

WiMe
Final Project Report

DT228
BSc in Computer Science

Eamonn Keogh
C16757629

Patricia O' Byrne

School of Computer Science
Technological University, Dublin

11/03/2020


Abstract

The aim of this project is to create a communication platform that users can choose to utilise instead of the existing internet that operates today. This platform will be a decentralized, peer-to-peer internet service that is resistant to interference, where the user of this platform will have complete freedom, atomicity and outright ownership of their data. Security of data will be an important aspect of this system as will the ability to exchange mobile data with other users in geographies where no internet or mobile phone communication is possible under the present platforms available. It is envisioned that users of this service can trade and purchase their internet time. This will be achieved by using blockchain and Ethereum technologies.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

A handwritten signature in black ink, reading "Eamonn Keogh", written over a horizontal line.

Eamonn Keogh
09/12/2019

11/03/2020

Acknowledgements

I would like to thank the module coordinators Mr Damian Gordon and Mr Johnathan Mc Carthy for their continued support throughout the year. I would especially like to thank Ms Patricia O Byrne, my project supervisor, who gave great support and guidance during this final year project. I wish to express my sincere thanks to everyone at TU Dublin for their continued work in supporting students from their first day until they complete their course of study. This project would not be possible without the backing and support from all the Department faculty and staff and I would like to express my deepest gratitude to all of them. Finally, I would like to thank my family who have provided me with the opportunity to take this path of study and who have gone on this journey with me, encouraged, supported and challenged me to complete this project.

Table of Figures

Figure 1 Traditional Top Down Organisation	12
Figure 2 Number of Internet Hosts (Clement, 2019)	13
Figure 3 Contract for the Web	14
Figure 4 Internet Penetration Rates by Continent (Miniwatts Marketing Group, 2020).....	15
Figure 5 Gap between rural and urban areas (Perrin, 2019)	15
Figure 6 DENT	22
Figure 7 Buying a Car with a Smart Contract (Voshmgir 2019)	26
Figure 8 https://decrypt.co/resources/what-is-infura	34
Figure 9 https://decrypt.co/resources/what-is-infura	35
Figure 10 https://blockonomi.com/interplanetary-file-system/	36
Figure 11 https://docs.web3j.io/	38
Figure 12 . Chai Assertion Styles (Chai Assertion Library, n.d.)	39
Figure 13 The Cloud Stack	48
Figure 14 Public cloud market share	51
Figure 15 Test Driven Development [http://bellintegrator.com]	54
Figure 16 Agile Methodology [https://www.getmailbird.com].....	55
Figure 17 SCRUM Process	56
Figure 18 Developer and User Archecture	57
Figure 19 Ethereum Virtual Machine	58
Figure 20 Android MVVM Architecture with Blockchain	59
Figure 21 WiMe Architecture.....	60
Figure 22 Marketplace use case	61
Figure 23 Low Fidelity Prototype.....	64
Figure 24 High Fidelity Prototype	66
Figure 25 Low Fidelity Prototype.....	67
Figure 26 High Fidelity Prototype	67
Figure 27 REST API Structure	70
Figure 28 WiMe REST API Sequence Diagram	70
Figure 29 Token and Product Purchase	71
Figure 30 Token and Product Sale.....	72
Figure 31 Smart Contract Design.....	73
Figure 32 Ganache GUI.....	75
Figure 33 Android File Permissions.....	82
Figure 34 Node Version Manager	87
Figure 35 Android Application Project Structure	89
Figure 36 React Web Application	90
Figure 37 REST Server project structure.....	90
Figure 38 Unit Test	97

Table of Contents

Chapter 1 - Introduction	12
Project Background	12
Today's Web	12
Internet Access in Rural and Developing Countries	15
The Future of Worldwide World	16
Project Description	17
Project Aims and Objectives	19
Project Scope	19
Thesis Roadmap	20
Chapter 2 - Literature Review	21
Alternative Existing Solutions to Problem	21
DENT	21
AirToken	22
Blockchain Consensus Models	23
Proof-of-Work	23
Proof-of-Stake	24
Proof-of-Authority (PoA)	25
Blockchain, A simple use case to get started	26
Blockchain and the Financial Sector	27
How are blockchain transactions validated?	28
Are blockchain transactions anonymous?	29
Block and the Financial Industry?	29
Can blockchain scale?	29
Blockchain, some extra details	29
Technologies researched	31
Languages	31
Solidity	31
Java	31
Kotlin	31
JavaScript	31
Operating Systems	31
Ubuntu	31

Debian	32
Android	32
Data Sources	32
Amazon Web Services (AWS) Elastic Beanstalk	32
AWS CodePipeline	33
Docker	33
Infura	33
MetaMask, Wallet Software	36
IPFS	36
Command Line Tools	36
Truffle IDE	36
Ganache CLI	37
AWS EB CLI	37
Geth	37
Frameworks	37
Web3j	38
Web3.js	38
Libraries	38
Mocha	38
Chai	38
Express	39
Axios	39
Retrofit	39
Room	40
ERC - Ethereum Request for Comments	40
Total Supply	41
Balance Of	41
Transfer	41
Transfer From	41
Approve	41
Allowance	42
CryptoKitties	42
Other Research	43

Udemy, Blockchain Courses	43
The Ethereum Virtual Machine	45
What are accounts?	45
What are transactions?	46
Gas	46
Digital Signatures/Wallets	46
Server-Side Technologies	47
Overview of Cloud Services	47
General Privacy Challenges of Cloud Computing	49
GDPR Specific Challenges	49
Amazon Web Services	49
Microsoft Azure	49
Google Cloud Platform	50
Conclusions	50
Existing Final Year Projects	51
Mesh	51
A Blockchain Ticket Transparency Solution	52
Conclusions	52
Chapter 3 - Experiment Design	53
Introduction	53
Software Methodology	53
Test Driven Development	53
Agile Development	54
SCRUM Development	56
Overview of System	57
How to interact with the Ethereum Virtual Machine	57
Initial Android Web3 Architecture	58
Use Cases	61
Android Front-End	62
Low Fidelity Prototyping	62
High Fidelity Prototype	64
React Front-End	67
Low Fidelity Prototyping	67

	67
High Fidelity Prototyping	67
Middle-Tier	68
SOAP vs REST	68
What is Web Service Security?	68
JavaScript Object Notation (JSON)	68
ACID	69
Atomicity	69
Consistency	69
Isolation	69
Durability	69
REST API Structure	70
WiMe REST API Sequence Diagram	70
Back-End	71
Token and Product Purchase	71
Token and Product Sale	72
Smart Contract Design	73
	73
Chapter 4 - Experiment Development	74
Introduction	74
Software Development	74
Geth	74
Ganache	74
Truffle	75
JavaScript Code to Compile Solidity Contract	75
JavaScript Code to Migrate Solidity Smart Contract to Blockchain	77
Using Truffle to Compile and Deploy to Blockchain	79
Android Application	79
Issues Encountered Creating an Ethereum Wallet	81
Allowing runtime permissions	82
Creating an EOA Wallet	84
Node Version Manager	86
Truffle, Blockchain Integrated Development Environment	86

Ganache, Local Ethereum Blockchain	86
Elastic Beanstalk, Amazon Web Services	86
MetaMask, Wallet Software	86
Installing the operating system	87
Installing the development dependencies	87
Server-side dependencies	87
Node Version Manager	87
Truffle, Blockchain Integrated Development Environment	88
Ganache, Local Ethereum Blockchain	88
Elastic Beanstalk, Amazon Web Services	88
MetaMask, Wallet Software	88
Client-side Development Environment	88
Front-End	89
Android Application Project Structure	89
React Web Application	90
Middle-Tier	90
Back-End	92
Ethereum Private Node	92
Genesis block	92
Timestamp	92
Gas Limit	93
Difficulty	93
Coinbase	93
Number	93
Gas Used	93
Parent Hash	93
Starting a private ethereum node	94
Connecting to the private ethereum node	94
Deploying smart contract to private ethereum node	95
Chapter 5 - Testing and Evaluation	96
Introduction	96
System Testing	96
Testing the REST api server	96

Solidity unit Tests	99
System Evaluation	101
1: Visibility of system status	101
2: Match between system and the real world	102
3: User control and freedom	102
4: Consistency and standards	102
5: Error prevention	102
6: Recognition rather than recall	102
7: Flexibility and efficiency of use	102
8: Aesthetic and minimalist design	102
9: Help users recognize, diagnose, and recover from errors	102
10: Help and documentation	103
Chapter 6 - Conclusions and Future Work	104
Introduction	104
Chapters Review	104
Literature Review	104
Experiment Design	104
Experiment Development	104
Testing and Evaluation	105
Conclusions	105
Future Work	105
Bibliography	107

Chapter 1 - Introduction

This section of the report presents the reader with the project background, describes the current issues with the internet today and discusses a potential solution to issues such as censorship, governance of user data, supply chain management tracking and cyber security.

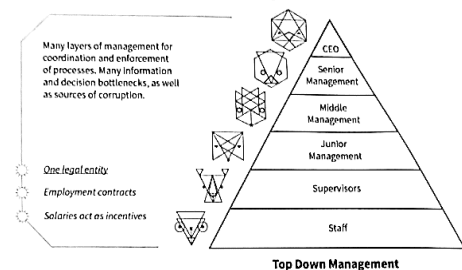
Project Background

The purpose of this project was to design and implement an Ethereum blockchain distributed application for its end users to buy and sell internet time with one another. The project attempts to disrupt the current business model that many software platforms currently use today and provide an alternative “outside-the-box” solution to the issues and weaknesses of the current web platform. The use case scenario for a blockchain alternative to many of these downfalls of the web will be delved into in this report and naturally the promotion of Ethereum blockchain will be presented. The use case for future applications to leverage blockchain to ensure immutability, security and censorship resistant programmes will be discussed. Also, given the current state of the world today and government censorship that is rampant in some parts of the globe, a blockchain solution could provide more freedom of expression to its users. The immutability and tamper resistant nature of blockchain to outside manipulation due to its consensus algorithmic model allows for this and will be described and discussed in detail.

Today's Web

The ethos of the web and the internet has altered. This is due to the migration of the internet and the web since its early 1990's existence as a decentralised digital sandbox for all users to enjoy, into a centralised monetised corporate platform where social networks and on-line business dominates. The internet and especially in this case, its ethos, has evolved since its birth in 1969 and continues to do so. Online data security as we recognise it today was not the same big issue bac (Marsan, 2009)k then. The early internet was primarily used to connect military scientists in a radically new way by the means of a decentralised network. The exponential growth of user numbers since this period

Traditional Top Down Organisation



Decentralized Autonomous Organization

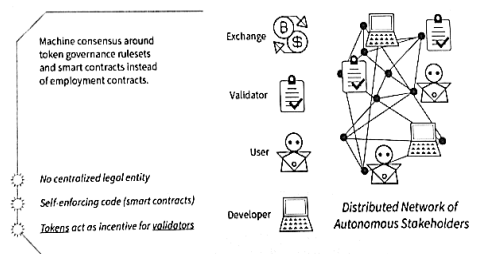


Figure 1 Traditional Top Down Organisation

has meant that everything about the platform has evolved as it grew into the all-encompassing network that it is today.

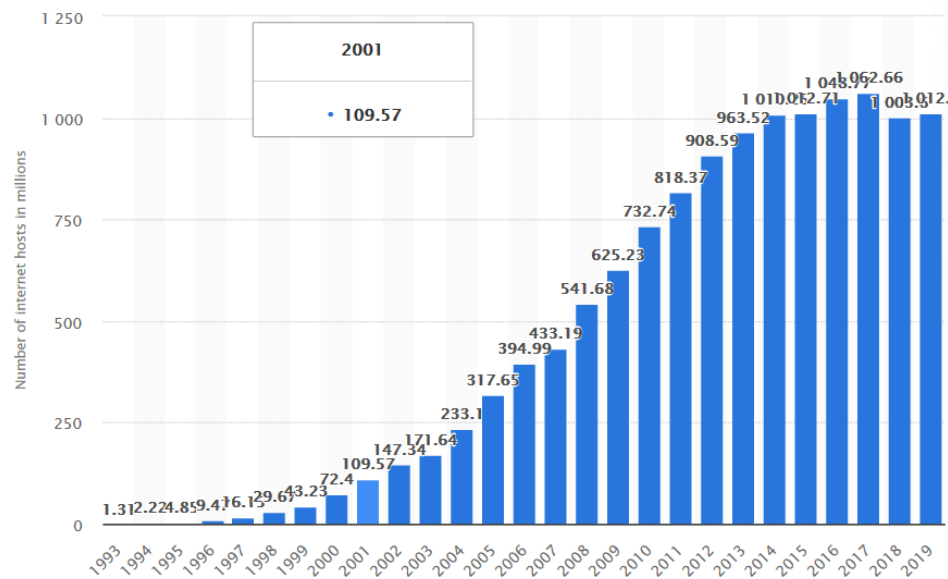


Figure 2 Number of Internet Hosts (Clement, 2019)

From 1993 onwards, after the World Wide Web was created by Tim Berners-Lee which utilised hypertext mark-up language (HTML) as the communication format and with the invention, by Marc Andreessen, of the web browser, the number of sites on the internet exploded. (Marsan, 2009) (Nukala, 2012) (Anon., n.d.)

In the mid 1990's things began to change with the introduction of online commerce. It is difficult to put accurate numbers on this type of business growth, but the platform has been a constant threat to traditional business since its introduction and continues today. By 2020, with a predicted four billion online users (Nukala, 2012; Marsan, 2009) (techcrunch, 2015) one may perceive that the possibility of the threat to data safety is growing. However, the internet's effect on political events and movements whether for good or bad is hard to call (BBVA, -).

The web's evolving ethos seems now to be reflected in the complexity and volume of the online business it facilitates and the world of politics. It is difficult to know what the ethos would develop into long term, but since it has been stated that surveillance is increasingly becoming the very business of the internet itself, we can deduce that it will be more than ever a profit ethos. This is because computers produce data and systems are being built that collect user's data in exchange for the service provided (Rashid, -).

We need to build a better web, and everyone has a role to play in safeguarding the future of the web according to the World Wide Foundation (World Wide Web Foundation, -). Technology exists today that, when bundled together, can rescue the ethos of the Tim Berners-Lee World

Wide Web from its present centralising influences. To design a system that will permit users to communicate free from data surveillance is the future. This is one of the aims of this project. An internet free from surveillance and the spread of misinformation and free from the prevalence of algorithmic bias would, I feel, be welcomed.

Security specialist Bruce Schneier has written that the internet is a surveillance state (Schneier, 2013) it tracks us everywhere (see Google mapping of journeys [4]). The global rise of the web even surprised its inventor and in a conversation with the Washington Post he talked about misinformation's rapid rise that is spread on the web and how this was weaponised in 2016 by Russia in the US election campaign. The explosion of web misinformation even caught Tim Berners-Lee completely off guard (Breslow, -) .

The perception that search engines do not exhibit bias is mistaken since they skew search results as a matter of course to an extent where some believe legislation is called for to help prevent this phenomenon (Goldman, n.d.)R (M, -). China is fast becoming the first state to implement algorithmic surveillance. Harnessing pervasive advances in artificial intelligence and data mining and storage to construct detailed profiles on all citizens (Diamond, 2018)chins

We need to build a better web, and everyone has a role to play in safeguarding the future of the Web. Thankfully, Tim Berners Lee, the creator of the world wide web, has released an ambitious plan for online governance designed to counteract the growing abuse of misinformation, data surveillance and censorship. This plan is referred to as the "Roadmap to Build a Better Web". This ambitious plan aims to hold governments, organisations and individual entities accountable and to commit to the core values outlined in Tim Berner Lee's influenced proposal, the "Contract for the Web", (World Wide Web Foundation, -)

Under the Contract for the Web, the following nine principles under the three following domains will be enforced. Guaranteeing that:

Governments will:

1. Ensure everyone can connect to the internet
2. Keep all the internet available, all the time
3. Respect and protect people's fundamental online privacy and data rights

Companies will:

4. Make the internet affordable and accessible to everyone
5. Respect and protect people's privacy and personal data to build online trust
6. Develop technologies that support the best in humanity and challenge the worst

Citizens will:

7. Be creators and collaborators on the Web
8. Build strong communities that respect civil discourse and human dignity
9. Fight for the web

Figure 3 Contract for the Web

Internet Access in Rural and Developing Countries

Another important point of interest is that of internet penetration in rural and developing countries. There is an unreliable and problematic connection quality in rural areas where mobile network coverage is next to non-existent. I can potentially see this being a large area of concern for individuals who have elderly neighbours who are isolated and contacting emergency services difficult in times of distress. This would expand the capacity of the project to be able to be used as a rescue communication tool for emergency services as a device's location and therefore a person's last known position can be correlated from other nearby devices.

Despite the rapid rate of penetration in developing countries, a large digital divide still exists between the developed and developing world. This gap is especially evident in African nations. For example, in Nigeria there are presently 0.68 computers and 1.39 internet users per hundred people in 2004.

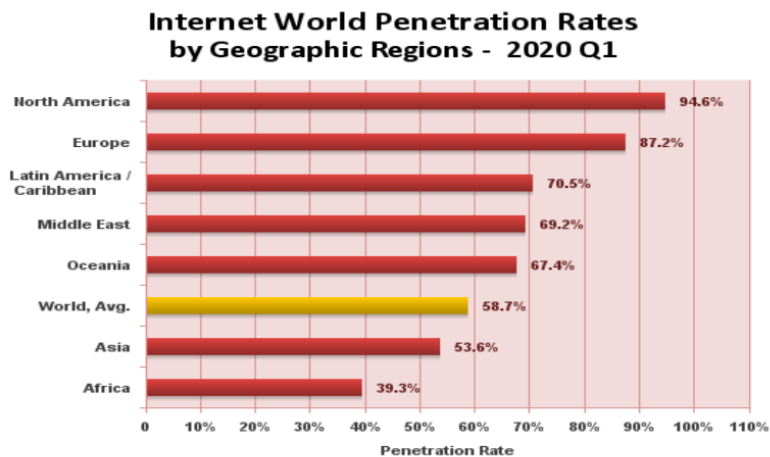
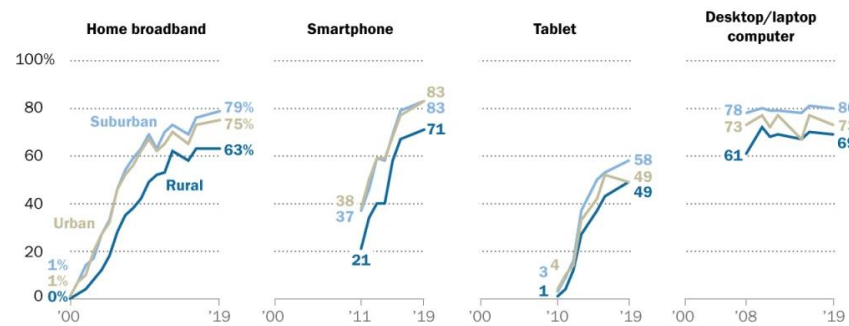


Figure 4 Internet Penetration Rates by Continent (Miniwatts Marketing Group, 2020)

These comparably low figures demonstrate a reluctance for rural residents to adopt technology. Even though rural areas are more connected today than in the past. It is also true that rural areas still lack proper high-speed internet infrastructure and lack the bandwidth speeds that their nonrural counterparts tend to have.

Rural Americans have consistently lower levels of broadband adoption

% of U.S. adults who say they have ...



Note: Respondents who did not give an answer are not shown.
 Source: Survey conducted Jan. 8-Feb. 7, 2019. Trend data from other Pew Research Center surveys.
 PEW RESEARCH CENTER

Figure 5 Gap between rural and urban areas (Perrin, 2019)

The Future of Worldwide World

The present centralised internet doesn't possess a native mechanism to transfer "State". State is what assigns who is who, ownership and permission for web resources. For this reason, state is an important missing property of the internet. Value cannot be managed without state. The efficiency of the marketplace, from earliest times, lies in the transfer of value from peer to peer and it is still true in today's efficient financial markets. Today information transfer has increased beyond recognition compared to what was possible even twenty years ago and this has facilitated the creation of products and services at lower cost and increased throughput. Internet platforms (service provider's) are required to process and broker these actions because of the lack of state in web 2.0. (Tekisalp, 2018)

Although efficient, the stateless protocols of today's internet manage information transfer where the sender and receiver have no knowledge of each other's state. The efficiency of the protocols to process information is in some part due to the simplicity of the protocols that the web is built on. Data is transmitted through TCP/IP, and other related technologies such as SMTP for emails or HTTP for Hypertext Mark-up Language pages. Transferring state would increase the server load and reduce the transfer efficiency. (Tekisalp, 2018)

Centralised data storage became the norm as it proved useful for settling payments for untrusted parties. The invention of session cookies and centralised service providers resolved the stateless issues of the web. User history is now stored using cookies on devices locally. The downside to this is that from a data governance and security perspective a user's data is controlled by the service provider who manages the users state and obtains value from this practice. Thankfully the GDPR is a positive mechanism in the control of these issues. However, the user of a service has no control over how long a cookie has been programmed to reside on their device without removing it themselves. However, it could be said that there is an exchange of value at play here. The online commercial entity holds onto all the value that the entity creates from using the services of the internet provider. In other words the service provider receives no share of the value created in this way. That said, the real issue here is the imbalance in this relationship, since it is the service provider companies who receive the grand share of the value that is created. (Tekisalp, 2018)

This was not intended by the web's inventor/s but over time a centralised structure grew up around web2 platforms. This led to everything being centralised, such as economic decision making, R&D and a concentration of power and value. However, since users had no history of paying for internet services it was inevitable that providers would find a way of capturing value. Advertising became the web2 providers vehicle of choice, a choice which led to the maximisation of commercial value through targeting advertising through the possession of the user's state. (Tekisalp, 2018)

Bitcoin and therefore blockchain brought structural changes and hastened the introduction of web3. Early adaptor developers understood the potential of the decentralised format. Bitcoin and other cryptocurrencies have the property of state and so are capable of both holding and

transferring value. The cryptocurrency protocol is designed to “remember” collectively, proceeding events and interactions. This system eliminated the problem of “double spending” through transparency and single source referencing. (Tekisalp, 2018)

Web3 is not just the preserve of present cryptocurrencies. Developers and entrepreneurs are busy building and leveraging value from other blockchain structures and technologies for the Web3 stack. Web3 has changed the internet rules since its users can now manage their own content on a public blockchain that is accessible by all interested participants in the blockchain network. (Tekisalp, 2018)

This project is quite enormous for one individual to adequately complete alone as there is an enormous amount of research involved. Researching the next iteration of the web, web3, learning numerous programming languages that aim to completely change the way one thinks about software development is quite a challenge. Issues, such as distributed applications, and the landscape for web3. The way blockchain is constantly evolving makes it difficult to find reliable documentation for web3 regarding java, and very time consuming. However, I chose this topic as my final year project because of all the above. The fact of having to become conversant with the newest technology will be my driver and I will give every effort to achieve my goal. The main risk that has become obvious at this stage is the time element and limiting the scope of the project in order to have a chance of completing it. (Tekisalp, 2018)

Project Description

With the ongoing issue of data governance, with the present web 2.0, it is my goal, in my Wireless Mesh Network and Ethereum Blockchain system (which can be referred to as WiMe from here forward) to develop a network model with the ethos of what initially the internet and web was meant to have. Basically, The WiMe system will provide a theoretical proof of concept solution for a distributed supply chain network to its participants allowing items to be requested, dispatched and to notify clients of the items state. The type of asset being supplied in the WiMe system is a digital asset known as a token. A smart contract completely defines a token. It's defined by the token name token symbol and the token exchange rate in digital currency. In WiMe this currency is Ether. The tokens are made on the Ethereum blockchain in the WiMe environment, to represent “internet time” on the blockchain network.

It is envisaged that when the purchaser initiates a transaction on the network the purchaser sends valid tokens, meaning they have sent a valid payment to the items smart contract address. After this the transaction is forwarded to the item manager's smart contract. The item manager then initiates the dispatcher, once the payment has been mined and added to the blockchain. The dispatcher then responds back to the purchaser via a dispatcher contract with the item that is now primed for delivery. The item will then be shortly in the purchaser's possession. In this way the supplier will exchange a product for the native ERC20 which is the WiMe network token that represents mobile internet time. The backend service will provide a mechanism for making these processes available by providing a service to purchase and to

trade internet time in a transparent and secure manner to ensure proper and complete ownership by the end user.

The main technology that makes this process possible and the core of WiMe, is Blockchain. Blockchain technology is a peer to peer system consisting of distributed devices that behave as nodes on a blockchain network. Each node on the network has a fragment of a copy of the transaction since, for security reasons, the data is not held completely by a single node on the chain. The data is distributed in a transparent manner which is another security feature of this technology. This results in the blockchain having a similar behaviour to a secure database. Blockchain is explained in greater detail later in the report

Since blockchain is still a new technology it was necessary to read all the literature available to me to increase my level of knowledge of the large sweep of application software that was the core of the WiMe project and imperative to its success. Increased knowledge was required in the following Technologies, Libraries and Testing suites:

Technologies		
Android	Blockchain	Bitcoin
Ethereum	DENT	Wallets
Cloud Computing	Test Driven Development	Agile
Repository	Node Version Manager	Truffle
Solidity	GETH	Ganache
Proof of Work	Proof of Stake	Proof of Authority
Server side	ERC 721	ERC 20
Metamask	Kovan	Elastic Beanstalk AWS
Rinkeby	Ropsten	Sokol
Air Token	Java	Kotlin
Java Script	Ubuntu	Debian
AWS CodePipeline	Docker	Infura
AWS EB CLI	Web3	Framework
	Gradle	
Frameworks		
	Web3j	Web3.js
Librares		
Mocha	Chai	Express
Axios	Retrofit	Room
Test Suites		
Java/Kotlin test	Solidity test	Unit test

This project is quite enormous for one individual to adequately complete alone as there is an enormous amount of research involved. Researching the next iteration of the web, web3, learning numerous programming languages that aim to completely change the way one thinks about software development is quite a challenge. Issues, such as distributed applications, and the landscape for web3. The way blockchain is constantly evolving makes it difficult to find reliable documentation for web3 regarding java, and very time consuming. However, I chose this topic as my final year project because of all the above. The fact of having to become conversant with the newest technology will be my driver and I will give every effort to achieve my goal. The main risk that has become obvious at this stage is the time element and limiting the scope of the project in order to have a chance of completing it.

Project Aims and Objectives

Following on from the project description above, the aim and objectives of this project are:

1. Design a system that allows users to communicate in a private manner.
2. Design a system that allows users transact in a secure manner.
3. Design a system that allows participants to tailor their own mobile time packages in an efficient way that suits their usage.
4. Design a system resistant to censorship
5. Design a system that's highly available and shared via a peer network
6. Design and run a private Ethereum node

This is accomplished through the development of a front-end, a middle-layer and a backend of a peer to peer system that leverages the benefits and technological advances of Ethereum blockchain that will install benefits to smart device users and allow them to be able to only purchase the amount of internet time that they require. At the innovative stage of this project the objective of the design was to encompass the developed system for other uses, and this will be the focus for future work.

The front-end of the application consists of the react web client and the node rest API.

The middle-layer of the application consists of the private Ethereum node which communicates transactional data to the backend.

The backend of the application is the Ethereum blockchain itself.

Project Scope

The scope of WiMe was an experimental one. At the outset the design included a crud application in Android for blockchain however I quickly realised that an Ethereum wallet was needed to store the transactions that occurred. However, this proved extremely problematic and due to the time constraint, the system had to be redesigned to operate instead as a react web application. The process of developing WiMe required the following inputs, learning, understanding, innovating, experimenting, assembling, reconfiguring, testing and manipulating the Ethereum technologies that were required in order to meet the aims and objectives of WiMe. As mentioned above the time element of the project was naturally a constraining factor with a project such as WiMe. Also, since a steep learning curve needed scaling from the outset, and

since the core of WiMe is blockchain and blockchain is still a new field of technology, information was not as abundant as I would have liked. The front end of the application will be a react web application as mentioned previously with a REST API acting as a middle layer and finally the backend will be the Ethereum blockchain itself which has the behaviour of a database.

Thesis Roadmap

Chapter	Purpose
2 Literature Review	Investigate existing solutions in project domain and see how blockchain technologies can be integrated with current project as well as further research into blockchain technologies
3 Experiment Design	Provide a high-level overview of the experiment. Including use cases, class and sequence diagrams and software methodologies.
4 Experiment Development	Provide the development process of the prototype and code samples as well as any issues encountered throughout the development process
5 Testing and Evaluation	Provide the tests and validations of the output of the development process that were carried out in the experiment.
6 Conclusions and Future Work	Concluding remarks on the overall project and closing statements
7 Bibliography	To document the list of references used in the report

Chapter 2 - Literature Review

In this section, the reader will be presented with background research and a critical analysis of already existing solutions in the consumer marketplace. Also, technologies that can be leveraged in the implementation of the final product and a comparison between them will form part of this section.

Alternative Existing Solutions to Problem

To gain a greater insight into what the actual result of this project could be it is sensible to analyse consumer applications in the same domain as my project and where possible to try and make improvements in addressing the needs of the user. Below the reader will find these technologies explained.

As the cryptocurrency blockchains continue to evolve at an ever-quicken pace, innovative experimentation and the implementation of new and different consensus models is ongoing. (

DENT

Is a blockchain marketplace where its users can exchange the cryptocurrency “DENT” for mobile data top ups from a chosen network provider within their country. A user can buy and sell these mobile data top up packs several ways on the DENT platform. For example, a user can send a top up directly to another user via their mobile phone number or browse the marketplace. According to the company, DENT technology is live and being used in over 44 countries, by over 148 mobile operators, servicing over 12 million users already. (Congress, 2018)

With the DENT exchange, as in all other Ethereum Blockchain technologies, the middleman is dispensed with and all contracts between a given service and its end consumer is transparent. Users can buy and sell, send and receive mobile data top ups, all by using the DENT cryptocurrency marketplace and can also donate their mobile data packs through this exchange from user to user.

As the Ethereum Blockchain is being leveraged the case for roaming fees is no longer relevant as the end user no longer needs to be routed through the network providers. Data packages can be purchased automatically, at the best price from the most suitable source in the region without the need to know exactly how much is needed. Before this technology came onstream, the consumer was stuck with unused excess capacity and instances where data can be saved are rare. Even in this case the consumers mobile data will only remain short term. (DENT, -)

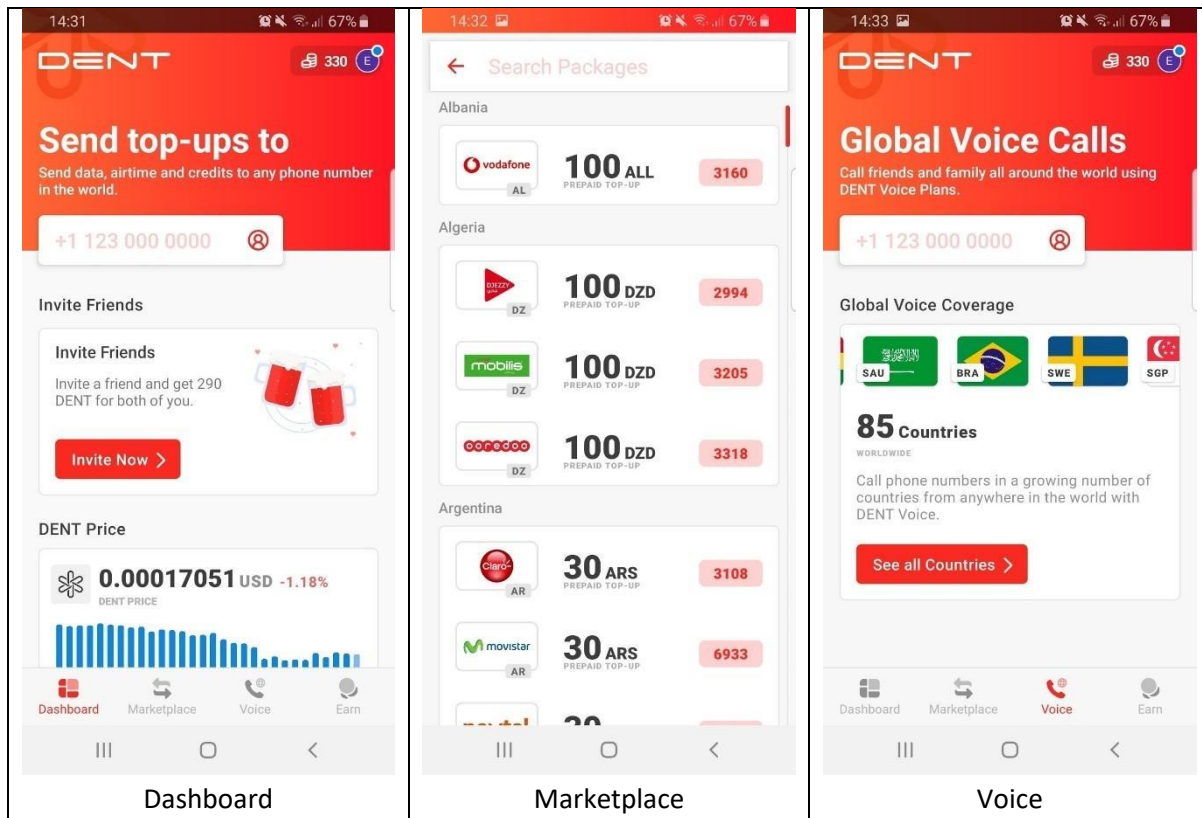


Figure 6 DENT

AirToken

Large numbers of the global population have no access to conventional banks and so have no access to basic financial services, however a large proportion of this unbanked population use mobile phones. The scarcity of proper financial services leaves individuals to fall back on the improper type provided by unregulated sources. Smart phones could well play a large part in solving this problem.

A company named "Airfox" is seeking to develop a software solution in the form of a technology platform, initially with two applications. This app will consist of a digital wallet that has the capability, through leveraging machine learning, to build an alternative, credit risk model that is smartphone based. This app is currently accessible on Android with the target on eliminating the requirement for financial institutions, thus allowing people who have no access to traditional banks to avail of this new financial model. This model will allow those with no account or credit card to make every day financial transactions easily and quick through their smartphone application. Over usage and time an individual's credit score will be built up. (AirFox, 2019) This is a peer to peer solution utilizing cryptocurrency and blockchain technology. Through blockchain technology a pool of micro financiers and investors interested in opening up this market through interest free micro loans to new entrepreneurs can be sourced. More and more financial services and regulated products can be rolled out having a major positive impact on the local economies. Therefore, through leveraging blockchain technology for this task a decentralised digital solution is possible. (AirFox, 2019)

The company is of the belief that all the structural problems and limitations can be solved through the recent advances in smartphones, applications and digital payment infrastructure etc. The core pillars of technology that are being leveraged to drive the company's solution model are; Blockchain cryptocurrency, mobile technology and Data science.

Blockchain – The company has developed a proprietary cryptocurrency and has been issued on the Ethereum blockchain the unit of which is referred to as an (Airtoken, -). This currency is aimed at being a bridge currency therefore reducing transaction costs, speeding up settlement time, strengthening security, decentralised with no intermediaries. This model will promote growth by reducing risk to funders through blockchain. (Airtoken, -) (Truffle Suite, -)

Mobile – The smart app is a mobile (wallet) that will be the gateway bank-less users to use the company's and third-party financial services. The company is also planning for a proprietary application for cheap phones the type owned by the target audience.

Data science – The company hopes to gather data through application transactions. This resource will enable, when combined with other data sources, the implementation of the company's machine learning algorithms to analyse such things as client behaviour and finance trends to produce a credit worthiness picture. (Airtoken, -)

Blockchain Consensus Models

Consensus algorithms, are absolutely necessary to a blockchain system and carry out two important tasks:

- That whatever model is employed it ensures that the next block in a blockchain has been verified and is the one and only version of the truth, and
- it keeps powerful malevolent forces from derailing the system by (Anon., n.d.)forking the chain.

The three most important consensus models (Proof of Work, Proof of Stake and Proof of Authority) are explained immediately below.

Proof-of-Work

Proof of work (PoW) is the mechanism through which the Ethereum network guarantees the validity of transactions. This mechanism is a consensus mechanism and its introduction was a historically important step for blockchain technology and is a universal source of trust. This trust is managed and secured cryptographically in the form of a publicly distributed ledger, stored on the network computers and managed collectively by all the network computers. Proof of Work's role is to guard against double spending of tokens and attempts to undermine the system. It achieves these using hashes to create a maths puzzle which requires solving by network actors to arrive at a block of verified transactions. The network actors are incentivised by earning a native Token. This reward system makes it a bad idea to try and cheat the network. Tokens have public ledger addresses. They belong to the owner of that address. Tokens can be sent by their owner on request to another address. Token owner's must prove ownership by signing

transactions with their private key. This is a mathematical process and other network computers can confirm that the transaction/s are valid without the need to know the person or their private key. The following steps are taken directly from the Shermin Voshmgir book titled "Token Economy": (Voshmgir, 2019)

1. Alice uses a wallet software that manages her private key. By performing standard mathematical operations, the wallet software can always derive her public key and her address from the private key. (Voshmgir, 2019)
2. If Alice wants to send tokens to Bob, she uses her wallet software to create a transaction that includes all the necessary details: her public key. Alice needs to specify Bob's address, and the number of tokens she wants to send.
3. Alice then creates a digital signature of this transaction, a hash performed by her wallet software).
4. In order to prove to the rest of the network that she is the owner of the address, Alice signs the hash with her private key (automatically performed by her wallet software).
5. Alice now broadcasts this transaction to any computer in the network; she sends out both the plaintext transaction, as well as the signed hash (performed by her wallet software)
6. Any computer in the network receiving the transaction can now verify the validity of the transaction.
 - a. They can use Alice's public key to mathematically verify whether the signed hash was really signed by her.
 - b. They can use the hashing operations on her readable transactions and will receive the same hash value
7. After confirming all of these details, by consensus of all network actors, validated transactions are stored into a block and hashed. This block becomes part of the blockchain if and when other computers in the network validate that the hash number on the block is valid. This process of creating new blocks of transactions is subject to the consensus mechanism "Proof of Work" (Voshmgir, 2019)
8. All network nodes will amend their ledger accordingly upon creation of the next block so that Bob now owns the funds that Alice sent him. This transaction becomes part of the universal state of the blockchain and is tamper resistant. It can be altered, but at prohibitively high costs.

The Proof of Work system makes it extremely difficult, near impossible, to cheat the blockchain system because of the computing power that would be required. As well as a high level of security, block offers participants traceability against a counterfeiting type of attack without the need for intermediaries. Again, the design of the system makes illegal manipulation of the blockchain an extremely expensive and unwise endeavour. (Voshmgir, 2019)

Proof-of-Stake

Lots of different consensus mechanisms have been tried but the Proof-of-Work and now Proof-of-Stake are the favoured ones. Proof-of-Stake is another consensus mechanism where blocks

may only be added by network actors who have a financial stake in the system. Participants must prove ownership of a certain number of tokens to generate blocks. (Voshmgir, 2019)

Delegated-Proof-of-Stake (DPoS) is a variation of Proof-of-Stake. Here there is no competition between validators to create block transactions. DPoS is a democracy where validators get voted in by the token holders. Token holders don't vote on the validation of a block, instead they vote for delegates to validate the blocks for them. The number of delegates can vary greatly that are given an order to deliver their blocks referred to as the block nonce. The nonce is essentially a block number. In this way a panel of delegates is formed, where the delegates can create blocks and make sure non-trusted parties cannot. In this system each delegate gets allotted time to publish their block. If delegates prove unworthy of trust token holders can withhold their vote. The benefit of this mechanism is the fact that transactions can be run orders of magnitude faster than most other consensus mechanisms. Variations of DPoS are run by other networks, such as – EOS, Lisk. and mechanisms like "Proof of Authority". (Voshmgir, 2019)

Proof-of-Authority (PoA)

Proof of Authority (PoA) was developed primarily to secure the chain against spam attacks on Ethereum's "Ropsten" test network. This network was really a test network and is now called "Kovan" and is available to all Ethereum users today. PoA is really a tweaked Proof of Stake model that instead of leveraging the token stake, leverages instead the form of stake. As in PoS validators are approved to validate the network with the number of validators capped at 25 or less to manage the security aspect of the network utilising this model. This means that not everyone can be successful and be approved as a validator. More apply than can be approved. Like PoS, PoA has a much lower power requirement than PoW as there are no cryptographic puzzles to be solved. In today's world lowering power consumption is getting more and more important with every-day. The PoA model is independent of the number of nodes in the system because they are all pre-approved and therefore trusted by cross verification in the public realm. (Curran, 2018)

PoA gets rid of a perceived weakness in the PoS model. This weakness may be simply stated as based on vested interest. If person A has 1,000 tokens staked in the chain and person B also has 1,000 tokens in play so at first glance, they would share the same commitment to the success of the chain. However, there may well be a gap in the level of their wealth that colours their relative commitment to the chain. Suppose Person B has €10,000 committed outside the chain, but Person A has 100 times more unpledged, Person B has more skin in the game relative to Person A since Person B has more of their wealth committed to the chain. (Curran, 2018) Other platforms utilise slightly different varieties of the PoS/PoA. However, all versions are based on financially incentivising the validators to stay honest and involved in the network and the threat of reputational loss is a distinct disincentive against acting dishonestly. (Curran, 2018)

Blockchain, A simple use case to get started

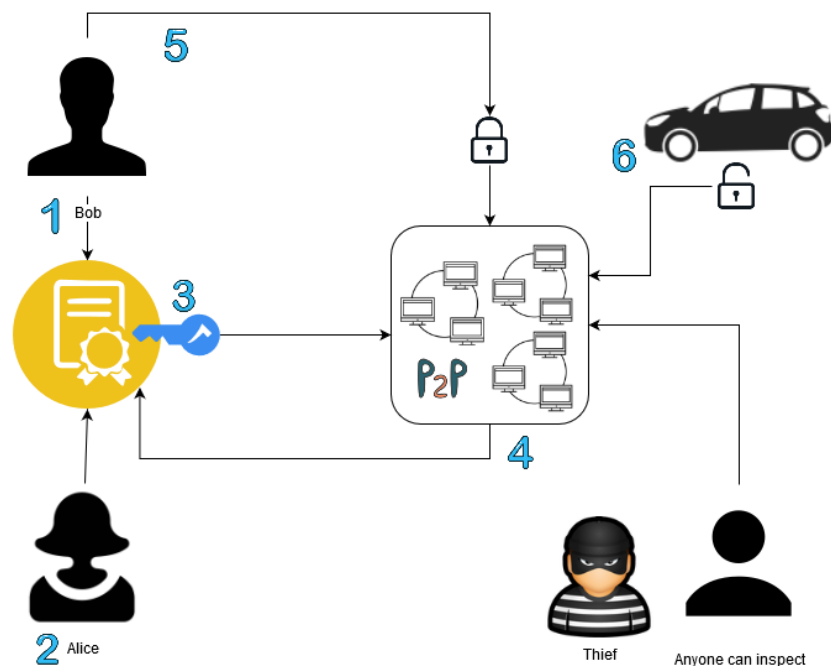


Figure 7 Buying a Car with a Smart Contract (Voshmgir 2019)

1. Bob wants to sell a car, he identified himself to the network with his Ethereum address, let's say it's 0x1111. Next he defined the terms of the sale to the smart contract and signed it with his private key.
2. Alice wants to purchase a car. She sees on a car marketplace that Bob has his car listed and up for sale. So, she signs the contract with her private key and transfers the funds to buy the car, let's say it's €30,000, from her Ethereum blockchain address of 0x2222 to Bob's Ethereum blockchain address of 0x1111.
3. Next, the smart contract verifies that Bob is indeed the owner of the car and that Alice has enough money to purchase the car.
4. If a consensus is reached by the network and both conditions equate to true, then Alice gets the unlock code to the garage where the car is parked and is waiting for her to collect and she becomes the new owner. Bob's account balance is incremented by €30,000 and Alice's account balance is decremented by €30,000.
5. The locked car has its own Ethereum address and its address is 0x3333 and it is stored on the blockchain. The smart lock of the garage communicates with the blockchain to ensure that the car and car-key are left in the garage by Bob with a smart contract to verify the smart lock credentials when Alice unlocks the car.
6. Finally, Alice can now pick up her new car and unlock the smart lock with her private key.

In case of someone stealing the car from Alice and claiming ownership of it. The blockchain can easily be audited as it can be viewed by anyone. The auditor will see that the car with an identifier of 0x3333 is owned by Alice and not by the thief. (Voshmgir, 2019)

Blockchain and the Financial Sector

The life of the stock market has been a continuous source of wealth creation for those who have engaged in and have tried to predict its short-term ups and downs. The yearly returns have returned an average 7% however this includes dividend reinvestment and adjustment for inflation. This works out as a doubling of investment for investors over ten years. (Williams, 2018)

There was only one system in play until cryptocurrency was born and so a second system was installed to compete. At the beginning of 2017 the total value of cryptocurrency was \$17.7 billion and within a year this had risen to \$836 billion, the combined capital wealth of 1,400 investable digital currencies. This figure represents an increase of 4,500% , a staggering rise at any time and way beyond what stock markets could achieve. The “traditional” stock would need decades to equal this return. However, the unpredictability of cryptocurrency movements or indeed what exactly cryptocurrency is, is not completely understood by many people. Also, the fact that cryptocurrency would not exist without Blockchain, the technology that supports it is important. (Williams, 2018) Blockchain is a decentralised digital ledger that records all transactions. The ledger records the following transactions that are paid for through digital. (Williams, 2018)

- Selling coins
- Transferring coins
- Buying a goods
- Buying services

These transactions are all performed in an encrypted form to secure and protect them. However, transactions in blockchain are performed without the need or input of a trusted third party. The shortcomings of the traditional banking systems led to the invention of cryptocurrencies and it's supporting technology, blockchain. The banking system suffered from shortcomings such as the clearing time required in most transactions especially in overseas money transfer and substantial fees also. This clearing time can be a real bone of contention for some as is the commission they charge. The developed protocols of Blockchain cut out the need for the middleman intermediary and allows the system to be decentralised and still operate securely. This means that there is no one location where information is stored and so Blockchain stores information throughout the system on servers and hard drives for three reasons already mentioned in the consensus models: (Williams, 2018)

- To ensure no one entity can gain control of the system by increasing the difficulty for malicious entities to exploit the system.
- Removing the middleman, therefore reducing fees considerably.
- Speed. Blockchain can process transactions much faster taking a few minutes at most regardless of borders.

Since banks usually close on weekends and operate during “normal banking hours”, transactions can and are carried out 24/7 using blockchain. However, no system is perfect so blockchain has certain flaws or drawbacks. (Williams, 2018)

Some industries are first adopters and some hesitant, the proportion of which is industry dependent it seems. Information technology is made up very largely of first adopters where very traditional businesses such as banking can be mostly hesitant. So large banking systems will possibly be slow to adopt a brand-new system. To do so, large institutions would have to possibly abandon their current systems and start completely from scratch. The concept of trying to integrate the two systems would be equally daunting. Also, it's not at all clear that cryptocurrencies other than bitcoin and Ethereum (which are globally the most popular digital currencies), could be scaled up to handle such a large amount of transactions. Blockchain on a large scale could consume large amounts of energy and could prove costly both financially and politically. The large energy consumption occurs in the verification of transactions in Bitcoin and Ethereum.. (Williams, 2018)

How are blockchain transactions validated?

Transactions between “untrusted” participants (no intermediary) needs a system that verifies the transactions. With cryptocurrency transactions, unlike physical currencies, there was the fear that the digital currency could be spent twice. This was referred to as “double spending” and could happen because digital information (the token) can be copied by individuals who had the capability (computing power) and the blockchain knowledge. To prevent this behaviour, bitcoin designed a mechanism based on transaction logs or blockchain. Blockchain verifies the transactional authenticity of each transaction and prevents double-spending. This system is referred to as “transaction validation”. Two of the most popular ways utilised for validating transactions are : (Williams, 2018)

1. Proof-of-work (PoW)
2. Proof-of-Stake (PoS)

Ethereum utilises the PoW model. The way this consensus model operates is that cryptocurrency miners compete against each other with high performance computers in order to solve a cryptographic puzzle that is part of the transaction verification of the blockchain network. First to solve the crypto-puzzle and validate a block of transactions, is rewarded with a block reward. For Ethereum, a block reward is a small fraction of the transaction in Ether. (Williams, 2018)

The other popular method is the PoS. model and in this model, transactions are validated differently and so, PoS is not as demanding on electricity as PoW. PoS utilises a deterministic method which means that the more of the currency one owns the more likely it will be you who will be chosen to validate blocks and add to the blockchain. Whereas the PoW uses coinage rewards to miners, the PoS rewards stakeholders with transaction fees that are paid by the users of the block being verified.

An interesting feature about blockchain is the fact that it can be applied to fit different goods or services by the developer or business and so, blockchain can be a public forum that anyone can join or entirely private. Ethereum is an open-source public chain and allows anyone to join whereas a private chain is more suitable for the corporate world.

Are blockchain transactions anonymous?

Blockchain transactions on the main or the test net are not private or anonymous even though no personal information needs to be lodged. With the blockchain, in order to trade in cryptocurrency the chain can be analysed to trace the sender or receiver of funds. (Williams, 2018)

There is a type of cryptocurrency called “privacy coins” focussed to render blockchain based transactions not traceable. To try and achieve “untraceable status” in the chain, the sender and receiver, as well as the transaction amount are shielded by beefing up the chain protocols. The existence of “Privacy coins” is problematic and although it is a tiny fraction of regular traffic they are often referred to as criminal funds by developers.

Block and the Financial Industry?

Ethereum with nearly \$116 billion market cap in 2018 had 200 organisations, testing a version of its blockchain technology some of which are banks as well as energy and technology organisations. The variety of end uses for blockchain is in the infancy stage. Personal ID's that we carry with us every day could be replaced by blockchain. It could also find use in the internet of things and companies such as Cisco and others are developing their own proprietary blockchain technology that can leverage advantages in this area. The uses where blockchain can add advantages are many and we are still in the dawn of this technology. (Williams, 2018)

Can blockchain scale?

It's very difficult to know just how viable a new technology is. Although it's about a decade old blockchain is only lately being viewed through the lenses of different industries. Even still the interested industry organisations are progressing the technology into their areas very slowly through small scale studies. The big question that has not been answered yet is how large can blockchain scale. (Williams, 2018)

However, one project is upping the scale, Ripple, American Express and Banco Santander agreed a partnership near the end of 2018 to develop a system that would allow American Express clients send cardless payments to Santander accounts over American Express FX International payment network processed by Ripple's blockchain.

Blockchain, some extra details

Blockchain is an immutable public distributed ledger that contains a list of all transactions entered into by all its participants. A record is only appended to the ledger after most participants have arrived at a consensus upon the validity of a transaction. All the verified transactions are transparent across every ledger of the chain. Blockchain was devised as a

decentralised system in order to accommodate cryptocurrency transactions and has been shown to be flawless in this regard. Not all transactions carried out are financial, any set of transactions can be handled successfully by blockchain. The cryptocurrency that the Ethereum blockchain uses is known as Ether and as of the time of writing, a single Ether has the value of €157.96. (Baliga, 2017)

There are two blockchain platforms, permission-less and permissioned. The Permission-less type is open ended and used for cryptocurrency transactions (bitcoin, ether) allowing members of the public to participate through any node. Since participants can join anonymously in an open-ended type, then by its very nature the number of nodes could be very large. For this reason, consensus mechanisms account for maliciousness especially Sybil attacks. Syble attack is where one participant generates multiple identities designed to influence and manipulate the consensus. To curb this Bitcoin has designed the consensus to be computationally difficult where participants must prove that they have solved a hard-cryptographic puzzle, referred to as Proof of Work (PoW). There are other versions of this type of test such as, NameCoin, LiteCoin and DogeCoin but Proof of Work is by far the most popular. (Monero) (Baliga, 2017)

The other platform is the Permissioned type (Hyperledger, Fabric and Multichain) is closed and only members of consortiums can input. This type of platform has semi-trusted members and only known nodes, part of the consortium is transaction verified and registered. Unlike the open ended these platforms have far fewer participants or members and utilise alternative consensus mechanisms. Permissioned platforms employ algorithms to arrive at consensus. Consensus models for closed systems are designed for a limited number of participants so do not scale properly and cannot tolerate any open-end participation. However, new models have been designed and this is a continuous process. Research over the past thirty years into consensus models has been carried out to allow secure updating of the blockchain. To finish there are a few bugbears with the framework. Blockchain is a complicated system to work with and storing data on Ethereum can be an expensive practice as it uses up a lot of space. (Baliga, 2017)

Technologies researched

In this section of the report the reader will be presented with the languages, operating systems, data sources and command line tools that were either researched or utilised in the project.

Languages

Solidity

Solidity is a Turing complete statically typed programming language to write Ethereum smart contracts. As of this moment in time it is the most popular programming language that exists to programmatically write Ethereum smart contracts for the Blockchain. Smart contracts are the equivalent of Java class in the fact that they support composition, inheritance and multiple inheritance, abstraction and polymorphism with the exception that they compile down to bytecode that can be read by the Ethereum Virtual Machine (EVM) instead. (Truffle Suite, n.d.)

Java

Developed by Sun Microsystems and released in 1995 Java is a coding language and computer platform. There are very many applications and websites that require Java installation before they will operate, and this is on-going with new devices coming on stream. Java is a reliable secure language and is used everywhere on many devices, data centres, scientific supercomputers to game consoles and the internet.

Kotlin

Kotlin was designed for the JVM language and Android. Amenable to operating with Java bytecode. Most developers say it was clearly inspired by and resembles Scala, but has been endowed with a different design philosophy, flatter learning curve and some features that are worth having such as null-safe types.

JavaScript

JavaScript was chosen as the primary programming language for many of the WiMe applications. JavaScript was utilised for the Representational State Transfer (REST) Server and writing the Solidity smart contract tests. The benefits of using JavaScript for both areas are that it simplifies the development cycle somewhat as the latter part of the tests can be written using the mocha and chai frameworks.

Operating Systems

Ubuntu

Mark Shuttleworth philanthropist, businessman and founder of Ubuntu, a software operating system free for everyone to use and has communities of users who support each other. Ubuntu

is also an open source Linux operating system that comes in various flavours for desktop, server and internet of things (IoT) devices. Ubuntu was required to create the private Ethereum node of WiMe. One good feature of Ubuntu is the fact that one can run Ubuntu side by side with the existing operating system on the computer or replace it and just use Ubuntu alone. As with other operating systems Ubuntu runs on Linux architecture for communication with the computer hardware. Ubuntu and Linux are really two halves of the whole, with Linux functioning as Ubuntu's kernel. (How stuff works tech, n.d.)

[https://computer \(How stuff works tech, n.d.\), and \(How Stuff Works tech, - \).howstuffworks.com/ubuntu2.htm](https://computer.howstuffworks.com/ubuntu2.htm)

Debian

Debian is usually mentioned in the same breath as Linux. Debian is the operating system, or a set of programmes and utilities and Linux is Debian's translator or Kernel. Debian is probably the most successful and popular UNIX-Like operating system on Linux distribution, judging by the amount of developers that use Debian and its derivatives across all devices. It is the basis of Ubuntu and other operating systems and can be used at home, on a server, or in office.

However, work is progressing towards providing Debian for more kernels, such as the Hurd which is a collection of servers running on microkernels to establish other features. One reason Debian is so popular is the number of packages available 59000 of them (bundled precompiled software for easy installation) and package manager to help with the installation into many computers as easily as a single device. (Quora, 2018) (Debian, n.d.)

Android

Android was used in the initial attempt of the final project. However, it was later abandoned due to time pressures and web3j framework's lack of documentation. Android proved to be problematic in this case and WiMe's time restriction meant the android input had to be abandoned and the project redesigned to operate without an Android device. However, Android is the software for smart devices and has been the standard since 2003. An Open source operating system, referred to as Android Open Source Project or AOSP which was led by Google. Android's function, being a smart device software, is to be the interface between the user and their gadget. Android is a specially tailored operating system for touch screen devices. Developed in conjunction with Google, Android is kept refreshed and up to date on a yearly basis and most of the leading smart device manufactures run the operating system on their products. Manufactures such as Sony, Samsung, LG and others. (Schmidt, 2016)

Data Sources

Amazon Web Services (AWS) Elastic Beanstalk

The function of Elastic Beanstalk is to deploy and manage applications in the Amazon Web Services Cloud and reduce the complicated management function while not restricting the user's choice or control. All that is required of the user is to deploy the application and the details of capacity provisioning, health monitoring, scaling and load balancing are all handled automatically. Elastic Beanstalk builds and supports platform versions and provisions of AWS

resources, to run your application. Interacting with Elastic Beanstalk is possible if the Elastic Beanstalk console is utilised. It is also possible through the Amazon Web Services Command Line Interface (aws, n.d.)

AWS CodePipeline

AWS Pipeline is a continuous integration and continuous deployment managed pipeline delivery service. This helps organisations manage and automate pipeline build and smooths out infrastructure updates. Each time your organisation has a code change CodePipeline tests, builds and deploys release process phases. This is helpful in delivering updated features in a timely manner. AWS CodePipeline can be integrated with other services like GitHub Jenkins, and/or BitBucket. AWS Pipeline seems to operate the same as my WiMe project in that you only pay for what is used with no fees or long-term commitments. (Williams, 2018)

Docker

In 2013, Docker was introduced to the industry and has become incredibly popular amongst developers and has become the standard when it comes to containerization. The docker platform on Amazon Web Services allows the building and testing of applications from a desktop environment to the cloud and allows fast deployment. This was of great benefit for the early stages and mid stages of the project as I could quickly try things out, check the compatibility of different software packages on different operating systems, such as alpine and ubuntu. Also, containers are self-contained in that they have everything the software needs to run such as code, system tools etc.

This was ideal as it added a sense of stability and security, as I wasn't too worried about installing a package with my device no longer working. Docker is ideal for these types of situations as the container is self-isolated from the main operating system. It is also best suited in scaling applications to suit any environment and AWS supports Docker. This was experimented with early in the development process and a few docker containers were created as I ran into some incompatibility issues with solidity version 0.4.24 source code.

Docker was a very useful tool and because of the reasons mentioned above it means that developers need only to concentrate on the code not on the system it will eventually run on. Docker like a lot of operating systems will run on Linux Kernel and users will only need other applications if what they add is not already running on the device and since it is open source anyone can extend it by contributing to meet their own personal needs. An example of this would be Docker Hub, which is a repository of official and community driven docker images free to download. (Amazon WebServices, -)

Infura

Infura works with the Ethereum blockchain and is a blockchain node provider. Blockchain technology is complicated. Infura makes working with Blockchain easier and quicker and is

utilised by: MetaMask, CryptoKitties, uPort and Truffle. Infura's stated aim is to make life easier for developers who can immediately begin to work on the Ethereum framework and not have to run complex infrastructure themselves. However, there are fears that Infura may increase centralisation which is completely against the ideals for a blockchain framework since Infura is a centralised set of tools and so vulnerable to being successfully attacked. This fear can be functionally limiting as is the fear of being utilised as a censoring tool by third parties including governments.

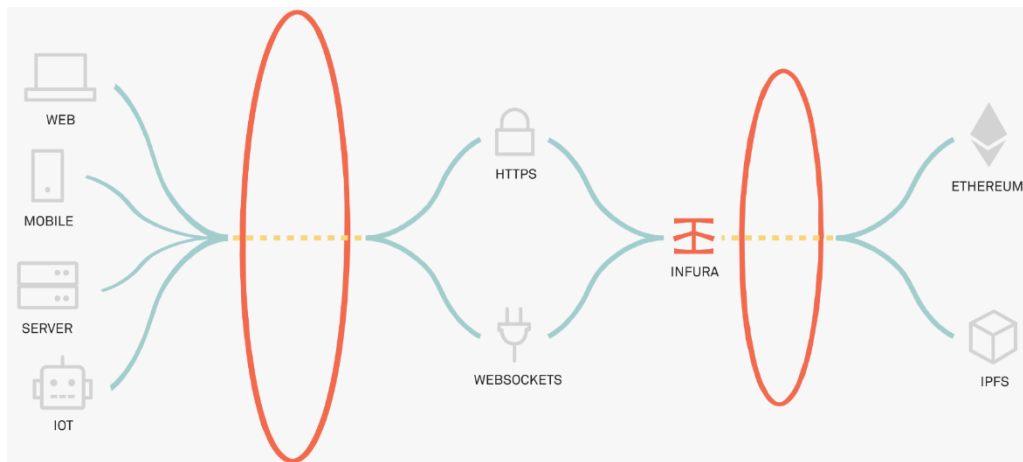


Figure 8 <https://decrypt.co/resources/what-is-infura>

However, Infura has almost 100% uptime across all the testnets and mainnet from the last three months, which is not an easy thing to accomplish and has an average response time of 370ms which can be seen below:

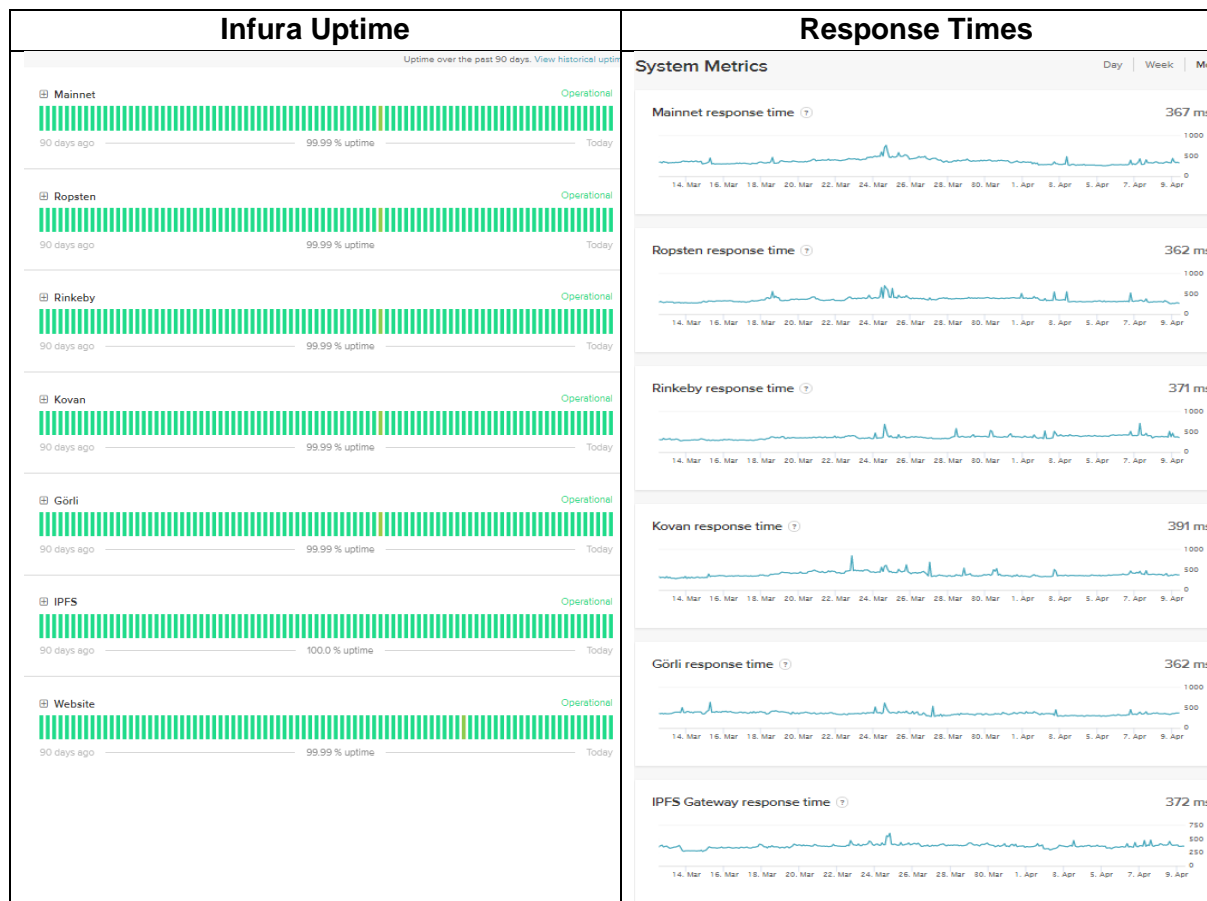


Figure 9 <https://decrypt.co/resources/what-is-infura>

As you can see from the Infura uptime graphic, all networks have 99.99% uptime. The response times of the various networks doesn't vary too much. However, it was not the only deciding factor in selecting a testnet. Clearly, Ropsten or Görli are the clear victors in terms of response time. However, Ropsten has been victim to spam attacks in the past as malicious users exploited its faucet, a faucet is a way to get ether from the testnet for development purposes. It's important to state that ether on a testnet do not have any value, they are essentially placeholders for the "real" ether on the main network. So, the clear winner here is Görli? Well not quite. Görli is still under development and therefore is unstable.

Initially I experimented with the Infura service especially for my prototype design. However, after learning about the centralization issue of Infura and the technical issues with AWS, I made a decision to remove it from the project as a Ethereum node provider. I instead opted for my own private Ethereum node that I had complete control and access over. By creating my own Ethereum private node I learned first-hand how to monitor, mine and create accounts. (Copeland, 2019)

MetaMask, Wallet Software

Often referred to as a cryptocurrency wallet but in actual fact it is a browser extension that allows the user to run dApps, without being an Ethereum node on the cryptocurrency network. Metamask allows connection to an Infura node in order to interact with smart contracts. An Ethereum wallet is managed by MetaMask where one's cryptocurrency Ether resides and allows online transactions i.e. send and/or receive Ether through a dApp of choice. When connected to d'Apps, participants can trade their different cryptocurrencies on the decentralised exchanges, gamble in other ways and/or play games (Choi, 2015) (Hussey, 2019)

IPFS

IPFS, Interplanetary file is a distributed storage system and an open-source, peer-to-peer distributed system for all devices which stores files, websites and data. As stated earlier in this report the present internet is fast developing problems with its centralised framework and outdated protocols. IPFS tries to address issues around the problems connected to the Http protocol which underpins communication, how messages are transmitted, across the internet. Http also controls how browsers respond to the commands requested by servers and is the protocol spine of client-server-paradigm (Curran, 2020)

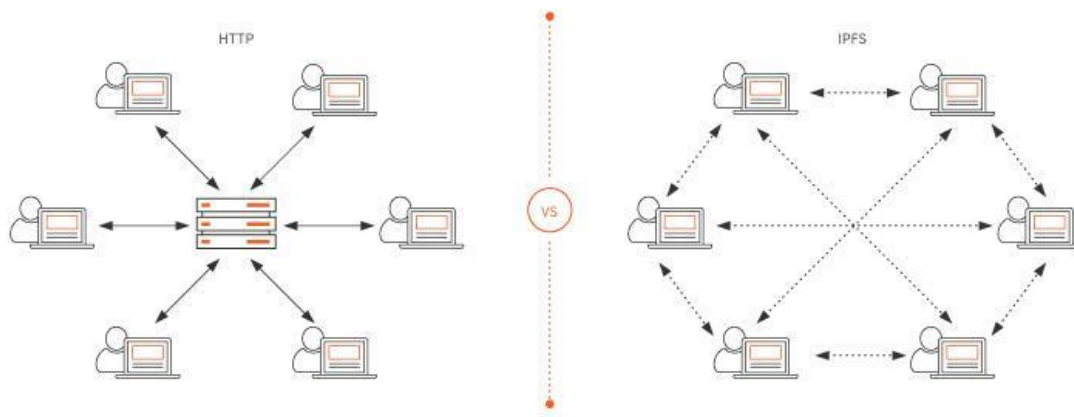


Figure 10 <https://blockonomi.com/interplanetary-file-system/>

Command Line Tools

Truffle IDE

Truffle is a collection of software tools or testing framework , pipeline and generally a development environment for blockchain technology with the Ethereum Virtual Machine with the specific focus of easing a developer's burden.

Truffle is a collection of software packages and tooling that allows a developer to write code for a smart contract that allows them to test, configure, refactor and migrate a smart contract to an Ethereum Virtual Machine (EVM). This EVM can either be a local instance on the ganache network, the truffle instance of the in memory blockchain, the public mainnet of blockchain or a

public testnet of blockchain such as rinkeby. Truffle is also a useful tool in the hunt for viruses utilising virus specific probes. (Truffle Suite, -) (Sourceforge, 2017)

Ganache CLI

An in-memory blockchain for Ethereum development that is utilised to deploy smart contracts, develop applications and run tests and is a desktop application and a command line tool. Ganache is available for Windows, Mac, and Linux. Ganache CLI is useful for simulating full client interaction and speeding the development of Ethereum applications and making the process easier and safer. (aws, -)

AWS EB CLI

Elastic Beanstalk command line (EB CLI) interface is utilised for a wide variety of operations. It's used to deploy and manage applications on Elastic Beanstalk and environments. EB CLI can integrate with Git for deploying application source code under Git source control. (aws, -)

Geth

Geth or Go Ethereum is a command line interface that allows you to run and operate a full Ethereum node in Go programming language. It's a program that operates as a node for the Ethereum blockchain, through which Ether can be mined to create software which runs on the EVM, the Ethereum Virtual Machine. Blocks are mined because Geth is implemented in Go and the blocks that are mined generate Ether that deploy and interact with the smart contracts to transfer funds, to inspect block history and to create accounts, etc. Geth can be used to connect to the public Ethereum network, also called the main net, or to create your own private network for development purposes. Employing Geth in the development process lets you fully test the application before deploying it to the main net. (M, 2018)

Running a private Ethereum node and running tests takes much longer in time as it is running locally on your machine in comparison to truffles development of blockchain which is running in memory. Each transaction has to be mined on your machine before it can be tested and one has to write several test cases. (M, 2018)

Frameworks

Frameworks or software frameworks are platforms where code with general functionality can be specialised. Frameworks are software libraries where software can be developed with a reusable application program interface (API).

Web3j

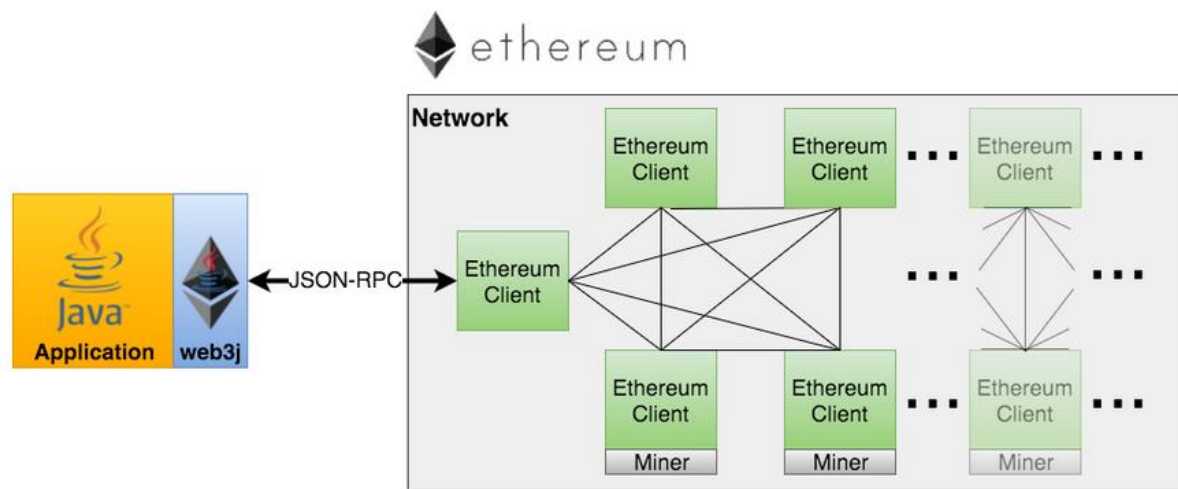


Figure 11 <https://docs.web3j.io/>

web3j is a highly modular, reactive, type safe Java and Android library for working with Smart Contracts and integrating with clients (nodes) on the Ethereum network: (Web3j, -)

Web3.js

Just like its cousin web3j the web3.js framework is a collection of libraries which allow a developer to interact with either a local Ethereum node such as ganache or a remote Ethereum node such as Geth using the HTTP suite of protocols or via Inter Process Communication via web sockets. This framework is written only for JavaScript compatible languages and it supports JavaScript ES2017 features such as async await. The current latest stable version as of this project is v1.2.6. There are development plans as the team behind this framework have released v2.0.0-alpha.1 of the web3.js framework. (Web3j, -)

Libraries

Mocha

Mocha is excellent for running tests. It combines with Node to run automatic tests. Mocha is full featured making Mocha the most dependable module on the node package manager. Mocha was utilized for writing tests for solidity.

Chai

Chai is an interface tool to make testing easier by making assertions more readable and closer to complete english sentences. Chai can be combined with any other JavaScript testing framework. Chai was used alongside Mocha when writing tests for solidity. For example the three interfaces that Chai provide to a developer are as follows:

Style	Code
Should	<pre>chai.should(); foo.should.be.a('string'); foo.should.equal('bar'); foo.should.have.lengthOf(3); tea.should.have.property('flavors').with.lengthOf(3);</pre>
Expect	<pre>var expect = chai.expect; expect(foo).to.be.a('string'); expect(foo).to.equal('bar'); expect(foo).to.have.lengthOf(3); expect(tea).to.have.property('flavors').with.lengthOf(3);</pre>
Assert	<pre>var assert = chai.assert; assert.typeOf(foo, 'string'); assert.equal(foo, 'bar'); assert.lengthOf(foo, 3); assert.property(tea, 'flavors'); assert.lengthOf(tea.flavors, 3);</pre>

Figure 12 . Chai Assertion Styles (Chai Assertion Library, n.d.)

Express

Express is a node package that encapsulates and abstracts HTTP requests routing, storing static files. In the project the REST API server utilised this package.

Axios

Axios is a JavaScript quality of life improvement library for the fetch api. With the difference being that with axios it immediately returns the data requested and there is no need like when using the fetch api to parse the response object to JSON. Axios supports promises, URL parameters and for the project it was used within the react client to consume the REST API endpoints.

Retrofit

Retrofit is an Android network library for all network transactions. It is easy to retrieve and to upload data through a REST based web service. Retrofit became the replacement for all other libraries including ending up being "Volleys" replacement because of retrofit's usability, extensibility and better performance. In the Android network the Retrofit library stands out from the others also through its usability where the need to parse JSON responses as this is performed by Retrofit itself utilising GSON. Retrofit outperforms AsyncTask and Volley across the three-network request and response metrics of one, seven and twenty discussions. This

library was experimented with in the project when implementing the android client (abhiandroid, -)

<https://abhiandroid.com/programming/retrofit>

Room

Room abstracts a lot of the complex boilerplate code needed to be written to interact with the SQLite database on android. Essentially the room persistence library allows one to quickly and with relative ease create CRUD operations for a SQLite database to cache an app's data even when offline.

<http://www.planetb.ca/syntax-highlight-word>

ERC - Ethereum Request for Comments

As with all standards in their pre-acceptance stage the “design” is dispersed among the targeted developer community for peer input and comments (hence the acronym ERC). After this well-informed input, if the “design” is accepted by the peer community, the “design” becomes the “Standard”. In this way a set of standards is built up to cover all aspects relating to, let's say “Tokens” of a blockchain mechanism. It is worth noting that - On Ethereum, all tokens are contracts but not all contracts are tokens. The most popular Ethereum standards are ERC-20 and ERC-721.

ERC-20. This proposed standard is a set of functions that will make up the suite of important standards to which smart contracts are digitally produced and is the most used specification in Ethereum. ERC-20 is a fungible standard for tokens so that ERC-20 tokens are interchangeable and are suitable and perform similarly on dApps.

ERC-20 solves the problems that would emerge if the Ethereum users began using their own unique non fungible tokens and functions and preventing smart contracts being broken and destroyed and hacking attacks through token transfers.

A utility token or common function is an application based on ERC-20. ERC-20 is made up of six functions, two events and three token information functions and if written into the smart contract then it is an ERC-20 token.

The following six functions define what the ERC20 token standard is:

1. Total supply
2. Balance of
3. Transfer
4. Transfer from
5. Approve
6. Allowance

Total Supply

This method is used to retrieve the current tokens that are supplied by the exchange. The method interface is given as such:

```
function totalSupply() public view returns (uint256)
```

(Fabian Vogelsteller, 2015)

Balance Of

This method finds the balance of an Ethereum account by providing the method with the address of a given Ethereum account and returns the balance as a unsigned integer value. The method interface is as such:

```
function balanceOf(address _owner) public view returns (uint256 balance)
```

(Fabian Vogelsteller, 2015)

Transfer

This method is used to transfer a given value of tokens to another address, ERC20 standard dictates that this method must execute a transfer event and if an error occurs it must throw an exception.

```
function transfer(address _to, uint256 _value) public returns (bool success)
```

(Fabian Vogelsteller, 2015)

Transfer From

This method is very similar to the method above however it transfers a given value amount between contracts to transfer a value of tokens on the method caller's behalf. An exception is thrown if an error occurs. The method interface is as follows:

```
function transferFrom(address _from, address _to, uint256 _value) public returns  
    (bool success)
```

(Fabian Vogelsteller, 2015)

Approve

The approve method allows a “_spender” to withdraw from your account, several times up until a specified “_value”. The method interface is as follows:

```
function approve(address _spender, uint256 _value) public returns (bool success)
```

(Fabian Vogelsteller, 2015)

Allowance

This method returns the amount that the “_spender” is allowed to withdraw from the “_owner”. The method interface is as follows:

```
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
```

(Fabian Vogelsteller, 2015)

ERC-721. Ethereum Improvement Proposal (2017) is a proposed standard for tradable Ethereum tokens much like ERC-20 with one notable exception, ERC-721 tokens are intended to be non-fungible. This means that each ERC-721 token has a different ID which is suitable for tracking and transacting assets which has spawned the CryptoKitties Ethereum game where each “Kitty” is an independent “product”. (M, 2018)

CryptoKitties

Although CryptoKitties is a game, the very first blockchain game built on the Ethereum network where players buy and trade digital cats and breed in desirable traits it highlighted several weaknesses about how Ethereum scaled. The game is best understood by those who remember, when they were very young and fans of Pokemon, Tamagochi and/or Pikachu except these are “cats” and at home on a decentralised blockchain network. Crypto Kitties are produced to crypto token standard ERC-721 since this is non-fungible to allow for differences and uniqueness in the Kitties. (Hussey, 2019)

ERC-20 Fungible Tokens	ERC-721 Non- Fungible Tokens
Identical Tokens of the same type are identical to one another. They have identical attributes.	Unique Each token is unique and differs from all other tokens of the same type. They have unique attributes.
Interchangeable A token can be interchanged with one another with the same value.	Non-Interchangeable NFTs cannot be replaced with tokens of the same type as they represent unique values or access rights. A university degree or driver's licence would be non-interchangeable.
Divisibility Necessary Fungible assets are divisible into smaller amounts. It is irrelevant which and how units one uses, as long as it adds up to the same value.	Non-Divisible Tokens that are tied to one's identity, like certificates and degrees are not divisible. It does not make sense to have a fraction of a degree or driver's licence.

Table 1. Tokens from a Fungibility Perspective (Voshmgir, 2019)

<https://medium.com/star-bit/introduction-of-ethereum-request-for-comment-%E4%BB%A5%E5%A4%AA%E5%9D%8A%E7%B3%BB%E5%88%97%E6%A8%99%E6%BA%96%E4%BB%8B%E7%B4%B9-erc20-erc721-dbbae53d701f>

Other Research

In this section of the report the reader will discover the additional research and course-work that was completed in order to gain greater understanding of the Ethereum Blockchain.

Coursera, Blockchain Specialization

For a theoretical background into this project and understanding the basics of blockchain and the development tools that one can use I signed up to several online courses from coursera.org. Coursera is an online e-learning platform that is based in america. It was founded by Andrew Ng and Daphne Koller who are both Stanford professors. Coursera offers an enormous catalog of online courses, degrees and specializations.

I completed “Blockchain Basics” and “Smart Contracts” from the University at Buffalo the State University of New York.

Udemy, Blockchain Courses

To further gain my understanding and get a more pragmatic approach to becoming a proficient Ethereum Blockchain developer, which was needed for this project, I self-studied several courses on the Udemy platform. The first of which courses I enrolled in was “Ethereum and Solidity: The Complete Developer’s Guide”, by Stephen Grider. In this course I gained exposure to writing JavaScript code to compile and deploy smart contracts to a blockchain as well as several “gotcha” moments that can happen when working with MetaMask. This is because MetaMask still uses, as of the time of writing this report, a much older version of the web3 framework and for the code that was written to work one must essentially “rip-out”, the provider in MetaMask and replace it with the updated web3 provider instance. This can be done with the following code which can be found from the truffle react box.

```

import Web3 from "web3";

const getWeb3 = () =>
  new Promise((resolve, reject) => {
    // Wait for loading completion to avoid race conditions with web3
    injection timing.
    window.addEventListener("load", async () => {
      // Modern dapp browsers...
      if (window.ethereum) {
        const web3 = new Web3(window.ethereum);
        try {
          // Request account access if needed
          await window.ethereum.enable();
          // Accounts now exposed
          resolve(web3);
        } catch (error) {
          reject(error);
        }
      }
      // Legacy dapp browsers...
      else if (window.web3) {
        // Use Mist/MetaMask's provider.
        const web3 = window.web3;
        console.log("Injected web3 detected.");
        resolve(web3);
      }
      // Fallback to localhost; use dev console port by default...
      else {
        const provider = new Web3.providers.HttpProvider(
          "http://127.0.0.1:8545"
        );
        const web3 = new Web3(provider);
        console.log("No web3 instance injected, using Local web3.");
        resolve(web3);
      }
    });
  });

export default getWeb3;

```

The Ethereum Virtual Machine

The Ethereum Blockchain can be described as a modified version of a general state machine. The first state being the creation of the genesis block and the final state being some arbitrary number of iterations after this creation of the genesis block. An example of state that the Ethereum blockchain maintains is information about account balance, reputation, trust arrangements and other relevant state data about the blockchain itself.

For the world state to change and be updated there is a mining reward for a miner, who is an individual or node on the network that solves the cryptographic puzzle and then a new block is written to the blockchain and the state is updated. This requires an enormous amount of computational power and to incentivise this computation within the network there must be an agreed method of rewarding these miners. This is where the concept of Ether or ETH cryptocurrency solves this issue of incentive as it has a real-world intrinsic value. After a block has been successfully mined a miner gets a small percentage of the transaction value as a reward.

ETH can be subdivided into the following units of value with Wei being the smallest representation of ether. Wei can be thought of as the cent value in the currency of Euro. The following is all the common amounts relative to a single ether:

Subunit Name	Factor	Value
Wei	10^0	1000000000000000000
Kwei	10^3	1000000000000000
Gwei	10^6	1000000000000
Szabo	10^9	1000000000
Finney	10^{15}	1000
Ether	10^{18}	1

From a pragmatic point of view the EVM can be viewed as a large distributed computational model that contains “millions of objects called, ‘accounts’, which have the ability to maintain an internal database, execute code and talk to each other”. (Chinchilla, 2019)

What are accounts?

On the Ethereum blockchain there are two types of accounts. An external account and a contract account. The former being controlled by a public-private keypair and the latter by the source code stored within a smart contract. The Ethereum address is derived from the public key, however the address of the contract is determined from the creator's address and the nonce value of the creator's address. The nonce is a convenient way to describe the number of transactions sent from an address.

What are transactions?

Transactions are a mechanism to describe the process of message sharing between two or more accounts on the Ethereum blockchain network. Message sharing can occur between two external accounts, who are real people, users of the network, or between two contract accounts.

Gas

Each transaction on the network needs some form of computation in order to be processed. This is where the idea of a fee-like system is introduced for the Ethereum blockchain. Gas also acts as a type of load balancer for the network as the more transactions that are occurring on the network, generally the higher the gas price will be, as much of the computational processing power is in use. This is especially true if the user wishes for the transaction to be mined as quickly as possible as this again increases the gas price. The initial gas price is set by the transaction creator and can be explained with the following formula:

$$C(t) = G \times U$$

C = Gas Cost, t = Transaction, G = Gas Price, U = Gas Used

If a transaction runs out of gas an exception is thrown and an out of gas error will be returned. It's important to state this as once gas is used it is gone for good. This is because the machine that mined the block did the work to decide if the transaction succeeded or not. The miner should not be penalized for performing the processing that was requested.

Digital Signatures/Wallets

All interaction with Web3 is accomplished through digital wallet software. Users store their private key, public key and blockchain address, and communicate with one and other through a blockchain wallet. Blockchain wallets allow participants to manage their tokens and tokens are rights of access to the system. When bitcoin or Ethereum is sent or received, the participants wallet is employed to sign off the transaction using the participants private key. Also, every transaction is logged in the blockchain ledger visible to all participants and validated. (Zago, 2018)

An Ethereum wallet, when initially established, generates a participant's public and private key. Firstly, the private key is a randomly generated 256-bit integer. The public key is then derived by elliptic-curve cryptography. The derivation mathematics encrypts the keys in one direction which means that it is possible to derive the public key from the private key but completely impossible, for all practical purposes, to derive the private key from the public. In this way the private key is completely secure. (Zago, 2018)

A participants blockchain address is derived from their public key. This is achieved by employing a different type of cryptography function, with added metadata, than the one used to generate the key itself. This adds an extra level of security, further protecting the private key. So, if someone has the blockchain address and cracked the elliptic-curve cryptography they still

need to crack the second layer of security used to derive the address from the public key. This is of the utmost importance since computer capabilities will always be increasing as time passes.

The blockchain wallet does not store a participant's tokens; it is a common tool to interact with the blockchain and log transactions. However, it does store the keys that are associated with their blockchain address and records all transactions where the public key is involved. The privacy of the private key is paramount and must always be kept a secret and never ever shared. If a participant loses their wallet with no backup to their address and private key, or alternatively, if they lose their private key, they will lose access to their funds. Their tokens will still be on the blockchain, but access will be denied. Also, if the device hosting a wallet is lost, or it breaks down, the participant's funds will not be lost if they still have their private key or "seed phrase" backup. It's a good idea to host one's tokens on the on-line exchange as a bank for your tokens. It's imperative to have one's private key backed up and secure since the public key can be derived from the private key and the address from the public key. (Zago, 2018)

Server-Side Technologies

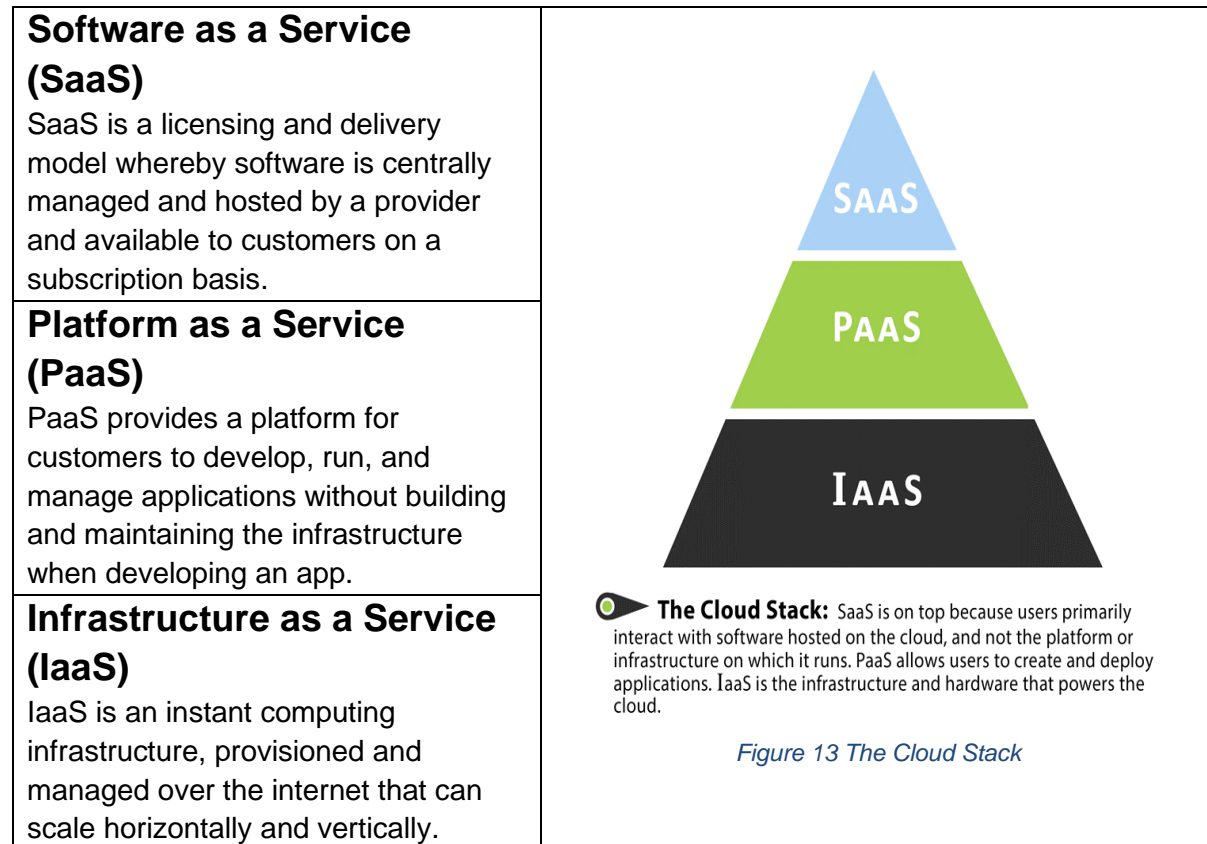
The WiMe project would require a server-side technology in order to be able to allow users to interact with the services that would be provided by the application. The peers would connect to an API endpoint on the server and fetch given data such as user data, transaction data etc. This would mean that the following key areas would need to be addressed:

1. AWS EC2 instance(s) that provide the computing power to run the backend services (e.g. consensus model, etc.)
2. Persistent data storage for application data and user data
3. Compatible framework/middle layer to allow communication between the API endpoints and the clients.
4. Turing complete logic to handle the user transactions when purchases and sales are made
5. User authentication and login system

Overview of Cloud Services

The technology of cloud computing now has been widely adapted and applied to day to day online businesses. In essence "the cloud", is a buzzword for an offsite computing platform that consists of a large set, scalable, cluster of servers that provide computing and storage resources to customers via the internet. Many cloud providers advocate that there is a cost saving and an increased security incentive in utilising their cloud computing platforms. Add reliability and scalability and one has the four most important characteristics that satisfy a customer's demands.

In cloud computing what might be thought of as software and hardware products become services on cloud platforms. There are three categories of cloud computing stack, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).



General Privacy Challenges of Cloud Computing

What type of data are you storing? Where is the cloud located? Is your data in one location? These are important questions since the physical is important criteria in EU law. If your cloud provider operates in more than one jurisdiction, within the EU and outside the EU then customer rights to the privacy of their data could vary greatly.

GDPR Specific Challenges

GDPR states that personal data can be stored no longer than the period required for the prior definition of the purpose. After the withholding period deletion of data can be an issue since complete deletion of data may prove problematic since backups may still exist. So, for security reasons a clear understanding of how the cloud company manages these situations is necessary. In order to cover the event of a security breach, notification obligation in the agreement with the provider. Also, breach events must be defined so there can be no ambiguity after the event.

As mentioned earlier since the cloud has the ability to store data in multiple geographies some personal data may reside in a geography outside the European Economic Area (EEA). If your data resides or a portion of it in a location with no safeguards in place, then safeguards that are appropriate to the nature of the data and its location must be taken

Ownership of the data and control, for security reasons, must lie with the customer and this must be made clear in the contract and stipulated that this applies regardless of the geography of the cloud data (Tolsma, 2018).

Gartner forecasts worldwide public cloud revenues to grow by 17.5% this year. Since this is an area of fast growth, I will choose one of the following organisations as the service provider for this project.

Amazon Web Services

A subsidiary of Amazon that provides on-demand cloud computing platforms, infrastructure and pre-built packages ready to go for developers on a pay as you go basis. Once an individual successfully registers for an account with AWS they are requested to provide the company with credit card details. In this case I used Revolut as I can more easily control the funds for the account through Revolut's service. Revolut is an intermediary for making payments easier by utilizing the capability of creating temporary virtual cards with only the amount of funds needed for a service. In this case the fee required for AWS. When a user has successfully registered their account details with Amazon, they are provided a one-year trial and free access to certain instances, such as EC2 instances from the free tier for the period of one year from the time of account creation. Amazon Web Services appear to be in over 72 regions.

Microsoft Azure

Microsoft Azure is a group of cloud computing services that help organizations meet business challenges. Azure gives companies the freedom to build, manage, and deploy applications on a

very large, global network using the corporate user's preferred tools and frameworks. It is of the uppermost importance that security and therefore user privacy in conjunction with compliance is guaranteed in the cloud computing industry. As far as this trio of needs is concerned Microsoft Azure states that the company employs a proactive approach to all three. Microsoft boasts that they lead the industry in establishing and meeting consistently the security and privacy requirements of industry and operate more data centres than other providers and so they can demonstrate a local presence that some organisations prefer to work with. Microsoft Azure appears to be in over 54 regions.

Google Cloud Platform

Google Cloud Platform consists of a set of physical assets, computers, hard disk drives, and virtual resources, such as virtual machines (VMs). These assets exist in the company's data centres worldwide. These data centres are regionalised. Regions include Central US, Western Europe, and East Asia and each region is divided into zones in isolation to each other within a region. The zones are all named, and the name combines with an identifier with the name of the region.

Resources are distributed and provide several benefits, including redundancy in case of failure and reduced latency by locating resources closer to clients. Google Cloud Platform appear in over 20 regions

Conclusions

Research was done into other cloud providers mentioned above and the next competitor to AWS 47% would be Microsoft Azure who have a similar but inferior offering with a market share of 22%. They also give a one-year trial to their services, however they give a new user “€170 credit to experiment with any Azure service in the first 30 days of access, beyond the free product amounts”. Google is the next competitor to AWS with an even smaller market share of 7%. They have a very similar offering in comparison to Azure and AWS in that they give a one-year trial access to their services along with a €270 offering.

In conclusion from the above, AWS have a more mature service offering and features, especially in regard to blockchain projects, in my opinion are more respectable and trustworthy company and I have more familiarity with their cloud services, and they were the first cloud provider that came to mind that could provide the services required for this project.

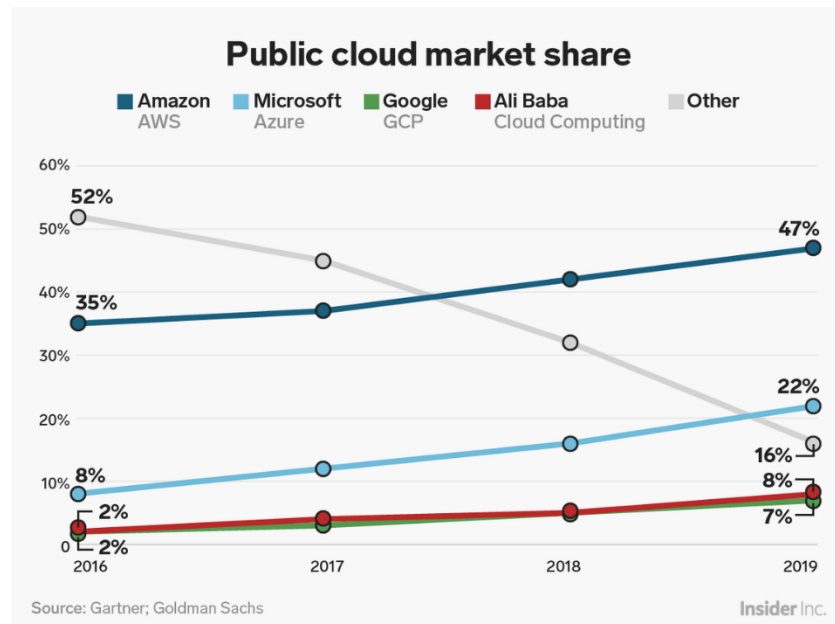


Figure 14 Public cloud market share

As the reader can observe from figure x Amazon AWS has the highest percentage of the market share in public cloud valued at 47% and slowly rising. Next, is Microsoft Azure with a public cloud market share of 22% rising slightly higher than AWS and lastly is Google Cloud Platform (GCP) with a market share of 7%.

Existing Final Year Projects

In this section of the report the reader will be presented with other final year projects from past students who inspired the idea for this project.

Mesh

Student: Hession, Padraig

Description (brief):

An iOS application which locates one another without requiring an internet connection. Communication between devices is performed by relaying information from one device to another until the data reaches its destination. The application makes use of Bluetooth Low Energy to connect devices.

What is complex in this project

Creating a mesh network of devices via Bluetooth had its limitations and difficulties as the framework used ran into connectivity issues and issues of identifying, sending invitations to other users to join the network prove to be difficult as each peer needs a unique id.

What technical architecture was used?

iOS, swift, objective-c, AWS, maps, mesh kit, hype

Explain key strengths and weaknesses of this project, as you see it.

Bluetooth is very slow and has a much shorter range than WIFI. Implemented a find nearby peers, a chatroom, a nice UI all on a mesh network without the need of the internet was impressive.

A Blockchain Ticket Transparency Solution

Student: Vaughan, Robert

Description (brief):

Design and implement of an application to prevent online Ticket scalping and inflating the price of concert tickets etc. Then can buy and sell tickets to friends and family.

What is complex in this project:

Generally, the technical architecture used was quite new and therefore complexity was increased in incorporating the architecture into the solutions.

Creating contracts in blockchain and ensuring that users were able to sell tickets to each other and then getting the near field communication ticket exchange functioning.

Some technologies used were not properly documented or supported as it was deprecated

What technical architecture was used?

Docker, Flask, Solidity, Blockchain, Ethereum, Firebase, Kotlin

Explain key strengths and weaknesses of this project, as you see it.

The project was documented incredibly thoroughly, and no detail was left out.

The project's solution was a great attempt at solving a real problem for concert goers.

Conclusions

This chapter provides insight into the depth of knowledge and learning that was required in order to address all the issues that the project would highlight over the design and implementation period. Therefore, in order to have the technical competency to address these topics, extra courses were relied on to bridge specialist knowledge discrepancies that were present.

Chapter 3 - Experiment Design

Introduction

The following section shall detail the design of the experiment, the software methodologies that were utilised and the key areas of interest in the implementation of the experiment. This is done to separate the experiment into different areas of concern, in order to manage the development of the experiment in a more efficient manner.

Software Methodology

A software methodology refers to the software development life cycle of a project and is an industry “process used by the software industry to design, develop and test high quality software’s” [ref.]. The software methodology also is “governed by the need to ensure a division of tasks into manageable elements and a process that goes through numerous, expanding iterations”. [Wisen, 2019]. Given that this project is quite large and that it is an individual project it made sense to apply these methodologies to the project in order to help keep track of my progress.

The core software methodology that was utilised in the project borrowed concepts from Test Driven Development (TDD) and the Agile Development process. This is because the WiMe system is not just composed of a single system. It is composed of three distinct systems which are:

1. A React front end web client
2. A REST server that clients use to request services from the backend
3. A Distributed Ethereum blockchain backend

The design of the system will be described later in this chapter with class and sequence diagrams.

Test Driven Development

For the Ethereum blockchain domain of the project TDD was heavily utilised as solidity code must be robust, secure and as bug free as possible. TDD best represents this type of outcome and for a developer to achieve this outcome one must follow its principle guidelines which are:

1. An aspect of the programme should be described by writing a single unit test
2. The unit test should fail since this feature has not been implemented yet
3. This should be rectified with the simplest implementation
4. Refactor the code again to ensure it is simple as possible
5. Repeat the process

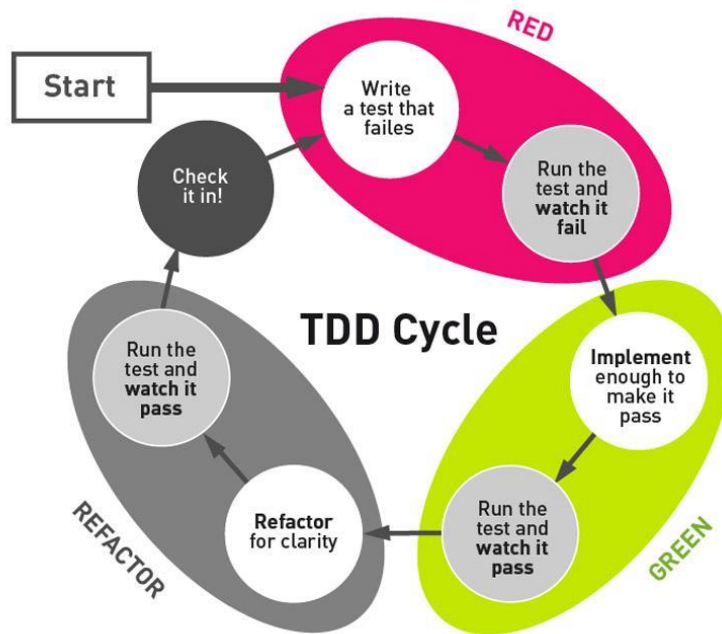


Figure 15 Test Driven Development [<http://bellintegrator.com>]

To test solidity code one can, write unit tests in JavaScript using the Mocha testing framework or one can write unit tests natively in the solidity programming language. Mocha and JavaScript were used to test the solidity smart contracts.

Although there are negatives to TDD as I discovered when writing the tests. Negatives such as false positives. This can occur since at the beginning of writing a test, there has not been a written implemented unit test, therefore, it will automatically show a pass simply because one hasn't told it to do anything and therefore the test assumes that this is what was expected test outcome to be and so the test is passed but, of course, this is not what the desired outcome should have been. This can be a flaw with TDD if one is not careful in how the unit tests are written when using this software methodology.

"Kent Beck's simplest code check" was followed to ensure that the development process of the Ethereum backend was arrived at and the code was expressed in the clearest, simplest and understandable manner. It is intended to subject the code to automated unit tests in order to have the minimum number of components to pass each test and this results in less duplication of code which is essential for such a large experimental system.

Agile Development

The React web client and the REST server were developed using the agile software methodology approach. This at the time of design made the most logical sense as the Ethereum Blockchain would have been mostly implemented and defined at this stage and therefore the React front end and REST server would need to implement and reflect the requirements that the Ethereum Blockchain has defined. Because of this the agile development process made the

most logical sense as its strengths are that the planned deliverables must be agreed upon before a “sprint” is started. This then means the development lifecycle is relatively fast in comparison to TDD.

Therefore, when a sprint begins all efforts are focussed on delivering the features agreed in the sprint time frame. It is normal for sprints to last for two weeks they are rarely of longer duration. All planned deliverables may not be achieved in the sprint duration and this is the case then the work gets rescheduled and the lessons learned help in further sprint planning. As deliverables are worked through the process is constantly reviewed and critiqued by the participants including the customer with the sprint’s product demonstrated to the customer. The customer’s position is one of great importance to the success of the agile methodology.

However, in this experiment and the WiMe system the developer plays both roles. This means that the developer and customer are the same person.



Figure 16 Agile Methodology [https://www.getmailbird.com]

In line with any project manager lifecycle the agile software development methodology has the following positives:

1. The customer is fully involved throughout the entire process of the project
2. The customer gains a strong sense of being an important member of the project
3. Customer focussed as the customer is involved more throughout the entire process
4. The process is agile and less time to market is achievable through the agreement of more basic requirements with the customer.

Also, it's important to note that the following negatives can be true also:

1. Very busy customers find difficulty in finding the time required to be closely involved in the process.
2. The process demands complete dedicated involvement by all participants
3. The process is less easy managed if all the sprint team are not located in the same location

4. The process does not always deliver all deliverables on time increasing cost due to further work.
5. Due to the close customer input, project creep can be a feature with some customers

Because the process is iterative, this can lead to refactoring if the architecture and design have not been properly scoped. However, without the refactoring quality can suffer greatly. This can become more pronounced with project integrations of scale.

SCRUM Development

SCRUM is another popular derivation of the agile development methodology that also encompasses the sprint lifecycle process. Personally, I was exposed to this process during my internship placement and the SCRUM master during my placement referred to it as “lightweight process framework”, this implies that the overhead costs are kept to a minimum to ensure that more time can be invested into a given process. Scrum too is an iterative process and useful for complex software development. The process handles changes to product specification very well and is well suited to this type of development environment. Scrum is extremely useful as it promotes for a superior quality of deliverables by:

1. Increasing the quality of the deliverables
2. Handles changes more easily
3. Easier to estimate accurately
4. Allows for greater control management of the sprint

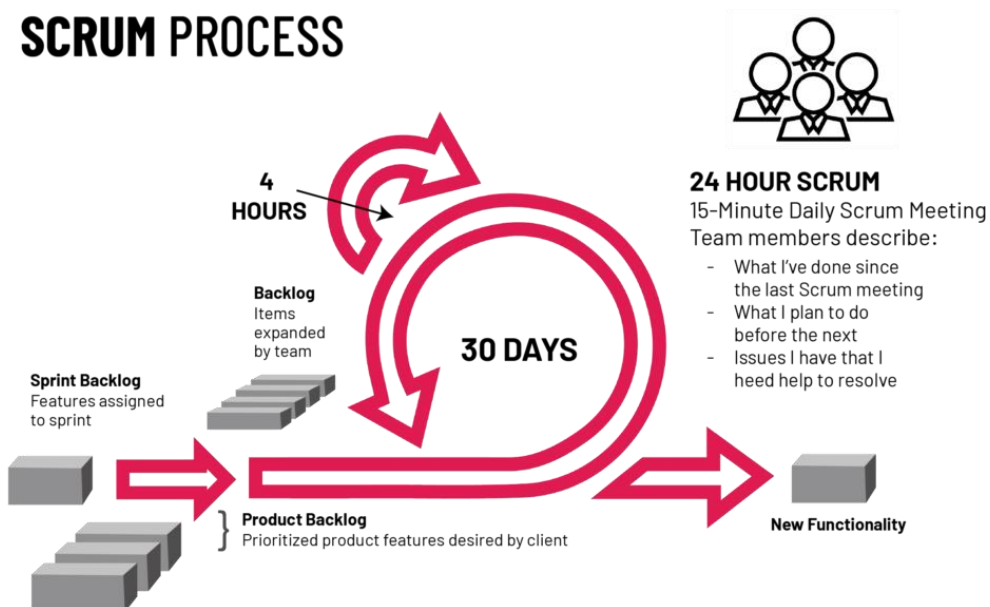


Figure 17 SCRUM Process

Overview of System

This section of the report describes, and the various components involved in the WiMe system and how the multiple areas of the system interact with one another. This design was used as a guideline for the system architecture to be followed in the development phase of the project.

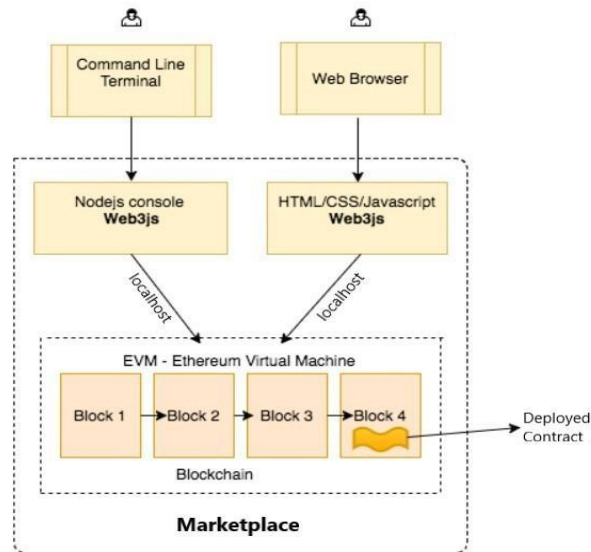


Figure 18 Developer and User Architecture

How to interact with the Ethereum Virtual Machine

An application binary interface (ABI) is how a program can call a smart contract method and return its data from the Ethereum blockchain. An Ethereum smart contract gets deployed to the blockchain as bytecode that is stored on the Ethereum Virtual Machine (EVM). This process is very similar to how Java code gets loaded by the class loader, which then compiles this into classes or bytecode for the Java Virtual Machine (JVM). For this reason, all interactions are with the ABI and not the EVM bytecode on the network. This also abstracts some of the complexity from the developer and results in a single interface that is compatible with any programming language that supports web3.

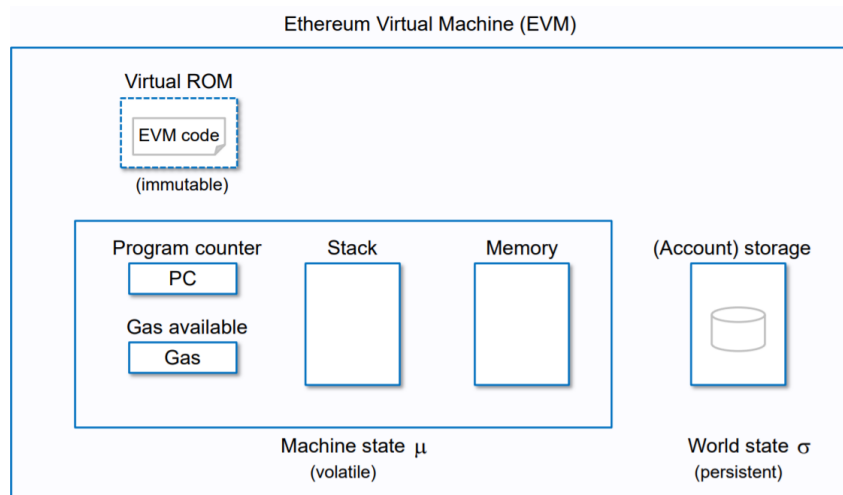


Figure 19 Ethereum Virtual Machine

User identification on Ethereum is determined by a user's Ethereum address which is derived from a user's public key in their keypair. The user's wallet that stores all their transactions with blockchain is just an encrypted version of the user's private key. It's important to note that a developer, user or blockchain advocate should never broadcast their private key across a network, even if encrypted. The reason for this should be obvious but the wallet is a communication tool to access a user's transactions and therefore access their funds. If this wallet or private key as mentioned previously is stolen or decrypted a user loses all their money and there is no way to retrieve their money as the blockchain is anonymous.

Initial Android Web3 Architecture

In the beginning phase of the design process of the WiMe system it was envisaged that the end user would interact with the Ethereum Blockchain from the front end with an Android application. However, the web3j java framework that Android requires to be able to transact with the Ethereum blockchain was deemed to be unusable and could not perform to its advertised stated capability. However, using the web3j command line tool was functional and straightforward to create a new wallet for a user, except that this wallet is a standalone wallet and not associated within a given programme application which was required for the WiMe system.

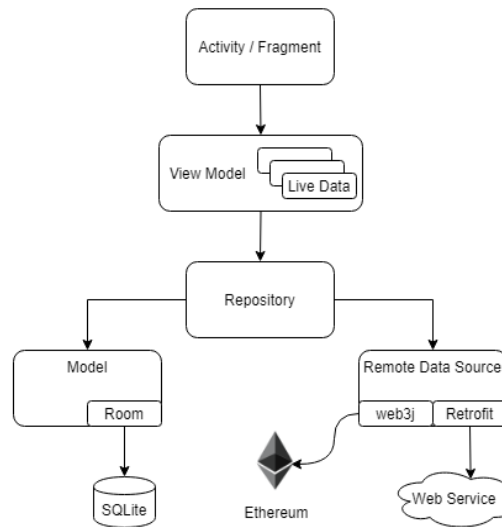


Figure 20 Android MVVM Architecture with Blockchain

It was incredibly difficult to create an Ethereum wallet for an android device and because of this reason it was decided to create a web application instead. It was also not as straightforward as expected due to the android development team changing how permissions are managed on android devices as of Android 10 (API 29), this is important for the system as the wallet file must be cryptographically stored on the device. It's also important to state that no official documentation exists for the web3 java framework. Therefore, to understand the web3j API it involved reading the open source code and deciphering what and how the code works and for this project it was deemed to be far too time consuming of a task for such a strict deadline. Because of these reasons it was decided to create a react web application using the JavaScript web3js framework instead. However, the creation of an Ethereum wallet was successfully achieved using the web3j java framework but not without much invested time and effort. If the reader would like to know more about creating an Ethereum wallet in Java they can discover this information in the next chapter.

As the reader can be seen in Figure x. The android client forwards a HTTP request to the REST server and then the REST server calls the methods the application binary interface (ABI). The ABI is the only way to transact, using its endpoints and then the node server transmits Ethereum Blockchain through its Application Binary Interface (ABI). The ABI is a JSON object that has been compiled from a solidity smart contract. The ABI contains all the web3 methods and the user defined methods that were programmed on the smart contract. This is a three-tier architecture, with the node server providing an entry point to the ABI.

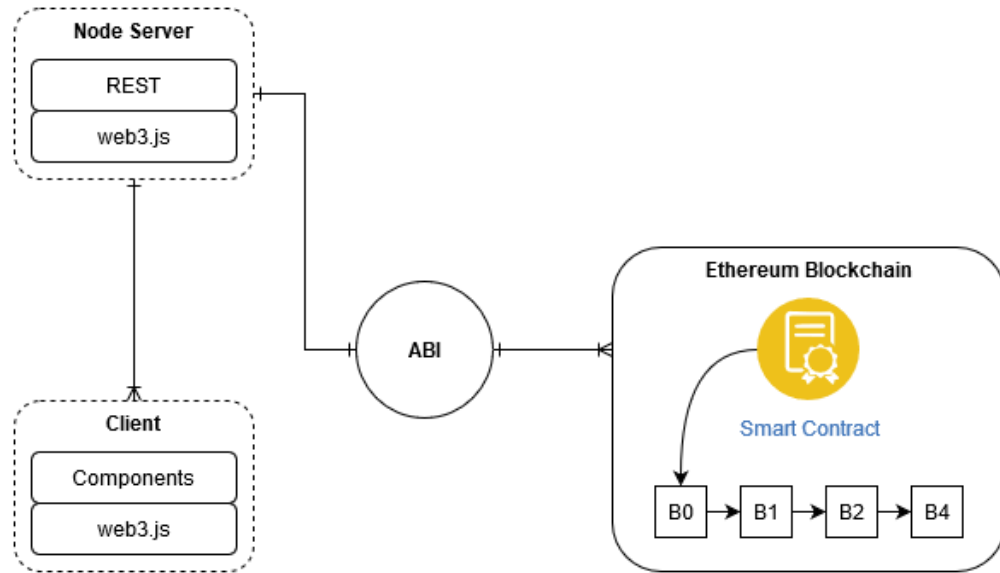


Figure 21 WiMe Architecture

Use Cases

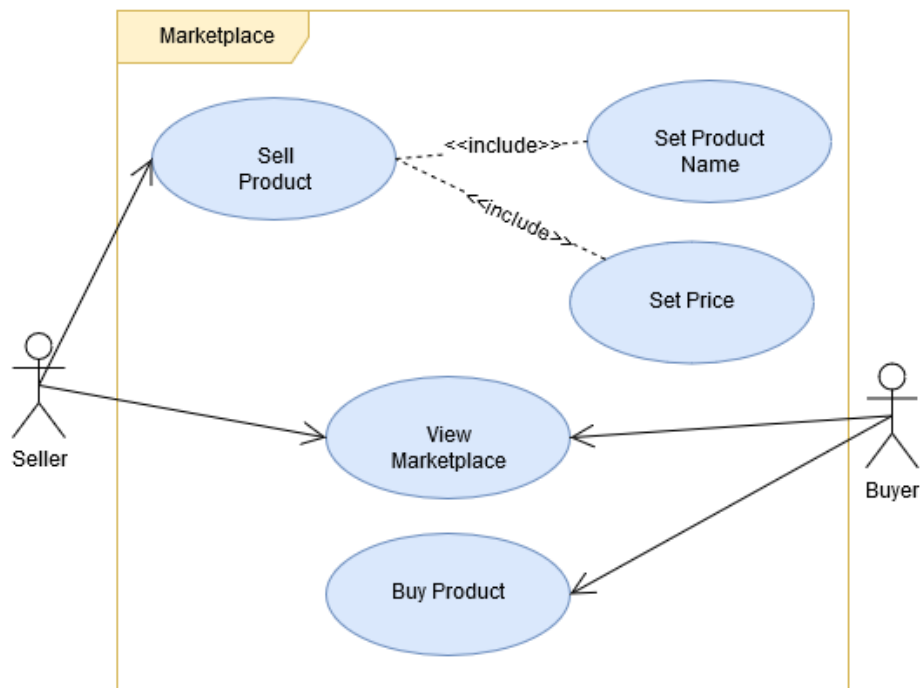


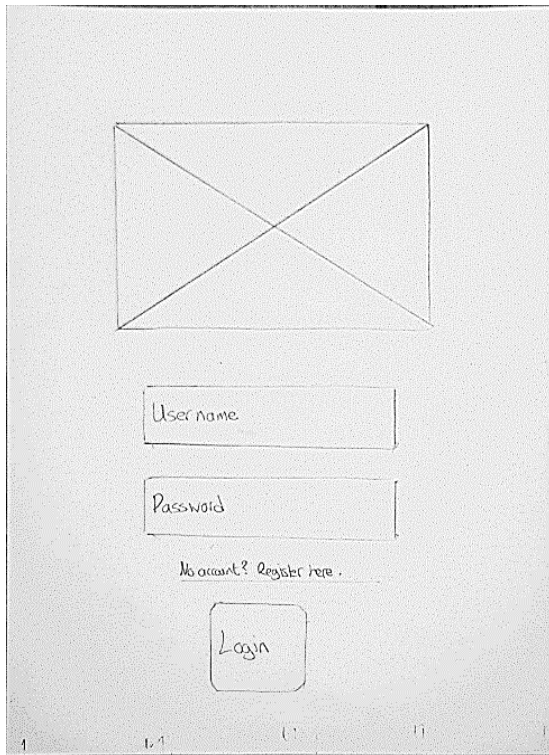
Figure 22 Marketplace use case

For example, the user, through interacting with the smart contract, which is deployed to the Ethereum Blockchain, can purchase a product, sell a product and buy and sell tokens. Therefore, the REST server should provide a service for a user to interact with the smart contract's methods. Also, for user experience the React client should provide a user interface to make interacting with the services provided by the REST server simple and understandable for the end user under the guidelines of Nelsons Heuristics.

Android Front-End

Low Fidelity Prototyping

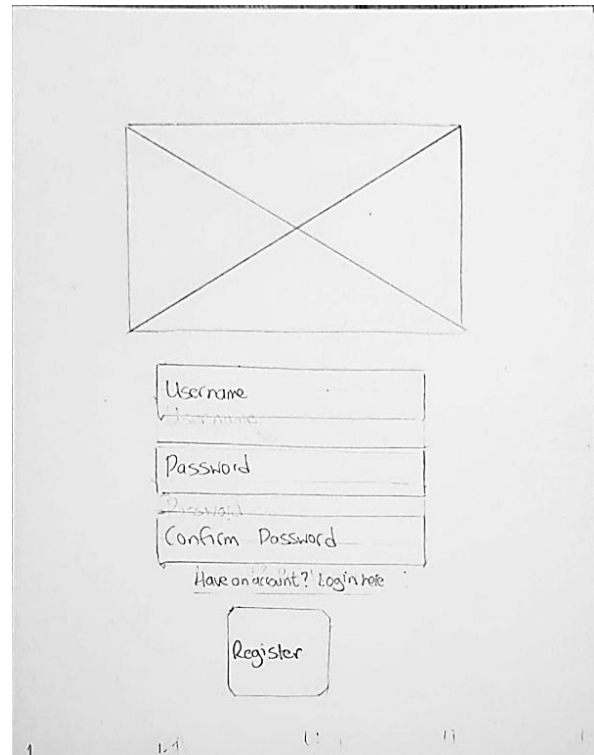
Login



A hand-drawn low-fidelity prototype of a login form. At the top is a large rectangle with an 'X' inside, representing a brand logo. Below the logo are two rectangular text input fields. The first is labeled 'User name' and the second is labeled 'Password'. Under the password field is a text view that reads 'No account? Register here.' At the bottom is a rounded rectangular button labeled 'Login'.

The login form to the application encompasses a brand logo in the central upper region. Followed underneath this by an edit text field for a username to be entered as well as a password field. There is a text view area that also acts as a link to another android fragment. Lastly a submit login button

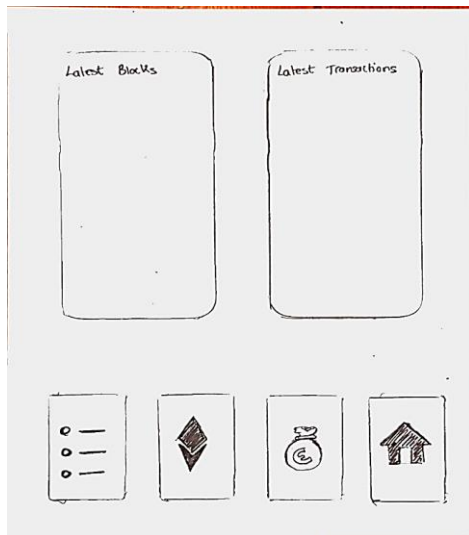
Register



A hand-drawn low-fidelity prototype of a registration form. At the top is a large rectangle with an 'X' inside, representing a brand logo. Below the logo are three rectangular text input fields. The first is labeled 'Username', the second is labeled 'Password', and the third is labeled 'Confirm Password'. Under the 'Confirm Password' field is a text view that reads 'Have an account? Login here'. At the bottom is a rounded rectangular button labeled 'Register'.

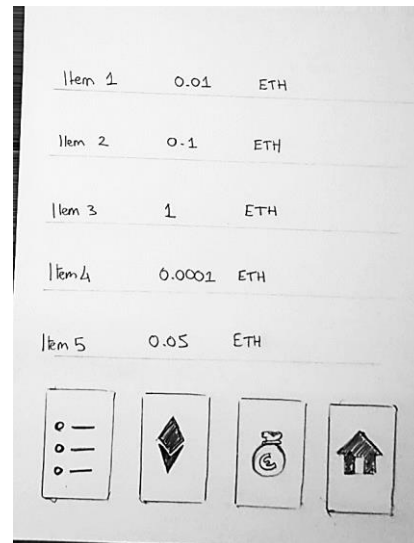
The signup or registration fragment has an image of the brand logo as well as an edit text field for the user to enter their username, as well as a password field for them to associate their account with a password and then a confirmation of their password. A text view area is present in case the user already has an account. Lastly a register button to send the users information to the database.

Home



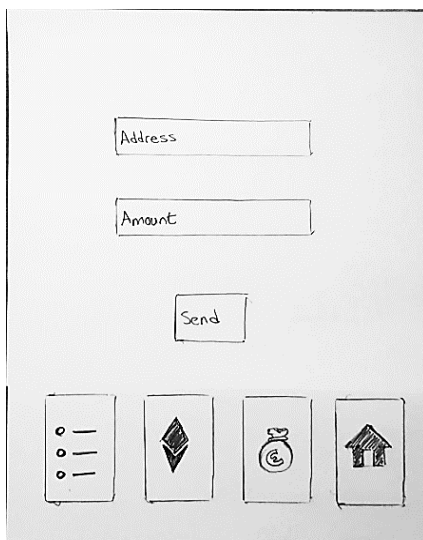
The homepage for the application shows the latest blocks on the left and the latest transactions on the right along with a bottom navigation. Going from left to right, my transactions, send ether, my balance and home.

My Transactions



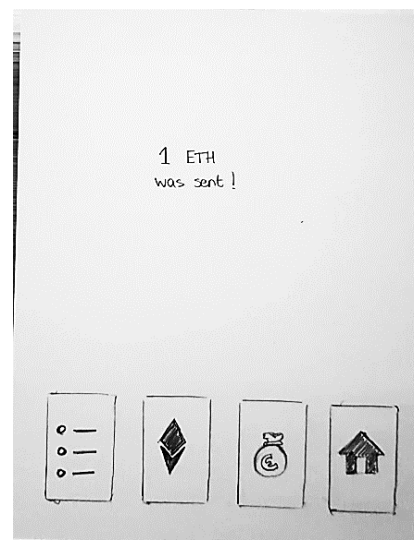
The design above shows the users latest transactions with a scoping of the last five that were added to the blockchain. The name of the item, its price then in Ether (ETH)

Send Ether



The following screen is the design for sending ether to someone on the blockchain. It contains an edit text field to enter their Ethereum address and a numeric field to enter the mount of Ether to send

Confirmation of Sent Ether



The following screen contains a text view to output to a confirmation message that the transaction block has been added to the blockchain and is about to be mined by the network

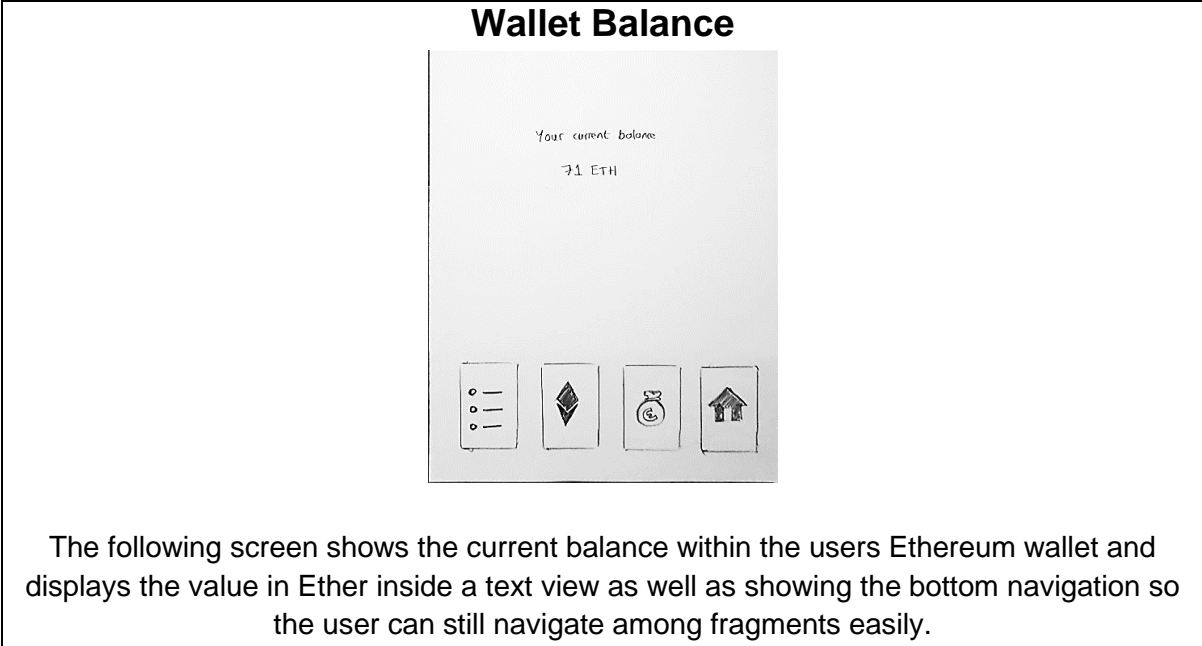
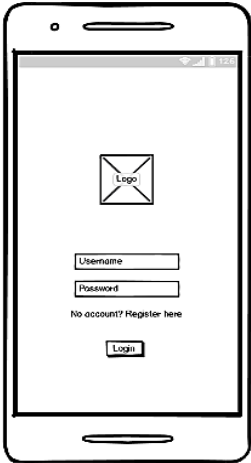
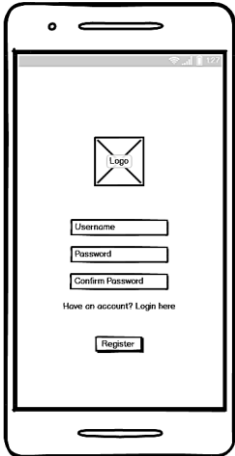
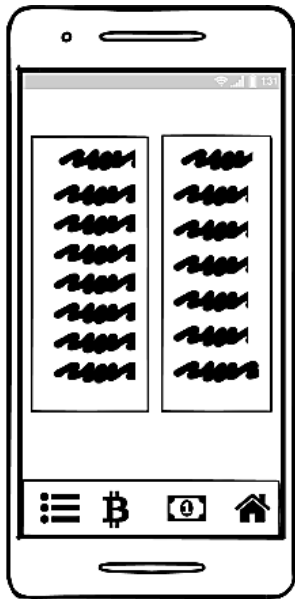


Figure 23 Low Fidelity Prototype

High Fidelity Prototype

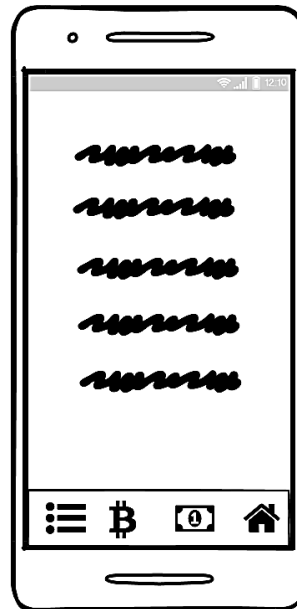
Login Fragment	Registration Fragment
	
<p>The login form to the application encompasses a brand logo in the central upper region. Followed underneath this by an edit text field for a username to be entered as well as a password field. There is a text view area that also acts as a link to another android fragment. Lastly a submit login button</p>	<p>The signup or registration fragment has an image of the brand logo as well as an edit text field for the user to enter their username, as well as a password field for them to associate their account with a password and then a confirmation of their password. A text view area is present in case the user already has an account. Lastly a register button to send the users information to the database.</p>

Home



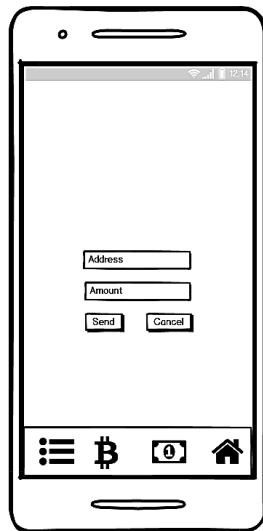
The homepage for the application shows the latest blocks on the left and the latest transactions on the right along with a bottom navigation. Going from left to right, my transactions, send ether, my balance and home.

My Transactions



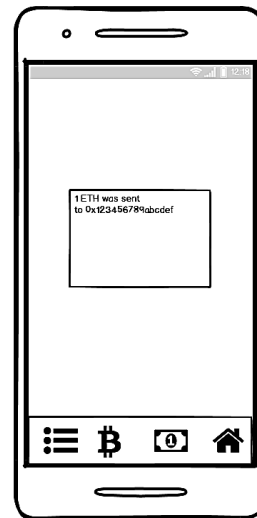
The design above shows the users latest transactions with a scoping of the last five that were added to the blockchain. The name of the item, its price then in Ether (ETH)

Send Ether



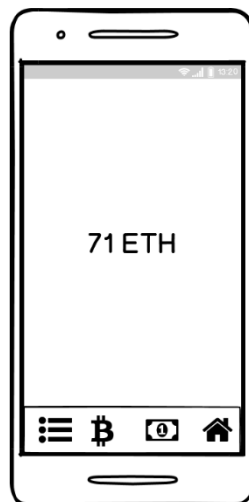
The following screen is the design for sending ether to someone on the blockchain. It contains an edit text field to enter their Ethereum address and a numeric field to enter the mount of Ether to send

Confirmation of Sent Ether



The following screen contains a text view to output to a confirmation message that the transaction block has been added to the blockchain and is about to be mined by the network

Wallet Balance



The following screen shows the current balance within the users Ethereum wallet and displays the value in Ether inside a text view as well as showing the bottom navigation so the user can still navigate among fragments easily.

Figure 24 High Fidelity Prototype

React Front-End

Low Fidelity Prototyping

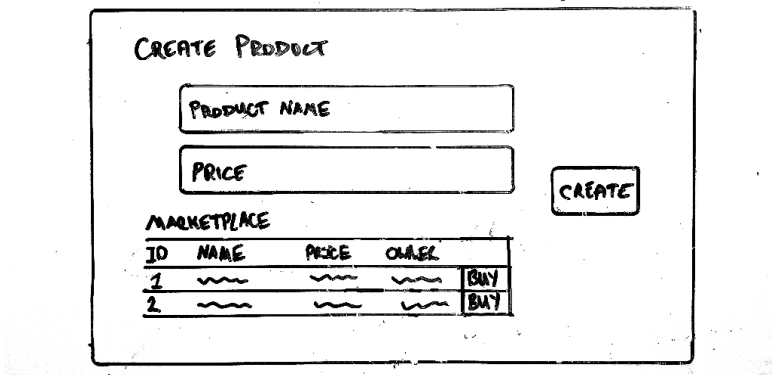


Figure 25 Low Fidelity Prototype

High Fidelity Prototyping

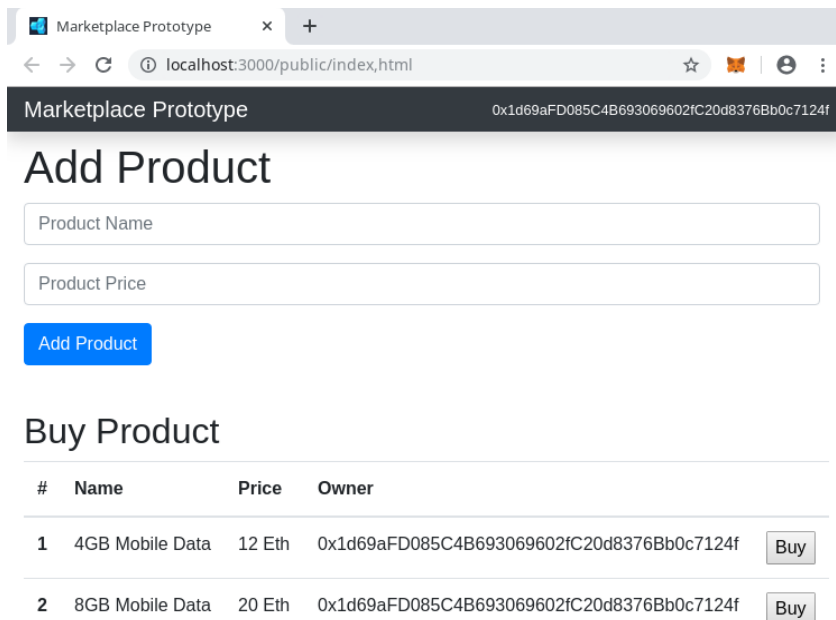


Figure 26 High Fidelity Prototype

Middle-Tier

The middle tier of the application had only two possibilities as I needed to expose the blockchain backend as a web service to its client. The two possibilities that I could choose from are Simple Object Access Protocol (SOAP) and Representational State Transfer (REST).

SOAP vs REST

As with many protocols and architectures in the computer science and the technology domain there are trade-offs with trying one approach over another. This is clear in the case of SOAP vs REST. The first main difference between SOAP and REST is that SOAP is a protocol that exchanges only Extensible Mark-up Language (XML) and requires higher bandwidth and resources. This is in stark contrast to REST, being that REST is an architecture style, not a protocol. REST also is more flexible, being that it allows the exchange of many different data formats. These data formats can be XML, Hypertext Transfer Protocol (HTTP), JavaScript Object Notation (JSON) and even raw text. (Oracle Technology Network, -)

In relation to security and ACID (atomicity, consistency, isolation, durability), SOAP supports more security features than REST, for example, Web Services Security in comparison to REST which only supports Secure Sockets Layer (SSL) or Transport Layer Security (TLS) and Hypertext Transfer Protocol Secure (HTTPS). To clarify, SOAP supports Web Services Security, SSL/TLS and HTTPS. REST only supports SSL/TLS and HTTPS. (Oracle Technology Network, -)

What is Web Service Security?

Web Service Security offers a general-purpose mechanism for associating security tokens with message content. The specification defines three approved token types:

1. UsernameToken Profile (UTP)
2. X.509 Certificate Token Profile
3. SAML (Security Assertion Markup Language) Token Profile

The above profiles define how the type of token is used within the WS Security specification. Taking the (UTP) for instance illustrates how Web Service Clients can produce username tokens to identify the requester by username with the option of supplying a password.

JavaScript Object Notation (JSON)

JSON is built on two structures:

- Key and value pairs collection in a variety of languages is experienced as an object, a record, a hash table, keyed list or associated array etc.
- Languages in ordered lists of values is experienced as a sequence, array, or vector

Example of JSON

```
{
  "host": "localhost",
  "port": 3030,
  "public": "../public/",
  "paginate": {
    "default": 10,
    "max": 50
  },
  "mongodb": "mongodb://localhost:27017/api"
}
```

ACID

In the context of transaction processing, A.C.I.D. "refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability"[ibm ref]. If these four properties are complied with the transaction will be carried in the most efficient manner. Suppose one has a consistent database, and a transaction takes then if the four properties of A.C.I.D. are complied with then the database will still be consistent afterwards. Also, if the database is already consistent then all attention must be focussed on the transaction itself and not the already consistent database.

Atomicity

If the database is consistent and a transaction is about to be performed then the transaction with a database, in order to maintain database consistency, the transaction must comply with the Atomicity property of A.C.I.D. so the transaction must all take place at the same time. That is, all the changes are performed together, or none of them are. This means that all the transaction's instructions must be executed together.

Consistency

A database is shown to have consistency if after some adjustments made to it, it still has the same total value before and after.

Isolation

If a number of transactions are going through at the same but none of the transactions are interfered with or with each other one can say that all the transactions enjoyed isolation. This makes for efficient handling since with this principle in mind the database will have the maximum transactions being processed at any one time.

Durability

If changes have been to the value of a database by someone, then the database must not, at any time, be changed in any way, even back to its original state by someone else, ever. This

makes perfect sense since if no one else interferes with the database it could be readjusted back to the original proper value with confidence.

REST API Structure

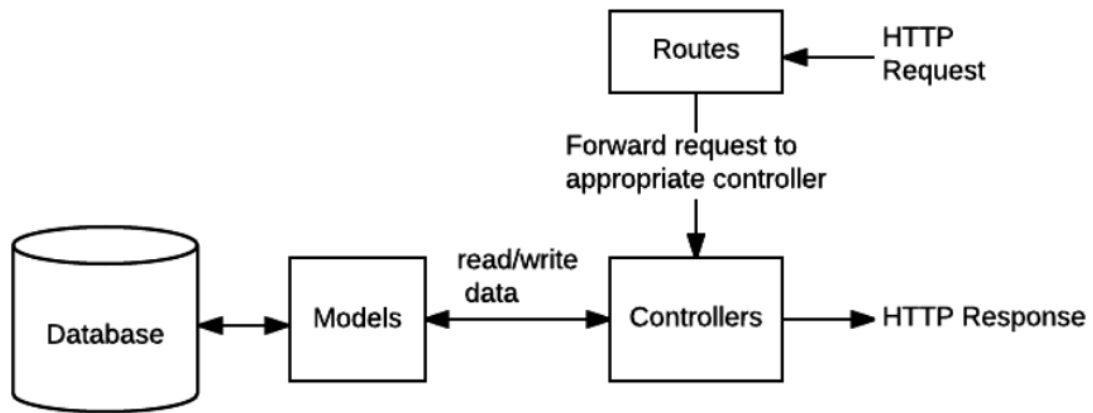


Figure 27 REST API Structure

WiMe REST API Sequence Diagram

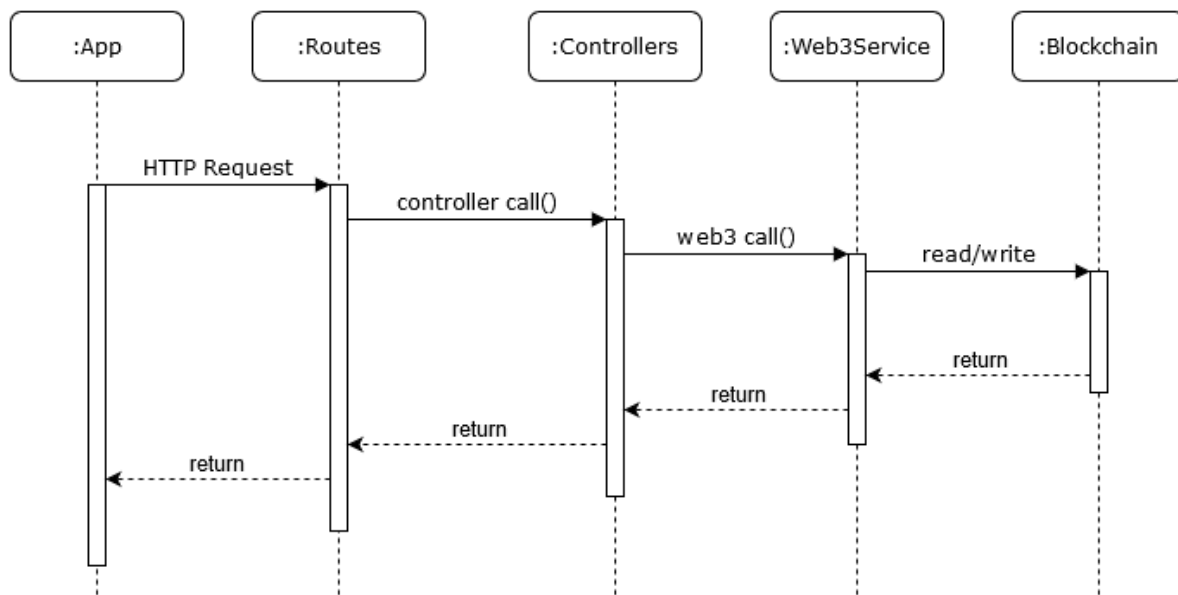


Figure 28 WiMe REST API Sequence Diagram

Back-End

Below is the sequence diagram for the user purchase of a token and then the use of this token to purchase a product from the marketplace

Token and Product Purchase

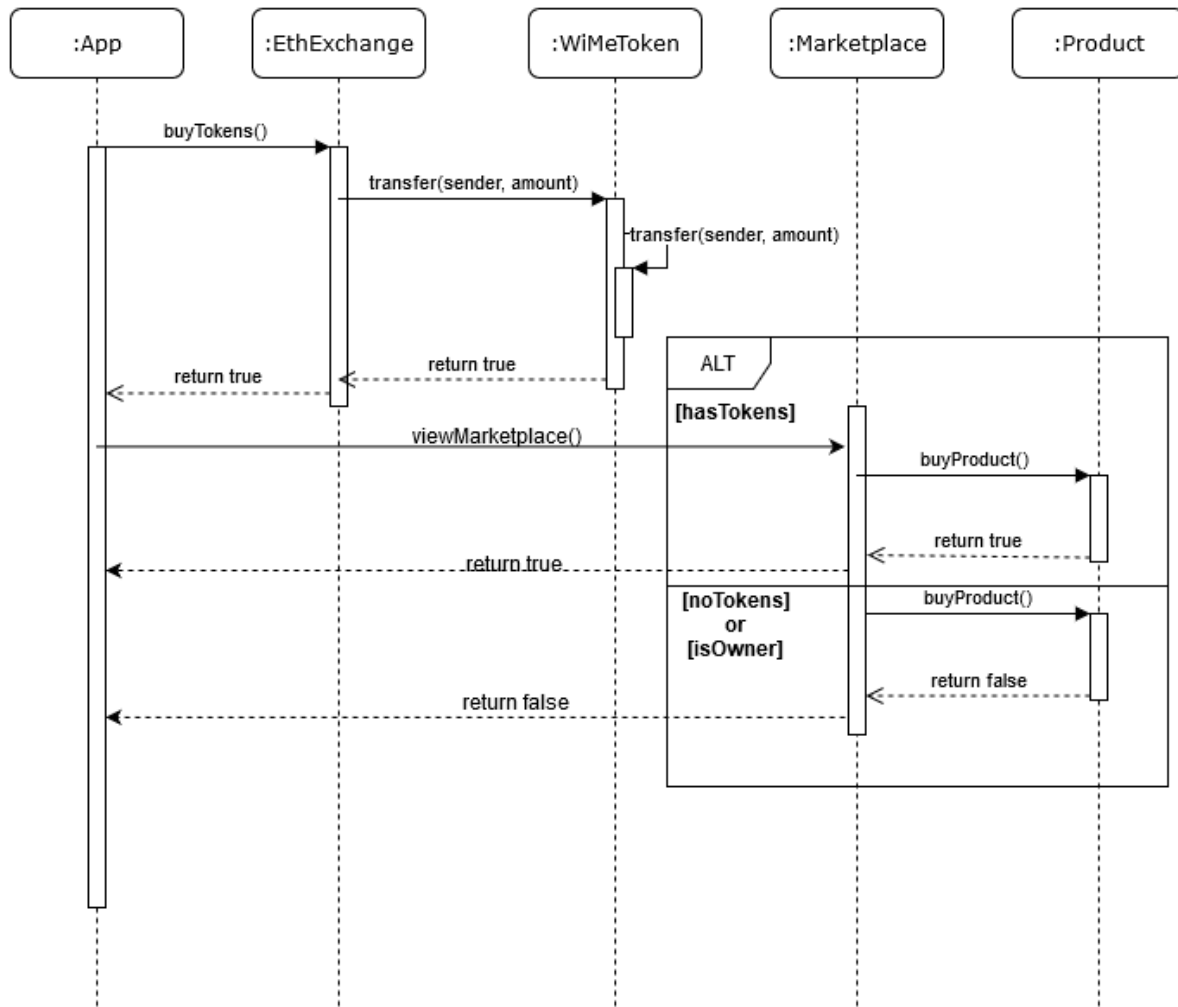


Figure 29 Token and Product Purchase

Token and Product Sale

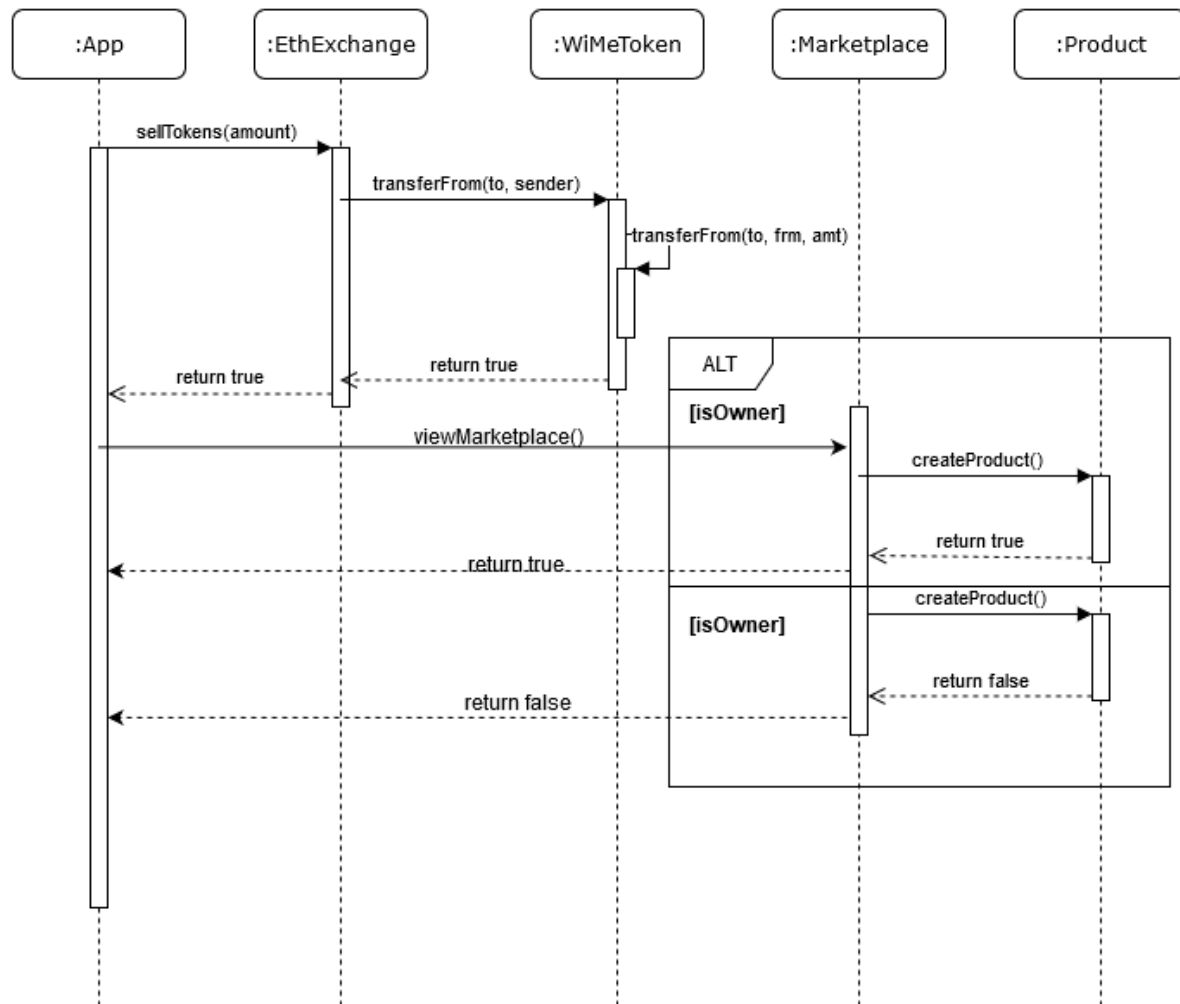


Figure 30 Token and Product Sale

Smart Contract Design

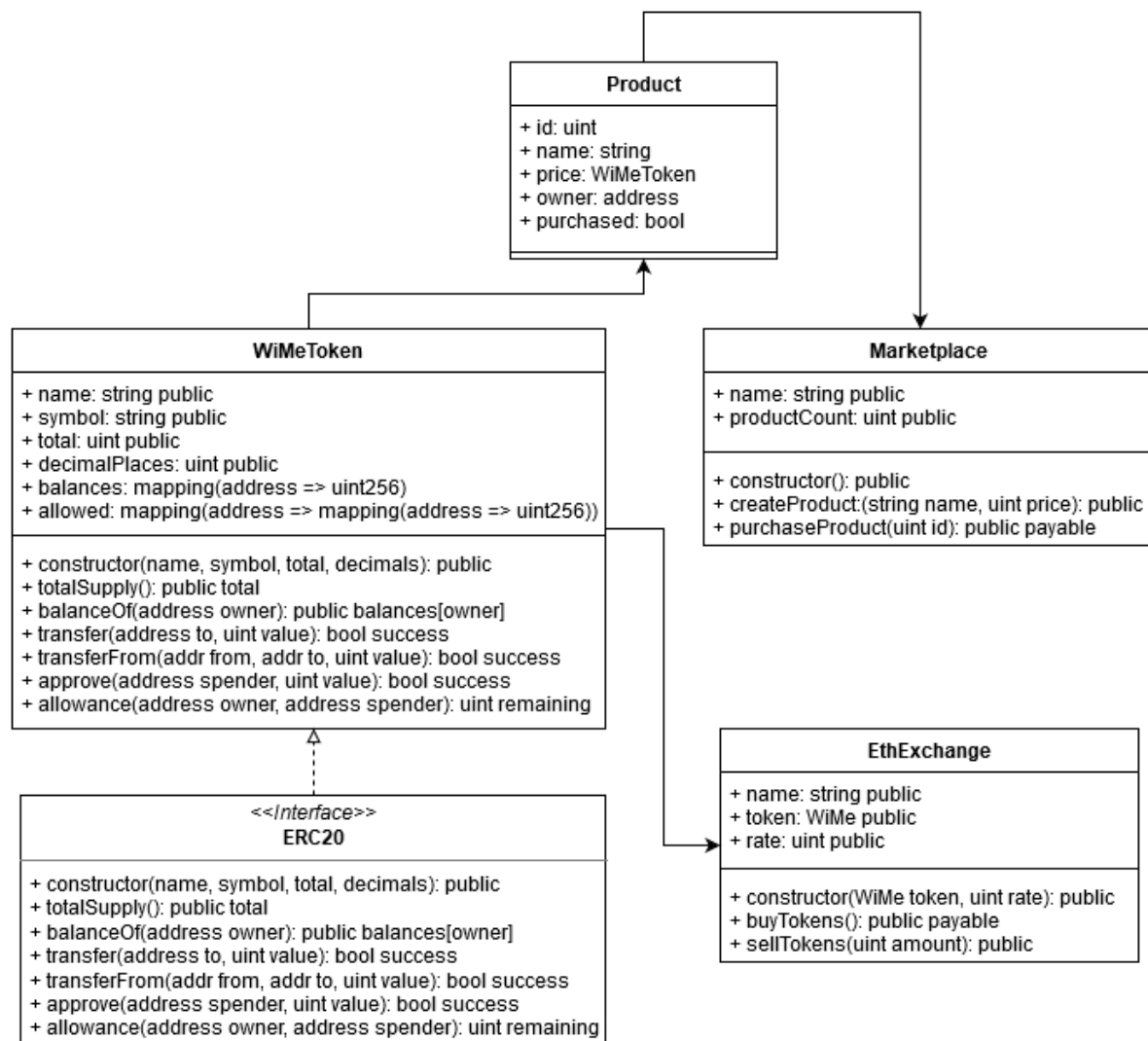


Figure 31 Smart Contract Design

Chapter 4 - Experiment Development

Introduction

Within this given chapter, an overview of the development process is presented to the reader. The tools used will be described, as well the installation process and any issues encountered.

Software Development

Geth

Geth or Go-Ethereum is a command line interface that allows you to run and operate a full private Ethereum node. Geth is implemented in Go and allows you to mine blocks, to generate Ether, to deploy and interact with smart contracts, to transfer funds, to inspect block history, to create accounts, etc. Geth can be used to connect to the public Ethereum network, also called main net, or to create your own private network for development purposes, which will be demonstrated later in this section of the report. The benefit of using Geth in the development process is that it allows one to fully test a distributed application before deploying it to the main net. Because there is a wallet on the browser the REST endpoint only needs to have the same functionality as the smart contract there is no need for a login and register system all that is handled on the Meta Mask wallet within the browser. Below is the startnode script to fire up the private ethereum node and it will be explained at how it works following the image:

```
1 geth --networkid 2165 --mine --minerthreads 2 --datadir "." --nodiscover --  
rpc --rpcport "8545" --port "30303" --rpccorsdomain "*" --nat "any" --  
rpcapi eth,web3,personal,net --unlock 0 --password ./password.sec --  
ipcpath "~/ethereum/geth.ipc" --allow-insecure-unlock
```

Ganache

Ganache is an in memory local development blockchain and for the project it was used initially quite heavily as development with this command line tool is simple, fast and efficient. However towards the end of the project I switched to using a private ethereum node with geth, as I was curious about creating my own Ethereum node. Ganache has a graphical version too which can be seen below:

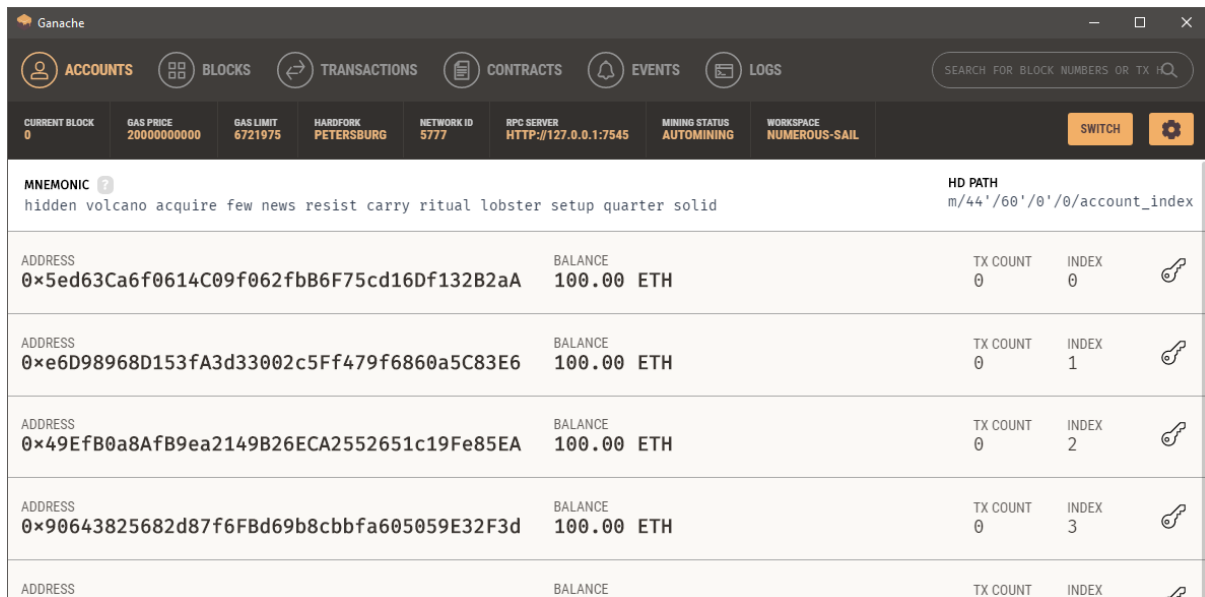


Figure 32 Ganache GUI

Ganache gives a developer ten test accounts all funded with 100 Ether and abstracts a lot of the more complex if not all of the details in creating a private ethereum node. Through the user interface (U.I.) a developer can view the blocks on the local blockchain, the transactions, the deployed contracts and any logs or events emitted.

Truffle

This is another blockchain development tool that abstracts and simplifies the compilation, migration testing and configuration of an ethereum blockchain development project. I experimented with this in the prototyping stage however I hadn't really a good understanding at the time of how to use the configuration file effectively for this tool. So, because of this reason for the prototyping and the beginning development phases of the final experiment development I used plain JavaScript to compile and deploy my smart contracts to the local instance of the blockchain, i.e. ganache. For the project I used the latest version of solidity which is v0.6.4 during the time of the project. Below is the compile script and deployment script that was created for this purpose:

JavaScript Code to Compile Solidity Contract

```
const path = require("path");
const solc = require("solc");
const fs = require("fs-extra");

const buildPath = path.resolve(__dirname, "build");
const contractsPath = path.resolve(__dirname, "contracts");

const createBuildFolder = () => {
```

```

    fs.emptyDirSync(buildPath);
  }

  const buildSources = () => {
    const sources = {};
    const contractFiles = fs.readdirSync(contractsPath);

    contractFiles.forEach(file => {
      const contract = path.resolve(contractsPath, file);
      sources[file] = {
        content: fs.readFileSync(contract, "utf8")
      }
    });

    return sources;
  }

  const input = {
    language: "Solidity",
    sources: buildSources(),
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
        details: {
          deduplicate: true,
          cse: true,
          constantOptimizer: true,
          yul: true,
          yulDetails: {
            stackAllocation: true
          }
        }
      },
    },
    evmVersion: "petersburg",
    outputSelection: {
      "**": {
        "**": [
          "abi",
          "evm.bytecode"
        ]
      }
    }
  }

```

```

    }
  }

  const compileContracts = () => {
    const compiledContracts =
JSON.parse(solc.compile(JSON.stringify(input))).contracts;

    for (let contract in compiledContracts) {
      for (let contractName in compiledContracts[contract]) {
        fs.outputJSONSync(
          path.resolve(buildPath, `${contractName}.json`),
          compiledContracts[contract][contractName],
          {
            spaces: 2
          }
        )
      }
    }
  }

  (function run() {
    createBuildFolder();
    compileContracts();
  })();

```

There is quite a lot going on in this compile script so let me take you through it section by section. Firstly we create an empty build directory to store the compiled contracts on the local file system. Next we start the build phase, this is done by creating an empty array of sources where it will later store a reference to each contract that's read from the file system. After this step the following step is to set the options for the solidity compiler. There's quite a lot of options here but I'll mention the most important ones. The compiler is told which language to compile, along with the sources object, it's also told to optimise the compilation process and that we want to compile to one of latest ethereum virtual machines called "petersburg", lastly we only are concerned with the abi (which the calling code uses) and the bytecode (which the blockchain uses) so we specify to only output these from the compiled contract. Next the compilation input is converted from JSON object into a string for compatibility reasons with the solidity compiler. The contracts field is parsed from the compiled source. In the final step the compiled script(s) are written to the file system and stored under the build directory.

JavaScript Code to Migrate Solidity Smart Contract to Blockchain

```

const HDWalletProvider = require("@truffle/hdwallet-provider");

```

```

const Web3 = require("web3");
const compiledContract = require("../build/MyContract.json");

const mnemonic = "sugar little silver system fog garlic actor finish follow
omit clutch electric";
const apiKey = "45459d75dee947218d2503d567d6307b"
const url = "https://rinkeby.infura.io/v3/"

const provider = new HDWalletProvider(mnemonic, url + apiKey);
const web3 = new Web3(provider);

const deploy = async () => {
  const accounts = await web3.eth.getAccounts();
  console.log("Attempting to deploy from account", accounts[0]);

  const result = await new web3.eth.Contract(compiledContract.abi)
    .deploy({ data: "0x" + compiledContract.evm.bytecode.object })
    .send({ from: accounts[0] });

  console.log("Contract deployed to", result.options.address);

  provider.engine.stop();

  return result;
};
deploy();

```

The logic behind the migration script is a bit less complex than the compile script. Firstly one must get a reference to the compiled contract which is a JSON ABI object. Next the mnemonic, api key and url for the infura provider are included as the wallet provider must have a reference to which ethereum network it's going to be published to. Then this provider object is set as the provider to web3. After this we need to use an account as a deployer, as it costs gas to deploy a smart contract. It is common practice to use the zeroth account i.e. the first account specifically for this purpose, otherwise known as the coinbase account. Next we get a new instance of the contracts application binary interface (ABI) and we deploy the bytecode object of this ABI, which is a representation of all the fields of the smart contract in bytecode, which is readable by the EVM. Finally we say that we are sending the smart contract transaction from the coinbase account and output the ethereum address of the contract on the network. To prevent a race condition from occurring I had to manually stop the provider engine once the contract was deployed.

Using Truffle to Compile and Deploy to Blockchain

To perform the equivalent compilation operations in the truffle IDE one only has to enter the following command from within the project directory in the console:

```
truffle compile
```

Likewise to deploy a smart contract to the development ethereum network one must type the following into the console:

```
truffle migrate
```

To specify the network to deploy the smart contracts to and to run all the migrations from the beginning, one can use the following command in the console:

```
truffle migrate --reset --network rinkeby
```

Android Application

For the android aspect of the project I had the ambition of being able to implement a fully functional android client that would work with my REST API and be able to communicate and send transactions on the blockchain. However, due to the undocumented nature of the web3j framework as of the time of writing this report, it proved to be very challenging to get all the features implemented in time. Primarily, a user interface that reflected the state of the blockchain. Luckily it was not a complete loss as I did manage to create an ethereum wallet from scratch which took an incredible amount of experimentation, trial and error. The reason for this is, as mentioned before, due to the lack of official web3j documentation and the various different types of crypto wallets one can create. I discovered that there are five types of wallets:

1. Standard wallet
2. Full wallet
3. Light wallet
4. Bip39 wallet
5. Bip44 wallet

From investigating the code implementation behind these five wallet types I deciphered that Bip39 and Bip44 wallets appeared to be a more standardised method to create wallets as they also generated a mnemonic which MetaMask also uses to import an account. The main difference I could figure out from analyzing the code implementations behind these two wallets were that the `generateBip39Wallet` is to create a singular wallet from a single account whereas the `generateBip44Wallet` can be used to store multiple wallets from a single

account, which is useful for test nets as it creates a master key pair as well as an separate keypair for each wallet.

Below is the web3j framework code to create a bip39 wallet:

```
94  /**
95   * Generates a BIP-39 compatible Ethereum wallet. The private key for the wallet can be
96   * calculated using following algorithm:
97   *
98   * <pre>
99   *   Key = SHA-256(BIP_39_SEED(mnemonic, password))
100  * </pre>
101  *
102  * @param password Will be used for both wallet encryption and passphrase for BIP-39 seed
103  * @param destinationDirectory The directory containing the wallet
104  * @return A BIP-39 compatible Ethereum wallet
105  * @throws CipherException if the underlying cipher is not available
106  * @throws IOException if the destination cannot be written to
107  */
108  @ public static Bip39Wallet generateBip39Wallet(String password, File destinationDirectory)
109      throws CipherException, IOException {
110      byte[] initialEntropy = new byte[16];
111      secureRandom.nextBytes(initialEntropy);
112
113      String mnemonic = MnemonicUtils.generateMnemonic(initialEntropy);
114      byte[] seed = MnemonicUtils.generateSeed(mnemonic, password);
115      ECKeypair privateKey = ECKeypair.create(sha256(seed));
116
117      String walletFile = generateWalletFile(password, privateKey, destinationDirectory, useFullScript: false);
118
119      return new Bip39Wallet(walletFile, mnemonic);
120  }
```

The method works by taking a string password as a parameter and the directory to where the wallet is saved on the device. Next, it sets an entropy and generates a secure random value from the given entropy then a secure mnemonic of twelve words is generated. After these crucial steps a seed is generated from the mnemonic phrase and password and this seed is used as input when generating the key pair. The private key is used to generate the wallet file along with the password and output directory. Lastly the new wallet is unlocked and returned as an object to the caller.


```

36  /**
37   * Generates a BIP-44 compatible Ethereum wallet on top of BIP-39 generated seed.
38   *
39   * @param password Will be used for both wallet encryption and passphrase for BIP-39 seed
40   * @param destinationDirectory The directory containing the wallet
41   * @param testNet should use the testNet derive path
42   * @return A BIP-39 compatible Ethereum wallet
43   * @throws CipherException if the underlying cipher is not available
44   * @throws IOException if the destination cannot be written to
45   */
46  @ public static Bip39Wallet generateBip44Wallet(
47      String password, File destinationDirectory, boolean testNet)
48      throws CipherException, IOException {
49      byte[] initialEntropy = new byte[16];
50      SecureRandomUtils.secureRandom().nextBytes(initialEntropy);
51
52      String mnemonic = MnemonicUtils.generateMnemonic(initialEntropy);
53      byte[] seed = MnemonicUtils.generateSeed(mnemonic, passphrase: null);
54
55      Bip32ECKeypair masterKeypair = Bip32ECKeypair.generateKeypair(seed);
56      Bip32ECKeypair bip44Keypair = generateBip44Keypair(masterKeypair, testNet);
57
58      String walletFile = generateWalletFile(password, bip44Keypair, destinationDirectory, useFullScript: false);
59
60      return new Bip39Wallet(walletFile, mnemonic);
61  }

```

The above method is almost identical to the prior method except that this method creates two keypairs, one to unlock the master account and the other to unlock the individual wallet.

Issues Encountered Creating an Ethereum Wallet

The largest issues with getting the above methods to work was that in android 10 (API 29) the android development team changed how file permissions work and deprecated a large proportion of the prior methods of interacting with the file system. They restricted access to the file system outside of the scope of the application, which in my use cases would result in the user losing their wallet if they chose to uninstall the application. This would have disastrous results as they would no longer have access to their wallet and they could potentially have had millions in their wallet. The solution to this was to investigate the android documentation and find a solution that works. After lots of experimentation and reading I found that the provided code from the documentation below allowed write and read access outside of the applications file storage through the use of run-time permissions.

	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files	Files meant for your app's use only	From internal storage, <code>getFilesDir()</code> or <code>getCacheDir()</code>	Never needed for internal storage	No, if files are in a directory within internal storage	Yes
		From external storage, <code>getExternalFilesDir()</code> or <code>getExternalCacheDir()</code>	Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	Yes, if files are in a directory within external storage	
Media	Shareable media files (images, audio files, videos)	MediaStore API	<code>READ_EXTERNAL_STORAGE</code> or <code>WRITE_EXTERNAL_STORAGE</code> when accessing other apps' files on Android 10 (API level 29) or higher Permissions are required for all files on Android 9 (API level 28) or lower	Yes, though the other app needs the <code>READ_EXTERNAL_STORAGE</code> permission	No
Documents and other files	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker	No
App preferences	Key-value pairs	Jetpack Preferences library	None	No	Yes
Database	Structured data	Room persistence library	None	No	Yes

Figure 33 Android File Permissions

As the reader can see, above are all the various types of run-time file permissions for android. For my particular use case I could only choose between “Media” file storage or “Documents and other files”. I chose to go with the latter as I couldn't afford the time at this stage of the project to learn another framework.

Below is the code from the android docs to allow particular runtime permissions:

Allowing runtime permissions

```
// Here, thisActivity is the current activity
if (ContextCompat.checkSelfPermission(thisActivity,
    Manifest.permission.READ_CONTACTS)
    != PackageManager.PERMISSION_GRANTED) {

    // Permission is not granted
    // Should we show an explanation?
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
        Manifest.permission.READ_CONTACTS)) {
        // Show an explanation to the user *asynchronously* -- don't block
        // this thread waiting for the user's response! After the user
        // sees the explanation, try again to request the permission.
```

```

} else {
    // No explanation needed, we can request the permission.
    ActivityCompat.requestPermissions(thisActivity,
        arrayOf(Manifest.permission.READ_CONTACTS),
        MY_PERMISSIONS_REQUEST_READ_CONTACTS)

    // MY_PERMISSIONS_REQUEST_READ_CONTACTS is an
    // app-defined int constant. The callback method gets the
    // result of the request.
}
} else {
    // Permission has already been granted
}

```

Android comes with a partial library implementation of the bouncy castle cryptography library however this was a problem in generating keypairs and as i discovered I was not the only one to encounter this problem. It is a common issue with android, luckily a member of the android community provided a solution to this problem below with the following code:

```

88     private void setupBouncyCastle() {
89         final Provider provider = Security.getProvider(BouncyCastleProvider.PROVIDER_NAME);
90         if (provider == null) {
91             // Web3j will set up the provider lazily when it's first used.
92             return;
93         }
94         if (provider.getClass().equals(BouncyCastleProvider.class)) {
95             // BC with same package name, shouldn't happen in real life.
96             return;
97         }
98         // Android registers its own BC provider. As it might be outdated and might not include
99         // all needed ciphers, we substitute it with a known BC bundled in the app.
100        // Android's BC has its package rewritten to "com.android.org.bouncycastle" and because
101        // of that it's possible to have another BC implementation loaded in VM.
102        Security.removeProvider(BouncyCastleProvider.PROVIDER_NAME);
103        Security.insertProviderAt(new BouncyCastleProvider(), position: 1);
104    }
105 }

```

This code above removes the current bouncy castle provider and instantiates a new one which corrected the bug in android. However I encountered another issue specifically a DEX issue. The issue refers to the android application files exceeding the executable bytecode size of the Dalvik Executable (DEX) which is defined as being 65536 bytes in size. The solution to this problem was to increase the DEX limit beyond 65536 bytes. This was solved by adding the following line to the app gradle file:

```

implementation 'androidx.multidex:multidex2.0.1'

```

After a lot of problems with the web3j framework, gradle, DEX and android I finally tried debugging the emulator. However, to use the debugger it only works with a certain android emulator, the google api emulator to be specific, which is not the default emulator that android studio recommends. After some more head scratching and experimenting with configuration files and settings, the Android Debugger (ADB) finally worked and I could now view the file system of my now rooted emulator! The next step was to create the wallet.

I was forced to use a non bip standard way of doing it. The code below creates a wallet on android and the following screenshots of the android logcat validate that indeed the wallet was created.

Creating an EOA Wallet

```
public void createAccount() throws Exception {
    ECKeyPair keyPair = Keys.createEcKeyPair();

    BigInteger publicKey = keyPair.getPublicKey();
    String publicHexKey = Numeric.toHexStringWithPrefix(publicKey);

    BigInteger privateKey = keyPair.getPrivateKey();
    String privateKeyHex = Numeric.toHexStringWithPrefix(privateKey);

    Credentials credentials = Credentials.create(new ECKeyPair(privateKey,
publicKey));
    String address = credentials.getAddress();

    EthGetBalance ethGetBalance = web3j.ethGetBalance(address,
DefaultBlockParameterName.LATEST).sendAsync().get();
    BigInteger balanceWei = ethGetBalance.getBalance();

    Log.d("account", "private key: " + privateKeyHex);
    Log.d("account", "public key: " + publicHexKey);
    Log.d("account", "address: " + address);
    Log.d("account", "balance: " + balanceWei);

    WalletFile walletFile = Wallet.createStandard("password", keyPair);
    Log.d("account", "wallet address: " + walletFile.getAddress());

    ObjectMapper objectMapper = ObjectMapperFactory.getObjectMapper();
    String wallet = objectMapper.writeValueAsString(walletFile);
    JSONObject jsonObject = new JSONObject(wallet);
    String keystore = jsonObject.toString(2);
    Log.d("account", "keystore: " + keystore);
}
```

```
}
```

The output of the above code:

```
D/account: private key: 0x4aeba70a968fada312fc83f838145bf6ebce9b0c89605991d1149d7d87ad3f9f
D/account: public key: 0xc0fb56bd4a18bb50cc59435bd55f2360e0a5defca41aa9b216f65fe45083c311cc106d79feaa592ee95a47f3371bdc072faa2598676224787c79447615f092d
```

```
D/account: address: 0x77f1dfa0ef9791ddbe48d4b11119d9e286e8f1e8
D/account: balance: 0
D/account: wallet address: 77f1dfa0ef9791ddbe48d4b11119d9e286e8f1e8
```

```
{
  "address": "77f1dfa0ef9791ddbe48d4b11119d9e286e8f1e8",
  "id": "3cb81093-053d-4127-832c-95bc2ba34d59",
  "version": 3,
  "crypto": {
    "cipher": "aes-128-ctr",
    "cipherparams": {
      "iv": "b06aa9bd685ce974b565eb4ff00e9fed"
    },
    "ciphertext": "0d5e457ed2e140168486f27c009b7c331f79304f782b38c007aa6d2b56b9b76b",
    "kdf": "scrypt",
    "kdfparams": {
      "dklen": 32,
      "n": 262144,
      "p": 1,
      "r": 8,
      "salt": "6df73263a78725f467bc3dd10bc550138fb4933691d33f9edf6d48ad7b06efb6"
    },
    "mac": "558410d656c384a80509c480b8c262d2848ed4c86c47a9edf953b7545cc21150"
  }
}
```

The encryption scheme used by the wallet is AES-128-CTR. This refers to the keysize being 128 bytes. As the reader can see the address field does not get encrypted because this value is the public ethereum address of the given wallet. The AES-128-CTR is a block cipher and this also uses an initialization vector and a salting value which results in the ciphertext field which is the encrypted representation of the private key.

Node Version Manager

Node Version Manager or NVM is a downloading and installing tool that allows you to download and install Node. Node allows JavaScript code to run directly in the computer process itself instead of in a browser. (Code Academy, -)

Truffle, Blockchain Integrated Development Environment

Truffle is a development environment and testing framework that has been designed for use with smart contracts. Truffle interacts with Ganache and can be used to interact with public Ethereum client providers such as Infura. (Truffle Suite, -)

Ganache, Local Ethereum Blockchain

Several platforms support Ganache and when Ganache is set up it is utilised for testing Solidity contracts on the local Ethereum blockchain. Ganache includes a Blockchain test environment with ten test accounts all credited with 100 Ether, which is the Ethereum blockchain cryptocurrency.

Elastic Beanstalk, Amazon Web Services

The function of Elastic Beanstalk, is to deploy and manage applications in the Amazon Web Services Cloud and reduce the complicated management function while not restricting the user's choice or control. All that is required of the user is to deploy the application and the detail of capacity provisioning, health monitoring, scaling and load balancing are all handled automatically. Elastic Beanstalk builds and supports platform versions and provisions of AWS resources, to run your application. (aws, -)

<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html>

You can interact with Elastic Beanstalk by using the Elastic Beanstalk console, the AWS Command Line Interface (AWS CLI), or **eb**, a high-level CLI designed specifically for Elastic Beanstalk. (aws, -)

MetaMask, Wallet Software

Often referred to as a cryptocurrency wallet but in actual fact it is a browser extension that allows the user to run dApps, without being an Ethereum node on the cryptocurrency network. Metamask allows connection to an Ethereum Node called INFURA in order to run smart contracts. Your Ethereum wallet is managed by MetaMask where your cryptocurrency Ethers reside and allows transactions ie send and/or receive Ethers through your dApp of choice. (Choi, 2015)

Installing the operating system

To set up a development environment, like the one used in this project, it is recommended that the reader install Debian, which can be found at [Once the reader has installed Debian on their device](#) it is necessary to ensure that all the software packages are up to date and that unused packages are removed. This can be done by using the aptitude package manager by executing the following commands in the Linux terminal:

```
sudo apt-get update && sudo apt-get upgrade -y  
sudo apt-get autoremove && sudo apt-get autoclean
```

Installing the development dependencies

The next step in re-creating the development environment is to install the required software packages and their dependencies.

Server-side dependencies

Node Version Manager

NVM is a bash script to manage multiple versions of NodeJS. This is recommended as one can test out a node application against the stable branch and the testing branch, to see if there are any areas where their app breaks and therefore resolve it. To install NVM, please go to [and follow their instructions for installation on Debian Linux](#).

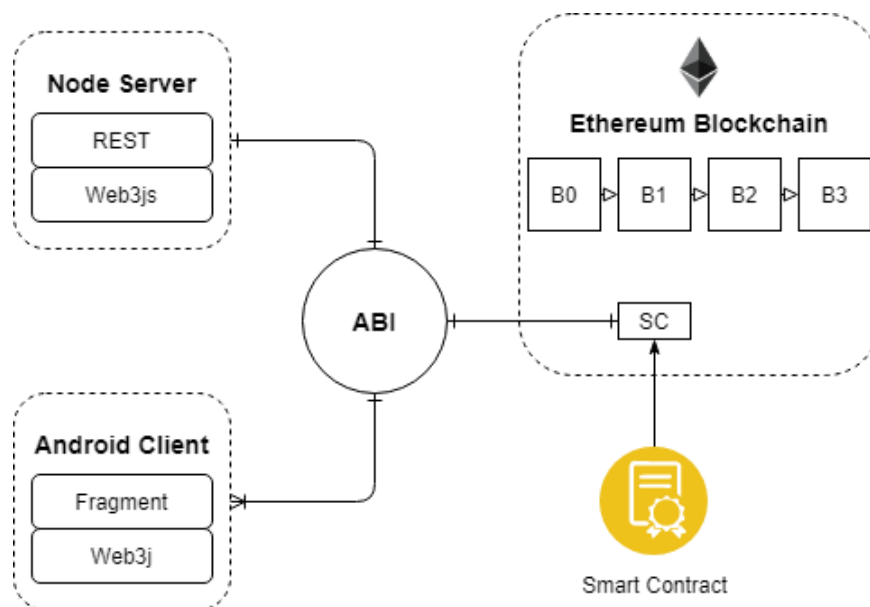


Figure 34 Node Version Manager

Truffle, Blockchain Integrated Development Environment

The next recommendation is to install the truffle integrated development environment for solidity. Solidity is an object-oriented programming language for writing smart contracts. This tool allows the user to write tests, code and migrate contracts to a blockchain, all from the command line. To install truffle please visit [truffle.io](#) and follow their instructions.

Ganache, Local Ethereum Blockchain

Another requirement is to have a local development Ethereum blockchain on your device which the reader can use to run tests, execute commands, and inspect the Ethereum blockchains state. To achieve this please install Ganache from [https://www.trufflesuite.com/ganache](#) or if you prefer the command line version please install the “ganache-cli”, via the node package manager. Another option is to use the included blockchain environment that comes with the truffle IDE.

Elastic Beanstalk, Amazon Web Services

This is optional, but the reader is recommended to create an account on Amazon Web Services and install the elastic beanstalk command line tool (EB CLI) The EB CLI can be used to migrate a local development project and its source code to an Elastic Beanstalk environment that finally gets deployed to the AWS Cloud and has a static IP Address associated to the environment so that it is publicly accessible.

MetaMask, Wallet Software

The last requirement is to have a wallet extension for your browser for testing the server side. MetaMask is highly recommended and can be found on the chrome extensions store at [https://chrome.google.com/webstore/detail/metamask](#)

Client-side Development Environment

For the client side of the application, it is strongly recommended to use Android Studio with Kotlin. Also, be sure to include the web3j dependencies into the project build.gradle file. Web3j. The documentation can be found at [https://web3j.org/](#)

Once the reader has all these dependencies in place, they are all set to experiment with Ethereum blockchain and create decentralized web applications.

Front-End

Android Application Project Structure

Trying to get the android application to work in the system took a lot of code and ultimately couldn't get it finished in time as the codebase needed to keep growing and growing even for basic functionality as many libraries and dependencies were needed to work with the web3j java framework. It was incredibly difficult to create an Ethereum wallet for an android device and because of this reason it was decided to create a web application instead. It was also not as straightforward as expected due to the android development team changing how permissions are managed on android devices as of Android 10 (API 29), this is important for the system as the wallet file must be cryptographically stored on the device. It's also important to state that no official documentation exists for the web3j java framework. Therefore, to understand the web3j API it involved reading the open source code and deciphering what and how the code works and for this project it was deemed to be far too time consuming of a task for such a strict deadline. Because of these reasons it was decided to create a react web application using the JavaScript web3js framework instead. However, the creation of an Ethereum wallet was successfully achieved using the web3j java framework but not without much invested time and effort.

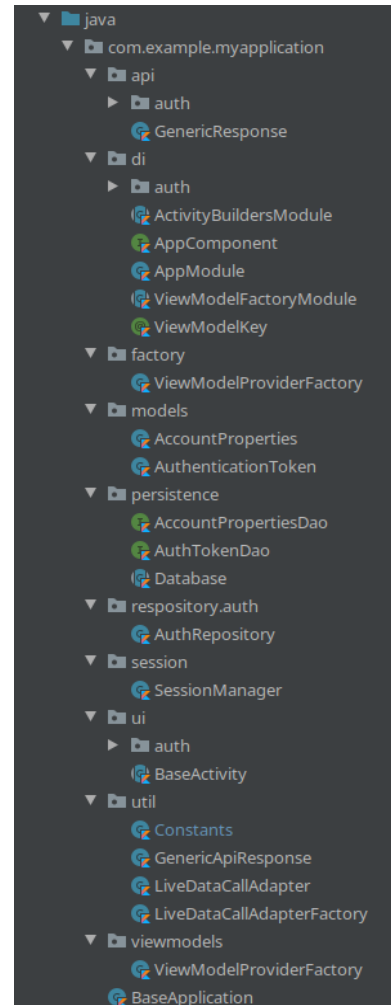


Figure 35 Android Application Project Structure

Android Ethereum Encrypted Wallet File

```
{
  "address": "77f1dfa0ef9791ddbe48d4b1119d9e286e8f1e8",
  "id": "3cb81093-053d-4127-832c-95bc2ba34d59",
  "version": 3,
  "crypto": {
    "cipher": "aes-128-ctr",
    "cipherparams": {
      "iv": "b06aa9bd685ce974b565eb4ff00e9fed"
    },
    "ciphertext": "0d5e457ed2e140168486f27c009b7c331f79304f782b38c007aa6d2b56b9b76b",
    "kdf": "scrypt",
    "kdfparams": {
      "dklen": 32,
      "n": 262144,
      "p": 1,
      "r": 8,
      "salt": "6df73263a78725f467bc3dd10bc550138fb4933691d33f9edf6d48ad7b06efb6"
    },
    "mac": "558410d656c384a80509c480b8c262d2848ed4c86c47a9edf953b7545cc21150"
  }
}
```

React Web Application

#	Name	Price	Owner	
1	4GB Mobile Data	12 Eth	0x1d69aFD085C4B693069602fC20d8376Bb0c7124f	Buy
2	8GB Mobile Data	20 Eth	0x1d69aFD085C4B693069602fC20d8376Bb0c7124f	Buy

Figure 36 React Web Application

Middle-Tier

For the middle-tier of the application I used a variation of the Model-View-Controller design pattern. The REST server has no view so it was simply omitted. The project structure for the REST server can be seen on the right. The controller code contained a lot of the business logic so if I were to redesign this type of web3 project in the future I would try to incorporate Bob Martin's clean architecture patterns. However, I did run out of time in the development phase and for this reason I'm content with keeping the code structure simple as shown here. Next I will present the reader with the code the controller uses to communicate with the private ethereum blockchain node and how this service is provided as a web service behind a REST endpoint. First we must get an instance of web3, the networkId that can be used to get the right smart contract from the network along with its application binary interface all methods on the blockchain are asynchronous so the async await style of promise handling was used in Node.js.

Below is the following code snippet of a controller class:

```
async function getExchangeInstance() {  
  try {
```

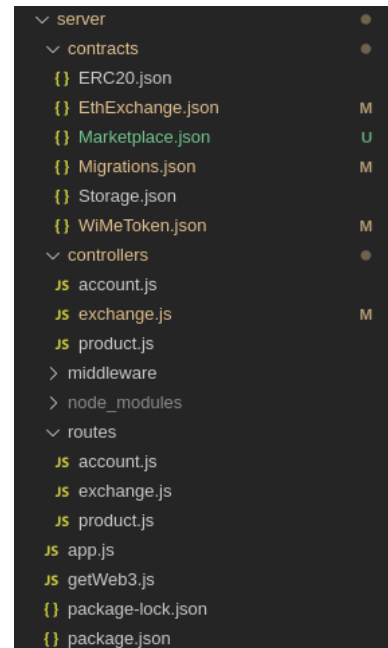


Figure 37 REST Server project structure

```

    const web3 = await getWeb3();
    const networkId = await web3.eth.net.getId();
    const exchange = await new web3.eth.Contract(
      EthExchange.abi,
      EthExchange.networks[networkId] &&
      EthExchange.networks[networkId].address,
    );
    return exchange;
  } catch (error) {
    console.log(error.message);
  }
}

```

Once we have an instance of the smart contract we next need to get accounts on the network, this is done by getting a web3 instance and calling the `getAccounts()` method on this instance as follows:

```

async function getAccounts() {
  try {
    const web3 = await getWeb3();
    const accounts = await web3.eth.getAccounts();
    return accounts;
  } catch (error) {
    console.log(error.message);
  }
}

```

Now that we have an instance of the contract and an account to go along with it the next step is to call the specific method on the smart contract we are interested in, which is `buyTokens()`. First we must parse the body of the request to get the amount of tokens the user wants. We also get the `buyTokens()` method by accessing the `methods` key on the smart contracts ABI. The next code snippet shows how that was done in the project:

```

exports.buyTokens = async (req, res, next) => {
  const amount = req.body.amount;
  try {
    const app = await getExchangeInstance();
    const accounts = await getAccounts();
    const tokens = await app.methods.buyTokens().send({ from: accounts[0],
value: amount });
    res.status(201).json({
      response: tokens
    });
  } catch (error) {
    console.log(error.message);
  }
}

```

```

    });
  } catch (error) {
    res.status(404).json({
      response: error.message
    });
  }
};

```

Lastly, this data is exposed through the REST API by the routes in the following way:

```

router.post("/exchange", exchangeController.buyTokens);

```

Back-End

Ethereum Private Node

Genesis block

```

1  {
2    "config": {
3      "chainId": 4224,
4      "homesteadBlock": 0,
5      "eip150Block": 0,
6      "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
7      "eip155Block": 0,
8      "eip158Block": 0,
9      "byzantiumBlock": 0,
10     "constantinopleBlock": 0,
11     "petersburgBlock": 0,
12     "istanbulBlock": 0,
13     "ethash": {}
14   },
15   "nonce": "0x0",
16   "timestamp": "0x5e847898",
17   "extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",
18   "gasLimit": "0x47b760",
19   "difficulty": "0x80000",
20   "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
21   "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",
22   "alloc": {},
23   "number": "0x0",
24   "gasUsed": "0x0",
25   "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
26 }

```

Timestamp

Is used by eth to adjust level of difficulty in consensus algorithm if the difference between two blocks timestamps is too small the difficulty will be increased and therefore if the time difference is too high the difficulty will be reduced the timestamps value is also used to ensure that blocks are in the correct order.

Gas Limit

Defines the max value of gas that transactions in a block can spend. This limits the processing power of each block and therefore limits the size of each block on the network.

Difficulty

Defines used to validate a block and refers to the number of times a miner will have to run the consensus algorithm in order to solve the puzzle and successfully mine a transaction. Higher values equate to longer computation times so therefore on a private node it made sense to use a low value for faster puzzle solving and therefore mining.

Coinbase

The address that gets the rewards for mining the current block

Number

This is the block number and is zero as this is the first block i.e. the genesis block

Gas Used

Is the sum of all the gas used by all the transactions on this block

Parent Hash

Is the hash of the previous block and because this is the genesis block it has a value of zero. To create geth blockchain database and initialise it with the parameters of the genesis block run the command

```
Geth -datadir . init Wime.json
```

Then to create a single new account on private node run the command

```
Geth -datadir . Account new
```

The user is prompted to enter a secure password and the public key refers to the ethereum address of this account and the keystore is the user's encrypted private key and wallet.

```

Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key: 0x803D39F7c2D4eb41c1789ea8Aab689ef1BFd4b4b
Path of the secret key file: keystore/UTC--2020-04-01T12-04-20.249202502Z--803d39f7c2d4eb41c1789ea8aab689ef1bfd4b4b

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!

```

Starting a private ethereum node

The simplest way to start a private node is to write a script. We want our node to mine right away so the `-mine` option is used and we can specify how many threads it can use on our machine we also don't want our node to connect to any peer so its important to disable the discovery option with the following command `-nodiscover` but we want to enable the remote procedure call or rpc method in order to connect to the node with metamask later to do this add the following command `-rpc` also we want to be able to connect to this rpc endpoint with any domain so the following command `-rpccorsdomain "*" --net "any"` is used then to enable modules required the command `-rpcapi eth,web3,personal,net` then to unlock an account so that mining rewards can be added to the account use the command `-unlock 0 -password ./password.sec` lastly it's very important that other tools can communicate with our private node such as the geth console so the `-ipcpath "~/ethereum/geth.ipc"` command allows for inter process communication on the given path and these tools to automatically connect to the endpoint of our ethereum node and finally to get around the "Fatal: Account unlock with HTTP access is forbidden!" error append `-allow-insecure-unlock` to the `startnode.sh` script

Next the DAG or directed acyclic graph is generated and this is a data structure needed by the ethash mining algorithm to perform operations on the ethereum node. The DAG is generated every 30,000 blocks and each generation is referred to an epoch in this case.

```

INFO [04-01|13:33:04.933] Generating DAG in progress      epoch=0 percentage=23 elapsed=1m1.428s
INFO [04-01|13:33:07.558] Generating DAG in progress      epoch=0 percentage=24 elapsed=1m4.053s
INFO [04-01|13:33:09.882] Generating DAG in progress      epoch=0 percentage=25 elapsed=1m6.378s
INFO [04-01|13:33:12.118] Generating DAG in progress      epoch=0 percentage=26 elapsed=1m8.613s
INFO [04-01|13:33:14.612] Generating DAG in progress      epoch=0 percentage=27 elapsed=1m11.108s

```

Connecting to the private ethereum node

To connect to the running private ethereum node one can use a javascript console. To achieve this execute the command `"geth attach"` from the terminal. It's important to note that the ethereum node must be already running and mining to be able to connect to it. As the ethereum node has been running for a long period of time at this point it has successfully mined some ether which we can now transfer from the coinbase account to the 3 accounts created earlier.

To do this execute the following command which transfers 100 ether from the coinbase account to the given account

```
eth.sendTransaction({from: eth.coinbase, to: eth.accounts[1], value:
                      web3.toWei(100, "ether")})
```

This command then returns the hash of the transaction

```
> eth.sendTransaction({from: eth.coinbase, to: eth.accounts[1], value: web3.toWei(100, "ether")})
"0x04b9e987e7f46dca924d765f4368f7be18d8a19cb949f12d054d5e100444937f"
```

Then this transaction can be seen on the private ethereum node and that it will be mined in the preceding block

```
INFO [04-01|14:33:10.889] Setting new local account          address=0xCFaDe4E4E724463B6C17
d24Ec1B91E20AA6e1B1E
INFO [04-01|14:33:10.889] Submitted transaction              fullhash=0x04b9e987e7f46dca924
d765f4368f7be18d8a19cb949f12d054d5e100444937f recipient=0xe73ABA3880b35Aa335e4adea0dE727dE077a8e4
f
```

Deploying smart contract to private Ethereum node

```
truffle migrate -compile-all -reset -network development
```

```
File Edit View Bookmarks Settings Help
> block timestamp: 1586580586
> account: 0x90C2E7a447736c9Eef6F0cfc4a6a3eED495b09d5
> balance: 99.89733704
> gas used: 161274
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00322548 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.02272496 ETH

Summary
=====
> Total deployments: 4
> Final cost: 0.02530868 ETH

eamonn@debian:~/Documents/fyp2/ethereum$
```

Postman: Postman fyp2: node server: node client: node ethereum: bash

Source Code can be found at: <https://github.com/mondo192/wime>

Chapter 5 - Testing and Evaluation

Introduction

Testing is a vital component of software development and is a key factor in evaluating the given quality and performance of a software product. Testing is even more critical when it comes to ethereum development as the reader will discover in chapter. Solidity unit tests will be the primary topic of this chapter as well as manual testing of REST endpoints and the client application.

System Testing

To ensure that the ones test files are treated individually so they don't share state with one and other, Truffle utilises features such as advanced snapshotting. Truffle re-deploys all of one's migrations at the very beginning of each and every test file, when running against other Ethereum clients such as go-ethereum, in order to ensure one has a fresh set of contracts to test against.

Testing the REST api server

Manual tests were carried out on the REST api server using a tool called Postman. Postman is a tool that can be used to check REST and SOAP responses and below is a screen showing a valid JSON response from the smart contract which is behind the route

```
http://localhost:3000/api/exchange
```


Definition by ISTQB

<http://softwaretestingfundamentals.com/unit-testing/>

Solidity unit Tests

These contracts exist along with Javascript tests as .sol files. The running of Truffle test includes Solidity as a separate test per test contract. The benefits of Javascripts tests, such as, clean room environment per test suite, the ability to import any contract dependency, direct access to your deployed contracts, and the importation of any contract dependency are not diminished or lost. Other benefits built into Truffle's Solidity testing framework includes:
Making tests as minimal as possible and giving total control over the contracts you write
Not being indebted to assertion libraries
The ability to run Solidity tests against Ethereum clients. (Truffle Suite, -)

<https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-solidity>

The truffle integrated development environment has a utility to create unit tests and system tests. They encourage a developer to run a test before migrating a contract to the blockchain as doing a migration costs GAS, whereas running tests doesn't need to be performed on the blockchain and therefore costs nothing. Tests for solidity can be written in NodeJS.

```
File Edit View Bookmarks Settings Help
Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Contract: WiMe Ethereum Exchange
  Token
    ✓ token has a valid name
  Exchange
    ✓ contract has a name
    ✓ contract has the exchanged tokens
  Purchase tokens
    ✓ exchanges tokens with the investor successfully
    ✓ updates the total supply of the exchange after investing
  Sell tokens
    ✓ investor successfully sold tokens on the ethereum exchange

6 passing (441ms)
eamonn@debian:~/Documents/fyp2/ethereum/contracts$ █

contracts : bash  ethereum : node
```

```
const WiMeToken = artifacts.require("WiMeToken");
const EthExchange = artifacts.require("EthExchange");

function tokens(x) {
  return web3.utils.toWei(x, "ether");
}
```

[illegible]

```

    it("updates the total supply of the exchange after investing", async()
=> {
    let balance = await token.balanceOf(exchange.address);
    assert.equal(balance.toString(), tokens("30"));
    balance = await web3.eth.getBalance(exchange.address);
    assert.equal(balance.toString(), web3.utils.toWei("0.7", "ether"));
  });
});

describe("Sell tokens", async () => {
  let result;
  before(async () => {
    await token.approve(exchange.address, tokens("70"), { from:
accounts[1] });
    result = await exchange.sellTokens(tokens("70"), { from: accounts[1]
});
  });

  it("investor successfully sold tokens on the ethereum exchange", async
() => {
    const balance = await token.balanceOf(accounts[1]);
    assert.equal(balance.toString(), tokens("0"));
  });
});
})

```

System Evaluation

If the outcome of the project matches and fulfils the requirements stated earlier then the system is good. These ten principles, which are the standard of good design, were kept firmly in focus throughout WiMe. The ten principles are listed as follows

Nielsen Heuristics can be used as a guide to evaluate a product under the following ten conditions:

1: Visibility of system status

The system needs to be designed in such a way that the user knows exactly what is happening in real time through an appropriate form of feedback. The reason is that the more and the clearer the information transmitted to the user, the more this translates into better decision making.

2: Match between system and the real world

Feed-back and communication between the system and the user must be in a language and utilizing concepts that the user can readily understand through everyday familiarity. The communication should be in a conventional form and follow the natural real-world logical order. The reason is that we find comfort in familiarity ie. What is familiar to the user, not the designer.

3: User control and freedom

Sometimes the wrong function is chosen by mistake and users need a quick “way out” of the unwanted function and back to the initial state. The emergency exit button or action must not incur an unwieldy routine but needs to be an instant “fix”, ie. Undo and redo.

4: Consistency and standards

Words conveying required actions by users need to be unambiguous. Therefore, standardisation is required to make sure that important action wording is not “brand” dependent.

5: Error prevention

Errors should be prevented by good design. Problem prevention through design is much more valuable and efficient. Error prone conditions should be identified and eliminated or when identified the user is presented with a confirmation option before the user commits to the action.

6: Recognition rather than recall

The user’s memory should not be relied upon and the reliability of a user’s memory should not be part of any system design. Therefore, the need for the user to memorize actions or options should be completely minimised. Visible clear user instruction is the way to go to delete recall in favour of recognition.

7: Flexibility and efficiency of use

There are two types of user to be catered for in a design, experienced user and inexperienced user. Unseen accelerators known only to experienced users are often utilised to speed up system user interaction. This may allow experienced users to tailor make frequent actions.

8: Aesthetic and minimalist design

Irrelevant information or rarely needed instruction should not form part of the communication dialogue. The more instruction included decreases the visibility and clarity of the most important information. Therefore, only instruction that covers the necessary actions and options should be catered for.

9: Help users recognize, diagnose, and recover from errors

Plain language, no coded language must be utilised especially in error messages. A problem should be precisely diagnosed, and the solution constructively indicated.

10: Help and documentation

It is usually necessary to supply documentation with most systems even though the system would be thought of as a better system if help or documentation was not required. However, supplied documentation should be task focussed and easy to search for the relevant information and neither long winded or overly concise.

(nn/g Nielsen Norman Group, -)

Chapter 6 - Conclusions and Future Work

Introduction

This final chapter provides an overview of the WiMe system and the dissertation as a whole and provides a summary of each of the chapters in this report. The primary objective of this project was to create an ethereum blockchain marketplace for users to buy and sell internet time. To represent this “internet time” value, it was necessary to create a token called a WiMe Token that can be exchanged for ether on the testnet. As highlighted throughout the report this came very close to being fully implemented if not for the issues encountered with the web3j framework on android.

Chapters Review

To gain a greater sense of depth and the lifecycle of the project a concise summary of each chapter will be presented to the user in the following sections.

Literature Review

In this chapter the reader will discover information about the key areas of literature that helped the author gain a concrete understanding and appreciation of the ethereum blockchain. Also, similar technologies and products that are already existing that are close to the use-cases of the project can be found in this chapter. Furthermore, extra research can be found in this chapter such as cloud computing and GDPR issues. With the background knowledge gained from this chapter, one would have a foundational grasp of the ethereum blockchain and some of the areas of computing it touches and therefore could create a design for the project.

Experiment Design

Measure twice, cut once. In order to build a software solution on a largely experimental and emerging technology it required careful planning and consideration of proper management for a large project like this one. Implementing the correct software methodology for the relevant area can aid in the smoothness and efficiency of a project's life cycle. The reader will find the wireframes and prototypes as well as class and sequence diagrams and a critical analysis of the given technologies required to implement an ethereum blockchain application

Experiment Development

With the research, planning and design in place it was time to build the architecture, write code, set up the environment and ensure all software packages were up to date. In this section the reader will be provided with the development story for the project and solutions to issues they may encounter also. Although, there was not a fully functional end user, commercially viable product fully delivered in the time allocated, the application provides a proof of concept. It demonstrates how blockchain could be utilized and used to allow end users to essentially

become their own internet service provider,s by exchanging “Internet time” through the exchange of tokens.

Testing and Evaluation

In this chapter the reader will find information on the testing methodologies that were utilized during the development process and how one tests an ethereum blockchain and the best practices for an ethereum developer and why a test driven development lifecycle is the only logical approach for this domain of software development.

Conclusions

Now it’s time to reflect on and try to assess what has been the focus of my endeavours for the past five months. Although the author underestimated the scale and complexity of the project hopefully this report conveys the knowledge of depth and the fact that the author has been intellectually stretched by the whole experience, which is good.

The main intention in pursuing this project was to design and implement an Ethereum blockchain distributed application for its end users to buy and sell internet time with one another. Behind this too, there is the passion to help the promote the case for re-tweaking the ethos of the world wide web back to its roots. The lessons have taught the author so much about not giving up even when problems are complex and their solution just out of reach through time constraints and other uncontrollable factors.

Future Work

With the research and development of the WiMe system successfully completed, it naturally makes sense to critique the project and the project domain. If I were to continue to work on the WiMe system I would invest more time into implementing the web3 Java framework version of the application. Also, perhaps write documentation like that of the web3 JavaScript framework for others to be able to understand the Java framework for web3 more easily. The reason for this is that there is no documentation that is understandable currently available for Android. Also, if there were more time allowed, I would have liked to improve the user interface and user experience to better reflect the state of the Ethereum Blockchain. I can see myself attempting a similar project in the future that utilizes the Ethereum Blockchain but integrating even more sophisticated technologies such as machine learning in peer to peer networks.

Would blockchain be a valid area to research and do further work on in the future? In my opinion, yes, it most definitely is. Ethereum development is still an emerging and maturing technology field, with many encompassing areas as mentioned throughout the dissertation. Although its origins are that of the finance and banking sector it is quickly becoming the new way to create applications for the web and smart devices. It is difficult to estimate when it will be fully adopted by organizations and companies as one can see that they’re currently only a handful of jobs for a blockchain developer currently.

Ethereum 2.0 is under development and they have released phase 0 and phase 1 specifications and some developers already have a phase 0 public Ethereum 2.0 testnet running
<https://prylabs.net/>

Bibliography

(TMFULtraLong), S. W., 2018. *The Basics of Blockchain Technology Explained in Plain English*. [Online]
Available at: <http://www.fool.com/investing/2018/01/10/the-basics-of-blockchain-technology-explained-in-p.aspx>

[Accessed 6th March 2020].

abhiandroid, -. *Retrofit Tutorial with Example in Android Studio*. [Online]

Available at: <http://abhiandroid.com/programming/retrofit>

[Accessed 16th March 2020].

AirFox, 2019. *AirToken: a token to power more inclusive financial services*. [Online]

Available at: <http://airfox.com/images/uploads/airtoken-white-paper-updated-april-2019.pdf>

[Accessed 7th March 2020].

Ashley, 2019. *AirToken: a token to power more inclusive financial services*, s.l.: s.n.

aws, -. *AWS Code Pipeline*. [Online]

Available at: <http://aws.amazon.com/codepipeline/>

[Accessed 13th March 2020].

aws, -. *AWS Elastic Beanstalk*. [Online]

Available at: <http://aws.amazon.com/elasticbeanstalk/>

[Accessed 12th March 2020].

aws, -. *EB CLI Command Reference*. [Online]

Available at: <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-cmd-commands.html>

[Accessed 14th March 2020].

aws, -. *Using the Elasticbeanstalk Command Line Interface (EB CLI)*. [Online]

Available at: <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html>

[Accessed 14th March 2020].

aws, -. *What is AWS Elasticbeanstalk*. [Online]

Available at: <http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/welcome.html>

[Accessed 12th March 2020].

aws, -. *What is Docker*. [Online]

Available at: <http://aws.amazon.com/docker>

[Accessed 13th March 2020].

Baliga, 2017. *Understanding Consensus Models*. [Online]

Available at: -

[Accessed 6th March 2019].

BBVA, 2018. *The Internet, Politics and the Internet Debate on Politics*. [Online]

Available at: <http://www.bbvaopenmind.com/en/articles/the-internet-politics-and-the-politics-of-the-internet-debate/>

[Accessed Thursday Dec 2019].

Breslow, S., 2019. *Inventor of World Wide Web speaks out about Online Misinformation..* [Online]
Available at: <http://prospect.org/environment/inventor-world-wide-web-speaks-online-misinformation>
[Accessed 6 December 2019].

Chai Assertion Library, n.d. *Chai Assertion Library*. [Online]
Available at: <https://www.chaijs.com/>

Chinchilla, C., 2019. *Ethereum Development Tutorial*, s.l.: Ethereum.

Choi, S., 2015. *What is MetaMask? Really, What-is-it-really-what-is-it.* [Online]
Available at: <http://medium.com/@seanchoi/what-is-matamask-really-what-is-it-7bc1bf48c75>
[Accessed 14th March 2020].

Clement, J., 2019. *Number of worldwide internet hosts in the domain name system (DNS) from 1993 to 2019.* [Online]
Available at: <https://www.statista.com/statistics/264473/number-of-internet-hosts-in-the-domain-name-system/>
[Accessed 15 04 2020].

Code Academy, -. *What is Node? Java Script and Node.JS.* [Online]
Available at: <https://www.codecademy.com/articles/what-is-node>
[Accessed 17th March 2020].

Congress, M. W., 2018. *Blockchain goes mainstream:.* [Online]
Available at: <https://www.dentwireless.com/dent-android-press>
[Accessed 4 2 2020].

Copeland, T., 2019. *What is Infura.* [Online]
Available at: <http://decrypt.co/resources/what-is-infura>
[Accessed 13th March 2020].

Curran, B., 2018. *What Proof of Authority Consensus? Staking Identity on the Blockchain.* [Online]
Available at: <http://blockonomi.com/proof-of-authority>
[Accessed 5 March 2020].

Curran, B., 2020. *What is IPFS?.* [Online]
Available at: <http://blockonomi.com/interplanetary-file-system>
[Accessed 14th March 2020].

Debian, -. *About Debian.* [Online]
Available at: <http://www.debian.org/intro/about>.
[Accessed 12 March 2020].

Debian, -. *Getting Debian.* [Online]
Available at: <https://www.debian.org/distrib/>.
[Accessed 17th March 2020].

Diamond, A., 2018. *China's Surveillance State Should Scare Everyone..* [Online]
Available at: <http://www.theatlantic.com/international/archive/2018/02/china-surveillance/552203/>
[Accessed 6th december 2019].

Fabian Vogelsteller, V. B., 2015. *ERC-20 Token Standard*. [Online]
Available at: <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md>
[Accessed 5 4 2020].

Goldman, E., -. *Search Engine Bias and the Demise of Search Engine Utopianism..* [Online]
Available at: http://link.springer.com/chapter/10.1007/978-3-540-75829-7_8
[Accessed 25th November 2019].

How stuff works tech, n.d. *What's Ubuntu*. [Online].

How Stuff Works tech, -. *What's Ubuntu and how is it different from Linux*. [Online]
Available at: <http://computer.howstuffworks.com/ubuntu2.htm>
[Accessed 10th March 2020].

Hussey, M., 2019. *What are CryptoKitties?*. [Online]
Available at: <http://decrypt.co/resources/cryptokitties>
[Accessed 14th March 2020].

Hussey, M., 2019. *What is MetaMask?*. [Online]
Available at: <http://decrypt.co/resources/metamask>
[Accessed 14th March 2020].

M, 2018. *Introduction of Ethereum Request for Comment\ (ERC20/ERC721*. [Online]
Available at: <http://>
[Accessed 16th March 2020].

M, 2018. *Introduction of Ethereum Request for Comment\ (ERC20/ERC721*. [Online]
Available at: <https://medium.com/star-bit/introduction-of-ethereum-request-for-comment-%E4%BB%A5%E5%A4%AA%E5%9D%8A%E7%B3%BB%E5%88%97%E6%A8%99%E6%BA%96%E4%BB%8B%E7%B4%B9-erc20-erc721-dbbae53d701f>
[Accessed 16th March 2020].

M, 2018. *Setting up the Go Ethereum (geth) Environment in Ubuntu Linux*. [Online]
Available at: <http://medium.com/@priyalwalpita/setting-up-the-go-ethereum-geth-environment-in-ubuntu-linux-67c09706b42>
[Accessed 16th March 2020].

Marsan, C. D., 2009. *The Evolution of the Internet*. [Online]
Available at: https://www.pcworld.com/article/159471/evolution_internet.html
[Accessed 5 03 2020].

Miniwatts Marketing Group, 2020. *Internet Usage Stats*. [Online]
Available at: <https://internetworldstats.com/stats.htm>
[Accessed 4 2 2020].

nn/g Nielsen Norman Group, -. *10 Usability Heuristics for User Interface Design*. [Online]
Available at: <https://www.nngroup.com/articles/ten-usability-heuristics/>
[Accessed 18th March 2020].

Nukala, M., 2015. *How will the web Monetise in 2020*. [Online]

Available at: <https://techcrunch.com/2012/07/15/how-will-the-web-monetize-in-2020/>
[Accessed 4 03 2020].

Opensource.com, -. *What is Docker*. [Online]

Available at: <http://opensource.com/resources/what-docker>
[Accessed 13th March 2020].

Oracle Technology Network, -. *WS-Security Authentication*. [Online]

Available at: <https://www.oracle.com/technetwork/topics/ws-audit-089711.html>
[Accessed 18th March 2020].

Perrin, A., 2019. *Digital gap between rural and nonrural America persists*. [Online]

Available at: <https://www.pewresearch.org/fact-tank/2019/05/31/digital-gap-between-rural-and-nonrural-america-persists/>
[Accessed 3 2 2020].

Quora, -. *What is Debian*. [Online]

Available at: <http://www.quora.com/what-is-debian-and-how-is-it-used?>
[Accessed 12th March 2020].

Rashid.F.Y., 2018. *Schnier on Security*. [Online]

Available at: <http://www.schnier>
[Accessed 4th. December 2019].

Schmidt, C., 2016. *What is Android? Here is a complete guid for beginners*. [Online]

Available at: <http://www.androidpit.com/what-is-android>
[Accessed 12TH. March 2020].

Schnier, b., 2013. *The Internet is a Surveillance State*. [Online]

Available at: [http://homepages.uc.edu/totmam/Implication_IT/pdf/ch2/the internet is a surveillance state.pdf](http://homepages.uc.edu/totmam/Implication_IT/pdf/ch2/the%20internet%20is%20a%20surveillance%20state.pdf)
[Accessed 5 December 2019].

Software Testing fundamentals, -. *Unit Testing*. [Online]

Available at: <http://softwaretestingfundamentals.com/unit-testing/>
[Accessed 18th March 2020].

Sourceforge, 2017. *Truffle*. [Online]

Available at: <http://sourceforge.net/projects/truffle>
[Accessed 14th March 2020].

techcrunch, 2015. *How will the web monetise in 2020*. [Online]

Available at: <https://techcrunch.com/2012/07/15/how-will-the-web-monetize-in-2020/>
[Accessed 4 03 2020].

Tekisalp, E., 2018. *Understanding Web 3 - A User Controlled Internet*. [Online]

Available at: <https://blog.coinbase.com/understanding-web-3-a-user-controlled-internet-a39c21cf83f3>
[Accessed 17th March 2020].

Tekisalp, E., 2018. *Understanding Web3-A User Controlled Internet..* [Online]
Available at: <http://blog.coinbase.com/understanding-web-3-a-user-controlled-internet-a39c21cf83f3>.
[Accessed 3rd March 2020].

Tolsma, A., 2018. *GDPR and the Impact on Cloud Computing.* [Online]
Available at: www2.deloitte.com/nl/nl/pages/risk/articles/cyber-security-privacy-gdpr-update-the-impact-on-cloud-computing.html
[Accessed 8 12 2019].

Truffle Suite, -. *Sweet Tools for Smart Contracts.* [Online]
Available at: <http://www.trufflesweet.com/>
[Accessed 14th March 2020].

Truffle Suite, -. *Writing Tests in Solidity.* [Online]
Available at: <http://www.truffle.com/docs/truffle/testing/writing-tests-in-solidity>
[Accessed 6th March 2020].

Truffle Suite, -. *Writing Tests in Solidity.* [Online]
Available at: <https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-solidity>
[Accessed 18th March 2020].

Voshmgir, S., 2019. Token Economy. In: P. Prophet, ed. *How Blockchains and Smart Contracts Revolutionise the Economy.* s.l.:BlockchainHub Berlin, p. 310.

Voshmgir, S., 2019. *Token Economy-How Blockchains and Smaet Contracts Revolutionise the Economy.* First ed. Berlin: BlockchainHub Berlin.

Web3j, -. *Web3j.* [Online]
[Accessed March March 2020].

Williams, S., 2018. *The Basics of Blockchain Technology, Explained in Plain English.* [Online]
Available at: <https://www.fool.com/investing/2018/01/10/the-basics-of-blockchain-technology-explained-in-p.aspx>
[Accessed 1 4 2020].

World Wide Web Foundation, -. *A Contract for the Web.* [Online]
Available at: <https://9nrane41lq4966uwmljcfggv-wpengine.netdna-ssl.com/wp-content/uploads/Contract-for-the-Web-2.pdf>
[Accessed 6 12 2019].

Zago, M., 2018. *Why the Web 3.0 Matters and You Should Know About It.* [Online]
Available at: <https://medium.com/@matteozago/why-the-web-3-0-matters-and-you-should-know-about-it-a5851d63c949>
[Accessed 17th March 2020].