasted one. If a new block is valid, it is appended to the chain and the miner receives a reward as an incentive for spent computational power.

A reward consists of two parts - block reward and transaction fees. Block reward is a set reward for mining the block and it is halved every 210k blocks in an event called Bitcoin halving. The second reward comes from transaction fees which are paid by users to incentivize miners to include their transaction in the next block. Since every bitcoin block can contain only 1MB of data which can hold up to 4000 transactions, miners pick the ones with the highest transaction fees first to increase their potential reward. Additionally, the initial block reward started with 50 BTC, and as of November 2024 it is 3.125 BTC. Consequently, as block reward decreases, transaction fees are expected to become higher, making up the bigger part of the total block reward.

Benefits of PoW are numerous. Firstly, the protocol offers high security that withstands the test of time since 2009 and numerous unsuccessful attacks to overtake the network. Because of the high mining costs, an attacker would need 51% of the total mining power to overtake the network, which would be immensely expensive. Additionally, immutability is guaranteed since tampering with one block requires re-mining all the blocks after it and successfully linking them which would be computationally impossible.

On the other hand, PoW does have certain drawbacks. Firstly, the high requirement for computation power leads to high energy consumption. It has been estimated that a single bitcoin transaction has the same carbon footprint as 1M VISA transactions, while bitcoin's annual carbon footprint is similar to the one of Qatar [4]. Furthermore, PoW has a limited throughput compared to traditional systems as bitcoin processes around 7 transactions per second [5].

1.2.2. Proof of Stake

PoW paved the way for various consensus protocols to be developed with motivation to overcome its weaknesses. Proof of stake presents one of the most widely used consensus protocols, initially conceptualized in 2011 by a developer named QuantumMechanic on the BitcoinTalk forum and first implemented in 2012 in Peercoin. The main idea behind it was to address the

growing concern of PoW's environmental impact, as well as its scalability concerns.

While PoW uses computational power as means of voting, PoS uses a staked amount of cryptocurrency of the blockchain it is running on. There can be only one chosen validator that creates a block hash, appends it to the chain and broadcasts it. Other nodes receive a block, verify it and update their local ledger if the block is valid. The more cryptocurrency a user has staked, the higher the chances to be picked as the next validator. The staked amount is locked and cannot be spent. It is used as a guarantee and to deter validators from acting maliciously. In case of a user not mining a block when chosen as a validator, the staked amount is lost and cannot be recovered.

PoS provides a more ecological consensus protocol since only one validator is chosen and creates a block. On the contrary, in PoW all miners are trying to create a block and are solving the same problem using computational power that brings reward only to one of them, while the rest of the power is essentially lost and disregarded. Additionally, PoS differs in rewards it gives to a validator. Compared to PoW's rewards consisting of block reward and transaction fees, PoS gives only transaction fees to the chosen validator, while block reward is non-existent.

PoS comes with its own drawbacks too. It is often criticized for not being truly democratic, because statistically the higher the stake a user has, the higher the chances are for them to get picked as the next validator and claim the reward. To overcome it, numerous approaches have been proposed and implemented. For example, Ethereum 2.0, which initially used PoW, before it switched to PoS to lower its energy impact, uses random number generators to assign a validator unpredictably [6]. Additionally, Ethereum's implementation of PoS balances rewards with the staked amount in a way that it incentivizes smaller stakers to participate. In that way, those stakers can earn fairer rewards when compared to the more significant stakers.

1.3. Ethereum

Bitcoin is often referred to as the first generation of blockchains. It introduced the concept of decentralized digital currency and enabled peer-to-peer trading without the need for an intermediary. It still is the most widely used blockchain and it has a widespread impact on the whole crypto industry. According to Marion Laboure, senior economist and market strategist at Deutsche Bank, it has a potential to become "digital gold of the 21st century" [7].

The next generation of blockchains, generation 2, builds upon the capabilities of Bitcoin and tries to integrate more functionalities to the network apart from just peer-to-peer transfers and transaction scripts. The second generation of blockchain tries to address the huge energy demand coming from the PoW, by switching to alternative protocols such as PoS and others. By doing that, they are also able to improve scalability issues and low transaction throughput that come with it. Equally important, this second blockchain generation provides programmable blockchains which can run arbitrary code making it Turing complete. Achieving Turing completeness means that a system can perform any computation that can be defined by an algorithm given enough resources like time and memory [8]. It opens a whole new world to use cases of blockchains beyond mere cryptocurrency, such as DeFi, NFTs and dApps.

Among the second generation of blockchains, Ethereum is considered as the key figure. According to Marion Laboure, if Bitcoin is sometimes referred to as "digital gold", Ethereum would then be "digital silver" [7]. Conceptualized and developed by Vitalik Buterin in 2015, Ethereum at the beginning used PoW but planned to switch to PoS to achieve scalability and sustainability. It successfully switched to Ethereum 2.0 on September 15, 2022, in the event called the Merge, when it deprecated its PoW chain and reduced energy consumption by astonishing 99.95% [9].

1.3.1. Smart contracts

What made Ethereum stand out was the fact that it was able to build upon the basis of peer-to-peer trading and enable execution of arbitrary code on blockchain thus proposing the concept of the first "world computer" [10], so called Ethereum Virtual Machine (EVM). EVM is composed of smart contracts, which are executable code snippets that are written in a special programming language and are deployed on blockchain. They are interesting for numerous characteristics:

- Immutable - once deployed, their execution in inevitable and code cannot be altered
- Deterministic - they always produce the same output based on the same input which is predictable and verifiable
- Transparent - the code is always open source and publicly visible on blockchain
- Trustless - their execution is independent and not even the smart contract owner can influence it

Smart contract execution is triggered when a transaction interacts with it. It has its own address on blockchain, and users interact with it by sending

transactions with function parameters to its address. Smart contract components are similar to those of any other software. It has its own state variables stored on blockchain within smart contract storage. It has functions which contain business logic and enable interactions with smart contract. Lastly, it can emit events which can notify external off-chain components to start execution, further processing or log result data.

While smart contracts greatly improved capabilities of blockchain technology and enabled a variety of use cases which will be discussed in the following chapter, they still face some challenges. Firstly, running smart contracts does not come for free. Each execution, no matter if it is a basic calculation or a complex one, requires computational effort which is paid by the transaction sender. These costs are called gas fees and are paid in ether (ETH), the native currency of Ethereum. These gas costs are paid to validators, and they incentivize them to include the transaction in the next block. The gas price, i.e. amount of ether a user is willing to pay per computation effort, varies depending on the network congestion. In the peaks of traffic, triggering smart contract execution could be quite expensive depending on how complex the logic is, therefore, optimizing a smart contract is essential to ensure that its execution can be as economical as possible.

Another caveat of smart contracts derives from their immutability. On one side it brings deterministic behavior and trustlessness to the network, however because the code cannot be altered any vulnerability can lead to exploits. In the past, there have been incidents of attacks, the most notable one called “The DAO hack” happened in 2016 when a hacker exploited a vulnerability in smart contract code which allowed them to transfer to their account 3.6 million ETH which was worth around 50 million euros at the time [11]. Consequently, thorough testing is of absolute importance in smart contract development. Any mistake is hardly reversible.

In this chapter only Ethereum blockchain was mentioned as it played a key role in smart contract development. However, it is not the only platform that supports smart contracts. There are other notable ones such as Avalanche, Polygon, Binance Smart Chain and many others that differ in their performance, scalability, fees and level of centralization, tailoring to specific use cases while keeping EVM functionality.

1.3.2. dApps

dApps is a common term to refer to decentralized applications which rely on blockchain technology to provide their users various software services in a decentralized, transparent and secure way. The term originated opposing

traditional centralized applications which are run on centralized servers and where all decisions are made by the central authority governing it.

Decentralized applications are enabled by smart contracts, as their backend logic is completely executed by one or a set of them. Smart contracts are interoperable, thus dApps make use of it to split logic in multiple smart contracts respecting the separation of concern principle. Additionally, dApps are designed to be open sourced from the ground up allowing the community of users to thoroughly audit them and contribute to their further development.

Along with the set of benefits that come with the usage of smart contracts, many dApps want to incentivize users by a tokenized ecosystem. In this ecosystem, dApps make use of tokens, which are digital assets built on top of existing blockchain. Unlike native cryptocurrencies BTC or ETH which were mentioned, these tokens are made by smart contracts using Ethereum Request for Comment (ERC) standards. These standards define a set of rules and behaviors for tokens. The most famous of these standards are ERC-20 and ERC-721 which describe the behavior of fungible and non-fungible tokens, the latter known as NFT. This system allows dApps to create their own tokens for various purposes such as voting on further decisions, using tokens to access services within a dApp or rewarding users.

The way dApps are built is similar to the traditional apps with a few differences. As in traditional apps, decentralized applications provide their services through frontend interfaces using the same technological stack. This frontend layer uses libraries to interact with smart contracts deployed on blockchain which serve as a business logic layer. To be able to interact with dApps, users need to have web3 wallets which manage private keys and enable transaction signing, making the experience more seamless. Some dApps additionally use off-chain components such as IPFS (InterPlanetary File System) which provides decentralized storage, or even centralized databases to improve performance and lower usage costs. It is important to note that while dApps rely on smart contracts for backend logic, the other components built on top of it offer varying degrees of centralization.

The dApp ecosystem is still rapidly developing and so far, various solutions have been developed in a wide range of use cases. Most prominently, dApps have been utilized in finance and created a powerful system of decentralized finance platforms (DeFi) such as Uniswap and Aave, which offer decentralized exchange and even lending and borrowing assets without the need of traditional banks. According to Forbes as of November 2024, DeFi market cap is 106 billion US dollars and continues to grow [12]. DApps have also found use cases in gaming and NFTs by creating play-to-earn ecosystems and virtual worlds such as Decentraland. In enterprise context, dApps have been

employed in supply chain management to secure transparency and traceability of parts. Additionally, with the growing misuse of user's data, social media dApps have been developed as well, providing users with the autonomous control of their data.

2 Related projects overview

With the rise of the second generation of blockchains and their code execution capability, various use cases appeared, among them decentralized marketplaces. They offer alternatives to traditional marketplaces which have become monopolistic companies that have taken complete control over global e-commerce. In such a monopolistic market, the ruling companies such as Amazon, Facebook, eBay and others keep adding more fees, and restrictions to their platforms, while users often have no other option than to succumb to the new rules. Additionally, they misuse users' data and personal information without consent that is sold to third parties to generate purchase profiles [13].

Decentralized marketplaces use blockchain technology to deliver to users greater control over their data, transaction and privacy, without the need of intermediaries. Still under development and wide adoption by users, they offer various benefits such as absolute freedom and unrestricted access to products, without censorship. Additionally, no middleman is required because there is direct peer-to-peer trading with others, thus more money is retained by sellers since there are no third-party costs or commissions. Lastly, all their code is provably secure, open-sourced, audited and publicly visible on blockchain for anybody to review.

Numerous projects have attempted to bring a decentralized way of trading to a wider audience. In this chapter, the most prominent ones will be discussed, with OpenBazaar standing out as the most popular and recognized. Furthermore, platforms such as Particl Marketplace, and Origin Protocol made their contribution to the idea with their own unique approaches which will be explored.

2.1. OpenBazaar

The most successful decentralized marketplace so far was without a doubt OpenBazaar. Launched in 2016 as a result of a hackathon in Toronto, its goal was to disrupt traditional marketplaces by creating a peer-to-peer network for users to trade directly, without relying on a middleman, and avoiding restrictions imposed by governments [14]. Unfortunately, it was shut down in 2021 due to user adoption issues, lack of funding and its failure to find a way to monetize its investments.

From the ground up, the platform was designed to be free, with no commissions, open-sourced and driven by strong community sense between its members. There was no censorship on the platform, so users could trade without any constraints.

The core of the platform was its peer-to-peer architecture. The users had to run a local instance of the OpenBazaar software, which would be a node in the network and would host their own personal store. In such a decentralized approach, there was no single point of failure making it resistant to shutdowns and censorship.

It is important to note that at the beginning, OpenBazaar didn't rely on the blockchain completely as other decentralized apps mentioned. Instead, it integrated blockchain technology solely for payments which used cryptocurrencies such as bitcoin, ethereum, and others. Transactions were processed on-chain through a multi-signature wallet which required signatures as approval from all parties (buyer, seller, and optional moderator) involved so that the funds could be released [14]. This feature enabled a trustless escrow system guaranteeing that no single party has complete control over the funds during the purchase.

As the Ethereum and smart contracts gained popularity, the creators behind OpenBazaar noticed its benefits and transitioned to using smart contracts for every trade that was using Ethereum, or ERC-20 tokens. They introduced in late 2018 the escrow smart contract which replicated the functionality of multi-signature wallets they already had implemented for UTXO-based transactions [15]. This smart contract held funds until the agreement on how the escrowed funds are to be distributed between the buyer and the seller.

This shift to smart contracts also enabled OpenBazaar to support trades using Ethereum and ERC-20 tokens. Additionally, it enabled the platform to introduce its own OpenBazaar token (OBT) to further incentivize participation and promote decentralized governance [16]. Firstly, the OBT token was used as a reward token for contributors and early adopters who actively participated in the platform. Secondly, it allowed users to vote on proposed changes and features of the platform. Furthermore, it was used to sell and purchase ad space on a channel, which was essentially the OpenBazaar version of a store to which users could subscribe to get updates about new listings. The token was released with a fixed supply and was intended to add value to the network.

The platform also used Ricardian contracts which are human-readable contracts defining the terms of the trade and are digitally signed [17]. They are different from smart contracts in that they are not executable, are stored locally and rely on participants to consciously abide by them, but they serve as proof

of the mutually agreed order [17]. In the OpenBazaar context, they contained information such as item description, price, shipping terms and dispute resolution conditions.

The platform featured expandable design through APIs and plugins which third-party developers utilized to create various tools for inventory management, analytics and other needs [18]. Other features of the platform included end-to-end encrypted messaging which allowed members to communicate directly without exposing any information to parties not involved. To store other data such as product listings, and images, OpenBazaar relied on IPFS, a distributed file storage protocol, which complemented the decentralized nature of the platform by removing the need for centralized servers to store the content [18]. The way IPFS works is by breaking data into smaller chunks, assigning them a unique cryptographic hash, which is then distributed across the network.

Unfortunately, OpenBazaar faced several challenges that contributed to its shutdown. Firstly, the peer-to-peer network required users to spin off a local instance of the software to use the platform creating a big barrier of entry, especially for users not familiar with new technologies. With it, came the issues of scalability as the performance of the platform was linked to the availability of the nodes in it. All these conditions made it difficult to attract a broader user base on which the platform relied heavily. Lastly, it struggled to pay maintenance and further development costs and generate sustainable revenue.

Despite its shutdown, OpenBazaar inspired many to create open and censorship-free marketplaces. With advancements of smart contracts and blockchain technology, many dApps came after OpenBazaar that built upon its idea and used it as a blueprint for decentralized e-commerce.

2.2. Particl Marketplace

Another marketplace with decentralization in mind is Particl Marketplace. Launched in 2019 as a part of the broader Particl Ecosystem, it puts the most focus on protecting user privacy. Particl Ecosystem originated from the former contributors to ShadowCash (SDC) cryptocurrency which was designed to enable anonymous transactions using advanced cryptographic techniques such as ring signatures and stealth addresses [19]. Later, the aim became to create a wider privacy-centered ecosystem and Particl Marketplace was born.

In its core, Particl Marketplace relies on Particl blockchain and its native PART cryptocurrency. The blockchain uses PoS consensus mechanism to secure scalability [20] . By relying on its own blockchain and limiting transactions to

its native currency, Particl Marketplace is able to ensure privacy for users throughout the whole platform. It employs advanced techniques such as confidential transactions and ring signatures to obscure any possible traces of the origin of a transaction. Additionally, it provides temper-proof end-to-end encrypted messaging by using decentralized SecureMessaging (SMSG) protocol [20].

When it comes to the trading logic, it differs from OpenBazaar in that Particl implements a two-party escrow system in which there is no need for a moderator. The smart contracts are deployed to Particl blockchain and to this escrow, both seller and buyer commit equal security deposit which is kept there until both parties confirm that the purchase was successful [21]. Failure to reach the agreement between two parties results in the loss of their deposits. In that way, honest behavior is incentivized.

Despite its privacy focused innovative approach, Particl Marketplace faces challenges similar to those of OpenBazaar, the biggest one being user adoption. As it requires proprietary software installed locally to run, it creates a barrier of entry to users. Additionally, limiting trading to solely native cryptocurrency deters the rest of the crypto community to join the platform, limiting its reach. Lastly, being privacy focused, the platform is often associated with trading illicit goods and is under pressure by the regulators.

2.3. Origin Protocol

Different from OpenBazaar and Particl Marketplace that provide a full-fledged decentralized marketplace solution, Origin Protocol provides a framework for creation and development of decentralized marketplaces. It doesn't give a ready-made solution but empowers others to build custom dApps on top of it utilizing its ecosystem.

Originally founded in 2017 by Josh Fraser and Matthew Liu with the idea to simplify development of decentralized marketplaces, Origin Protocol provides a system of smart contracts deployed on Ethereum blockchain that serves as its secure infrastructure [22]. These smart contracts handle the very logic of every marketplace, such as escrow, payments and introduce an interoperable reputation system. As a result, developers can focus on user experience and adoption, integrating their unique features into the provided base.

Trading logic is covered by escrow contracts ensuring that the funds are released only when predefined conditions are met. In case of dispute, moderators can take action and decide on the fair distribution of the funds, as is the case in OpenBazaar implementation as well. What makes Origin

Protocol interesting is its decentralized identity solution which enables users to build their reputation across different marketplaces developed using this protocol [23]. The way it works is that each public address is linked to one identity and is easily verifiable by anybody on the blockchain making sure that the reputation system is interoperable on-chain.

Another interesting mechanism Origin Protocol has is its tokenized ecosystem. It uses Origin Token (OGN) to incentivize users to join the marketplace and participate in decision-making processes. The token is used both as a utility token enabling access to premium features, as well as governance token enabling taking part in voting [22]. This approach ensures that everybody in the system is incentivized and contributes to its growth.

Founders of Origin Protocol were aware of the barriers of entry, so they put emphasis on user and developer accessibility. They developed tools to simplify onboarding, such as Origin Marketplace Creator Tool which provides a user-friendly and fast way to create a decentralized marketplace, while they created thorough documentation and APIs for developers [24]. By doing this, they want to lower barriers of entry, hoping for wider user adoption.

As other dApps mentioned in the chapter, Origin Protocol also faces scalability issues, given the fact that it is limited by Ethereum's throughput and gas fees. To solve it, the protocol is exploring layer 2 solutions like zk-rollups which enable faster and cheaper transactions as they process data off-chain but maintain security of Ethereum [25].

3 DecentMarkt

DecentMarkt presents the very core of this thesis. The name DecentMarkt comes from abbreviation decentralized marketplace, and it leverages blockchain technologies to provide a trustless platform for trading items. Initially, the idea was to create a decentralized marketplace only for second-hand clothes, inspired by the Vinted platform. However, during the project development, it made sense to extend the scope of the platform to include various physical goods to cover a wider range of use cases, similar to Subito.it and eBay.

The goal of this project is to explore the use of blockchain technology in e-commerce and provide potential solutions for some challenges it carries. In this chapter, the functionalities of DecentMarkt will be explained, as well as chosen solutions to ensure honest behavior between all parties involved in item purchase.

3.1. Product functionalities

DecentMarkt has all the functionalities one marketplace should offer. Firstly, and most importantly, the user is able to create product listings and view those of others. In the process of creating a listing, a user can choose a category and subcategory the product belongs to. The decision to introduce categories was to make filtering the listings easier, both from developer's and UX side. When listing an item, the user also inputs the title, description, condition, country where it ships from, list of corresponding images, and finally price and corresponding currency. DecentMarkt supports blockchain's native currency, which is POL on Polygon, and stablecoin USDC. Additionally, users can update any attributes of the product listing if necessary, keeping it up to date. If no longer needed, they can delete the listing after which the listing will be taken off the platform and will no longer be able to be purchased.

3.2. Stablecoins

Stablecoins are a type of cryptocurrency which are pegged to fiat currencies such as US dollar, or even commodities like gold, silver etc. [26]. On EVM blockchains, they are implemented as ERC-20 token standard. The crypto

market is often associated with volatility, thus using stablecoins in transactions and having an item's price listed in stablecoins, provide users with reliable prices with no fluctuations. Additionally, this approach fosters trust and tries to lower barriers of entry for new users since fiat currencies are something they are familiar with. In this implementation, USDC is the only stablecoin supported, as it is adopted for trading around the globe and has a strict regulatory compliance.

To pay for an order in stablecoins, a user needs to possess first the specified currency in their wallet. Upon paying, they will be asked to approve the spending of stablecoins on behalf of the marketplace smart contracts in the amount specified by the item price. This approval step is necessary as stablecoins are ERC-20 tokens and by the standard no contract has permission to access and transfer them. It is a security measure that safeguards user's funds and prevents unauthorized access. After the approval and transfer, the escrow contract holds the funds and releases them once the order is confirmed, and all the parties have fulfilled the requirements.

3.3. Wallet

When joining the marketplace for the first time, a user will be prompted to connect their cryptocurrency wallet to it. Wallet, in the context of blockchain, is a software tool that allows users to store, send, and receive cryptocurrencies with ease. It manages private and public keys necessary for the interaction with any blockchain and allows users to easily sign transactions while keeping the process safe and secure [27]. There are different kinds of wallets, offering different trade-offs between security and convenience. The most popular and easy to use ones are browser extensions such as TrustWallet. Additionally, these types of wallets are necessary for interacting with any decentralized app as they authenticate users without requiring traditional login credentials.

The integration of a wallet with DecentMarkt provides various benefits. It makes it convenient for users to perform any actions on the platform that requires interaction with a smart contract. In the wallet, users can easily sign transactions and verify the amount of funds they are sending to smart contracts. It also allows them to see the status of the transactions and verify the ownership directly from their wallet. Although this piece of software is necessary to be installed and configured first, the UI and onboarding process wallet offers, makes it easier for a newcomer to understand how blockchain interactions work, enhancing accessibility with easy onboarding.

3.4. User profile

On DecentMarkt every user interacting with the platform must have a user profile which is identified by their account address. User profile consists of basic information such as name, username, profile picture, email, and short description. Here users can also choose whether they are a moderator and input their moderator fee, which will be discussed in greater detail in the escrow chapter.

Complete user profile is a mix of on-chain and off-chain data. The aforementioned user data is public and stored in the smart contract, while the user's shipping address is stored in Firebase, a centralized cloud database. With this solution, the platform architecture becomes hybrid, using blockchain for transparency and decentralization with a centralized database to protect users' sensitive data. A user's shipping address is only visible to the seller who has to ship the purchased item to the buyer.

3.5. Reputation system

The very reason why any user performing interaction with the platform such as buying or listing an item must have a registered user profile is building the platform's trust and user reputation. The reputation system is an important part of DecentMarkt functionality as it fosters trust and accountability between users in a decentralized environment. In the lack of centralized authority, it is essential to ensure that users can trust each other and show that through verifiable data.

The way that the reputation system works is through reviews which can be submitted by all participants involved in completed purchase of an item - buyer, seller and optional moderator. Each review is tied to a specific purchase and is public, easily verifiable by everyone on the platform. Once submitted, the review cannot be changed, making it immutable.

All reviews a user has received are displayed on their public profile, along with GPA (grade point average) representing the average rating received from all reviews and offering a quick view of somebody's reliability on the platform. Along with GPA and reviews, additional information is visible on the public profile, such as the time when the user was last active on the platform, date they joined it, and total number of purchased and listed items. All this information offers a reliable way to evaluate potential sellers before purchasing an item from them.

It is important to note that the information about the last time a user was active on the platform is stored on Firebase. Such a design choice was made to ensure fast information retrieval and real-time updates, without requiring users to

pay gas fees to interact with the contract. It is an important piece of information since it prioritizes interactions with those users who are actively engaged with the platform. As a result, buyers who are in search of a product can easily identify those sellers who were recently active, increasing the likelihood of fast shipment and transaction completion.

3.6. Escrow system

The essential part of creating a secure, transparent and decentralized e-commerce is the escrow system. In general context, the escrow system is a type of agreement where a neutral third-party temporarily holds funds on behalf of the two parties involved in a transaction [28]. This third party moderates the transaction making sure that both sides fulfil their obligations, thus reducing the risk of fraud or dispute. In traditional e-commerce, an external escrow system is not necessary since the central authority acts as a trusted intermediary between buyers and sellers, enforcing rules and managing disputes. However, in blockchain, all transactions are irreversible making it impossible to reimburse funds in case of a dispute. Consequently, it is necessary to establish some kind of an escrow system or incentive for honest behavior. In the previous chapter, we've seen how different projects implemented different solutions, but in this chapter the escrow system of DecentMarkt will be discussed.

The way escrow mechanism in DecentMarkt is designed is through the use of moderators. Each user can mark themselves as a moderator by updating their profile and inputting their moderator fee. No user is excluded, and the only limitation is that the moderator cannot be a seller or buyer in the same transaction since it would defeat the purpose of a neutral third-party. When purchasing an item, the buyer is presented with a choice to involve a moderator or not.

For purchases executed without a moderator, the escrow system is disabled and funds are transferred directly and immediately to the seller. Buyers are warned about the risks of it beforehand and it should be used only for highly rated sellers or those with whom the buyer has already transacted with. In this type of transaction, buyers are fully responsible since there is no mechanism to get their funds back in case of a dispute. Finally, this is a low-cost option since no moderator fee has to be paid by the buyer and provides fast transaction execution.

For those buyers that opt in on moderator, they face an additional charge of moderator fee which is capped at maximum 10% but could be lower depending on the moderator. This type of purchase is recommended for higher value items requiring additional fund security. During the process,

buyers are presented with a list of potential moderators with their reviews, moderator fee and activity data. Moderator fee ensures that the moderator is incentivized to provide impartial and fair judgment. In this type of transaction, there are two ways in which a transaction can be completed, depending if both parties (seller and buyer) agree jointly on its outcome.

If both buyer and seller are satisfied with an item, they approve the transaction. For example, when a seller ships the item, they mark the transaction as approved, signaling that they fulfilled their role in the transaction. When the buyer receives the item and is satisfied with it, they also approve the transaction. In such a case, when there is approval from both sides, the transaction is automatically finalized and the funds are released to the seller, completing the transaction without involving any action from the moderator, but either way moderator fee has to be paid.

In a case when one of the parties involved files a dispute, the transaction can only be finalized by the moderator. It is then up to the moderator to review evidence provided in the chat. Based on their assessment they decide on the outcome and input the percentages of funds to be received by each party. For example, if the buyer received an item, but the seller forgot to approve the function, the moderator can step in and finalize the transaction allocating the funds to the seller, even though they forgot to approve it. In another case, when the item may have been damaged during shipment, they can decide to split the funds partially. This mechanism ensures a nuanced approach to every transaction and incentivizes all parties to behave honestly and fairly, since the outcome of their actions will be subject to reviews after the transaction has been completed.

3.7. Chat

For an effective shopping experience, communication between the participants is an essential element. It enables all parties to discuss listed items, clarify order details and resolve any issues or disputes. DecentMarkt supports chat functionality to enable seamless and secure communication between the users, fostering trust and real-time open communication between all participants.

While DecentMarkt utilizes blockchain for most of its data processing and storing, the chat system works off-chain, storing all messages in Firebase. While it is a centralized database, it is highly scalable, fast and can handle large volumes of data efficiently. A blockchain-based messaging system would be too costly and slow due to the need for gas fees and block confirmations. Additionally, chat messages would be exposed to the public. On the other hand, Firebase offers safe and secure storage of private chat data.

Chat functionality is seamlessly integrated in the user interface, with users being able to start the chat directly from the product listing, even before purchasing it. Buyers can thus ask about product details, check shipping options, and resolve any doubts beforehand. When a user initiates a chat, a chat document is immediately created, tied to the specific item and identified by the wallet address of participants. If a user purchases an item involving a moderator, the moderator is also added to the chat and has access to the whole message history, making sure they stay informed from the very beginning of the conversation. In that way, the same chat can be used later throughout the dispute process in case it happens. Furthermore, chat participants receive notifications for new incoming messages, making sure they stay up to date without the need to constantly check their conversations. Lastly, each chat is linked with item, seller and buyer, ensuring that the conversations stay bound to that specific transaction or inquiry.

3.8. Notifications

The notifications system in DecentMarkt is an important part in keeping the users stay up to date with important actions relevant to them. It ensures that the users stay engaged throughout the process and remain responsive to the other party's needs.

As with the chat system, the notification data is stored in Firebase for real-time delivery. Keeping notifications off-chain also provides fast and cheap storage, all while protecting private data. Frontend components interact with it directly by triggering a new notification storage whenever important events occur such as seller's item being bought, dispute initiated, transaction finalized, new chat message etc. These notifications are always visible on UI ensuring that the user sees them and stays up to date. In transactions, all parties receive notifications relevant to them, staying informed and taking actions if necessary.

3.9. Other functions

Other functionalities that are worth mentioning improve user experience and navigation throughout the platform. They provide an easily accessible view over favorite items, orders, listings and moderator items.

Each one of those sections can be easily filtered by attributes, such as order filtering based on completion, dispute or waiting for approval by one of the parties. User's personal listings can be filtered by purchase status - bought or still listed. Additionally purchased listings can be filtered by previously mentioned filters for orders. The same functionality is with moderated items

that are filterable as well. On the homepage, all listed items are easily viewable, paginated, and can be searched and filtered by any of the item's attributes.

4 System architecture and technology stack

DecentMarkt has a layered approach to architecture which integrates both off-chain and on-chain components for improved scalability, cost efficiency and privacy. In this chapter all components of the decentralized marketplace will be explained in greater detail, as well as the underlying technologies powering them.

In Figure 4.1 below the overall system infrastructure can be seen with the interactions between them.

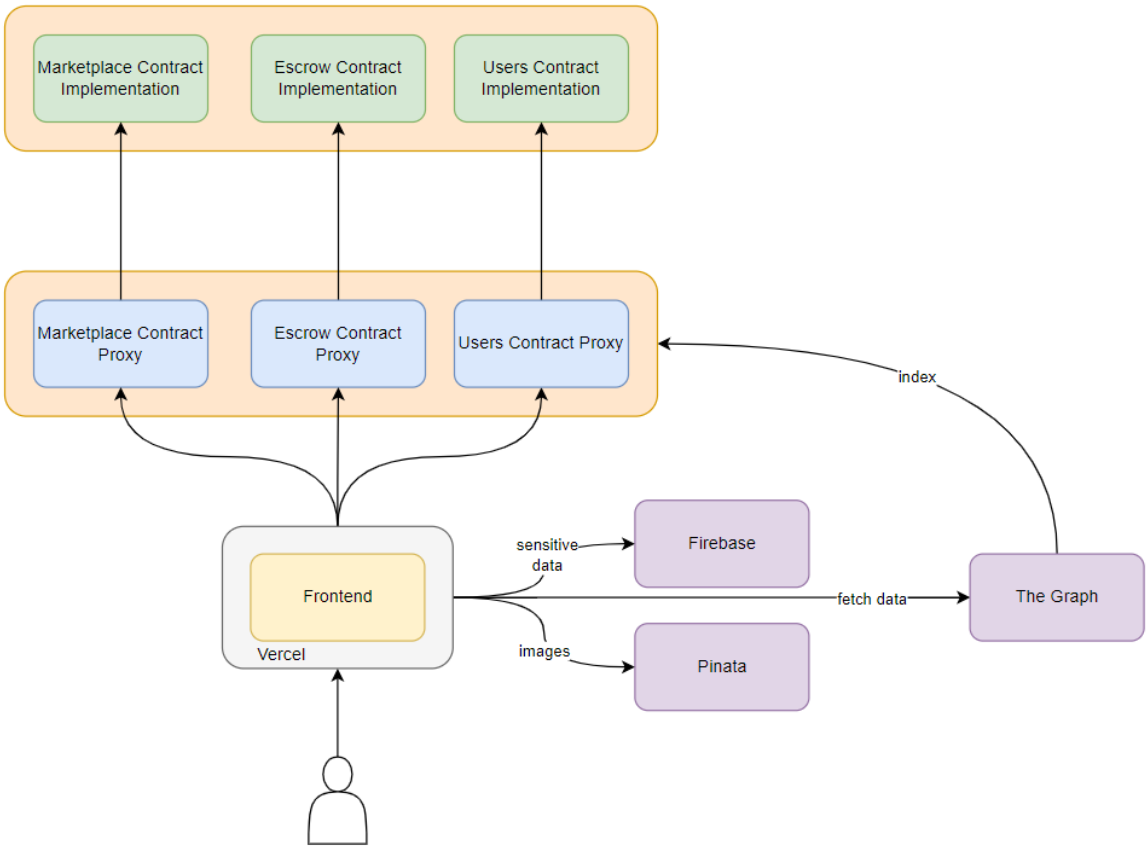


Figure 4.1: DecentMarkt System Architecture

4.1. Smart contract architecture and design

DecentMarkt business logic is powered by 3 smart contracts - Marketplace, Escrow and Users. These smart contracts contain all logic and rules for trading, user profiles and item listings. While they present different domains of responsibility, they are interlinked and dependent on each other to provide the full functionality of the platform. Such an approach also ensures modularity allowing each contract to be upgraded or replaced as long as their interfaces remain compatible.

4.1.1. Marketplace contract

Marketplace smart contract presents the primary contract with which sellers and buyers interact. It is responsible for storing all items and links them with its owner. All externally facing methods emit events which allow an indexer to listen to them and update their off-chain ledger. The main interface methods of this smart contract are:

- **list item** - To list a new item, sender has to send a transaction with all relevant item attributes such as title, description, price, currency (native or the supported ERC-20 stablecoin USDC), photos, item condition, belonging category and subcategory. Upon listing a new item, a call will be made to the User contract to validate that the user has registered to the platform, since only registered users can list items and interact with it. This decision was made to increase the trust between the users and ensure their commitment to the platform. Users that are not registered can still view listed items and browse them but not interact with them in any other way. To decrease the transaction costs and improve smart contract storage efficiency, all photos are first sent to IPFS service provider Pinata, and then only their IPFS hash is stored in the smart contract. In this way, photos take less storage in the smart contract leading to lower transaction costs.
- **update item** - If the item has not been purchased yet, its seller can update all its attributes, including the price and the currency. This allows sellers to adjust prices, or any other attributes based on the market conditions or popularity of the item.
- **delete item** - As with the other basic operations, a seller can delete their item only if it hasn't been bought. This simple operation just changes the item status which prevents it from being purchased. All CRUD operations can be performed only by the seller, blocking any malicious attempt to modify an item by other users.

- **buy item** - When purchasing an item, a buyer can opt for a moderator or not. Depending on the choice, a different function is called with different logic. For both types of purchases, various checks are first performed, such as availability of buyer's funds for purchase completion, and, in case of stablecoin purchase, allowance of spending in the required amount for escrow contract is required. Additionally, the moderator fee is paid by the buyer. Furthermore, the user must be registered, the item cannot belong to them, and it has to be currently listed, i.e. not already bought or deleted. In the case of a purchase with a moderator, it is required that the moderator is also registered, and they cannot take any other role in the purchase, i.e. moderator cannot be seller or buyer at the same time for the obvious conflict of interest. Finally, to proceed with the trading logic, a call to the escrow contract is made to different functions depending on the purchase type.

4.1.2. Escrow contract

The escrow contract locks the funds and handles their distribution to the parties involved in the purchase. It encapsulates all the trading logic ensuring that the trust required between the parties involved is minimized. When a buyer purchases an item, the funds are locked in the contract until the transaction finalizes, either through agreement between a seller and a buyer, or through a chosen moderator. This approach guarantees that the seller will receive funds only when the buyer confirms satisfactory delivery. If the dispute arises, the funds distribution is handled by moderator who intervenes and fairly distributes the funds by looking at the evidence provided through the chat.

By separating trading logic to escrow contract, the platform ensures that sensitive functions such as creating a transaction or finalizing it can be called only by specified actors. Additionally, it increases the modularity of our smart contract system and makes it easier to audit financial flows.

The main methods of escrow contract are the following:

- **create transaction** - This method is highly sensitive because it effectively creates an item order, thus the item is taken off the marketplace. Because of this, the method can only be called my marketplace contract directly. When calling buy item, firstly, marketplace contract is called to perform various checks if the user is able to buy the item. Secondly, funds are then transferred to the escrow contract which takes responsibility to conclude the transaction. In this method, a transaction is saved with information about parties involved, price, and moderator fee if present. In the case of no moderator, funds

are transferred immediately to the seller without the need for parties' approval. This transfer is irreversible and cannot be disputed, hence it is highly risky and only recommended for those sellers with whom the buyer has already had some previous successful transactions.

- **approve transaction** - For purchases with moderator involved, both parties can approve and dispute the order. To release the funds to the seller, it is necessary for both parties to approve the transaction. When the second party, no matter the order, approves the transaction, all funds will be automatically released to the seller and the transaction will be finalized. Afterwards, all parties are invited to review each other.
- **dispute transaction** - In case when one of the parties is not satisfied with the outcome of the order, the item is not delivered, it is damaged, or not corresponding to the description, or the buyer forgot to approve the order, they can file a dispute for it.
- **finalize transaction** – It is a method that can be only called by the assigned moderator of the transaction in case of a dispute. The dispute will trigger the intervention of the chosen moderator who can then inspect the evidence provided by the parties and decide on the fair percentages of the funds each party will receive. After inputting the percentages, the funds are distributed, and the transaction is finalized. Afterwards, all parties are invited to review each other, contributing to the platform's greater trust and reliability.

4.1.3. Users contract

The Users contract serves as a registry of all users registered to the decentralized marketplace. It stores user profiles and reviews associated with it. This contract provides a secure and transparent way to manage users and their reputation which is verifiable by anyone on the blockchain. Some functions of the contract are called by marketplace and escrow contract when performing certain checks, such as if a user is authorized to perform a certain operation, or if the transaction is finalized and ready for review.

The main methods of this contract are the following:

- **create profile** - Allows participants to register to the platform which is a prerequisite for interacting with other platform functionalities such as buying or listing an item. To register on the platform, a user needs to input their username, first and last name, profile description, email, country, and choose their profile image. As with an item's images, a user's image will be stored on IPFS and only the image IPFS hash will

be stored in the contract to save on gas fees. During registration users can also choose if they want to be a moderator on the platform and have to provide their moderator fee. The moderator fee is capped at a maximum of 10% and that threshold can be lowered only by the contract owner.

- **update profile** - A user can update any of their profile information, including the choice for being a moderator and their moderator fee. It's important to note that for each transaction moderator fee at that point in time is saved in escrow contract, so moderator fee modification in users contract will only affect transactions occurring after the fee update to prevent potential malicious behavior by moderators.
- **create review** - Enables transaction participants to leave feedback for each other after a finalized transaction. Reviews have an important role in the reputation and overall trust of the platform, fostering respect and accountability in the marketplace. To create a review, the user has to input the id of an item that the review is referring to, the wallet address of the participant being reviewed, textual content of the review and the overall rating from 1 to 5. During a review creation numerous checks are performed such as a check for transaction status to verify that the transaction is finalized, and that the participant is actually a participant in the transaction. By maintaining all reviews on-chain, the system prevents tampering and provides a transparent record of user's activity on the platform.

4.1.4. Contract proxies and upgradeability

One of the challenges of developing a decentralized application on blockchain is the very immutability of the underlying ledger. In traditional software development, software upgrades are standard and occurring practice to improve performance, bring new features and fix bugs. However, since deployed smart contracts contain immutable logic, which brings many benefits such as trust and security, this creates a challenge when the new logic has to be implemented.

To overcome this issue, there are many upgradability patterns that allow developers to introduce new changes without compromising on immutability of the deployed address or user data. The solution adopted in this project is a transparent upgradable proxy pattern. This pattern ensures that the storage remains consistent while the logic can be updated seamlessly.

The transparent upgradeable proxy pattern is a smart contract upgrade mechanism based on proxy contracts and was popularized by OpenZeppelin,

a blockchain organization providing open-source, secure and audited smart contract libraries for developing dApps [29]. The way it works is through separating logic from storage allowing developers to replace the implementation when needed. It is composed of 3 components:

- **proxy contract** - a lightweight contract that functions as a storage. The only logic it holds are delegation calls to respectable functions in the logic contract. The way proxy contract works is by utilizing *delegateCall* opcode in Ethereum which allows a contract to execute functions from another contract with its own storage. Essentially, it is a way of forwarding a call to the implementation contract while the results remain in the proxy contract. User interacts only with the proxy contract and sees no difference from the implementation contract since the interfaces are the same. They are completely unaware of the underlying delegation mechanism occurring. The address of the logic contract to which all calls are forwarded is stored in the proxy contract and can be only modified by the assigned admin. Since the proxy contact contains only delegate calls, its impact on the overall gas fees is minimized.
- **logic contract** - contains the implementation of the business logic in the form of functions. This contract can function on its own, but in this pattern its address is saved in the proxy contract which calls it. In this pattern it is important that all calls to the system happen through the proxy contracts since they hold the state of the application. Direct calls to the implementation of contracts can have unintended consequences such as state inconsistencies or lost results of operations.
- **admin role** - address that owns the proxy contracts and has permission to upgrade them to point to a new logic contract. It is the only entity allowed to change the address of the implementation contract in the proxy contract and any other attempts will be disregarded. The assigned admin can be modified through authorized functions in the proxy contract.

The way that the update process works is through *upgradeTo* function in the proxy contract which is only accessible to the assigned admin. It changes the address of the implementation contract and from that point all calls will be delegated to the new version. This separation of the logic and proxy contract ensures that the storage is persistent between the logic contract versions. It is important that the storage layout stays consistent between the logic contract updates - new state variables can be added at the end of the existing layout, but the already existing variables cannot be reordered or removed.

The biggest benefit of this pattern is that the updates occur without any data lost. For the case of the decentralized marketplace, new features can be introduced without requiring participants to recreate their user profiles or list their items again. Another benefit is that the only contracts needed to be indexed are proxy contracts, which simplifies the maintenance and update of the system.

4.2. Indexing with The Graph

Blockchain applications produce various data in the form of transactions and events they emit. To access this data on the frontend or other off-chain components, it can be cumbersome and quite resource-intensive to query blockchain every time some data is needed. This inefficiency becomes a greater problem in the context of decentralized applications where fast querying of item listings and user actions is necessary for a seamless user experience. To address this problem, indexers are utilized. They listen to event data emitted from smart contracts and extract, process and store them in their database for easy, fast and cost-efficient access. There is a wide range of indexers available, most of which are centralized, but for this project The Graph indexer was chosen since it is based on a decentralized protocol.

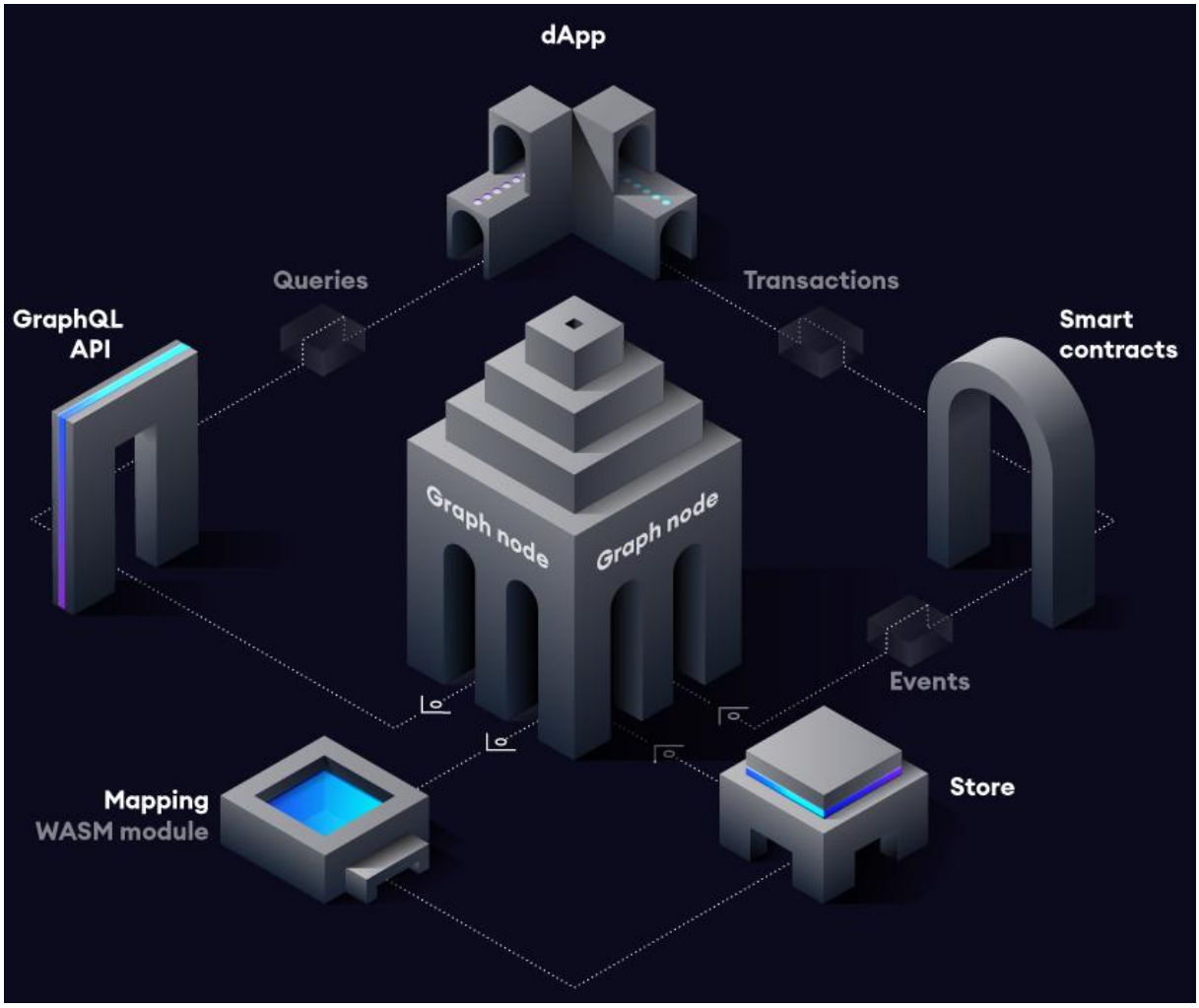


Figure 4.2: The Graph infrastructure

The Graph decentralized indexing protocol is based on a system of subgraphs and graph nodes [30]. In the context of The Graph, a subgraph is a description of how to process and store on-chain data for a specific set of smart contracts. The data emitted from smart contracts are picked up by a graph node that actively listens to their events, then decodes them using the provided ABI (application binary interface) in the subgraph specification. Finally, it stores them in a format suitable for queries through GraphQL API. The infrastructure of the protocol can be seen in Figure 4.2.

A subgraph is created by defining the data model that will be used to retrieve and store the entities to be indexed. Additionally, it requires an ABI to decode event data and event handlers that map emitted events to specific functions for further processing. When an event is emitted, a graph node will call a specific event handler and store the event data in specific format.

In the case of DecentMarkt, data is indexed from three smart contracts - Marketplace, Escrow and Users. It is important to note that the addresses that

are being listened to are those of the proxy contracts, while ABI is that of the implementation contracts. The opcode *delegateCall* also emits those events from the logic contracts as if they were emitted from the proxy contracts, thus only the proxy contracts are listened to since they store the application state. In each of these three contracts, specific events are emitted relating to the distinct domains of each contract. For example, the marketplace contract emits a *MarketplaceItemListed* event which contains data of the listed item which is then processed by *handleItemListed* function and lastly stored in the Item entity in the subgraph. There are various events, some of which are *UserRegistered* emitted from Users contract which is mapped to User entity. Another example of an event is *TransactionCreated* from Escrow contract that is mapped to Transaction entity via specified event handler in the subgraph.

The Graph decentralized protocol for indexing offers various benefits. Firstly, it is very efficient in data retrieval since it uses GraphQL API which allows developers to specify only those data that are needed, reducing the need for retrieval of redundant data and resource waste. Additionally, since it is a decentralized protocol, it offers a robust, reliable infrastructure without any service interruption and single point of failure. Lastly, The Graph indexing protocol supports more than 80 networks, including test networks, making it easy for developers to easily deploy and test subgraphs on different networks expanding their services to the wider blockchain audience.

4.3. Firebase

In the development of decentralized applications, there are challenges of scalability, cost efficiency and data privacy. To overcome this problem, DecentMarkt can be seen as a hybrid using blockchain as a decentralized ledger for the most important functionalities, while relying on a centralized service such as Firebase to provide seamless user experience, protect user data in a scalable way.

Firebase is a platform developed by Google which provides developers with a variety of tools and services to help them build applications. It is a backend-as-a-service (BaaS) platform that offers hosted backend services and databases with additional features such as authentication mechanisms, cloud storage and static files hosting service [31].

DecentMarkt relies on different firebase collections to provide full user experience. The following collections are hosted on Firebase:

- **addresses** - stores shipping addresses associated with the user's blockchain address. Each user can have multiple shipping addresses

which must be kept private and away from publicly accessible blockchain since they contain sensitive data.

- **users** - stores the last seen timestamp of each user. The last seen attribute helps identify users that regularly visit the platform and offers an insight to other users on how active the user is helping them with their decision if they should rely on them for timely resolution of the order and transaction. Each time a user logs onto the platform or performs any activity, the timestamp is saved in this Firebase collection.
- **orders** - stores a chosen shipping address associated with each order. During the purchase process buyers are asked to select which address they want their order to be shipped to since they can have multiple addresses. The chosen shipping address is then added to the order and shown to the seller only. Each order is identified by item id which is unique across the platform and relates directly to the order for easy information access.
- **favorites** - tracks items favorited by each user. The favorite items section within the frontend provides easy access to the most desired items which could be later purchased by the user. When favorited items are bought by another user, all users that have favorited that item are notified, and the item is deleted from their list of favorites.
- **chats** - manages user chats and messages within chats. Each chat is identifiable by a hash created from unique blockchain addresses of each participant. Each entry in the chats collection contains an array of participants which is used to easily fetch chats belonging to one user. Before an item is purchased, the user can chat with its seller. After the purchase the previous chat messages relating to that item are persisted. Additionally, a moderator can join the chat and see the full history of correspondence between the order participants.
- **notifications** - stores user specific notifications. Each user has a document in the collection containing an array of notifications belonging to them. Each individual notification contains fields like *content*, *actionUrl*, *type* and *isRead* status. The *actionUrl* field allows users to go directly to the area of notification interest, e.g. if their item is purchased notification directs them straight to the order containing all the relevant information. Also, the type of notification allows for different styling of each notification relating to their type. For example, notifications relating to purchase of an item different icons compared to dispute notifications, making them easily distinguishable.

By organizing Firebase collection hierarchy this way, DecentMarkt achieves optimal performance and offers great scalability, while maintaining ease of information management and its privacy.

4.4. Pinata & IPFS

As with any other software application, data storage is a critical component. In decentralized applications storing data in a decentralized way can present a challenge due to increased costs and scalability issues. To address these challenges, a decentralized storage protocol called IPFS that stands for InterPlanetary File System is developed. It provides alternative to centralized storage solutions through peer-to-peer distributed storage across the network of nodes.

IPFS provides storage for static files, websites and even applications. To ensure immutability and data integrity, IPFS uses content addressing in which every file is assigned a content identifier (CID) based on a unique cryptographic hash [32]. Any change to the file would result in different hash, ensuring that the data changes are easily noticeable. When a file is uploaded to IPFS, it is split into smaller chunks each of which is hashed, and the user is given a unique CID which uniquely identifies the file. The file is then stored locally on the node that added the file, and other nodes can store it as well by voluntarily pinning it. The more nodes pin the file, the more its availability and faster retrieval are ensured.

Since pinning files is done completely voluntarily by the nodes, which can be challenging to achieve, pinning services have been developed, one of which is Pinata. By pinning the files, Pinata guarantees that the file remains available and accessible even in the case of a local IPFS node shutting down [33]. Additionally, for seamless IPFS integration, Pinata provides APIs for file upload, retrieving and deleting files from the IPFS network.

In DecentMarkt, IPFS and Pinata are used to store only images of item listings and user profiles. For example, when a user lists an item, the item images are uploaded to IPFS via Pinata API and pinned to the network. Their unique identifier, CID, is stored in the smart contract ensuring its immutability. Afterwards, when the item is viewed, the CID of its image is fetched from the blockchain and through IPFS gateway the image is retrieved and displayed.

An alternative to storing images in a distributed way would be to store them in a byte format which would be extremely expensive and resource intensive. For this reason, only a hash of the image is stored, while the image is stored off-chain, either centralized on Firebase or decentralized on IPFS.

Additionally, with this mechanism of storing, there is no compromise on scalability and on image quality since IPFS saves images in high-resolution.

4.5. Next.js and Vercel

Modern web applications require robust framework and scalable deployment to ensure accessibility, fast feature integration and seamless user experience. To develop a user-facing side of the platform, Next.js is employed and paired with Vercel for a seamless continuous and scalable deployment.

Next.js is an open-source framework based on React that provides developers with all the tools necessary to build modern web applications [34]. It is developed by Vercel, the same company that provides the deployment platform. Next.js simplifies development through features like API routes, dynamic routing and server-side rendering. It also has built-in support for tailwind CSS which makes designing and styling the application easier.

Vercel is a cloud platform that offers hosting and deployment services for frontend applications [35]. Its interface is easy to use and has built-in support for the deployment of the most popular frontend frameworks such as Next.js, Angular and Vue.js. It packs numerous features such as automatic scaling, serverless functions and continuous deployment through integration with a git repository.

4.6. Hardhat

Hardhat is a framework for creating Ethereum-based blockchain applications. It simplifies the process of developing, testing and deploying smart contracts on EVM based blockchains. This framework is used to develop and test various functionalities of smart contracts powering the DecentMarkt platform.

Hardhat offers various features to make development efficient and seamless. Firstly, it includes a local Ethereum network which simulates blockchain functionality making testing much easier. In that way, contracts are deployed in a local, controlled environment where debugging and tracing error messages is easier and faster than in the real-world use case, resulting in reduced development time. Furthermore, Hardhat integrates popular testing libraries such as Mocha and Chai which are used to write and execute tests that validate the behavior of smart contracts. Additionally, it has a wide plugin ecosystem enabling integration with Ethers.js that is used for interacting with the deployed smart contracts, and various other plugins that analyze gas fees and give insights into how to optimize them. Lastly, it simplifies the deployment of smart contracts through deployment scripts that are easily

configured and support various deployment patterns with a wide range of blockchain networks [36].

5 Application testing and evaluation

Testing and evaluation are the final steps in the software development process that ensure proper functioning of the developed components. In the blockchain context especially, their security, accuracy and cost efficiency are of paramount importance. For this reason, thorough auditing is necessary. This chapter presents the testing methods used during the development of the decentralized marketplace platform. Furthermore, it evaluates associated blockchain costs on Ethereum and Polygon and compares them, highlighting the importance of choosing the right network in relation to the use case.

5.1. Smart contract unit tests

To test smart contract functionality, unit tests were developed. Unit testing is a type of testing which isolates individual components and verifies their correctness in various conditions isolated from the rest of the system [37]. In blockchain-based applications it is essential to ensure that the contracts' functions work properly since upgrading them after the deployment can be a tedious and cumbersome process.

Hardhat framework provides a thorough testing environment by enabling a local blockchain simulation and various libraries such as ethers.js to interact with a local deployment. Additional tools like Mocha which is a widely used JavaScript testing framework for automated unit testing, while Chai provides assertions of the expected results with real outputs. The tests were structured into separate files, each for the main contracts - Marketplace, Escrow and Users - validating the correctness of their unique domain.

The following tables (Table 5.1, Table 5.2 and Table 5.3) show various tests performed on the smart contracts with their description and expected result. During the tests, special attention was given to ensuring that, even in the case of failure, the smart contract reverts with the right custom exception, creating an easier debug process for the end user.

Table 5.1: Users Tests

Test Name	Description	Expected Outcome

Deployment Validity	Verifies if the Users contract is deployed successfully.	No errors or reverts.
Profile Update	Ensures users can update their profiles.	Profile update completes without reverting.
Set Moderator Max Fee (Owner Only)	Tests if only the owner can set the maximum moderator fee.	Only the owner can modify the max fee; non-owner operations are reverted.
User Registration Check	Confirms if a user is registered.	Returns true for a registered user.
Moderator Status Check	Checks if a user is a moderator.	Returns true if the user is a moderator.
Profile Retrieval	Tests if a user's profile can be retrieved.	Successfully retrieves the user profile without errors.
Create Review	Validates the review creation process.	Allows one review per user per item; subsequent reviews are reverted.

Table 5.2: Marketplace Tests

Test Name	Description	Expected Outcome
Deployment Validity	Verifies if the Marketplace contract is deployed successfully.	Returns an initial item count of zero.
Invalid Photo Hash	Ensures item listing reverts with an invalid photo hash.	Reverts with the "NotIPFSHash" custom error.
Valid Photo Hash	Confirms item listing succeeds with a valid photo hash.	Item listing completes without reverting.
Photo Limit Exceeded	Ensures listing fails if the number of photos exceeds the limit.	Reverts with the "PhotoLimitExceeded" custom error.

Set Users Contract Address (Owner Only)	Verifies only the owner can set the users contract address.	Allows owner to set; reverts for non-owners.
Add Supported Token (Owner Only)	Tests if the owner can add supported tokens.	Successfully adds token for owner; reverts for non-owners.
Prevent Seller From Buying Own Item	Checks that sellers cannot buy their own items.	Reverts with "SellerCannotBuyItsItem" custom error.
Purchase Without Moderator (ETH)	Ensures items can be purchased without a moderator using ETH.	Purchase completes without reverting.
Purchase Without Moderator (Tokens)	Confirms items can be purchased without a moderator using ERC-20 tokens.	Purchase completes without reverting if approvals and allowances are valid.
Invalid Moderator	Ensures only valid moderators can oversee transactions.	Reverts with "MustBeModerator" custom error for invalid moderators.
Purchase With Moderator (ETH)	Validates purchases with a moderator using ETH.	Purchase completes without reverting.
Purchase With Moderator (Tokens)	Tests purchases with a moderator using ERC-20 tokens.	Purchase completes without reverting if approvals and allowances are valid.
Delete Item	Ensures listed items can be deleted.	Item deletion completes without reverting.
Update Item	Verifies items can be updated after listing.	Item update completes without reverting.

Table 5.3: Escrow Tests

Test Name	Description	Expected Outcome

Deployment Validity	Verifies if the Escrow contract is deployed successfully.	No errors or reverts.
Finalize Transaction (ETH, Two Approvals)	Ensures transactions can be finalized with ETH after approvals from both parties.	Transaction finalizes successfully.
Finalize Transaction (Tokens, Two Approvals)	Ensures transactions can be finalized with ERC-20 tokens after approvals from both parties.	Transaction finalizes successfully.
Finalize by Moderator (ETH)	Validates dispute resolution and transaction finalization by a moderator.	Transaction finalizes successfully after dispute resolution.
Finalize by Moderator (Tokens)	Validates dispute resolution and transaction finalization by a moderator for ERC-20 tokens.	Transaction finalizes successfully after dispute resolution.

The testing process created high coverage overall with the following coverage in specific contracts shown in Table 5.4.

Table 5.4: Test Coverage Report

File	% Statements	% Branch	% Functions	% Lines
Escrow	95.95	56.86	91.3	86.92
Marketplace	96.88	61.02	100	88.29
Users	96.55	50	100	82

5.2. Blockchain costs for different networks

Decentralized applications based on blockchain inherently come with associated costs of blockchain usage, that is borne by its users. The costs of deploying the contracts and interacting with them are an important factor in the overall success of the application. Different blockchains come with different costs, but not only that, they offer different sets of features and

scalability options that should be considered when choosing which EVM-supported network to deploy smart contracts to. In this chapter, the costs between EVM blockchains Ethereum and Polygon will be analyzed and discussed.

Blockchain costs occur every time some computational resources are required to execute functions in smart contracts. These costs are referred to as gas, which is a unit representing computational effort on the Ethereum Virtual Machine. Gas has several purposes for EVM. Firstly, it presents a computational cost required to execute operations. Secondly, it deters spam and controls access to the network by making it expensive to execute spam attacks and take over the network activity. Furthermore, gas fees are part of the reward going to the validators for verifying and adding transactions to their proposed block.

Gas prices fluctuate and are based on the network's current demand and activity. If the activity of the network is high, users must pay more in gas prices for their transactions to be included in the next block. It is important to note as well that gas spent on operation execution depends on its complexity and each atomic operation has a specific gas cost associated with it. Consequently, special attention should be paid to optimizing smart contracts, so they don't waste gas unnecessarily. Lastly, gas is paid in blockchain's native currency - ether (ETH) for Ethereum and POL for Polygon.

For calculating the complete cost for a transaction, the following Equation 5.1 is used:

$$cost\ in\ fiat\ currency = gas\ used \times gas\ price\ \times exchange\ rate$$

Equation 5.1: Fiat Cost of Gas Calculation

The cost analysis of the most common smart contract functions is of great importance when assessing the cost of overall smart contract interactions and general business viability. It helps in choosing the right network to deploy the smart contract infrastructure. For the estimation of gas costs and their conversion to fiat currency, hardhat gas reporter plugin was used along with coin market cap API which fetches the latest gas price and exchange rates. The hardhat gas reporter plugin estimates gas costs by running tests and measuring the gas consumed during their execution. By integrating gas costs within tests, it helps developers write real world scenarios and evaluate the efficiency of contract functions.

The following tables (Table 5.5, Table 5.6, and Table 5.7) show the most used functions of each smart contract with their associated gas costs. The last two columns show the complete costs expressed in EUR of running these functions on Ethereum and Polygon networks.

Table 5.5: Users Costs

Method	Avg Gas	Ethereum Cost (EUR)	Polygon Cost (EUR)
createProfile	356031	5.95	0.00
createReview	157425	2.63	0.00
initialize	71961	1.2	0.00
setMaxModeratorFee	28823	0.48	0.00
updateProfile	78522	1.31	0.00

Table 5.6: Marketplace Costs

Method	Avg Gas	Ethereum Cost (EUR)	Polygon Cost (EUR)
buyItem	325935	5.45	0.00
buyItemWithoutModerator	254527	4.25	0.00
deleteItem	48262	0.81	0.00
initialize	1473418	24.63	0.00
listNewItem	414834	6.93	0.00
updateItem	135388	2.26	0.00

Table 5.7: Escrow Costs

Method	Avg Gas	Ethereum Cost (EUR)	Polygon Cost (EUR)
approve	45018	0.75	0.00
finalizeTransactionByModerator	89760	1.5	0.00
initialize	94242	1.58	0.00
raiseDispute	31282	0.52	0.00

Table 5.8: Deployment Costs

Contract	Gas Used	Ethereum Cost (EUR)	Polygon Cost (EUR)
Escrow	2195505	36.7	0.01
Marketplace	3333997	55.73	0.01
Users	2174539	36.35	0.01

The comparison between Ethereum costs and those on the Polygon network highlights substantial cost savings of deploying (Table 5.8) and interacting with the smart contracts on Polygon. Ethereum is the leading and the most popular EVM-based blockchain network with wide community and support. However, it suffers from scalability limitations and high transaction fees. Both networks rely on proof of stake as consensus mechanism, but Ethereum can process 12-15 transactions per second while Polygon currently handles about 38 TPS, with 429 TPS maximum recorded [38]. Additionally, block time on Ethereum is 12 seconds, while Polygon creates a new block every 2 seconds.

The substantial cost savings of the Polygon network can be attributed to its architecture. As a layer 2 solution, which is built on top of a primary blockchain (also called layer 1), it improves scalability and reduces cost. Polygon achieves this by creating a sidechain from Ethereum's main chain and processing transactions off-chain on its side chain [39]. It periodically commits them to Ethereum, which leverages its security while keeping gas prices lower and higher TPS. Additionally, Polygon's network is less congested, making gas prices relatively stable. Lastly, its native currency POL is valued substantially less than ETH (0.4531 USD to 1 POL compared to 3355.18 USD to 1 ETH as of December 2024), making the cost difference greater when converted to fiat currencies.

As a result of the cost analysis of these two networks, the choice was made to deploy the smart contracts to Polygon's testnet Amoy. DecentMarkt is a platform that requires frequent interactions with the blockchain, thus it is essential to choose a network with lower gas fees and higher throughput. Polygon offers a wide community and documentation with minimal gas fees, making it a great platform for its affordability and scalability. On the other hand, Ethereum offers paramount security and decentralization, but high transaction fees have pushed developers into looking for other solutions such as layer 2 solutions like Arbitrum, Polygon and Solana. On its roadmap Ethereum wants to transition to sharding, which divides the network into smaller segments called shards which would increase its TPS and lower gas costs. The implications of such a solution are to be seen in the future, but at the time of writing, layer 2 solutions present a great cost-effective alternative to Ethereum.

6 Conclusion and future development

The last chapter of this thesis summarizes the work done, reflecting on the state of the current blockchain technology, exploring its implementation in e-commerce, and providing a solution with the practical project of decentralized marketplace called DecentMarkt. The aim of this thesis was to explore through the project various benefits of blockchain technology in e-commerce space, as well as its drawbacks, highlighting what areas need to be improved to ensure wide accessibility and ease of use.

Blockchain technology has the potential to revolutionize the way people trade around the world. The initial idea was to create a way for people to trade directly, but that idea evolved and the current generation of blockchain brings more functionalities, such as the ability to run decentralized applications (dApps), decentralized finance (DeFi) services, support for fungible and non-fungible tokens that extend beyond the basic exchange functionality of cryptocurrency, etc.

Blockchain brings numerous benefits, such as transparency, censorship-resistance and decentralization, allowing people to trade directly, without the need of centralized authority. In a decentralized public ledger, all information present on it is publicly visible, enabling anyone to verify and prove the correctness of the information. On the other hand, blockchain technology still has some drawbacks, such as problems with scalability, costs of usage and lack of regulation that has an impact on the public perception of it. All aforementioned blockchain features have been properly introduced and analyzed in depth in the previous chapters, along with technical background necessary for understanding the broader context.

After introducing the fundamentals of blockchain, research was made to offer an insight into the related projects in the field of decentralized e-commerce. One of the most well-known decentralized marketplaces was OpenBazaar, which inspired the creation of this project. Its goal was to create a censorship-free marketplace without any restrictions imposed by the governments. Several other projects were created as well, two of which were analyzed in this

thesis - Particl and Origin Protocol. Particl marketplace is privacy-focused and relies on its own blockchain Particl which supports payments only in the native cryptocurrency PART. Because of that, they can offer a high degree of user privacy. On the other hand, Origin Protocol provides a framework for creation and development of decentralized marketplaces, instead of a ready-made solution. It offers additional features such as an interoperable reputation system and uses tokenization to incentivize users to join the marketplace and participate in decision-making processes. These analyzed projects demonstrate that while decentralization offers great benefits, achieving mass adoption is quite difficult and requires addressing scalability, usability and regulation challenges of using blockchain.

To overcome these challenges, DecentMarkt was created to offer a trustless, transparent, and user-friendly decentralized platform for trading second-hand goods. The core logic of the platform is powered by three smart contracts - Escrow, Marketplace and Users, each of which addresses a specific domain. Users contract stores user profiles with their reviews, while Marketplace contract is used for listing and purchasing items. The Escrow contract ensures fair trade by temporarily holding funds until both the buyer and seller confirm successful order completion. In case of a dispute, a chosen moderator can step in and decide on the fair distribution of the funds, collecting a fee for their service. Furthermore, to enhance trust in the platform, DecentMarkt has a reputation system where all parties in the order can leave a review for each other after its completion. Additionally, DecentMarkt supports stablecoin USDC in the form of ERC-20 tokens, which helps users mitigate volatility risks while trading.

The infrastructure of DecentMarkt presents a hybrid on the spectrum of decentralization. It relies on smart contracts for business logic, and decentralized indexing protocol called The Graph for fast information retrieval. Additionally, it uses centralized database Firebase for storing sensitive user information such as shipping addresses, chats and notifications. Such an approach allows both decentralization and cost-effective user experience. To optimize smart contract storage and reduce usage costs, images are stored on IPFS and pinned by Pinata, while smart contracts only store their CID hash.

The architecture of smart contracts is designed to be modular and upgradeable, allowing for future development. It relies on a transparent upgradable proxy pattern which forwards the function calls to implementation contracts, while all the data is kept in proxy contracts. This pattern allows for contract upgrades and new features, while keeping the data intact.

After developing the platform, the next step in the process was to test the functionalities to ensure they work properly. To test smart contract functionality, unit tests were utilized and developed in the Hardhat testing environment which provides a local instance of a blockchain. These tests also helped estimate the costs of calling the most important smart contract functions. Each blockchain comes with associated costs of usage, so it was important to decide on which blockchain network to deploy the smart contracts system to. After extensive cost and throughput analysis, it was decided to deploy the smart contracts to Polygon's testnet Amoy which is a layer 2 solution offering lower gas fees and greater scalability than the most popular EVM blockchain called Ethereum. Additionally, Polygon has faster block time that leads to quick transaction confirmation improving DecentMarkt usability overall.

6.1. Future developments

While DecentMarkt has all the functionalities to offer a good, decentralized marketplace experience, several new features could improve its user experience.

One of the features present in other traditional marketplace platforms such as eBay or Subito.it is price negotiation. The current DecentMarkt logic doesn't allow item prices to be negotiated between the two parties but including it would create a more interactive trading environment.

Another feature from which the platform would benefit is an auction system for certain items of high value or rarity. Additionally, the auction system presents a great use case for underlying blockchain infrastructure since all submitted bids would be transparent and verifiable, leading to greater trust in the bidding process.

One of the features missing in other decentralized marketplace solutions are content moderators. For one platform to have a wide user base, it is necessary to have a mechanism that would prevent illegal or fraudulent items from being sold. Certain moderations are necessary to enhance security and help maintain a safe and trustworthy marketplace. The implementation of such a system is left for further research, as it requires a careful balance between level of decentralization and enforcement of content moderation.

Bibliography

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] L. Lamport, R. Shostak and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems,* p. 382–401, 1982.

[3] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," in *Advances in Cryptology — CRYPTO '92, Lecture Notes in Computer Science*, Springer, 1993, p. 139–147.

[4] "Bitcoin Energy Consumption Index," Digiconomist, [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption.

[5] "Bitcoin," [Online]. Available: https://chainspect.app/chain/bitcoin.

[6] "Proof-of-Stake (PoS)," Ethereum.org, [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[7] M. Laboure, "I could potentially see Bitcoin to become the 21st century gold," DeutscheBank, [Online]. Available: https://www.db.com/what-next/digital-disruption/dossier-payments/i-could-potentially-see-bitcoin-to-become-the-21st-century-gold.

[8] "Turing Complete," UEEx Blog, [Online]. Available: https://blog.ueex.com/crypto-terms/turing-complete/.

[9] "The Merge," Ethereum.org, [Online]. Available: https://ethereum.org/en/roadmap/merge/.

[10] "Ethereum's Strategic Vision: Embracing The 'World Computer' Concept," Binance, [Online]. Available: https://www.binance.com/en/square/post/12-08-2024-ethereum-s-strategic-vision-embracing-the-world-computer-concept-17271205015561.

[11] "The DAO," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/The_DAO.

[12] "Top Decentralized Finance (DeFi) Coins By Market Cap," Forbes, [Online]. Available: https://www.forbes.com/digital-assets/categories/decentralized-finance-defi-crypto/.

[13] D. N. Forbes, "Social-media firms lack adequate privacy controls, FTC report says," The Wall Street Journal, 12 2023. [Online]. Available: https://www.wsj.com/business/telecom/social-media-cos-lack-adequate-privacy-controls-ftc-report-says-3f5b68db.

[14] P. C. Earle, M. Gulker and E. P. Stringham, "Decentralized marketplaces with privately enforced contracts: A case study of OpenBazaar," *J. Private Enterp.*, vol. 37, no. 4, pp. 43-59, 2022.

[15] OpenBazaar, "Escrow Smart Contract Specification in OpenBazaar," Medium, [Online]. Available: https://medium.com/openbazaarproject/escrow-smart-contract-specification-in-openbazaar-94c0937822c3.

[16] OpenBazaar, "OpenBazaar Tokens and Smart Contracts," Medium, [Online]. Available: https://medium.com/openbazaarproject/openbazaar-tokens-and-smart-contracts-ce0e82b6dc08.

[17] "Ricardian contracts: Definition, components, and examples," 101blockchains, [Online]. Available: https://101blockchains.com/ricardian-contracts/.

[18] OpenBazaar, "OpenBazaar Roadmap," Medium, [Online]. Available: https://medium.com/@therealopenbazaar/openbazaar-roadmap-46b9e774f319.

[19] "In-depth presentation of the Particl project," Particl, [Online]. Available: https://particl.news/in-depth-presentation-of-the-particl-project-b3a41923a07e/.

[20] "Particl marketplace explained," Particl, [Online]. Available: https://academy.particl.io/en/latest/particl-marketplace/marketplace_explained.html.

[21] "Particl marketplace escrow," Particl, [Online]. Available: https://academy.particl.io/en/latest/particl-marketplace/marketplace_escrow.html.

[22] "Origin Protocol," IQ.wiki, [Online]. Available: https://iq.wiki/wiki/origin-protocol.

[23] C. Maher, "Origin + Spin & Layer Protocol: Scooters and Reputation on the Blockchain," Origin Protocol, [Online]. Available: https://blog.originprotocol.com/origin-spin-pin-protocol-scooters-and-reputation-on-the-blockchain-737d4532fd44.

[24] "Introducing the Origin Marketplace Creator: Instant peer-to-peer marketplaces," Origin Protocol, [Online]. Available: https://blog.originprotocol.com/introducing-the-origin-marketplace-creator-instant-peer-to-peer-marketplaces-c7f58bb576c4.

[25] "Layer 2 Roadmap," Origin Protocol, [Online]. Available: https://docs.originprotocol.com/protocol/super-oeth/layer-2-roadmap.

[26] "Stablecoin," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Stablecoin.

[27] "Cryptocurrency wallet," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Cryptocurrency_wallet.

[28] "Escrow," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Escrow.

[29] "Proxies," OpenZeppelin, [Online]. Available: https://docs.openzeppelin.com/upgrades-plugins/proxies.

[30] "About The Graph," The Graph, [Online]. Available: https://thegraph.com/docs/en/about/.

[31] "Firebase," Google, [Online]. Available: https://firebase.google.com/.

[32] "How IPFS Works," IPFS, [Online]. Available: https://docs.ipfs.tech/concepts/how-ipfs-works/.

[33] "Getting Started with the Pinata Web3 SDK," Pinata, [Online]. Available: https://docs.pinata.cloud/web3/sdk/getting-started.

[34] "Next.js Documentation," Next.js, [Online]. Available: https://nextjs.org/docs.

[35] "Vercel Documentation," Vercel, [Online]. Available: https://vercel.com/docs.

[36] "Hardhat Documentation," Hardhat, [Online]. Available: https://hardhat.org/docs.

[37] "Unit testing," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Unit_testing.

[38] "Polygon," Chainspect, [Online]. Available: https://chainspect.app/chain/polygon.

[39] "Polygon Documentation," Polygon, [Online]. Available: https://docs.polygon.technology/.

A Appendix A: Smart Contracts

This appendix contains the code of Users, Marketplace and Escrow smart contracts written in Solidity programming language.

A.1. Marketplace contract

```
contract Marketplace is Initializable, OwnableUpgradeable {

    enum ItemStatus {
        LISTED,
        BOUGHT,
        DELETED
    }

    enum Condition {
        NEW,
        LIKE_NEW,
        EXCELLENT,
        GOOD,
        DAMAGED
    }

    struct Item {
        uint256 id;
        address seller;
        uint256 price;
        string currency;
        string description;
        string title;
        string[] photosIPFSHashes;
        ItemStatus itemStatus;
        Condition condition;
        string category;
        string subcategory;
        string country;
        bool isGift;
    }

    uint8 constant public MAX_PHOTO_LIMIT = 3;
```

```
    uint256 itemCount;
    mapping(uint256 => Item) private items; //mapping of id to Item
    mapping(string => string[]) private categories; // mapping category
name to list of subcategories
    mapping(string => address) public supportedTokens; // mapping
tokenName ("POL", "USDC"...) to token contract address

    event ItemListed(uint256 indexed id, address indexed seller, string
title, string description, uint256 price, string currency, string[]
photosIPFSHashes, Condition condition,
        string category, string subcategory, string country, bool
isGift);
    event ItemUpdated(uint256 indexed id, address indexed seller,
string title, string description, uint256 price, string currency,
string[] photosIPFSHashes, Condition condition,
        string category, string subcategory, string country, bool
isGift);
    event ItemBought(uint256 indexed id, address indexed seller,
address indexed buyer);
    event ItemDeleted(uint256 indexed id, address indexed seller);

    IEscrow public escrowContract;
    IUsers public usersContract;

    function initialize(address initialOwner) public initializer {
        __Ownable_init(initialOwner); // Initialize OwnableUpgradeable
        transferOwnership(initialOwner); // Set the owner explicitly

        // Initialize categories
        categories["Electronics"] = ["Mobile Phones", "Laptops",
"Cameras", "Headphones", "Smart Watches", "Other"];
        categories["Clothing"] = ["Men's Apparel", "Women's Apparel",
"Kids' Apparel", "Accessories", "Man's shoes", "Women's shoes",
"Other"];
        categories["Furniture"] = ["Living Room", "Bedroom", "Office",
"Outdoor", "Other"];
        categories["Toys"] = ["Educational", "Outdoor", "Action
Figures", "Building Sets", "Other"];
        categories["Books"] = ["Fiction", "Non-Fiction", "Children's
Books", "Textbooks", "Other"];
        categories["Sports"] = ["Equipment", "Apparel", "Footwear",
"Accessories", "Other"];
        categories["Home Appliances"] = ["Kitchen", "Laundry", "Heating
& Cooling", "Other"];
        categories["Beauty"] = ["Skincare", "Makeup", "Fragrances",
"Hair Care", "Other"];
```

```
        categories["Automotive"] = ["Parts", "Accessories", "Tools",
"Other"];
        categories["Collectibles"] = ["Coins", "Stamps", "Trading
Cards", "Art", "Other"];

        // Initialize supported tokens
        supportedTokens["POL"] = address(this);
    }

    modifier isListed(uint256 id) {
        if (items[id].price < 0) {
            revert ItemNotListed(id);
        }
        else if (items[id].itemStatus != ItemStatus.LISTED) {
            revert ItemNotListed(id);
        }
        _;
    }

    modifier belongsToSeller(address sellerAddress, uint256 id) {
        if (items[id].seller != sellerAddress) {
            revert ItemNotBelongsToSeller(sellerAddress, id);
        }
        _;
    }

    modifier notSeller(address sellerAddress, address buyerAddress) {
        if (buyerAddress == sellerAddress) {
            revert SellerCannotBuyItsItem(sellerAddress);
        }
        _;
    }

    modifier mustBeModerator(address moderator) {
        if (!usersContract.isModerator(moderator)) {
            revert MustBeModerator(moderator);
        }
        _;
    }

    modifier notGift(uint256 id) {
        if (items[id].isGift) {
            revert MustNotBeGift();
        }
        _;
    }

    modifier priceMustBeAboveOrEqualZero(uint256 price) {
```

```
        if (price < 0) {
            revert PriceMustBeAboveZero();
        }
        _;
    }

    modifier numberOfPhotosMustBeBelowLimit(string[] memory
photosIPFSHashes) {
        if (photosIPFSHashes.length > MAX_PHOTO_LIMIT) {
            revert PhotoLimitExceeded();
        }
        _;
    }

    modifier moderatorCantBeBuyerOrSeller(address _moderator, address
sellerAddress) {
        if (_moderator == sellerAddress || _moderator == msg.sender) {
            revert ModeratorCantBeBuyerOrSeller();
        }
        _;
    }

    modifier isValidCategoryAndSubcategory(string memory category,
string memory subcategory) {
        bool isOkay = false;
        for (uint i = 0; i < categories[category].length; i++) {
            if (keccak256(abi.encodePacked(categories[category][i])) ==
keccak256(abi.encodePacked(subcategory))) {
                isOkay =true;
            }
        }
        if (!isOkay) {
            revert NotValidCategoryAndSubcategory();
        }
        _;
    }


    modifier isValidIPFSHashes(string[] memory photosIPFSHashes) {
        for (uint i = 0; i < photosIPFSHashes.length; i++) {
            if (!isIPFSHash(photosIPFSHashes[i])) {
                revert NotIPFSHash(photosIPFSHashes[i]);
            }
        }
        _;
    }

    modifier supportedToken(string memory tokenName) {
```

```
        if (supportedTokens[tokenName] == address(0)) {
            revert TokenNotSupported();
        }
        _;
    }

    modifier userRegistered() {
        if (!usersContract.isRegisteredUser(msg.sender)) {
            revert UserNotRegistered();
        }
        _;
    }

    function setUsersContractAddress(address _usersContractAddress)
external
        onlyOwner {
        usersContract = IUsers(_usersContractAddress);
    }

    function setEscrowContractAddress(address _escrowContractAddress)
external
        onlyOwner {
        escrowContract = IEscrow(_escrowContractAddress);
    }

    function addSupportedToken(string memory tokenName, address
tokenAddress) external onlyOwner {
        supportedTokens[tokenName] = tokenAddress;
    }

    function listNewItem(
        Item memory item
        )
        external
        priceMustBeAboveOrEqualZero(item.price)
        numberOfPhotosMustBeBelowLimit(item.photosIPFSHashes)
        isValidCategoryAndSubcategory(item.category, item.subcategory)
        isValidIPFSHashes(item.photosIPFSHashes)
        supportedToken(item.currency)
        userRegistered()
        returns (uint256)
        {
        uint256 id = finalizeListItem(item);
        return id;
    }

    function finalizeListItem(
        Item memory item
```

```
    )   internal returns (uint256) {

        itemCount++;
        uint256 id = createHash(itemCount, msg.sender);
        if (item.isGift)
            item.price = 0;

        items[id] = Item(id, msg.sender, item.price,
item.currency,item.description, item.title, item.photosIPFSHashes,
ItemStatus.LISTED, item.condition, item.category, item.subcategory,
item.country, item.isGift);
        emit ItemListed(id, msg.sender, item.title, item.description,
item.price, item.currency, item.photosIPFSHashes, item.condition,
item.category, item.subcategory, item.country, item.isGift);
        return id;
    }

    function updateItem(Item memory item) external
        belongsToSeller(msg.sender, item.id)
        priceMustBeAboveOrEqualZero(item.price)
        numberOfPhotosMustBeBelowLimit(item.photosIPFSHashes)
        isListed(item.id)
        isValidIPFSHashes(item.photosIPFSHashes)
        isValidCategoryAndSubcategory(item.category, item.subcategory)
        supportedToken(item.currency)
        {
        if (item.isGift)
            item.price = 0;

        finalizeUpdateItem(item);
    }

    function finalizeUpdateItem(Item memory item) internal {
        items[item.id] = Item(item.id, msg.sender, item.price,
item.currency,item.description, item.title,
item.photosIPFSHashes,ItemStatus.LISTED, item.condition, item.category,
item.subcategory, item.country, item.isGift);
        emit ItemUpdated(item.id, msg.sender, item.title,
item.description, item.price, item.currency, item.photosIPFSHashes,
item.condition, item.category, item.subcategory, item.country,
item.isGift);
    }

    function isIPFSHash(string memory hash) private pure returns (bool)
{
        bytes memory hashBytes = bytes(hash);
        if (hashBytes.length != 46) {
            return false;
```

```
        }
        if (hashBytes[0] != 0x51 || hashBytes[1] != 0x6D) {
            return false; // "Qm" prefix
        }
        for (uint i = 2; i < hashBytes.length; i++) {
            bytes1 char = hashBytes[i];
            if (!(char >= 0x30 && char <= 0x39) && // 0-9
                !(char >= 0x41 && char <= 0x5A) && // A-Z
                !(char >= 0x61 && char <= 0x7A)) { // a-z
                return false;
            }
        }
        return true;
    }

    function createHash(uint256 id, address addr) private pure returns
(uint256) {
        bytes32 hashInput = keccak256(abi.encodePacked(id, addr));
        uint256 idHash = uint256(hashInput); // Convert the
concatenated bytes32 hash to uint256
        return idHash;
    }

    function deleteItem(uint256 id) isListed(id) external
        belongsToSeller(msg.sender, id)  {
        items[id].itemStatus = ItemStatus.DELETED;
        emit ItemDeleted(id, msg.sender);
    }

    function buyItem(uint256 id, address _moderator) external payable
        isListed(id)
        notSeller(items[id].seller, msg.sender)
        mustBeModerator(_moderator)
        notGift(id)
        userRegistered()
        {
        uint8 moderatorFee =
usersContract.getProfile(_moderator).moderatorFee;
        uint256 totalCost = items[id].price + (items[id].price *
moderatorFee) / 100;

        if (keccak256(abi.encodePacked(items[id].currency)) ==
keccak256(abi.encodePacked("POL"))) {
            require(msg.value >= totalCost, "Incorrect POL amount");
        }
        else {
            // check if buyer(sender) has allowed marketplace contract
to transfer funds
```

```
            require(IERC20(supportedTokens[items[id].currency]).allowan
ce(msg.sender, address(this)) >= totalCost, "Insufficient allowance for
marketplace contract");

            // sender must have enough token balance on his address
            require(IERC20(supportedTokens[items[id].currency]).balance
Of(msg.sender) >= totalCost, "Insufficient token balance");
        }
        finalizeBuyItem(id, _moderator, moderatorFee, totalCost);
    }

    function finalizeBuyItem(uint256 id, address _moderator, uint8
moderatorFee, uint256 totalCost) private
        moderatorCantBeBuyerOrSeller(_moderator, items[id].seller){

        // transfer tokens to escrow address
        if (keccak256(abi.encodePacked(items[id].currency)) !=
keccak256(abi.encodePacked("POL"))) {
            bool success =
IERC20(supportedTokens[items[id].currency]).transferFrom(msg.sender,
address(escrowContract), totalCost);
            require(success, "Token transfer failed");
        }

        // create transaction
        try escrowContract.createTransaction{value: msg.value}(id,
items[id].seller,
msg.sender,_moderator,items[id].price,  items[id].currency,
moderatorFee) {
            items[id].itemStatus = ItemStatus.BOUGHT;
            emit ItemBought(id, items[id].seller, msg.sender);
        }
        catch {
            revert("Transaction creation failed");
        }
    }

    function buyItemWithoutModerator(uint256 id) external payable
        isListed(id)
        notSeller(items[id].seller, msg.sender)
        userRegistered()
        {

        if (keccak256(abi.encodePacked(items[id].currency)) ==
keccak256(abi.encodePacked("POL"))) {
            require(msg.value >= items[id].price, "Incorrect POL
amount");
        }
```

```
        else {
            // check if buyer(sender) has allowed escrow contract to
transfer funds
            require(IERC20(supportedTokens[items[id].currency]).allowan
ce(msg.sender, address(escrowContract)) >= items[id].price,
"Insufficient allowance for escrow contract");

            // sender must have enough token balance on his address
            require(IERC20(supportedTokens[items[id].currency]).balance
Of(msg.sender) >= items[id].price, "Insufficient token balance");
        }
        finalizeBuyItemWithoutModerator(id);
    }

    function finalizeBuyItemWithoutModerator(uint256 id) private {
        try escrowContract.createTransactionWithoutModerator{value:
msg.value}(id, items[id].seller, msg.sender,items[id].price,
items[id].currency) {
            items[id].itemStatus = ItemStatus.BOUGHT;
            emit ItemBought(id, items[id].seller, msg.sender);
        }
        catch {
            revert("Transaction creation failed");
        }
    }

    function getItemCount() public view returns (uint256) {
        return itemCount;
    }
}
```

A.2. Escrow contract

```
contract Escrow is Initializable, OwnableUpgradeable {

    struct Moderator {
        address moderator;
        uint256 fee;
    }

    struct Transaction {
        uint256 itemId;
        address seller;
        address moderator;
```

```
        address buyer;
        uint256 price;
        string currency;
        uint8 moderatorFee;
        bool buyerApproved;
        bool sellerApproved;
        bool disputed;
        bool disputedBySeller;
        bool disputedByBuyer;
        bool isCompleted;
        uint256 creationTime;
    }

    IMarketplace public marketplaceContract;
    IUsers public usersContract;

    mapping(uint256 => Transaction) private transactions; // item id to
transaction mapping -> 1:1 itemId Transaction
    mapping(string => address) public supportedTokens;

    event TransactionCreated(uint256 indexed itemId, address indexed
buyer, address indexed seller,
        address moderator, uint256 price, string currency, uint8
moderatorFee, uint256 creationTime);
    event TransactionCreatedWithoutModerator(uint256 indexed itemId,
address indexed buyer, address indexed seller,
        uint256 price, string currency, uint256 creationTime);
    event TransactionApproved(uint256 indexed itemId, address
approver);
    event TransactionCompleted(uint256 indexed itemId);
    event TransactionCompletedByModerator(uint256 indexed itemId, uint8
buyerPercentage, uint8 sellerPercentage);
    event TransactionDisputed(uint256 indexed itemId, address
disputer);

    modifier txExists(uint256 id) {
        Transaction memory item = transactions[id];
        if (item.price > 0) {
            revert TxExists(id);
        }
        _;
    }

    modifier onlyMarketplaceCanCall() {
        if (msg.sender != address(marketplaceContract)) {
            revert OnlyMarketplaceContractCanCall();
        }
        _;
```

```
    }

    modifier onlyUsersCanCall() {
        if (msg.sender != address(usersContract)) {
            revert OnlyUsersContractCanCall();
        }
        _;
    }

    function initialize(address initialOwner) public initializer {
        __Ownable_init(initialOwner); // Initialize OwnableUpgradeable
        transferOwnership(initialOwner); // Set the owner explicitly

        // Initialize supported tokens
        supportedTokens["POL"] = address(this);
    }

    function addSupportedToken(string memory tokenName, address
tokenAddress) external onlyOwner {
        supportedTokens[tokenName] = tokenAddress;
    }

    function setMarketplaceContractAddress(address
_marketplaceContractAddress) external onlyOwner {
        marketplaceContract =
IMarketplace(_marketplaceContractAddress);
    }

    function setUsersContractAddress(address _usersContractAddress)
external onlyOwner {
        usersContract = IUsers(_usersContractAddress);
    }

    modifier onlyModerator(uint256 itemId) {
        if (msg.sender != transactions[itemId].moderator) {
            revert OnlyModerator();
        }
        _;
    }

    modifier onlySeller(uint256 itemId) {
        if (msg.sender != transactions[itemId].seller) {
            revert OnlySeller();
        }
        _;
    }

    modifier onlyBuyer(uint256 itemId) {
```

```
        if (msg.sender != transactions[itemId].buyer) {
            revert OnlyBuyer();
        }
        _;
    }

    modifier onlyBuyerOrSeller(uint256 itemId) {
        if (msg.sender != transactions[itemId].seller && msg.sender !=
transactions[itemId].buyer) {
            revert OnlyBuyerOrSeller();
        }
        _;
    }

    modifier notCompleted(uint256 itemId) {
        if (transactions[itemId].isCompleted) {
            revert TxCantBeCompleted();
        }
        _;
    }

    modifier correctValueDistribution(uint256 itemId, uint8
percentageSeller, uint8 percentageBuyer) {
        if (percentageSeller + percentageBuyer != 100) {
            revert ValueDistributionNotCorrect();
        }
        _;
    }

    modifier mustBeDisputed(uint256 itemId) {
        if (!transactions[itemId].disputed) {
            revert MustBeDisputed();
        }
        _;
    }

    function createTransaction(
        uint256 _itemId,
        address _seller,
        address _buyer,
        address _moderator,
        uint256 _price,
        string memory _currency,
        uint8 _moderatorFee
    ) txExists(_itemId) onlyMarketplaceCanCall() external payable {

        transactions[_itemId] = Transaction({
            itemId: _itemId,
```

```
            seller: _seller,
            buyer: _buyer,
            moderator: _moderator,
            price: _price,
            currency: _currency,
            moderatorFee: _moderatorFee,
            buyerApproved: false,
            sellerApproved: false,
            disputed: false,
            disputedBySeller: false,
            disputedByBuyer: false,
            isCompleted: false,
            creationTime: block.timestamp
        });

        emit TransactionCreated(_itemId, _buyer, _seller, _moderator,
_price, _currency, _moderatorFee, block.timestamp);
    }

    function createTransactionWithoutModerator(
        uint256 _itemId,
        address _seller,
        address _buyer,
        uint256 _price,
        string memory _currency
    ) txExists(_itemId) onlyMarketplaceCanCall() external payable {

        transactions[_itemId] = Transaction({
            itemId: _itemId,
            seller: _seller,
            buyer: _buyer,
            moderator: address(0),
            price: _price,
            currency: _currency,
            moderatorFee: 0,
            buyerApproved: true,
            sellerApproved: true,
            disputed: false,
            disputedBySeller: false,
            disputedByBuyer: false,
            isCompleted: true,
            creationTime: block.timestamp
        });
        emit TransactionCreatedWithoutModerator(_itemId, _buyer,
_seller, _price, _currency, block.timestamp);

        finalizeTransactionWithoutModerator(_itemId);
    }
```

```
    function approve(uint256 _itemId) external
        onlyBuyerOrSeller(_itemId)
        notCompleted(_itemId) {

        Transaction storage transaction = transactions[_itemId];

        if (msg.sender == transaction.seller)
            transaction.sellerApproved = true;
        else
            transaction.buyerApproved = true;

        emit TransactionApproved(_itemId, msg.sender);

        finalizeTransaction(_itemId);
    }

    function raiseDispute(uint256 _itemId) external
        onlyBuyerOrSeller(_itemId)
        notCompleted(_itemId) {

        Transaction storage transaction = transactions[_itemId];
        transaction.disputed = true;
        if (msg.sender == transaction.seller)
            transaction.disputedBySeller = true;
        else
            transaction.disputedByBuyer = true;

        emit TransactionDisputed(_itemId, msg.sender);
    }

    function finalizeTransaction(uint256 _itemId) internal {
        Transaction storage transaction = transactions[_itemId];

        require(!transaction.isCompleted, "Transaction already
completed");

        if (transaction.buyerApproved && transaction.sellerApproved) {
            transaction.isCompleted = true;

            uint256 moderatorFeeAmount = (transaction.price *
transaction.moderatorFee) / 100;
            uint256 sellerAmount = transaction.price;

            if (keccak256(abi.encodePacked(transaction.currency)) ==
keccak256(abi.encodePacked("POL"))) {
                // Transfer to moderator their cut
```

```
                (bool successModerator, ) =
transaction.moderator.call{value: moderatorFeeAmount}("");
                require(successModerator, "Transfer to moderator
failed");

                // Transfer remaining amount to seller
                (bool successSeller, ) = transaction.seller.call{value:
sellerAmount}("");
                require(successSeller, "Transfer to seller failed");
            }
            else {
                address tokenAddress =
supportedTokens[transaction.currency];

                IERC20 token = IERC20(tokenAddress);

                bool successModerator =
token.transfer(transaction.moderator, moderatorFeeAmount);
                require(successModerator, "Token transfer to moderator
failed");

                bool successSeller = token.transfer(transaction.seller,
sellerAmount);
                require(successSeller, "Token transfer to seller
failed");
            }

            emit TransactionCompleted(_itemId);
        }
    }

    function finalizeTransactionWithoutModerator(uint256 _itemId)
internal
    {
        Transaction storage transaction = transactions[_itemId];

        // Transfer remaining amount to seller
        if (keccak256(abi.encodePacked(transaction.currency)) ==
keccak256(abi.encodePacked("POL"))) {
            (bool successSeller, ) = transaction.seller.call{value:
transaction.price}("");
            require(successSeller, "Transfer to seller failed");
        } else {
            address tokenAddress =
supportedTokens[transaction.currency];

            IERC20 token = IERC20(tokenAddress);
```

```
            bool successSeller = token.transferFrom(transaction.buyer,
transaction.seller, transaction.price);
            require(successSeller, "Token transfer to seller failed");
        }

        emit TransactionCompleted(_itemId);
    }


    function finalizeTransactionByModerator(uint256 _itemId, uint8
percentageSeller, uint8 percentageBuyer) external payable
        onlyModerator(_itemId)
        notCompleted(_itemId)
        correctValueDistribution(_itemId, percentageBuyer,
percentageSeller)
        mustBeDisputed(_itemId) {

        finalizePaymentsByModerator(_itemId, percentageSeller,
percentageBuyer);
    }

    function finalizePaymentsByModerator(uint256 _itemId, uint8
percentageSeller, uint8 percentageBuyer) internal {
        Transaction storage transaction = transactions[_itemId];

        transaction.isCompleted = true;

        uint256 moderatorFeeAmount = (transaction.price *
transaction.moderatorFee) / 100;

        // full item price is distributed between seller and buyer
        uint256 sellerAmount = (transaction.price * percentageSeller) /
100;
        uint256 buyerAmount = (transaction.price * percentageBuyer) /
100;

        if (keccak256(abi.encodePacked(transaction.currency)) ==
keccak256(abi.encodePacked("POL"))) {
            (bool successModerator, ) =
transaction.moderator.call{value: moderatorFeeAmount}("");
            require(successModerator, "Transfer to moderator failed");

            (bool successSeller, ) = transaction.seller.call{value:
sellerAmount}("");
            require(successSeller, "Transfer to seller failed");

            (bool successBuyer, ) = transaction.buyer.call{value:
buyerAmount}("");
```

```
            require(successBuyer, "Transfer to buyer failed");
        } else {
            address tokenAddress =
supportedTokens[transaction.currency];

            IERC20 token = IERC20(tokenAddress);

            bool successModerator =
token.transfer(transaction.moderator, moderatorFeeAmount);
            require(successModerator, "Token transfer to moderator
failed");

            bool successSeller = token.transfer(transaction.seller,
sellerAmount);
            require(successSeller, "Token transfer to seller failed");

            bool successBuyer = token.transfer(transaction.buyer,
buyerAmount);
            require(successBuyer, "Token transfer to buyer failed");
        }

        emit TransactionCompletedByModerator(_itemId, percentageBuyer,
percentageSeller);
    }

    function isTransactionReadyForReview(uint256 _itemId, address from,
address to) external view onlyUsersCanCall() returns (bool) {
        Transaction storage transaction = transactions[_itemId];

        if ((from == transaction.seller || from == transaction.buyer ||
from == transaction.moderator) &&
            (to == transaction.seller || to == transaction.buyer || to
== transaction.moderator) && (from != to) &&
            transaction.isCompleted)
            return true;

        return false;
    }
}
```

A.3. Users contract

```
contract Users is Initializable, OwnableUpgradeable {

    function initialize(address initialOwner) public initializer {
        __Ownable_init(initialOwner);
        transferOwnership(initialOwner);
```

```
    }

    struct Review {
        string content;
        uint8 rating; //1-5
        uint256 itemId;
        address from;
    }

    struct UserProfile {
        address userAddress;
        string username;
        string firstName;
        string lastName;
        string country;
        string description;
        string email;
        string avatarHash;
        bool isModerator;
        bool exists;
        uint8 moderatorFee;
    }

    mapping(address => UserProfile) public userProfiles;
    mapping(address => Review[]) public reviews;

    mapping(string => bool) private usernameExists;

    IEscrow public escrowContract;


    uint8 public maxModeratorFee = 10;

    event UserRegistered(address indexed userAddress, string username,
string firstName,
        string lastName, string country, string description, string
email, string avatarHash, bool isModerator, uint8 moderatorFee);
    event UserUpdated(address indexed userAddress, string username,
string firstName,
        string lastName, string country, string description, string
email, string avatarHash, bool isModerator, uint8 moderatorFee);
    event ReviewCreated(address indexed from, address indexed to,
string content, uint8 rating, uint256 itemId);


    modifier userMustExist(address userAddress) {
        if (!userProfiles[userAddress].exists) {
            revert UserDoesNotExist(userAddress);
```

```
        }
        _;
    }

    modifier userMustNotExist(address userAddress) {
        if (userProfiles[userAddress].exists) {
            revert UserAlreadyExists(userAddress);
        }
        _;
    }

    modifier usernameMustNotExist(string memory username) {
        if (usernameExists[username]) {
            revert UsernameExists(username);
        }
        _;
    }

    modifier moderatorFeeInRange(uint8 moderatorFee) {
        if (moderatorFee < 0 || moderatorFee > maxModeratorFee) {
            revert ModeratorFeeLimitsNotRespected(moderatorFee);
        }
        _;
    }

    function setEscrowContractAddress(address _escrowContractAddress)
external
        onlyOwner {
        escrowContract = IEscrow(_escrowContractAddress);
    }

    function setMaxModeratorFee(uint8 newFee) public onlyOwner {
        require(newFee <= 10, "Fee cannot exceed 10%");
        maxModeratorFee = newFee;
    }

    function createProfile(string memory _username, string memory
_firstName, string memory _lastName, string memory _country,
        string memory _description, string memory _email, string memory
_avatarHash, bool _isModerator, uint8 _moderatorFee)
        userMustNotExist(msg.sender) usernameMustNotExist(_username)
moderatorFeeInRange(_moderatorFee) external {

        uint8 fee;
        if (!_isModerator)
            fee = 0;
        else
            fee = _moderatorFee;
```

```
        userProfiles[msg.sender] = UserProfile({
            userAddress: msg.sender,
            username: _username,
            firstName: _firstName,
            lastName: _lastName,
            country: _country,
            description: _description,
            email: _email,
            avatarHash: _avatarHash,
            isModerator: _isModerator,
            exists: true,
            moderatorFee: fee
        });
        usernameExists[_username] = true;

        emit UserRegistered(msg.sender, _username, _firstName,
_lastName, _country, _description, _email, _avatarHash, _isModerator,
fee);
    }

    function updateProfile(string memory _username, string memory
_firstName, string memory _lastName, string memory _country,
        string memory _description, string memory _email, string memory
_avatarHash, bool _isModerator, uint8 _moderatorFee)
        userMustExist(msg.sender) external {

        UserProfile memory oldUser = userProfiles[msg.sender];
        if (!compareStrings(oldUser.username, _username)) { //username
is updated
            if (usernameExists[_username])
                revert UsernameExists(_username);

            delete usernameExists[oldUser.username];
            usernameExists[_username] = true;
        }

        uint8 fee;
        if (!_isModerator) {
            fee = 0;
        }
        else {
            fee = _moderatorFee;
        }

        userProfiles[msg.sender] = UserProfile({
            userAddress: msg.sender,
            username: _username,
```

```
            firstName: _firstName,
            lastName: _lastName,
            country: _country,
            description: _description,
            email: _email,
            avatarHash: _avatarHash,
            isModerator: _isModerator,
            exists: true,
            moderatorFee: fee
        });

        emit UserUpdated(msg.sender, _username, _firstName, _lastName,
_country, _description, _email, _avatarHash, _isModerator, fee);
    }

    function isRegisteredUser(address _user) external view returns
(bool) {
        return userProfiles[_user].exists;
    }

    function isModerator(address _user) external view returns (bool) {
        return userProfiles[_user].exists &&
userProfiles[_user].isModerator;
    }

    function getProfile(address _user) external view returns
(UserProfile memory) {
        return userProfiles[_user];
    }


    function compareStrings(string memory _a, string memory _b) private
pure returns(bool) {
        return keccak256(abi.encodePacked(_a)) ==
keccak256(abi.encodePacked(_b));
    }

    function isAlreadyReviewed(Review[] storage userReviews, uint256
itemId) internal view returns (bool) {
        for (uint256 i = 0; i < userReviews.length; i++) {
            if (userReviews[i].itemId == itemId && userReviews[i].from
== msg.sender) {
                return true;
            }
        }
        return false;
    }
```

```
    function createReview(address toWhom, uint256 itemId, uint8 rating,
string memory content) external {
        if (userProfiles[msg.sender].exists &&
userProfiles[toWhom].exists &&
            escrowContract.isTransactionReadyForReview(itemId,
msg.sender, toWhom)) {
            Review[] storage userReviews = reviews[toWhom];
            if (!isAlreadyReviewed(userReviews, itemId) && rating >= 1
&& rating <= 5) {
                userReviews.push(Review(content, rating, itemId,
msg.sender));
                emit ReviewCreated(msg.sender, toWhom, content, rating,
itemId);
            }
            else
                revert AlreadyReviewed();
        }
    }
}
```

B Appendix B: The Graph data structures

This appendix contains mappings of smart contract event data to the subgraph that will be used by The Graph indexer to extract data from the blockchain (Polygon Amoy Testnet), process them via assigned function handlers and store them in its database. In detail it provides an overview of the following data structures used for querying the database via GraphQL - User, Item, Transaction and Review.

B.1. Item

```
type Item @entity {
  id: String!
  buyer: Bytes! # Address 0x0000... if no one has bought it yet
  seller: Bytes!
  title: String!
  description: String!
  price: BigInt!
  currency: String!
  photosIPFSHashes: [String!]!
  itemStatus: ItemStatus!
  condition: Condition!
  category: String!
  subcategory: String!
  country: String!
  isGift: Boolean!
  blockNumber: BigInt!
  blockTimestamp: BigInt!
  transactionHash: Bytes!
}
```

B.2. Transaction

```
type Transaction @entity {
  id: String!
  itemId: BigInt!
  buyer: Bytes! # address
```

```
  seller: Bytes! # address
  moderator: Bytes! # address
  price: BigInt!
  currency: String!
  moderatorFee: Int!
  buyerApproved: Boolean!
  sellerApproved: Boolean!
  disputed: Boolean!
  disputedBySeller: Boolean!
  disputedByBuyer: Boolean!
  isCompleted: Boolean!
  creationTime: BigInt!
  blockNumber: BigInt!
  blockTimestamp: BigInt!
  transactionHash: Bytes!
  buyerPercentage: Int!
  sellerPercentage: Int!
}
```

B.3. User

```
type User @entity {
  id: Bytes!
  userAddress: Bytes! # address
  username: String!
  firstName: String!
  lastName: String!
  country: String!
  description: String!
  email: String!
  avatarHash: String!
  isModerator: Boolean!
  moderatorFee: Int!
  blockNumber: BigInt!
  blockTimestamp: BigInt!
  transactionHash: Bytes!
  isActive: Boolean!

  reviews: [Review!]! @derivedFrom(field: "user")
}
```

B.4. Review

```
type Review @entity {
  id: Bytes!
  from: Bytes! # userAddress of the reviewer
```

```
  content: String!
  rating: Int! # rating (1 to 5)
  itemId: BigInt! # reference to the item being reviewed
  user: User!
  blockNumber: BigInt!
  blockTimestamp: BigInt!
  transactionHash: Bytes!
}
```

List of Figures

List of Tables

Acknowledgments

Firstly, I would like to express my gratitude to my thesis supervisor professor Francesco Bruschi for his guidance, suggestions and feedback. It has been a true pleasure working with him towards the finalization of this thesis.

To my parents, Marko and Dejana, to whom this thesis is dedicated. Their endless support, and tremendous dedication towards my education helped me stay focused and pushed me till the very end. Also, I want to thank them for their trust in me and my vision, which encouraged me to step beyond my comfort zone.

To Dejan, who was next to me from the very first exam in my bachelor's till the very last one here. Who listened for many years to my nagging, stress over each exam yet was ready to assure me every single time. Who moved here to support me in achieving my goals, and, despite many challenges, showed unfailing support throughout this journey. I will always be truly grateful for it.

To my friends, both here and in Serbia, that kept me motivated and shared this experience with me. Milica, for her dad's jokes and for making it feel a bit more like home. Burak and Nehir, for their warmth and for always making me feel like a part of the group. To Ivana, who was both here and in Serbia whenever I needed her. To Shakiba, for her insider info and for being always there to listen. To Juliana, for her emotional support. To Giorgio and Matilde, who I met on the first day at this very place and who were there to calm me down and encourage me. To Vučka for her calming energy. To Željko for keeping me grounded. To Nevena, with whom I shared many nights talking and who always inspired me with her achievements. To Marina and Jelena who have seen me in all my phases and supported me regardless.

Lastly, I would like to thank Milan and Polimi and the many other incredible people that I met throughout this journey.