Final Year Project Report

A Blockchain-based Rating and Review Management System for eCommerce

Brian McDermott

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: John Murphy

UCD School of Computer Science
University College Dublin

April 8, 2019

Project Specification

Core:

Design and develop a blockchain-based system which can be used to store and manage the rating and review records generated by an online shopping platform.

Advanced:

Retrieve and visualize review data stored on the blockchain. For example, simple statistical analysis, such as, ratings received in the last week, etc.

Abstract

The value of e-commerce is growing year after year. With evidence that reviews and ratings are heavily influencing customer purchases, reviews are clearly a very valuable asset. E-commerce sites have total control over the content and visibility of the review data in their private databases. There exist the means, motive and opportunity to fake, edit or even delete review and rating data. From the consumer perspective, this is a problem of trust. How can they trust that they are reading authentic, unmodified reviews?

In order to demonstrate the ability of the blockchain to provide robust and reasonable solutions to this problem of trust, a decentralised application was developed on the Ethereum blockchain, to manage the rating and review data of the sort generated by an e-commerce site.

Acknowledgments

I would like to thank my Supervisor Dr John Murphy, and my Mentor Dr Long Cheng for their help and guidance throughout the project.

Table of Contents

Chapter 1: **Introduction**

1.1 Context

The value of e-commerce globally is expected to hit 4 trillion dollars by 2020[20]. Studies have found evidence that customer reviews have a direct influence on whether or not a consumer buys a product. This makes reviews a very valuable asset. It also makes reviews an obvious target for fraud. This project centers around a blockchain-based solution to a problem of trust. The main aim of this project was to design and implement a rating and review management system for e-commerce. In practical terms, the task was to store and retrieve data to and from the blockchain. To this end, a decentralised application (often called 'dApp' for short) was developed on the Ethereum blockchain. This application provides the user with a simulated e-commerce experience in which they may select and purchase an item. An invitation to review the item purchased is sent via email/SMS to the purchaser of the item. The method used ensures that only the verified purchaser is sent an invitation to review. The invitation received contains a URL at which a review and rating may be submitted. Once submitted, the review is stored on the blockchain immutably. All reviews can be accessed from the application by visiting the reviews page for a particular product. These are retrieved from the blockchain on demand.

The Ethereum blockchain is public and therefore all data is viewable by anyone who wishes to see it. This makes for a level of transparency rarely seen on systems that use a traditional database setup. Trust in a third party to ensure the integrity of the database now and in the future, is taken out of the equation, in favour of trust in the technology of blockchain. Trust is therefore no longer a concern.

1.2 Report Structure

Chapter 2 lays out the background research conducted for this project and gives an overview of the technologies and concepts that underpin blockchain. These are important for an appreciation of the potential of blockchain and its applications, as well as their relevance to the project. There is discussion about blockchain. This includes why it is a good choice for this project, and why Ethereum in particular was chosen as a development platform, along with some advantages and disadvantages over a traditional database setup. Finally, there is a brief consideration of some existing blockchain-based rating and review systems. Regarding the subjects researched, each is a large area in its own right, and space is finite, so efforts have been made to strike an appropriate level of detail. An attempt has been made to avoid explaining concepts which were not explicitly researched for the project. Where concepts are explained, it is because research was required to understand the topic.

Chapter 3 discusses the design of the decentralised application and decisions made during this process. The high-level system architecture is provided, along with architecture for an Ethereum decentralised application. Mock-ups for the graphical web front-end are displayed and a discussion of the smart contract design is also given, including an alternative design that was considered. The chapter concludes with thoughts on how this system might be funded. Every effort has been made to cover the design topics in the order in which they were dealt with during the project, with a view to providing a clear window to the design process.

Chapter 4 consists of an explanation of the technologies that were required to create the application, and how they were used. These technologies are not fundamental to blockchain or Ethereum itself, but are critical to development of applications on Ethereum. The testing and evaluation of the application is also discussed in this chapter, since testing was a central and constant activity during development. Chapter 5 provides some conclusions on the application as a solution. Weaknesses of the the system are considered along with possible solutions. Some candidates for possible future work are also suggested.

1.3 Overview of project life-cycle

1.3.1 First steps

The initial aim in this project was to undertake research into blockchain technology, and its fundamental components. This was important in order to understand why blockchain could offer a suitable solution to the problem at hand and how a blockchain based solution differed from other traditional database solutions. It was also important that development would begin only after a solid basic understanding of the underlying technology had been acquired. This seemed like a natural first step. Research involved informational and tutorial materials from many different literary, scholarly and media sources. This phase also involved looking at the current role of review and rating data in e-commerce, and the issues that surround this data at present. The findings and knowledge acquired from this initial phase are detailed in Chapter 2: Background Research.

1.3.2 The project core

A decentralised application was designed and developed on the Ethereum platform to manage rating and review data of the sort generated by an e-commerce platform. This was in line with the core objective of the project. In order to acquire the skills and experience necessary to complete this objective, it was necessary to learn to use a lot of new tools, and to become familiar with both the Javascript and Solidity languages. While Javascript is widely known, Solidity is a less known language used specifically for writing smart contracts on Ethereum. The learning process involved becoming familiar not just with blockchain and the Ethereum platform in particular, but also with web development using Javascript. Specifically, a library called React was used to create a web front-end for the application. Development of the application was a very challenging and rewarding experience. Much was learned in the process. Integrating Ethereum and the front-end technologies was also a very interesting and challenging experience which has opened the door to new possibilities not considered before.

The advanced objective was to visualise the data stored on the blockchain. The completion of this objective effectively demonstrates the completion of the core objective. The application allows the user to view the reviews stored for a product on the blockchain, both in a classic listed form, and in the form of a graph which simply shows review data by volume for a particular product over chosen time periods.

Chapter 2: **Background Research**

2.1 Introduction

This chapter will lay out the background research conducted for this project and give an overview of the technologies and concepts that underpin blockchain. In order to understand the implementation fully, it is important to understand the technologies that make it possible. Blockchain as a technology, is an amalgamation of various other technologies, most of which are very large and detailed subjects in their own right. It is, however, important to elucidate these technologies, to acknowledge their importance to blockchain. This will aid in an appreciation of the power and potential of blockchain-based software solutions, and how such a solution benefits this project.

There is discussion on the reasons for choosing blockchain and Ethereum in particular as a development platform. This includes some advantages and disadvantages of blockchain-based systems versus systems relying on a traditional database architecture. Finally, two existing blockchain-based rating and review systems are considered.

As previously stated, an attempt has been made to avoid explaining concepts which were not explicitly researched for the project. Where concepts are explained, it is because research was required to understand the topic. The intention has been to strike an appropriate level of detail that also allows space for all important topics.

2.1.1 Why use Blockchain for this Project?

This is a very pertinent question. Why use the blockchain over a traditional database setup? The problem to be solved here, is one of trust. It is through transparency and immutability of the data stored that this trust is to be established. When making use of traditional databases, it is always the case that at least one individual has administrative access and rights. These allow the modification of the databases contents. In the context of review and rating data, the question is not how likely it is that malicious or fraudulent modification, deletion or addition of this data should occur, but whether it is possible for it to occur. It is important to have the assurance that such actions cannot occur. Blockchain takes trust for an individual or organisation out of the equation entirely, and places it instead in the technology. Since the technology is open-source and transparent in its operations, trust is no longer an issue. Blockchain technology provides a solid foundation on which trust can be fixed, measured and promoted from a concern to an assurance. The following sections aim to expand on these claims with a look at the technologies 'under the hood' of blockchain.

2.2 Cryptography

2.2.1 Cryptographic Hash Functions

Cryptographic hash functions are very central to the integrity of the blockchain. A cryptographic hash function takes input data of any size, and outputs a fixed length string of bytes[1]. This string is called a 'hash-digest' or just 'hash', which uniquely identifies the input data. Input data could be a file, an image, or any other digital data, of any size. Importantly, any time we input the exact same piece of data, we will always get the same output hash[1]. This is because cryptographic hashing algorithms are deterministic[2]. Altering a single bit of the input data will produce a completely different hash.

2.2.2 Public-Key Cryptography

Blockchains use public-key cryptography to encrypt their transactions. Public-key cryptography(asymmetric cryptography) is a system of encryption whereby two cryptographic keys are generated which are mathematically related. One is called the public key, the other, the private key[3]. The public key can be generated using its private counterpart, but the private cannot be attained with the public. For this reason the public key is safe to distribute, and the private key needs to be kept a secret[3]. Any data encrypted with a public key, can only be decrypted with the corresponding private key, and vice-versa. This system can be used for encrypted communications and the creation of digital signatures for identity verification purposes[3].

2.3 Blockchain Technology

2.3.1 Blocks and Chains

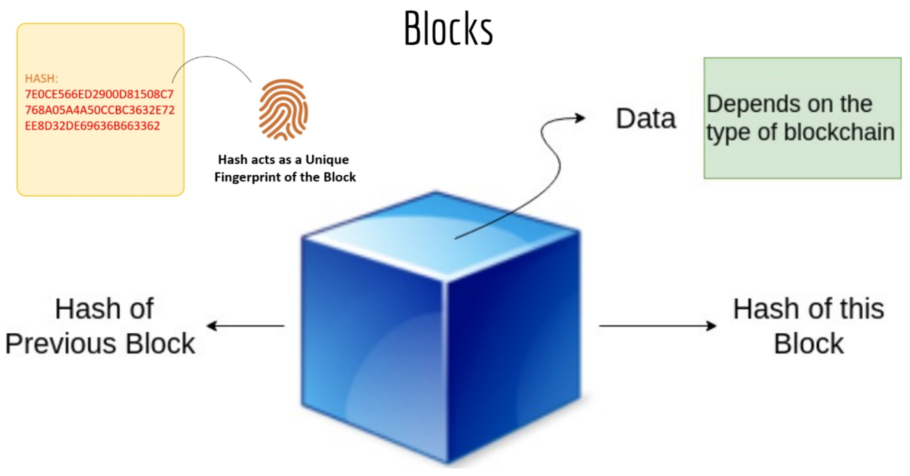


Figure 2.1: The make-up of a block

Blocks are fundamentally a data structure that at the most basic level contain three things as depicted in figure 2.1. Blocks contain data. What kind of data, depends on the type of blockchain[4]. In the context of a crypto-currency, blocks could contain transaction data(such as sender, receiver, amount transferred etc.). In other types of blockchain, the blocks could contain data relating to patients in a hospital, real-estate records, or the criminal records of convicts. The hash of a block uniquely identifies that block, and of course, change one bit of data in the block and its hash changes, as it's now a different block. Blocks also contain the unique hash of the previous block in the chain[4]. It is in this way that blocks can be said to be chained together.

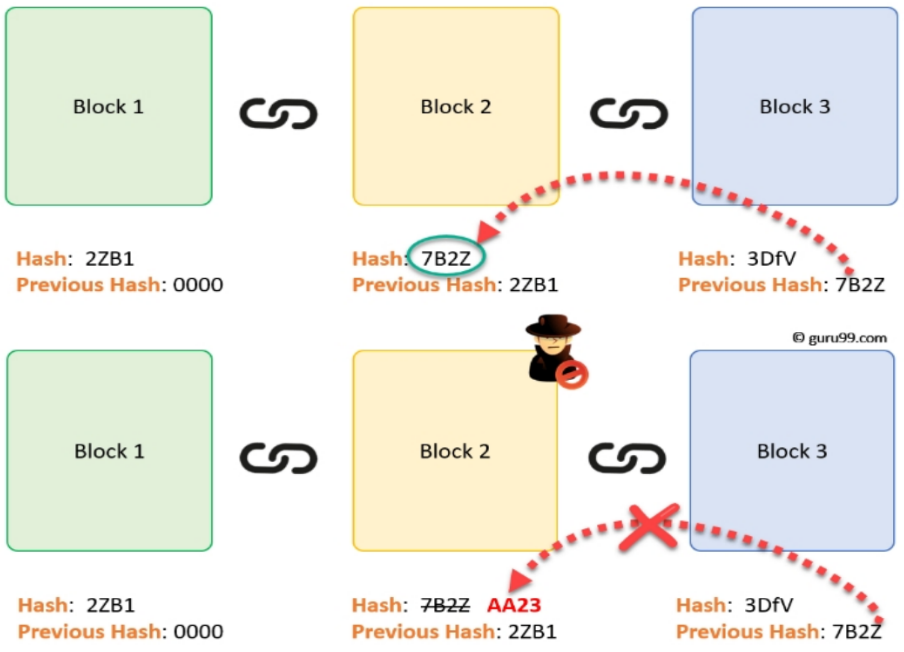


Figure 2.2: Each block in the chain contains the hash of the previous block

Based on the deterministic property of cryptographic hash functions[2], modification to data stored in any one block, will invalidate the hashes in every subsequent block, and break the chain. For instance, if someone wanted to maliciously alter the contents of block 2 in figure 2.2, this would invalidate block 3 and all subsequent blocks. The attacker would have to recalculate the hash for every block in order to reconstitute the chain[4]. But with the computing power currently available, this *should* be possible. However, blockchain has another layer of protection against tampering. It's called 'Proof-Of-Work'.

2.3.2 Proof-of-Work

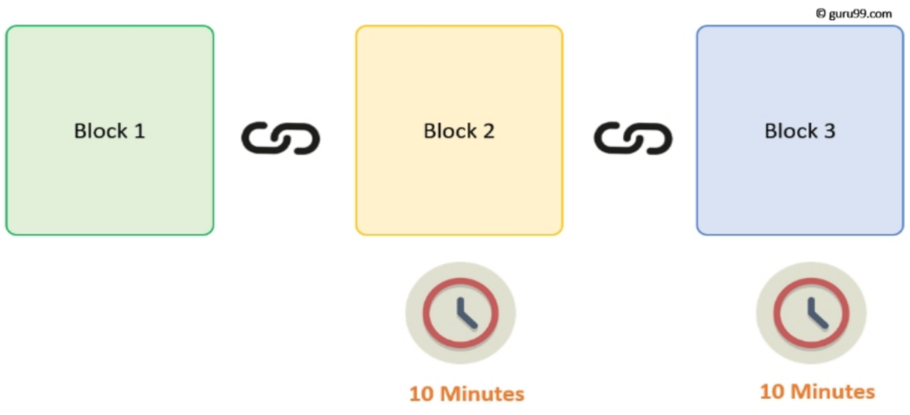


Figure 2.3: Proof-of-Work: A consensus algorithm

Proof-of-Work(PoW) is a hard problem requiring a certain amount of computational effort to solve[9]. It essentially regulates block creation, slowing it down. For example, the average time it takes to create and add a new block to the Bitcoin blockchain is 10 minutes[4]. Looking back at figure 2.2 again, in addition to recalculating the hashes for each block, a malicious actor would have to perform PoW for each and every block they invalidated with their tampering[4]. This becomes a costly undertaking. But that's not all the attacker would be up against, because there is another feature of blockchain that protects it from such tampering.

2.3.3 Peer-to-Peer Networks: Forming Consensus

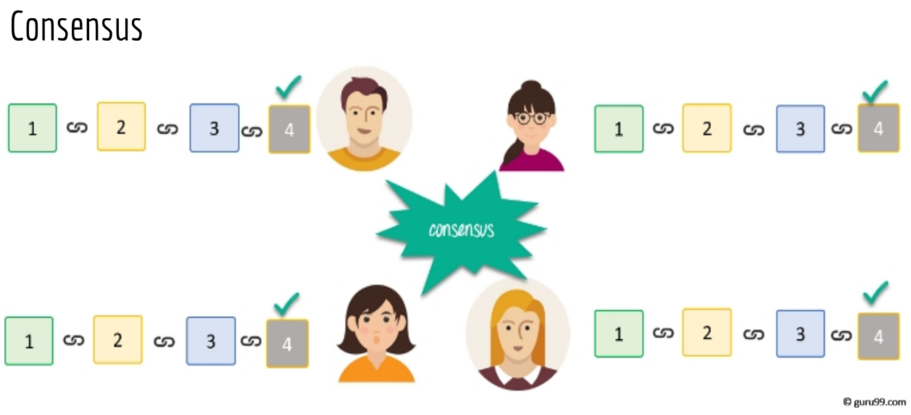


Figure 2.4: Blockchains operate over peer-to-peer networks, which form a consensus between nodes

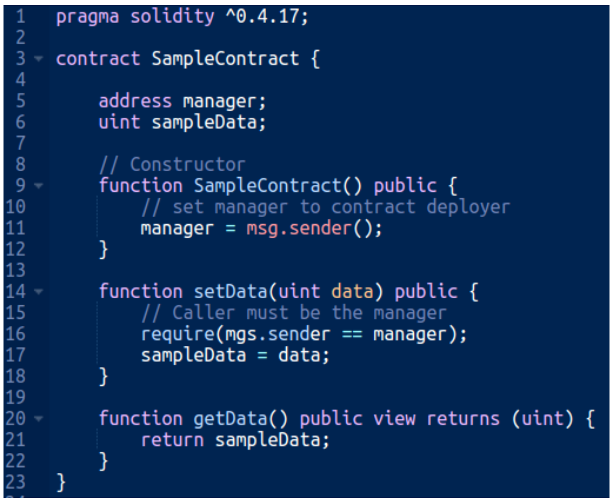
Blockchain is distributed over a peer-to-peer network. Each and every node on the network receives its own full copy of the blockchain[4]. Together, all of these nodes form a Consensus. They agree on what blocks are valid, and which blocks to reject[4]. When a new block is created, it gets broadcast to the network, and each node individually verifies this block. If the node finds the block to be valid, it appends it to its own copy of the blockchain[12]. So in the end, in order to tamper with a block, the attacker would have to redo PoW for all invalidated blocks, altering the contents of each one along the way. On top of this, they would have to control a majority of the network in order to achieve consensus on their tampered version of the blockchain[11]. While not impossible, it is highly improbable that an attacker would have the motivation as well as the resources for such a huge task.

The creation of a new block requires, in the case of Ethereum and Bitcoin, that all transactions in the block are verified. This essentially means making sure that inputs and outputs of a transaction correlate and make sense. Finally the proof-of-work must be performed for the block. Once any particular node has completed PoW for a new block, it broadcasts the block to the network, to be validated by all the other nodes. This process of block creation is referred to as 'mining'[10], and the nodes that take part are called 'miners'[10]. Nodes which mine a new block are rewarded with some tokens(Ether on Ethereum).

2.4 Ethereum

Ethereum is a blockchain technology originally described by Vitalik Buterin in a white paper published in 2013 called "A Next-Generation Smart Contract and Decentralized Application Platform"[24]. A co-founder of Bitcoin Magazine, Buterin argued that Bitcoin needed a better scripting capability for developing applications[24]. Bitcoin does have a very limited scripting language, which is not Turing-complete, and not suitable for creating complex applications. Not receiving much support in his ideas for Bitcoin, Buterin proposed the creation of a new decentralised platform. This resulted in the creation Ethereum[25].

2.4.1 Ethereum Smart Contracts



```
pragma solidity ^0.4.17;

contract SampleContract {

    address manager;
    uint sampleData;

    // Constructor
    function SampleContract() public {
        // set manager to contract deployer
        manager = msg.sender();
    }

    function setData(uint data) public {
        // Caller must be the manager
        require(mgs.sender == manager);
        sampleData = data;
    }

    function getData() public view returns (uint) {
        return sampleData;
    }
}
```

Figure 2.5: A simple smart-contract written in Solidity

One of the main features of Ethereum is support for a Turing-complete programming language, targeted at the Ethereum Virtual Machine(EVM)[25]. Smart contracts are programs that are stored on the blockchain[25]. Due to being deployed on a blockchain, they are immutable. This means the logic of the contract can not be altered in the future. Just like any other data on Ethereum, they are publicly viewable, at any time. This means that functionality is totally transparent. Redeployment of a smart contract is possible, but this results in an entirely new instance of that contract, which is unrelated to any previous deployments (This had been found to be true through experimentation during the project). Any such redeployment is therefore a fully publicly viewable event.

While smart contracts are written in a high-level language called Solidity, it is the compiled byte-code that gets stored on the blockchain[25]. The Solidity files, with extension '.sol' are compiled using the Solidity compiler. The compiler outputs, among other things, both an interface and the byte-code for the blockchain[13]. The interface, called the Application Binary Interface(ABI), is what allows front-end applications to know how to interact with the contract byte-code on the blockchain[13].

2.4.2 Why Ethereum?

There were a number of factors that went into the choosing of a platform on which to develop the rating and review management system. An absolute requirement was that the platform provided the ability to write smart contracts. Two main choices were top of the list from early on. These were Hyperledger Fabric, and Ethereum. Hyperledger Fabric is an open-source, permissioned, distributed ledger technology platform, which is designed for use in enterprise contexts[14]. This technology is hosted by the Linux Foundation. In considering these platforms, both allow for the writing of smart contracts(known as 'chain code' in Hyperledger)[15], but one main difference between the two came to the fore, which clinched the decision.

In order for trust to be established, and maintained, it seemed necessary that rating and review data that is to be stored, should be stored publicly, and immutably, such that transparency is maintained. While it is possible to allow a blockchain system based on Hyperledger Fabric to be public, it is not required. Indeed, this status could possibly be revoked at a later stage, since with Hyperledger, the entire blockchain can be under complete control of some private authority[14]. This is essentially what a permissioned blockchain allows for. In contrast, the Ethereum Main net is public, and that status cannot change. While an individual or organisation can and often should have control over a smart contract, all operations by the contract are publicly viewable. While development of the application would indeed be possible on Hyperledger, ultimately it was decided that Hyperledger was too easily controllable by the creators of the system. Ethereum was therefore the platform of choice. Ethereum is still a relatively new technology. The first Ethereum network was released in July 2015[26], but it has come a long way since, and appeared to be mature enough to create complex and interactive decentralised applications.

2.4.3 Ethereum Accounts

Blockchains such as Ethereum and Bitcoin can be thought of as big state transition machines. The current state of the blockchain is all the current ownerships of all blockchain assets(Bitcoin, or Ether)[25]. State transition functions take a transaction on the blockchain, along with the current state, and output the new state[25]. In Ethereum, the state is made up of Ethereum account objects, each with a unique 20-byte address. State transitions on Ethereum are the transference of data and value among accounts[25].

Two types of account exist on Ethereum. External accounts are owned and operated using private keys(often controlled by an individual) and do not contain any smart-contract code[25]. Contract accounts are governed by their code logic, which is executed whenever messages are received[25]. Both types of account have four attributes[25]:

- A nonce which ensures that a transaction can only occur once.
- The account balance in Ether.
- The contract code(if any).
- Data storage for the account which is empty by default.

Both account types can send messages, sign and create transactions[25]. Contract accounts can read and write to their account storage, and even deploy other contracts. Contracts are not the same as contracts in everyday life. They sit on the blockchain, and execute code on receipt of a message or transaction. They have total control over their own Ether balance and persistent internal storage[25].

2.5 Advantages & Disadvantages of using Blockchain

2.5.1 Traditional Web Application Architecture

There are some fundamental differences between decentralised applications built on Ethereum and web applications built in the traditional way. These heavily influence the design of an application.

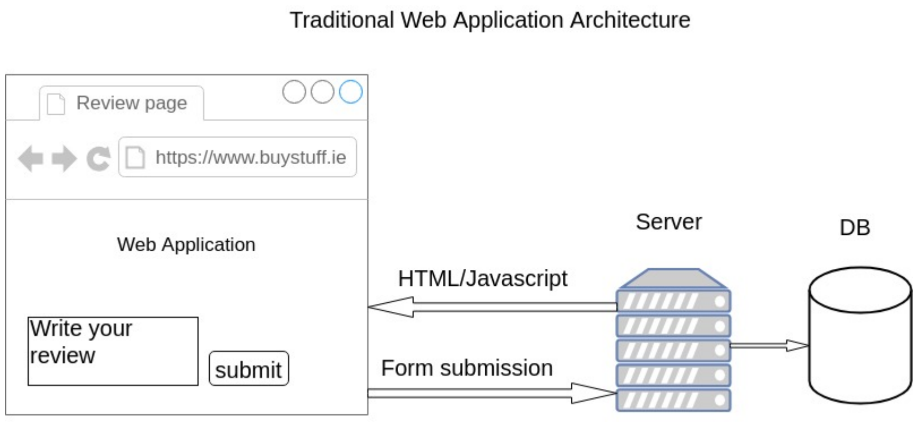


Figure 2.6: Traditional Web App Architecture

In a very basic, but common scenario for a traditional web application, the server stands between the client and a back-end database, processing requests and returning responses. In this classic setup, the server is the portal through which all communication flows. Interactions with the database, such as writing new data, are entrusted to the unseen functionality of the servers back-end code. The user must trust that their data will be stored securely, and unmodified, not just now, but in the future. This can not be guaranteed.

2.5.2 Vulnerability to Hackers & Malware

Traditional databases are controlled by some central authority. Access can be granted or denied based on a clients ability to provide a valid username and password. This system of authentication works in general, but the system itself is always vulnerable compromise. Since the central authority has ultimate control over the database, a hacker or piece of malware that hijacks that control can do whatever they desire with the data. Though backups typically exist for such systems, these too may be compromised, especially if they exist on systems connected to the network.

The news in recent years is awash with headlines about hackers, malware, and ransomware successfully compromising systems of all descriptions. Much of the private user data on the internet is stored in relational databases, which have been demonstrated vulnerable to attack, time and again. Deletion, modification, and simple exfiltration of data by hackers from the traditional database is a current daily issue, severely affecting the market value of companies[16]. This affect my be temporary in many cases[16], but there are no guarantees. Other costs incurred by this onslaught are in reputation and user trust. Having a database that contains private user information compromised by hackers or malware can have potentially devastating consequences for a company, and for its users[16]. Blockchain technology addresses this problem by virtue of its decentralised model and consensus mechanisms.

2.5.3 Robustness

Given that traditional databases are controlled by some central authority, it is this authority that is responsible for the maintenance and indeed the continued existence of the database. Ensuring database availability means its often necessary to have servers in more than one physical location, to mitigate loss of data in a disaster[17]. With a blockchain based system, most of these kinds of concerns are taken care of by the system itself which replaces the need for much of the staff and policies employed to maintain and safe-keep a traditional database system[17]. If some nodes in the blockchain network fail, its not the end of the world, the blockchain is so redundant that there is always another safe copy of the data to be broadcast to any nodes that require it.

2.5.4 Cost

For many organisations, a traditional database type system requires the hiring of special personnel to maintain it, to ensure the security of the organisations data, or to lay out the policies mentioned in the last section, aimed at mitigating disastrous loss of data[17]. Additionally, when data breaches do occur, due to malware or hacking etc., great costs can be incurred in finding and fixing the underlying security holes. A blockchain solution certainly avoids the need for most of the costs incurred by all of the above. Particularly with a public blockchain such as Ethereum, there are very many more eyes combing the open-source technology for vulnerabilities, making them less likely and easier to deal with when they arise.

2.5.5 Disadvantages - Performance & Energy Consumption

In comparison to a traditional database, transactions in a blockchain are encrypted with cryptography, which takes a computational toll. Likewise, algorithms for Proof-of-Work and other consensus are two more impediments to performance that traditional databases dont have to worry about[17].

Some blockchain implementations that involve the mining of blocks with Proof-of-Work, such as Bitcoin, use up very large amounts of energy. One study from 2014, claimed that at the time of writing, the energy consumption by the Bitcoin blockchain was comparable to the electricity consumption of the whole of Ireland[18]. A more recent study in 2017 argued that Bitcoin mining was at that time around 100500 MW[19].

2.6 Blockchain-based Rating & Review Systems

2.6.1 The Role of Reviews in E-commerce

Global e-commerce is thriving, with indications that by the year 2020, it will have become a 4 trillion dollar industry[20]. There is evidence that reviews play a significant role in consumer purchasing behaviour. According to one study, approximately 60% of online consumers look at the reviews of a product before making a decision to purchase[20]. Further studies have shown that good reviews are much more likely to inspire trust in consumers than either a stores reputation or its own assurances of quality[21]. In addition to these, consumers that are new to a particular vendor are likely, according to another study, to consider reviews of the vendor as entirely trustworthy indications of that vendors reliability and benevolence[22]. Textual reviews were shown to have a much greater impact than numerical ratings[22]. This same study found that a full 97% of their subjects read reviews before purchasing a product[22]. A considerable amount of studies have found evidence that reviews influence consumers purchasing behaviour. There is a strong case to be made that reviews are a very valuable asset to e-commerce.

2.6.2 The Problem with Current Review Systems

The e-commerce sites themselves have total control over the content and visibility of the review data submitted to their databases. Taking into account how much e-commerce is now worth, and the evidence that reviews and therefore reputation are quite literally money in this industry, there exists the means, motive and opportunity to fake, edit, or even delete review and rating data. The e-commerce site can also change the algorithm that calculates a reputation without anyone knowing, as eBay did in recent times[23]. How can a consumer be certain that the reviews they are reading are authentic, and have not been tampered with? From the point-of-view of the consumer, this is fundamentally a problem of trust. As has been established previously in this chapter, blockchain can provide solutions to this kind of problem.

2.6.3 Existing Blockchain-based Rating & Review Systems

Two notable blockchain-based rating & review management systems have been created by Revain, and Lina(who call it Lina.review). There are similarities between the two in that both companies have built their systems on the Ethereum blockchain, and created their own Ethereum-based tokens(Smart contracts that implement the ERC20 token standard) called RVN and LINA respectively[28, 27]. These are used as rewards for creating reviews that meet certain quality standards. This incentivises review writing.[28, 27] The idea is that companies who sign up to the review platform will pay to top up their account with the systems token(RVN or LINA). For each review written, this account pays both a reward to the reviewer and a fee to the company.[27] It remains to be seen how successful these companies will be, given that LINAs trading volume on crypto-currency exchanges is very low[29], and Revain currently only provides reviews for crypto-currency exchanges[30]. The issue of who pays for the transactions that store reviews to the blockchain have been handled in both cases through the creation of the custom tokens.

Chapter 3: **Designing the Decentralised Application**

This chapter will provide an account of the approaches taken towards developing the review and rating management system on Ethereum, with regards to the design of the implementation. An alternative solution will also be discussed. Where deemed appropriate diagrams have been provided.

3.1 System Architecture

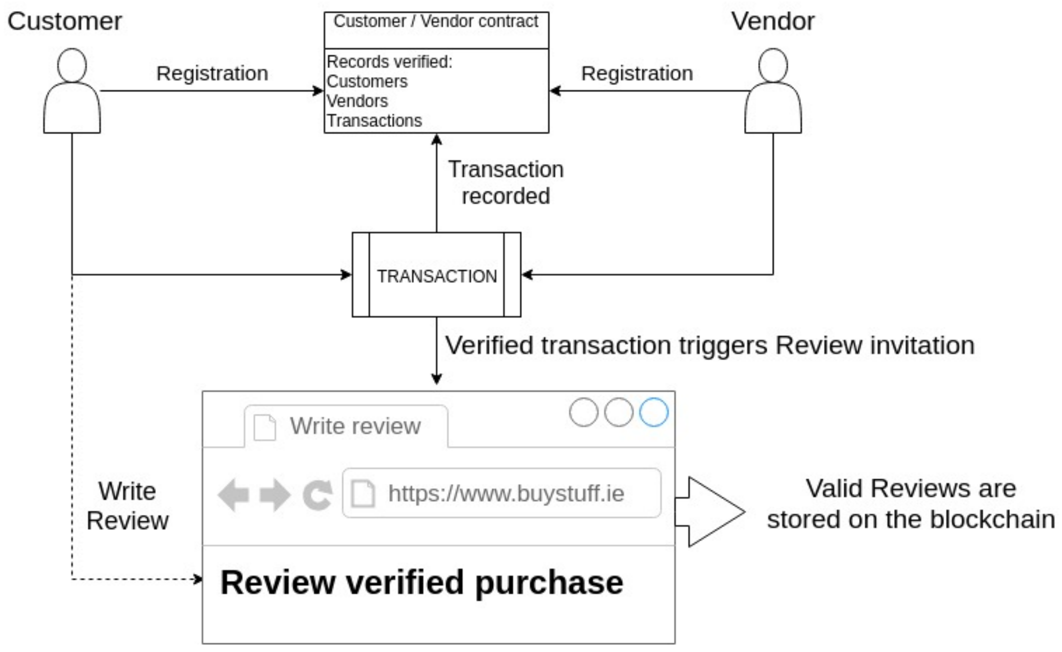


Figure 3.1: System Architecture

High-level system architecture was drawn up, as seen in figure 3.1. The basic functionality of the intended system was as follows. Customers and vendors would register themselves on the public blockchain. Since this is the Ethereum blockchain, all transactions are dealt with by smart contracts. A registration in this case is handled by a special customer/vendor contract.

When transactions take place between a vendor and a customer, they are verified, and these too are recorded on the blockchain with the customer and vendor data. Additionally, a verified transaction will trigger the sending of an invitation to the customer to review the item purchased in the transaction. The customer will receive the invite via email/SMS, which provides a URL at which they can enter their review, ranking, and possibly other media such as pictures of the item purchased. The review, once submitted, is stored publicly on the blockchain, linked to the product ID of the item, the registered vendor, and of course the registered customer. This data is now available to be retrieved and displayed in the browser to future potential customers.

3.2 Decentralised Application Design

With a high-level system architecture drawn up, it was now necessary to learn how to develop on Ethereum before any further design steps were taken. To this end, many online resources were utilised. Several smaller Ethereum projects were built with the use of tutorials, in order to learn the ropes of Ethereum dApp creation. A lot was learned along the way, and with the experience gained, the following designs were created for the decentralised application in this project.

3.2.1 Ethereum Architecture

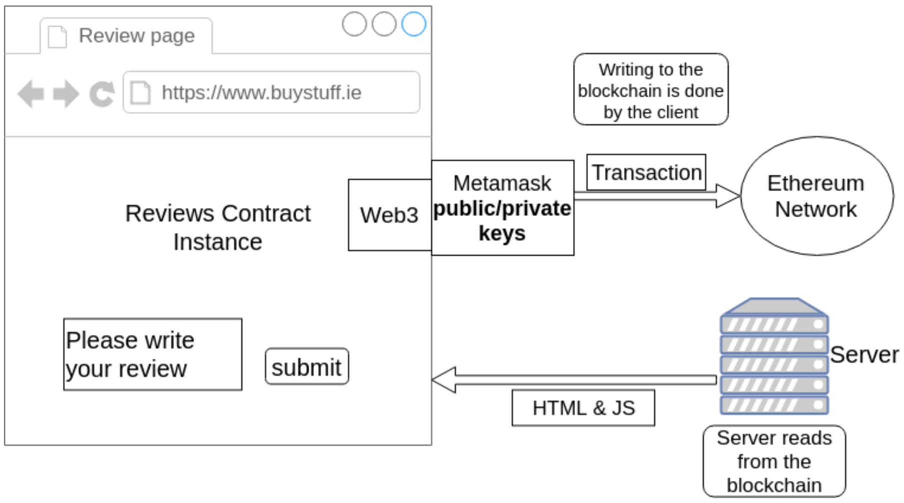
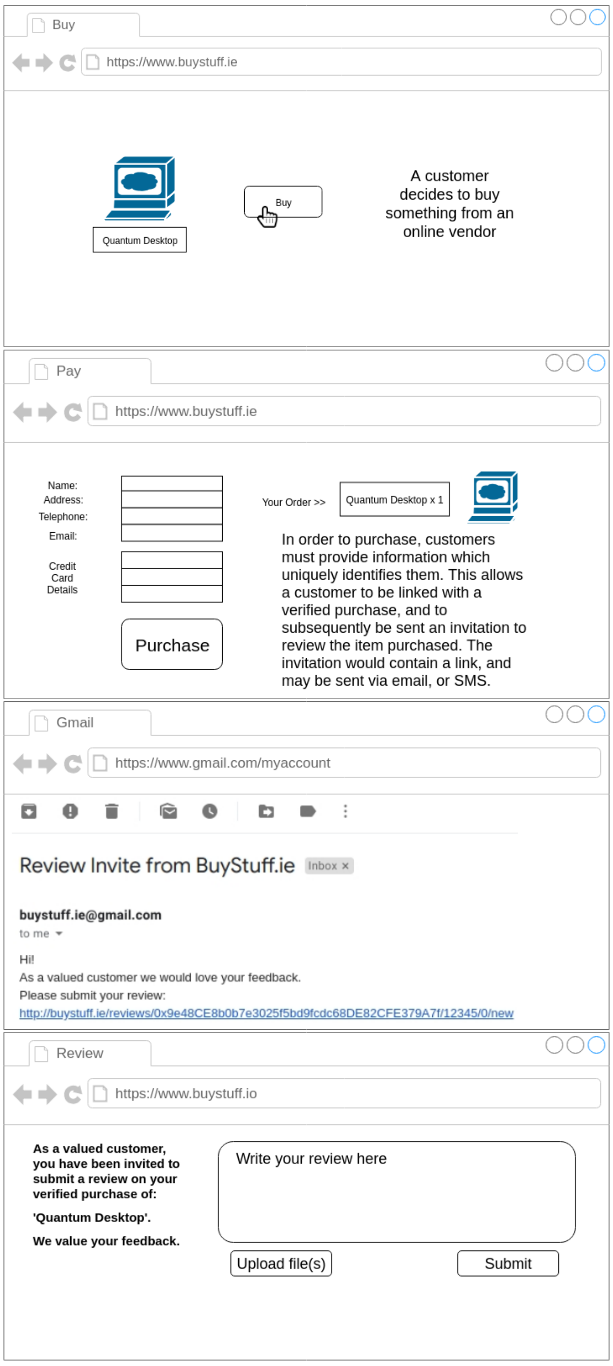

Review page
https://www.buystuff.ie
Writing to the blockchain is done by the client
Reviews Contract Instance
Web3
Metamask **public/private keys**
Transaction
Ethereum Network
Please write your review
submit
Server
HTML & JS
Server reads from the blockchain

Figure 3.2: Ethereum architecture

The biggest difference to point out between traditional web application architecture and a decentralised application on Ethereum, is that the server, though still present, has been supplanted as the most central component in the system[31]. Its responsibilities are greatly diminished. It still sends front-end view related data to the browser(HTML/Javascript assets etc.), but is no longer involved in writing the user data to a database[31].

The client is now interacting directly with a smart contract on the blockchain. When, in figure 3.2, the user presses 'submit', to change the data in the smart contract, the server is not involved in this process[31]. The browser now uses special technologies to create transactions with the public and private keys of the user. These transactions can be used to write data to the blockchain. The technologies involved will be discussed in detail in Chapter 4: Implementation. The key thing here is that writes to the blockchain are only performed by the client using their public/private keys. These keys only reside on the client, and will never(except perhaps in special circumstances) be sent to the server[31].

This shifting of responsibilities from the server to the client means there is a need for more complex applications inside the browser[31]. It is for this reason that ReactJS was chosen as the main framework for building the application front-end. ReactJS, combined with the other libraries and technologies used in this project allow for the creation of a more intelligent front-end. The details of how this was implemented are also in Chapter 4.



A customer decides to buy something from an online vendor

In order to purchase, customers must provide information which uniquely identifies them. This allows a customer to be linked with a verified purchase, and to subsequently be sent an invitation to review the item purchased. The invitation would contain a link, and may be sent via email, or SMS.

Review Invite from BuyStuff.ie

buystuff.ie@gmail.com

Hi!
As a valued customer we would love your feedback.
Please submit your review:
http://buystuff.ie/reviews/0x9e48CE8b0b7e3025f5bd9fcdc68DE82CFE379A7f/12345/0/new

As a valued customer, you have been invited to submit a review on your verified purchase of:

'Quantum Desktop'.

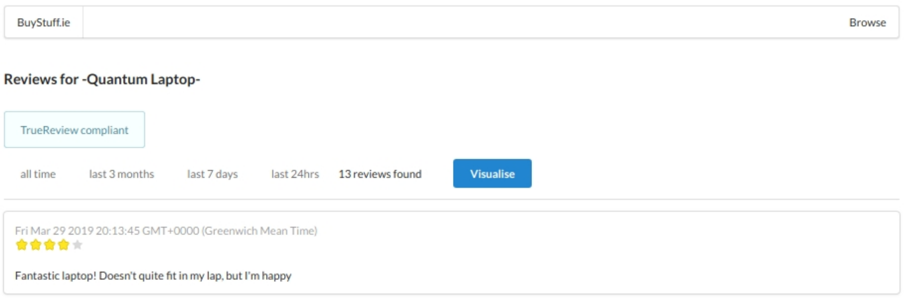
We value your feedback.

Figure 3.3: Ethereum application mock-ups

3.2.2 The Concept for the Application

Before writing any code, a suitable format was needed for the application. From the experience gained during the aforementioned 'learning' projects that were undertaken, it was clear that some form of user-interface would not only be necessary, but an essential part of the application. From very early on, the idea presented itself of creating a simple, simulated e-commerce experience in which a user could go through the motions of selecting and purchasing an item. The user would then be invited to review the item, for which their purchase was verified. Subsequently their review would be accessible from the application, which would retrieve this data from the blockchain on demand. This simulated e-commerce experience seemed an appropriate format, giving context to the system being built through a semblance of being in its intended production environment. To this end, the above mock-ups of the application were created. These are a set of simple images that take us through the intended flow of the application, at its most basic level. These mock-ups capture the essence of the application. The intention was to create a clean, simple interface in order to demonstrate a powerful technology.

3.3 Smart Contract Design



Reviews for -Quantum Laptop-

Fri Mar 29 2019 20:13:45 GMT+0000 (Greenwich Mean Time)

★★★★☆

Fantastic laptop! Doesn't quite fit in my lap, but I'm happy

Figure 3.4: TrueReview compliance label

To preface the detailing of the smart contract design, there are two assumptions that have been made. The first is the existence of a standard to which e-commerce sites can be compliant. This standard is called 'TrueReview'. TrueReview is a fictional standard, but within this projects Ethereum application, a label appears on the reviews page claiming the site to be "TrueReview compliant"(figure 3.7). This is an indication to the user that the e-commerce site manages its reviews and ratings on a blockchain-based system, with all the benefits this bestows. TrueReview compliance could be overseen by an e-commerce standards, or governmental body. It could alternatively be a simple statement by the e-commerce site, which is verifiable due to the public nature of the blockchain.

The approach taken in this project was to allow for the possibility of a standards body, to which e-commerce vendors may register themselves. This would grant them the right to be TrueReview compliant, and to display this label on their site. For simplicity, this standards body will henceforth be referred to as TrueReview. Since modelling this setup isn't the focus of this project, it has been kept quite simple, and is used, along with the simulated e-commerce experience, to give context to the systems core functionality(and indeed the core aim of this project), which is essentially to store and retrieve data to and from the blockchain.

The second assumption is that the e-commerce vendors and all users of the application have Ethereum accounts. While this would still be an unfounded assumption to make for the creator of an e-commerce site at this moment in time, it simplifies the e-commerce simulation, and allows for more focus on the core objectives of the project. In order to use most dApps built on Ethereum, users do in fact need Ethereum accounts. This is because transactions on Ethereum cost money, and transactions must be paid for by the Ethereum account from which they are sent, which generally belongs to the user who is interacting with the dApp. To ease this process, a browser extension called Metamask is employed which simplifies payments to single-click confirmations. Metamask will be discussed in more detail in the next chapter. With these two assumptions in tow, the design of the smart contracts can be discussed. Two smart contracts were created. These were named VendorFactory, and Vendor.

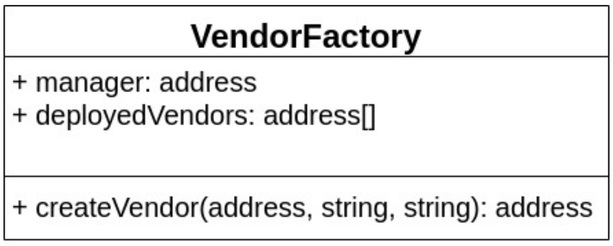

VendorFactory
+ manager: address
+ deployedVendors: address[]
+ createVendor(address, string, string): address

Figure 3.5: VendorFactory UML diagram

3.3.1 VendorFactory Contract

VendorFactory is a contract deployed once by TrueReview. VendorFactory does two things: It deploys new Vendor instances to the blockchain, and stores references to all the instances it deploys. When a vendor wants to register for TrueReview, they would provide proof of being a legally registered and licensed business through documentation and they pay a registration fee. TrueReview then uses VendorFactory to deploy a new Vendor instance to the blockchain, over which the newly registered vendor has control. The benefit of doing it through this Factory contract, is that, as mentioned above, VendorFactory keeps an immutable record of all deployed Vendors. The newly deployed Vendor instances are linked to their business which is registered with the appropriate authorities in the 'real' world. E-commerce businesses, just like any other business, must be properly registered and have valid licenses from the relevant authorities in their country of operation.

3.3.2 Vendor Contract

Instances of this contract get deployed to the blockchain by the VendorFactory contract, as detailed above. A Vendor instance is managed by the vendor themselves, having been effectively passed the keys by TrueReview, making them the manager of the contract. Being the manager of the Vendor contract grants the exclusive rights to add product listings, customers, and transaction records to the contract. As stated, the contract is deployed by TrueReview on behalf of the vendor, and therefore the vendor cannot alter the logic of the contract, since the deployed byte-code of the smart contract logic is immutable. The vendor contracts are explicitly written so that the vendor only has so much control over the data stored on the contract. For instance, reviews can only be created by customers who are verified as having purchased a particular product. Reviews cannot be altered by anyone. Product, Customers, and Transaction records between the vendor and a customer can only be added by the vendor, and none of these can later be deleted.

3.3.3 Alternative Smart Contract Design

Another possible design, was one in which VendorFactory is done away with entirely. In this design, there is only the Vendor contract. The difference is that the Vendor contracts would be deployed by the vendors themselves. This would of course mean that the vendor has total control over the logic of the smart contract, since they are deploying it. The smart contracts would be on the public blockchain, so functionality would still be openly observable. Anyone who wants to see how the ratings and reviews data are handled could do so. While this design is simpler, it does not in any way hold the vendor to account if they should suddenly decide to change their review management system logic. Neither does it hold the vendor to some particular standard in this regard. The design approach taken in the project means that all vendors are issued the same smart contract(the Vendor contract) as a part of their deployment. This goes some way to enforcing a particular standard in review management. The additional complexity was judged to be worth demonstrating the ability to ensure particular smart contract functionality.

3.4 How to Pay for the System

Certain types of transactions on the the Ethereum network cost money. While data retrieval is free, any transaction that changes the state of the blockchain has a cost. This is an inescapable fact. The system described in this report involves such transactions being initiated both by TrueReview, the vendors and the users of the system. The question of who should pay for these transactions is a very important one.

3.4.1 Deployment of Contracts

The deployment of the VendorFactory contract is clearly the sole responsibility of TrueReview. As such, cost of deployment for this contract would be paid by this organisation. The deployment of Vendor contracts on the other hand should be paid for by the vendor themselves. While it is the actual deployer(TrueReview) that must pay the fee for deployment, this is easily reimbursed through the registration fee which is paid by the vendor.

3.4.2 Storing Data to the blockchain

Transactions resulting in a change of state in the blockchain should be paid for by the vendor. Because of the Ethereum application architecture(figure 3.2), the user/client is interacting directly with the smart contracts, and is therefore footing the bill when a transaction cost is incurred. This can be acceptable, depending on the circumstances. For instance, if the user is registering themselves for a lottery, then it may be acceptable for them to pay for this transaction. However, it is undesirable for the user to pay for the storage of review and rating data unless there is an incentive to do so. This issue is discussed in more detail in the next chapter, and in Chapter 5.

Chapter 4: **Implementation**

4.1 Implementing a decentralised application on Ethereum

Ethereum is not an independent, standalone technology. It requires other technologies along side it to develop a dApp. These are necessary because the user of a dApp is going to require a graphical-user-interface(GUI), in the form of a web-based front-end, to interact with it. The front-end needs to be able to somehow communicate not only with a server which will deliver HTML and perhaps Javascript assets to the users browser, but also with the Ethereum blockchain itself. Interactions between the user and the front-end interface will need to be able to execute smart contracts on the blockchain. These require certain specific libraries and tools which, while not fundamental components of the underlying blockchain technology, are critical in providing a user experience of any utility.

4.1.1 Writing the Smart Contracts with Solidity

Solidity is an object-oriented, high-level programming language that was created specifically for writing smart contracts for Ethereum, targeting the Ethereum Virtual Machine(EVM)[32]. It has a similar syntax to Javascript, but has also been influenced by C++ and Python and is statically typed[32]. Solidity files have the .sol extension, and can be written in any regular code editor.

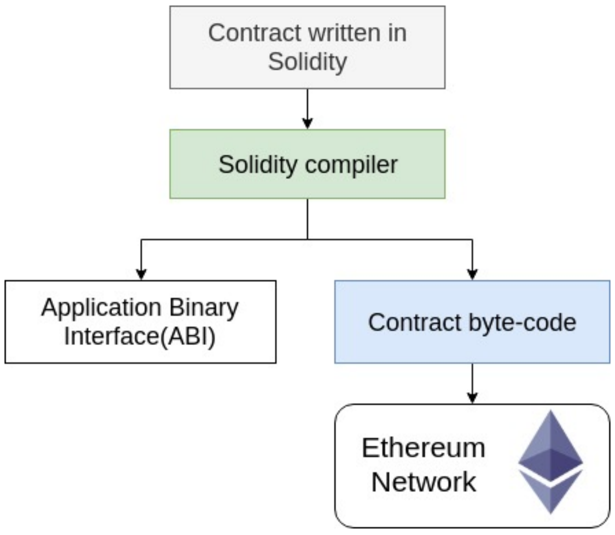

Contract written in Solidity
Solidity compiler
Application Binary Interface(ABI)
Contract byte-code
Ethereum Network

Figure 4.1: Compilation and deployment of smart contracts

The high-level human-readable contracts written in solidity are not the same thing as the code that is stored and executed on the blockchain[33]. They must be compiled with the Solidity compiler. Compilation results in various outputs. The focus here is on two items in particular: the Application Binary Interface(ABI), and the actual binary code that will be deployed in a smart contract on Ethereum[34].

The byte-code on the blockchain has no immediate meaning for the front-end Javascript which must interact with it. It is therefore necessary to tell the Javascript how to interact with the

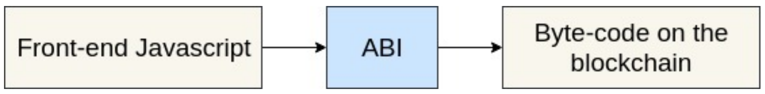


Figure 4.2: Application Binary Interace(ABI)

byte-code. The Application Binary Interface does just that. It is human-readable, and includes a list of all the methods which can be called on the smart contract[34]. While the smart contracts were mainly edited in a local IDE, they were initially drafted and frequently tested in an IDE called Remix. Two methods from the Vendor smart contract are displayed below.

```
function addTransaction(string customerID, uint productID,
    uint price, string date) public restricted {
    // Customer must be registered
    require(customers[customerID] == true);
    // Product must exist
    require(productsByID[productID].productID == productID);

    // only the Vendor can add a Transaction
    Transaction memory newTransaction = Transaction({
        customerID: customerID,
        productID: productID,
        price: price,
        date: date
    });

    transactions.push(newTransaction);

    // Record that the customer has completed a Transaction for this product
    customerTransactions[customerID][productID] = true;
}

function transactionExists(string customerID, uint productID) public view restricted
    return customerTransactions[customerID][productID];
}

function addReview(uint productID, string date, address vendorAddress,
    string customerID, string reviewText, uint rating) public {

    // Require that customer has completed a Transaction for this productID
    require(customerTransactions[customerID][productID] == true);

    Review memory newReview = Review({
        vendor: vendorAddress,
        customerID: customerID,
        productID: productID,
        date: date,
        reviewText: reviewText,
        rating: rating
    });

    productReviews[productID].push(newReview);
}
```

addTransaction and *addReview* do as their names suggest. Worth noting is that *addTransaction* requires that both the customer and the product genuinely exist in the system for the transaction to be stored. This method is also 'restricted'. This refers to a custom constraint whereby only the vendor themselves can execute this code. The *addReview* method requires that a customer is recorded as having purchased the item they are attempting to submit the review for. If requirements are met, both methods store the data to the smart-contract on the blockchain.

4.1.2 Remix IDE

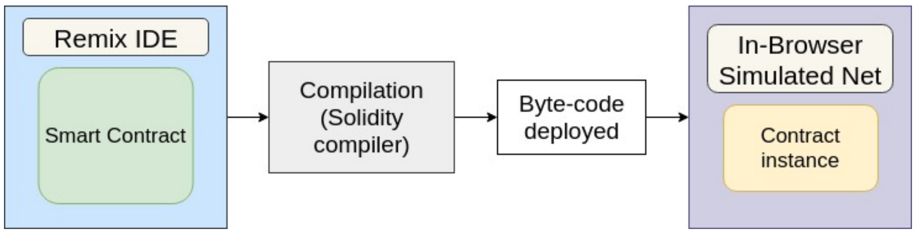
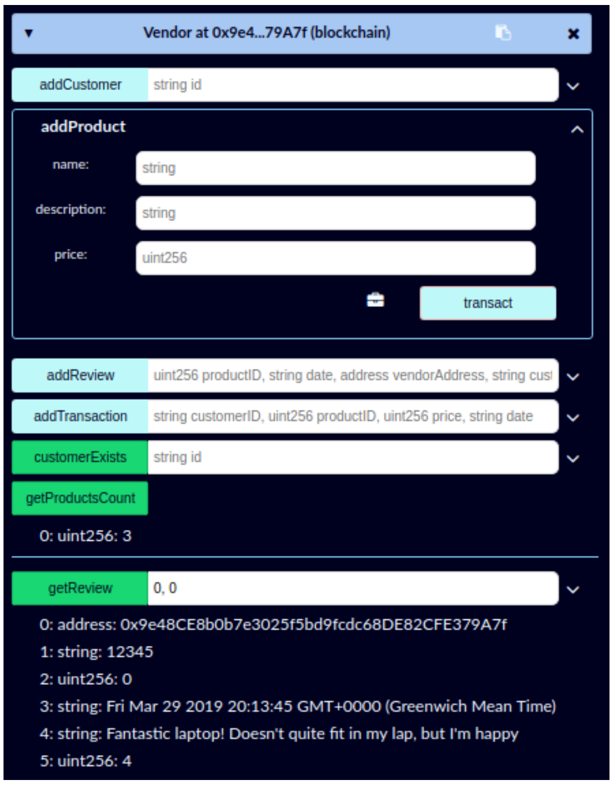


Figure 4.3: Remix IDE compilation and deployment

Remix is a browser-based IDE created specifically for writing smart contracts for Ethereum[35]. This is where both smart contracts were initially written. Remix was also employed for testing, debugging and deploying the smart contracts during development.

Remix hosts an in-browser simulated Ethereum network to which instances of smart contracts can be deployed, following automatic compilation(figure 4.4)[35]. This simulated network makes testing easier because transactions do not require the validation times that are a reality of the main and test networks. This means that transactions on this 'fake' network are immediate, and outputs quickly retrieved.



```
Vendor at 0x9e4...79A7f (blockchain)
addCustomer      string id
addProduct
  name:          string
  description:   string
  price:         uint256
                 transact
addReview        uint256 productID, string date, address vendorAddress, string cust
addTransaction   string customerID, uint256 productID, uint256 price, string date
customerExists   string id
getProductsCount
0: uint256: 3
getReview        0, 0
0: address: 0x9e48CE8b0b7e3025f5bd9fcdc68DE82CFE379A7f
1: string: 12345
2: uint256: 0
3: string: Fri Mar 29 2019 20:13:45 GMT+0000 (Greenwich Mean Time)
4: string: Fantastic laptop! Doesn't quite fit in my lap, but I'm happy
5: uint256: 4
```

Figure 4.4: Manually test contract methods with Remix

In remix, it is possible to manually execute methods on the contracts using input values, and to see the outputs. It is very manual, but also a very good way of testing that the contract methods are providing expected outputs. Automated testing would come a little later, but testing with Remix was also performed throughout development.

While the contracts were initially created in Remix, the limitations of having a browser-bound IDE prompted the move to a local IDE as soon as initial bugs were dealt with. The contracts having been completed, it was time to move on to deployment and testing on a 'real' network. The network of choice was the 'Rinkeby' Ethereum test network.

4.1.3 Metamask

In order to deploy contracts to Rinkeby, a real Ethereum account is needed. These can be created with Metamask. Metamask is a browser extension for Firefox and Chrome. It sits between Ethereum applications and the blockchain, and interacts with both[36]. Signing up creates an Ethereum account, with a unique address, and public/private keys[36]. A single account can be used across all Ethereum networks, including the Main and test networks such as Rinkeby, though an accounts balance is particular to each network. Several accounts were created during this project specifically for testing and demonstration purposes. The newly created Ethereum accounts were topped up with plenty of 'fake' Ether by visiting `http://faucet.rinkeby.io` and following the instructions there. A 12-word mnemonic was issued by Metamask that grants access to the accounts. This would be used later to connect to the Rinkeby network programmatically.

Metamask simplifies transactions with Ethereum. Whenever a transaction occurs inside an Ethereum application, Metamask will immediately pop up a little confirmation dialog, where the user can confirm or reject payment on the transaction. This is essentially a security feature, since automatic payments without the users explicit permission is definitely not desirable. Metamask gives regular users a simple way to interact with Ethereum. It makes creation of an Ethereum account very simple and it is easy to top-up the account with Ether through Metamask. This tool does a lot to break down barriers to usage of dApps and Ethereum in general.

4.1.4 Project setup

In order to create a user-interface for the application, it was necessary to set up a React project. React is a popular Javascript library for building user-interfaces. To get started, a Node.js project was set up on the local machine. Node.js supports asynchronous programming, which is perfect for both serving the front-end while interacting with Ethereum. Necessary packages were added to the project including Solc(the solidity compiler). A new directory for the ethereum side of the project was created, and into this were placed the smart contracts written in Remix. React was also added to the project. Essentially there was now a fully set up React project on the machine, which contained a folder for the Ethereum smart contracts. Work could now begin on the the graphical web front-end.

4.1.5 Web3

Web3 is a Javascript library that allows developers to interact programmatically with smart contracts deployed on the Ethereum blockchain[37]. It is the 'gateway' to Ethereum, allowing transactions to be initiated from the front-end code. These may store data, send ether(for instance, to pay someone), deploy contracts or retrieve existing data[37].

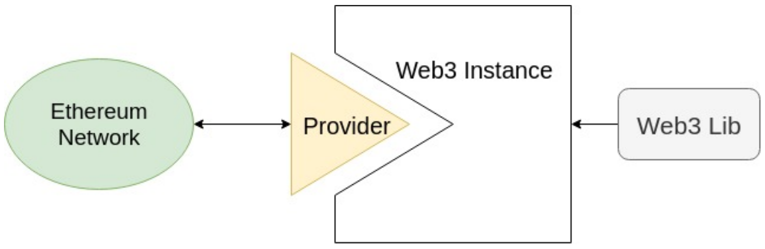


Figure 4.5: Web3 needs providers to communicate with Ethereum networks

We can imagine a Web3 instance to be a kind of phone that can connect to any Ethereum network. However, in order to connect to any specific network, Web3 needs the provider for that network[37]. In figure 4.5, the possibility of Web3 receiving providers for different networks is depicted.

With the project set up and Web3 installed, it was now possible to use providers to connect to different Ethereum networks. An immediate advantage to this is the ability to begin testing the smart contracts with Mocha and Ganache, described next.

A custom script was created to compile the smart contracts locally, and retrieve the contract interface (ABI section 4.2.1) output from the solidity compiler. This enabled interaction with deployed contracts through Web3.

4.1.6 Ganache

Ganache allows the creation of a local test network to which contracts can be deployed. This is in a similar vein to Remix, but Ganache operates entirely on the local development machine.

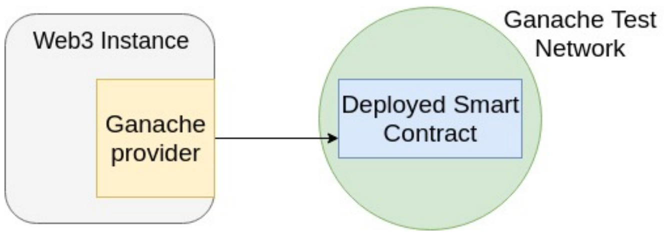


Figure 4.6: Ganache creates a locally hosted test network and set of 'fake' accounts

In order to make use of this test network, Ganache has a provider that can be used by a Web3 instance. As seen in the last section, Web3 will then allow programmatic access to the contracts deployed on this network. Ganache also provides a set of automatically generated Ethereum accounts on the test network that can be used during testing. These accounts are also accessed using Web3. A series of tests were written for the main functionalities of the smart contracts which could be run from the terminal. The tests were run with the Mocha test framework which does asynchronous testing on Node.js.

Having performed a suite of tests to ensure that the smart contracts deployed to the Ganache network were functioning correctly, it was time to deploy them to the Rinkeby network. Development on Ethereum is often done on a local network hosting an Ethereum node. But this setup comes with a lot of complexity since setting up a local node is not a trivial task. For the purposes of this project, an alternative was found.

4.1.7 Infura nodes

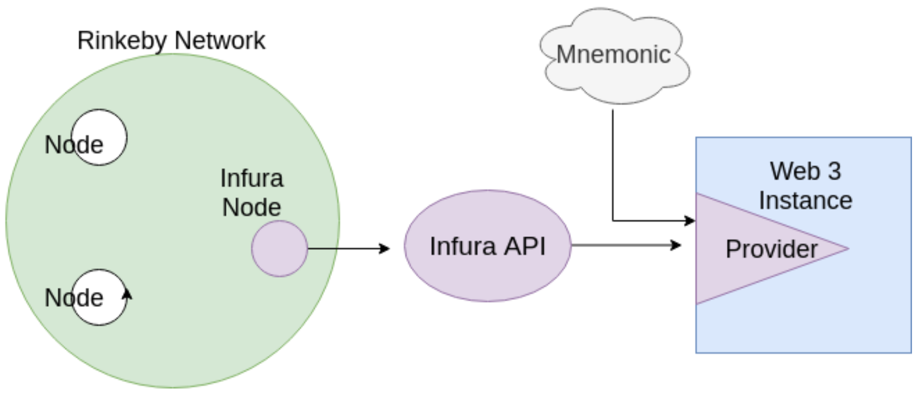


Figure 4.7: Using an Infura node to connect to Rinkeby with a custom provider

During initial development and testing, the Ganache provider was used to connect to a local test network, with access to 'fake' unlocked Ethereum accounts. To deploy to Rinkeby, a real Ethereum account is needed to pay for deployments and transactions, and a provider for Rinkeby is needed by Web3.

As stated at the end of the last section, hosting a local Ethereum node was not a good option, so another solution was found. Infura provides a public API that provides access to remote Ethereum nodes that they maintain. They host nodes on all the big Ethereum networks, including Rinkeby[39].

To get access to the Infura API, an Infura account was created[39]. A package called 'truffle-hdwallet-provider' was added to the project, which allows for the creation of a custom provider. The provider was created using this package, along with the Infura API and the 12-word mnemonic produced by Metamask which grants access to the Ethereum accounts created. This provider, when plugged into Web3, allows access to Rinkeby, using the Ethereum account created with Metamask. Contracts could now be deployed to a real Ethereum network.

4.1.8 Completion of the Front-end Interface

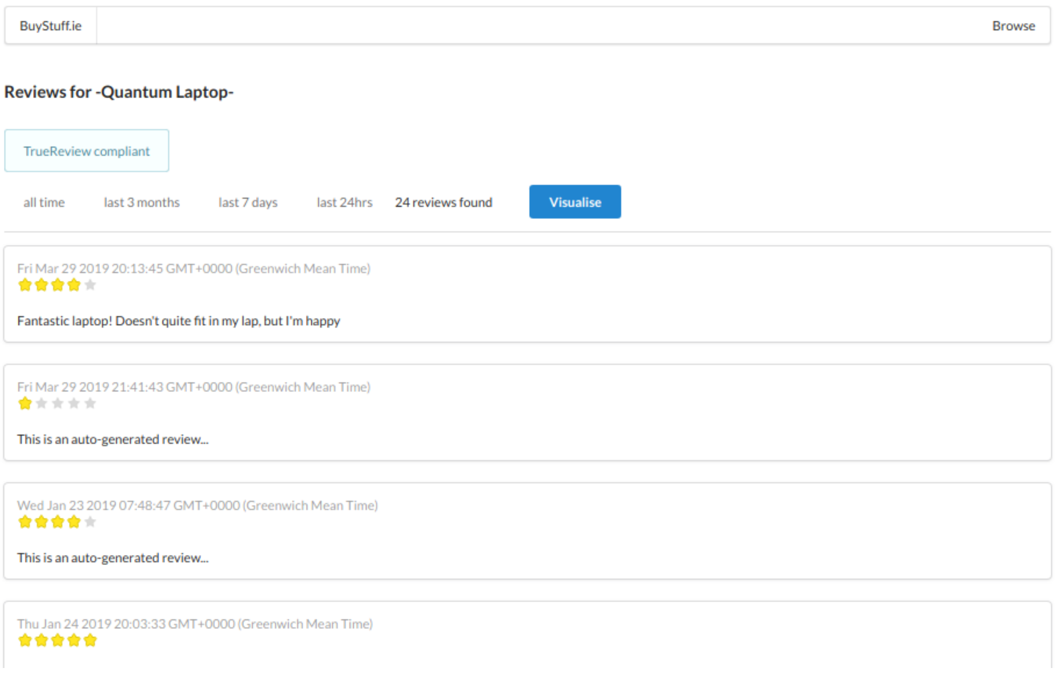
Developing the front-end web interface was a process that required a lot of learning. The smart-contracts had been created and tested, for the most part, before the front-end was started on. However, integrating the two was a very challenging experience. Because transactions on Ethereum can take between 15 and 30 seconds to complete, it can make testing a slow process. For this reason, Remix continued to be used to check that data was being stored correctly. However, the majority of the testing on the front-end was a very slow and manual process. It took a long time because it had to be rigorous. At this stage, it proved to be a great help that the smart contracts had been properly tested using Ganache and Mocha during their development, as this meant that their functionality was known to be correct, and any errors were most likely caused by the front-end code.

4.2 Testing and Evaluation

4.2.1 Testing the smart contracts

Testing the smart contracts was an ongoing task throughout development. Using Remix, the contracts were being tested right from the start as shown in Section 4.1.2. But testing on Remix is not enough. Many more issues arise with the interface between Ethereum and the front-end. To deal with this, the testing framework Mocha was used to run a test suite created to test the methods of the contracts. The test suite can have a setup at the beginning which deploys an instance of the contract under testing to the Ganache test network. The unit tests targeted individual method functionality, and various assert type statements were used on the outputs. This was very similar to using something like JUnit. The aim was the have close to 100% code coverage with the tests.

4.2.2 Testing the Front-end Interface



Reviews for -Quantum Laptop-

24 reviews found

Fri Mar 29 2019 20:13:45 GMT+0000 (Greenwich Mean Time)

Fantastic laptop! Doesn't quite fit in my lap, but I'm happy

Fri Mar 29 2019 21:41:43 GMT+0000 (Greenwich Mean Time)

This is an auto-generated review...

Wed Jan 23 2019 07:48:47 GMT+0000 (Greenwich Mean Time)

This is an auto-generated review...

Thu Jan 24 2019 20:03:33 GMT+0000 (Greenwich Mean Time)

Figure 4.8: Testing & evaluating application functionality

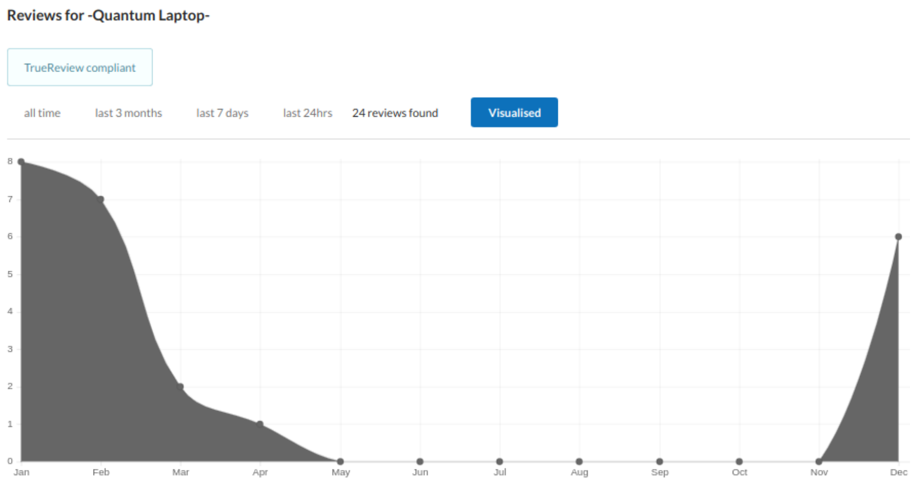


Figure 4.9: Testing & evaluating application functionality

Testing the front-end interface was more of a case of really rigorous manual testing. Going through the applications functionality carefully, over and over, to make sure that everything worked as expected. Correct functionality could be evaluated by making sure that review data stored on the blockchain was being properly retrieved. Review data was entered both manually, and with a script written to auto-generate and submit large numbers of reviews. Going to the reviews page of the product in question would then confirm that the review data had indeed been stored to the blockchain, and retrieved successfully. This could be verified by checking review content in the case of manual submissions, and number of reviews in the case of generated submissions, which can be seen in figure 4.9.

Chapter 5: **Conclusion & Future Work**

5.1 Project Objectives Completed

The core objective has been completed with development of the decentralised application on Ethereum. The application is in line with the objective, and has demonstrated that the review and rating data was both successfully stored and retrieved to and from the blockchain.

The advanced objective of visualising the data stored on the blockchain was effectively a way of proving that the data was indeed being retrieved successfully. The data was visualised through the graphical web front-end using React, in direct text form, and represented as a chart which displays the volume of reviews received by time period specified in the interface.

5.2 Weaknesses and possible solutions

5.2.1 Payment of Review Storage on the Blockchain

This is an issue that was given much thought throughout development. The issue is as follows. When a user interacts with a dApp and performs some action that results in the storage of data to the blockchain, this incurs a cost. The Ethereum application architecture(figure 3.2) is such that interactions between the client and a smart contract can be considered 'direct'. They do not involve an intermediary server. The client writes to the smart contract back-end via transactions and consequently pays for these using their own Ethereum account(with their public/private keys). Because of this setup, it makes it difficult to construct a user experience in which such transactions are paid for by the creator of the application.

The main problem is that this functionality is just not supported yet on Ethereum. For instance, a contract cannot use its own balance of Ether to pay for transactions sent from outside. The reason this was not factored into consideration in the choice of development platform, was due to the initial lack of any experience in Ethereum development. Consequently, this particular issue was unforeseen until development had reached a certain point.

When a user is signed into Metamask, and their Ethereum account, they have the ability to confirm or reject any transaction payments that arise. The question was, how could the costs of a transaction be offset to the app creator? After all, is it fair to ask the user to pay for storage of a review which they were invited to submit? It doesn't seem so.

After much searching, the general consensus seemed to be that this was not really possible yet. However, one solution that was found was the idea of immediately reimbursing the transaction fee just paid by the client. As soon as the transaction to store a review is completed, the cost would be extracted from the transaction data and sent back to the sender of the transaction, which is the client. This seems like a workable solution. Another idea would be to provide the user with store credit to the value of the transaction fee.

Even with these workarounds, there is still the matter of the user having to confirm a transaction payment with Metamask. Though it could be explained to the user that they will be reimbursed, it is by no means an ideal situation.

Another workaround to this issue was figured out after much thought, but was there a caveat. The idea behind this workaround is that when the user submits some data to be stored to the blockchain, instead of a direct interaction with the smart contract involved, a post request is sent to server with the data to be stored. From the server, it is possible to create a transaction on behalf of the user, which is paid for by the application creator. This avoids the need for a Metamask popup for the user. However, the problem with this approach arises when the data involved is sensitive. This approach can work when the data isn't the kind of data that could cause any harm if tampered with. For instance, transaction data that is just for the records of the application, and do not affect the user.

5.2.2 Incentive to Write a Review

The previous issue leads into this one. What incentive is there for someone to write a review? Especially given that they are, initially at least, paying the transaction fee to store it. This is the biggest weakness in the application. As seen in Section 2.6.3, one solution to this would be to create a custom Ethereum-based token which can be kept in reserve by an applications creators to pay for transactions instead of the clients. Another solution would be to reimburse the client the transaction fee, plus a little extra reward. This means the user actually profits from creating a review. However, a much more complex system would need to be designed around these solutions to ensure that client could not take advantage of the rewards system.

5.2.3 Performance of the Application

A current reality of the Ethereum network is that it is slow. Transactions can and do take anywhere between 10 and 30 seconds to validate. While this can't be helped at the moment, it would be possible for a competent web developer to basically hide this fact, by not forcing the user to stare at a 'processing' sign for the full validation period.

5.3 Future work

Following on from the above mentioned issues, a candidate topic for future work would be finding a solution to the issue of payment for transactions by the application creator. Especially a solution that doesn't compromise sensistive data. Closely related to this would be the topic of smart contracts being able to use their own Ether balance to pay for transactions sent to them. Exploration of these topics seems important due to the fact that the use of Metamask and need for ownership of an Ethereum account, while currently necessary for interactions with dApps, is also a barrier to usage. The average online user does not have any knowledge of these technologies. That may very well change in coming years, but simpler solutions will still be desirable.

Bibliography

[1] *What Are Hash Functions*, https://learncryptography.com/hash-functions/what-are-hash-functions

[2] *What Is Hashing? Under The Hood Of Blockchain*, https://blockgeeks.com/guides/what-is-hashing

[3] *What is public-key cryptography? A look at the encryption algorithm and its security benefits*, https://www.globalsign.com/en/ssl-information-center/what-is-public-key-cryptography

[4] *Blockchain Tutorial for Beginners: Learn Blockchain Technology* https://www.guru99.com/blockchain-tutorial.html

[5] Adil Moujahid, *Blockchain Python Tutorial*, https://github.com/adilmoujahid/Blockchain-python-tutorial

[6] Adil Moujahid, *A Practical Introduction to Blockchain with Python*, http://adilmoujahid.com/posts/2018/03/intro-Blockchain-bitcoin-python/

[7] Bhavani Shankar, *Dumbcoin - An educational python implementation of a bitcoin-like Blockchain*, https://github.com/julienr/ipynb_playground/blob/master/bitcoin/dumbcoin/dumbcoin.ipynb

[8] Andreas M. Antonopoulos , *Mastering Bitcoin (OReilly)*. Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6.

[9] Andreas M. Antonopoulos , *Mastering Bitcoin (OReilly)*. Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6, p. 27

[10] Andreas M. Antonopoulos , *Mastering Bitcoin (OReilly)*. Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6, pp. 5963

[11] Andreas M. Antonopoulos , *Mastering Bitcoin (OReilly)*. Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6, p. 346

[12] Andreas M. Antonopoulos , *Mastering Bitcoin (OReilly)*. Copyright 2017 Andreas M. Antonopoulos, 978-1-491-95438-6, pp. 296362

[13] *Contract ABI Specification* https://solidity.readthedocs.io/en/develop/abi-spec.html

[14] *About Hyperledger* https://www.ibm.com/blockchain/hyperledger

[15] *Chaincode tutorials* https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode.html

[16] Alessandro Acquisto, Allan Friedman, Rahul Telang, *Is There A Cost To Privacy Breaches? An Event Study, International Conference on Information Systems(ICIS)*, ICIS 2006 Proceedings, December 2006

[17] Greenspan Gideon, *Blockchains vs Centralized Databases*, 2016, https://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases

[18] Karl J. ODwyer, David Malone, *Bitcoin Mining and its Energy Footprint*, ISSC 2014 / CIICT 2014, Limerick, June 2627

[19] Harald Vranken, *Sustainability of bitcoin and blockchains*, Current Opinion in Environmental Sustainability, Volume 28, October 2017, Pages 1-9

[20] Rajesh Ramachandiran, *Using Blockchain Technology To Improve Trust In eCommerce Reviews*, University of Maryland - Robert H. Smith School of Business, Independent Study - Spring 2018

[21] S. Utz, P. Kerkhof, J. van den Bos, *Consumers rule: How consumer reviews influence perceived trustworthiness of online stores*, Election. Commer. Res. Appl. 11(1), 49-58

[22] Paul A. Pavlou, Angelika Dimoka, *The Nature and Role of Feedback Text Comments in Online Marketplaces: Implications for Trust Building, Price Premiums, and Seller Differentiation*, Information Systems Research Vol. 17, No. 4, December 2006, pp. 392414

[23] Richard Dennis, Gareth Owensen, *Rep on the Roll: A Peer to Peer Reputation System Based on a Rolling Blockchain*, International Journal of Digital Society (IJDS), Volume 7, Issue 1, March 2016

[24] Jimmy Aki, *Ethereum Founder Vitalik Buterin Receives Honorary Doctorate*, `https://bitcoinmagazine.com/articles/ethereum-founder-vitalik-buterin-receives-honorary-doctorate/`

[25] Vitalik Buterin *A Next-Generation Smart Contract and Decentralized Application Platform* `https://github.com/ethereum/wiki/wiki/White-Paper`

[26] *Ethereum Launches* `https://blog.ethereum.org/2015/07/30/ethereum-launches/`

[27] *Revain Full Whitepaper: New generation feedback platform based on the blockchain technology* `https://revain.org/pdf/wp/en-wp.pdf`

[28] *Lina.Network(LINA): Blockchain based application for Innovation*, January 10th 2019, Version 2.0, `https://drive.google.com/file/d/1zqNL12rC83DRBynAJFZxYfZg3v2n0al9/view`

[29] *Q A* `https://lina.network/qa/`

[30] `https://revain.org/`

[31] *Ethereum and Solidity: The complete developers guide*, Section 4, Lecture 88, `https://www.udemy.com/ethereum-and-solidity-the-complete-developers-guide/learn/v4`

[32] *Solidity*, `https://solidity.readthedocs.io/en/v0.5.7/`

[33] *Ethereum and Solidity: The complete developers guide*, Section 2, Lecture 35, `https://www.udemy.com/ethereum-and-solidity-the-complete-developers-guide/learn/v4`

[34] *Ethereum and Solidity: The complete developers guide*, Section 6, Lecture 135, `https://www.udemy.com/ethereum-and-solidity-the-complete-developers-guide/learn/v4`

[35] *Remix IDE* `https://remix.ethereum.org`

[36] `https://metamask.io/`

[37] `https://web3js.readthedocs.io/en/1.0/getting-started.html`

[38] https://www.overleaf.com/project/5c98bb82480dff5a1ee054fd

[39] https://infura.io/