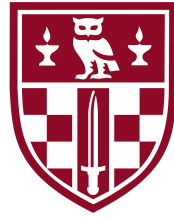


Birkbeck



University of London

BSc Computer Science - Final Year Project Report

Department of Computer Science and Information Systems

Decentralized Web Store Built With IPFS and Blockchain

Author:

Octavian Contis



<https://github.com/octipus/dappStore-eth-ipfs>

Academic Declaration

University of London, 2018.

This report is substantially the result of my own work except where explicitly indicated in the text. I give my permission for it to be submitted to the Plagiarism Detection Service. I have read and understood the sections on plagiarism in the Programme booklet and the University website.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

This project proposes a solution for removing friction cost of transactions in e-commerce, aiming to reduce entities involved in the ecosystem. Ethereum blockchain[1] and smart contracts are used in order to enforce automated functions that enable users to conduct transactions on an open source platform. Blockchain provides immutable proof of transactions between sellers and buyers, along with a third party that can intervene in case of a dispute. The application is accessed through a web interface where users can list, buy or sell products with cryptocurrency. There is no entity in control of the application and once deployed live it will be accessible by anyone with an Ethereum wallet. Media files are stored on IPFS File Sharing[27] system, pursuing the decentralization aspect of the application. I have developed the application in a test environment and plan to extend the functionalities further after receiving suggestions from people that tested the application. Furthermore, I am considering deploying the application on the main Ethereum network, making it available for the wide public.

Glossary

1. *Decentralized* - not governed by any entity, instead is governed by the rules of the protocol.
2. dApp - a decentralized application
3. *Wallet and addresses* - used to manage blockchain transactions.
4. *The Blockchain* - a public ledger that holds all transactions ever occurred.
5. *Mining* - a resource-intensive process used to validate transactions and generate tokens.
6. *Miners* - users that validate transactions and generate tokens.
7. *Cryptocurrency* - a digital currency with cryptographic properties
8. *Digital Signature* - a mathematical technique used to verify the integrity of an electronic document.
9. *Consensus algorithm* - a process used to achieve agreement on a single data value among distributed systems.

Acknowledgements

Firstly I would like to thank my supervisor and tutor, Professor Keith Leonard Mannock for providing me with guidance and advice throughout the project. I am extremely thankful to him for offering invaluable feedback and support during the creation of this project.

I also want to thank my family who has supported me not only during this project but throughout my studies at the Birkbeck University of London.

Finally I would like to thank my good friend Sergiu Octavian for his motivation and encouragement that kept me going during difficult times throughout the project.

Contents

Academic Declaration	1
Abstract	2
Glossary	3
Acknowledgements	4
Contents	5
Chapter 1 - Introduction	7
1.1 Current practices	8
1.2 Alternative	9
1.3 Roadmap	11
Chapter 2 - Background research	12
2.1 Cryptography	12
2.1.1 Hashing	12
2.1.2 Public Key Cryptography	13
2.1.3 Merkle trees	15
2.2 Blocks	16
2.2.1 Block headers	16
2.3 Consensus Algorithms	17
2.3.1 POW - Proof of Work / Mining	18
2.3.2 POS - Proof of Stake / Minting	19
2.4 The Blockchain	19
2.4.1 Early days of Blockchain	19
2.4.2 Blockchain Today	20
2.5 Ethereum Blockchain	22
2.5.1 Ether and Gas	22
2.5.2 dApps	22
2.5.3 Wallets	23
2.5.4 Smart Contracts	24
2.6 IPFS - Interplanetary File System	25
Chapter 3 - Requirements and Analysis	26
3.1 Application Components	26
3.2 Application Functions	27
3.3 Dispute resolutions	28
3.4 Decentralized storage with IPFS	30

3.5 Web Application Interface	31
Chapter 4 - Application Walkthrough	32
4.1 User Journeys	32
4.2 Application interface	33
Chapter 5 - Design and Implementation	38
5.1 Architecture	38
5.1.1 Framework and Libraries	39
5.2 Smart contracts	40
5.2.1 Ecommerce Smart Contract	40
5.2.2 Escrow Smart Contract	43
5.3 Ethereum and IPFS integration	46
Chapter 6 - Testing and Evaluation	48
6.1 Local Testing	48
Chapter 7 - Conclusion	52
7.1 Future work	53
References	55
https://github.com/octipus/dappStore-eth-ipfs	0

Chapter 1 - Introduction

The following project will describe the process of developing a decentralised application (dApp) on the Blockchain using key cryptographic elements and open source protocols. The scope of the project is to provide an alternative solution for the current e-commerce practices.

By using blockchain technology, friction cost of transactions can be cut to a fraction, removing extraneous third parties from the ecosystem. The e-commerce example will be used in order to explore these open source technologies and current practices of distributed systems.

Key project objectives are as follows:

- Setup a local test environment for interacting with the blockchain.
- Develop an E-commerce smart contract enabling various functions for user interaction such as listing, viewing, and buying a product.
- Develop a Multisig Escrow smart contract with a buyer, seller and arbiter as participants.
- Interact with the IPFS protocol to store data (image and text files).
- Utilization of Ethereum blockchain for verifying transactions
- Designing a clean interface for user interaction

This project will not deal with the moderation aspect of the Escrow contract, as explained in Section 3.3. Instead, it will focus on the application as a whole. The contracts will be tested in a test environment and not be uploaded on the mainnet. Reason for that is that once the contracts are uploaded on the mainnet, they cannot be modified. Deploying them in a test environment allows further improvements for the contracts.

1.1 Current practices

The e-commerce retail is a fast-growing industry with platforms like eBay, Amazon and Alibaba revolutionising the industry by connecting buyers and sellers globally through the open Web. The total value of e-commerce business in the UK for 2015 is estimated to have reached 533 billion pounds[6].

Even though these platforms don't provide the best search algorithms or user experiences, they hold a monopoly on the industry. As a matter of fact, according to marketingland.com[3], Amazon raised to a level of 40.9% dominance of the total market.

One of the problems with the current providers of e-commerce services involves buyers and sellers being at the mercy of the company[52]. The company can decide to block the merchant from transacting on their platform at their own discretion, having consequences for the merchants whose livelihood might depend on this.

Other problems with current e-commerce solutions are listing fees which are sometimes too high. Amazon merchants pay commission on sales[41] from 6 to 45%¹ while other marketplaces like Etsy[42] also include a listing fee on top of commissions. Paying fees in itself is not all a bad thing because users benefit from the services provided by these companies. However, the friction cost for transactions can get exuberantly high pushing the merchant to either pass the fees to consumers or end up with little margin. Users engaging in online sales are forced to pay a commission towards entities involved in the e-commerce ecosystem as depicted in Fig 1.1

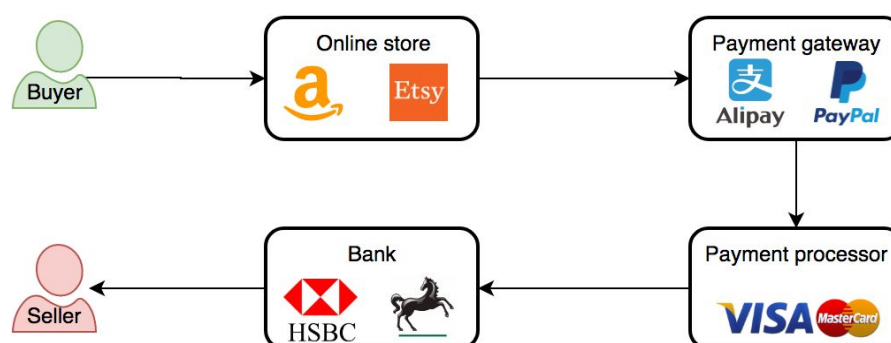


Fig 1.1 - Entities involved in online transactions

Lastly, merchants and consumers don't own any of their data. Reviews, purchase history etc are all owned by these companies sometimes using them against the merchants[44]. The new EU General Data Protection Regulation (GDPR) coming into place in Q2 of 2018, address this issue by introducing significant changes in how user data is managed and used[43]. However, while GDPR is a solution for the current practices, we should explore further options in building and design online marketplaces that fundamentally solve these issues.

¹ Varies per category of product

1.2 Alternative

Blockchain technology has been around since the creation of Bitcoin cryptocurrency in 2008[2]. By definition, blockchain it is a public ledger that records all transactions that ever occurred, allowing distribution of information without being copied. The underlying technologies of blockchain include cryptographic elements and mathematical proofs that ensures fairness of the system. These concepts are explored in Chapter two of this paper, offering an overall understanding of the technology.

Today, blockchain technology goes beyond Bitcoin cryptocurrency and comes in different flavours. Figure 1.2 is a representation of a handful of blockchain solutions available today, each with their own set of qualities, suited for different use cases.





				
	Hyperledger Fabric	Ethereum	Ripple	Bitcoin
Description of Platform	General purpose Blockchain	General purpose Blockchain	Payments Blockchain	Payments Blockchain
Governance	Linux Foundation	Ethereum Developers	Ripple Labs	Bitcoin Developers
Currency	None	Ether	XRP	BTC
Mining reward	N/A	Yes	No	Yes
Consensus network	Pluggable: PBFT ¹	Mining	Ripple Protocol	Mining
Privacy	Open to Private	Open / Private	Open	Open
Smart contracts	Multiple programming languages	"Solidity" programming language	None	Possible, but not obvious

Fig 1.2 - Blockchain solutions[53]

This project will explore Ethereum[1] blockchain which allows the creation of smart contracts, enabling transactions to hold business logic and perform functions. Smart contracts used to carry these functions are encoded and deployed on the blockchain. This type of approach in distributed networks opens up the possibility of building Decentralized Autonomous Organizations (DAO), operating without hierarchical management.

Radex[5] fits into the concept of a DAO being a decentralised marketplace for cryptocurrencies. It consists of two parts: a smart contract that handles all the financial transactions and the web application that makes interacting with the smart contract easier. The application lives on the blockchain and it performs a free exchange service for its users without the need for maintenance. The smart contract is well reviewed, tested, and most importantly open source and easily verifiable.

Using Ethereum blockchain and cryptocurrencies, we can remove reduce the friction cost for transactions by eliminating unnecessary third parties from the ecosystem. Smart contracts can handle safely and transparent the payment logic with minimal fees, removing the need for complicated processes of the business logic like the *checkout* or *cart*. Open source projects like OpenBazaar[4] have taken an initiative in this direction by providing a platform focused collaboration of community and cryptocurrency payments using Bitcoin blockchain.

Users connecting to OpenBazaar peer-to-peer network benefit of listing and buying products in a safe environment and without any constraints. Taking this concept further, we can build an application on Ethereum blockchain, while keeping the same core principles. By making so, buyers and sellers will be in control of their data and they can transfer their entire history (purchases, reviews etc) to other e-commerce dApps.

1.3 Roadmap

The following roadmap describes topics covered in the following chapters:

Chapter two will contain the background on the technologies used for building the decentralised application and how they fit in the scope of this project. I will cover topics on Cryptography and distributed networks like IPFS and Blockchain

Chapter three will describe system requirements and design that application aims to fulfil and discuss how smart contracts are used for the application logic.

Chapter four will cover the roles of different user types for interaction with the dApp with provided screenshots for each steps taken.

Chapter five will explore the functions of the application and how are implemented. This Section will also discuss key decisions taken during the implementation process. Furthermore, I will provide an explanation for the system architecture and, and smart contracts implementation.

Chapter six will cover the testing procedures used to check the application functionalities and. Additionally, this chapter will contain the evaluation and reflection of the project.

Chapter Seven will offer an overview of the final application including personal insight over the development of the application. Additionally, I will list a few possible implementations for the future

Chapter 2 - Background research

This chapter will focus on introducing the Blockchain technology and distributed systems including broad explanations of key concepts used by these technologies, like cryptography and smart contracts.

2.1 Cryptography

I will start with a brief overview of key cryptographic elements like Hash algorithms, Public key cryptography and Merkle trees, explaining how are they relevant to Blockchain technology.

2.1.1 Hashing

Hash functions are mathematical processes that take as an input a string of any length and return a string of fixed length. The process of executing a hash function is aided by a hash algorithm. For example, the SHA-256 hashing algorithm, used in Bitcoin, will always produce a string of 256 bits fixed length.

To be secure, hash functions need to have certain properties like:

- Deterministic: the hash value for the same input should always be the same. This will ensure the same output is always given for the same input to keep track of the input value.
- The hash function should produce the output as quickly as possible
- Every small change in the input should change the hash output completely
- Having the hashed output should be infeasible to find the original input.
- Every hash should be unique, collision-free. $H(a)$ should never be equal to $H(b)$

The examples below are a representation of the hashed value for different variants of the same string satisfying the conditions above:

- Hello World

```
A591A6D40BF420404A011733CFB7B190D62C65BF0BCDA32B57B277D9AD9F146E
```

- hello world

```
B94D27B9934D3E08A52E52D7DA7DABFAC484EFE37A5380EE9088F7ACE2EFCDE9
```

- hello world!

```
7509E5BDA0C762D2BAC7F90D758B5B2263FA01CCBC542AB5E3DF163BE08E6CA9
```

Unfortunately, there is no bulletproof hash algorithm that guarantees collision free hashes. However, currently, SHA-2 and SHA-3 are widely used as the chances of a collision is more of a theoretical nature. In Bitcoin protocol the hashes are used in multiple instances, one of it being block hashing algorithm used to write new transactions into the block through mining (fig 2.1)

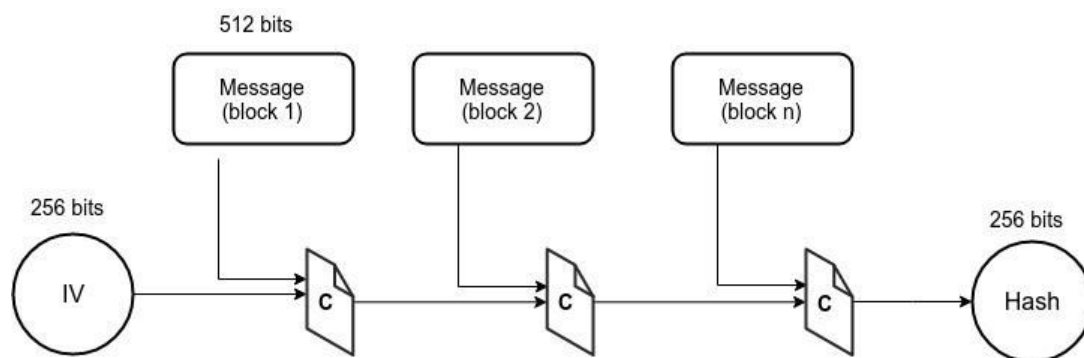


Fig 2.1 - Hashing transactions

2.1.2 Public Key Cryptography

Also, known as asymmetric cryptography, public key cryptography is an encryption standard that uses mathematically related keys; a public key and a private key. In asymmetric cryptography, the public key is used to encrypt and the private key is used for decryption (fig 2.2). The function of having a pair of keys for encryption and decryption is what differentiates asymmetric cryptography from symmetric cryptography where a single key is used for both encryption and decryption[40].

Due to their properties, public keys can be shared among users allowing them to verify digital signatures and encrypt content while keeping private keys safe. The idea of a digital signature is that a message can be signed by one person but verified by many[40]. Public keys are stored on the digital certificates for ease of use while private keys are usually stored safely in the software or hardware that use them.

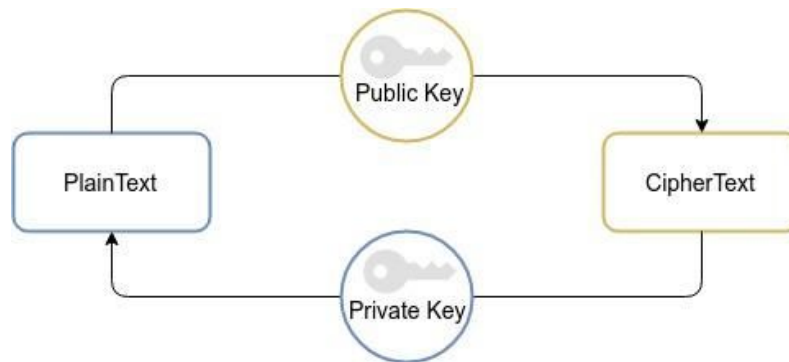


Fig 2.2 - Public key cryptography

In Bitcoin and Ethereum blockchains, coins are represented by a chain of digital signatures transferable between the users of the network.

Digital signatures do not differ much from actual signatures on a document. They provide data integrity for the messages created in the protocol by using public key encryption. Digital signatures, as explained by Ross Anderson in Security Engineering[40], depending on the following properties:

- The public key can not be used, by any means, to compute the private key.
- The sign function produces a unique digital signature for any given message and a private key.
- The verify function returns a binary output after verifying whether the signature is authentic after checking the message, the signature and the public key.

The concept for a chain of digital signatures is described in the original whitepaper of Bitcoin written by the anonymous person known as Satoshi Nakamoto[2].

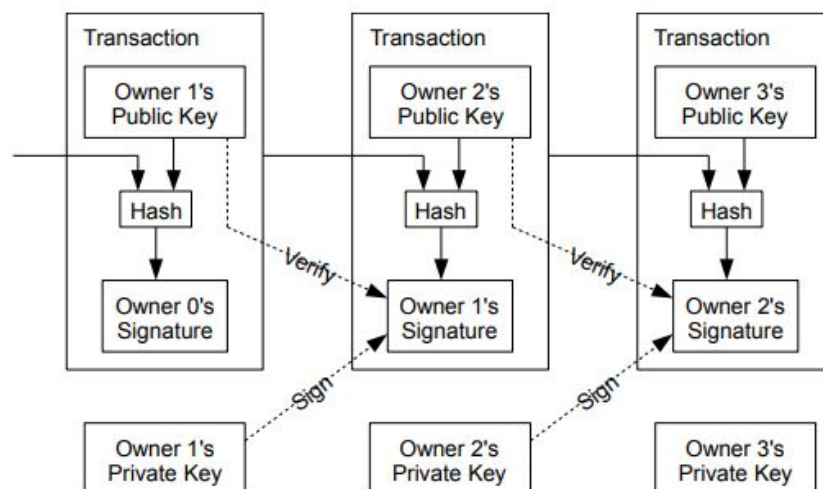


Fig 2.3 - Digital signature verification

2.1.3 Merkle trees

In distributed peer-to-peer systems like Bitcoin and Ethereum, Merkle trees are used to check inconsistencies in data structures and provide efficient and secure verification of large amounts of data. The information in the blocks need be in its original state without it being altered or corrupted[7].

Essentially Merkle trees are composed of the following:

- a set of nodes with a large number of data at the bottom of the tree, containing the underlying data
- a set of intermediate nodes where each node is a hash of two nodes from the bottom of the tree.
- a single root node formed similarly from the hashes of the previous nodes.

The fundamental purpose of Merkle trees, also known as hash trees, is to ensure blocks of data can be received from different peers ensuring data is correct when put together. Any attempt of changing the data in a merkle tree is enough to lead to inconsistency in the chain (Fig 2.4)

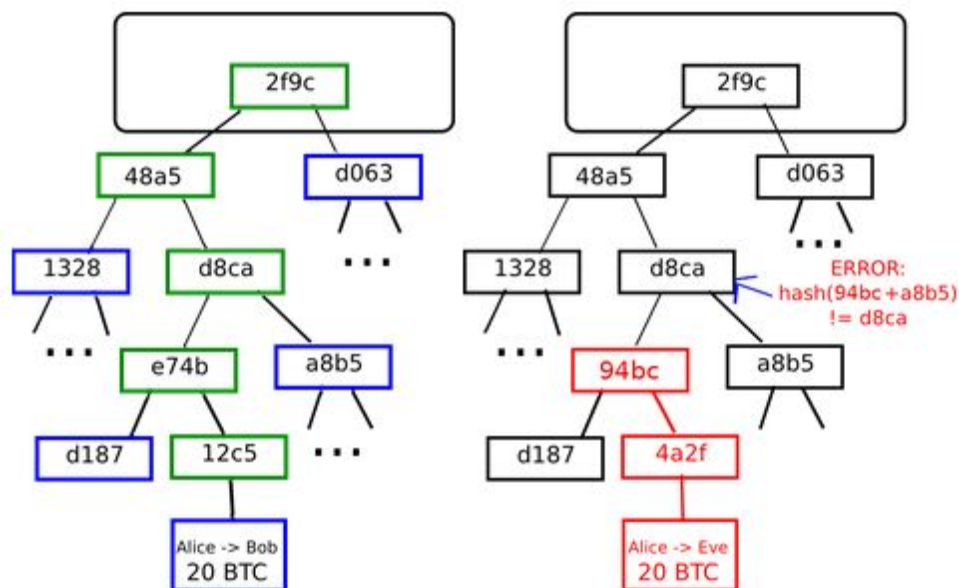


Fig 2.4 - Inconsistency example in Merkle Tree [7]

Left: the integrity of a branch can be easily verified with a small number of nodes

Right: any attempt to change the data will lead to inconsistency

2.2 Blocks

In the context of the blockchain, blocks are containers of cryptographic data are composed of a header and a list of transactions (fig 2.5). The average size of a block is around 1mb for bitcoin[14] or 200Kb for ethereum[15] and contains an average of 500 transactions. They are added to the blockchain at a fixed interval of time.

Blockchain immutability of data is achieved by each block header containing a hash of the previous block header, this way forming a chain. Changing any part of the block header will result in a different hash, thereby changing the value of the hash referenced in the upcoming header and so on. Stopping an attacker of tampering with data written in a block or chain of blocks can be achieved by different mining techniques which are covered in Section 2.3.

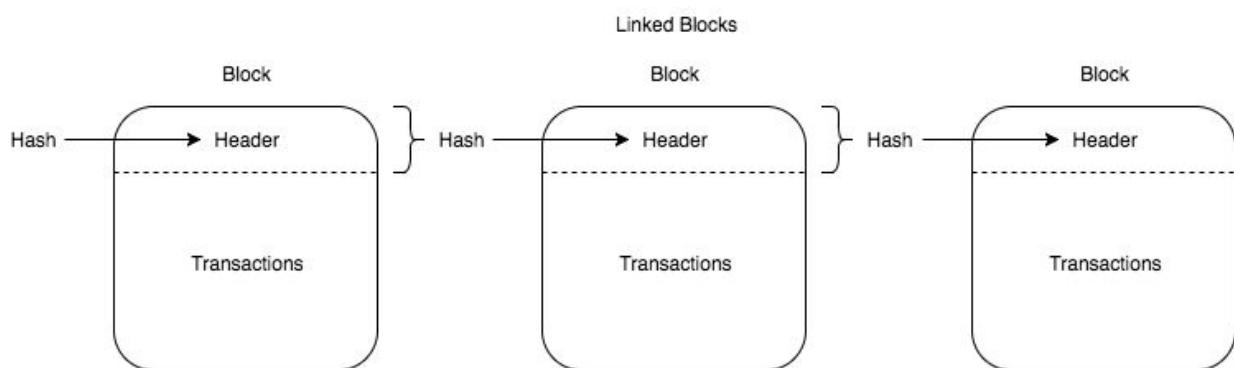


Fig 2.5 - Blocks example

2.2.1 Block headers

Block headers also contain a Merkle root which, is the data structure that summarises the transactions in a block. Merkle trees work by hashing two transactions so as to form a new hash. The new hash is eventually paired with other newly formed hash and the process is repeated until remains just one hash – called Merkle root (fig 2.6). This information is stored in the header of each block on the blockchain. Merkle roots are binary trees, meaning that if there is an odd number of transactions the last transaction will duplicate so as to create the root.

Longest Proof-of-Work Chain

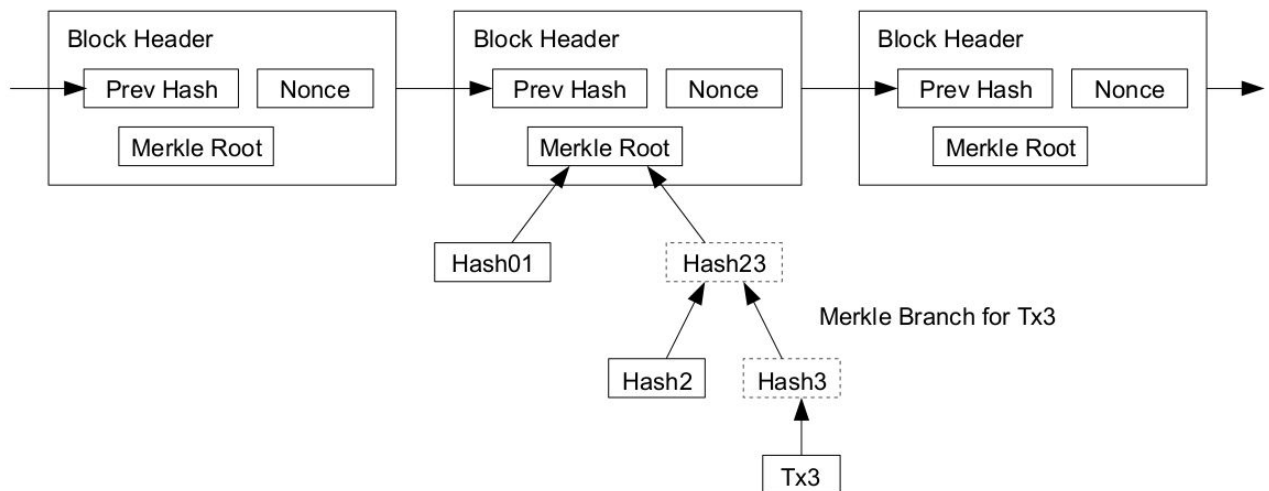


Fig 2.6 - Merkle root

2.3 Consensus Algorithms

In Blockchain technology, consensus algorithms are used to ensure that transactions are valid and distributed among many participants. They provide a solution to double spending problem[37] achieving reliability in a network with untrusted peers. In the current evolving space, there are four major consensus algorithm each with its own unique set of benefits.

- POW - Proof of work[37]: First successful consensus algorithm, used by Bitcoin. Require users to contribute with computational power in order to validate blocks.
- POS - Proof of stake[35]: Involves users staking their tokens in order to validate blocks. More efficient than POW as it does not require computational power.
- DPOS - Delegated proof of stake[39]: Similar to POS, users stake their tokens to vote on a selected group to validate blocks. More centralized than POS.
- BFT - Byzantine fault tolerance[38]: Uses validators to manage the state of the chain and ensure honesty.

It is very likely that new approaches of consensus algorithms are to emerge in the following years. However, this Section will discuss and explain two currently popular consensus algorithms.

2.3.1 POW - Proof of Work / Mining

Mining is the mechanism of validating and adding transactions to the global ledger. Mining activity secures the blockchain protocol and allows it to function in a decentralised manner. Parties who compete with computational power to solve difficult mathematical problems based on cryptographic hash algorithms are also known as miners. The first property of mining, is ensuring that a predefined amount of work has been invested by the party providing the proof of work. The second property ensures the proof is easily verifiable. Finding solutions to a proof of work puzzle is a probabilistic process with the success of probability defined by the amount of computation available and a predefined difficulty.

For example, the hashing algorithm used in Bitcoin[16] is *double-SHA256* ($SHA256^2$) and the predefined structure is a hash less or equal to a target value T . The success probability PR , of finding a nonce n for a given message msg , such that $H = SHA256^2(msg||n)$ is less or equal to the target T is

$$Pr[H \leq T] = T / 2^{256}$$

This will require a party attempting to find a proof of work to perform on average, the following amount of computations

$$1 / Pr[H \leq T] = 2^{256} / T$$

Multiple valid solutions can be found for any given block. However, only one of the solution needs to be found in order for the block to be solved.

For every block added to the network there are two types of rewards. The transaction fees reward, which is the amount of fees available in each block from transactions. Coinbase transactions are stored inside a block and serve the role of paying the miner his block reward. For every block added to the chain, new coins are added to the system as a form of payment to the miner.

The number of bitcoins given as rewards for a single block started at 50 bitcoins. This number is halved every 2,10,000 blocks or about 4 years time. Currently, the reward for each block is 12.5 bitcoins and it is expected to halve again In 2020 (fig 2.7) [31]. The halving will continue to happen until the maximum of 21 million supply coins has been reached, estimated to happen in the year 2140.

Amount of bitcoin created with each mined block (dates are indications)									
Before November 2012	Nov 2012 - Jul 2016	Jul 2016 - 2020	2020 - 2024	2024 - 2028	2028 - 2032	2031 - 2036	2036 - 2040	2040 - 2044	2044 - 2048
50	25	12,5	6,25	3,125	1,5625	0,78125	0,390625	0,1953125	0,09765625

Fig 2.7 - Amount of bitcoin created with each mined block[17]

2.3.2 POS - Proof of Stake / Minting

Comparatively to proof of work algorithm where blocks are created by miners and their computational power at hand, proof of stake ensures blocks are added to the system by *minters* who stake their tokens for proposing the next block. The weight of each validator depends on the number of tokens (stake) available in their wallet.

This approach for consensus algorithm has economical advantages as blocks can be added to the chain without requiring huge amounts of computational power while at the same time reduces the risk of centralisation. Currently, major blockchains such as Cardano[35] and Ethereum are working to implement POS algorithm into their Blockchains. Ethereum's Casper[36] POS consensus algorithm is due to be implemented in Ethereum blockchain later in 2018. Besides the mentioned above, Casper algorithm aims at improving network scalability and security of the network resulting in faster blocks and lower fees for the users.

2.4 The Blockchain

Blockchain technology a shared, immutable public ledger for tracking history of transactions. Secure and transparent by design, blockchain aims at enabling global business transactions to be more direct and trustworthy.

In blockchain, each computer taking part in the network owns a copy of the same data. Data in this context is a continuous list of *blocks* that can contain records of transactions for digital assets, information or facts. New blocks are added to the blockchain after they pass the consensus algorithm used by the particular blockchain. Consensus algorithms, explained in Section 2.3, use principles of game theory so as to incentivise participants of the network to validate blocks.

2.4.1 Early days of Blockchain

Originally conceptualised for the digital currency Bitcoin, blockchain is introduced in the official white paper[2] as a peer-to-peer electronic cash system. It has started as an experiment for offering decentralised² programmable money to the general public; a way of testing the boundaries of money in modern society. The white paper was released by the pseudonymous author/group of authors Satoshi Nakamoto during the same time of financial crisis in 2008. Nakamoto, claims to be a thirty-six years old Japanese programmer, but his proficiency in English as noted in his emails raise doubts for many[48].

² Decentralized in this context means that is not governed by any entity, instead is governed by the rules of the protocol.

Bitcoin Blockchain was released early in 2009 and the first block ever mined, or the *genesis block* contained the following embedded message: “*The Times 03/Jan/2009 Chancellor on brink of second bailout for banks*”[46]. The message, which happens to be a headline from the Times newspaper, was intended to point out the instability caused by the fractional-reserve banking system, also suggesting that Satoshi Nakamoto could have live / lived in the United Kingdom[47]. Satoshi Nakamoto has been an active member in the cryptography forums, discussing freely the concept of Blockchain until later 2012 when he disappeared entirely from the web.

For many years Bitcoin has remained just that, a secure digital cash system with properties of anonymity and little interest from the general public. It was not long before underground markets like Silk Road start taking interest in the blockchain as it provided global untraceable transactions. Silk road[8] marketplace, created by Ross Ulbricht went online in mid-January 2011 and operated until 2013 when it was shut down by FBI[49]. During this period, Silk road operated as an illicit online marketplace for selling illegal products.

When currencies provide anonymity and are not governed by any legal body it is very easy to find illegal use cases for them. Blockchain technology has a lot more to offer rather than simple means of buying illicit products on the darkWeb. However, in order for them to be widely accepted and function correctly, they have to be used beyond these activities and regulated by the right institutions.

2.4.2 Blockchain Today

It is important to understand that unlike client-server architecture that requires a server in order to retrieve data, blockchain technology is peer-to-peer that facilitates transactions between users without having to involve a server. This makes the blockchain a decentralized platform without any authority in control where users interact with each other directly.

Considering the bad reputation from illicit markets, blockchain has prevailed and steadily evolved in a more complex system beyond the use of cryptocurrency. According to Forbes[9] today, 80% of the banks worldwide are developing blockchain solutions for their business with other industries following the lead. If used correctly, Blockchain technology can potentially disrupt industries like Voting, Real Estates, Supply chain management or any other industry that is mediated by a third party.

The following examples will further explore how blockchain technology is used today in different pilot projects around the world. While some of these projects are a better alternative to current flawed systems, others use this technology to pursue innovation and development. However, the potential of blockchain technology is far greater than initially conceptualised and can disrupt any industry where there is a “trusted” third party which can corrupt transactions.

Digital identity

Citizens of Zug, Switzerland are the first community in the world to allow its citizens to use a blockchain based digital identity, starting from September 2017. Locals have the opportunity to acquire a digital identity through uPort[45] decentralized application, accessing services like e-signatures and parking tickets. The local council states that their role is to verify and confirm identities, not store them[11]. Digital identities are explored further by Zug local council covering services like e-voting and ePassports. If successful, the project model can be a catalyst for other councils and governments to adopt this technology.

National Currency

Recently the Venezuelan government has announced the launch of the first government-issued cryptocurrency. The petroDollar is a response to the endless political-economic conflict present in the country and is backed up by the country oil and gas reserves. The currency uses Ethereum blockchain to handle both crypto and fiat currency transactions. The petroDollar is aimed at offering a more stable form of currency using open source technologies that engender trust and transparency[12].

Finance

Most world banks are currently looking into the potential of blockchain technology to leverage their services, speeding up and simplifying cross-border transactions. Identity management is also explored by financial institutions aiming to provide simpler solutions to KYC (Know-your-customer) and AML (Anti-money-laundering) procedures. The blockchain is an infrastructure of great complexity that can leverage a wide range of services within fintech industry[13].

Hyperledger

Recently, IBM has recently developed their own blockchain aiming to leverage blockchain technology and open source best practices. Hyperledger is a tokenless business blockchain framework for enterprise solutions. It is hosted by the Linux Foundation and benefits of a modular architecture that solves scalability and performance issues[10]. Hyperledger is open source and does not require tokens/cryptocurrency to function which makes it easier for large institutions to adopt the technology without having involving cryptocurrencies.

2.5 Ethereum Blockchain

Referring to Bitcoin and Ethereum as blockchains is completely accurate, however, there are fundamental differences that set them apart. Bitcoin has started from the idea of being digital peer-to-peer money[2] and payment mechanism without any entity controlling it. While it successfully achieved that, it has also open new possibilities in regards to transactions and digital signatures. Ethereum differs from Bitcoin by allowing the creation of smart contracts which can be highly programmable digital money. Ethereum allows efficient insertion and deletion of items by using specialised Merkle trees significantly reducing the data stored on the blockchain. This technology facilitates the creation of smart contracts through Solidity programming language which in effect are used for the creation of Decentralized Applications or dApps. Ethereum foundation offers detailed implementation of the blockchain on Ethereum wiki[18]

2.5.1 Ether and Gas

The main resource on the ethereum network is *ether*. It is used to pay transaction fees which in effect incentivize the miners to maintain the network. Ether can be divided into four units of division as follows:

- 1 - wei
- 10^{12} - szabo
- 10^{15} - finney
- 10^{18} - ether

Gas is the unit of measurement for the amount of computation needed to run the contract operations/functions. Complex business logic and large data stored in smart contracts can have an impact on the amount of ether paid so as to run the contract. Gas and payments are explored in depth in the ethereum paper[20]

2.5.2 dApps

Applications running on the blockchain are called dApps or decentralised applications. For regular users, dApps are the gateway for interacting with the blockchain. Currently, there are 1377 functional dApps[21] with many others still under development. They vary from games to trading platforms to cloud storage etc. By making use of the blockchain technology and smart contracts, dApps are decentralised, open-source and fuelled by cryptocurrency.

Users can access the functions of a dApp by connecting to an ethereum node. A range of Ethereum clients are available that allows users to connect to the network. The most widely used implementation is written in GO (geth/ go-ethereum) however, other implementations

are available like Python or Ruby[22]. The job of any chosen client is to provide access to the blockchain by downloading the blockchain and verify new blocks allowing the user to perform mining operations. It also facilitates the users with a command line interface allowing them to manage their accounts, deploy smart contracts or initiate transactions.

Having ethereum clients aimed more for developers and users wanting to experiment with the blockchain, there are easier ways for interacting with dApps on the blockchain. Metamask[19] browser plugin provide an alternative for interacting with the ethereum blockchain facilitating interaction with dApps without using an ethereum client. This method does not download the whole blockchain on the client providing users to manage their blockchain wallets (fig 2.8).

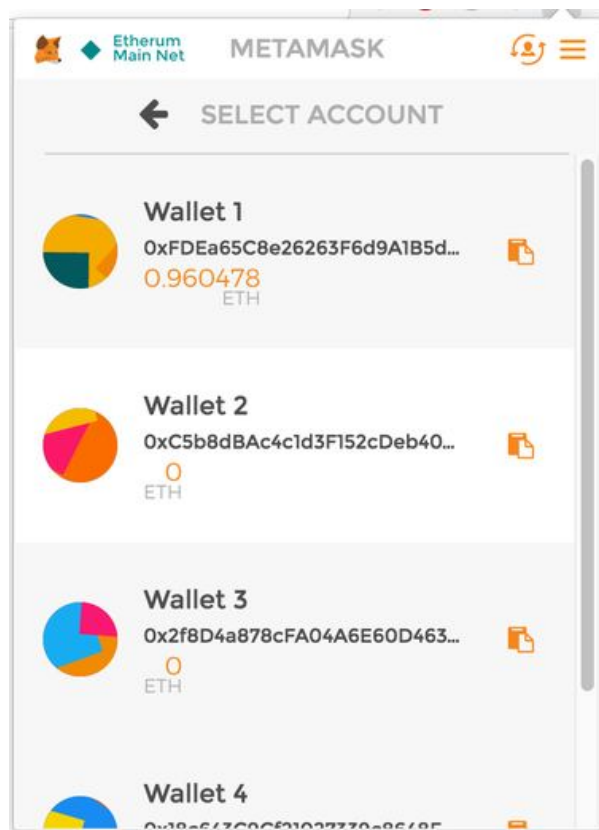


Fig 2.8 - Metamask wallet

2.5.3 Wallets

Wallets are used to manage identities and funds on the blockchain. Users interacting with the blockchain are automatically generated a set of private and public keys. The private key has to be kept secret from anyone as it is the user way of accessing his ethereum wallet/account. In contrast, the public key is the identifier of the wallet and is shared with other peers wanting to send funds.

Users interacting with the ethereum blockchain through an ethereum client benefit of a built-in wallet. Wallets can also be a standalone application that user installs on their pc, tablet or phones. They provide an easy interface for users to manage transactions and accounts or deploy smart contracts.

2.5.4 Smart Contracts

Smart contracts are suitable for various type of transactions, particularly those involving a transfer of value without relying on an intermediary. This makes them good for any type of exchange of goods or data. As they are deployed on the Ethereum blockchain, smart contracts are easy to verify in order to make sure the operations performed are valid. Once deployed on the blockchain, the data stored in smart contracts is immutable and tamper proof.

Solidity

Solidity[24] is a contract-oriented, high-level programming language inspired by Javascript, Python and C++ used for writing and executing smart contracts on Ethereum virtual machine (EVM). Similar to objects in object-oriented programming, contracts contain common data types, functions and state variables. Solidity also contains contract specific features like event notifiers for listeners, modifier (guard) clauses and global custom variables.

Ethereum contracts can be used to create multi-signature transactions, crowd fundings, voting systems etc. Contracts developed in solidity are later on deployed on the ethereum blockchain where anyone can interact with them. Code Block 2.1 depicts the content of a simple storage smart contract containing a *get* and *set* function so as to retrieve and store data on the blockchain.

```
pragma solidity ^0.4.13;
contract SimpleStorage {
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public constant returns (uint) {
        return storedData;
    }
}
```

Code Block 2.1

Solidity provides deterministic execution through smart contracts that can be easily programmable to execute business logic upon receipt, removing the need for intermediaries in some scenarios. Combined with the immutability aspect of transactions of the blockchain, solidity provides proof of transactions and execution of business logic upon delivery. Solidity documentation[24] offers an extensive overview of how the code works together with examples and tutorials.

2.6 IPFS - Interplanetary File System

The InterPlanetary File System (IPFS) is a hypermedia peer-to-peer protocol that connects all computing devices with the same system of files. Similar to the initial aims of the Web, IPFS is trying to solve issues of distribution and addressing of content. In contrast with HTTP which is an IP addressed protocol, IPFS is content addressed. As described in the official whitepaper[27] (uploaded and hosted on IPFS) the protocol has the following underlying technologies:

- Distributed Hash Tables[32] determine how data is distributed between the nodes offering great performance and scalability.
- (SFS) Self-Certified Filesystems[34] address the issue of cryptographic key management providing security over large distributed networks.
- Block Exchanges : BitTorrent[27] is a peer-to-peer file sharing protocol that facilitates efficient distribution of data in a system with untrusted nodes.
- Version Control Systems : Git[33] is a distributed version control system for tracking changes in data over time. Git provides data assurance through cryptographic integrity.

Any document uploaded on IPFS is automatically hashed and stored on the network nodes. Referring an object requires accessing the hash of the file through a request. Requests on the network are global and users will get their response back from the closest node that has the hash (document) available.

Chapter 3 - Requirements and Analysis

This chapter will explore the requirements the application aims to fulfil, discussing how they can be implemented using smart contracts and application logic.

3.1 Application Components

The goal of the decentralised web store application is to provide a proof of concept for the idea of decentralisation of services by using open source protocols. This Subsection will explore the components used to deliver the e-commerce dApp.

User interact with the application through a browser, accessing the interface which communicates with the blockchain through smart contracts for performing the various functions.

The web interface is a collection of HTML, CSS and JavaScript files that render the page for users. I have decided to use Bootstrap framework[50] in order to deliver the front-end as it provides quick means for prototyping and designing modern, responsive web interfaces. This decision comes as an alternative of using Meteor framework[60], initially intended to for this project.

Solidity language[24], is used for the creation of smart contracts that are deployed on the Ethereum test network. For interaction with the blockchain, I used Web3.js[51], a collection of JavaScript libraries that act as a bridge in order to communicate with a local ethereum node.

Storing large files on the blockchain can become costly, therefore IPFS explained in Section 2.6 is used for storing product pictures and description.

Smart contracts are deployed and tested in a local environment using Truffle framework[29]. During the testing stage, the interaction with smart contracts is achieved through truffle console. Figure 3.1 is a representation of the components interaction mentioned above.

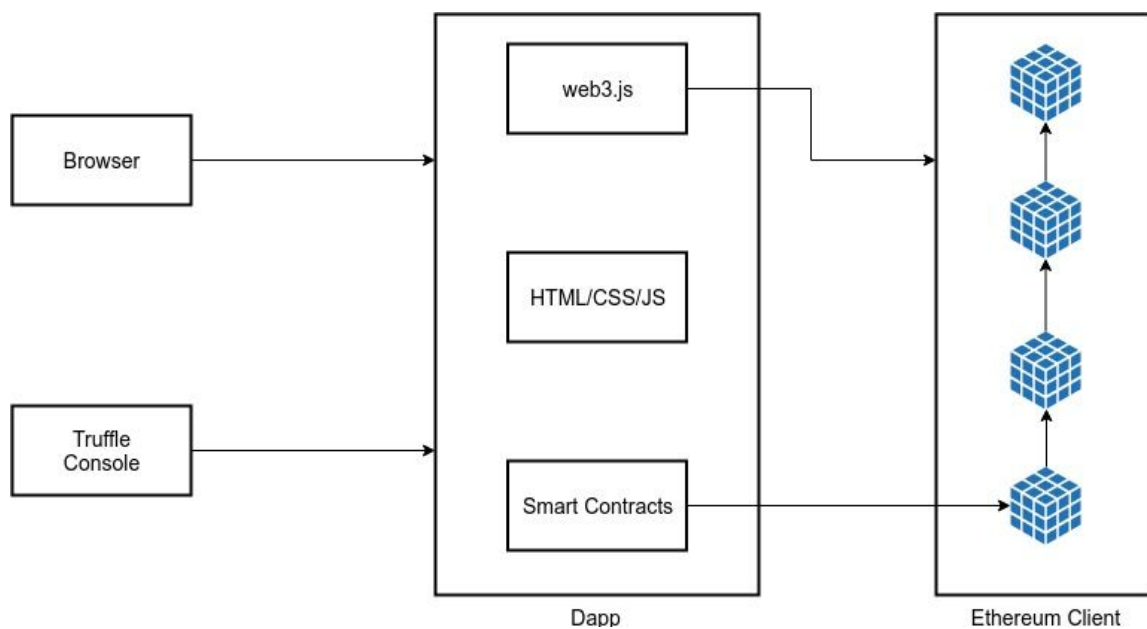


Fig 3.1 -dApp components interaction

3.2 Application Functions

- List an item – a webstore should allow its users to list their items for sale. This functionality will enable to upload the item they want to sale for free. The listing will be stored both on the blockchain and in a database for ease of querying
- Add files to IPFS – Functionality to add product images and description (large text) to ipfs
- Browse products – This function will enable users to browse through products based on categories, price etc
- Buy function – Will enable users to buy a product using ethereum through their metamask account.
- Escrow Contract – When a product is bought, an escrow contract will be created between the seller, buyer and a 3rd party arbitrator.
- 2-of-3 signature - Will add fraud protection by implementing the 2-of-3 signature solution where 2 of the 3 participants have to vote to release the funds to the seller or refund the amount to the buyer.

3.3 Dispute resolutions

In order for an e-commerce platform to function, it requires credibility. All parties conducting trades on the platform must be (and believe) they are protected from frauds and scams. Online retailers don't know their customers personally and fraudsters can take any persona to launch massive scams.

The first question to consider when building a decentralised e-commerce application is how to approach fraud enforcement and prevention? The three ways generally used in enforcing and preventing fraud are explored below.

- Legal Enforcement - if one of the parties feels the other party has breached the contract, they can file litigation resulting in some cases to trial.
- Private Enforcement - resolves disputes between two parties with moderation through a third party.
- Multisignature escrow - facilitates transactions between two parties with a third party holding the funds until the contract is signed by 2 of 3 parties.

The first method is probably the oldest and most popular way of resolving a dispute. It involves a legal authority acting as a third party between two entities found in a dispute. However, using legal enforcement to resolve disputes for online sales is infeasible. Trial processes can take a huge amount of time to resolve and involve legal costs that can get much higher than the actual value of the fraud.

This method also does not fit into a global market economy where sellers and buyers could be anywhere on the globe, meaning different laws apply to them. It is a tedious process that takes time and puts all the responsibility in the hands of the customers. Legal enforcement is a highly manual process.

Private approaches to law enforcement and prevention have partially led to the continuous growth in e-commerce space over the last years. Online payment processes like PayPal provide their own methods of resolving disputes[25]. They provide credibility and trust for customers using the platform, by offering a variety of automated³ processes to resolve disputes.

These companies receive all incoming disputes raised from sellers and buyers and try to resolve them, penalising the losing party. Since they control the platform directly, they are entitled to release refunds, uphold any negative reviews or ban people from using the platform entirely.

Private law enforcement and prevention works great for its purpose and has provided a trust layer in e-commerce space. However, this solution is far from perfect and has fundamental problems like centralisation and data control. Users must provide enough information to these companies so that the cases of fraud could be litigated in the court system if needed.

³ Mostly automated. In some cases a representative may take the final decision.

The multi-signature escrow approach for addressing fraud combines decentralised moderation with commitment mechanism of smart contracts. This method allows users that engage in purchasing items, to place their funds in a special smart contract that require 2 of 3 signatures for funds to be released. The three participants are buyer, seller and an arbitrator. The smart contract is self-enforcing and arbitrators need only to sign successful transactions. However, if a dispute takes place, the moderators may intervene and listen to both parties so as to make a rightful decision. The four possible outcomes for using multi-signature escrows are explained below and depicted in fig 3.2.

1. The seller cannot deliver and the buyer is unhappy. They mutually agree to a refund.
2. Buyer and seller are both happy and funds are released to the seller.
3. In a dispute, moderator favours the seller. Buyer and moderator agree to send the funds to the seller.
4. In a dispute, moderator favours the buyer. Seller and moderator agree to send the funds to the buyer.

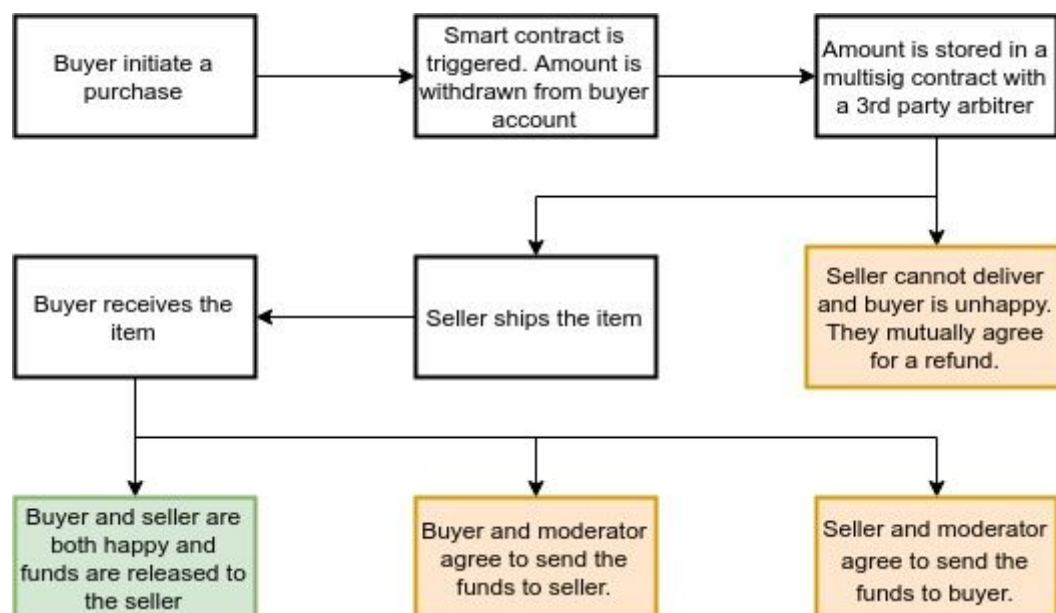


Fig 3.2 - Dispute resolution flow

There are different approaches to selecting the arbitrators so as to provide fair resolutions. OpenBazaar mentioned in chapter 2, provides a decentralised marketplace for arbitrators encouraging the community to be part of it. Buyers and sellers can give arbitrators a score for resolving disputes which will influence the arbitrator credibility. Different incentives schemes can be applied to this approach to motivate arbitrators for participation in disputes.

This project will not deal with the moderation aspect of the multi-signature contract and will focus on the application as a whole. The arbitrator in the escrow contract built for this project will be the same address that deploys the e-commerce contract on the blockchain (the contract creator). When a transaction takes place, the contract creator/shop owner will act as an arbitrator between the transacting parties facilitating any dispute if needed.

If it was deemed necessary to add this functionality in the future, the approach would likely resemble OpenBazaar mentioned previously. Instead, this project will focus on the stages and operations the contract needs to fulfil so as to deliver functionality for the e-commerce contract.

3.4 Decentralized storage with IPFS

The decision of using IPFS in this project comes as a solution for storing large data in a decentralised manner without involving high transaction fees for users. As mentioned in Section 2.5, contracts containing large data sets or complex business logic can result in high fees for the user. In the case e-commerce application, the amount of *gas* used by smart contracts can be drastically reduced by storing large files of data (pictures and product description) off the blockchain.

An alternative to storing media files can be a traditional server. However, this solution is highly centralized and can include extra costs. The server needs to be operational 24/7 in order to deliver the content properly.

Querying the blockchain and IPFS when users browse through the listings, can be time-consuming. In order to benefit from the decentralization aspect of IPFS and speed of traditional servers, I have taken the decision to implement both methods in order to provide a great user experience. Transactions will be stored on the blockchain and media files on IPFS while MongoDB will hold a copy of both.

Users browsing through the listing will receive the data from MongoDB through the server. If the server happens to be inaccessible, the user will retrieve the data from the blockchain and IPFS. moreover, the users will be able to view the listed items without being connected to Ethereum mainnet.

This solution facilitates availability and faster retrieval of the content providing a better experience for the user.

3.5 Web Application Interface

In order to improve the user experience further, the application needs having a few qualities. I have consulted with few potential users of the platform about this, and been advised to meet the following requirements:

- Clean user interface without any extraneous content and easy to interact with. All functions of the application have to be clear, having users to perform the tasks with minimal clicks.
- Provide clear instructions on how to use the application and how it works without using extensive jargon.
- Users must be aware at all times of the state of the application. Requests should be clearly notified upon success or errors.

Providing the above, the application will provide clear instructions on how to use it, presuming that the user is familiar with Ethereum and cryptocurrencies.

A full introduction of the ethereum blockchain, cryptocurrencies and how to use metamask will be available on a separate page.

The application should be available for users without having to connect to the ethereum mainnet. This will enable users to browse through the listings, without having the option of buying or selling items. Users will be prompted to connect to the Ethereum mainnet using metamask in order to perform operations.

After successfully connected to the mainnet, users can interact with the blockchain through the smart contract, facilitating them the options to list, buy or sell products. The web application has to be clear, intuitive and minimal steps during the process of performing these actions.

Chapter 4 - Application Walkthrough

This chapter will discuss the application flow presenting the steps users can take in order to interact with the web interface of the application.

4.1 User Journeys

This Section will explore the user types and possible operations users can perform on the e-commerce web app. Screenshots are provided for each operation in order to illustrate the process step by step.

- Visitor - Visitors are granted access to the web application through the web browser. This type of users has limited access to the application without being able to list and buy items. They can browse through the listings, see prices or sign up.
- Buyer/Seller - this type of users have all the above mentioned. Moreover, they are granted access to list or buy items through the web app after logging in.
- Arbitrator - Arbitrators are granted permissions to resolve disputes between users. In addition to a buyer/seller, arbitrators have access to a special menu where they can check the status of the transaction and intervene if any dispute arises.

4.2 Application interface

Fig 4.1 is a representation of the web application homepage. The main navigation bar, positioned at the top, contain the logo, a sign in button, links to other pages and a message informing visitors to sign in as illustrated in fig 4.2. Additionally, the main page contains a list of recently listed items that are accessible to the visitors.

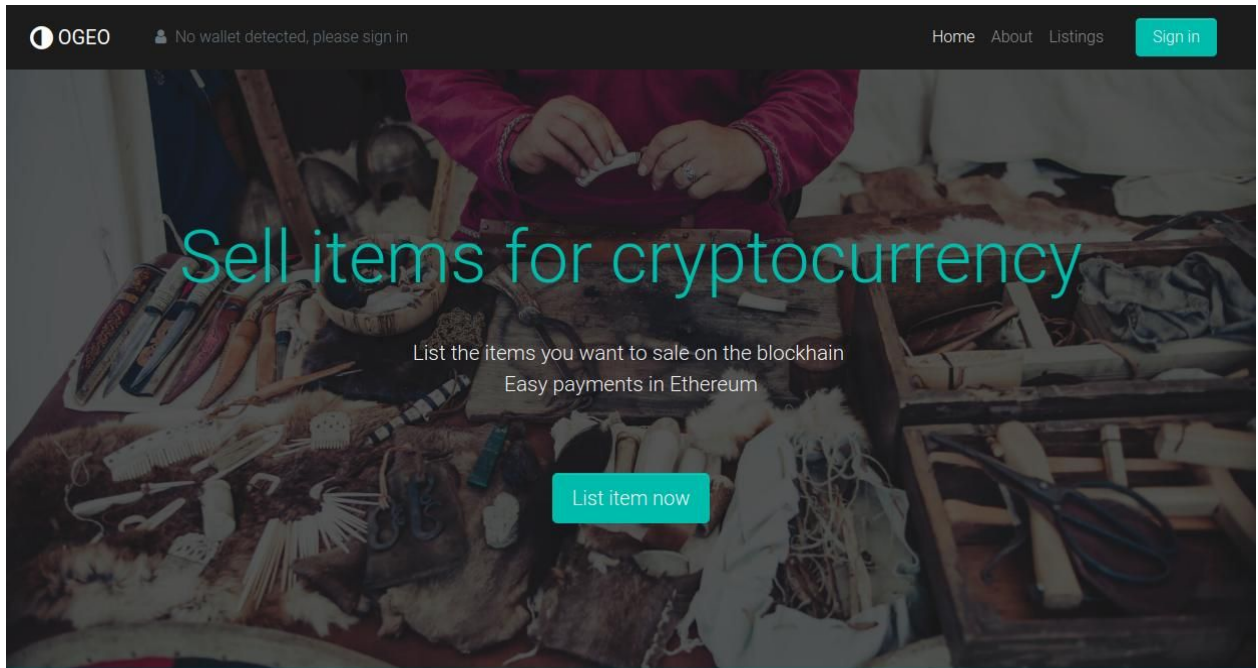


Fig 4.1 - Homepage



Fig 4.2 - Web application header

The *Sign in* button prompts the user with a modal display message containing simple steps for connecting to Ethereum network in order to use the Web application (fig 4.3). A broad description of the Blockchain, Ethereum and how to connect will be available to visitors through the *More information* button as a separate page. This decision has been taken in order to aid the user throughout the process of using the application while keeping the instructions at the minimum.

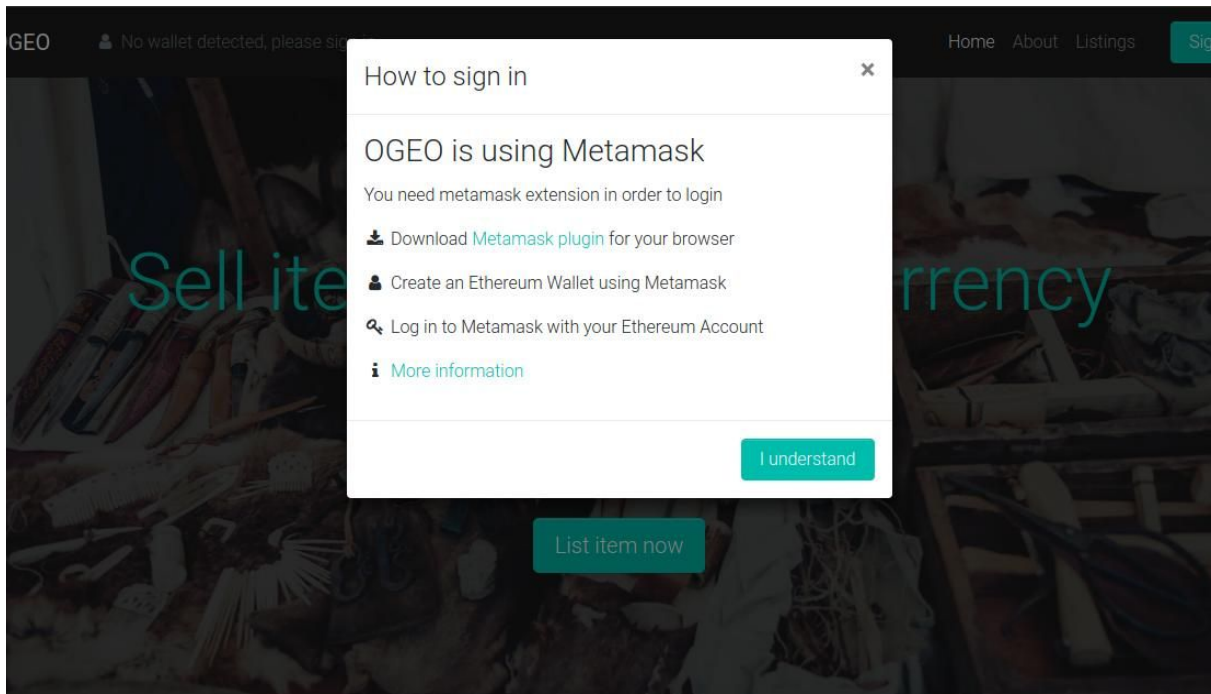


Fig 4.3 - Sign in popup

Users with Metamask plugin[19] available in their browsers can instantly connect to Ethereum Network. Metamask will create a wallet for them and generate a pair of private and public keys. Users will have to store safely their private key in order to access their wallet later and fund the wallet with cryptocurrency. Once the user is logged in to Metamask and has funded their wallet, the application detects their public address and let them perform all actions of a buyer/seller mentioned in Section 4.1

Figure 4.4 depicts the login process with Metamask while figure 4.5 is a representation of the metamask account showing the balance of the account and recent transactions.

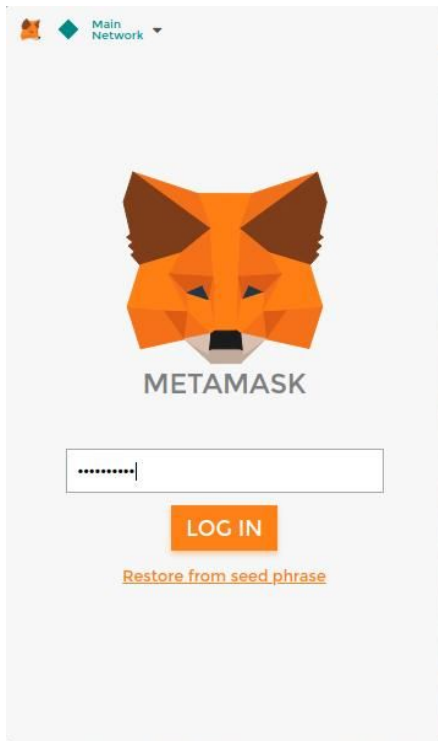


Fig 4.4 - Metamask Login

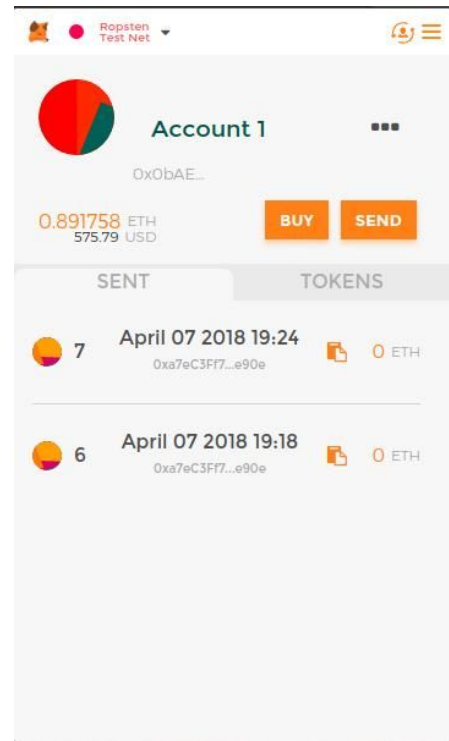



Fig 4.5 - Metamask Wallet



Fig 4.6 - User address showing in Main header

After the login process, the top navigation bar displays the wallet address of the connected metamask account as seen in Fig 4.6. By displaying the wallet address, users know they are connected and can navigate through the web application, buying and selling products. Users logged in can access the *List item* page where they are provided with a form for listing the desired product. All form fields are mandatory and users need to provide the name, description, image, category, condition and price of the product.

The frontend interface of the listing process is developed as seen in fig 4.7, however, the current state of the application does not allow listing of the product through the web interface. Currently, the interface is for demonstration purpose and all the products seen in fig 4.8 have been added to the backend using Truffle console. The process is covered in Chapter 6 - Testing and Evaluation.


0x0baefeacc205bfdc01dceadf858ab3d17b5aaa67

[Home](#)
[About](#)
[Listings](#)
[List item](#)
[Sign in](#)

List an item

Sell item for cryptocurrency

Product details

Name

Description

Photo


No file chosen

Category

Condition

Price

Fig 4.7 - List a product page


0x0baefeacc205bfdc01dceadf858ab3d17b5aaa67


[Home](#)
[About](#)
[Listings](#)
[List item](#)
[Sign in](#)

Categories

- Art
- Books
- Jewellery
- Digital media
- Electronics
- Handmade
- Fashion


Available products

Leather Bracelet




Handmade

Painting




Art

Vintage Leather Bag




Fashion

Jeans




Fashion

Wall Clock




Handmade

Silver Ring




Jewellery

Leather Bracelet



Handmade

Wall Clock




Handmade


Painting

Vintage Leather Bag

Jeans

Fig 4.8 - Listings page


 OGEO

 0x0baefec205bfdd01dceadf858ab3d17b5aaa67

[Home](#) [About](#) [Listings](#) [List item](#)

[Sign in](#)

OIL PAINTING



Art

Thundercats migas plaid before they sold out cray farm-to-table cornhole. Poutine tumblr iceland, pitchfork leggings literally art party kickstarter copper mug raclette vice salvia pinterest hexagon viral.

Chia taiyaki hella 90's, DIY portland mustache intelligentsia master cleanse thundercats pour-over. Offal semiotics pinterest plaid.

Synth mustache etsy, messenger bag art party cray hella hammock prism swag normcore gastropub. Butcher chartreuse marfa, occupy retro twee chillwave cloud bread.

Fam meh pitchfork, vape deep v scenester art party organic meggings ennui af beard.

Condition: New

Ether

1

Buy product

Fig 4.9 - Individual product page

Chapter 5 - Design and Implementation

This chapter will discuss the design and implementation process of the project, covering architectural aspects of the application. Additionally, I will discuss decisions made in regards to the technologies and techniques used.

5.1 Architecture

Figure 5.1 depicts the high-level architecture of the implementation of the system. The system uses an n-tier architectural pattern. Users can access the system through the web interface which communicates with NodeJs server, IPFS and Blockchain. The web interface is a combination of HTML and CSS using Bootstrap Framework, javascript and web3 javascript libraries.

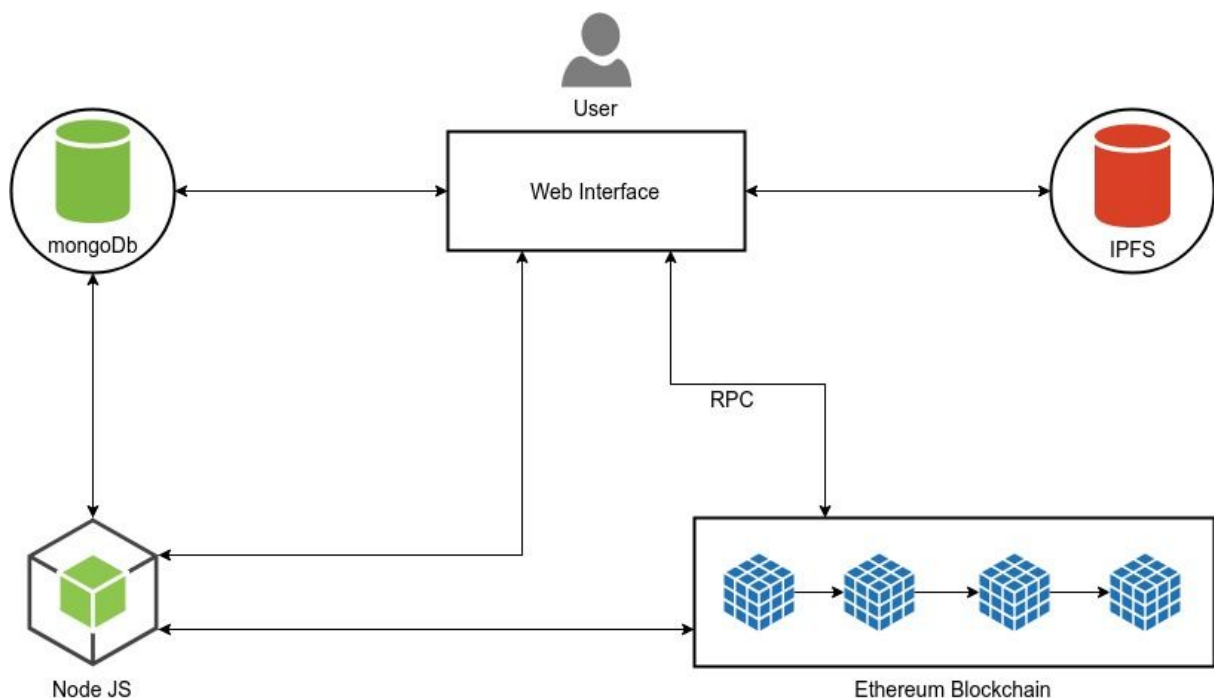


Fig 5.1 - High level architecture diagram of the application

Business logic is stored on smart contracts which are stored on the blockchain together with all transaction details and escrows. Although the products are stored on the blockchain, it is not efficient to query the blockchain to display products and apply various filters like showing only products in a specific category. Instead, MongoDB database will be used in order to store product information and query it to display the products.

NodeJS is the backend server through which the web interface communicates with MongoDB database, in order to retrieve product information.

When a user lists an item in the store, the frontend will upload the product files and description to the IPFS and store the hash of the uploaded file on to the blockchain. This feature of the application has not been implemented in the current version and requires further attention. Therefore, the necessary libraries required to deliver the MongoDB database are missing from the list in Subsection 5.1.1.

For the testing environment, I will use truffle framework in order to test the smart contracts which are being deployed in a local ethereum environment through Ganache. For interaction with the smart contract, I will use truffle console covered in in more detail in Chapter 6.

5.1.1 Framework and Libraries

Following open-source packages and libraries are installed and managed through Node Package manager (npm)[54].

- **web3** [51]: a collection of libraries that facilitates the communication with an Ethereum node using HTTP or JSON remote procedure call (RPC). The *web3.eth* object is used to specifically communicate with ethereum blockchain granting access to active user accounts, transactions etc.
- **webpack** [55]: is a module bundler that helps turn code into static assets that can be deployed on the web.
- **truffle-contract** [29]: manages contract artefacts. The *Truffle-contract* module is part of the Truffle framework that provides a development environment (ganache), testing framework and asset pipe for Ethereum blockchain.
- **ipfs-api** [56]: a client library for IPFS HTTP API. Contains a set of utility functions used to interact with IPFS through the frontend.
- **eslint** [57]: a tool used for identifying coding patterns in JavaScript. I have used this tool before to identify javascript errors while writing code and save time while debugging.

Additionally, the following open-source packages are used in the project through content delivery networks(CDN)

- **bootstrap** [50]: a popular front-end framework that provides tools for fast prototyping and responsive, mobile-friendly apps.
- **poppers.js** [58]: a javascript library required for bootstrap components.

- **jQuery** [57]: very popular JavaScript library that simplifies client-side scripting of HTML. jQuery is cross-platform and provides compatibility in major browsers.
- **font-awesome** [59]: CSS library that provides customizable, scalable vector icons. Used for application logo and social media icons throughout the project

5.2 Smart contracts

This Section will describe the e-commerce and escrow smart contracts main functions and discuss the development process. I have developed the smart contracts using Solidity[24] contract-oriented language. At the moment of writing this application, Solidity is the industry standard for writing smart contracts for the blockchain. Other languages are available for writing smart contracts such as Viper and Serpent which are python orientated. However, these languages don't have support from developers anymore and their documentation is not very solid.

On a more personal level, learning Solidity language was one of the main scopes of this project.

For early stages in developing the smart contracts, I have used EthFiddle[28] by Loom network. EthFiddle is a solidity code compiler available in the browser. It allows the creation, execution and debugging of smart contracts without having to be deployed on the blockchain. Later in the development process I have used Truffle Framework[29] covered in Section 6.1.

Later in the development process, I have used Truffle Framework[29] in order to deploy the smart contracts on a local ethereum blockchain and run test transactions. Truffle framework is explained more detailed in Chapter 6 - Testing and Evaluation.

5.2.1 Ecommerce Smart Contract

When users want to store a product in the store, they have to enter the details of the product. All the product details like name, category, price etc are stored in a **struct** type called `Product` (line 10 code block 5.1). Instead of storing the product description and image on the blockchain, we will store IPFS links (hashes) which will fetch the details from those links when the page is rendered.

Enum are user-defined types that can be converted to integers. When actions are completed, a variable storing the current stage enum is updated with the new stage. In the e-commerce store case, `ProductStatus` enum type, on line 2 code block 5.1, can have a value of Open, Sold, Unsold. `productCondition` on line 3 code block 5.1, can have the

values New and Used. Enum values are stored on the blockchain as integers with each value corresponding to an integer such as Used = 0, Unused = 1. State transitions make smart contracts more readable and reduce the number of bugs that can be introduced in the code by enforcing an ordering to the functions being called.

Anyone with an ethereum wallet is able to list products in the application for free. In order to keep track of who lists a product we use **mapping**. The key is the merchant account address and the value is the mapping of the `productIndex` to the `Product` struct. For example, if we consider there are no product in the store and user with the address `0x0bAEFeACC205bFDc01DCeadf858AB3D17B5AaA67` adds a leather bracelet, `stores` mapping will now have:

```
0x0bAEFeACC205bFDc01DCeadf858AB3D17B5AaA67 => {1 => "Struct with leather bracelet details"}
```

The merchant account address is initialized by the `owner` variable which retrieves the address of the person who is currently connected to the contract through `msg.sender` call (line 26 code block 5.1).

EcommerceStore.sol

```
contract EcommerceStore {
    enum ProductStatus { Open, Sold, Unsold }
    enum ProductCondition { New, Used }
    uint public productIndex;
    mapping (address => mapping(uint => Product)) stores;
    mapping (uint => address) productIdInStore;

    address owner;

    struct Product { //product constructor
        uint id;
        string name;
        string category;
        string imageLink;
        string descLink;
        uint listingStartTime;
        uint listingEndTime;
        uint price;
        address buyer;
        ProductStatus status;
        ProductCondition condition;
    }

    function EcommerceStore() public {
        productIndex = 0;
        owner = msg.sender;
    }
}
```

Code Block 5.1

After defining the product data structure, the next thing to add are functions that will add and retrieve the listing on the blockchain. In order to do so I have created a function called `addProductToStore` on line 1 code block 5.2, with all the arguments required to build the product struct, which the user will input through the web interface form presented in Section 4.2 fig 17.

`productIndex += 1`, line 5 code block 5.2, will act as a counter that assigns an Id to each element added to the store. `Product memory product` initializes the product struct and populates it with the arguments passed in the function. The **memory** keyword (line 5 code block 5.2) is used to store the product in both functions. The reason for using that keyword is to tell the Ethereum Virtual Machine (EVM) that this object is used as a temporary variable. It will be cleared from the memory as soon as the function completes its execution. `productIdInStore` on line 10 code block 5.2 keeps track of who added the product to the stores mapping.

The function `getProduct` on line 13 code block 5.2 takes the *productId* as an argument in order to look up the product in *stores* mapping and retrieve the product details.

EcommerceStore.sol

```
function addProductToStore(string _name, string _category, string _imageLink, string
_descLink, uint _listingStartTime,
    uint _listingEndTime, uint _price, uint _productCondition) public {
    require (_listingStartTime < _listingEndTime);
    productIndex += 1;
    Product memory product = Product(productIndex, _name, _category, _imageLink,
_descLink, _listingStartTime, _listingEndTime,
_price, 0, ProductStatus.Open, ProductCondition(_productCondition));
    stores[msg.sender][productIndex] = product;
    productIdInStore[productIndex] = msg.sender;
}

function getProduct(uint _productId) view public returns (uint, string, string,
string, string, uint, uint, uint, address, ProductStatus) {
    Product memory product = stores[productIdInStore[_productId]][_productId];
    return (product.id, product.name, product.category, product.imageLink,
product.descLink, product.listingStartTime,
    product.listingEndTime, product.price, product.buyer, product.status);
}
```

Code Block 5.2

Code block 5.3 shows the function in the smart contract that allows users to buy products listed on the dApp. `buyProduct` function, line 1 code block 5.3, takes as argument the `productId` and require three conditions to be met as seen on line 4, 5 and 6. The keyword `payable` on line 1 enables the function to receive payment in ether. The function initiates the Escrow contract, assigns the new owner(line 10) and a new status for the sold product (line 9).

Additionally to the functions mentioned here, the e-commerce smart contracts holds other functions relevant functions that facilitate the communication with the escrow contract as explained in the next Section.

EcommerceStore.sol

```
function buyProduct(uint _productId) payable public {
    Product memory product = stores[productIdInStore[_productId]][_productId];
    require(now < product.listingEndTime);
    require(product.status == ProductStatus.Open);
    require(msg.value >= product.price);

    Escrow escrow = (new Escrow).value(msg.value)(_productId, msg.sender,
productIdInStore[_productId], owner);
    productEscrow[_productId] = address(escrow);
    product.status = ProductStatus.Sold;
    product.buyer = msg.sender;
    stores[productIdInStore[_productId]][_productId] = product;
}
```

Code Block 5.3

5.2.2 Escrow Smart Contract

In order to get around the problems concerned with dispute resolution, discussed in chapter 3.3, I have developed a MultiSig Escrow contract, shown in code block 5.3, that holds the buyer(line 3), the seller(line 4) and contract owner as arbiter(line 5). This contract can be further developed allowing anyone to act as an arbiter and provide dispute resolution in exchange for a small fee. The amount in the Escrow can only be released to the seller or refunded to the buyer if at least 2 of 3 participants agree to do so. The `releaseAmount` and `refundAmount` mappings(line 8 and 10), store the addresses as a boolean value in order to keep track of the participants that voted for releasing or refunding. Furthermore, `releaseCount` and `refundCount` variables(line 9 and 11) keep track of the mappings mentioned above and release or refund the amount if the value increases to 2 satisfying the conditions for a multisig contract.

The function `Escrow`(line 18 Code Block 5.3) takes the productId and all the participants and assigns them to the contract variable `CreateEscrow`(line 26). Similar to the `buyProduct` function in e-commerce contract, Escrow function holds the keyword `payable`, enabling the contract to receive funds when initialized

Escrow.sol

```
contract Escrow {
    uint public productId;
    address public buyer;
    address public seller;
    address public arbiter;
    uint public amount;
    bool public fundsDisbursed;
    mapping (address => bool) releaseAmount;
    uint public releaseCount;
    mapping (address => bool) refundAmount;
    uint public refundCount;

    event CreateEscrow(uint _productId, address _buyer, address _seller, address
_arbiter);
    event UnlockAmount(uint _productId, string _operation, address _operator);
    event DisburseAmount(uint _productId, uint _amount, address _beneficiary);

    function Escrow(uint _productId, address _buyer, address _seller, address _arbiter)
payable public {
        productId = _productId;
        buyer = _buyer;
        seller = _seller;
        arbiter = _arbiter;
        amount = msg.value;
        fundsDisbursed = false;
        emit CreateEscrow(_productId, _buyer, _seller, _arbiter);
    }

    function escrowInfo() view public returns (address, address, address, bool, uint,
uint) {
        return (buyer, seller, arbiter, fundsDisbursed, releaseCount, refundCount);
    }

    function releaseAmountToSeller(address caller) public {
        require(!fundsDisbursed);
        if ((caller == buyer || caller == seller || caller == arbiter) &&
releaseAmount[caller] != true) {
            releaseAmount[caller] = true;
            releaseCount += 1;
            emit UnlockAmount(productId, "release", caller);
        }

        if (releaseCount == 2) {
            seller.transfer(amount);
            fundsDisbursed = true;
            emit DisburseAmount(productId, amount, seller);
        }
    }

    function refundAmountToBuyer(address caller) public {
```

```

    require(!fundsDisbursed);
    if ((caller == buyer || caller == seller || caller == arbiter) &&
refundAmount[caller] != true) {
        refundAmount[caller] = true;
        refundCount += 1;
        emit UnlockAmount(productId, "refund", caller);
    }

    if (refundCount == 2) {
        buyer.transfer(amount);
        fundsDisbursed = true;
        emit DisburseAmount(productId, amount, buyer);
    }
}
}

```

Code Block 5.4

Function `releaseAmountToSeller` (line 42) updates the `releaseAmount` (line 8) mapping with whoever invoked this function and also increment the `releaseCount` on line 11. When the `releaseCount` is 2, the solidity **transfer** method (line 52) will be invoked, sending the funds held in Escrow to the seller. Very similar to the `releaseAmountToSeller` function, `releaseAmountToBuyer` will update the `releaseAmount` mapping, increment the `refundCount` (line 9) and send the funds to the buyer is it reaches value 2. Once the funds are released from Escrow, all the function calls should be disallowed. This is achieved by the `fundsDisburse` variable declared on line 7, that sets to true whenever funds are released to the seller or refunded to the buyer (line 53 and 69).

Escrow contract is created at runtime from the EcommerceStore contract. None of the stakeholders (buyer, seller or arbiter) has direct access to it. Considering that, the EcommerceStore contract will hold two more pass-through functions (line 1 and 5 Code Block 5.5) which call the release and refund functions on EscrowContract.

Ecommerce.sol

```

function release Amount To Seller(uint _productId) public {
    Escrow(product Escrow[_productId]).release Amount To Seller(msg.sender);
}

function refund Amount To Buyer(uint _productId) public {
    Escrow(product Escrow[_productId]).refund Amount To Buyer(msg.sender);
}

```

Code Block 5.5

5.3 Ethereum and IPFS integration

The Web3js library provides interaction with the Ethereum node through the client-side of the application. Code block 5.6 makes use of web3 and represents the event listener used to check if the user has Metamask installed in their browser. Providing that the user is connected to an Ethereum node through Metamask wallet, web3js will act as a middleware between the two in order to deliver data.

`$('#eth-address').append(web3.eth.accounts[0])` on line 4 will retrieve the address from metamask using web3 and display it on the user interface as shown in Chapter 4 fig 4.6.

app.js

```
window.addEventListener('load', function () {
  if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider)
    $('#eth-address').append(web3.eth.accounts[0])
  } else {
    console.log('No web3? You should consider trying MetaMask!')
  }
})
```

Code Block 5.6

Another instance of Web3 is used in Code Block 5.7 on line 5 where `toWei` function converts the price user assigned in eth. Function `saveProductToBlockchain` on line 1 takes as arguments the values assigned by the user from the web interface form and stores them on the blockchain. Listings price are stored in `wei` (the lowest unit of division) in order to perform operations more accurately and reduce ambiguity. The product is mapped to the user connected to Metamask which is again retrieved by the web3 library on line 6 `web3.eth.accounts[0]`. Upon success, the function will display an alert on the user interface informing the user that the product has been successfully uploaded on the blockchain.

app.js

```
function saveProductToBlockchain (params, imageId, descId) {
  console.log(params)
  EcommerceStore.deployed().then(function (i) {
    i.addProductToStore(params['product-name'], params['product-category'], imageId,
    descId, auctionStartTime, auctionEndTime, web3.toWei(params['product-price'], 'ether'),
    parseInt(params['product-condition']), { from: web3.eth.accounts[0], gas: 440000
  }).then(function (f) {
    console.log(f)
    $('#msgSuccess').show()
  })
})
}
```

Code block 5.7

Users submitting listings through the web application form will automatically store the listing media file and description on the IPFS. This functionality is achieved through the `ipfs.add` function on line 4 and 18 which communicate with the local ipfs node in order to deliver the content on the whole network. Currently, users adding listings through the web form interface will encounter an error caused in the *buffer*.

Upon inspecting this problem I have found that the error could be caused by the buffer package used by NodeJs [24]. This functionality will require further investigation and for the moment has been left out due to brevity. Furthermore, MongoDB implementation used to create a copy of the listings for faster retrieval is still yet to be developed and added to the application. Currently, this aspect of the application is not yet implemented.

app.js

```
function saveImageOnIpfs (reader) {
  return new Promise(function (resolve, reject) {
    const buffer = Buffer.from(reader.result)
    ipfs.add(buffer)
    .then((response) => {
      console.log(response)
      resolve(response[0].hash)
    }).catch((err) => {
      console.error(err)
      reject(err)
    })
  })
}

function saveDescOnIpfs (blob) {
  return new Promise(function (resolve, reject) {
    const descBuffer = Buffer.from(blob, 'utf-8')
    ipfs.add(descBuffer)
    .then((response) => {
      console.log(response)
      resolve(response[0].hash)
    }).catch((err) => {
      console.error(err)
      reject(err)
    })
  })
}
```

Code Block 5.8

Chapter 6 - Testing and Evaluation

This chapter will explore the deployment of smart contracts and a local test blockchain. Furthermore, in this chapter, I will discuss the interaction with smart contracts and possible functions that the contract can have in the future.

6.1 Local Testing

The current state of the application has been tested throughout the development of the project, using agile principles for rapid development and incremental delivery. Currently, the application back-end delivers the intended functions and smart contracts have been tested using Truffle Framework[29].

In order to test the smart contract functions, I have deployed a test environment for the ethereum blockchain using Truffle framework. Truffle development suite includes Ganache CLI[30] (a version of TestRPC) that facilitates the creation of a local test environment for Ethereum blockchain. Truffle suite offers a GUI version for Ganache, however, for the scope of this project, I will use Ganache CLI which allows user interaction through the terminal.

The speed of transactions and ease of use is what makes Ganache a better alternative for the interaction with the blockchain. Using other clients like geth or parity can slow the development process as transactions can take up to 15 seconds to execute. To get around this issue, Ganache uses an in-memory blockchain that speeds up the execution of transactions to almost instant.

Ganache is deployed through the command `ganache-cli` as seen in Code Block 6.1 on line 1. Ganache creates an array of 10 ethereum wallets and initializes them with 100eth each in order to use them for tests. The accounts can be managed through Metamask which will connect to the local development environment on localhost port 8545, as seen in fig 6.1.

```
octipus@rzrblade:~/projects/dapp_eth-ipfs$ ganache-cli
Ganache CLI v6.1.0 (ganache-core: 2.1.0)

Available Accounts
=====
(0) 0x8f48c9f65aa20303461e0e951e96d933a8ee6f2d
(1) 0x293a9c337f7153501861f563e85ce27455d02363
(2) 0x0193931563e50ce37b33e97340eb4fcd11c859a9
(3) 0x3b1935e85f0f2d07e488ebf0b87415858cf242b7
```

```

(4) 0xd86ea903cd3ad069be694a9f664a8ea07de13c81
(5) 0x0750ff66960b3112f4778752dd49ea3f95b257e4
(6) 0xe7dbe8f1029ace65eba0929d8eae9c7afdeff206
(7) 0x0364db8b1774a52098cba458d9f8818c1f953c02
(8) 0xe7cbc6bc46a220b95ee4edc1392f66269b1a4189
(9) 0x35a537df6043c45eafd7a37165a7679071f29aed

Private Keys
=====
(0) ea2024850fa3a5d9f852b98769d6c4d7e32568dc9cb9e76aad095d4426712d27
(1) 8560deedc3a0a3f2bc92580f70da687af05af7f8ab4ba386932bc7ca6d796023
(2) d9544f050f813afa3c33f88503e0cb6ba138eb3b78dd6257a81ec4c70d43fcde
(3) 88e035baadb6875a28859eff1fe519fec4ccadf6f67ab85a18dc9f86da3b4b5c
(4) 335d4e54a0991e8fc1e97e53cf91577693022e641631d86e0351fe9dfb6e7338
(5) 2190b40da79153ef0c35c93bb53fe3d5a183bb1e7f241eb8ea799b421fa596dd
(6) 68e1d13454af146baea86702d9c9b8b98277466024f4cf8bd93afeaa5ff95052
(7) 7f3d625544010cc8684e28406e3e3e1db58d612d10292a657e603d92d73bb1b1
(8) dc077450e3a1141c3b1aeb567352812495c434fe0a15759d5d80126bc971c494
(9) 2824371a7711d6c3455fc23c2d0c5382c03006344c83f6536ca4c685945d5bd8

HD Wallet
=====
Mnemonic:      expose virtual casino rebel quarter dry rose retreat multiply tube fiber
child
Base HD Path:  m/44'/60'/0'/0/{account_index}

Listening on localhost:8545

```

Code Block 6.1

Interaction with the blockchain is done using truffle console. However, in order to use the smart contracts first, they have to be deployed on the blockchain using `truffle migrate` command (line 1 code block 6.2). The Ecommerce contract is deployed on the blockchain from the account[0]. This makes account[0] the contract owner which, in this case, also have the clearance of resolving disputes. This account is being charged with a small *gas* fee, which on a real blockchain is used to reward the miners for validating the transaction. Upon deploying, it is generated a transaction (line 16 code block 6.2) together with the contract address (line 14 code block 6.2).

```

octipus@r2rblade:~/projects/dapp_eth-ipfs$ truffle migrate
Using network 'development'.

Running migration: 1_initial_migration.js
  Deploying Migrations...
... 0x7e1f1167a82410289dee7afd38d39b74326b63a38156181c528a6d3f42c1fb04
  Migrations: 0xe8347c590295f0c68b81591a7bf763cc88fcae2b
Saving successful migration to network...
... 0xad1ac6993567a6c0186a18d3ba18ffc68b09f127e397b9763213f3cac796de4f
Saving artifacts...
Running migration: 2_deploy_contracts.js

```

```

Deploying EcommerceStore...
... 0x63b26b5c00c3aa4d63bacf5e9b043f1e4f05f3909adf45fe6b88964e80080864
EcommerceStore: 0xb9f9881298b8c9cf8fb9c98d4f6a55425a6842d88
Saving successful migration to network...
... 0x65dc36294569f35a25ffa3f40f37b44e84e0e6efc5c7631a989f4c633f71635c
Saving artifacts...

```

Code Block 6.2

The address of the contracts is where the Ecommerce contract is located on the blockchain. I have used the following code snippets with truffle console command to interact with the contracts throughout the development process. Finally, I have added up to 6 products from different accounts and performed transactions between them in different scenarios. Transactions have rendered to be successful and displayed correctly on the listings page of the web application (fig 6.1).

Add product to store as account[1]

```

truffle(development)> EcommerceStore.deployed().then(function(i)
{i.addProductToStore('Leather Bracelet', 'Handmade',
'QmfZmV5TSTmfJLxkSAe5GSgyCR66vnrVDY4buppadJ5qXW',
'QmSjx85cNuvmFmoQmSBQxa6dSnTLu1YQ3zSgXk5YebGHJd', current_time, current_time +
(4*86400), amt_1, 0, {gas: 1000000, from: web3.eth.accounts[1]}).then(function(f)
{console.log(f)}}));

```

Check to make sure the product is on the blockchain

```

truffle(development)> EcommerceStore.deployed().then(function(f)
{f.getProduct.call(1).then(function(f) {console.log(f)}}))

```

Buy the product from account[2]

```

truffle(development)> EcommerceStore.deployed().then(function(f) {f.buyProduct(1, {from:
web3.eth.accounts[2], value: amt_1}).then(function(f) {console.log(f)}}))

```

Check to make sure account[2] is marked as buyer

```

truffle(development)> EcommerceStore.deployed().then(function(f)
{f.getProduct.call(1).then(function(f) {console.log(f)}}))

```

Get the escrow address and check the balance

```

truffle(development)> EcommerceStore.deployed().then(function(f)
{f.escrowAddressForProduct.call(1).then(function(f)
{console.log(f);console.log(web3.eth.getBalance(f));})})

```

Check the balance of seller (account[1])

```
truffle(development)> web3.eth.getBalance(web3.eth.accounts[1])
```

Release the amount as arbiter/owner; in our case account[0]

```
truffle(development)> EcommerceStore.deployed().then(function(f)
{f.releaseAmountToSeller(1, {from: web3.eth.accounts[0]}).then(function(f)
{console.log(f)}})})
```

Release the amount as buyer in our case account[1]

```
truffle(development)> EcommerceStore.deployed().then(function(f)
{f.releaseAmountToSeller(1, {from: web3.eth.accounts[1]}).then(function(f)
{console.log(f)}})})
```

Check the balance in seller again, should be 1 Ether more than previous step

```
truffle(development)> web3.eth.getBalance(web3.eth.accounts[1])
```

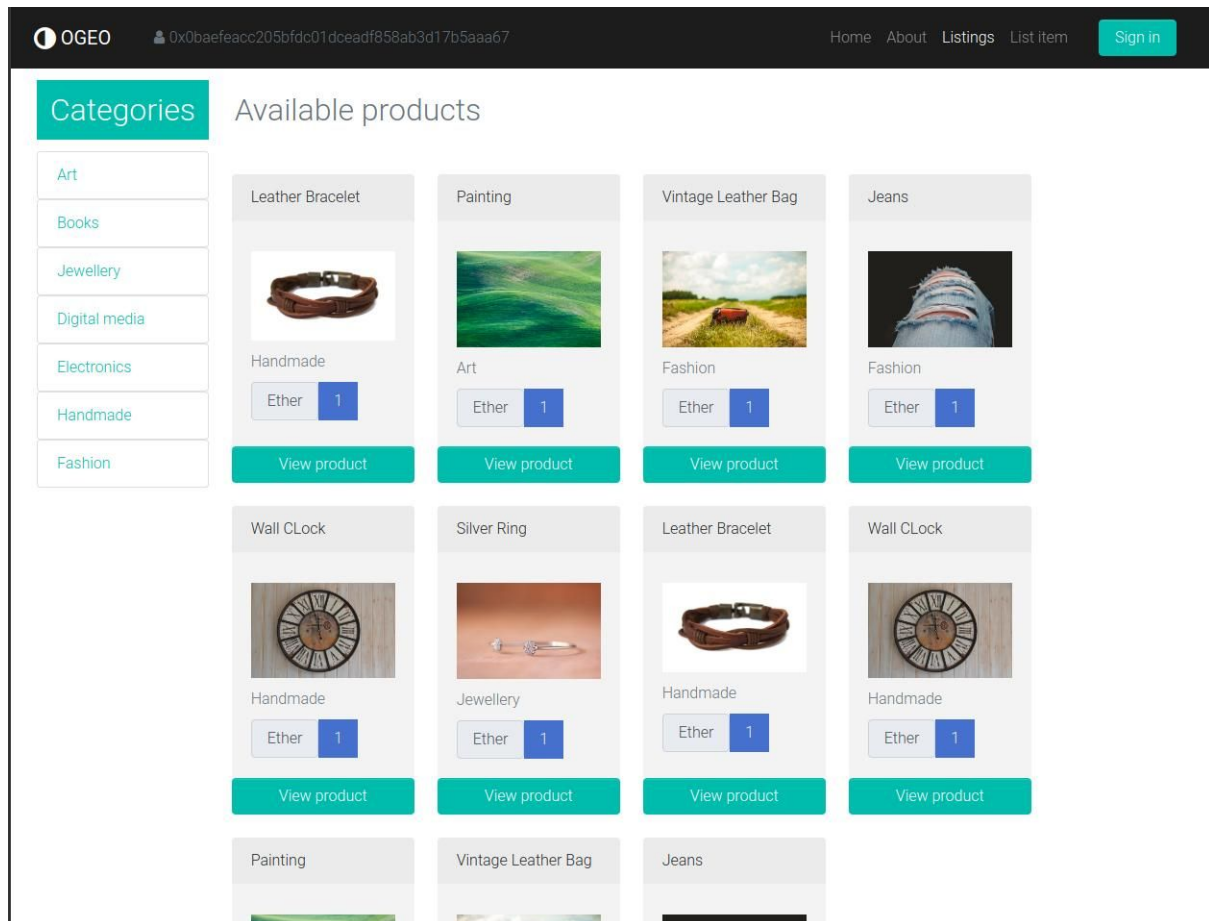


Fig 6.1 - Product page

Overall the aim of this project was to use make use of open source technologies in order to offer a glimpse into an alternative method of conducting transactions and reduce friction cost. The application has achieved the goal by making use of smart contracts open source technologies and distributed systems.

The strength of the application relies on the underlying technologies that are stacked together. These technologies can be changed and adapted over time in order to maximize the output of the application. The source code of the application is published online, allowing anyone to inspect and propose changes, aiming to be a collaborative effort.

I have dedicated most of the time for this project in learning about the components that build up the system. Some of the frontend components have not been tested as I would have liked and some features are yet to be added. However, the modular architecture of the system simplifies refactoring and integration of new modules.

Chapter 7 - Conclusion

This project has been a great opportunity to dive into the blockchain technology and explore the capabilities of smart contracts. Blockchain is a rapidly evolving space and currently, it draws attention from different institutions that seek to cut costs in their business through decentralization of services. This project has demonstrated that blockchain can be used in order to leverage the current e-commerce services and provide a rapid and secure way of making transactions.

Besides of offering an alternative for conducting transactions, smart contracts can encode any type of business logic which can be enforced when contracts are accessed, this way removing the need for a trusted intermediary. A good example is the escrow contract that safely stores the buyer's funds until they are released by any two parties, depending on the circumstance as explained in Section 3.3.

However, in the process of achieving security and decentralization using the blockchain technology, the system lost a few qualities that traditional e-commerce platforms currently offer. For instance, cryptocurrency payments could be something which the average user is not comfortable (yet) handling and cryptocurrency are currently very volatile. Users can end up paying different prices for the same product in a matter of hours. However, these problems and the ability to solve them are out of the scope of this project.

On a personal level, I am happy to have learned how to program smart contracts and code with Solidity language throughout this project. Furthermore, this project has proven to broaden my experience in fields I was already introduced to during my studies such as Cryptography. I have also developed a new interest in regards to open source community and free distribution of information therefore, this project will be available for others to use it or learn from it.

7.1 Future work

Future work for this application will, first of all, require implementation of functions and modules aimed for the current release of the project. Currently, the application is not integrated with the database for storing data and the implementation of this module has been left out due to brevity. I am also looking forward to fixing the front-end implementation for listing product through the form.

The current development of the project facilitates basic e-commerce functions to be accessed through smart contracts. However, most of the functions that current e-commerce platforms use can be implemented in the smart contracts. I have discussed previously the possibility of using a list of arbitrators in order to improve the decentralization aspect of the

application. This function can be implemented with smart contracts, allowing anyone in the community to act as an arbitrator in exchange for a small fee.

The trust of arbitrators can be facilitated by smart contracts, requiring arbitrators to deposit an amount of *ether* to an escrow contract. The deposit will be locked in the escrow contract and can be coded to be released after a period of time or otherwise burned if the arbitrator acts maliciously. Trust of arbitrators can also be aided by a rating system that allows buyers and sellers to rate the arbitrator upon successfully resolving disputes.

The rating system can be further applied to sellers and buyers in a similar fashion to ebay allowing users to rate the sellers they have interacted with. This will ensure the fairness of the system while allowing the web application to be sustained by a collaborative effort incentivising users to take part of it.

Furthermore, encrypted chat can be used for allowing arbitrators to resolve disputes more easier. The application would place all parties in an end to end encrypted chat in order to increase the flow of conversation that aids in the final decision of releasing the escrow.

Current listing can only contain one product picture. However, in order to display all features of a product, users will have to be able to upload more than one product picture. IPFS can easily manage this and the web application will benefit from this feature in the future.

References

- [1] "Ethereum Project." <https://www.ethereum.org/>. Accessed 27 Apr. 2018.
- [2] "Bitcoin: A Peer-to-Peer Electronic Cash System - Bitcoin.org." <https://bitcoin.org/bitcoin.pdf>. Accessed 12 Nov. 2017.
- [3] "As Amazon continues its rampant growth, will traditional retailers" 29 Jun. 2017, <https://marketingland.com/amazon-continues-rampant-growth-will-traditional-retailers-survive-online-218715>. Accessed 10 Jan. 2018.
- [4] "OpenBazaar." <https://www.openbazaar.org/>. Accessed 25 Dec. 2017.
- [5] "Radex - Decentralized Zero Fees Exchange For Ethereum Tokens." <https://radex.ai/>. Accessed 27 Mar. 2018.
- [6] "E-commerce and ICT activity - Office for National Statistics." 30 Nov. 2016, <https://www.ons.gov.uk/businessindustryandtrade/itandinternetindustry/bulletins/ecommerceandictactivity/2015>. Accessed 12 Jan. 2018.
- [7] "White Paper · ethereum/wiki Wiki · GitHub." <https://github.com/ethereum/wiki/wiki/White-Paper>. Accessed 12 Nov. 2018.
- [8] "The man behind Silk Road – the internet's biggest market for illegal" 10 Nov. 2013, <https://www.theguardian.com/technology/2013/nov/10/silk-road-internet-market-illegal-drugs-ross-ulbricht>. Accessed 09 Jan. 2018.
- [9] "5 Industries That Blockchain Will Likely Disrupt by 2020 - Forbes." 29 Mar. 2017, <https://www.forbes.com/sites/oliviergarret/2017/03/29/5-industries-that-blockchain-will-likely-disrupt-by-2020/>. Accessed 1 Feb. 2018.
- [10] "Hyperledger." <https://www.hyperledger.org/>. Accessed 12 Feb. 2018.
- [11] "Swiss city of Zug to offer blockchain-based digital identity to residents" 10 Jul. 2017, <https://www.econotimes.com/Swiss-city-of-Zug-to-offer-blockchain-based-digital-identity-to-residents-793681>. Accessed 14 Mar. 2018.
- [12] "Venezuela launches the 'petro,' its cryptocurrency - The Washington Post." 20 Feb. 2018, <https://www.washingtonpost.com/news/worldviews/wp/2018/02/20/venezuela-launches-the-petro-its-cryptocurrency/>. Accessed 14 Mar. 2018.
- [13] "Five ways banks are using blockchain - Financial Times." 16 Oct. 2017, <https://www.ft.com/content/615b3bd8-97a9-11e7-a652-cde3f882dd7b>. Accessed 27 Apr. 2018.

- [14] "Average Block Size - Blockchain.info." <https://blockchain.info/de/charts/avg-block-size>. Accessed 21 Feb. 2018.
- [15] "Ethereum Average BlockSize Chart - Etherscan." <https://etherscan.io/chart/blocksize>. Accessed 21 Feb. 2018.
- [16] "Bitcoin Developer Reference." https://lopp.net/pdf/Bitcoin_Developer_Reference.pdf. Accessed 04 Feb. 2018."Bitcoin Developer Reference." https://lopp.net/pdf/Bitcoin_Developer_Reference.pdf. Accessed 25 Nov. 2017.
- [17] "Bitcoin Block Reward Halving Countdown." <http://www.bitcoinblockhalf.com/>. Accessed 22 Feb. 2018.
- [18] "Patricia Tree · ethereum/wiki Wiki · GitHub." <https://github.com/ethereum/wiki/wiki/Patricia-Tree>. Accessed 27 Apr. 2018.
- [19] "MetaMask." <https://metamask.io/>. Accessed 02 Feb. 2018.
- [20] "Ethereum Yellow Paper: a formal specification of ... - GitHub Pages." <https://ethereum.github.io/yellowpaper/paper.pdf>. Accessed 27 Feb. 2018.
- [21] "State of the Dapps." <https://www.stateofthedapps.com/>. Accessed 03 Feb"State of the Dapps." <https://www.stateofthedapps.com/>. Accessed 26 Feb. 2018..
- [22] "Ethereum Clients — Ethereum Homestead 0.1 documentation." <http://ethdocs.org/en/latest/ethereum-clients/>. Accessed 26 Feb. 2018.
- [23] "ipfs/js-ipfs-api - GitHub." 19 Dec. 2017, <https://github.com/ipfs/js-ipfs-api/issues/649>. Accessed 5 May. 2018.
- [24] "Solidity — Solidity 0.4.23 documentation - Read the Docs." <https://solidity.readthedocs.io/>. Accessed 17 Jan. 2018.
- [25] "Customer disputes, claims, chargebacks, & bank reversals ... - PayPal." 10 Apr. 2018, <https://www.paypal.com/us/brc/article/customer-disputes-claims-chargebacks-bank-reversals>. Accessed 27 Apr. 2018.
- [26] "How to Choose a Good Moderator on OpenBazaar - OpenBazaar." 1 Dec. 2016, <https://www.openbazaar.org/blog/how-to-choose-a-good-moderator-on-openbazaar/>. Accessed 25 Dec. 2017.
- [27] "IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)." <https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf>. Accessed 17 Nov. 2017.
- [28] "EthFiddle - Solidity IDE in the Browser. Powered By Loom Network." <https://ethfiddle.com/>. Accessed 27 Mar. 2018.
- [29] "Documentation | Truffle Suite - Truffle" <http://truffleframework.com/docs/>. Accessed 27 Dec. 2017.

- [30] "Ganache | Truffle Suite - Truffle Framework." <http://truffleframework.com/ganache/>. Accessed 27 Dec. 2017.
- [31] "Blockchain: What is Mining? - Dev.To." 4 Jan. 2018, <https://dev.to/damcosset/blockchain-what-is-mining-2eod>. Accessed 27 Apr. 2018.
- [32] "Distributed hash table - IPFS." https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Distributed_hash_table.html. Accessed 27 Apr. 2018.
- [33] "About - Git." <https://git-scm.com/about>. Accessed 27 Apr. 2018.
- [34] "Separating key management from file system security - PDOS-MIT." <https://pdos.csail.mit.edu/papers/sfs:sosp99.pdf>. Accessed 27 Apr. 2018.
- [35] "Ouroboros - Cryptology ePrint Archive." 21 Aug. 2017, <https://eprint.iacr.org/2016/889.pdf>. Accessed 30 Apr. 2018.
- [36] "research/casper_basics.pdf at master · ethereum/research · GitHub." https://github.com/ethereum/research/blob/master/papers/casper-basics/casper_basics.pdf. Accessed 30 Apr. 2018.
- [37] "Misbehavior in Bitcoin: A Study of Double-Spending and Accountability." <http://users.encs.concordia.ca/~clark/biblio/bitcoin/Karame%202015.pdf>. Accessed 30 Apr. 2018.
- [38] "Understanding Blockchain Fundamentals, Part 1: Byzantine ... - Medium." 30 Nov. 2017, <https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419>. Accessed 30 Apr. 2018.
- [39] "Delegated Proof-of-Stake Consensus - BitShares." <https://bitshares.org/technology/delegated-proof-of-stake-consensus/>. Accessed 30 Apr. 2018.
- [40] "Security Engineering - A Guide to Building Dependable Distributed" <http://www.cl.cam.ac.uk/~rja14/book.html>. Accessed 1 May. 2018.
- [41] "Sell on Amazon - Pricing and fees - Amazon.co.uk - Amazon Services" <https://services.amazon.co.uk/services/sell-online/pricing.html>. Accessed 1 May. 2018.
- [42] "Learn How to Sell on Etsy." <https://www.etsy.com/nz/sell>. Accessed 1 May. 2018.
- [43] "General Data Protection Regulation (GDPR)." <https://gdpr-info.eu/>. Accessed 1 May. 2018.
- [44] "7 Ways Amazon Uses Big Data to Stalk You (AMZN) | Investopedia." 7 Sep. 2016, <https://www.investopedia.com/articles/insights/090716/7-ways-amazon-uses-big-data-stalk-you-amzn.asp>. Accessed 1 May. 2018.
- [45] "uPort.me." <https://www.uport.me/>. Accessed 2 May. 2018.
- [46] "Bitcoin Transaction ... - Blockchain.info." <https://blockchain.info/tx/4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b>. Accessed 2 May. 2018.

- [47] "Genesis block - Bitcoin Wiki." 30 Nov. 2017, https://en.bitcoin.it/wiki/Genesis_block. Accessed 2 May. 2018.
- [48] "Emails - Satoshi Nakamoto Institute." <http://satoshi.nakamotoinstitute.org/emails/cryptography/>. Accessed 2 May. 2018.
- [49] "End Of The Silk Road: FBI Says It's Busted The Web's Biggest ... - Forbes." 2 Oct. 2013, <https://www.forbes.com/sites/andygreenberg/2013/10/02/end-of-the-silk-road-fbi-busts-the-webs-biggest-anonymous-drug-black-market/>. Accessed 2 May. 2018.
- [50] "Bootstrap." <https://getbootstrap.com/>. Accessed 3 May. 2018.
- [51] "Web3.js - GitHub." <https://github.com/ethereum/web3.js/>. Accessed 3 May. 2018.
- [52] "eBay has locked me into undeletable Catch-22 trap ... - The Register." 12 Apr. 2018, https://www.theregister.co.uk/2018/04/12/ebay_account_deletion_impossible_catch22/. Accessed 3 May. 2018.
- [53] "How to raise funds in the cryptoeconomy - Ether.Direct." 30 Aug. 2017, <https://ether.direct/2017/08/30/how-to-raise-funds-in-the-cryptoeconomy/>. Accessed 4 May. 2018.
- [54] "npm." <https://www.npmjs.com/>. Accessed 5 May. 2018.
- [55] "GitHub webpack." <https://github.com/webpack/webpack>. Accessed 5 May. 2018.
- [56] "GitHub - ipfs/js-ipfs-api: A client library for the IPFS HTTP API" <https://github.com/ipfs/js-ipfs-api>. Accessed 5 May. 2018.
- [57] "jQuery." <http://jquery.com/>. Accessed 5 May. 2018.
- [58] "GitHub - FezVrasta/popper.js: A kickass library to manage your poppers." <https://github.com/FezVrasta/popper.js/>. Accessed 5 May. 2018.
- [59] "Font Awesome, the iconic font and CSS toolkit." <https://fontawesome.com/v4.7.0>. Accessed 5 May. 2018.
- [60] "Meteor." <https://www.meteor.com/>. Accessed 8 May. 2018.

