

# Blockchain Development: Exploring Blockchain Technologies Through the Development of a Decentralised Crowdfunding Application

Patrick Carmody

N00191685

Supervisor: Sue Reardon

Second Reader: Mohammed Cherbatji

Year 4 2022-223

DL836 BSc (Hons) in Creative Computing

## Abstract

The aim of this project is to explore the integration of Ethereum blockchain technology into a MERN stack application, and to examine the available options for developers. The cryptocurrency market capitalization reached a peak of almost \$3 trillion USD in 2021 (Best, 2023). With this amount of assets, this project aims to demonstrate the applications of cryptocurrency beyond peer-to-peer transfers.

This application is a platform for both users and organisations to accept cryptocurrency payments through crowdfunding campaigns. It enables donors to connect their browser based Ethereum wallet and easily contribute cryptocurrency to charitable causes.

Outlined in this report is the design and implementation of the application, as well as examining existing applications and literature. Smart contract development is discussed and resulted in the development and deployment of a crowdfunding smart contract, along with a full-stack web application that interacts with it. Usability testing was conducted to assess ease-of-use and learnability of the application, which received positive feedback.

Further work suggested in this thesis discusses business-focused changes to the application requiring deeper levels of research and programming.

## Acknowledgements

I would first and foremost like to express my deepest appreciation to my supervisor Sue Reardon for the guidance and support throughout the entire process of this project. Her valuable insights helped me especially with the design and user experience aspects of developing the application.

I would also like to thank Mohammed Cherbatji who taught me MERN stack development during the year. My education from this module provided me with the foundation of knowledge that enabled me to undertake this project and inspired me to aim to become a full-stack web developer.

I am also deeply grateful to the staff at IADT who facilitated my transfer into Creative Computing and supported me throughout the course.

Finally, I would like to thank my family for their continued support and encouragement. I must express my gratitude to my mother, Bairbre, for her unwavering support throughout my academic journey, and for her invaluable constructive criticism and writing advice while writing my thesis.

**The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.**

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

**WARNING:** Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not leave copies of your own files on a hard disk where they can be accessed by others. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

**The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below**

*Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.*

**DECLARATION:**

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student : Patrick Carmody

Signed *Patrick Carmody*

---

Failure to complete and submit this form may lead to an investigation into your work.

## Contents

1	Table of figures .....	8
2	Introduction .....	10
3	Research.....	11
3.1	Introduction .....	11
3.2	Blockchain Technology.....	11
3.3	Ethereum Virtual Machine.....	12
3.4	Decentralised Applications .....	14
4	Requirements.....	17
4.1	Introduction .....	17
4.2	Requirements gathering .....	17
4.2.1	Similar applications .....	17
4.3	Requirements modelling.....	20
4.3.1	Personas.....	20
4.3.2	Functional requirements.....	22
4.3.3	Non-functional requirements .....	22
4.3.4	Use Case Diagrams.....	22
4.4	Feasibility .....	23
4.5	Conclusion.....	24
5	Design.....	25
5.1	Introduction .....	25
5.2	Program Design.....	25
5.2.1	Technologies .....	25
5.2.2	Application architecture .....	27
5.2.3	Database design.....	27
5.3	User interface design .....	28
5.3.1	Wireframe .....	28
5.3.2	User Flow Diagram.....	30
5.3.3	Style guide.....	31
5.4	Conclusion.....	32
6	Implementation .....	33
6.1	Introduction .....	33
6.2	Scrum Methodology.....	34
6.3	Development environment.....	35
6.4	Sprint 1.....	35
6.4.1	Goal .....	35

6.4.2	Storage Smart Contract.....	35
6.4.3	UI Prototype.....	36
6.5	Sprint 2.....	37
6.5.1	Goal .....	37
6.5.2	Development Blockchain .....	37
6.5.3	React Application for Storage Contract .....	38
6.6	Sprint 3.....	40
6.6.1	Goal .....	40
6.6.2	'Bank' Smart Contract .....	40
6.6.3	React Application for Bank Contract.....	41
6.7	Sprint 4.....	42
6.7.1	Goal .....	42
6.7.2	Backend Development.....	43
6.7.3	Website layout .....	46
6.8	Sprint 5.....	47
6.8.1	Goal .....	47
6.8.2	UI Components .....	47
6.8.3	Wallet Connection Interface .....	49
6.8.4	Crowdfunding Smart Contract .....	50
6.9	Sprint 6.....	52
6.9.1	Goal .....	52
6.9.2	Connection to Back-End.....	52
6.9.3	Contract Read/Write Hooks.....	56
6.10	Sprint 7.....	57
6.10.1	Goal .....	57
6.10.2	Deploying Smart Contract from React Application.....	57
6.10.3	Unit Testing.....	58
6.11	Sprint 8.....	58
6.11.1	Goal .....	58
6.11.2	User Testing .....	58
6.11.3	Changes based on usability testing.....	58
6.12	Conclusion.....	58
7	Testing.....	59
7.1	Introduction .....	59
7.2	Functional Testing.....	59
7.3	Unit Testing.....	59

7.3.1	Manual Functional Testing.....	60
7.4	Usability Testing.....	61
7.4.1	Objective .....	61
7.4.2	Process and Materials.....	61
7.4.3	Results.....	61
7.5	Conclusion.....	65
8	Project Management .....	66
8.1	Introduction .....	66
8.2	Project Phases.....	66
8.2.1	Proposal .....	66
8.2.2	Requirements.....	66
8.2.3	Design.....	67
8.2.4	Implementation .....	67
8.2.5	Testing.....	68
8.3	SCRUM Methodology.....	68
8.4	Project Management Tools.....	69
8.4.1	Trello .....	69
8.4.2	GitHub .....	69
8.5	Reflection .....	69
8.5.1	Introduction .....	69
8.5.2	Completing a large software development project.....	70
8.5.3	Working with a supervisor .....	70
8.5.4	Technical skills.....	70
8.6	Conclusion.....	70
9	Business Opportunities .....	71
10	Conclusion.....	72
10.1	Summary .....	72
10.2	Future work.....	73
	References .....	74
	Appendices.....	78
	Appendix A – API Unit Testing Suites.....	78
	Appendix B – Smart Contract Testing Suite .....	78
	Appendix C – Functional Testing Results .....	78
	Appendix D – Usability Testing Tasks.....	90
	Appendix E – Usability Testing Survey .....	90
	Appendix F – Usability Testing Results .....	90

## 1 Table of figures

Figure 1: Vent Launchpad home page.....	18
Figure 2: Vent Launchpad project page.....	18
Figure 3: The Giving Block home page.....	19
Figure 4: The Giving Block donation process.....	20
Figure 5: Persona 1.....	21
Figure 6: Persona 2.....	21
Figure 7: Use case diagram.....	22
Figure 8: Application Architecture Diagram.....	27
Figure 9: Database Entities.....	28
Figure 10: Explore page paper prototype.....	28
Figure 11: Campaign page paper prototype.....	28
Figure 12: Dashboard paper prototype.....	29
Figure 13: Profile page paper prototype.....	29
Figure 14: Explore page wireframe in Figma.....	29
Figure 15: Campaign page prototype in Figma.....	30
Figure 16: Donate window prototype in Figma.....	30
Figure 17: User flow diagram.....	30
Figure 18: Website layout.....	31
Figure 19: Campaign grid.....	31
Figure 20: Style guide.....	32
Figure 21: Scrum framework diagram ( <i>Source: scrum.org</i> ) .....	34
Figure 22: Storage.sol smart contract.....	36
Figure 23: High-fidelity prototype in Figma.....	37
Figure 24: Ganache graphical user interface.....	38
Figure 25: Adding Ganache network to MetaMask.....	38
Figure 26: Basic React dapp interface.....	39
Figure 27: Requesting accounts from MetaMask.....	39
Figure 28: Instantiating Ethers provider, signer and contract.....	39
Figure 29: Calling get function from React using Ethers.....	40
Figure 30: Bank.sol smart contract.....	40
Figure 31: React page for interacting with Bank smart contract.....	41
Figure 32: Calling 'deposit' function from React using Web3.js.....	41
Figure 33: Confirming a transaction in MetaMask.....	42
Figure 34: Calling 'withdraw' function from React using Web3.js.....	42
Figure 35: Server folder structure.....	43
Figure 36: Connecting to MongoDB with Mongoose.....	43
Figure 37: Mongoose Campaign Schema.....	44
Figure 38: Mongoose User Schema.....	44
Figure 39: Create and edit API functions.....	44
Figure 40: Testing API routes and functions in Insomnia client.....	45
Figure 41: Creating theme using createTheme.....	46
Figure 42: Implementing theme in App.js.....	46
Figure 43: Layout.js.....	46
Figure 44: Loading React components using Lazy.....	47



Figure 45: Completed site layout.....	47
Figure 46: Charity Card.....	48
Figure 47: Campaign Card.....	48
Figure 48: Campaign donation card.....	48
Figure 49: Charity donation card.....	48
Figure 50: Campaign creation form.....	49
Figure 51: Web3Modal Interface.....	49
Figure 52: Wagmi and Web3Modal implementation in App.js .....	50
Figure 54: Crowdfunding.sol smart contract.....	51
Figure 55: Sending API request from React with axios.....	52
Figure 56: Creating data with axios request.....	53
Figure 57: Filtering campaigns in axios requests.....	53
Figure 58: Filter options in BlockAid.....	54
Figure 59: Mongoose Charity Schema.....	54
Figure 60: Creating authentication token with jsonwebtoken.....	55
Figure 61: Authentication Middleware.....	55
Figure 62: useDeposit React hook.....	56
Figure 63: Implementing deposit hook.....	56
Figure 64: Calling deposit.....	56
Figure 65: useContractDeploy hook.....	66
Figure 67: Results of edit user profile ease-of-use question.....	62
Figure 68: Results of edit campaign ease-of-use question.....	62
Figure 69: Create account window before usability testing.....	63
Figure 70: Create account window after usability testing.....	63
Figure 71: Card image before usability testing.....	64
Figure 72: Card image after usability testing.....	64
Figure 73: Example implementation of transaction fees.....	73

## 2 Introduction

In 2022, the global crowdfunding market was valued at \$1.25 billion USD and is projected to increase to \$1.62 billion USD by the year 2030 (Fortune Business Insights, 2023). Meanwhile, the market capitalization of the cryptocurrency market grew from \$1.54 billion USD in 2013 to \$3 trillion in late 2021 (Best, 2023). There is clearly opportunity for those who raise money through online crowdfunding platforms to diversify the types of donations they can receive by accepting cryptocurrency donations.

The release of the Ethereum blockchain network in July 2015 allowed developers to create 'decentralised applications'. These are programs built with Ethereum 'smart contracts', interactable programs that reside on the blockchain. Smart contracts facilitate the execution of an agreement without the need for an intermediary. Once deployed to the blockchain, they are immutable and run autonomously. The potential uses range from facilitating financial transfers to building decentralized autonomous organisations (DAOs) (Cryptopedia, 2021).

A decentralised crowdfunding platform, 'BlockAid', was developed for this project. BlockAid allows both users and charity organisations to create fundraising campaigns that receive cryptocurrency directly from users. An Ethereum smart contract was developed using the Solidity programming language to facilitate this functionality. The smart contracts are deployed to the Polygon Mumbai Testnet where it is publicly available.

The aim of this project is to explore the available technologies used to develop and interact with smart contracts from a traditional MERN stack application. As new technologies could prove overwhelming for users of traditional crowdfunding platforms, significant focus is also on the usability of the application. This thesis documents the process of planning and developing this application.

Research was conducted in the areas of blockchain technology, the Ethereum network and decentralised applications to inform the design of the program. Existing decentralised applications were examined to assess the requirements for BlockAid, and where it could improve on other projects. The process of designing the structure of the application as well as the user interface is documented.

The implementation of the web application used scrum methodology, being carried out over eight sprints. Progress made in each sprint is documented in this thesis.

Unit tests were carried out to test the internal logic of the application, and usability testing was conducted to assess the ease-of-use for first-time users.

Also discussed is the process and challenges of undertaking a large-scale web application development project, business opportunities for such an application and further development which could be done.

## 3 Research

### 3.1 Introduction

The purpose of this review is to examine existing literature to cover these fundamentals of blockchain, and the recent developments that have taken place to allow more traditional processes to utilize it. This subject is vast, and a great deal of information is outside the scope of what is required for development. Thus, this review is focused on three main areas:

1. Blockchain Technology
2. The Ethereum Virtual Machine
3. Decentralised Applications

### 3.2 Blockchain Technology

To understand the technology behind decentralised applications, knowledge of blockchain fundamentals is required. IBM defines blockchain as “a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network” (IBM, 2022).

Blockchain networks are decentralised, meaning they are processed by a number of nodes. This is different to a central bank, for example, where one authority operates the process. Nodes are individual computers connected to the network that share their processing power. Each node has access to the same ledger (record of transactions and balances) and is required to have identical data to the other nodes in the network in order to process a transaction.

The first well-known blockchain ‘network’ is Bitcoin (Schwab Center for Financial Research, 2022). The architecture of Bitcoin’s system is what newer blockchain networks like Ethereum have expanded on. The use of Bitcoin’s network has grown significantly since its inception in 2010 with one Bitcoin initially costing \$0.0008 (Kelly, 2023). This value peaked at just over \$69,000 in 2021, with the number of daily transactions reaching almost 440,000 in April 2019 (Blockchain.com - Confirmed Transactions per Day).

An overview of how the Bitcoin network operates is provided by Arya et al. (2021). Blockchain architecture is the section of this paper of most relevance to the purposes of this review. The structure of the blockchain network is explained by the authors in its individual parts.

Examining the Bitcoin network, the ledger’s data is stored in ‘blocks’ – a chunk of data containing the ‘hash’ (a digital ‘fingerprint’) of the previous block and the transactions carried out within that block. In Bitcoin’s case, this ledger is publicly viewable online (*Blockchain Explorer - Bitcoin Tracker & More* 2023).

A Bitcoin 'account' or 'wallet' is how a user accesses their currency. Users have two keys – a public key and a private key. The public key is like an email address, used to receive tokens. The private key is the user's password – this is used to cryptographically sign each transaction that the user performs. The network requires each transaction to be signed.

The Bitcoin network operates with what is known as 'Proof of Work', or 'POW'. This means miners are rewarded for using the computing power of their systems to validate transactions, creating blocks. This is known as 'mining blocks'. When a miner processes a transaction, the data is broadcasted to the network where it is grouped with other transactions to create a block. Each transaction contains a timestamp to ensure the transaction has not been created before, preventing repeat transactions inside blocks.

The Bitcoin protocol has several safety measures to prevent security problems such as forged transactions. The fact that the network is vast and requires all nodes to agree upon the information contained in their copy of the Bitcoin ledger prevents one node processing a transaction that wouldn't be allowed given the block history of the public ledger. The transactions themselves contain a random 32-bit number called a 'nonce' (number only used once) to ensure each transaction can only be added to the ledger once. These characteristics play a large part in why so many have adopted blockchain technology and believe in its potential as an alternative to traditional banking.

While this paper as a whole covers subjects outside the scope of this review, information on the origins and fundamentals of blockchain technology are explained well. This knowledge is vital in understanding the developments since the Bitcoin network was released, and how the blockchain industry has progressed to integrate with more traditional and popular technologies.

### 3.3 Ethereum Virtual Machine

Bitcoin's network is known as 'Blockchain 1.0', its purpose is simply to transfer currency from user to user. While companies already support Bitcoin as a payment method for goods and services (Modderman, 2022), the capabilities of Bitcoin are limited. In 2014, the whitepaper for the 'Ethereum' project was published (Buterin, 2014).

Rather than creating a network solely designed to support the transfer of currency, Ethereum aims to take advantage of the blockchain's inherent security and decentralisation benefits to allow programs to reliably run within blocks, written in their own programming language.

Accounts on the Ethereum have their own 20-byte address, and function similarly to accounts on Bitcoin's networks with users owning a private key to sign transactions. The balance of the account is in Ether, the currency used to pay transaction fees on the network.

As well as a user account, an address could point to a 'Smart Contract', the scripts that reside on the blockchain and execute code when interacted with. Each Ethereum transaction contains a 'message', and when this message is sent to the smart contract's address, the code is run. Like other scripts, the

contract can contain a number of functions, which could be used to transfer tokens between accounts or process mathematical problems when conditions are met.

For example, a contract could:

- Return the balance of an account when given an address
- Lock tokens, acting as a savings account
- Create a custom cryptocurrency with value tied to a real-life asset
- Create a record of non-financial data, for example a voting system

Contracts reside on the blockchain, making the code permanent. The code is executed by the Ethereum Virtual Machine (EVM) and is in EVM bytecode format. According to the Ethereum whitepaper, this code is Turing-complete, meaning “EVM code can encode any computation that can be conceivably carried out, including infinite loops”. There are, however, languages that contract code is written in before being compiled into this bytecode. One such language is Solidity (*Solidity 0.8.19 Documentation*, 2023) which holds 87% of contract balances on the network (Chainlink, 2022).

Since the launch of the Ethereum, there have been numerous forks of the network (*Chainlist* 2023). One such example is the Binance Smart Chain created by centralised exchange Binance (*BNB chain: An ecosystem of blockchains*, n.d.). While the fundamental architecture and way in which users interact with other networks is the same, developers of these forks may optimise their own back end to reduce the computing power needed to process transactions, reducing the overall cost of fees to users.

In addition, there are services such as Ganache which emulate the Ethereum network in a local environment, which will be used for development and testing of contract code required for the application (*What is ganache?*, n.d.).

This overview of Ethereum by its creator explains how the network builds on previous blockchain networks, and it’s potential to be used where existing technologies are dominant.

In the development of the crowdfunding application, smart contracts are used to manage user accounts, receive funds from donors and keep records of these transactions. The functionality of Ethereum smart contracts is ideal for these actions and is the only viable system for doing so currently. This document explains well how these are interacted with, as well as listing possible uses. The whitepaper also mentions that smart contracts can have Ether balances – contracts that handle donation pools on the crowdfunding application will need to hold tokens so that automation in payments can be done.

### 3.4 Decentralised Applications

Decentralised Applications (DApps) are applications that use Ethereum smart contracts to run. The crowdfunding web application will use these to carry out its functions.

Examining a case of smart contracts being used to perform traditional real-world tasks gives insight into the practical aspects not covered in the Ethereum documentation, such as cost and efficiency. An article by Christodoulou et al. (2019) demonstrates the development methodology and practicalities of using smart contracts for these purposes.

The paper describes the process of developing and testing a decentralised application for intended for the logistics industry that can be used to track the shipping progress of an item from its origin to its destination. It also analyses the test use of the smart contract that was developed, examining its efficiency and cost.

The authors discuss why the addition of blockchain technology may benefit this real-world process. Logistics is an area where there are often many different parties involved with the transportation of an item, and the consumer often does not have the most accurate information from different carriers. Recording item data to a permanent ledger accessible via a web application aims to consolidate this information and solve the issue of transparency using a public blockchain.

The smart contract was written in Solidity and tested on the Ropsten Test Network before being deployed on the Ethereum main network.

Writeable functions were written in the contract code to log an item at the various points in its journey – `sendProduct()`, `sign()`, `maintenance()`. A number of readable functions were also available for a customer to get various details about their package.

It is proposed that customers would view the progress of their item either through a public blockchain explorer such as Etherscan (*Etherscan.io Blockchain Explorer*) or through a web app developed using the Ethers.js development library, which provides components to interact with smart contracts through JavaScript web apps.

When examining a test use of the contract, the authors recorded gas values (transaction fees), as well as execution time for each contract interaction. This concluded that the maximum fee for invoking the functions was between \$0.25 and \$1.19, and the execution time was between 3 and 20 seconds. Also noted is the fee for deploying the contract on the blockchain (\$1.48).

The paper concludes that the use of decentralised applications in this particular industry allows developers to create a “secure and transparent application”.

This resource provides many useful insights into the methodology behind developing a decentralised application for a real-world use case. Having the details of each item reside on the blockchain eliminates the need for more complex databases, the contract is coded once and is ready to use at all times on the network.

While there are transaction fees for interacting with the contract, further research is required to investigate the cost-efficiency of other blockchains, such as the aforementioned Binance Smart Chain, and perhaps compare this aspect with the traditional infrastructure required to carry out these tasks.

Also referenced in the paper is the smart contract deployed by the authors which is viewable on the Ethereum network. This contains the detailed code of the functions which may prove useful to development of functions for this project.

While this example examines the approach to smart contract use in real-world applications, it does not investigate displaying this data with a front-end interface. To develop the crowdfunding application, frontend development must be considered, as this is how the user perceives the service. In a thesis by Nguyen (2022), the author described the methodology of using Solidity and the React framework to develop a decentralised application.

The author aims to understand the blockchain development environment and examine various development tools that can be used to build a decentralised application and put this in practice by developing such an application that can send Ether from one account to another containing a message. The result can be seen on the blockchain. The benefits and limitations of using smart contracts are outlined in this paper. Their automaticity, security and transparency are highlighted as positives, however the limitation of smart contract's inability to send HTTP requests is identified. This is used frequently in standard web applications, meaning the purposes of each contract need to be strictly defined for each case in which they will be used. The use of the Solidity programming language is required to develop smart contracts, and the author provides a strong argument for its use over other languages, such as Vyper.

When considering options for web application frameworks to be used in combination with Solidity, the author chose to use JavaScript and React, a front-end framework (*Getting started - React*). This decision to use JavaScript was made because of its speed, popularity and versatility – particularly with the use of Node.js.

Particularly notable is the use of the Ethers.js JavaScript library (*Documentation - Ethers.js*). This library provides ways for traditional web applications to interact with the Ethereum Blockchain, and smart contracts deployed on it. Users can connect their personal accounts to a web app using the library, meaning they can initiate transactions and contract interactions. Ethers.js also provides a reliable testing process for blockchain based apps.

As well as this, the author includes some of the code use to create the smart contracts and integrate them with the web app. This is critical for the development of this application, and inspiration can be taken from this project's methodology.

The result of this development was a React application capable of connecting to a user's personal account and sending a specified amount of Ether to a given address by connecting to the Ethereum blockchain.

While the first paper examined in this section covers the uses of smart contracts in real world situations, this thesis demonstrated the practical aspects of developing a decentralised application. The resources and development tools used to create the application are well-described, and compelling reasons to use such tools are given. A conclusion can be made that JavaScript and React should be used to develop this project's application.

The literature covered in this review covers the three primary areas of blockchain within the scope of the project's requirements. The subject of blockchain fundamentals is crucial for understanding the Ethereum network and other networks that use the Ethereum Virtual Machine.

From examining the real-world examples of decentralised application development, recommendations can be made regarding methodology and development tools, particularly the use of JavaScript and the frameworks mentioned. Additional research into the use of Solidity for smart contract development is likely needed to ensure proper practices regarding efficiency and security.

Overall, the resources examined in this review provides important knowledge of blockchain technology and the Ethereum protocol, demonstrates the potential uses of decentralised applications and suggests development tools and practices for designing and implementing this knowledge in this project's web application.



## 4 Requirements

### 4.1 Introduction

This chapter discusses the process by which the application requirements were developed. Similar applications were researched and analysed to assess what should be expected of this type of application, and where BlockAid could improve over competitors.

The requirements modelling section focuses on establishing the requirements for the application and considers the needs of the user.

### 4.2 Requirements gathering

The requirements phase aims to determine what the functional needs of the web application should be. This involves examining the needs of the different types of users and analysing similar existing applications. User personas were developed to aid the design and development process by having an end-user in mind. This section also maps out the use cases of the different user types, and the feasibility of the project.

#### 4.2.1 Similar applications

When examining existing website that support raising funds with cryptocurrency via crowdfunding, two websites were identified that operate solely in cryptocurrency. These are Vent Launchpad and The Giving Blocks. This section examines the efficacy and usability of these websites.

##### 4.2.1.1 *Vent Launchpad*

Vent is a decentralised application for users to raise funds for their projects through user donations. Often these projects are 'tokens', another word for cryptocurrencies. The money raised through the platform is usually used to fund development costs for project utility, as well as the liquidity needed to allow their tokens to be traded. Users are rewarded by receiving their tokens at a pre-determined price.

On the home page, the user is presented with a table of fundraisers active on the site (Figure 1). The project logos are displayed along with some limited data about each item.

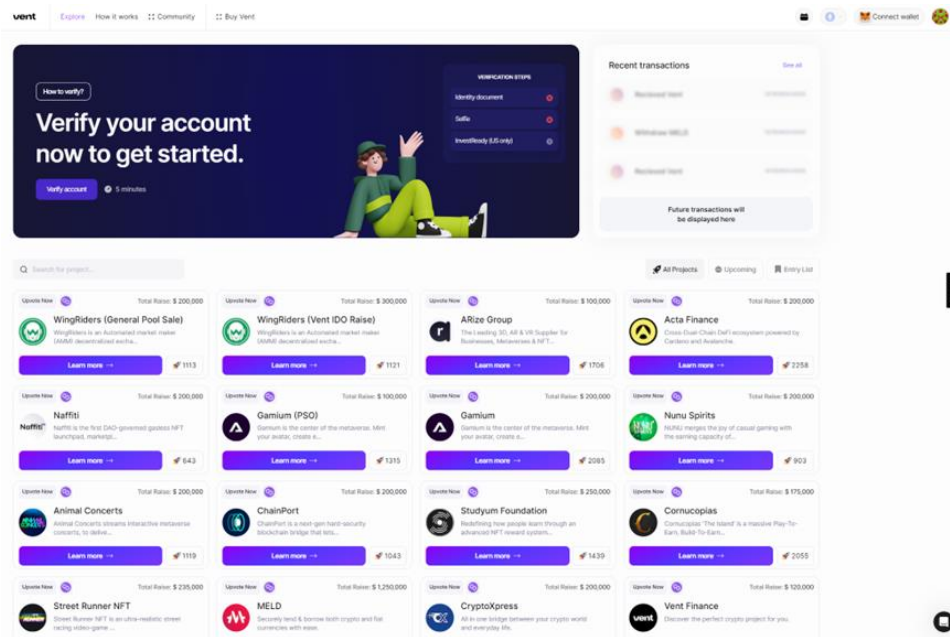


Figure 1 Vent Launchpad home page

The main page for each fundraiser (Figure 2) shows the user a description, logo, and information about the creators of the fundraiser.

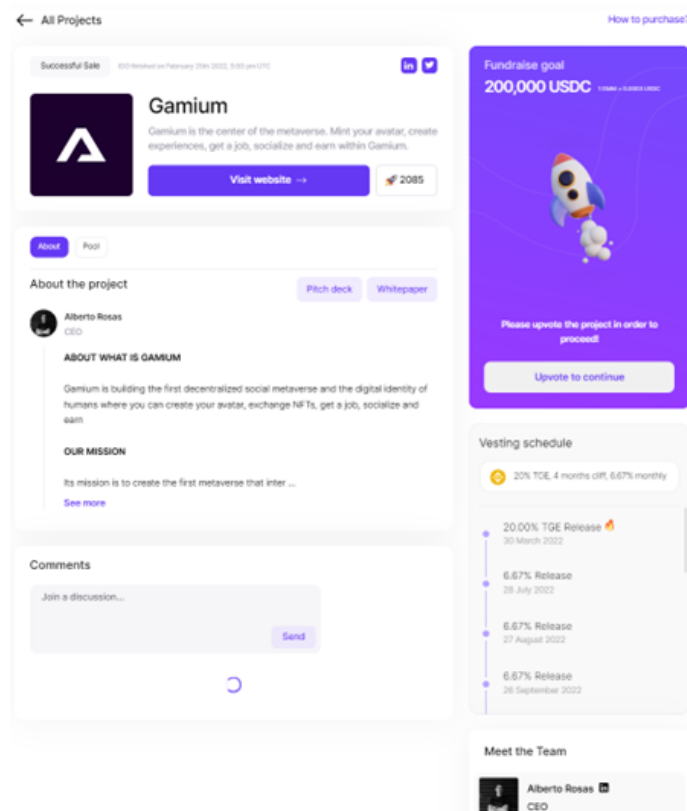


Figure 2 Vent Launchpad project page

The presentation of the details the user needs to know at first glance is well presented in a concise format. However, there are some usability problems when considering the perspective of those with no experience with decentralised applications.

Regular users need to register to contribute to a project's fundraiser. This makes sense for project owners, but regular users should only need to have their wallet connected, as only they have access to their private key, acting as verification.

Requiring users to 'upvote' a fundraiser before seeing where they should contribute could be confusing to those not familiar with this type of website. There is also no clear guide on how to set up a browser-based wallet, making it less accessible for users

#### 4.2.1.2 *The Giving Block*

Another website examined when researching existing similar web application was 'The Giving Block' (Figure 3). This site is intended for non-profit organizations to accept cryptocurrency as donations. 86 different currencies are supported for donations.



Figure 3 The Giving Block home page

Users can manually send cryptocurrency from their exchange account or personal wallet, as well as donate through MetaMask in their browser. This is the functionality being examined on this site.

While the site is robust, it is not immediately clear what users need to get started. When the user reaches the page for the charity they wish to donate to, they must navigate through numerous dialogue pages to get to the point where the donation is made (Figure 4).

The figure displays a sequence of five screenshots from the Giving Block website, illustrating the donation process for 'Big Brothers Big Sisters of Miami'.

- Screenshot 1: Make a Donation** - Shows the 'Crypto' tab selected, with options for BTC, ETH, and USDC. The amount '0.1' is entered, and the value is shown as '\$189.51'. A 'Next' button is at the bottom.
- Screenshot 2: Personal Info** - A form for personal information including 'First name', 'Last name', 'Email', 'Address 1', 'Address 2', 'Country', 'State/Province', 'City', and 'ZIP/Postal Code'. A 'Next' button is at the bottom.
- Screenshot 3: Want A Tax Receipt?** - A form asking if the user wants a tax receipt. It includes a text input for 'Enter email for tax receipt' and 'Skip' and 'Get receipt' buttons.
- Screenshot 4: 0.1 ETH** - Shows the amount '0.1 ETH' and a QR code for the donation. It includes a 'Donate with MetaMask' button and a 'Start Over' button.
- Screenshot 5: Confirmation** - Shows the transaction details, including the amount '0.1 ETH' and the value '\$189.51'. It includes a 'Reject' button and a 'Confirm' button.

Figure 4 The Giving Block donation process

Customers wishing to raise money through cryptocurrency on The Giving Block must go through an application process and be manually approved. While this is important for security reasons with charity organisations, individual crowdfunding sites like GoFundMe allow users to immediately begin receiving donations.

## 4.3 Requirements modelling

### 4.3.1 Personas

To aid in the design of the applications, 'personas' were created that represent potential users of the application. In user experience design, personas are fictional characters that represent the different user types that use a product, designed based on research (Dam & Siang, 2022).

Two personas were designed for the two user types – donors and creators. The first persona (Figure 5) represent a user that already holds cryptocurrency, and wants to find a cause to donate to

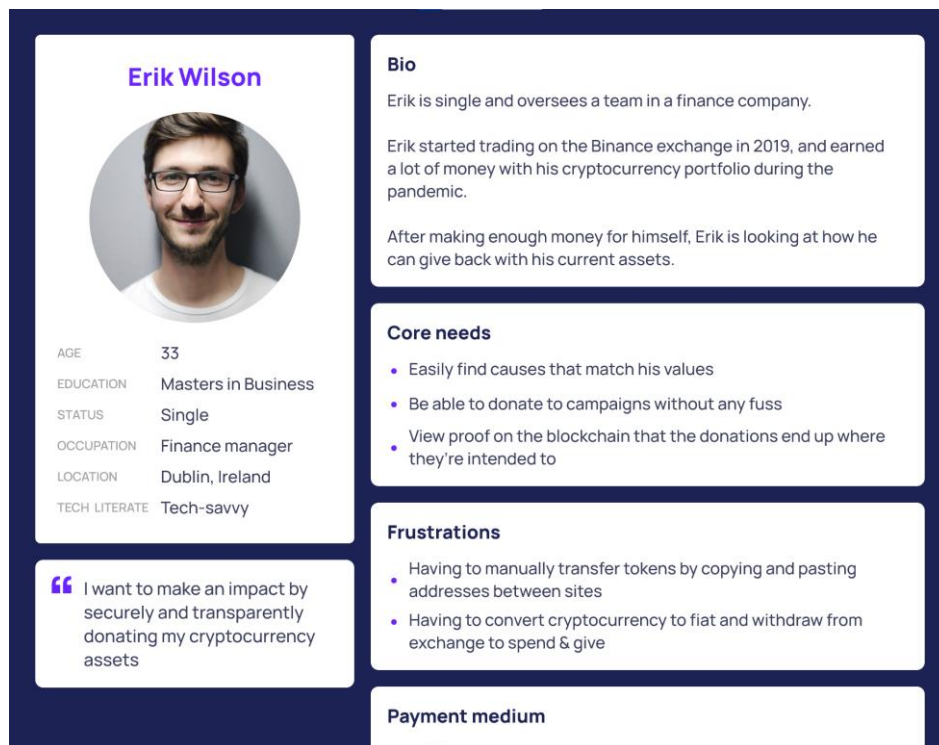


Figure 5 Persona 1

The other type of user on this platform is someone who wants to raise money. The second persona (Figure 6) represents this demographic. This persona is not as tech-literate as the first, so design with this type in mind is important for user experience.

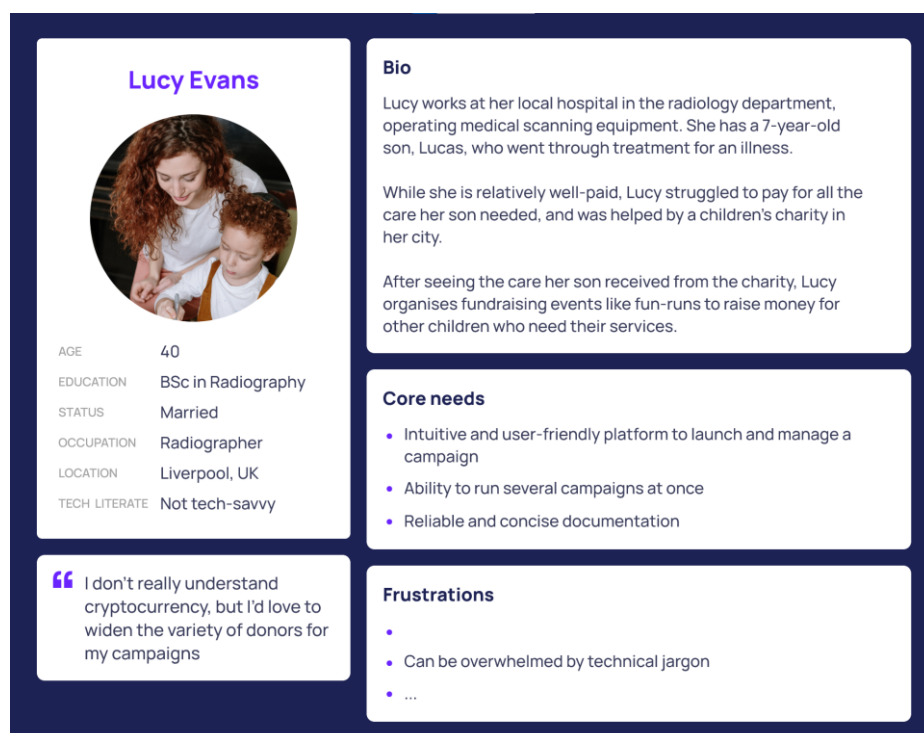


Figure 6 Persona 2

#### 4.3.2 Functional requirements

Functional requirements are features that are necessary for the application to operate. Based on the research conducted, the following features are deemed necessary:

- Connect to users' existing browser wallets
- Users can search campaigns by title and category
- Users can donate a specified amount of cryptocurrency without manually sending transactions
- Registered users can create crowdfunding campaigns, deployed to the blockchain
- Campaign owners only can withdraw raised funds to their wallets

#### 4.3.3 Non-functional requirements

Along with functional requirements, non-functional requirements focus on how well the application performs and how usable it is. Non-functional requirements of this application include:

- Clearly showing campaign progress from reading blockchain data
- Image upload and display for campaigns and users
- Responsive layout to suit varying screen sizes

#### 4.3.4 Use Case Diagrams

The use case diagram (Figure 7) illustrates the two characters that interact with the application. The campaign creator may create a fundraising campaign and edit its details, as well as claim the cryptocurrency that it has raised.

A contributor, after connecting their wallet to the website, can browse and search for campaigns, and contribute a specified amount to them through the user interface.

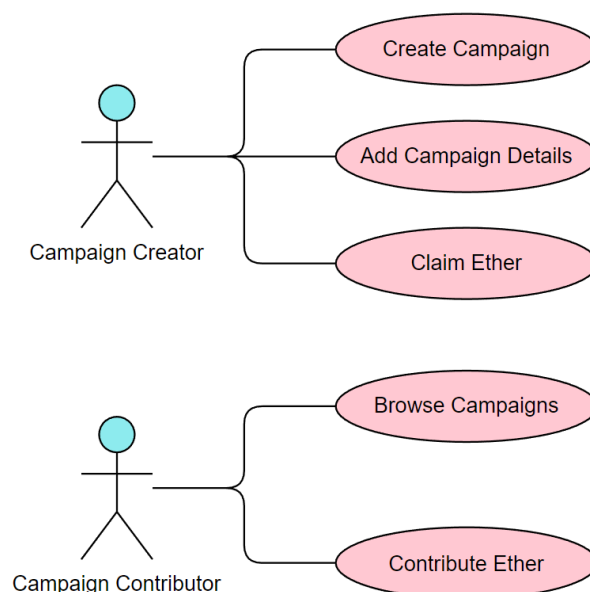


Figure 7 Use case diagram

## 4.4 Feasibility

Table 1 outlines challenges that may be faced during the development of this application.

*Table 1 Feasibility Table*

Challenge	Description	Solution
Variety of technologies	There are many options for technologies to be used, especially relating to blockchain. Must determine which are best to use.	Use research and requirements gathering to inform choices, assess options during early implementation phases.
Inexperience with Solidity	With no experience with using the Solidity smart contract programming language, creating the blockchain-side logic will be difficult.	Solidity documentation and smart contracts from other web sources should be reviewed to gain understanding.  Break down overall functionality into separate smart contracts.
Time	The completion of this project has a strict deadline. Additionally, progress reports and sprint reviews throughout have strict deadlines.	Follow sprint outline guidelines provided. Document sprint reviews for reflection.
Project Management	This web application is of a larger scale than previously completed projects, managing time and resources along with prioritization of tasks could prove challenging.	Utilise SCRUM methodology and good project management practices to keep to schedule. Use support of supervisor to plan sprints.

## 4.5 Conclusion

Two existing blockchain-related crowdfunding web applications were examined. Their usability was assessed, and strengths and weaknesses were identified. This analysis aided in the definition of functional and non-functional requirements for the development of the crowdfunding web application. Additionally, two user personas were created based on this research. In the design phase, these promote user-focused design practices.

The feasibility of this project was also discussed. Four potential challenges were noted, and potential solutions were suggested for each.



## 5 Design

### 5.1 Introduction

This section described the design process for the web application. Using information gathered in the research and requirements phases, the design for all aspects of the application was finalised. This includes the design for the user interface as well as the technologies that make up the application, aiming to make the implementation process straightforward.

### 5.2 Program Design

This section outlines the design of the application itself and informs the implementation phase of the project.

#### 5.2.1 Technologies

MERN stack was chosen for the development of this application. This stack is a combination of MongoDB, Express.js, React.js and Node.js. Some prior experience using these technologies was held, but knowledge was greatly expanded during the implementation of this application.

MongoDB is a NoSQL database (*What is mongodb?*) that stores data in JSON-like documents. MongoDB provides free drivers in the form of an npm package, that can be integrated into the express.js backend to access the database. MongoDB offers free online database hosting which was easy to operate.

Express.js is a minimal and flexible Node.js application framework (*Express - Node.js web application framework*). In this project, it was used to build a robust API that handles requests from the React frontend, retrieving data from the MongoDB database and handling authentication. Combined with authentication libraries, middleware can be used where needed to expand the applications functionality

React.js is a front-end library that was used for the interface of this application. React allows the creation of reusable components that can be used across the application. Using React means that JavaScript operations can be carried out, and can control the end-result of what the user sees in the DOM. React works well when integrated with libraries such as wagmi and Web3Modal, which are used to interact with the blockchain. These libraries provide 'hooks', state-modifying functions that allow components to access and reuse logic, and share state data across the application.

React provides built-in hooks such as 'useEffect' for managing side effects and 'useState' for updating state variables, allowing components to render dynamic content and trigger re-renders when necessary.

Finally, Node.js is the environment used for running server-side JavaScript code. It was used with the express.js framework to develop the API. Additionally, the Node Package Manager ('npm') is a tool used to install packages and dependencies across both the front and backend.

Web applications that interact with the Ethereum Virtual Machine have a variety of JavaScript libraries to suit the individual needs of the project. Using React.js with a Node.js backend allows these libraries to be quickly and easily integrated.

To develop Ethereum smart contracts, the Solidity programming language was implemented. Alternatives to Solidity include 'Vyper', which features a Python-like syntax.

The final prototype connects to the Polygon Mumbai test network. This test network emulates the system of the public mainnet, meaning all processes work identically, but the currency on the network has no value. This means that as much testing can be done without a financial cost, and users anywhere can connect to this network, unlike a local blockchain client like Ganache.

To interact with this blockchain, the 'Ethers.js', 'wagmi' and 'Web3Modal' libraries were utilised. Ethers is a JavaScript library that provides functionality for interacting with smart contracts on EVM-based blockchains, such as the Polygon Mumbai test network that is in use. This includes calling functions on smart contracts, as well as creating data for other types of transactions such as sending currency or deploying contracts. Ethers also includes a number of utilities to work with the various datatypes used on the blockchain. The 'wagmi' JavaScript library is a package of React.js hooks that build on the functionality of Ethers by providing functions to get and interact with data on the blockchain from a React application. It provides an instance that can be accessed across components, and provides hooks that can auto-refresh blockchain data to be displayed on the front end. Wagmi hooks are integrated into the 'Web3Modal' package which was used in the project. Web3Modal offers an interface and capability of connecting to a variety of different cryptocurrency wallet providers, such as MetaMask, TrustWallet, and Coinbase wallet. It provides an interface to quickly implement these connections, and allows components to connect to the users wallet, and handle changes in network and account.

### 5.2.2 Application architecture

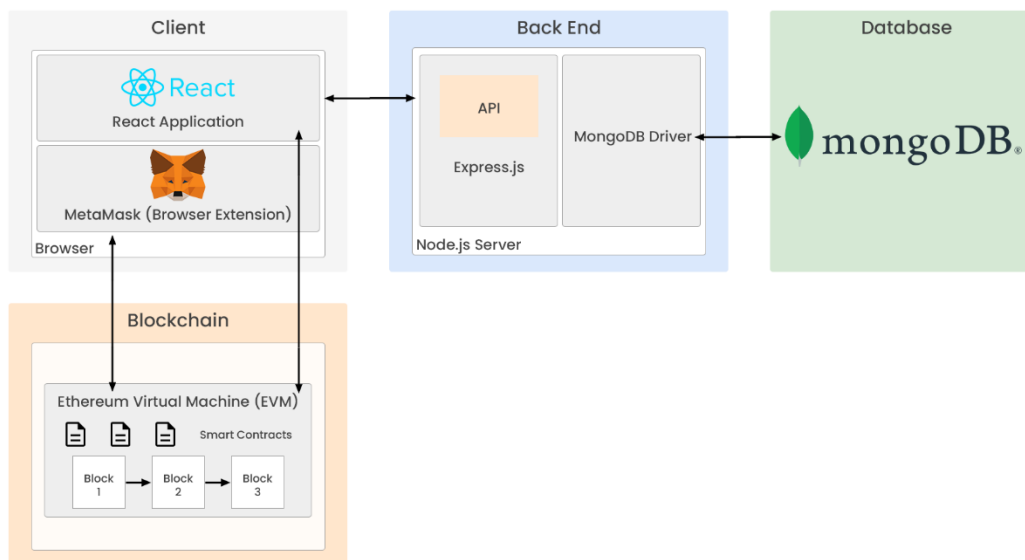


Figure 8 Application Architecture Diagram

Figure 8 illustrates the architecture for this web application.

Conventional MERN stack application architecture consists of a React frontend app, a Node.js and Express.js server acting as an API to carry out the application functions, and a MongoDB database to store information. The user's browser loads the React application, which uses a number of libraries. Most notably, Material UI is the CSS framework used to build the frontend components. The connection from React to the Express server is handled by 'axios', a promise-based HTTP client (*Getting started - Axios Docs*).

Also in the front end are the libraries that connect to the chosen Ethereum-compatible blockchain. Ethers.js and the libraries that depend on it access the blockchain via an RPC node (endpoint). This acts as a 'second backend'. The connection in the case of this application happens solely from the front end, as is separated from the Express server.

Functions are directed to the 'address' of the smart contract, which functions similar to a URL. The Ethereum virtual machine handles the internal logic of the smart contract and returns data to the front-end.

### 5.2.3 Database design

There are three types of data stored in the database for this application. Campaigns, Charities and Users. MongoDB is a document-based NoSQL database, meaning it is non-relational. However, there is a virtual relationship between the campaign and charity 'creator' fields and the user 'address' field. The structure of each object can be seen in Figure 9.

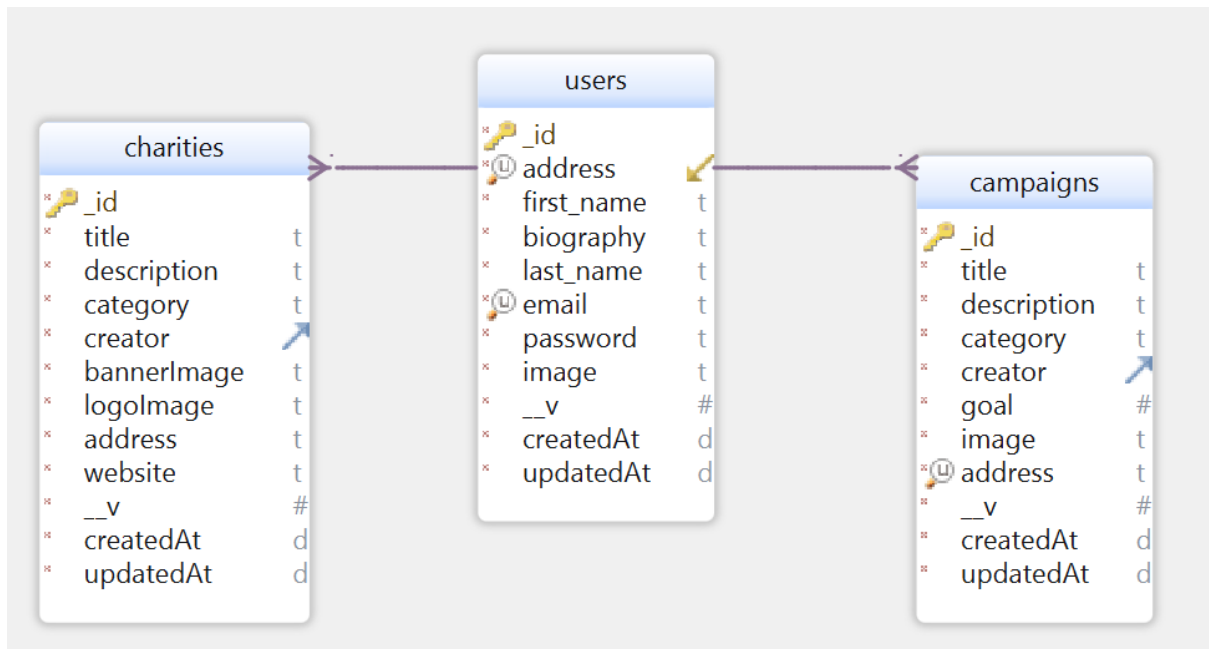


Figure 9 Database Entities

### 5.3 User interface design

This section outlines the aims for the user interface of this application, as well as the stages in its implementation.

#### 5.3.1 Wireframe

Initially, the layout of the web application was hand drawn on paper, as seen in Figure 10 through 13.

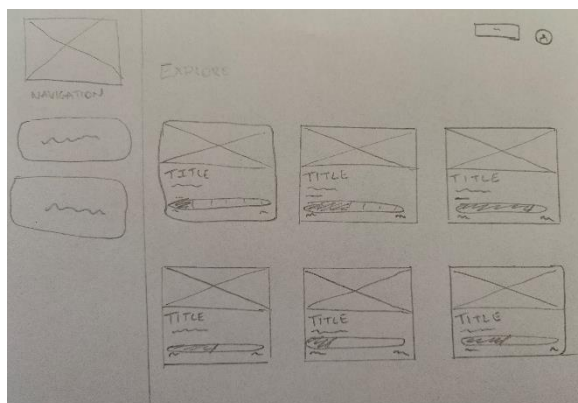


Figure 10 Explore page paper prototype

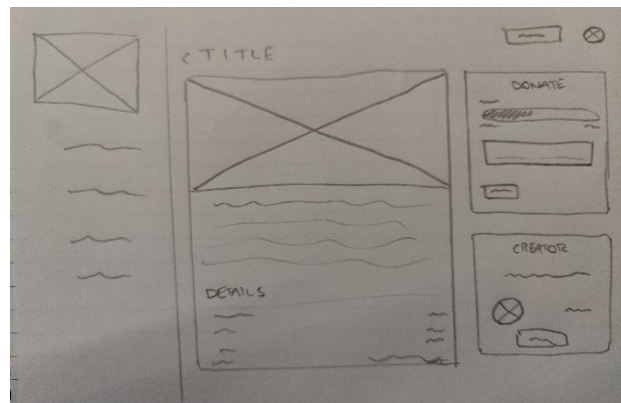


Figure 11 Campaign page paper prototype

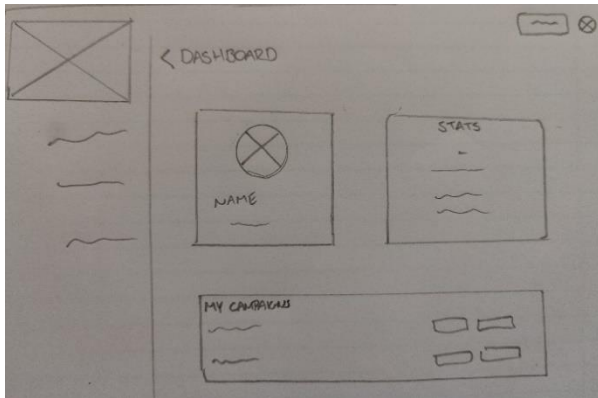


Figure 12 Dashboard paper prototype

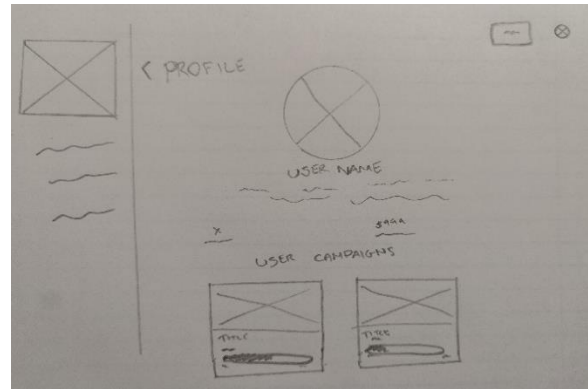


Figure 13 Profile page paper prototype

This prototype was later developed further using the Figma design tool as shown in Figure 14 through 16.



Figure 14 Explore page wireframe in Figma

Figure 15 Campaign page prototype in Figma

Figure 16 Donate window prototype in Figma

These prototypes were then used during the implementation phases to construct the layout of the web application.

### 5.3.2 User Flow Diagram

The user flow diagram in Figure 17 shows how the user navigates through the web application for the various tasks.

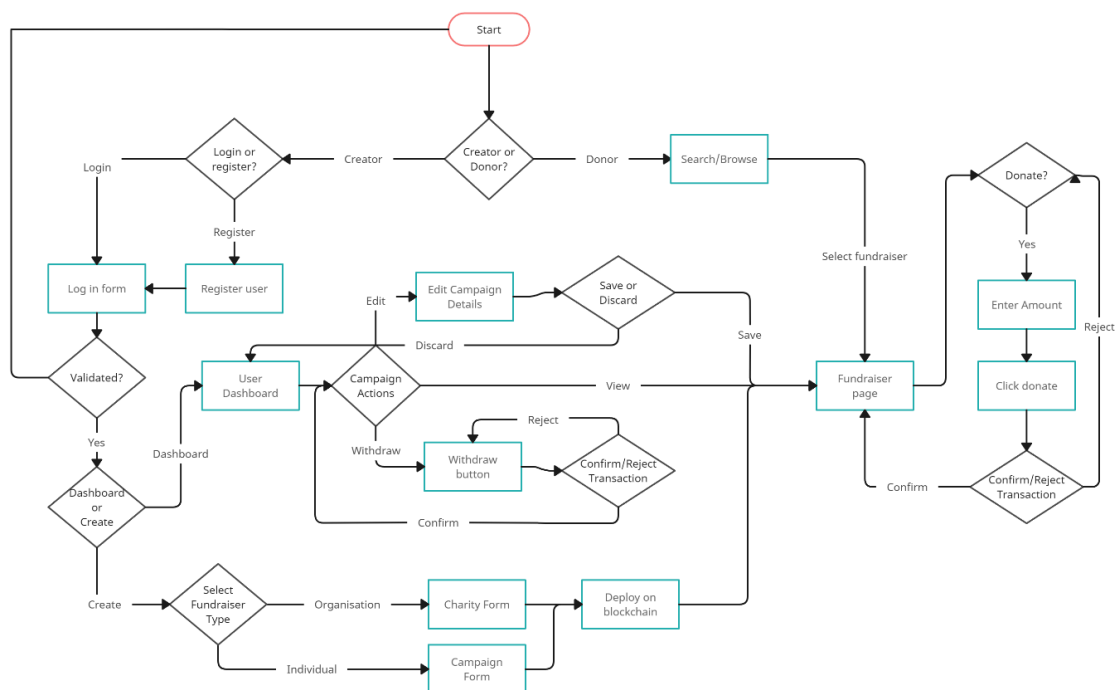


Figure 17 User flow diagram

### 5.3.3 Style guide

The web application employs a simple and straightforward layout, with navigation handles by the sidebar and user actions located in the header, including the connect wallet button and access to account settings (Figure 18). This approach intuitively differentiates between these two types of actions. The responsive layout adjusts to varying display widths by closing the sidebar when necessary.

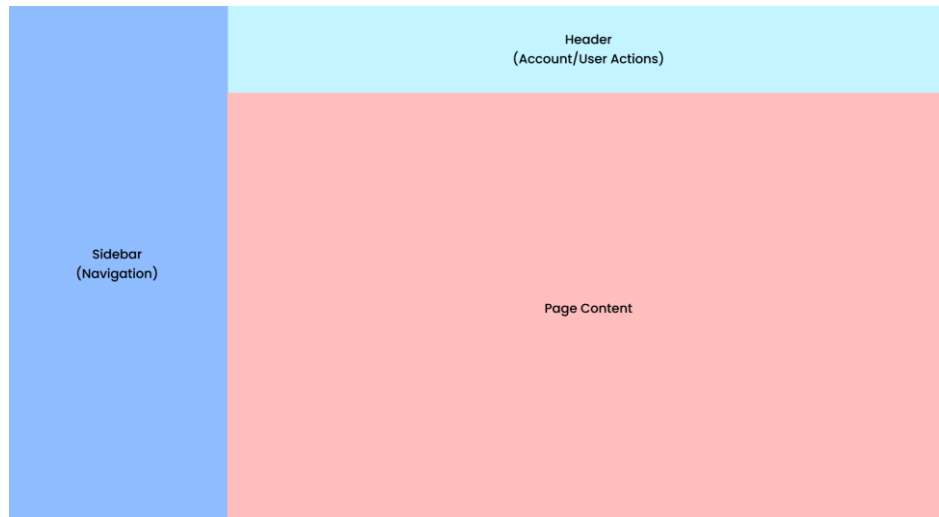


Figure 18 Website layout

When browsing fundraisers, the user is presented with a grid of cards arranged in a 3x3 format (Figure 19). Each card provides a quick preview of the campaigns title, image, and progress. This grid also adjusts responsively to display width, automatically reducing the number of columns.

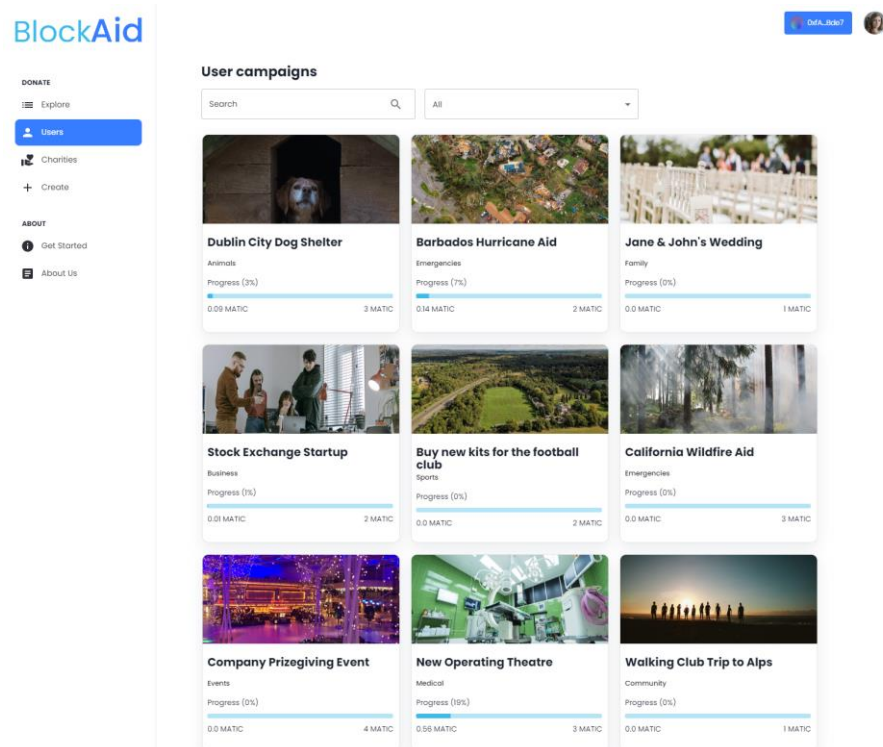


Figure 19 Campaign grid

Upon clicking on a card, users are provided with a detailed description, including blockchain details and a donation window. Figure 20 shows the colour scheme, typography and shadows used in the application.

Typography is designed with varying shades of black, creating contrast against the white background. White typography is reserved for button elements, enhancing their visibility. The modern and versatile Poppins sans-serif font is used throughout, aligning with the web application's modern design and functionality.

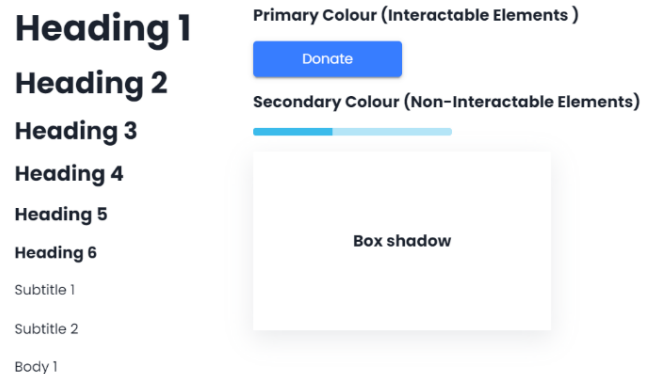


Figure 20 Style guide

## 5.4 Conclusion

In this chapter, the analysis of technologies researched informs the program design to be implemented to achieve the requirements set out for this project.

A layout for the application has been designed, focusing on separating navigation actions from user actions – an important focus due to the functionality of the app being different to traditional websites. A simple two-colour scheme is used to focus the user's eyes on interactable elements, and the modern Poppins font provides an easily readable interface.



## 6 Implementation

### 6.1 Introduction

This web application has been developed using the following technologies:

- **React.js**  
React is an open-source front-end JavaScript framework
- **Ethers.js, wagmi**  
Ethers.js is a JavaScript library used to interact with smart contracts on the Ethereum blockchain. Wagmi is a collection of React hooks built with ethers.js.
- **Solidity**  
Solidity is an object-oriented programming language, used to write smart contracts on the Ethereum blockchain.
- **Material UI (MUI)**  
Material UI is a React component library and CSS framework, containing prebuilt components that can be used and customised in React apps.
- **Truffle**  
The truffle suite is a collection of tools for smart contract development, including a local development blockchain (Ganache), contract compilation tools and testing tools.
- **MongoDB**  
MongoDB is a document-oriented, NoSQL database platform.
- **Express.js**  
Express is a back-end web application framework for building RESTful APIs with Node.js

The finished product from this project allows users to create their own fundraisers, as well as donate to others' fundraisers through their personal, browser based Ethereum wallet. This is done without the need to manually send their cryptocurrency from their exchange or personal wallet. The user enters an amount and clicks donate, their browser popup appears and they can send the transaction, transferring the tokens to the smart contract for that fundraiser. Their tokens are kept on the blockchain until the owner of the fundraiser claims them.

## 6.2 Scrum Methodology

Scrum is an agile project management methodology based on incremental development of a product (*What is scrum methodology? & scrum project management, n.d.*). Figure 21 shows a diagram of the scrum framework.

### SCRUM FRAMEWORK

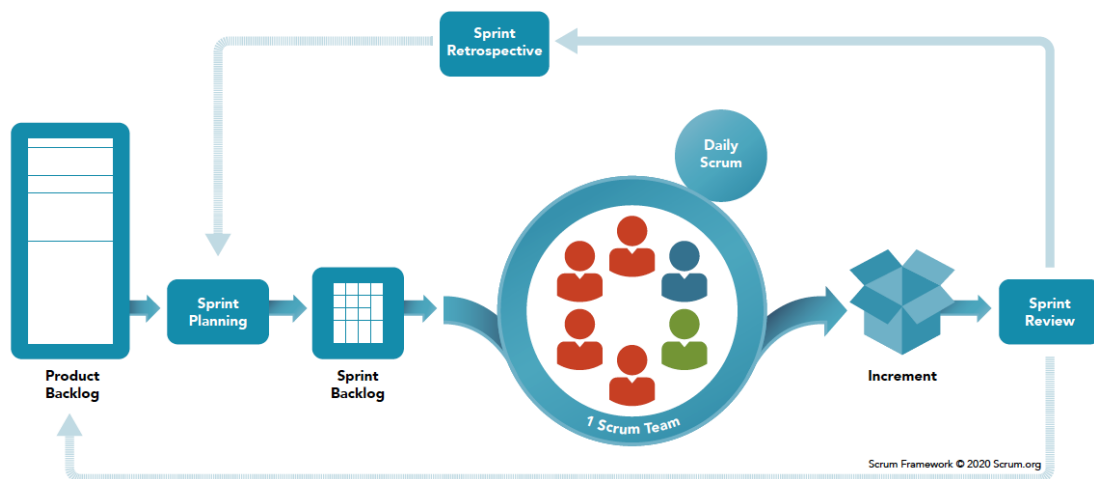


Figure 21 Scrum framework diagram (Source: scrum.org)

Scrum focuses on breaking down the overall development process into short lengths of time called 'sprints', where components of the product are delivered. In a team environment, the product owner defines the deliverables and the backlog – the tasks that need completing to ensure delivery of the product. The scrum master organises the development team, who are tasked with delivering the product. Sprint reviews are an important part in scrum, where the progress of the development can be assessed and previous work can be learned from.

Although this was not a team project, scrum methodology was used during the course of development to plan and monitor the process. Sprints were categorised based on the project requirements. The backlog on Trello was labelled by sprint. For example, sprint 4 focused on delivering the foundations of the server and layout of the website.

Sprint review meetings took place at the end of each sprint between the developer and the supervisor. There were 9 sprints overall in this project, each lasting approximately two weeks.

## 6.3 Development environment

Microsoft's Visual Studio Code was used in the development of the application. It provides syntax highlighting for JavaScript and HTML, the two most used languages in React. Additionally, the Solidity extension was installed, which added syntax highlighting for Solidity smart contracts and solidity compiling capabilities. Visual Studio Code provides a built-in terminal, and the ability to open multiple terminals at different directories. One terminal was used to run the express server, one to run the React application and another during development to run truffle commands for smart contract deployment and debugging on the Ganache test network.

In the initial stages of development, smart contracts were deployed to the Ganache local test network. This allowed smart contracts to be quickly tested and debugged without any cost, and transaction times were faster than public networks.

Further into the development process, the project was migrated to the Polygon Mumbai public test network (*Networks - Polygon Wiki*). Like Ganache, this network has no cost and tokens have no value, but users anywhere can connect and interact with it using its native token, MATIC.

Git was used for version control in the development of the application. GitHub desktop was used to visualise changes and to manage branches. Two branches were used, a development branch and a main branch, with the development branch used for making changes which were then merged with the main branch once working.

## 6.4 Sprint 1

### 6.4.1 Goal

The first sprint aimed to deliver a smart contract deployed to the Ethereum Goerli test network. This smart contract is capable of storing a string and retrieving this string through the blockchain. Additionally, this is the stage where a much higher fidelity prototype was created in Figma from the original wireframes.

### 6.4.2 Storage Smart Contract

Ethereum smart contracts have functions that can perform several tasks.

To investigate how smart contracts can be used to display stored data, a smart contract called `Storage.sol` (Figure 22) was written in the Remix online IDE (*Ethereum IDE & Community*). It was then compiled and deployed to the Ethereum Goerli test network.

```

1 // SPDX-License-Identifier: Unlicensed
2 pragma solidity >0.4.23;
3
4 contract Storage {
5
6     string public storedData;
7
8     event myEvent(string eventOutput);
9
10    constructor() {
11        storedData = "Default value";
12    }
13
14    function set(string memory myText) public {
15        storedData = myText;
16        emit myEvent(myText);
17    }
18
19    function get() public view returns (string memory) {
20        return storedData;
21    }
22 }

```

Figure 22 Storage.sol smart contract

A public variable, 'Storage', is declared. This can be read from outside the contract. The event 'myEvent' is also declared, which takes in a variable that the event outputs. The constructor function is executed when the contract is deployed and sets the storedData variable to the string 'Default Value'.

The 'set' function takes in a string and sets the storedData variable to that string. It also triggers the myEvent to emit the value inputted so it can be read externally. Finally, the get() function is a getter that returns the value of storedData.

This contract allows a user to set a value and later retrieve this value via the blockchain. Once the contract was compiled in the Remix IDE, it's ABI (Application Binary Interface) is available. This defines the functions of the blockchain and is used to call the methods from the front-end.

### 6.4.3 UI Prototype

A wireframe model of the main campaign browse page was also created during this sprint (Figure 23). It also implements the sidebar navigation and header layout. The design was created in Figma using components and builds on the initial graphic shown in the design section.

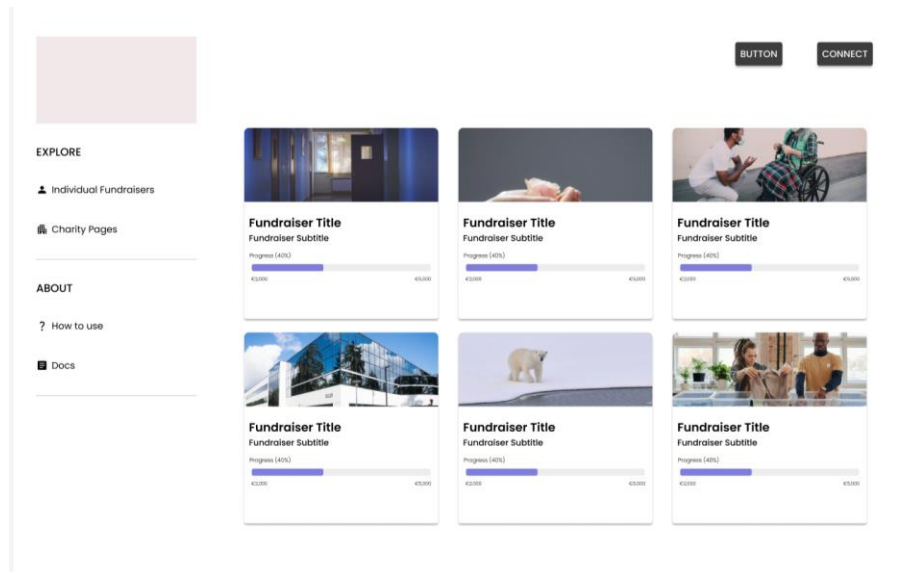


Figure 23 High-fidelity prototype in Figma

## 6.5 Sprint 2

### 6.5.1 Goal

During the second sprint, the Ganache local test network was set up locally for development purposes. This blockchain emulates other Ethereum-compatible networks locally. Transactions are quicker than public test networks and there is no real cost for transactions.

A React application was created that interacts with an instance of the string storage smart contract developed in sprint 1.

### 6.5.2 Development Blockchain

The Truffle framework was installed using the command 'npm install -g truffle'. A 'Truffle' folder was set up in the clients 'src' directory and the command 'Truffle Unbox' was run. This created folders for contracts, tests and other scripts. It also creates a truffle\_config.js file that is used to configure Truffle's settings, such as the chain it is connected to.

'Ganache' was also downloaded and installed from the Truffle suite website. Ganache emulates the Ethereum blockchain in a local environment. It provides numerous accounts, each with 100 Ether to test contracts with. It also provides a GUI where information about blockchain accounts, transactions and contracts can be viewed (Figure 24).

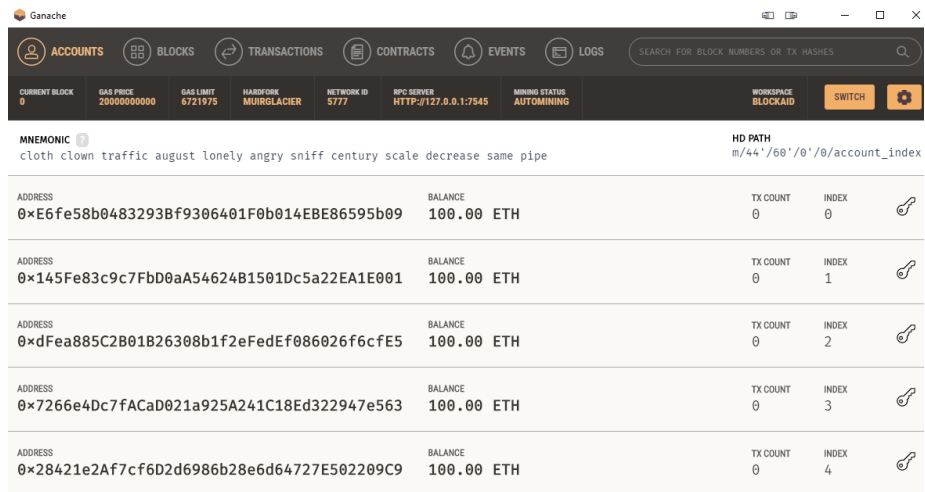


Figure 24 Ganache graphical user interface

This network was also added to the MetaMask networks in order to connect to the React application (Figure 25).

The screenshot shows the 'Add a network' form in MetaMask. It has a blue 'Add a network' button at the top right. Below it, there are five input fields: 'Network name' (containing 'Ganache'), 'New RPC URL' (containing 'http://127.0.0.1:7545'), 'Chain ID' (containing '1337'), 'Currency symbol' (containing 'ETH'), and 'Block explorer URL (Optional)' (empty). At the bottom, there are three buttons: 'Delete' (red), 'Cancel' (blue), and 'Save' (blue).

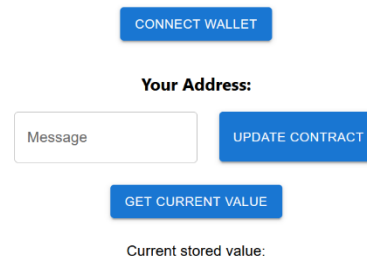
Figure 25 Adding Ganache network to MetaMask

### 6.5.3 React Application for Storage Contract

A React application was initialised using the 'npx create-react-app' script. Ethers.js and Material UI libraries were also installed through npm (Node Package Manager).

A single web page (Figure 26) was created with three buttons and a text input field:

# Hello World Ethereum DApp



*Figure 26 Basic React dapp interface*

Connecting the user's wallet gives the application access to basic data about the user's account address and balance and allows it to create transactions for the user to confirm and execute.

This is done by requesting a method to the browsers 'Ethereum' window (Figure 27), which is used by a number of browser-based wallets.

```
const connectWalletHandler = () => {
  if(window.ethereum) {
    window.ethereum.request({method: 'eth_requestAccounts'})
    .then(result => {
      accountChangedHandler(result[0]);
      setConnButtonText('Wallet Connected!')
    })
  }
  else {
    setErrorMessage('Need to install MetaMask!')
  }
}
```

*Figure 27 Requesting accounts from MetaMask*

Once connected, ethers uses the information to create provider, signer, and contract instances (Figure 28). The contract variable now uses the ABI from when the contract was compiled to access its methods.

```
const updateEthers = () => {
  console.log("Executing updateEthers")
  let tempProvider = new ethers.providers.Web3Provider(window.ethereum);
  setProvider(tempProvider);
  console.log("Provider set to: ", tempProvider);

  let tempSigner = tempProvider.getSigner();
  setSigner(tempSigner);
  console.log("Signer set to: ", tempSigner)

  let tempContract = new ethers.Contract(contractAddress, SimpleStore_abi,
    tempSigner);
  setContract(tempContract);
  console.log("Contract set to: ", tempContract)
}
```

*Figure 28 Instantiating Ethers provider, signer and contract*

This instance can then be used to call the get method on the contract (Figure 29) and display the value on the web page.

```
const getCurrentVal = async () => {  
  let val = await contract.get();  
  console.log(val);  
  setCurrentContractVal(val);  
}
```

Figure 29 Calling get function from React using Ethers

## 6.6 Sprint 3

### 6.6.1 Goal

While in previous sprints a smart contract was successfully deployed to the blockchain and interacted with from a react application, more functionality was required. The crowdfunding application requires currency, 'Ether', to be moved between user's accounts and smart contracts, so a contract called Bank.sol was developed.

### 6.6.2 'Bank' Smart Contract

The 'bank' smart contract was developed to allow users to deposit and withdraw Ether from a smart contract through the application. Unlike the initial contract, this was deployed to the Ganache local blockchain from the Truffle command line interface, meaning it can be written and debugged in Visual Studio Code. The code of this smart contract is shown in Figure 30.

```
1  //SPDX-License-Identifier: Unlicensed  
2  pragma solidity >0.4.23;  
3  
4  contract Bank{  
5      mapping(address => uint) public balances;  
6  
7      function deposit() public payable{  
8          balances[msg.sender] += msg.value;  
9      }  
10  
11     function withdraw(uint _amount) public{  
12         require(balances[msg.sender]>= _amount, "Not enough ether");  
13         balances[msg.sender] -= _amount;  
14         (bool sent,) = msg.sender.call{value: _amount}("Sent");  
15         require(sent, "failed to send ETH");  
16     }  
17  
18     function getBal() public view returns(uint){  
19         return address(this).balance;  
20     }  
21 }
```

Figure 30 Bank.sol smart contract

The balances variable is stored at the beginning of the contract. This is used to track the balance of each account that deposits and withdraws ether from the contract.



The deposit function adds the value of the transaction to the balance of the address that sent the transaction. The withdraw function takes in an amount to withdraw, and first checks if the balance of that address great enough to send that amount. The value is then subtracted from the user's balance and the ether is then sent to the user's account.

To read the balance of a particular user, the getBal() function returns the balance of the address that calls the function.

### 6.6.3 React Application for Bank Contract

Previously, the ethers.js library was used to interact with the smart contract. Another library, web3.js, is similarly popular based on npm downloads and was investigated during this sprint by rebuilding the previous 'Hello World' react application using it in place of ethers. A new web page was created for interacting with the Bank contract (Figure 31).

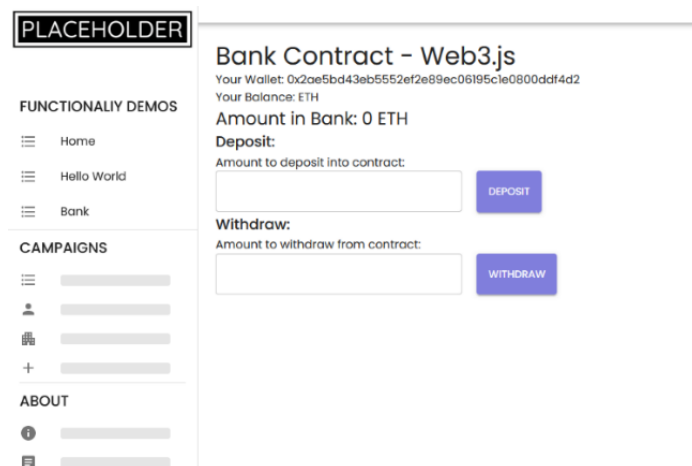


Figure 31 React page for interacting with Bank smart contract

A guide of the layout developed in the design phase was also implemented using the Material UI library, using placeholder components. On this page, a deposit function was written that accesses the 'deposit' function on the smart contract (Figure 32). Note that as well as calling the deposit function, the transaction must include a 'send' containing the amount of tokens to deposit.

```
const depositEther = async () => {
  const depVal = depositValue * 1000000000000000000;
  try {
    await contract.methods.deposit().send({from: account, value: depVal});
  } catch (err) { console.error(err) }
  finally {
    getContractBalance();
  }
};
```

Figure 32 Calling 'deposit' function from React using Web3.js

The user must enter the amount they would like to deposit or withdraw, and their browser wallet prompts them to confirm or reject the transaction (Figure 33).

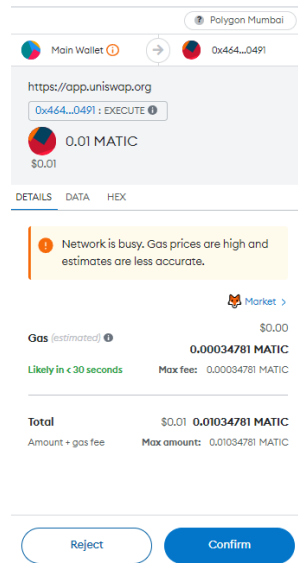


Figure 33 Confirming a transaction in MetaMask

Clicking the 'withdraw' button will access the withdraw function on the contract. It was noted that when operating with ether values, the amount needs to be converted to and from BigNumber object types, the format that the Ethereum virtual machine uses to carry out mathematical operations (Figure 34). Web3.js has built in methods to convert values to and from BigNumbers (web3.utils.toBN()).

```
const withdrawEther = async () => {
  try {
    const inputWei = web3.utils.toWei(withdrawValue);
    const amount = web3.utils.toBN(inputWei);
    await contract.methods.withdraw(amount).send({from: account});
  } catch (err) {
    console.error(err)
  }
  finally {
    getContractBalance();
  }
};
```

Figure 34 Calling 'withdraw' function from React using Web3.js

## 6.7 Sprint 4

### 6.7.1 Goal

This sprint focused on creating the foundations of the backend server using Express.js and MongoDB. The website layout was also fully implemented during this sprint.

### 6.7.2 Backend Development

The final application features an express.js API server. Its purpose is to interact with the MongoDB database, retrieving and writing data about crowdfunding campaigns and users. API calls are made from the React front-end.

A 'server' folder was created (Figure 35), and the node package initialised using the 'npm init' script. Express and mongoose packages were installed via npm. Folders for database object schemas, API functions, and routes were created.

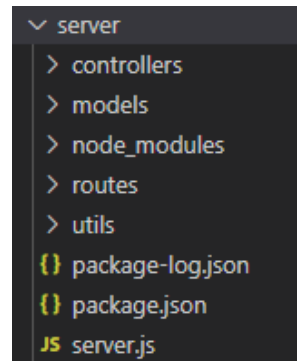


Figure 35 Server folder structure

A MongoDB project was set up on the MongoDB Atlas website. Connection to the database is done through MongoDB's 'mongoose' node package. The connect function is written in 'db.js' (Figure 36).

```
1  const mongoose = require('mongoose');
2
3  const connect = async () => {
4    let db = null;
5
6    try {
7      await mongoose.connect(process.env.DB_ATLAS_URL, {
8        useNewUrlParser: true,
9        useUnifiedTopology: true
10     });
11
12     console.log("Connected to database")
13     db = mongoose.connection;
14   }
15   catch(error) {
16     console.log(error);
17   }
18 };
19
20 module.exports = connect;
```

Figure 36 Connecting to MongoDB with Mongoose

Mongoose schemas were created for both the 'Campaign' (Figure 37) and 'User' (Figure 38) data types. These specify what parameters are included in the database.

```
3  const campaignSchema = Schema({
4    {
5      title: {
6        type: String
7      },
8      description: {
9        type: String
10     },
11     category: {
12       type: String
13     },
14     creator: {
15       type: String
16     },
17     goal: {
18       type: Number
19     },
20     image: {
21       type: String
22     },
23     address: {
24       type: String
25     }
26   }
27 });
```

Figure 37 Mongoose Campaign Schema

```
4  const userSchema = Schema ({
5    {
6      address: {
7        type: String,
8        required: [true],
9      },
10     first_name: {
11       type: String,
12       required: [true],
13     },
14     biography: {
15       type: String,
16     },
17     last_name: {
18       type: String,
19       required: [true],
20     },
21     email: {
22       type: String,
23       required: [true]
24     },
25     password: {
26       type: String,
27       required: [true]
28     },
29     image: {
30       type: String,
31     }
32   },
33   { timestamps: true }
```

Figure 38 Mongoose User Schema

Basic CRUD (create, read, update, delete) functions and routes were written for both users and campaigns (Figure 39).

```
66  const createData = (req, res) => {
67    let campaignData = req.body;
68    Campaign.create(campaignData)
69      .then((data) => {
70        console.log(`Fundraiser Created: ${data._id}`)
71        res.status(201).json(data);
72      })
73      .catch((err) => {
74        console.error("Error Message: " + err.message);
75        res.status(500).json(err);
76      });
77  };
78
79  const editData = (req, res) => {
80    let id = req.params.id;
81    let body = req.body;
82
83    Campaign.findByIdAndUpdate(id, body, {
84      new: true
85    })
86      .then((data) => {
87        if(data) {
88          console.log(`User edited campaign ${id}`)
89          res.status(200).json(data);
90        }
91        else {
```

Figure 39 Create and edit API functions

These functions were then tested using the Insomnia REST client (Figure 40)

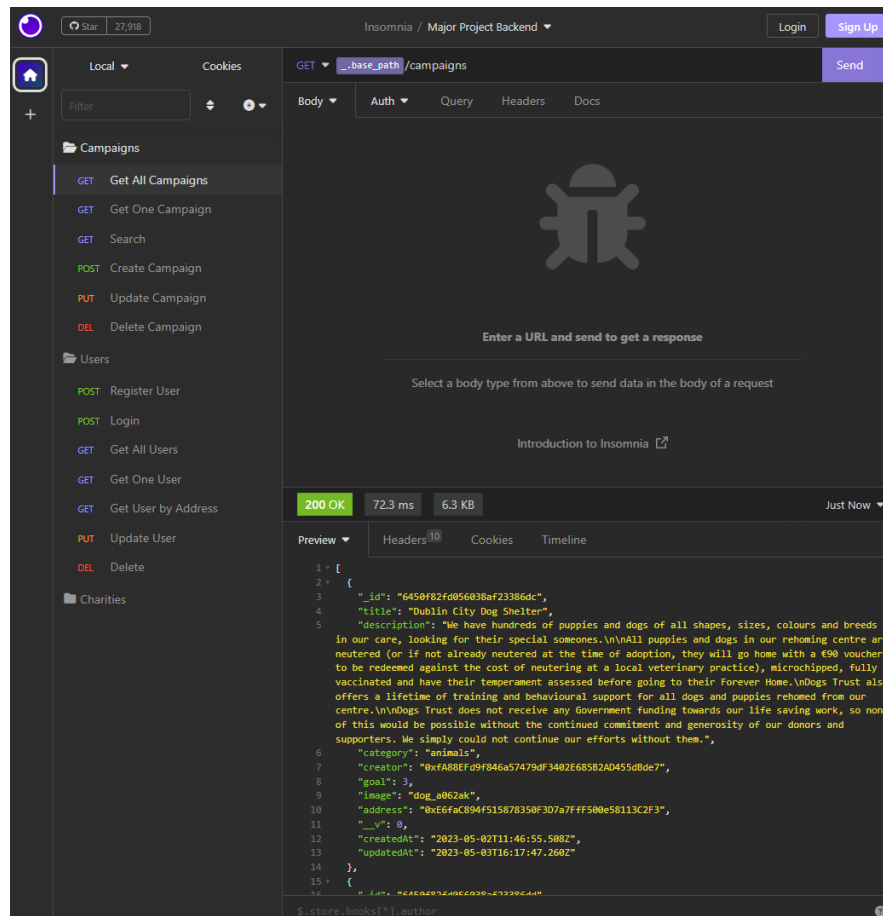


Figure 40 Testing API routes and functions in Insomnia client

### 6.7.3 Website layout

The layout for the side contains a sidebar for navigation, a header that handles user functionality such as login, settings and wallet connection.

A theme was created to be used across the application with MUI's createTheme component (Figure 41). This allows colour palettes and component styles to be stored in a variable and loaded into the main App.js file as a prop to the ThemeProvider component (Figure 42).

```
1 import { createTheme } from "@mui/material/styles";
2 import typography from "../Typography";
3
4 const mainTheme = createTheme({
5   direction: 'ltr',
6   breakpoints: {
7     values: {
8       xs: 0,
9       sm: 600,
10      md: 900,
11      lg: 1200,
12    },
13  },
14  palette: {
15    primary: {
16      main: '#377DFF',
17      light: '#ECF2FF',
18      dark: '#4570EA',
19    },
20    secondary: {
21      main: '#38BBEB',
22      light: '#E8F7FF',
23      dark: '#23afdb',
24    },
25    success: {
26      main: '#13DEB9',
27      light: '#E6FFFA',
28      dark: '#02b3a9',
29      contrastText: '#ffffff',
30    },
31    info: {
32      main: '#539BEE'
```

Figure 41 Creating theme using createTheme

```
function App() {
  const routing = useRoutes(Router);
  const theme = mainTheme;

  return (
    <>
      <WagmiConfig client={wagmiClient}>
        <ThemeProvider theme={theme}>
          <CssBaseline/>
```

Figure 42 Implementing theme in App.js

Layout.js stores the main layout for the site (Figure 43). It is made up of the sidebar, header, and a container for the page content. The outlet component displays the children, which are loaded in the router.

```
const Layout = () => {
  const [isSidebarOpen, setSidebarOpen] = useState(true);
  const [isMobileSidebarOpen, setMobileSidebarOpen] = useState(false);

  return (
    <MainWrapper
      className='mainwrapper'
    >
      <Sidebar isSidebarOpen={isSidebarOpen}
        isMobileSidebarOpen={isMobileSidebarOpen}
        onSidebarClose={() => setMobileSidebarOpen(false)} />
      <PageWrapper
        className="page-wrapper"
      >
        <Header toggleSidebar={() => setSidebarOpen(!isSidebarOpen)} tog
        <Container sx={{
          padding: "20px",
          maxWidth: "1200px",
        }}
        >
          <Box sx={{ minHeight: 'calc(100vh - 170px)' }}>
            <Outlet />
          </Box>
        </Container>
      </PageWrapper>
    </MainWrapper>
  );
};
```

Figure 43 Layout.js

To handle the routes in the application, the 'react-router-dom' library (*React Router Feature overview*) is implemented in the App.js file. Router.js contains an array of the routes used on the site. The components are loaded using React lazy (Figure 44), a function that allows components to be imported dynamically, improving performance (Ivanovic, 2022).

```
// Pages
const Home = (Loadable(lazy(() => import('./pages/main/Home'))));
const About = Loadable(lazy(() => import('./pages/main/About')));
const Start = Loadable(lazy(() => import('./pages/main/Start')));
```

Figure 44 Loading React components using Lazy

Once implemented, the site layout was prepared for displaying the required components (Figure 45)

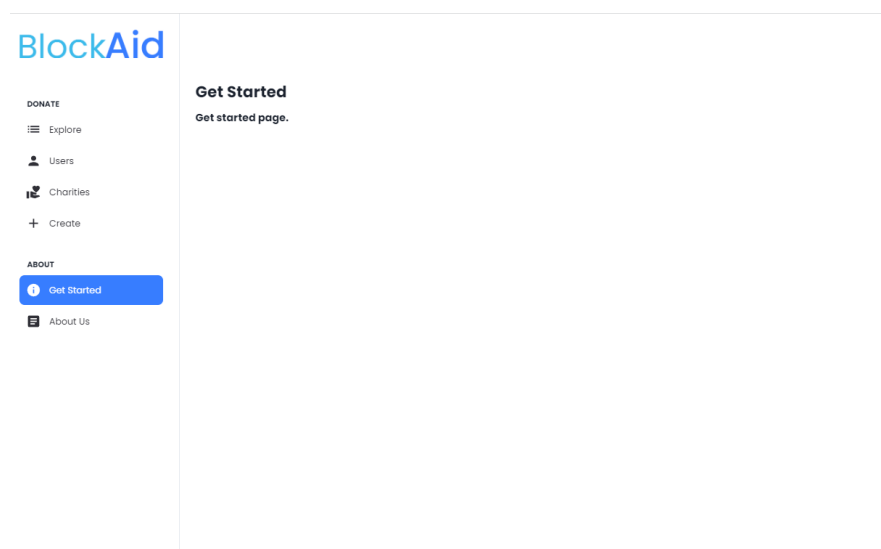


Figure 45 Completed site layout

## 6.8 Sprint 5

### 6.8.1 Goal

In sprint 5, a list of UI components required was made and first versions of these were created. Additionally, a new method of connecting user accounts was implemented, and the Crowdfunding smart contract was coded.

### 6.8.2 UI Components

The UI components required for the application were created in this sprint, informed by the design phase. These were built as React components, JavaScript functions that return HTML code.



Figure 46 Charity card



Figure 47 Campaign Card

Charity cards (Figure 46) and charity cards (Figure 47) are displayed on the home and explore pages, giving the user a brief preview of the fundraiser and its progress. These were created using MUI box components.

Donation cards to be displayed on the fundraiser pages were created, displaying the progress of the campaigns, and the donations of the user (Figure 48). As the charity pages do not have 'goals', no progress bars are on the donation window (Figure 49).

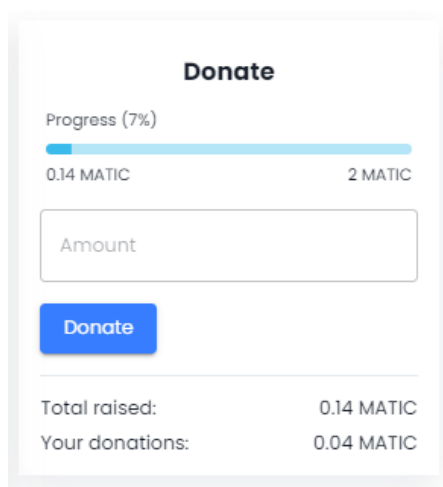


Figure 48 Campaign donation card

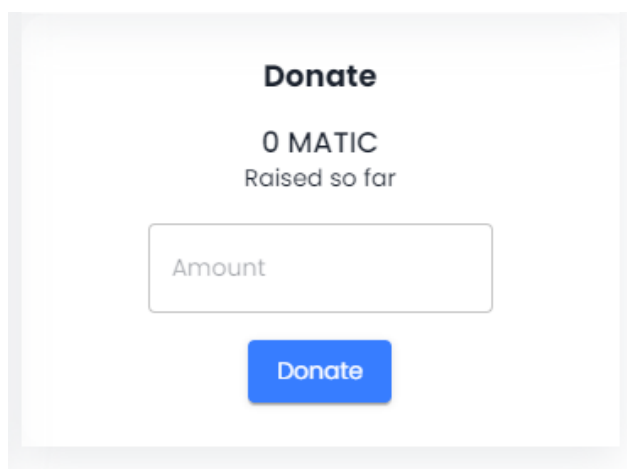


Figure 49 Charity donation card



The final component created in this phase was the campaign creation form. Material UI's Stepper component was utilized to show users just one step of the form at a time, with a horizontal stepper across the top to show the progress of the form (Figure 50).

1

Title

2

Category

3

Target

4

About

5

Image

6

Confirm

## Give your campaign a title

This is how users will find your campaign on our site.

Title

My new campaign!

Next

Figure 50 Campaign creation form

### 6.8.3 Wallet Connection Interface

Previously, connection to user accounts and the blockchain network was handled manually by web3.js and ethers. This sprint implemented the use of a library called 'wagmi'.

Wagmi is a connection of React hooks built with ethers that can be used across a React application (*Wagmi: React hooks for Ethereum*). This means that multiple components can use these hooks to access their current state, making it simpler to implement blockchain interactions.

Another package, 'Web3Modal' was also implemented. It provides an intuitive modal for users to connect wallets from a variety of providers (*Introduction - WalletConnect*), not just MetaMask as previously implemented (Figure 51).

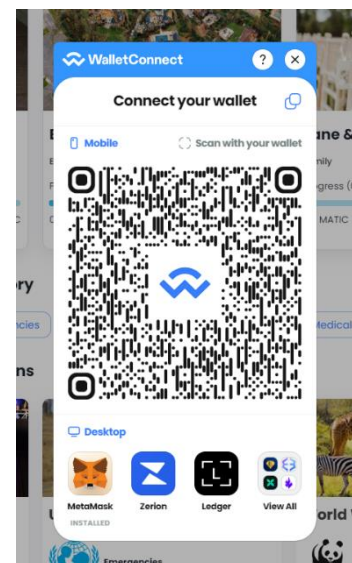


Figure 41 Web3Modal interface

To create and use the instance of 'wagmi' across the entire application, the App.js file is wrapped in a WagmiConfig provider. This is also where Web3Modal is instantiated (Figure 52).

```
15 //Web3
16 { const chains = [polygonMumbai]
17   const projectId = '0afb09d091a22717a29a967913a0a531'
18
19   const { provider } = configureChains(chains, [w3mProvider({projectId})])
20   const wagmiClient = createClient({
21     autoConnect: true,
22     connectors: w3mConnectors({ projectId, version: 1, chains}),
23     provider
24   })
25   const ethereumClient = new EthereumClient(wagmiClient, chains)
26
27   function App() {
28     const routing = useRoutes(Router);
29     const theme = mainTheme;
30
31     return (
32       <>
33         <WagmiConfig client={wagmiClient}>
34           <ThemeProvider theme={theme}>
35             <CssBaseline/>
36             <UserProvider>
37               {routing}
38             </UserProvider>
39           </ThemeProvider>
40         </WagmiConfig>
41         <Web3Modal
42           projectId={projectId}
43           ethereumClient={ethereumClient}
44           themeVariables={{
45             '--w3m-font-family': 'Poppins, sans-serif',
46             '--w3m-accent-color': '#377DFF',
47             '--w3m-text-medium-regular-weight': 'heavy',
48             '--w3m-text-medium-regular-size': '13px',
49             '--w3m-button-border-radius': '4px',
50           }}
51         />
52       </>
53     );
54   }
55 }
```

Figure 52 Wagmi and Web3Modal implementation in App.js

At this point in the development process, the application migrated from running on a local blockchain network to running on the public Polygon Mumbai Testnet. This is a testnet of the Polygon EVM blockchain, that uses a token called MATIC as its currency.

#### 6.8.4 Crowdfunding Smart Contract

The Crowdfunding.sol smart contract (Figure 53) is designed to be deployed by the user creating a crowdfunding campaign. It allows anyone to deposit ether into it and tracks the amount that they contribute. It also allows the deployer, 'controller', with withdraw ether from the campaign.

```

1  // SPDX-License-Identifier: Unlicensed
2  pragma solidity ^0.8.0;
3
4  contract Crowdfunding {
5      address payable public controller;
6      uint256 public totalDeposits;
7      mapping(address => uint256) public contributions;
8
9      constructor() {
10         controller = payable(msg.sender);
11     }
12
13     function deposit() public payable {
14         totalDeposits += msg.value;
15         contributions[msg.sender] += msg.value;
16     }
17
18     function withdraw() public {
19         require(msg.sender == controller, "Only the controller can withdraw funds");
20         uint256 balance = address(this).balance;
21         controller.transfer(balance);
22     }
23
24     function getTotalDeposits() public view returns (uint256) {
25         return totalDeposits;
26     }
27
28     function getController() public view returns (address) {
29         return controller;
30     }
31
32     function getContribution(address contributor) public view returns (uint256) {
33         return contributions[contributor];
34     }
35 }

```

*Figure 54 Crowdfunding.sol smart contract*

When deployed, the controller variable is set to the sender of the deploy transaction. This is used to control who can send a transaction to withdraw from the contract. The withdraw function requires that the address of the user that sent the transaction is the same as the controller, then transfers the contracts balance to the user's wallet.

The deposit function adds the value of ether sent with the address to the 'contributions' variable which stores the amount that each address has deposited so it can be retrieved for the web page.

Two additional functions are included that give data for the frontend to request. `getTotalDeposits()` gets the total amount deposited to the contract. This is so that even after the owner of the campaign withdraws currency from it, the total amount that the campaign has raised can still be displayed on the website, rather than the current balance of the transactions. There is an additional get function that takes in an address as a parameter and returns the amount that that address has contributed, which also can be displayed on the front end.

## 6.9 Sprint 6

### 6.9.1 Goal

Sprint 6 aimed to connect the existing React application to the back-end API. Several React hooks were also created for reading and writing blockchain information.

### 6.9.2 Connection to Back-End

The React application communicates with the back end through the server's API endpoints. These are for using CRUD functions on Campaigns, Charities and Users.

Axios is used to send HTTP requests from React components to the API. In Figure 55, an axios request is sent to get the specific campaign the user wants to visit.

```
useEffect(() => {
  axios.get(`/campaigns/${id}`)
    .then((response) => {
      setCampaign(response.data);
    })
    .catch((err) => {
      console.error(err);
      console.log(err.response.data);
    });
}, [id])
```

*Figure 55 Sending API request from React with axios*

The 'get' request is sent to the '/campaigns' endpoint with the ID of the campaign, which is read from the page URL. The state variable 'campaign' is then set to the HTTP response data. If there is an error with the request, it will output the error to the browser's console. The request is wrapped in a React useEffect hook, that executes when the 'id' dependency updates. This 'campaign' object is then used to render the campaign information on the page.

To create data, axios post requests are sent to the API. For example, Figure 56 shows how a new campaign is created with the data from the creation form.

```
const handleSubmit = () => {
  deployContractWithPromise()
    .then((address) => {
      console.log("Deployed at: " + address)
      axios.post('/campaigns', {
        title: formData.title,
        description: formData.description,
        category: formData.category,
        creator: account,
        goal: formData.goal,
        image: formData.image,
        address: address
      }, {
        headers: {
          "Authorization": `Bearer ${userData.token}`
        }
      })
        .then((response) => {
          setId(response.data._id);
          setCreated(true);
        })
        .catch((err) => console.error(err));
    })
    .catch((err) => {console.error(err)})
}
```

Figure 56 Creating data with axios request

First the contract is deployed using the `deployContractWithPromise` function. This returns an address, which is then added to the form data that the user filled in and sent to the `/campaigns` route in a 'post' request along with the user authentication token, as only registered users can create campaigns. This process overall is similar for campaigns, charities and users.

User-created campaigns and charity pages can be filtered and searched for on BlockAid. This is done by adding parameters to the API request. The user can search by title, category, and sort by the date created (Figure 57).

```
useEffect(() => {
  setLoading(true);
  axios.get(`/campaigns?title=${title}&category=${selectedCategory}&sort=${sortDate}`)
    .then((response) => {
      setCampaigns(response.data)
    })
    .catch((err) => {
      console.error(err)
      if(err.response.status === 404) {
        setCampaigns({});
      }
    })
    .finally(() => setLoading(false));
}, [title, selectedCategory, sortDate]);
```

Figure 57 Filtering campaigns in axios requests

This useEffect hook refreshes the data when any of the search parameters are modified. A search bar, category dropdown and sort dropdown allow users to set these parameters (Figure 58)

## Charity Organisations



Figure 58 Filter options in BlockAid

The backend was also updated in this sprint with a mongoose schema for charities (Figure 59). These include extra fields such as the charity's website, logo, and banner. There is also no 'goal' property as charity pages are constantly active and do not finish.

```
3  const charitySchema = Schema(  
4    {  
5      title: {  
6        type: String,  
7        required: [true]  
8      },  
9      description: {  
10       type: String,  
11       required: [true]  
12     },  
13     category: {  
14       type: String,  
15       required: [true]  
16     },  
17     creator: {  
18       type: String,  
19       required: [true]  
20     },  
21     bannerImage: {  
22       type: String  
23     },  
24     logoImage: {  
25       type: String  
26     },  
27     address: {  
28       type: String,  
29       required: [true],  
30       unique: true,  
31     },  
32     website: {  
33       type: String  
34     }  
35   },  
36   { timestamps: true }  
37 )
```

Figure 59 Mongoose Charity schema

Finally, user authentication was added to the backend in this sprint. When a user uses the login function and their password is successfully verified, they are sent back a login token generated by the jsonwebtoken JavaScript module (Figure 60).

```
const token = jwt.sign(
  {
    email: user.email,
    name: user.name,
    _id: user._id,
  },
  process.env.APP_KEY,
);
res.status(200).json({
  msg: "Login Successful",
  token,
  user: user
});
```

Figure 60 Creating authentication token with jsonwebtoken

When a user makes any request to the API, it goes through the authentication middleware (Figure 61).

```
app.use((req, res, next) => {
  if(req.headers?.authorization?.split(' ')[0] === 'Bearer'){
    console.log("User")
    jwt.verify(req.headers.authorization.split(' ')[1], process.env.APP_KEY, (err, decoded) => {
      console.log("Authorization Success")
      if(err) req.user = undefined;
      req.user = decoded;
      next();
    });
  }
  else {
    console.log("No Authorization")
    req.user = undefined;
    next();
  }
});
```

Figure 61 Authentication Middleware

This middleware checks if the request has included an “Authorization” header, and if so, it verifies that the token has been decoded. If it does, req.user is set to true. This is used to allow access to certain routes of the API.

### 6.9.3 Contract Read/Write Hooks

React hooks were created using the features provided by wagmi. useDeposit uses the useContractWrite, usePrepareContractWrite and useWaitForTransaction hooks (Figure 62).

```
2 function useDeposit(address, amount) {
3   const {
4     config,
5     error: prepareError,
6     isError: isPrepareError,
7   } = usePrepareContractWrite({
8     address: address,
9     abi: [
10      {
11        name: "deposit",
12        type: "function",
13        stateMutability: "payable",
14        inputs: [],
15        outputs: [],
16      },
17    ],
18    functionName: "deposit",
19    overrides: {
20      value: amount,
21    },
22  });
23
24  const { data, write: deposit, error, isError } = useContractWrite(config);
25
26  const { isLoading, isSuccess } = useWaitForTransaction({
27    hash: data?.hash,
28  });
29
30  return({
31    deposit,
32    isLoading,
33    isSuccess,
34    isError: isPrepareError || isError,
35    error: (prepareError || error)?.message
36  })
37 }
38
39 export default useDeposit;
```

Figure 62 useDeposit React hook

It takes the contract address and amount in as parameters and uses the usePrepareContractWrite hook to configure transaction data which is later used by useContractWrite. This includes passing it the contract's address, and the ABI of the contract so it knows the function to call. useWaitForTransaction returns feedback on the status of the transaction, which can be used to display UI elements, such as a progress circle.

This hook is implemented in the campaign donation card, where the deposit function is called and is passed the address for the campaign and the amount that the user has entered (Figure 63 and Figure 64)

```
38 // Deposit Function
39 const {
40   deposit,
41   isLoading: depositLoading,
42   isSuccess: depositSuccess,
43   error: depositError,
44 } = useDeposit(campaign.address, amountWei);
```

Figure 63 Implementing deposit hook

```
const handleDeposit = () => {
  deposit?.();
}
```

Figure 64 Calling deposit



## 6.10 Sprint 7

### 6.10.1 Goal

In sprint 7, the final blockchain-related element was implemented. This is the capability to deploy a smart contract from within the React application. Additionally, functional testing was carried out to measure the application's success against its functional requirements.

### 6.10.2 Deploying Smart Contract from React Application

The crowdfunding smart contract was written and compiled in Visual Studio Code using the Truffle suite. In order to interact with this contract, a 'useContractDeploy' react hook was written. This hook uses ethers' 'ContractFactory' to generate a transaction that the user sends to deploy the smart contract to the blockchain (Figure 65).

```
1  import { ethers } from "ethers";
2  import { useState, useEffect } from "react";
3  import { useSigner } from "wagmi";
4
5  import ContractABI from '../assets/abi/Crowdfunding.json'
6  const bytecode = '608060405234801561001057600080fd5b50336000806101000a81548173fffffffff
7
8  const useContractDeploy = () => {
9
10     const [data, setData] = useState({});
11     const {data: signer} = useSigner();
12
13     useEffect(() => {
14         setData(prevState => ({
15             ...prevState,
16             signer: signer
17         }));
18     }, [signer]);
19
20     const contractFactory = new ethers.ContractFactory(ContractABI, bytecode, signer);
21
22     const deployContract = async () => {
23         const contract = await contractFactory.deploy();
24         setData(prevState => ({
25             ...prevState,
26             deployedAt: contract.deployTransaction.hash
27         }));
28
29         await contract.deployTransaction.wait();
30
31         setData(prevState => ({
32             ...prevState,
33             deployedAt: contract.address
34         }));
35
36         return contract;
37     }
38
39     return({
40         deployContract,
41         data,
42         deployContractWithPromise: async () => {
43             const contract = await deployContract();
44             return contract.address;
45         }
46     })
47 }
48
49 export default useContractDeploy;
```

Figure 66 useContractDeploy hook

After creating the contract instance using the ABI and bytecode generated by compiling the contract, the `deployContract` asynchronous function uses the `deploy` method on the contract instance and waits for the transaction to complete. If successful, it returns the address of the contract so that it can be saved in the database. The hook returns an asynchronous function which can be called from a react component. This hook is implemented at the end of the campaign creation form, after the user has filled out the required details. A prompt in the user's wallet will appear asking to confirm the transaction.

### 6.10.3 Unit Testing

During this sprint, unit testing was also carried out using the 'jest' and 'truffle' testing libraries. This process is further detailed in the testing chapter.

## 6.11 Sprint 8

### 6.11.1 Goal

The eighth and final sprint focused on the usability of the application. Usability testing was conducted, and changes were made following recommendations.

### 6.11.2 User Testing

Usability testing was conducted with a sample of five participants. Participants were monitored while completing a set of tasks and asked to complete a follow-up survey. The details of the process and results are detailed in the testing chapter of this thesis.

### 6.11.3 Changes based on usability testing

Several changes to the user interface were made after reflecting on usability testing participant feedback. The details of these changes are described in the testing chapter of this thesis.

## 6.12 Conclusion

This chapter discussed the implementation of features outlined in the design phase of the project. Various JavaScript libraries for Ethereum blockchain interaction were compared through implementation, resulting in the creation of a MERN stack application allowing donations to user-created crowdfunding campaigns. The application connects to the Polygon Mumbai Testnet blockchain which functions as it would on any EVM-compatible mainnet chain. User-interface design which was prototyped in the design phase was implemented using React.js and the Material UI framework.

## 7 Testing

### 7.1 Introduction

This chapter outlines the testing that has been carried out for BlockAid. Testing was divided into functional testing and usability testing.

Functional testing tests the application against its functional requirements by providing inputs to the different functions and verifying that its output matches the expected output.

Usability testing tests whether the application is easy to use and learnable for the user.

### 7.2 Functional Testing

Function testing refers to the testing of software against its functional requirements. Its purpose is to test each of these requirements by providing an input, and verifying the output is as expected.

This chapter covers unit tests which were coded using testing frameworks, and manual functional testing, where the inputs were given through the application's user interface.

### 7.3 Unit Testing

Unit testing tests individual parts of the software independently to determine if they output the expected result. This involves writing test scripts that are given a certain input, and the expected output. If the result matches the expected output, the unit passes the test.

To test the various API routes, the jest JavaScript test framework was used. Three test suites were written for campaign, charity and user routes (Appendix A). These are stored in the 'tests' folder of the server. Each suite tested that each type of data could be successfully created, read, updated and deleted.

Unit testing of the Crowdfunding smart contract used Truffle's built-in testing framework. This allows smart contracts to be tested on a local blockchain network quickly and is free of cost.

The Crowdfunding.sol smart contract was compiled and deployed on the local Ganache blockchain using ``truffle compile`` and ``truffle migrate`` terminal commands. OpenZeppelin's 'test-helpers' library (*Test helpers*) was installed with npm in order to use the 'expectRevert' function, which expects a transaction that should be reverted to be reverted.

A test file (Appendix B) was developed. Three test cases were written to assess the functionality and efficacy of the three contract write functions critical to the applications operation.

### 7.3.1 Manual Functional Testing

Functional testing was carried out by the developer before user testing. Test cases were created for each of the functional requirements and were executed through the application front-end. The results can be found in (Appendix C). Table 2 shows an example of a test case.

Table 2 Test case example

Test ID		REGISTER-P		Priority		HIGH
Test Description		Register – Positive test case. Visit site and register new account with unused email and wallet address.				
Pre-Requisites		Ethereum Wallet (MetaMask) Connected to Polygon Mumbai Testnet		Post-Requirement		
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Launch website	http://localhost:3000	Home page	Home page	Pass	
2	Click ‘Log in’ button	Click	Login page	Login page	Pass	
3	Click ‘Register’ link text	Click	Register form	Register form	Pass	
4	Enter form fields	First name: John Last name: Smith Email: john@gmail.com Password: s3cure	None	None	Pass	
5	Click ‘Connect Wallet’ button	Click	Web3Modal popup	Web3Modal popup	Pass	
6	Select ‘MetaMask’	Click	MetaMask popup	MetaMask popup	Pass	
7	Confirm ‘Connect’ on MetaMask	Click	Connect button change to ‘Verify’ button	Connect button change to ‘Verify’ button	Pass	
8	Click verify button	Click	Verification dialog open	Verification dialog open	Pass	
9	Click ‘Verify Wallet’ button	Click	Button loading animation MetaMask popup	Button loading animation MetaMask popup	Pass	
10	Sign Transaction	Sign MetaMask Transaction	Wallet address displayed on form Register button enabled	Wallet address displayed on form Register button enabled	Pass	
11	Register	Click register button	User created component Redirect to dashboard	User created component Redirect to dashboard	Pass	

## 7.4 Usability Testing

In addition to functional testing by the developer, presenting the application to first-time users provided feedback from the target audience of the app.

This section discusses the usability testing that was conducted on the web application, along with its results and changes made following user feedback.

### 7.4.1 Objective

The objective of testing the usability of BlockAid is to assess the learnability and ease of use of the application for first-time users. Learnability in user experience design pertains to how intuitively first-time users can grasp how the product should be used. This is used to evaluate the processes within the application, as well as the user interface design. Feedback from participants was used to make changes and suggest future modifications.

### 7.4.2 Process and Materials

Five participants were recruited for usability testing. A moderated in-person test was conducted. The participants were asked to carry out several tasks (Appendix D) one at a time under the supervision of the developer. These tasks cover the functional requirements of the application. Participants were then asked to complete a short survey (Appendix E) on their overall experience. This included commenting on the features of the application, and how easy they found each task carried out.

In addition, this survey included a ten-question section which uses the System Usability Scale (SUS) to measure the overall usability of the website. The System Usability Scale aims to measure the usability of a product by asking the user how much they agree with a ten-question questionnaire (Brooke, 1996)

### 7.4.3 Results

The results gathered from the feedback form were collected into a spreadsheet using Google sheets (Appendix F). SUS scale values and averages of responses to quantitative questions and were also calculated in the spreadsheet in order to analyse the results.

#### 7.4.3.1 Participant backgrounds

Five participants took part in this usability testing and completed the follow-up survey. Forty percent of participants had experienced using a decentralised application in the past. Of these, one participant had used a decentralised crowdfunding platform. Sixty percent of all participants had donated to a fundraiser on a crowdfunding platform before, and twenty percent had created a fundraiser on a crowdfunding platform. When asked to rate their knowledge of blockchain technology on a scale of one to five, sixty percent responded between one and three.

This range allows the application to be seen from both an experienced and unexperienced perspective.

#### 7.4.3.2 Overall experience

Participants were first asked some general questions about their experience using BlockAid. The overall experience of the application was positive with an average rating of 3.8 out of 5.

#### 7.4.3.3 Ease-of-use

A series of questions about ease of use was presented next, asking participants to rate the ease at which they were able to complete the tasks given. The average rating across ease-of-use questions was 3.75. Notably, a low average score of 2.8 was recorded for the ease-of-use of editing user and campaign details (Figure 67 and Figure 68).

On a scale of 1-5, how easy was it to edit your user profile?  
5 responses

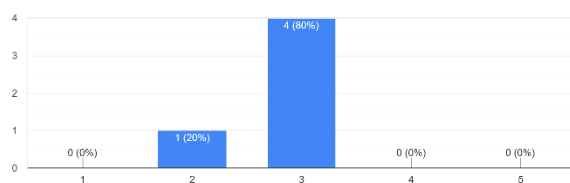


Figure 67 Results of edit user profile ease-of-use question

On a scale of 1-5, how easy was it to edit your campaign?  
5 responses

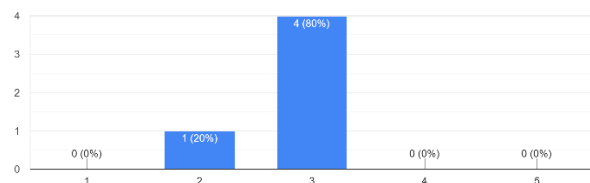


Figure 68 Results of edit campaign ease-of-use question

#### 7.4.3.4 System Usability Scale

The general usability of the site was measured with SUS using the questions asked in the survey. The results were calculated and resulted in an average score of 70.5. According to research, SUS scores above 68 are considered above average (Sauro).

#### 7.4.3.5 Blockchain Interaction and understanding

The blockchain section of the survey focused on the features of the website that are connected to the blockchain. Understanding of what MetaMask was used for based on reading the 'get started' guide was not strong in the participants that had not used blockchain technology before, with participants scoring their understanding 2.33 out of 5. However, in practice the users rated the ease of connecting MetaMask to the site and donating to a campaign 4.4 and 3.8 respectively.

When asked for comments about the blockchain elements of the application, multiple users stated they would like to see better instructions:

*"Some of the language in the 'get started' article was quite technical for me."*

*"I've used MetaMask a lot, but for people that haven't, there should be more information during the process rather than all in the help page."*

Those who answered ‘yes’ to having used decentralised applications before were asked questions comparing BlockAid to other decentralised applications. Both participants that had used these applications before stated they found the blockchain elements of the application intuitive. The wallet integration was rated very good by these users.

As a result of this feedback, additional instructions have been added to the registration form (Figure 69 and Figure 70).

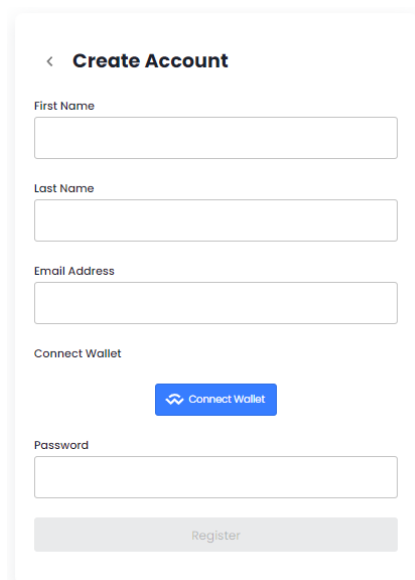
The image shows a mobile app registration screen titled 'Create Account'. It features a back arrow in the top left. The form includes input fields for 'First Name', 'Last Name', and 'Email Address'. Below these is a 'Connect Wallet' section with a blue button containing a wallet icon and the text 'Connect Wallet'. At the bottom is a 'Password' field and a grey 'Register' button.

Figure 69 Create account window before usability testing

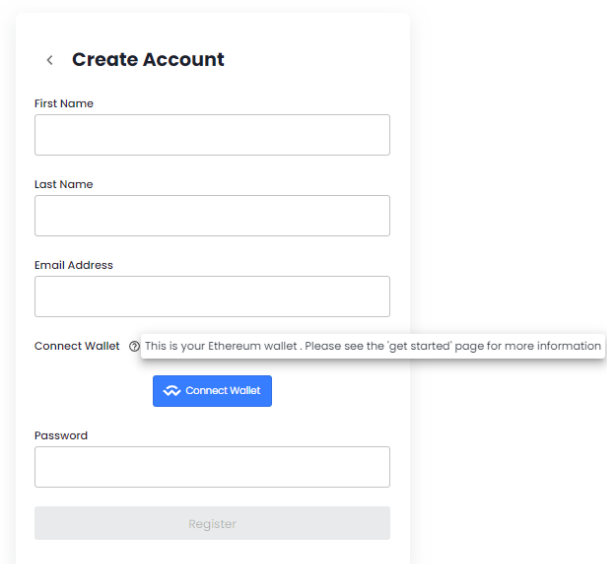
This image shows the same 'Create Account' form as Figure 69, but with an added tooltip. The tooltip is a light grey box with rounded corners, containing the text: 'This is your Ethereum wallet. Please see the 'get started' page for more information'. It points to the 'Connect Wallet' button.

Figure 70 Create account window after usability testing

#### 7.4.3.6 User Interface

The user interface was well received by participants, with 80% rating it 4 out of 5. On average, the ease of navigating through the site was rated 4.2. Three of the participants commented on the “simple” and “clean” design of the user interface

Visibility of the campaigns progress was well-rated, and the campaign creation form was intuitive.

One participant reported confusion relating to the positioning of the ‘Connect wallet’ and ‘Login’ buttons:

*“It might be confusing to have the wallet button and the account icon beside each other for people not used to it.”*

Differentiating between unregistered users with wallets connected from registered user accounts was not clear. This information has been updated in the ‘Get started’ page, to avoid user’s only intending to donate creating accounts.

One participant reported that the images appeared ‘squashed’ on some campaign preview cards. Appropriate image formatting has been implemented as a result of this (Figure 71 and Figure 72).

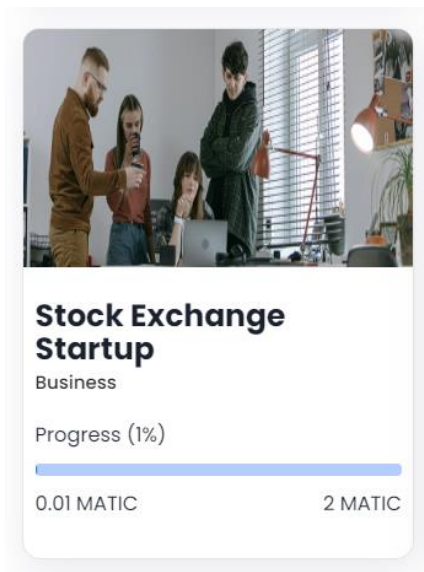


Figure 71 Card image before usability testing

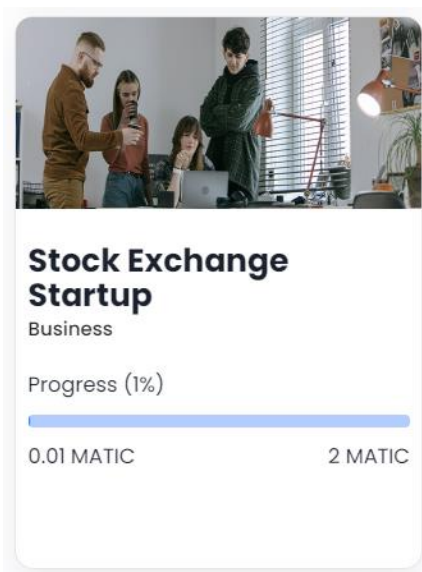


Figure 72 Card image after usability testing

Overall, the feedback on the user interface was positive, and the aim to have a minimal layout to maximise the ease of navigation was successful.

#### 7.4.3.7 Comments and Suggestions

When asked if participants had frustrations, it was clear that the concept of browser wallets, in this case MetaMask, may not initially be easy to grasp for first-time users. From the results however, it is apparent that the ease of connecting wallets and donating was rating highly, indicating that through using MetaMask, users become accustomed to it.

Participants were also asked about features they would have liked in the application. The following were suggested:

1. Donation receipts and tax information
2. Videos on campaign pages
3. Links on campaign pages
4. Social aspects (comments, messages)



## 7.5 Conclusion

Both functional and usability testing were carried out during the development of the application. The use of Truffle's built-in user testing framework allowed each of the smart contract functions to be verified as functional. This is of significant importance, as users' tokens need to be securely and reliably moved to and from the correct locations. Unit testing of the API routes and functions ensured that calls made from the front-end would receive the correct response.

Functional testing of the backend components from the React frontend ensured that the correct data was displayed based on the input of the user.

Usability testing was conducted to assess the websites usability and learnability. Participants successfully completed the tasks they were given. Participants responses were overall positive, and the application scored 70.5 on the System Usability Scale, which is above average. Feature suggestions were also made by participants, some of which have been implemented where practical.

## 8 Project Management

### 8.1 Introduction

This chapter describes how the project was managed through the development process. The phases of the project are outlined, highlighting issues that may have arisen. Also discussed is the use of project management tools and methodologies.

### 8.2 Project Phases

This section outlines the phases of this project. It discusses the aims and outcomes of each phase, in addition to issues that arose and how they were resolved.

#### 8.2.1 Proposal

During the proposal phase, the initial concept of the web application was established. The developer had previous experience as a user of decentralised applications, but not developing such applications. High level research was conducted on existing crowdfunding websites and decentralised applications. Node packages were briefly examined to gain an approximate understanding of the options that would be available in later phases. When examining the npm package website and browsing GitHub for decentralised application projects, Web3.js and Ethers.js were identified as commonly used JavaScript libraries for interacting with the blockchain.

#### 8.2.2 Requirements

The requirements phase was focused on existing applications and their capabilities. A website for users to donate cryptocurrencies to charity organisations was examined. A crowdfunding website for cryptocurrency business projects was also analysed, and a number of positive and negative points about the usability and features of the sites were made.

The needs of users and potential frustrations about the use of these two websites were used to formulate two user-personas, which are fictional characters used to aid in the design of the web application. A use case diagram was also developed to gain an approximate picture of how the application will be structured leading into the design phase.

The project's feasibility was also considered, and potential challenges or limitations were identified.

### 8.2.3 Design

The design of the application and user interface was completed in the design phase of the project.

As rudimentary MERN stack applications had been developed before, the primary challenge was the integration of blockchain interaction into the application. Research revealed the variety of ways this is implemented, and it was determined that these features should be accessed from the front-end and would remain physically separated to the back-end server and database. This limited the dependence on virtually two separate back-ends.

Solidity was chosen as the language to develop smart contracts due to its wide usage and documentation. This was a new programming language to learn, and a great deal of time in the design phase was spent understanding both the workings of the Ethereum Virtual Machine and the Solidity language. This was crucial to ensuring the resulting smart contract and the implementation of interacting with it was robust and secure.

As referenced in the previous phases, selection of technologies to carry out the blockchain-related tasks changed on multiple occasions based on the requirement of the application and the practicalities of implementing them.

The importance of a good user interface was established while analysing existing similar applications during the requirements phase, along with general strong user experience practices. As the target audience of the application may not be familiar with personal Ethereum wallets, the design must be intuitive along with provided clear instruction.

Due to the challenges this poses, a minimal layout was decided on, with clear differentiation between navigation actions and user actions using the sidebar and header. The layout also needed to have a familiar feel to other sites to maintain intuition.

### 8.2.4 Implementation

The implementation of the application was carried out in phases. Development focus was primarily on blockchain elements, as such an application had not been attempted by the developer previously.

Deciding on the libraries used to interact with the blockchain from the front-end was then initial priority, and changed several times based on the structure of the application and the requirements. The project originally used the Web3.js library, but this changed to using ethers.js for several reasons. The deployment of smart contracts through the browser was a more straightforward process with ethers. The use of 'wagmi' React hooks requires ethers as a dependency, and the instances of providers and signers used in wagmi can be used directly in ethers. This is the primary reason for the switch.

Initially, smart contracts had been planned to contain information about the campaign. This would have required each instance of a smart contract to be individually compiled with the solidity compiler before being deployed to the blockchain network. The Solidity Compiler 'solc' for JavaScript was deemed not viable for use in the application, due to difficulties with running it in a browser environment.

Instead, one master contract was created and compiled. The ABI and bytecode acquired from compiling it is used to deploy a separate instance of the contract on the blockchain for each fundraiser created. This change removed the need to compile the contract in the browser. Campaign details were instead stored in the 'campaign' objects in the MongoDB database. It also led to a consistent amount needed to pay transaction fees on the network, as the contract data was the same for each deployment.

### 8.2.5 Testing

During the testing phase of the project, both functional and usability testing was carried out. Functional testing consisted of unit testing the individual components in the backend and smart contract, as well as the developer carrying out the same actions from the frontend.

Truffle's built-in testing framework proved very useful for unit testing smart contracts as it could be done on a local blockchain, reducing cost and time requirements.

Usability testing was also conducted with a sample of five participants. There were clear limitations identified given the less traditional nature of the application.

Requiring individual participants to each download a browser extension and set up their wallets would have been a lengthy and potentially troublesome process. This procedure is unaffected by the performance of the application, thus it was decided that participants would use the testers computer, where the MetaMask wallet had been installed and set up, as well as being provided with an adequate amount of MATIC needed for the tasks participants were given.

Participants were given a number of tasks to complete, testing all of the functional requirements of the application and providing feedback about the usability of the site. Several participants had experience using other blockchain applications and were able to provide feedback and comparison.

Several changes to the application's interface were modified following feedback from usability testing participants.

## 8.3 SCRUM Methodology

The outline of sprints provided by the project requirements was used as a guide to structure the sprints. There were eight sprints in total. Each sprint improved on the product delivered previously.

In practice, some sprints did overlap. For example, sprints labelled as design sprints sometimes involved coding and vice versa. Reviews at the end of sprints proved beneficial for reflecting on progress and keeping accountability for the developer.

## 8.4 Project Management Tools

### 8.4.1 Trello

Tracking tasks during the development was done using Trello. During the requirements gathering stage, a section of the Trello board was created for each sprint, and the expected tasks were added as items to these sections. A kanban format was used, one column contained a list of tasks to be done, with items colour coded and labelled by sprint. When the task is in progress, it is moved to the 'doing' column and finally moved to the 'done' column when completed. There is an additional backlog column for important items that were time sensitive.

### 8.4.2 GitHub

As previously mentioned, git was utilised for version control and an overview of progress. For each change that was made, that change was noted as a message in the commit. After a section of tasks was completed and tested on the development branch, merges were carried out to bring this set of changes into the main branch, with a list of additions and updates to the code.

## 8.5 Reflection

### 8.5.1 Introduction

Overall, the project successfully met the expectations set out in the proposal stage. A decentralised crowdfunding web application was created and is functional on a public blockchain.

Attention to the usability of the application for a target user base that may not have experience using cryptocurrency payments or blockchain technology resulted in positive user feedback during usability testing of the application.

Knowledge of advanced React features was expanded. Understanding of and programming ability in the Solidity language was built from no prior experience using online resources and resulted in the development of a smart contract that meets the requirements of the application.

Learning new concepts, such as the basics of the Ethereum virtual machine was a challenging and daunting task, but establishing the correct scope during the research and requirements gathering phase aided this learning.

As anticipated, managing the backlog of features and tasks for this project required constant focus, and on occasion, tasks were not completed within the sprint timeframes.

### 8.5.2 Completing a large software development project

Working on a software development project of this scale had not previously been attempted. Early in the development process, the importance of planning and general project management techniques was evident. The application design phase was critical in mapping out the path needed to meet the requirements set out in the proposal.

### 8.5.3 Working with a supervisor

Undertaking this project with the guidance of a supervisor was particularly valuable. Having not attempted a process of this scale before, the insights and experience of a supervisor was crucial to the project's success, particularly in relation to project management and structuring the workflow.

### 8.5.4 Technical skills

Knowledge of all the technologies utilized in the development of this project has been advanced throughout the process. In particular the advanced use of built-in React.js features greatly furthered knowledge. For example, using more complex state-management functions, and building a responsive interface.

The area with the least experience was blockchain and smart contract development. Using Solidity at first was challenging at first, but research provided strong examples and along with the Solidity documentation allowed the finished smart contract to be robust. Additionally, the JavaScript libraries used to interact with blockchains were a new experience and required creative use of React's built-in features.

## 8.6 Conclusion

The project was separated into proposal, requirements, design, implementation and testing phases. Scrum methodology was used to plan and track a backlog tasks, divide tasks into sprints, and constantly reflect on progress throughout. Project management tools such as Trello and git were used in combination to maintain this methodology. This was effective overall but could be challenging to maintain at times. Using these tools, in addition to seeking advice from the project supervisor, helped structure the undertaking of such a large task.

Technical skills were greatly improved during the development of the application, with particular emphasis on programming smart contracts and interacting with the blockchain from the React front-end through the use of JavaScript libraries.

## 9 Business Opportunities

In order for BlockAid to be implemented as a business, there are some key aspects to be considered.

Although this application serves the charitable causes of its users, a revenue source should be implemented to support the running costs in production. The crowdfunding website 'GoFundMe' takes a transaction fee of 2.9% in addition to \$0.30 USD for each donation a fundraiser receives (*Learn about fees – GoFundMe help centre*). As a smart contract is used for payment, the donate function could be modified to take a fee, for example 1%, and send that to the developers Ethereum wallet (Figure 73).

```
function deposit() public payable {
    uint256 fee = msg.value / 100;
    totalDeposits += msg.value;
    contributions[msg.sender] += msg.value;
    address payable devWallet = payable(0x9bc72d38226C9433C8cd18bd758E732aB494b00C);
    devWallet.transfer(fee);
}
```

Figure 73 Example implementation of transaction fees

Additionally, many blockchain-based applications create their own cryptocurrency token for users to buy. In the first application examined in the requirements gathering sections released a token called \$VENT (Vent, 2022), and requires users to obtain a certain number of tokens to gain access to crowdfunding pages. ERC-20 tokens (a standard for Ethereum tokens) can have features such as fees that are transferred to the developer to support the project and can be traded on decentralized exchanges. This provides opportunities for additional revenue, but potentially at the cost of user satisfaction.

As this application involves the transfer of user's currency, the security implications would need to be very carefully considered. More research is required to assess the security and stability of the smart contract.

The smart contract could also provide a way for the application owner to recover funds in the event that the user loses access to their wallet. This would require creating an additional 'controller' for the contract, being the developer's wallet. However, this would conflict with the decentralised aspect of the app, as campaign owners would lose total control over their funds. Additionally, a more comprehensive user support system would need to be implemented.

Many different countries have strict regulations surrounding cryptocurrency and its use in donations. Extensive knowledge of these regulations would be needed to legally operate the application. These facts would need to be easily visible for users to ensure that they too follow regulations regarding tax implications. In addition, there are rules regarding transparency around cryptocurrencies depending on the country of the user and the business.

## 10 Conclusion

The aim of this project was to explore the integration of Ethereum blockchain technology into a MERN stack application, and to examine the available options for doing so. Through the development of this app, frameworks and libraries were identified that met the needs of the project and were intuitive to use for React developers.

### 10.1 Summary

A proposal was made for a decentralised application that uses Ethereum smart contracts. Research was undertaken to gain an understanding of blockchain technology, the Ethereum network and decentralised applications. This understanding was critical to begin developing smart contracts for the application.

Existing decentralised applications were analysed and informed the list of functional requirements for the application. User personas were created to design with the end-user in mind. Investigation into the JavaScript libraries available revealed several packages to be considered for use. Ethers.js was identified as the strongest library to implement blockchain interactions within the front-end application, and the Solidity programming language was chosen for smart contract development.

Information gathered from these phases informed the design process of both the application architecture and the user interface. The decision was made to emphasise the clarity of the user interface as to not overwhelm the user, considering the technical aspects of the application.

Scrum methodology was utilised to ensure adequate planning and reflection took place during the development, and to define deliverables. Sprint reviews provided an opportunity to assess and learn from completed work, and gain insight from.

A smart contract was developed using the Solidity smart contract. This smart contract allows users to donate cryptocurrency to crowdfunding campaigns. Only the owner of the campaign can withdraw cryptocurrency from the smart contract's balance. The smart contract also stores data about the campaigns progression which can be displayed on the front-end of the application.

The web application, 'BlockAid', was created using MongoDB, Express.js, React.js and Node.js. Users can register for an account to create and edit fundraising campaigns for charitable causes. Unregistered users can donate to campaigns and charity organisations by connecting their browser-based wallet to the website and entering an amount to donate. The front-end was built with the React front-end JavaScript library, combined with the Material UI React component library.

When users create a campaign, an instance of the crowdfunding smart contract is deployed to the Polygon Mumbai Testnet, where all transactions can be publicly viewed.

Functional testing verified the functionality of the application was robust. Usability testing was carried out to assess the ease-of-use of the application. The concept and design were well-received, with an above-average score of 70.5 System Usability Scale. User suggestions were considered and implemented following usability testing.



## 10.2 Future work

Further work on this project in the future would consist of a more business-focused approach to features.

Security of transferring live monetary assets on the blockchain would need to be robustly tested and meet standards. Legal implications regarding the use of cryptocurrency in a business environment should be further researched, and this information should be clearly communicated to the user.

The use of a wider range of cryptocurrencies should also be considered for live use. Users may hold fiat-tethered assets such as USDC, or other popular tokens.

## References

- Arya, J., Kumar, A., Singh, A. P., & KumarMishra, T. (2021). BLOCKCHAIN: BASICS, APPLICATIONS, CHALLENGES AND OPPORTUNITIES. <https://doi.org/10.13140/RG.2.2.33899.16160>
- Best, R. de. (2023, April 26). *Overall cryptocurrency market capitalization per week from July 2010 to April 2023*. Statista. Retrieved from <https://www.statista.com/statistics/730876/cryptocurrency-maket-value/>
- Blockchain Explorer - Bitcoin Tracker & More*. Blockchain.com. (2023). Retrieved April 4, 2023, from <https://www.blockchain.com/explorer>
- BNB chain: An ecosystem of blockchains*. BNB Chain Documentation. (n.d.). Retrieved February 22, 2023, from <https://docs.bnbchain.org/docs/overview/>
- Buterin, V. (2014). Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. *Ethereum.org*. Retrieved from [https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum\\_Whitepaper\\_-\\_Buterin\\_2014.pdf](https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf).
- Chainlink. (2022, October 28). *Solidity vs. Vyper: Which Smart contract language is right for me?* Chainlink Blog. Retrieved February 24, 2023, from <https://blog.chain.link/solidity-vs-vyper/#:~:text=Vyper%20is%20meant%20to%20be,general%2Dpurpose%20smart%20contract%20language>.
- Chainlist. (n.d.). Retrieved April 14, 2023, from <https://chainlist.org/>
- Charts - confirmed transactions per day*. Blockchain.com | Charts. (n.d.). Retrieved April 28, 2023, from <https://www.blockchain.com/explorer/charts/n-transactions>
- Christodoulou, P., Christodoulou, K., & Andreou, A. (2019). A Decentralized Application for Logistics: Using Blockchain in Real-World Applications. *Cyprus Review*, 30(2), 181–193.
- Crowdfunding market size, share & covid-19 impact analysis, by type (equity-based, debt-based, blockchain-based, and others), by end-user (startups, ngos, and individuals), and Regional Forecast, 2023-2030*. Crowdfunding Market Size, Share & Growth Analysis [2029]. (n.d.).

Retrieved May 5, 2023, from <https://www.fortunebusinessinsights.com/crowdfunding-market-107129>

Cryptopedia. (2021, December 23). *Real World examples of smart contracts*. Gemini. Retrieved from <https://www.gemini.com/cryptopedia/smart-contract-examples-smart-contract-use-cases>

Dam, R. F., & Siang, T. Y. (2022, April). *Personas – A Simple Introduction*. The Interaction Design Foundation. Retrieved April 2023, from <https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them>

Department of Health and Human Services. (2013, September 6). *System usability scale (SUS)*. Usability.gov. Retrieved May 4, 2023, from <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html#:~:text=Based%20on%20research%2C%20a%20SUS,to%20produce%20a%20percentile%20ranking.>

*Documentation - Ethers.js*. docs.ethers.org. (n.d.). Retrieved April 2023, from <https://docs.ethers.org/v5/>

*Ethereum IDE & Community*. Remix. (n.d.). Retrieved April 2023, from <https://remix-project.org/>

*Etherscan.io Blockchain Explorer*. Etherscan. (n.d.). Retrieved March 29, 2023, from <https://etherscan.io/>

*Express - Node.js web application framework*. Express. (n.d.). Retrieved April 2023, from <https://expressjs.com/>

Fortune Business Insights. (2023, February 1). With 14.5% CAGR, Crowdfunding Market Size to Surpass USD 3.62 Billion by 2030. Retrieved from <https://www.globenewswire.com/news-release/2023/02/01/2599133/0/en/With-14-5-CAGR-Crowdfunding-Market-Size-to-Surpass-USD-3-62-Billion-by-2030.html>.

*Getting started - Axios Docs*. Axios Docs. (n.d.). Retrieved April 2023, from <https://axios-http.com/docs/intro>

*Getting started*. React. (n.d.). Retrieved January 12, 2023, from <https://legacy.reactjs.org/docs/getting-started.html>

*Introduction - WalletConnect*. WalletConnect Docs. (n.d.). Retrieved April 2023, from <https://docs.walletconnect.com/2.0/web3modal/about>

Ivanovic, I. (2022, March 12). React lazy loading and performance [web log]. Retrieved from <https://retool.com/blog/react-lazy-loading-and-performance/>.

Kelly, L. (2023, March 22). *Bitcoin: A brief price history of the first cryptocurrency (updated 2023)*. INN. Retrieved March 29, 2023, from <https://investingnews.com/daily/tech-investing/blockchain-investing/bitcoin-price-history/>

*Learn about fees – gofundme help centre*. GoFundMe. (n.d.). Retrieved April 6, 2023, from <https://support.gofundme.com/hc/en-ie/articles/203604424-Learn-about-fees>

Modderman, G. (2022, June 5). Who accepts Bitcoin as payment? Retrieved from <https://cointelegraph.com/explained/who-accepts-bitcoin-as-payment>.

*Networks - Polygon Wiki*. Polygon Wiki. (n.d.). Retrieved April 2023, from <https://wiki.polygon.technology/docs/integrate/network/#testnets>

Nguyen, A. (2022, May 6). *Build a blockchain application using React and Solidity* (thesis).

*React Router Feature overview*. React Router. (n.d.). Retrieved April 2023, from <https://reactrouter.com/en/main/start/overview>

Sauro, J. (n.d.). *Measuring usability with the system usability scale (SUS)*. MeasuringU. Retrieved April 2023, from <https://measuringu.com/sus/>

Schwab Center for Financial Research. (2022, June 14). *Cryptocurrencies: What are they?* Schwab Brokerage. Retrieved February 14, 2023, from <https://www.schwab.com/learn/story/cryptocurrencies-what-are-they>

*The scrum framework poster*. Scrum.org. (n.d.). Retrieved April 2023, from <https://www.scrum.org/resources/scrum-framework-poster>

*Solidity 0.8.19 Documentation*. Solidity. (2023). Retrieved April 21, 2023, from

<https://docs.soliditylang.org/en/v0.8.19/>

*Test helpers*. OpenZeppelin Docs. (n.d.). Retrieved April 28, 2023, from

<https://docs.openzeppelin.com/test-helpers/0.5/>

Truffle Suite. (n.d.). *What is ganache?* Ganache | Overview - Truffle Suite. Retrieved March 2023,

from <https://trufflesuite.com/docs/ganache/>

Vent. (2022). *Token utility*. Vent. Retrieved from [https://vent-finance.gitbook.io/vent-](https://vent-finance.gitbook.io/vent-finance/ventonomics/token-utility)

[finance/ventonomics/token-utility](https://vent-finance.gitbook.io/vent-finance/ventonomics/token-utility)

*Wagmi: React hooks for Ethereum*. React Hooks for Ethereum –. (n.d.). Retrieved April 2023, from

<https://wagmi.sh/>

*What is blockchain technology?* IBM. (2022). Retrieved April 2, 2023, from

[https://www.ibm.com/topics/blockchain?mhsrc=ibmsearch\\_a&mhq=what+is+blockchain](https://www.ibm.com/topics/blockchain?mhsrc=ibmsearch_a&mhq=what+is+blockchain)

*What is mongodb?* MongoDB. (n.d.). Retrieved April 2023, from [https://www.mongodb.com/what-](https://www.mongodb.com/what-is-mongodb)

[is-mongodb](https://www.mongodb.com/what-is-mongodb)

*What is scrum methodology? & scrum project management*. Nimblework. (n.d.). Retrieved April

2023, from <https://www.nimblework.com/agile/scrum-methodology/>

## Appendices

### Appendix A – API Unit Testing Suites

Suites for unit testing Express API endpoints and functions.

<https://bit.ly/blockaid-api-tests>

### Appendix B – Smart Contract Testing Suite

Truffle testing suit for crowdfunding smart contract.

<https://bit.ly/blockaid-contract-tests>

### Appendix C – Functional Testing Results

Results of the functional testing carried out by the developer.

Test ID	LOGIN-P		Priority	HIGH		
Test Description	Login – Positive test case. Visit site and log in with valid credentials.					
Pre-Requisites	Account exists		Post-Requisite	None		
Testing steps:						
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Launch website	http://localhost:3000	Home page	Home page	Pass	
2	Click 'Log in' button	Click	Login page	Login page	Pass	
3	Enter valid login details and click login button	Email: george@example.com Password: password	Authenticated Redirect to dashboard	Authenticated Redirect to dashboard	Pass	

Test ID	LOGIN-N		Priority	HIGH		
Test Description	Login – Negative test case. Visit site and log in with invalid credentials.					
Pre-Requisites	None		Post-Requisite	None		
Testing steps:						
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Launch website	http://localhost:3000	Home page	Home page	Pass	
2	Click ‘Log in’ button	Click	Login page	Login page	Pass	
3	Enter invalid login credentials and click login button	Email: <a href="mailto:name@invalid.com">name@invalid.com</a> Password: 12345	Helper text: ‘Invalid email or password’	Helper text: ‘Invalid email or password’	Pass	
Test ID	REGISTER-P			Priority	HIGH	

Test Description		Register – Positive test case. Visit site and register new account with unused email and wallet address.				
Pre-Requisites		Ethereum Wallet (MetaMask) Connected to Polygon Mumbai Testnet		Post-Requisite		None
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Launch website	http://localhost:3000	Home page	Home page	Pass	
2	Click ‘Log in’ button	Click	Login page	Login page	Pass	
3	Click ‘Register’ link text	Click	Register form	Register form	Pass	
4	Enter form fields	First name: John Last name: Smith Email: john@gmail.com Password: s3cure	None	None	Pass	
5	Click ‘Connect Wallet’ button	Click	Web3Modal popup	Web3Modal popup	Pass	
6	Select ‘MetaMask’	Click	MetaMask popup	MetaMask popup	Pass	
7	Confirm ‘Connect’ on MetaMask	Click	Connect button change to ‘Verify’ button	Connect button change to ‘Verify’ button	Pass	
8	Click verify button	Click	Verification dialog open	Verification dialog open	Pass	
9	Click ‘Verify Wallet’ button	Click	Button loading animation MetaMask popup	Button loading animation MetaMask popup	Pass	
10	Sign Transaction	Sign MetaMask Transaction	Wallet address displayed on form Register button enabled	Wallet address displayed on form Register button enabled	Pass	
11	Register	Click register button	User created component Redirect to dashboard	User created component Redirect to dashboard	Pass	

Test ID		REGISTER-N			Priority		HIGH	
Test Description		Register – Negative test case. Visit site and register new account with already registered email address.						
Pre-Requisites		User already registered Ethereum Wallet (MetaMask) Connected to Polygon Mumbai Testnet			Post-Requisite		None	
	Action	Inputs	Expected Output	Actual Output	Action	Comments		
1	Launch website	http://localhost:3000	Home page	Home page	Pass			
2	Click ‘Log in’ button	Click	Login page	Login page	Pass			
3	Click ‘Register’ link text	Click	Register form	Register form	Pass			
4	Enter form fields	First name: John Last name: Smith Email: john@gmail.com Password: s3cure	None	None	Pass			
5	Click ‘Connect Wallet’ button	Click	Web3Modal popup	Web3Modal popup	Pass			
6	Select ‘MetaMask’	Click	MetaMask popup	MetaMask popup	Pass			
7	Confirm ‘Connect’ on MetaMask	Click	Connect button change to ‘Verify’ button	Connect button change to ‘Verify’ button	Pass			
8	Click verify button	Click	Verification dialog open	Verification dialog open	Pass			
9	Click ‘Verify Wallet’ button	Click	Button loading animation MetaMask popup	Button loading animation MetaMask popup	Pass			
10	Sign Transaction	Sign MetaMask Transaction	Wallet address displayed on form Register button enabled	Wallet address displayed on form Register button enabled	Pass			
11	Register	Click register button	Helper text: Email already in use	Helper text: Email already in use	Pass			



Test ID		CAMPAIGN-DEPLOY		Priority	HIGH	
Test Description		Deploy user campaign				
Pre-Requisites		Account held, wallet connected, MATIC for deploy fees		Post-Requisite	None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Go to create page	Click	Navigate /create Show select type	Navigate /create Show select type	Pass	
2	Select 'user' type	Click	Navigate /create/campaign	Navigate /create/campaign	Pass	
3	Fill in form Click next between steps	Title: New User Campaign Category: Emergencies Target: 10 Description: New description	Cycle through form steps	Cycle through form steps	Pass	
4	Upload image button	Click	Cloudinary popup	Cloudinary popup	Pass	
5	Upload image	Drag and drop image 'earthquake.png'	Image displayed on page	Image displayed on page	Pass	
6	Click submit	Click	Deploy confirmation popup	Deploy confirmation popup	Pass	
7	Click deploy and confirm in Wallet	Click	MetaMask popup Submit button changes to loading 'Campaign created' shows when transaction confirmed Redirect to campaign page	MetaMask popup Submit button changes to loading 'Campaign created' shows when transaction confirmed Redirect to campaign page	Pass	

Test ID		CHARITY-DEPLOY		Priority	HIGH	
Test Description		Deploy organization fundraiser				
Pre-Requisites		Account held, wallet connected, MATIC for deploy fees		Post-Requisite	None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Go to create page	Click	Navigate /create Show select type	Navigate /create Show select type	Pass	
2	Select 'user' type	Click	Navigate /create/charity	Navigate /create/charity	Pass	
3	Fill in form Click next between steps	Title: New charity Category: Emergencies Website: https://charity.org/ Description: New description	Cycle through form steps	Cycle through form steps	Pass	
4	Upload image button	Click	Cloudinary popup	Cloudinary popup	Pass	
5	Upload image and click next	Drag and drop image 'logo.png'	Logo Image displayed on page	Logo Image displayed on page	Pass	
6	Upload image and click next	Drag and drop image 'banner.png'	Banner Image displayed on page	Banner Image displayed on page	Pass	
7	Click submit	Click	Deploy confirmation popup	Deploy confirmation popup	Pass	
8	Click deploy and confirm in Wallet	Click	MetaMask popup Submit button changes to loading 'Campaign created' shows when transaction confirmed Redirect to charity page	MetaMask popup Submit button changes to loading 'Campaign created' shows when transaction confirmed Redirect to campaign page	Pass	

Test ID		CAMPAIGN-DONATE		Priority		HIGH	
Test Description		Donate to a user campaign					
Pre-Requisites		MetaMask Installed, MATIC Balance		Post-Requisite		None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments	
1	Navigate to user campaigns		Show user campaign grid	Show user campaign grid	Pass		
2	Select campaign	Click	Selected campaign details	Selected campaign details	Pass		
3	Connect wallet	Go through WalletConnect modal	Wallet connected, donate window visible	Wallet connected, donate window visible	Pass		
4	Enter amount and click donate	0.1	MetaMask popup	MetaMask popup	Pass		
5	Confirm transaction in MetaMask	Click	Progress bar increase, “Total raised” increased by 0.1, “Your Donations” increased by 0.1	Progress bar increase, “Total raised” increased by 0.1, “Your Donations” increased by 0.1	Pass		

Test ID		CHARITY-DONATE	Priority		HIGH	
Test Description		Donate to a charity				
Pre-Requisites		MetaMask Installed, MATIC Balance	Post-Requisite		None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Navigate to Charities		Show charities grid	Show charities grid	Pass	
2	Select charity	Click	Selected charity details	Selected charity details	Pass	
3	Connect wallet	Go through WalletConnect modal	Wallet connected, donate window visible	Wallet connected, donate window visible	Pass	
4	Enter amount and click donate	0.1	MetaMask popup	MetaMask popup	Pass	
5	Confirm transaction in MetaMask	Click	“Raised so far” increased by 0.1	“Raised so far” increased by 0.1	Pass	

Test ID	CAMPAIGN-WITHDRAW		Priority	HIGH		
Test Description	Withdraw balance from campaign smart contract – Positive test case (Owner wallet connected)					
Pre-Requisites	Account with existing campaign Campaign has balance > 0.1 MATIC Correct wallet connected		Post-Requisite	None		
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Visit dashboard	Navigate	Correct user dashboard	Correct user dashboard	Pass	
2	Go to withdraw page of first campaign	Click “Withdraw”	Correct campaign shown. Balance shown Total raised shown	Correct campaign shown. Balance shown Total raised shown	Pass	(If wallet connected is not that of user, page not displayed)
3	Withdraw balance	Click ‘Withdraw’ button	MetaMask popup	MetaMask popup	Pass	
4	Confirm Transaction	Click	Button displays ‘loading’ until transaction completed	Button displays ‘loading’ until transaction completed	Pass	
5	Transaction Confirmed	Wait	Withdraw button disabled Balance 0 MATIC Wallet balance increased by withdraw amount	Withdraw button disabled Balance 0 MATIC Wallet balance increased by withdraw amount	Pass	

Test ID	CHARITY-WITHDRAW		Priority	HIGH		
Test Description	Withdraw balance from charity smart contract – Positive test case (Owner wallet connected)					
Pre-Requisites	Account with existing charity Campaign has balance > 0.1 MATIC Correct wallet connected		Pre-Requisite	None		
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Visit dashboard	Navigate	Correct user dashboard	Correct user dashboard	Pass	
2	Go to withdraw page of first charity	Click “Withdraw”	Correct charity shown. Balance shown Total raised shown	Correct charity shown. Balance shown Total raised shown	Pass	(If wallet connected is not that of user, page not displayed)
3	Withdraw balance	Click ‘Withdraw’ button	MetaMask popup	MetaMask popup	Pass	
4	Confirm Transaction	Click	Button displays ‘loading’ until transaction completed	Button displays ‘loading’ until transaction completed	Pass	
5	Transaction Confirmed	Wait	Withdraw button disabled Balance 0 MATIC Wallet balance increased by withdraw amount	Withdraw button disabled Balance 0 MATIC Wallet balance increased by withdraw amount	Pass	

Test ID		EDIT-CAMPAIGN	Priority		MEDIUM	
Test Description		Change description of campaign				
Pre-Requisites		Existing account Existing campaign	Post-Requisite		None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Visit dashboard	Navigate	Correct user dashboard	Correct user dashboard	Pass	
2	Go to edit page of first campaign	Click “Edit”	Correct campaign details Edit form	Correct campaign details Edit form	Pass	
3	Change description and save	Replace description with “This is an updated description”	Helper text: “Saved!”	Helper text: “Saved!”	Pass	
4	Navigate back to dashboard	Click back arrow	Dashboard	Dashboard	Pass	
5	View updated campaign	Click “View” on campaign	Description: “This is an updated description”	Description: “This is an updated description”	Pass	

Test ID		EDIT-CHARITY	Priority		MEDIUM	
Test Description		Change description of charity				
Pre-Requisites		Existing account Existing charity	Post-Requisite		None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Visit dashboard	Navigate	Correct user dashboard	Correct user dashboard	Pass	
2	Go to edit page of first charity	Click “Edit”	Correct charity details Edit form	Correct charity details Edit form	Pass	
3	Change description and save	Replace description with “This is an updated description”	Helper text: “Saved!”	Helper text: “Saved!”	Pass	
4	Navigate back to dashboard	Click back arrow	Dashboard	Dashboard	Pass	
5	View updated charity	Click “View” on charity	Description: “This is an updated description”	Description: “This is an updated description”	Pass	



Test ID		EDIT-USER	Priority		MEDIUM	
Test Description		Change user biography				
Pre-Requisites		Existing account	Post-Requisite		None	
	Action	Inputs	Expected Output	Actual Output	Result	Comments
1	Visit settings	Navigate	Correct user dashboard	Correct user dashboard	Pass	
2	Go to edit page of first charity	Click “Edit”	Account settings page Details correct	Account settings page Details correct	Pass	
3	Change biography and save	Replace empty biography with: “This is my user biography”	Helper text: “Saved!”	Helper text: “Saved!”	Pass	
4	Visit profile	Navigate	User profile Biography reads: “This is my user biography”	User profile Biography reads: “This is my user biography”	Pass	

## Appendix D – Usability Testing Tasks

Tasks and verbal prompts given to usability testing participants.

Task no.	Goal	Prompt
1	Navigate to user campaigns	"You're on the home page of the BlockAid website. Read through the page and..."
2	Find specific campaign	"There's a user campaign raising money for walking club trip to the alps. Find it and visit the page"
3	Donate to campaign	"You have been given a wallet with a balance of 1 MATIC. Donate 0.1 MATIC to this campaign"
4	Find and donate to charity	"A charity called 'UNICEF' has set up a charity page on the site. Find it and donate another 0.2 MATIC to the charity."
5	Create account	"Shortly, you will be asked to create a crowdfunding campaign on BlockAid. First of all, make an account"
6	Log in to new account	"Great, now go ahead and log in to the account"
7	Edit user	"Before you create a campaign, you should personalise your profile page. Head to the settings and add the profile image on the desktop. You should also add a biography"
8	Create a campaign	"Every year you organize a fun run in your neighbourhood for the community association. This year, you're trying to increase donations by being able to accept cryptocurrency through BlockAid. Find the form and follow the instructions to create the campaign. You have a list of the details for this new fundraiser in front of you."
9	Edit campaign	"Now that you have created a campaign, you need to edit the description to the text you are being shown. Visit the user dashboard and edit the description of the campaign."
10	Log out and log in to existing account	"Use this login information to sign into BlockAid"
11	Withdraw	"Visit the dashboard and withdraw the balance of the campaign"

## Appendix E – Usability Testing Survey

The survey completed by participants following usability testing.

<https://bit.ly/blockaid-usability-testing-survey>

## Appendix F – Usability Testing Results

Google Sheets spreadsheet containing usability testing results and calculations.

<https://bit.ly/blockaid-usability-testing-results>