Direct To Consumer Online Voting System Utilising Blockchain Technology

David O'Loughran

B.Sc.(Hons) in Software Development

April 24, 2022

Final Year Project

Advised by: Joseph Corr
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract This project has been designed by me for the module Applied Project and Dissertation with the intention of creating a Semi Decentralised Online Voting System built with the latest in blockchain technology. The project should allow users to register and login to their dashboard where they can create and participate in in online lections and polls. The main purpose of this project is to utilise blockchain technology and smart contracts as an immutable database providing tamper free voting functionality to give users the comfort of knowing their votes are secure. The project will also utilise React on the front-end to create a simplistic and intuitive web application to improve the user's experience. This project will utilise the latest in blockchain development to improve user onboarding by removing the need for users the pay gas fees when interacting with a decentralised application while combining web 2 registration/logins with secure web 3 wallet functionality as a form of two factor authentication to improve account security. I plan to use this project to show case the developer skills I have learned throughout my study and expand my knowledge into the world of decentralised applications. I plan to improve my skills of adapting new and evolving technologies such as utilising smart contract to combine standard frameworks with blockchain based decentralised application's.

Authors My name is David O'Loughran and I am a fourth year student attending the Galway Mayo Institue of Technology. I have a keen interest in blockchain development and hope to aqcuire some of the neccesary skills to become a blockchain based developer.

Chapter 1

Introduction

1.1 Blockchain and Smart Contracts Explained

Blockchain is a decentralised public digital ledger used to record transactions that are distributed in a peer to-peer network using multiple nodes to secure and maintain the transactions in blocks [1]. These blocks contain data structures which link to its previous block through a hash code. This concept was first introduced by Satoshi Nakamoto when he wrote his white paper on Bitcoin in 2008 [2]. Others took inspiration from Satoshi Nakamoto such as Vitalik Buterin who seen the potential of applying smart contracts to a blockchain network. Instead of utilising blockchain technology as a form of performing monetary transactions it was proposed to add automated and executable agreements between two parties using programs stored on a blockchain network when predetermined conditions are met[3]. In simpler terms smart contracts allow us to create executable classes known as contracts on a blockchain network where each node in the network is aware of the contract removing the need to rely on third parties to enforce the integrity of applications.

1.2 Why create an Online Blockchain Voting System

Before I even started researching different topics for this project, I was already focused on blockchain development as it is the field of work, I am most interested in. With the recent events related to Covid 19 there appears to be a clear need for potential alternatives to voting systems during unprecedented times. Blockchain technology provides the necessary tools to ensure election data is kept secure and can't be manipulated by malicious third parties making it a prime example of how blockchain can be utilised to improve current systems that haven't been able convert to the digital age due to security concerns. This allows me to dive into the world of decentralised applications while not sacrificing the need for valid use case. I will talk more about the different benefits online voting systems built on the blockchain in the next chapter on the context of this project.

1.3 How I plan to make blockchain voting systems more attractive to users

1.3.1 Convenience

Blockchain based voting systems is not a new idea in terms of blockchain development with a wide variety of different companies offering it as a service. However, I noticed all these companies require you to contact their sales team in order to organise an online event which I felt isolates a large portion of the potential market. There was a lack of services available to the public where they could just easily register for an application and carry out their own elections or polls. This is when I decided I would like to create a direct-to-consumer alternative that allows anybody to register, create and participate in blockchain based voting systems as long as they have access to the internet.

1.3.2 Reduce Technical Expertise Needed

Since I already had a keen interest in Web 3 related products, I was aware of some of the key issues users faced that ultimately led to them losing interest in blockchain based applications. In order to interact with blockchains users normally have to pay gas fees whenever their interaction changes data on the blockchain. This meant users not only had to pay to utilise a decentralised applications features but also go through the lengthy process of funding their wallet with the desired cryptocurrency which in most cases would require them signup for an exchange and learn how to transfer funds to their wallet address. I knew this issue would invalidate the need for a direct-to-consumer online voting system secured by the blockchain, So I began researching alternative solutions. This is when I came across Biconomy who provide a lot of useful services for improving decentralised applications. Most importantly they provide a free service for enabling gasless transactions meaning users only have to sign transactions rather than pay for the interactions themselves[11]. This would allow me to drastically improve user on boarding without sacrificing the necessary security benifits that comes with blockchain technologies.

1.4 Chosen Tech Stack and why?

1.4.1 Why I chose the Polygon Blockchain Network

In order to perform a transaction on EVM compatible blockchain network gas fees have to be paid to miner who own nodes that are responsible for validating transactions. The most used blockchain network is Ethereum but they have large gas fees due to its low number of transactions per second. Polygon is a Layer 2 blockchain network that utilise Plasma [4] technology to run independent "child chains" increasing their transactions per second to approximately 65000. This vastly reduces the amount of time needed to confirm transactions and execute smart contract functions on the blockchain. This helps reduce the cost of creating and voting in elections/polls within our application as otherwise it would be far too expensive and slow to run elections/polls on a public blockchain.

1.4.2 Why I Chose Moralis

Moralis describe themselves as the Firebase for Web 3 development providing a fast and reliable managed backend for developing serverless blockchain based decentralised applications [5]. Moralis uses MongoDB for database storage which has gained massive popularity with developers due to it being open source and in a lot of cases faster than other alternatives such as MySQL. They have built in login and registration functionality allowing me to focus more attention on learning the intricacies of blockchain development. Since Moralis is built specifically for Web 3 development it provides many useful APIs for interacting with EVM compatible blockchains that other services have not yet implemented. They also provide you with your own nodes for interacting with the polygon blockchain which are the fasted and most reliable nodes available on the market allowing us to quickly render data from the blockchain improving the users experience which compliments the responsiveness of the react front end [8].

1.4.3 Why I Chose React

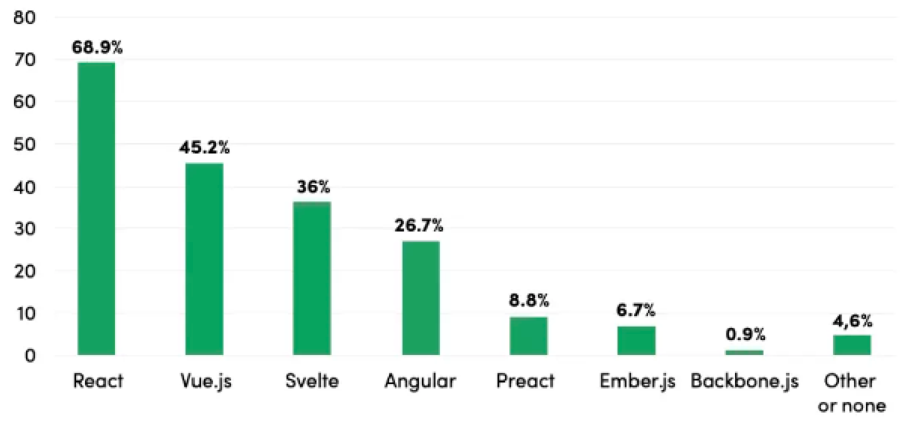


Figure 1.1: State of Frontend 2020 Survey

The reason I chose React for the front-end of my project is because it is currently the most popular and sought-after front-end language for single page web applications on the market [6]. I have very limited experience with React and would like to expand my knowledge on front-end frameworks. React provides the building tools to create intuitive and responsive single page web applications that are much faster at rendering pages than the standard combination of HTML, CSS and JavaScript. It utilises JSX to easily re-render webpage components when the state of the applications data is changed. This is crucial for this application as we will constantly be fetching and displaying data from the Polygon Blockchain. Initially I planned on using VueJS as my front-end framework but due to it still being in its infancy it was missing a lot of key libraries needed to create this specific application.

1.4.4 Why I Chose Truffle

I chose truffle because it is currently the most popular framework for compiling and deploying smart contracts to EVM compatible blockchains [7]. They also provide another service called Ganache for setting up your own personal blockchain in an interactive environment helping me gain a better understanding of how blockchains work. Since blockchain development is relatively new it's important to use a framework with good documentation with plenty of online resources which truffle has. It also utilises JavaScript and is capable of utilising Node modules which I already familiar with helping speed up the learning process.

1.5 System Requirements

- Login/Registration: Users should be able to register and login to the applications using an email and password
- Simultaneous Interactions: The application should support multiple users interacting with all features at the same time
- View data from blockchain: Users should be able retrieve live data from the blockchain
- Create Election/Polls: Users should be able to create elections and polls
- Transparency: Users should be able to validate the integrity of the voting functionality
- Tamper Proof: Malicious users should not be able to manipulate election/poll data
- 2FA Authentication: Users must be able to utilise their metamask wallet as two factor authentication
- User Sessions: Cookies should be used to maintain the users session while 2FA is used to regain access to their account

1.6 Chapters Explained

1.6.1 Context

In Chapter 2 I will be discussing the context of my project and how Blockchain based voting systems can disrupt the traditional forms of onsite voting. I will also outline the projects objectives and provide the GitHub repository that was used throughout the development process.

1.6.2 Methodology

In Chapter 3 I will be discussing the methodology I utilised throughout the development process along with how I utilised different forms of testing to ensure the application is robust.

1.6.3 Technical Review

In chapter 4 I will be discussing the different technologies that have been utilised during the development process of the project while giving context on why these technologies were chosen.

1.6.4 System Design

In Chapter 5 I will be discussing the project system architecture and design of the project. I will provide figures of front-end implementation along with some of the code I utilised to implement integral features of the application.

1.6.5 System Evaluation

In Chapter 6 I will be analysing the implementation of my project. Outlining the different techniques, I used to ensure robustness throughout the project. Explain some of the choices made to reduce gas consumption with the smart contract functionality. I will also discuss the limitations and potential opportunities I have discovered with the technologies used.

1.6.6 Conclusion

In Chapter 7 I will be going over the different things I learned throughout the duration of this project. I will explain how I met or did not meet my objectives and what I would of done differently in this project.

1.6.7 Github

GitHub was utilised throughout the project to keep track of the projects versions making it easy to retrace my steps in the case any issues appeared in my project. This repository contains two branches. One containing the blockchain based voting application and another containing all materials related to the dissertation. The repository also contains a README file giving a brief explanation of the project and how to run it on your own machine.The GitHub repository can be found here.

Chapter 2

Context

2.1 Current Voting Systems

Most current voting systems are setup, controlled, and audited using centralised trusted third parties which leads to multiple points of failure. The most popular form of voting today is through Paper Ballot Systems which pose the potential risk of third parties tampering with results. These systems prove to have many disadvantages. A lot of money tends to be wasted due to the costs involved such the setup, printing, transportation, and all the personnel labour required to carry out an election [9]. There are also notable issues with voter turnout in younger demographics as young people spend more time in the digital space and don't like the inconvenience of voting in person.

2.2 Cons of Onsite-Voting

2.2.1 Accessibility

Voter turnout can be a major issue when carrying out onsite voting as not all participants may be able to attend the available voting centres. This can be due to the likes transportation issues where some participants may not be able to organise transport to the voting location [9]. Participants may be busy and are unable to cancel plans on the days the voting is taking place.

2.2.2 Cost

Onsite election or polling stations require a lot of manpower and resources to be carried out with the organisers often having to rent locations for the event. Security also has to be organised to maintain control and ensure there is no malpractice going on at the event. People have to be paid to carry out the counting of votes[9].

2.2.3 Third Party Interference / Inaccuracies

Since onsite elections and polling stations require people to manually handle the counting process any person with ill-intent may attempt to skew the vote based on their own preference. Humans are also prone to making involuntary errors and may accidentally miscount the votes resulting in inaccurate outcomes. Covid-19 highlighted these issues as mail in voting was the recommended alternative which has the potential to be intercepted by people with intentions to damage the integrity of the election.

2.2.4 Time Consumption

Onsite voting can take up a large chunk of time between organising the event. Manually counting the votes can take time and sometimes recounts must also be carried out which can prolong the wait time for results.

2.3 How does Online Voting tackle these issues

2.3.1 Accessibility

Online election can give voters the ability to participate in elections and polling from anywhere they want and even from the comfort of there own home. As long as they have access to the web they will be able to cast their vote without the need to potential cancel any plans. The voting process can be extended as organisers have much more freedom over when the voting process can take place.

2.3.2 Cost

The cost of elections is drastically reduced as organisers no longer have to pay security to watch over the voting process. The counting process is handled automatically so staff won't need to be hired to manually count the votes. Organisers don't have to rent out a venue for the duration of the event.

2.3.3 Third Party Interference / Inaccuracies

Blockchain based smart contracts can be utilised to ensure the voting process can't be manipulated by third parties. The immutability of the blockchain ensures integrity that all votes are accounted for. All interactions with the voting process can easily be audited with speed and high precision as all transaction are recorded by the blockchain.

2.3.4 Time Consumption

Time spent setting up the election or poll is drastically reduced as organisers don't have to waste time finding an available venue, security, and staff to work the event. Since voters can vote online, they no longer must wait in a lengthy queue just to spend 1 minute casting their vote. The counting process is handled programmatically so results can be calculated almost instantaneously.

2.4 Project Objectives

- Improve consumer accessibility to participating in elections and polls
- Decrease technical expertise needed for interacting with blockchain based Decentralised applications.
- Secure voting functionality by utilising blockchain based smart contracts
- Improve user security by implementing identity verification to cast votes
- Improve user account security through Web 3 wallet provider
- Reduce the cost of carrying out elections and polls
- Gain a better understanding of blockchain development
- Create a simplistic and intuitive user interface using React

2.4.1 Github

GitHub was utilised throughout the project to keep track of the projects versions making it easy to retrace my steps in the case any issues appeared in my project. This repository contains two branches. One containing the blockchain based voting application and another containing all materials related to the dissertation. The repository also contains a README file giving a brief explanation of the project and how to run it on your own machine.The GitHub repository can be found here.

Chapter 3

Methodology

3.1 Methodology Overview

Throughout the course of the development process, I utilised the Agile methodology by dividing up my project into sprints. I utilised Test Driven Development when writing my smart contracts by writing basic functions and testing them before increasing their functionality. After I was happy with the functions, I would refactor them to try optimising the performance and reduce gas consumption. I had regular meetings with my project supervisor to help keep myself on the right track and ensure resources were not unnecessarily wasted.

3.1.1 Agile Methodology

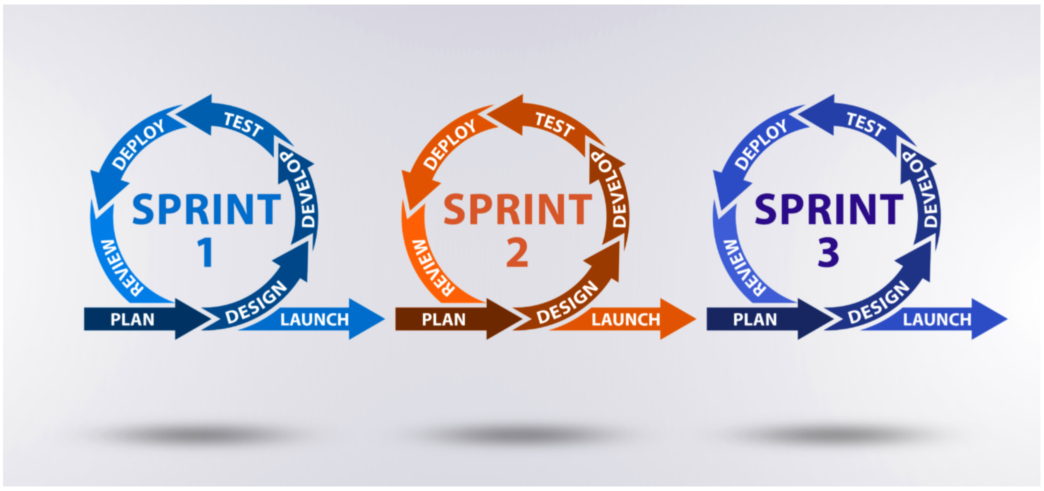


Figure 3.1: Agile Methodology

In software development the Agile approach involves breaking up the project into several different phases while continuously building up the project. The Agile approach prioritises "Individuals and interactions over processes and tools" as described by Wrike as project management service provider [10]. An iterative approach is utilised through continuous testing allowing for quick and responsive changes as the project is being developed. Agile allows you to deliver working applications frequently and iteratively building and improving the existing software. Teams will meet regularly to discuss the current progress and how to adjust accordingly to improve efficiency throughout the rest of the development process.

3.1.2 How Agile was used in this project ?

The agile approach was crucial for the development of this project specifically as I was utilising multiple technologies that I had zero experience with and would have re-evaluate the projects scope on multiple separate occasions. The iterative approach to creating working software from early on in the project's life cycle was key in gaining a better understanding of blockchain development. I initially started by creating a basic implementation of my project in a more familiar front-end technology HTML. Within the first couple weeks of starting the project I had created a very basic voting system with predetermined candidates and limited functionality. After familiarising myself with some of the technologies I refactored my front-end to React and iteratively built on the functionality of my application.

3.1.3 Time Management

I utilised a Gannt chart to split up my project into different sprints of varying length which could easily be adjusted throughout the development life cycle. I had regular meetings with my project supervisor through out however due to personal and covid related issues a good few of these meetings were missed and had to be rescheduled accordingly. These meetings helped motivate me at times where I was especially busy with other projects. Keeping through to the agile approach these sprints were adjusted regularly as in some cases there were very little resources on my chosen technologies due to their recent nature.

3.1.4 Applying Test Driven Development

Although I utilised Test Driven development throughout writing my blockchain based smart contracts it was not the standard approach of writing automated tests. I utilised the Remix IDE provided by Ethereum the founders of solidity the language used to write EVM compatible smart contracts. This IDE provided an interactive environment where you could quickly and easily modify the inputs and execute functions while working on them. This allows me to focus on testing my smart contracts as I was developing them by meeting the barebone requirements before iteratively adding more and more functionality to the contract's functions. This was especially useful as I could adjust the inputs on the fly testing my contracts without having to spend a lot of time writing different test cases.

3.1.5 Github

GitHub was utilised throughout the project to keep track of the projects versions making it easy to retrace my steps in the case any issues appeared in my project. This repository contains two branches. One containing the blockchain based voting application and another containing all materials related to the dissertation. The repository also contains a README file giving a brief explanation of the project and how to run it on your own machine.The GitHub repository can be found here.

Chapter 4

Technology Review

4.1 Moralis

Figure 4.1: Moralis Logo

4.1.1 Overview

While originally planning to use AWS and PostgreSQL, after researching potential Web3 alternatives I came across the free services of Moralis who describe themselves as the Firebase for Web3 Development helping bridge the gap between web 2.0 and web 3. They are an infrastructure provider that provides a managed backend for blockchain based applications packaged with MongoDB for quick and easy database storage. MongoDB is the perfect fit as they format data using JSON which is also the same data format that we receive when fetching data from the Polygon Blockchain allowing us to stay consistent with data formatting throughout the development process.

4.1.2 Web Hosting

Moralis also provides free web-hosting however there was some issues with routing to different pages inside the app and with Moralis being a new and evolving technology there was very little resources for my specific issue. When testing on another webhosting service called Netlify there was no issues and since they are both free I decided to stick with what worked and not get hung up trying to find my own solution.

4.1.3 Speedy Nodes

In order to deploy and interact with smart contracts on the Polygon Blockchain you need access to your own node in order send information back and forth between other nodes in the blockchain to verify and check that transactions are valid.

Moralis also provide their own nodes called Speedy Nodes which function the same as standard nodes but come with “exceptional speed, reliability and are fully integrated with all Moralis services” [8] . This is extremely useful for monitoring potential malicious activity as we can easily access the nodes transaction history and logs through the Moralis dashboard which is crucial for upholding integrity for an online voting application.

4.2 Solidity

Figure 4.2: Solidity Logo

Solidity is a high-level object-oriented programming language created by Vitalik Buterin the founder of Ethereum. It is utilised to write smart contracts for EVM compatible blockchains such as Ethereum, Avalanche and in this case Polygon. Solidity is Turing complete meaning in theory it can calculate anything that is computable considering there is enough memory. This made the transition to Solidity much easier while migrating from more popular languages like python and JavaScript while also ensuring the ability to implement and deploy sophisticated logic to the blockchain which was necessary for my election contract. However, there are some noteworthy limitations to Solidity such as issues with Arithmetic Precision [13] as solidity does not support floating point numbers. This became an issue when dealing with dates for starting and ending elections as solidity's timestamps are in epoch. This meant all dates and times had to be parsed to integers before interacting with the smart contracts.

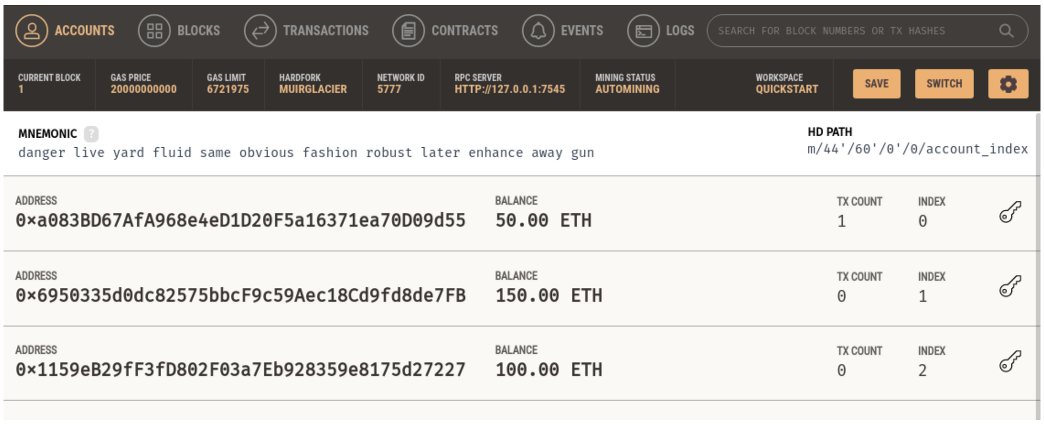
4.3 Truffle

Figure 4.3: Moralis Logo

4.3.1 Truffle Framework

Truffle is currently the most popular development framework for compiling, deploying and testing smart contracts for EVM compatible blockchains. In order to execute functions in smart contracts with JavaScript we need to provide the contracts Application Binary Interface(ABI). An ABI is an interface which is used for encoding/decoding solidity smart contract calls. Truffle handles all this for us by creating Artifacts which contain the necessary JSON data including the ABI and placing it inside a contracts folder in the root directory of your project. Truffle also supports scriptable deployment of contracts giving you more control over the migration process and also allowing you to deploy multiple contracts in order at the same time. Truffle was especially useful for this project as it allows you to utilise other files and node modules in the directory making it much easier to customize specific variables. For example, truffle comes with a require keyword primarily used for accessing the solidity contract into your migration and deployment scripts. This opens doors for scaling to other EVM compatible blockchain networks as you can have a list of different networks that can be accessed through the command line when migrating/deploying contracts. This means nothing has to be changed to switch between main and test nets as it can all be controlled through the command line.

4.3.2 Ganache



CURRENT BLOCK 1 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MUIRGLACIER | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:7545 | MINING STATUS AUTOMINING | WORKSPACE QUICKSTART

MNEMONIC
danger live yard fluid same obvious fashion robust later enhance away gun

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×a083BD67AfA968e4eD1D20F5a16371ea70D09d55	50.00 ETH	1	0
0×6950335d0dc82575bbcF9c59Aec18Cd9fd8de7FB	150.00 ETH	0	1
0×1159eB29fF3fD802F03a7Eb928359e8175d27227	100.00 ETH	0	2

Figure 4.4: Ganache Interface

Allows you to create and run an interactive personal Ethereum blockchain locally on your device which was extremely useful during the earlier stages of development for getting a better grasp on how the blockchain operates under the hood. It gave access to 10 different wallet addresses which can be easily synced with the metamask extension used to make standard browsers like google chrome compatible with web 3 features allowing us to interact with blockchain smart contracts.

4.4 Remix IDE

Remix is an online IDE created by Ethereum the creators of solidity themselves. Remix provides a high-quality online IDE fully capable of compiling and deploying contracts to the blockchain. Remix allows you to interactively test the functionality of your contracts functions as you develop them. You can interact with any contract function after compiling and deploying the contract to their onsite Ethereum virtual machine as well as set up automated testing of solidity contracts. This is where I would develop and test the functionality of all contracts and functions before officially implementing them within project itself as the interactive environment was perfect for quickly checking if a function has been implemented as intended.

4.5 Biconomy

Biconomy provides powerful SKD/APIs allowing you to customize your applications transaction process which plays a major part in creating seamless interactions between the applications smart contracts and the end-users [12]. In a traditional web3 set up a user may have to purchase the likes of Ethereum on an exchange before transferring it to their metamask wallet. The user will then also have to use this Ethereum when performing any contract functions that write data to the blockchain. This process can take days if the exchange requires KYC checks and drastically reduces user on-boarding for web 3 applications. Biconomy solves many current issues that negatively impact the users experience with multiple different features. Most importantly for our application they allow you to enable gasless transactions, so users only must sign transactions rather than paying the gas fees themselves[11]. In order to enable gasless transactions, you must make your contract EIP2771 compatible to allow meta transactions in a contract. While this can take a lot of time to do manually, Biconomy supply their own Inheritable contract allowing for you to enable meta transactions with just a few extra functions.

4.6 Netlify

Netlify is a Platform as a Service (PaaS) for hosting web-based applications with a particular focus on serverless backends which made them a perfect alternative when Moralis web hosting was giving issues. Netlify was easy to set up within my application and has the ability to automatically update the web application when changes have been made to the GitHub repository.

4.7 Microsoft Teams

Microsoft teams was utilised for organising meetings with my project supervisor and provides great tools such as screen sharing and interactive calendars for planning and scheduling meetings. It allowed me to easily showcase my current progress to my project supervisor and get instant feedback on the current state of my application. File sharing is also a key feature of Microsoft teams allowing me to share files with my project supervisor at times where meetings had to be rescheduled. It was the obvious choice for these interactions as it is utilised by GMIT for delivering online education so both myself and my supervisor had access to all its features through our college related Microsoft accounts.

4.7.1 Sendgrid

SendGrid is a cloud-based email delivery service which can be utilised for sending automated emails to users. We can connect this to our Moralis server and run cloud functions to send verification emails when a user signs up for the application. They allow you to design how the email will be presented to the user giving you full customisation.

4.7.2 Visual Studio Code

Visual Studio Code was created by Microsoft as a source code editor with Windows, macOS and Linux support built into to the editor. Visual Studio Code is arguable the most widely used text editing software for creating web-based applications. It gives access to a wide variety of different useful features such as debugging support, syntax highlighting and intelligent code completion plus many more. It has a massive package library allowing you to install many different extensions for making the development process much easier such as auto formatting libraries allowing you to focus on functionality while the editor keeps your code looking pretty. It also provides built in GitHub commands allowing for seamless interactions between your code editor and a projects GitHub repository.

4.7.3 Latex / Overleaf

Latex provides a typesetting system allowing developers to create high quality documentation and others uses such as research papers. I utilised Latex to write this projects dissertation as it comes with a wide variety of extremely useful tools for building high quality documentation. It allowed me to create and label images as figures and split my project into different chapters, sections and subsections at ease with a multitude of different shortcuts to speed up the process. Whiile you can install Latex on your own machine this process was taking hours due to poor region control on the Latex Live installer. This is when I decided to research other alternatives for using Latex and found Overleaf which provide high quality documentation on how to harness the full potential of Latex while also providing a free to use online Latex editor and compiler. I became rather fond of using Latex when it was utilised for writing my research paper which made it the obvious choice for writing this dissertation.

4.8 Javascript Libraries

4.8.1 Create-React-App

Create-React-App is a quick-start environment allowing developers to quickly set up new single page applications with react. It does not come with any prepacked backend logic allowing you to utilise the back-end of your choosing which was perfect for our serverless application and maintaining a close relationship with decentralisation.

4.8.2 Chakra UI

Chakra is a front-end library for simple, modular, and accessible components for building single page NextJS and React applications. They have high quality documentation along with flexible and free to use templates to help speed up the front-end development process. Chakra appears to be a popular UI library in the web 3 space as throughout my research it was recommended through multiple different tutorials including Moralis own tutorials for implementing their services in react.

4.8.3 Web 3

The web 3 API allows you to interact with blockchain networks provided the user has access to a web3 wallet provider such as metamask which is required for this web application. It allows you to create instances of your smart contracts from the blockchain and interact with all functions publicly available. Web 3 was especially useful when the Moralis API was incapable of initialising Biconomy for gasless transactions as you could easily combine the biconomy provider with the metamask wallet provider. It also can be used to access all details from your web 3 wallet account such as transaction history and all balances.

4.8.4 React-Router-Dom

React Router DOM is a popular routing library for react applications for implementing dynamic routing inside of React web applications. It perfectly complements the serverless application as its fully featured on client side allowing for seamless navigation throughout a single-page application.

4.8.5 Effect Hooks (react)

Effect hooks are quality features that come with the standard react library itself which are utilised all throughout the project. They are used for telling a react component that it needs to re-render when specific data has been updated which was crucial for fetching both users' details from Moralis and loading data from the blockchain using the web3 API as It allows you to asynchronously fetch data through APIs.

4.8.6 DatePicker

The react data picker library give you access to a highly customizable date selection interface for allowing users to select specific dates and time inside of an interactive calendar. It also allows you to call different functions to get specific data types which was especially useful for getting the users selected date converted to epoch. However, Solidity uses a different measurement for epoch so all results would have to also be divided by 1000 to get the epoch time in milliseconds

Chapter 5

System Design

5.1 Election Smart Contract (Backend)

5.1.1 createElection()

The createElection() function is responsible for initialising each individual election and populating them with the users given parameters. It starts my pushing an empty element to the array of Election structs described above. We then create an instance of our Election structure and use the electionCount variable to get the newly created elections index and assign it to our newly created instance. The storage keyword is used as this instance is assigned to storage that is outside of our function call (memory can be used for temporary storage that is not stored outside of the function itself). This instance is then populated with the election details from the given parameters before using for loops to populate our voters and candidate's arrays. Finally, the election count is updated to be prepared for the next created election. By populating the voters array we can later ensure that only valid voters are able to participate in the election.

```
    function createElection(string memory _elecName, uint256
  ↪ _startDate, uint256 _endDate, string[] memory _name,
  ↪ string[] memory _info, string[] memory _image, string[]
  ↪ memory _voters) public {
        elections.push();

        Elections storage e = elections[electionCount];
        e.elecID = electionCount;
        e.elecName = _elecName;
        e.startTime = _startDate;
        e.endTime = _endDate;

        for(uint256 i=0; i < _name.length; i++) {
            e.candidates.push(Candidate(1, 0, _name[i],
    ↪   _info[i], _image[i]));
        }

        for(uint256 i=0; i < _voters.length; i++) {
            e.voters.push(Vote(_voters[i], false, 0));
        }

        electionCount++;
    }
```

5.1.2 getCandidates()

The getCandidates() function is utilised to return all necessary election and candidate details based on the given election id as a parameter. Arrays are returned instead of an instance of the election as Soldity does not support returning objects from a function. An abi encoder would have to be used in both our contract and front end to turn the data into bytes which will require more gas to execute the transaction unnecessarily increasing the cost of execution. Election ID's are determined by their index in the election array to reduce the amount of loops executed within the function to reduce gas consumption.

We start by declaring arrays for all the required candidate details using the keyword memory as solidity requires us to specify which type of storage is being used when initialising arrays within a function. Solidity doesn't allow for dynamic arrays to be declared inside of functions, so we use the election ID from our input parameter to make our candidate detail arrays the size of the length of the specified elections candidate array. A for loop is then used to iterate over the elections candidates and populate the arrays with the appropriate candidate details to be returned.

```
    function getCandidates(uint256 _elecID) public view
    ↪   returns (string[] memory, uint256[] memory, string[]
    ↪   memory, string[] memory, string memory, uint256) {

        string[] memory names = new
    ↪   string[](elections[_elecID].candidates.length);
        uint256[] memory voteCounts = new uint256[](elections[_elecID].candidates.length);
        string[] memory info = new string[](elections[_elecID].candidates.length);
        string[] memory image = new string[](elections[_elecID].candidates.length);
        string memory elecName = elections[_elecID].elecName;
        //uint256 startTime = elections[_elecID].startTime;
        uint256 endTime = elections[_elecID].endTime;


        for(uint i = 0; i < elections[_elecID].candidates.length; i++){
            names[i] = elections[_elecID].candidates[i].name;
            voteCounts[i] = elections[_elecID].candidates[i].voteCount;
            info[i] = elections[_elecID].candidates[i].info;
            image[i] = elections[_elecID].candidates[i].image;

        }
        return (names, voteCounts, info, image, elecName, endTime);
    }
```

5.1.3 isElectionLive()

The isElectionLive function is used to check the start and end time of a specific election using the election id as an input parameter. First, we create a temporary Boolean value initialised to false and set an integer value to the current time from the latest block confirmed by the Polygon blockchain by accessing the timestamp from solidity's built-in keyword block. An if statement is then used to check if the current time retrieved from the block is in between the specified elections start and end time. If this condition is true, we set our temporary Boolean variable to true and return it at the end of the function to signify the election is currently live.

```
function isElectionLive(uint256 _elecID) public view returns (bool) {

    bool isLive = false;

```

```
        uint256 lastUpdated = block.timestamp;

        if (lastUpdated >= elections[_elecID].startTime && lastUpdated <= elections[_elecID].endTime) {
            isLive = true;
        }

        return (isLive);
    }
```

5.1.4 hasVoted()

The hasVoted function is used as validation to check if a voter has already participated and casted a vote in the specified election. This function takes in the election's id and the voters id as parameters. The voters ID consists of the user's wallet address as it will be unique to each user. A for loop is used to iterate over the specified elections voters to check if the given voters id is present and if the voters hasVoted property is set to true. If the condition is met in the if statement, the temporary Boolean value is set to true and returned at the end of the function. Solidity does not support the comparison of strings, so I had to create my own function to perform the comparison check. The two strings to be compared are taken in as parameters and an Abi encoder is used to convert both strings to bytes so they can be compared returning true if a match is found.

```
    function hasVoted(uint256 _elecID, string memory _voteID) public view returns (bool) {

        bool valid = false;

        for(uint256 i=0; i < elections[_elecID].voters.length; i++) {
            //elections[_elecID].voters[i]

            if(compareStrings(elections[_elecID].voters[i].voteID, _voteID) && elections[_elecID].voters[i].hasVoted){
                valid = true;
            }
        }

        return valid;
    }
```

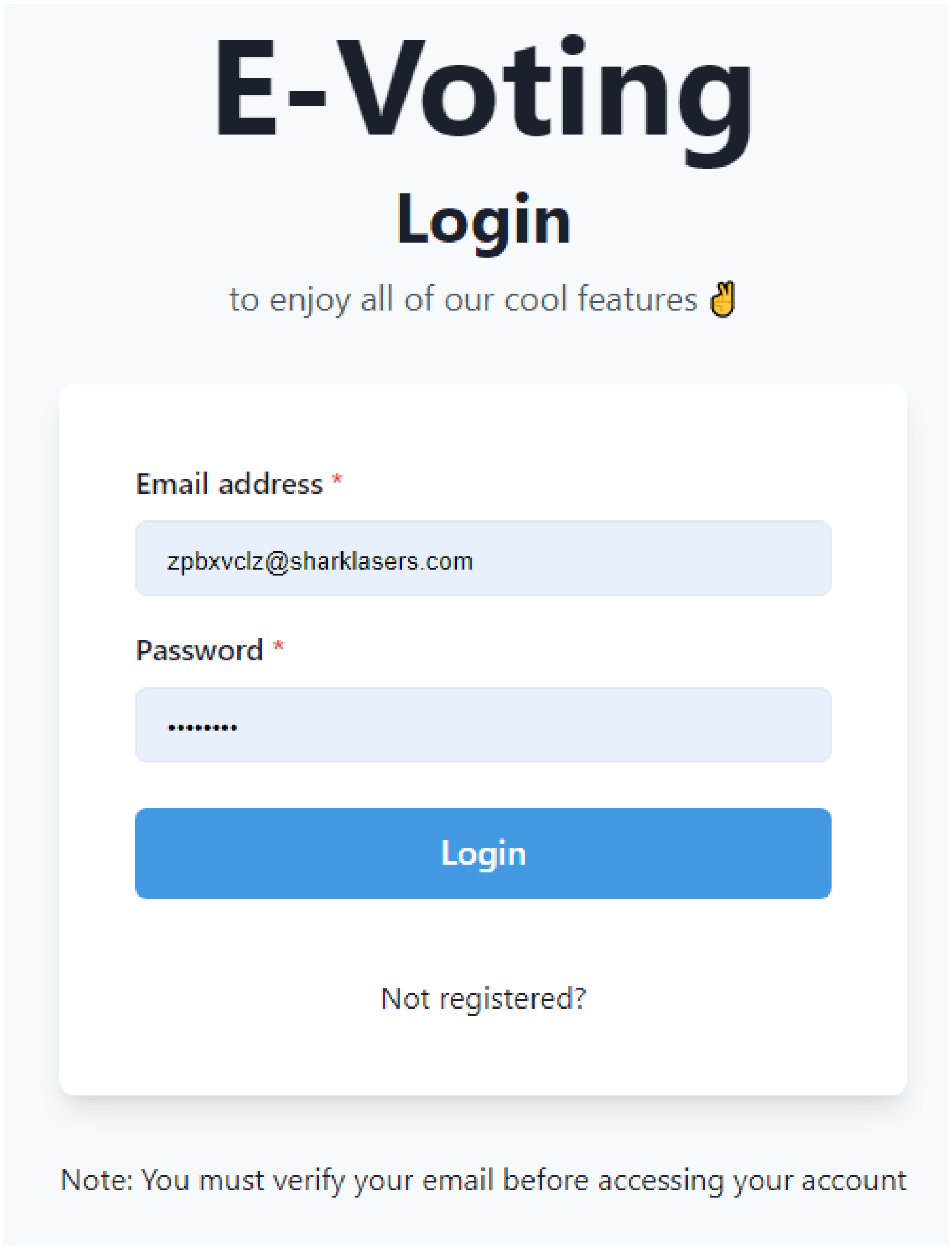
5.1.5 createVote()

The createVote function provides the voting functionality for our election. It takes in an election id, candidate id and finally the voters id as parameters. In order for a user to cast a vote in an election we want to make sure that the election is currently live, a valid voter is trying to cast the vote and that they have not already voted in this election. We can do this by utilising the require function that is built into solidity. Require checks if a condition is valid and returns true if the conditions are met. If the condition is not met it returns false alongside a string containing an error message of the developers choosing. We take the specified parameters and pass them into the two previously discussed functions isElectionLive and hasVoted to check if the specified voter can cast a vote in the election. After the validation checks have been passed using require the function will begin execution and update the desired candidates vote count. A for loop is then used to iterate over the voters for the specified election to update the voters details setting their hasVoted property to true and setting the candID property to their chosen candidate.

```
    function createVote(uint256 _elecID, uint256 _candID, string memory _voterID) public {

        require(hasVoted(_elecID, _voterID) == false && isElectionLive(_elecID), "Already Voted or Election is not live");

        elections[_elecID].candidates[_candID].voteCount++;

        for(uint256 i=0; i < elections[_elecID].voters.length; i++) {
            //elections[_elecID].voters[i]

            if(compareStrings(elections[_elecID].voters[i].voteID, _voterID)){

                elections[_elecID].voters[i].hasVoted = true;
```

```
                    elections[_elecID].voters[i].candidateID =
 ↪  _candID;
                }
            }
        }
```

5.2 Login/Registration


E-Voting

Login

to enjoy all of our cool features ✌

Email address *

zpbxvclz@sharklasers.com

Password *

••••••••

Note: You must verify your email before accessing your account

Figure 5.1: Login Component

5.2.1 Login

The login component loads after the user selects getting started from the home page. The login process itself is handled by our serverless backend with Moralis. We start by initialising the login and user components from the Moralis library to give us access to their login function. Here we can focus on creating a clean front-end as Moralis will handle the login functionality behind the scenes. The useEffect hook is utilised to update and rerender the data inside of the input tags whenever they are modified. Unless a user is authenticated through this login form they can not access any other component of the application other than the homepage. Moralis also handles user session cookies allowing the users browser to rememeber their session key until they decide to logout.

5.2.2 Sign-Up

The Signup component can be accessed after the user selects getting started on the home page. The Signing up process is very similar to logging in and also utilises the serverless backend of Moralis to create an account in our MongoDB database. While Moralis provides authentication through Web 3 wallets such as Metamask this is not suited towards our objective of combining traditional web applications with web 3. First we initialise the signup function from the Moralis library which takes in three parameters in order to create a user. The user is presented with a responsive form with inputs for their email and password. The useState hook is utilised to update and rerender the data inside of the input tags whenever they are modified. When the user has entered their details they can click the forms sign up button which executes a function and carries out checks to ensure the password meets the following criteria.

- Password must contain at least 8 characters
- Password must contain a capital letter bullet.
- Password must contain a number.

Javascript alerts are then used to inform the user which criteria they failed to meet. If all criteria is met we pass in the email and password as arguments into the Moralis signup function. Moralis handles the encryption of the password to ensure a breach in the database will not result in users passwords being leaked.

```
    const checkPassword = () => {

        // Using regex to validate entered password meets the
        ↪  neccesary requirements

        if (password.length < 8) {
            alert("Password must be atleast 8 characters")

        } else if (password.search(/[A-Z]/) < 0) {
            alert("Password must contain a capital letter")

        } else if (password.search(/[0-9]/) < 0) {
            alert("Password must contain a number")

        } else {
            signup(email, password, email)
        }
    }
```

5.3 Metamask Verification as 2FA

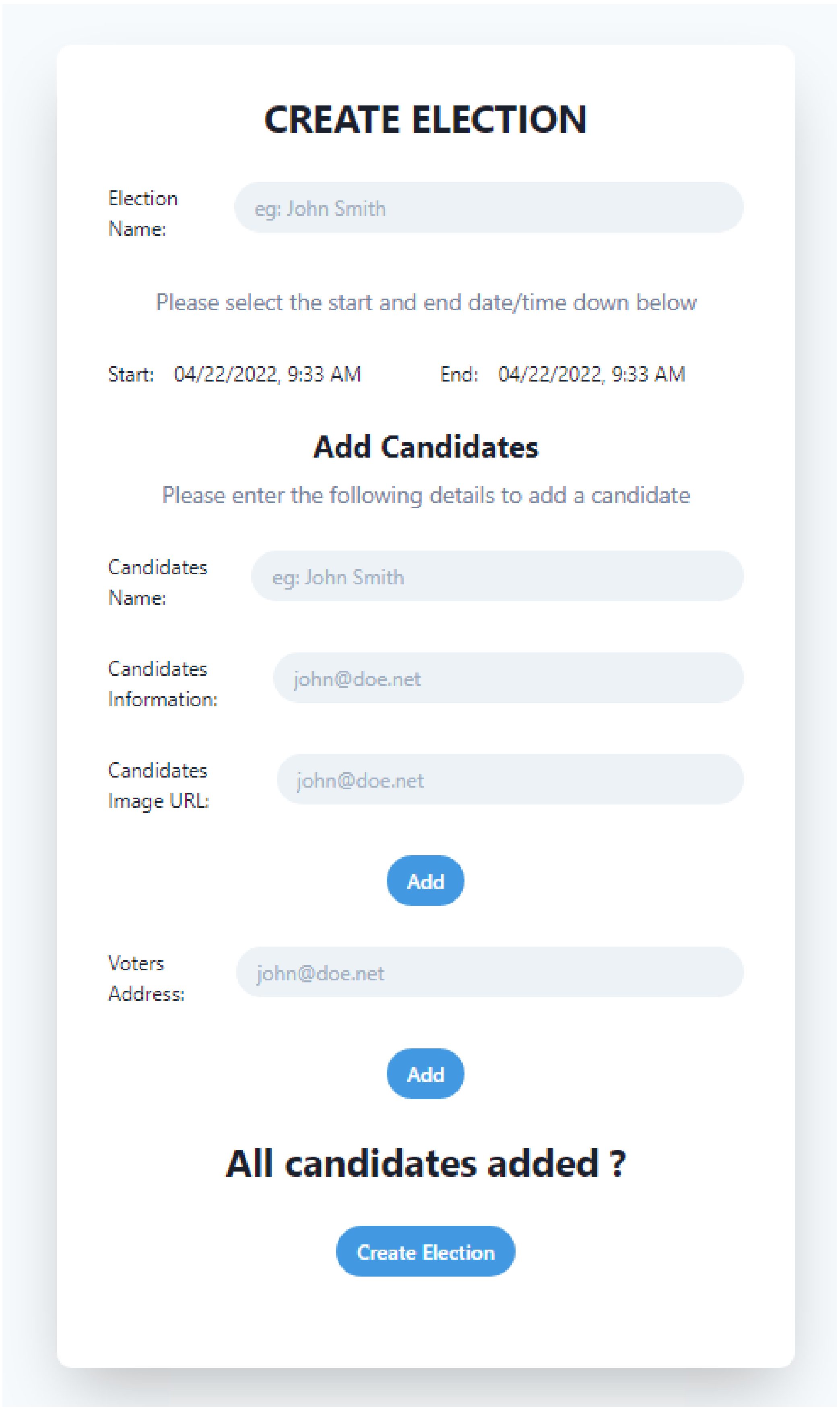
For a user to utilise the web application they will first need to link a metamask wallet to their account. Metamask is used as 2FA for logging into the application combining traditional logins with the web 3 standard of wallet logins. Metamask enhances users account security as it can only be accessed on the device used to create the metamask wallet. The only way for an intruder to gain access to the account is through a private key which is a sha256 hash that currently takes longer than a lifetime to crack even when taking Moore's law into account. After the signup process the user will be presented with a Metamask Authentication failed page where they will have the option to link their currently connected metamask address to their account or logout of the application. The Moralis Web3 API is used to retrieve the user's wallet address from their metamask account through the account property of the useMoralis() function. This is all done through the App.js component so the web application can quickly render the web application after the user has linked their address. When a user clicks the button to link their metamask we call the setUserData() function from the Moralis library. This function is used to set/update users' details in our MongoDB database. We set the users Boolean isLinked property to true and set the linkedAddress property to the users current metamask address.

```
    const [linked, setLinked] = useState(null);
    const [linkedAddress, setAddress] = useState(false);

    useEffect(() => {

    //console.log(linked)
    if (user !== null) {
      //console.log(linked)

      if (user.attributes.linkedAccount === account) {
        //console.log(linked)
        setLinked(true)

      } else if (user.attributes.linkedAccount !== null &&
       ↪  user.attributes.isLinked === null) {
        setAddress(true)
      }

    }
  });

  const checkLinked = () => {

    const users = Moralis.User.current();

    console.log(users.attributes.isLinked)


    if (users && users.attributes.isLinked) {

      alert("Cannot link metamask account as one is already
       ↪  linked to this account")

    } else if (users && !users.attributes.isLinked) {
      setUserData({
        isLinked: true,
        linkedAccount: account,
      })
    }

  }

  else if (isAuthenticated && user && linked === null) {
    return (
      <Container align={"center"} spacing={4}>

        <Incorrect></Incorrect>

        <Button
          onClick={() => checkLinked()}
          disabled={isUserUpdating}
          colorScheme={'blue'}
          bg={'green.400'}
          rounded={'full'}
          px={6}
          _hover={{
            bg: 'green.500',
          }}
        >
          Link Metamask
    </Button>
```

5.4 Email Verification

If the user has singed up but not yet verified their email, they will be instantly logged out from the Moralis server and a JavaScript alert will be displayed informing them to verify their email address before accessing their account.

5.5 Create Election/Poll


CREATE ELECTION

Election Name: eg: John Smith

Please select the start and end date/time down below

Start: 04/22/2022, 9:33 AM End: 04/22/2022, 9:33 AM

Add Candidates

Please enter the following details to add a candidate

Candidates Name: eg: John Smith

Candidates Information: john@doe.net

Candidates Image URL: john@doe.net

Voters Address: john@doe.net

All candidates added ?

Figure 5.2: Create Election

When dealing with elections and polls the front end implementation is the exact same so both have been bundled under the same heading with polls separate feature being explained at the end. The create election component of the web application is loaded when the user navigates to it using the navigation bar along the left side of the screen. This page provides the user with a form with inputs for all the necessary details needed to create an election instance through our smart contract on the polygon blockchain as discussed above under the createElection section when describing our decentralised backend. Once again, the useState hook from the react library is used to update an re-render any data inserted into the input boxes by the user and the react DatePicker library is used to give the user an clean and interactive way to select the start and end dates for the election.

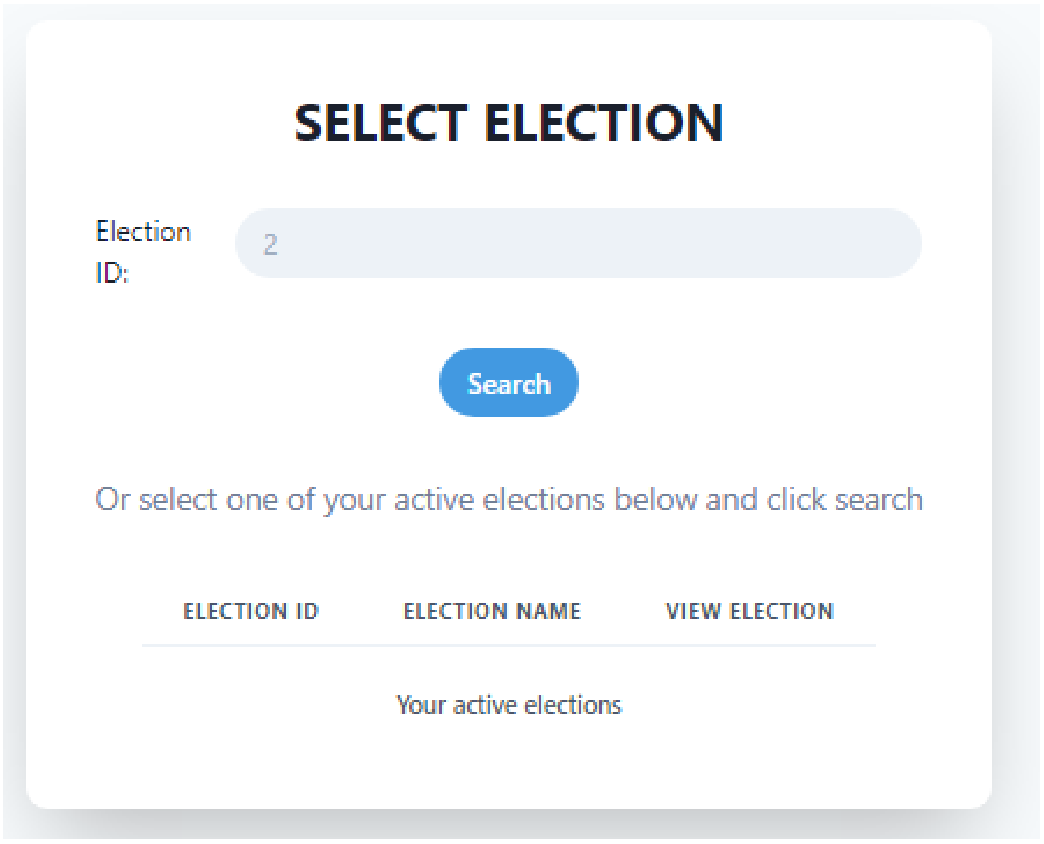
To enable Gasless transaction through the Biconomy library we must initialise an instance of a biconomy provider with our Moralis Speedy Node as the web3 provider for interacting with the polygon Mumbai blockchain network through our nodes RPC URL. We then use the web 3 library to initialise metamask as the wallet provider for interacting with our node along with our Biconomy API key to connect to our specific biconomy application. We then initialise a new web 3 instance with biconomy as the network provider before access the onEvent function from our biconomy instance which is an event listener for checking if Biconomy failed or was successfully initialised. However, if the user was to try create an election before biconomy was initialised they would be forced to pay gas fees themselves. This would ruin the user experience and may scare users away if they must pay for each interaction with the web application. To avoid users running into the issue we use the useState hook once again to set a Boolean variable to true once biconomy is initialised. This variable is then used when rendering the button component by using embedded javascipt and putting isBiconomy AND before rendering the component. The right side of the and operator will only execute if the first condition passes removing the button from the UI unless biconomy is initialised. This is all performed within a useEffect hook that runs once when the component is mounted so any potential errors will not affect the rendering process.

```
    useEffect(() => {

        biconomy = new Biconomy(new
         ↪  Web3.providers.HttpProvider("https://speedy-nodes-nyc.moralis.io/786
         ↪  {
         walletProvider: window.ethereum,
         apiKey:
         ↪  process.env.REACT_APP_BICONOMY_API_KEY_MUMBAI,
         debug: true
        });

        web3 = new Web3(biconomy);

        biconomy.onEvent(biconomy.READY, () => {
         console.log("Biconomy Successful")
         setBiconomy(true)

        }).onEvent(biconomy.ERROR, (error, message) => {
         // Handle error while initializing mexa
         console.log("Failed to initalize")
         alert("Failed to connect to the neccesary smart
         ↪  contract, Please contact support if this issue
         ↪  persists")
        });
    }, []);
```

We can then use the instance of our smart contract to initialise a transaction call of our createElection function from the smart contract passing in the users inputted election details from the form as arguments. A transaction call requires us to specify the senders address and the signature type. Moralis built in Web3 API is used to get the users current metamask address to be for signing the transaction. The signature type is set to EIP712 as it allows for more detailed signature request be presented to the user by metamask. We then call the .on listener which confirms when the user has signed the transaction and a javascript alert is displayed to inform the user their election was submitted to the blockchain. Once the transaction listener sees the transaction has been confirmed by the polygon blockchain the user is given another alert to inform the election was successfully created.

```
    const createElection = (elecNameAdd) => {


        let contract = new web3.eth.Contract(
            ABI.abi,
            process.env.REACT_APP_CONTRACT_ADDRESS
        );

        let start = parseInt(startDate.getTime() / 1000)
        let end = parseInt(endDate.getTime() / 1000)

        console.log(account)

        let tx = contract.methods.createElection(elecNameAdd, start, end, candidateNames, candidateInfo, candidateImage, allParticipants).send({
            from: account,
            signatureType: biconomy.EIP712_SIGN,
            //optionally you can add other options like gasLimit
        });

        tx.on("transactionHash", function (hash) {
            console.log(`Transaction hash is ${hash}`);
            //showInfoMessage(`Transaction sent. Waiting for confirmation ..`);
            alert("Election submitted. Waiting for confirmation .. ")
        }).once("confirmation", function (confirmationNumber, receipt) {
            console.log(receipt);
            console.log(receipt.transactionHash);
            alert("Election has succesfully been created")
        });
    }
```

5.6 Election/Poll Selection


SELECT ELECTION

Election ID: 2

Search

Or select one of your active elections below and click search

ELECTION ID	ELECTION NAME	VIEW ELECTION

Your active elections

Figure 5.3: Select Elections Component

The Select Election/Poll component is loaded after the user navigates to the elections/polls option on the sidebar navigation. The user is presented with an input box prompting them to enter the election id that they would like to display/interact with. Below this is a table containing all the elections they are currently a participant in containing the election ids and names of said elections.

In order to populate the table of user's current elections we call the getElection function from our backend smart contract as described above. Moralis Web3 api is once again utilised the retrieve the users current meta-mask account to be passed as an argument into the getElection function. The smart contract is called using useWeb3ExecuteFunction imported from the react-moralis library. This function does not have to valid properties to initialise biconomy as the network provider so is primarily used for calling read only functions from the smart contract. Once again, we start by getting the contracts ABI that was built by truffle using the require the file from our projects contract folder created by truffle. We initialise the function with the contracts ABI, the contact address, the functions name, and finally pass in the users metamask account as an argument to our function so their elections can be retrieved from the blockchain. The useWeb3ExecuteFunction gives us access to the following variables and functions

- Data: Contains the retrieved data from the executed function
- Error: An error message if the execution failed
- Fetch: Function call for executing and fetching the data from the blockchain
- isFetching: Boolean value to confirm we are currently fetching data from the blockchain
- isLoading: Boolean value to confirm the data retrieved is loading in our application

```
    const { data, error, fetch, isFetching, isLoading } =
     ↪  useWeb3ExecuteFunction({
    abi: ABI.abi,
    contractAddress: contract,
    functionName: "getElection",
    params: {
      _elecID: 0,
      _voteID: account,

    },
  });
```

useEffect and useState hooks are utilised to asynchronously fetch the data from the blockchain without out breaking the render before the data has been retrieved from the smart contract call. An if statement is used inside the useEffect to only fetch the data while isFetching and isLoading are false a long with ensuring the useState variable has not already been populated. This is followed by an else if statement to set the fetched data only when it has been populated to our responsive useState variable. This hook is set to run whenever the isLoading value changed to ensure it is only called a second time after the data has been fully loaded.

```
    useEffect(() => {
    if (!isFetching && !isLoading && !electionEffect) {
      fetch();
      //console.log("Fetching data");
    } else if (data !== null && !electionEffect) {
      console.log(data)

      console.log(data)

      setElections(data)

    }
  }, [isLoading]);
```

Once the data is loaded, we use mapping to loop over our table populating each row with the users retrieved election ids, names and a button to view the election. The election ID is then passed as an argument into our component for retrieving the elections candidates to be used when calling the getCandidates function from our smart contract which we will cover in the next section.

5.7 Displaying Candidates

5.7.1 GetCandidate

Multiple different components are utilised to display candidates from an election or poll. After the user selected their election as described at the end of the previous section, the getCandidate component is passed all candidate details along with the elections name. This component is responsible for converting all candidate details into their own separate candidates so they can be displayed separately in the next component as each candidate requires slightly different parameters for the voting functionality.

We start by declaring an empty array of objects with no defined properties to be later used to store details of all candidates. A for loop is used to iterate over candidates' data passed in from the SelectElection component. Inside the for loop, we define the object structure for a single candidate so we can assign each candidates details to it own object before being pushed our empty object for storing all of our candidates. The details passed from the previous component consists of a two-dimensional array. We use the counter from the for loop to specify the current candidate and populate each separate candidate and push it to our candidates array at the end of each iteration.

```
for (let i = 0; i < data[0].length; i++) {

        let candidate = { name: "Default", voteCount: 0, info:
         ↪  "Deafault info", candID: 0, image: "Default" };
        //console.log(i)
        console.log(data[0][i])
        candidate.name = data[0][i]
        candidate.voteCount = parseInt(data[1][i]._hex)
        candidate.info = data[2][i]
        candidate.image = data[3][i]
        candidate.candID = i;
        console.log(candidate)
        candidates.push(candidate)

        console.log(candidates)
        //console.log(candidate)
    }
```

After populating the array of candidates we create a mapping of all candidates to loop over each individual candidate and pass them into our DisplayCandidate.js component along with the election id. This is done as the DisplayCandidates.js component is also used to implement the voting functionality which requires different parameters for each candidate.

```
{candidates.map(candidate => (
                <DisplayCandidates candidate={candidate}
                 ↪  id={id} />
            ))}
```

5.7.2 DisplayCandidates

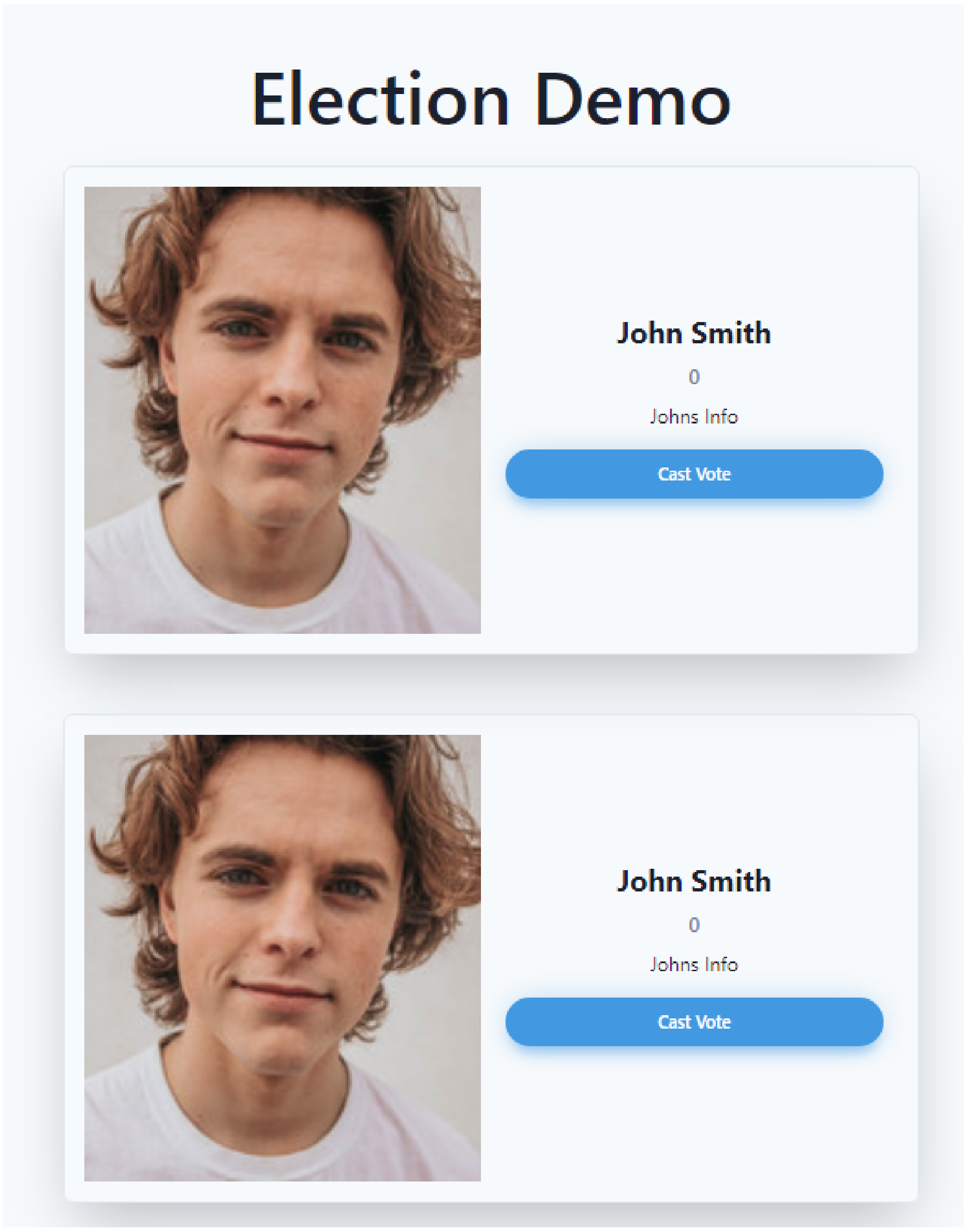


Figure 5.4: Display Candidates Component

The DisplayCandidates component is used to display a single candidate within and sleek and responsive card containing the candidate's image, name, vote count and the candidate's information. This card also contains a button used execute the voting functionality through the smart contract.

```
return (

        <Center py={6}>

            <Stack
                borderWidth="1px"
                borderRadius="lg"
                w={{ sm: '100%', md: '700px' }}
                height={{ sm: '476px', md: '25rem' }}
                direction={{ base: 'column', md: 'row' }}
                //bg={useColorModeValue('white', 'gray.900')}
                boxShadow={'2xl'}
                padding={4}>
                <Flex flex={1} bg="blue.200">
                    <Image
                        objectFit="cover"
                        boxSize="100%"
                        src={
                            candidate.image
                        }
                    />
                </Flex>
                <Stack
                    flex={1}
                    flexDirection="column"
                    justifyContent="center"
                    alignItems="center"
                    p={1}
                    pt={2}>
                    <Heading fontSize={'2xl'}
                    ↪   fontFamily={'body'}>
                        {candidate.name}
                    </Heading>
                    <Text fontWeight={600} color={'gray.500'}
                    ↪   size="sm" mb={4}>
                        {candidate.voteCount}
                    </Text>
                    <Text
                        textAlign={'center'}
                        //color={useColorModeValue('gray.700',
                        ↪   'gray.400')}
                        px={3}>
                        {candidate.info}

                    </Text>

                    <Stack
                        width={'100%'}
                        mt={'2rem'}
                        direction={'row'}
                        padding={2}
                        justifyContent={'space-between'}
                        alignItems={'center'}>

                        {isBiconomy && <Button

                            flex={1}
                            fontSize={'sm'}
                            rounded={'full'}
                            bg={'blue.400'}
                            color={'white'}
                            boxShadow={
                                '0px 1px 25px -5px rgb(66 153
                                ↪   225 / 48%), 0 10px 10px
                                ↪   -5px rgb(66 153 225 / 43%)'
                            }
                            _hover={{
                                bg: 'blue.500',
                            }}
                            _focus={{
                                bg: 'blue.500',
                            }}
                            onClick={() => castVote1()}>
                            Cast Vote
                </Button>}
                    </Stack>
                </Stack>
```

```
            </Stack>

        </Center>
    );
```

5.8 Voting functionality

We start by initialising biconomy following the same steps previously described under creating an election to enable gasless transaction so the user can use there metamask to sign the transaction without paying any gas fees. To ensure the user can't cast a vote before biconomy has been initialised we once again utilise the useState hook to update a Boolean value when biconomy gets initialised and using the AND operator before rendering the cast vote button component, so the button component is not rendered until biconomy has been initialised.

5.8.1 CastVote Function

The cast vote function in our DisplayCandidates component is utilised to call the createVote function from our smart contract on the polygon blockchain. We start by creating a new instance of the smart contract using the contract ABI and address. We then create an instance of the contracts create vote function passing in the election id, candidate id and the users current metamask account as the voter id using the Moralis Web3 API. Event listeners are then used on the transaction to keep the user updated while the vote is being processed on the polygon blockchain. When the user signs the transaction, they are presented an alert informing them that it has successfully been submitted but is waiting to be confirmed by the polygon blockchain. Once the event listener sees the transaction has been confirmed another alert is used to inform them that their vote has been confirmed by the blockchain. If the vote fails the required validation check in the smart contract as described in (2.22 etc) they will get an alert of an error message informing them that they have either already voted, or the election is currently not in progress.

```
const castVote1 = () => {
        let contract = new web3.eth.Contract(
            ABI.abi,
            process.env.REACT_APP_CONTRACT_ADDRESS
        );


        console.log(account)
        const users = Moralis.User.current();
        console.log(users.id)

        let tx = contract.methods.createVote(id, candidate.candID, account).send({
            from: account,
            signatureType: biconomy.EIP712_SIGN,
            //optionally you can add other options like gasLimit
        });

        tx.on("transactionHash", function (hash) {
            console.log(`Transaction hash is ${hash}`);
            //showInfoMessage(`Transaction sent. Waiting for confirmation ..`);
            alert("Vote submitted. Waiting for confirmation ..")
        }).once("confirmation", function (confirmationNumber, receipt) {
            console.log("success");
            console.log(receipt.transactionHash);
            alert("Your vote for " + candidate.name + " has been confirmed by the blockchain!")
        }).on("error", function (error) {
            alert("This election is no longer live or you have already voted")
        });
    }
```

5.9 All Polls

When the user navigates to the all polls section of the web application they are presented with a similar table to the one described in the select election section. Once again we utilise the useWeb3ExecuteFunction from the Moralis Web3 API to call the getElection function from are polls smart contract. The function returns an array containing all the polls names and vote counts so we can sort them in descending order so the most popular election can be displayed first. The combination of useState and useEffect hooks are utilised to set the data. We then pass the retrieved data into the AllPolls component as we call only utilise useWeb3ExecuteFunction once inside a component. While its techinally possible to use useState variables to update the properties before calling fetch this solution proved to an overcomplicated solution and would cause the webpage to slow down as data is being fetched from the blockchain too many times.

The AllPolls component is then used to iterate over polls data and push it to an empty object using the same technique as described in(etc). However, this time the ids are not returned by our function. This is because all polls are being returned so we can use the counter in the for loop to assign the poll id values.

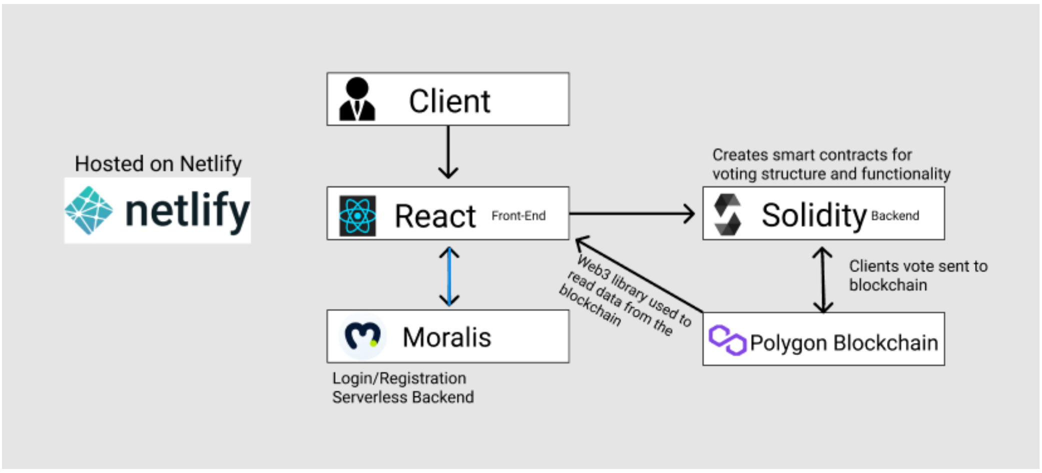


Figure 5.5: System Architecture

Chapter 6

System Evaluation

6.1 Robustness

Due to the nature of an online voting application, It is important for the system to be robust. A wide variety of choices were made to ensure the robustness of the voting functionality.

6.1.1 Testing

Throughout the development process I utilised the Remix IDE for Solidity from the founders of Ethereum. Remix provides an intuitive an easy-to-use way of compiling, deploying, and testing EVM compatible smart contracts. Remix offers a high-quality development environment allowing you to test smart contracts in an interactive environment. It keeps track of deployments which was extremely useful as I could test different versions of my smart contracts side by side without having to realter my code. I also briefly utilised Selium for testing the the front-end of my application as I no longer had to repeatedly enter inputs into the browser as Selium could remember and automate the process for me.

6.1.2 Error Handling

Error handling was utilised all throughout the project to keep the user informed and ensure the don't perform any tasks that may break the application or give undesired results. I performed error handling on all sections of user input into the web application as well as utilising the require function in my smart contracts to ensure only eligible voters can participate in elections. Although due to some limitations with the Moralis API when switching between different routes in the web application the user will briefly see the

Metamask authentication page before their desired route loads. However, validation checks are run when the user attempts to link their metamask account ensuring users can't accidentally link to a new account. Any issues that appear in the project do not disrupt the functionality or break the application but may cause slight inconveniences to users with slow connections between loading web pages.

6.1.3 Speed

The project utilises React as the front-end which is a single-page web application framework allowing for all components to be rendered at high speeds. The decentralised nature of my backend being hosted on the Polygon Blockchain Network means the majority of data has to be retrieved through API calls. Moralis Speedy Nodes were utilised to maintain my own fast and reliable node on the blockchain network. These nodes are extremely fast and allowed for almost instantaneous API calls from the blockchain.

6.2 Objectives Evaluated

6.2.1 Identity Verification

Identity verification is one of the main objectives I wanted to implement as it would ensure only real people are able to participate in elections removing some of the main concerns around online voting systems. However, this unfortunately could not be implemented as all identity verification services require you to be an officially registered company due to legal obligations with data security laws. If this project was to be realised and associated with a registered company, it would be easily implemented with Yoti's (reference) ID verification and liveness checks as they have high quality support and documentation.

6.2.2 Voting Functionality

Blockchain based smart contracts were utilised as the backend of my application to ensure pre-determined conditions were met before a user can cast a vote in an election or poll. As mentioned in the systems design section validation checks can be run before executing functions in smart contracts to ensure users can only vote once in an election or poll. Blockchain based smart contracts are immutable meaning malicious third parties are unable to hack into the contracts and manipulate elections and polls.

6.2.3 Web 3 Provider 2FA

Account security is a necessity in any online voting system. I utilised the most popular Web 3 wallet provider Metamask to create my own 2FA where users have to link their Metamask account and are unable to access the applications dashboard without also connecting to their Metamask account. Metamask accounts can only be accessed by the device where it was created and can only be recovered with a sha256 hashed private key.

6.2.4 Reduce Election/Poll Costs

There would be no point moving voting systems to an online environment if the costs of setting up and running the election/polls were not drastically reduced. This was a key factor when choosing which blockchain network would be used for this project. The Polygon Blockchain was the obvious choice and ensured creating and participating in elections only cost a fraction of a penny. This allows us to sacrifice minor costs to improve user onboarding by not forcing users to pay for blockchain transactions. This was achieved through Biconomy and their gasless transaction service that allows us to pay the fees while users enjoy a cost-free experience. Biconomy also allowed us to put limits on our applications daily transaction to avoid malicious users attempting to drain the applications funding wallet.

6.2.5 Reduce Technical Expertise Needed for Blockchain Based Applications

A key issue with Web 3 applications is the technical expertise normally required for users to interact with the blockchain. Biconomy was not only important in reducing costs but due to users no longer having to fund their own wallet which would usually require the lengthy process of purchasing cryptocurrencies through an exchange. This meant users only needed to create a metamask account to utilise the application without having to fund their wallet for performing transactions with the blockchain. This helped bridge the gap between web 2 and web 3 applications while still taking advantage of the security features web 3 wallets provide.

Chapter 7

Conclusion

7.1 What I learned from this project

This project gave me the opportunity to learn a lot regarding the full development process of creating an industry standard web application. I got to take a dive into blockchain development and gain a greater understanding of the possibilities Web 3 infrastructure offers.

7.1.1 Blockchain Development

I learned a lot regarding the inner workings of blockchain networks and the benefits of using Layer 2 solutions such as Polygon compared to the standard and more widely used Ethereum network. I learned a lot about how decentralised applications can be created utilising blockchain networks and smart contracts for the backend functionality of web applications. It also however thought me the difficulties of developing with newer technologies as there's not as many readily available resources when encountering errors in the application. It allowed me to put the developer skills I have learned throughout the course of my study and apply existing knowledge to new and evolving technologies.

7.1.2 Moralis

Moralis opened my eyes to the possibilities of serverless back ends and how they can be combined with blockchain based smart contracts to create truly decentralised applications. The Moralis framework forced me to no longer over rely on the likes of forum posts for guidance due to it being a newer framework with a smaller and less active community. Although the Moralis documentation was of very intuitive it used rather one-dimensional approaches to explaining some specific libraries functionality helping me improve my problem-solving skills as on multiple occasions I had to figure out my own work arounds to find solutions. I would most certainly advise anyone interested in Web 3 development to consider Moralis infrastructure as although not perfect they provide an insanely high-quality service completely free of charge and are constantly releasing new updates and features.

7.1.3 React

My experience with React was extremely limited before starting this project and showed me the true potential of single page applications and how it can be used to create fast and responsive user interfaces. React gave me a far greater understanding of asynchronous functions as I had to utilise useEffect hooks throughout my project as I was constantly retrieving and updating components with live data from the blockchain.

7.1.4 Solidity

I learned a lot about Solidity through this project as it is very recent language that I had never used before. It showed me how the last four years have given me the skills necessary to develop applications using new and evolving languages that require a different approach to understanding a unique syntax as it can be rather unforgiving compared to other languages that don't punish you as much for poor coding practices. It showed me the difference between developing smart contracts stored on blockchain networks compared to more commonly used languages and the different limitations that come with it.

7.1.5 Truffle

Truffle taught me more than I expected regarding the deployment of smart contracts to blockchain networks. I learned a lot about how Node modules can be utilised to combine different languages such as JavaScript and Solidity to create far superior development environments.

7.2 Importance of planning

This project showed me the true importance of extensive planning. I learned a lot about the Agile methodology and how prioritising an iterative approach to the development process can be majorly beneficial in not just reaching the end goal but the importance of all the steps taken along the way. The Agile approach also taught me the importance of regular meetings to help keep the project on track and to keep myself motivated throughout the whole process as there was always a goal regardless of the stage in the development life cycle. This was emphasised during times where meetings had to be rescheduled or were missed as there was a noticeable trend of progress deteriorating during these times.

7.2.1 Scope Management

I learned a lot about managing the scope of projects as my initial ideas proved to be out of reach when considering all the new technologies and frameworks I was planning to implement. While I did get to utilise all the proposed technologies except Yoti for identity verification, I was forced to sacrifice some key features that would vastly improve the users experience such as more details in areas such as displaying statistics when elections or polls had ended. At times I felt overwhelmed with all the new technologies that resulted in lots of time been spent on functionality that got scrapped for simpler solutions that I was unaware of due to a lack of experience in the field. To end on a positive note, I feel I successfully outlined the more critical features of the application from an early stage of development and even though I sacrificed features that make the application more aesthetically pleasing and informative I was able to successfully lay the necessary groundwork that can easily and quickly be built upon to fully fledge out the desired application.

7.3 What Would I Do Differently ?

7.3.1 Testing

Looking back on how tested the application I should have put more emphasis on automated testing with my smart contracts to gain a greater understanding of true Test-Driven-Development. While the Remix IDE was an extremely useful tool for testing it limited the developer skills I could of show cased throughout the development process. Automated tests also provide good context to other developers about the true requirements of different functions in my smart contracts.

7.3.2 Error Handling

I definitely could of put more focus on error handling even though there's no known issues which will break the application I definitely could of utilised more validation on user inputs to provide them with a more informative experience and reduce the potential inconvenience if the user makes slight mistakes with their inputs. For example, users can technically create elections during time frames where the election will never be considered live but does not break the application itself.

7.3.3 Time Management

Throughout the course of the development process, I most certainly could have managed my time a lot better. I found myself getting hyper focused on the problem-solving aspect of utilising the new technologies and struggled to motivate myself when it came to the more minor details which can greatly enhance the users experience. I've always had issues with tedious tasks that take up time as I find the process boring, and it showed with the final product.

7.3.4 NodeJS as Backend

Following research into decentralised applications I became fascinated with the idea of a serverless application using Moralis. However, in hindsight utilising NodeJS could have saved a lot of time that I spent solving issues with rendering data in my application. I found myself running into a few issues when trying to asynchronously retrieve data from the blockchain and the Moralis database where NodeJS could have loaded the data on the back end before rendering components in React reducing the time spent fixing data related errors.

7.4 Closing Statement

Overall, I am relatively happy with the final product as it allowed to apply the skills, I have gained over the years to utilise new technologies with limited resources to create a decentralised application. It allowed me to gain a greater understanding of the specific industry that piques my interest in the world of software development giving me experience with highly sought-after technologies in the word of blockchain development. I learned a lot about the importance of the entire process involved in creating an application. However, I am rather disappointed with the lack of motivation I had when developing more minor features to enhance the user experience as it resulted in a less appealing application that I feel doesn't represent the true potential I have as a developer. Finally, I would like to thank you for taking the time to review my final year project and dissertation and give a special thanks to my project supervisor Joseph Corr, who was a massive help throughout the development life cycle of this project. I would also like to thank all lecturers and staff in GMIT that helped accommodate my progress in becoming a software developer.

Bibliography

[1] Khan, S.N., Loukil, F., Ghedira-Guegan, C. et al. Blockchain smart contracts: Applications, challenges, and future trends. Peer-to-Peer Netw. Appl. 14, 2901–2925 (2021). https://doi.org/10.1007/s12083-021-01127-0

[2] Nakamoto S Bitcoin: A peer-to-peer electronic cash system. Available online at https://bitcoin.org/bitcoin.pdf (2008). Last accessed: 2020-10-20

[3] Buterin Vitalik (2014) A next-generation smart contract and decentralized application platform. White paper

[4] Poon, J. and Buterin, V., 2022. Plasma: Scalable Autonomous Smart Contracts. [online] Plasma.io. Available at: ¡https://www.plasma.io/¿ .

[5] Moralis Team: Moralis Documentation README. Available at: ¡https://docs.moralis.io/introduction/readme/¿ .

[6] WebXSoftware: The most popular front-end frameworks for 2021. Available at: ¡https://docs.moralis.io/introduction/readme/¿ .

[7] Kaleido: Get From Idea to Dapp Quickly and Easily. Available at: ¡https://www.kaleido.io/blockchain-platform/truffle

[8] Moralis Team: Speedy Nodes. Available at: ¡https://moralis.io/speedy-nodes//¿ .

[9] YesElections (12/19/2019): The Pros and Cons of On-Site Voting. Available at: ¡https://www.yeselections.com/blog/pros-cons-on-site-voting/¿ .

[10] Teamwork: Everything you need to know about agile project management. Available at: ¡https://www.teamwork.com/project-management-guide/agile-project-management//¿ .

[11] Biconomy Documentation: Mexa - Enable Gasless Transactions. Available at: ¡https://docs.biconomy.io/products/enable-gasless-transactions/¿ .

[12] Biconomy Documentation: Why Biconomy?. Available at: ¡https://docs.biconomy.io/introduction/why-biconomy/¿ .

[13] 101 Blockchains: Top 10 Common Solidity Issues. Available at: ¡https://101blockchains.com/solidity-issues//¿ .