



Faculty of Computer Science and Information Technology

***CryptoVote: A Blockchain Based Voting Platform For
UNIMAS MPP Elections***

Fakhrur Hijjaz Danial Bin Abdul Taib

Bachelor of Computer Science with Honours

(Computational Science)

2025

CryptoVote: A Blockchain Based Voting System For UNIMAS MPP Election

Fakhrur Hijjaz Danial Bin Abdul Taib

The project is submitted in partial fulfilment of the

Requirements for the degree of

Bachelor of Computer Science with Honours

(Computational Science)

Faculty of Computer Science and Information Technology

UNIVERSITI MALAYSIA SARAWAK

2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE Blockchain-Based Voting System for UNIMAS MPP Election

ACADEMIC SESSION: Sem 2 24/25

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (✓)



CONFIDENTIAL

(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)

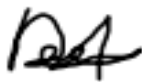


RESTRICTED

(Contains restricted information as dictated by the body or organization where the research was conducted)

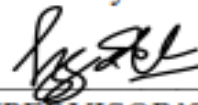


UNRESTRICTED



(AUTHOR'S SIGNATURE)

Validated by



(SUPERVISOR'S SIGNATURE)

Permanent Address

No 82 Lot 3149 Taman

Pelita Jaya Jalan

Sultan Tengah

Date: 28/7/2025

Date: 28/7/2025

Declaration

I declare that the work in this thesis was carried out in accordance with the regulations of Universiti Malaysia Sarawak. Except where due acknowledgements have been made, the work is that of the author alone. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.



Signature

Name: Fakhrur Hijjaz Danial Bin Abdul Taib

Matric No: 77971

Faculty of Computer Science and Information Technology University Malaysia Sarawak

Date: 28/7/2025

Acknowledgement

In the name of God, the Most Merciful and Compassionate, I would like to begin by thanking God for giving me the strength and guidance to complete this project. I would also like to thank God for giving me Dr. Izzatul Nabila as my supervisor. She has been patient with me and always provided advice to help me improve, even

when I made the same mistakes. Despite her busy schedule as both a lecturer and a mother, she still made time to guide and support me throughout my progress.

-

Abstract

Traditional voting systems often rely on centralized authorities, making them susceptible to fraud, manipulation, and inefficiencies. Issues such as vote tampering, high operational costs, and delays in result processing are highlighted. To address these challenges, this project explores the potential of blockchain technology in creating a decentralized voting system. By leveraging blockchain's immutability and transparency, this project aims to develop a secure and verifiable voting platform for university elections. The system will ensure voter anonymity, prevent unauthorized access, and streamline the election process through digital verification methods. A mobile application will be designed to facilitate voting, while an administrative interface will manage election details and candidate registration. Through this initiative, the project seeks to modernize student elections, enhancing trust, accessibility, and efficiency while demonstrating how blockchain can improve electoral processes in academic environments.

Abstrak

Sistem pengundian tradisional sering bergantung kepada pihak berkuasa pusat, menjadikannya terdedah kepada penipuan, manipulasi, dan ketidakefisienan. Isu seperti pemalsuan undi, kos operasi yang tinggi, dan kelewatan dalam pemprosesan keputusan sering menjadi cabaran utama. Untuk mengatasi masalah ini, projek ini meneroka potensi teknologi blockchain dalam membangunkan sistem pengundian yang terdesentralisasi. Dengan memanfaatkan ketelusan dan keselamatan blockchain, projek ini bertujuan membangunkan platform pengundian yang selamat dan boleh dipercayai untuk pilihan raya universiti. Sistem ini akan memastikan anonimitas pengundi, mencegah akses tanpa kebenaran, dan memperkemas proses pengundian melalui kaedah pengesahan digital. Aplikasi mudah alih akan dibangunkan bagi memudahkan proses pengundian, manakala antara muka pentadbiran akan digunakan untuk mengurus maklumat pilihan raya dan pendaftaran calon. Melalui inisiatif ini, projek ini berusaha untuk memodenkan pilihan raya pelajar dengan meningkatkan kepercayaan, kebolehcapaian, dan kecekapan, serta menunjukkan bagaimana teknologi blockchain dapat memperbaiki proses pemilihan dalam persekitaran akademik.

Table of Contents

Acknowledgement	i
Abstract	ii
Abstrak	iii
List of Figures	viii
List of Tables	xi
CHAPTER 1: INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement	1
1.3 Objective	2
1.4 Brief Methodology	2
1.5 Scopes	3
1.6 Significance of Project	3
1.7 Project Schedule	4
1.8 Expected Outcome	5
1.9 Project Outlines	5
1.9.1 Chapter 1: Introduction	5
1.9.2 Chapter 2: Literature Review	5
1.9.3 Chapter 3: System Design and Requirements	5
CHAPTER 2: LITERATURE REVIEW	7
2.1 Introduction	7
2.2 Review on Similar Existing Systems	8
2.2.1 Uvotz	8
2.2.2 Voatz	10
2.2.3 Follow My Vote	15
2.2.4 Comparison of Features between Existing System and CyptoVote.	20

2.3 Proposed System.....	20
2.4 Chapter Summary	21
CHAPTER 3: REQUIREMENT ANALYSIS AND DESIGN	22
3.1 Introduction.....	22
3.2 Methodology	22
3.2.1 Requirement Analysis.....	23
3.2.2 System Design	23
3.2.3 Implementation	23
3.2.4 Testing	23
3.2.5 Deployment.....	24
3.2.6 Maintenance	24
3.3 Requirements	24
3.3.1 Functional Requirements	24
3.3.2 Non-Functional Requirements	26
3.3.3 Software Requirements	27
3.3.4 Hardware Requirements.....	31
3.4 Design.....	31
3.4.1 Context Diagram.....	31
3.4.2 Data Flow Diagrams	32
3.4.4 Entity Relationship Diagram.....	35
3.4.5 Wireframes.....	39
3.5 Summary....	43
Chapter 4: Implementation	44
4.1 Introduction.....	44
4.2 Technology Stack.....	44
4.3 Installation.....	47
4.3.1 Software Tools	47

4.3.2 Smart Contract Dependencies	48
4.3.3 Backend Dependencies	49
4.3.4 Frontend Dependencies.....	50
4.4 Smart Contract Development and Deployment to Blockchain.....	52
4.4.1 Solidity Code	52
4.4.2 Smart Contract Deployment to Sepolia Blockchain	61
4.5 Backend API Endpoints and Frontend Pages	65
4.5.1 Backend API Endpoints.....	65
4.5.2 Frontend Pages.....	67
4.6 Sign Up Workflow.....	68
4.6.1 UNIMAS Siswa Email Verification	68
4.6.2 Create Password.....	72
4.6.3 Passkey Registration Using WebAuthn.....	74
4.7 Login Workflow.....	75
4.7.1 Login.....	75
4.7.2 Login Using WebAuthn.....	77
4.8 Reset Password Workflow.....	78
4.9 Connect Wallet Workflow	81
4.10 Claim Token Workflow	83
4.11 Logout Workflow.....	84
4.12 Admin Workflow.....	86
4.12.1 Admin Registration.....	86
4.12.2 Creating Poll	87
4.12.3 Edit Poll	90
4.12.4 Delete Poll.....	92
4.13 Contestants Workflow	94
4.13.1 Contestants Registration	94

4.13.2 Contesting Poll.....	95
4.14 Voter Workflow	97
4.15 Summary	99
Chapter 5: Testing.....	100
5.1 Introduction.....	100
5.2 Testing Strategy	100
5.3 Functional Testing	100
5.3.1 Smart Contract Functional Testing	101
5.3.2 Authentication Functional Testing.....	107
5.4 Non-Functional Testing	115
5.4.1 Participants.....	115
5.4.2 Authentication and Registration Experience	115
5.4.3 Authentication and Registration Experience	116
5.4.4 Usability and User Satisfaction Ratings	117
5.5 Summary	119
Chapter 6: Conclusion and Future Work	120
6.1 Overview.....	120
6.2 Conclusion	120
6.3 Future Work.....	122
6.4 Summary	123
REFERENCE	124
APPENDICES.....	127

List of Figures

Figure 2.1 Uvotz Splash Screen.....	8
Figure 2.2 Uvotz Student Verification.....	8
Figure 2.3. Uvotz Home Screen.....	9
Figure 2.4. Uvotz Voter Account Registration, One-Time Password, and Account PIN number	10
Figure 2.5. Voatz Log In In	11
Figure 2.6. Voatz Home Page	11
Figure 2.7. Voatz Voter Verification, Government-based ID document Upload, and face scan	12
Figure 2.8. Voatz Successful Verification	13
Figure 2.9. Voatz Vote Casting.....	13
Figure 2.10. Voatz Voter Verifying their Vote.....	14
Figure 2.11. Follow My Vote Start Page	15
Figure 2.12. Follow My Vote Voter Verification Process.....	15
Figure 2.13. Follow My Vote Voter Upload Selfie and State ID	16
Figure 2.14. Follow My Vote Voter Pending Verification.....	16
Figure 2.15. Follow My Vote Admin Verifying Voter.....	17
Figure 2.16. Follow My Vote Voter Successful Verification.....	17
Figure 2.17. Follow My Vote Event Page	18
Figure 2.18. Follow My Vote Successful Vote.....	19
Figure 3.1. Waterfall Model.....	22
Figure 3.2. A Simplified Structure of Blockchain	27
Figure 3.3. Blockchain Decentralize Network.....	28
Figure 3.4. Ethereum Logo	29
Figure 3.5. React Native Logo	29
Figure 3.6. Context Diagram	31
Figure 3.7. Data Flow Diagram Level 0	32
Figure 3.8. Data Flow Diagram Level 1 for User Registration Process	34
Figure 3.9. Data Flow Level 1 For Smart Contract Interaciton Process.....	35
Figure 3.10. Entity Relationship Diagram	35
Figure 4.1. Core Technology Stack	44
Figure 4.2. Solidity and Hardhat.....	45

Figure 4.3. Sepolia Ethereum Testnet.....	45
Figure 4.4. Crypto Wallet MetaMask	46
Figure 4.5. MetaMask as a bridge to blockchain	46
Figure 4.6. Node.js.....	47
Figure 4.7. Node Package Manager (NPM).....	47
Figure 4.8. Visual Studio Code.....	48
Figure 4.9. Poll & Contestant Struct.....	52
Figure 4.10. Smart Contract Variables & Counters	53
Figure 4.11. Create Poll Function	55
Figure 4.12. Update Poll Function	56
Figure 4.13. Delete & Get Poll Function	57
Figure 4.14. Contest Function.....	58
Figure 4.15. Get Contestants Function	59
Figure 4.16. Vote Function	60
Figure 4.17. Hardhat Network Configuration Script	61
Figure 4.18. Hardhat Deploy Script.....	62
Figure 4.19. Hardhat Deploy Command.....	63
Figure 4.20. Hardhat Contract Verification on Etherscan	63
Figure 4.21. Contract Verification on Etherscan	64
Figure 4.22. Transaction History on Etherscan	65
Figure 4.23. Client to Server Communications	65
Figure 4.24. Sign In Page.....	69
Figure 4.25. Input Validation for Sign In Page.....	69
Figure 4.26. User Already Exist Message	69
Figure 4.27. Brevo Email Service.....	70
Figure 4.28. Email Verification Code.....	70
Figure 4.29. Email Verification Page	71
Figure 4.30. Invalid or Expired OTP Message	71
Figure 4.31. Create Password Token	72
Figure 4.32. Hashed Password in Database	73
Figure 4.33. Home Page.....	73
Figure 4.34. Welcome Email	73
Figure 4.35. WebAuthn Registration	74
Figure 4.36. Input Validation in Login Page	75

Figure 4.37. Login Page.....	75
Figure 4.38. Login Page Invalid Credentials	76
Figure 4.39. Login via WebAuthn	77
Figure 4.40. Forgot Password Link.....	78
Figure 4.41. Forgot Password Page	78
Figure 4.42. Reset Password Email	79
Figure 4.43. Reset Password Page	80
Figure 4.44. Reset Password Success Email.....	80
Figure 4.45. Choose Wallet	81
Figure 4.46. Confirm Wallet Connection	82
Figure 4.47. Connect Wallet Success	82
Figure 4.48. Claim Token Button	83
Figure 4.49. Claim Token Success Message	83
Figure 4.50. Logout Button.....	84
Figure 4.51. Admin Manual Registration	86
Figure 4.52. Admin Create Poll	87
Figure 4.53. Create Poll Modal.....	88
Figure 4.54. Admin Confirm Poll Creation Transaction Request	89
Figure 4.55. Successful Poll Creation.....	90
Figure 4.56. Admin Edit Poll.....	91
Figure 4.57. Edit Poll Transaction and Confirmation.....	91
Figure 4.58. Updated poll	92
Figure 4.59. Admin Delete Poll	93
Figure 4.60. Contestant Registration	94
Figure 4.61. Contest Button for Contestants.....	95
Figure 4.62. Contest Poll Form Modal	95
Figure 4.63. Contest Poll Transaction and Confirmation	96
Figure 4.64. New Contestant Displayed	96
Figure 4.65. Vote flow	97
Figure 4.66. Updated Poll Details After Vote.....	98

List of Tables

Table 2.1. Features Comparison	20
Table 3.1. Hardware Requirements	31
Table 3.2. Data Dictionary for Voter	36
Table 3.3. Data Dictionary for Candidate	37
Table 3.4. Data Dictionary for Vote	37
Table 3.5. Data Dictionary for Election	38
Table 3.6. Data Dictionary for Smart Contract	38
Table 4.1. List of Backend API Endpoint	66
Table 4.2. List of Frontend Pages	67
Table 6.1. Functional Requirements and Implementation Status	120

CHAPTER 1: INTRODUCTION

1.1 Background

Blockchain technology, first introduced by Satoshi Nakamoto in 2008 with the creation of Bitcoin, has since revolutionized how we handle digital data. Nakamoto's idea eliminated the need of a centralized entity by creating a decentralized, tamper-proof ledger (Nakamoto, 2008). While initially focused on cryptocurrency, this technology has since found applications in various fields, especially those that require secure and transparent handling of data.

In today's world, secure and transparent voting systems are crucial for maintaining the integrity and trustworthiness of elections. Traditional voting systems often rely on central authorities for counting votes and keeping data, which opens the door to risks like fraud, vote tampering, and a lack of transparency.

A blockchain-based voting system, however, can address these issues by offering a secure, anonymous, and transparent platform. By recording every vote on the blockchain, it ensures that votes cannot be changed or deleted, creating a verifiable and trustworthy process (Vladucu et al., 2023). Additionally, according to Mccorrey et al. (2021), Blockchain prevents the risk of double voting, fraud, external and internal threats due to its decentralized nature. This project aims to implement such a system to MPP elections, to increase accessibility, security, and confidence in the results.

1.2 Problem Statement

In traditional voting system, data storage for votes usually controlled by central authorities like election commissions or governments. This centralized control creates several problems, such as the possibility of vote manipulation, data tampering, fraud, and delayed results. Centralized voting systems also depend on physical infrastructure, which increases costs. Furthermore, the process of verifying votes through intermediaries often leads to delays and inefficiencies.

Blockchain technology presents an effective solution to these challenges. By decentralizing the vote storage and storing each vote in a tamper-proof ledger, it ensures that the votes are both immutable and verifiable without the need for a central authority (Malkawi et al., 2021). This system eliminates the risks of fraud and manipulation, speeds up vote processing, and cuts costs by removing the need for physical polling stations and intermediaries.

1.3 Objective

The objectives of this project include:

1. To develop a secure, transparent, and accessible to design and develop a secure, transparent, and accessible blockchain-based voting system for university student elections.
2. To implement a blockchain-based system that provides a high level of security, preventing unauthorized access and ensuring data integrity throughout the voting process.
3. To test the functionality of the blockchain-based voting system among UNIMAS students.

1.4 Brief Methodology

The development of this app follows the Waterfall Methodology where the six phases of Software Development Life Cycle (SDLC) are completed sequentially. In the waterfall model, each phase is completed before moving to the next phase. Initially all the requirements are gathered to ensure the fulfilment of the objectives. Next, the design phase includes both the frontend and backend components to understand the activity and data flow of the app. After the design phase, the app is implemented, where it is developed and deployed.

After the implementation, testing phase of the cycle ensure that the app is tested to meet the requirements and expectations. Once testing is successful, the app is launched and made

available for use. Finally, an ongoing maintenance is carried out to monitor performance and apply necessary updates. By following the SDLC, the development process is organized, ensuring a secure, reliable, and user-friendly blockchain-based voting system.

1.5 Scopes

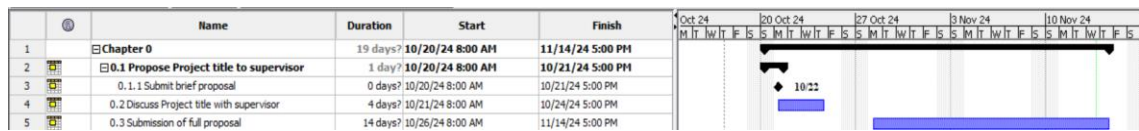
The scope of this project is to develop a mobile application voting platform on Android and iOS for UNIMAS students MPP elections and integrating blockchain to store the votes data to enhance security by preventing data tampering. Only UNIMAS students will be eligible to register to the app by utilizing One-Time Password(OTP) authentication process via their student siswa email address. The application will also include an admin interface, where administrators can manage the election process by entering the election details and candidate registration.

1.6 Significance of Project

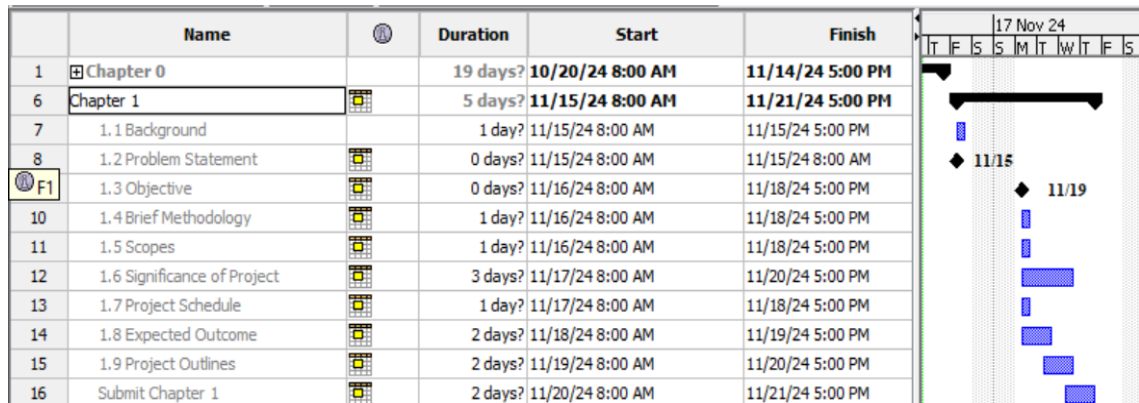
The significance of this project is to improve the vulnerabilities of centralized online voting platforms by integrating blockchain technology. While automating the election process offers benefits such as reducing human error in vote counting, saving time, and cutting costs, centralized systems still face challenges with data integrity. These systems can be vulnerable to both external hacks and internal threats. By using blockchain, this project seeks to address these issues, as blockchain provides an open, tamper-proof ledger to securely store election data, ensuring the transparency and security of the voting process.

1.7 Project Schedule

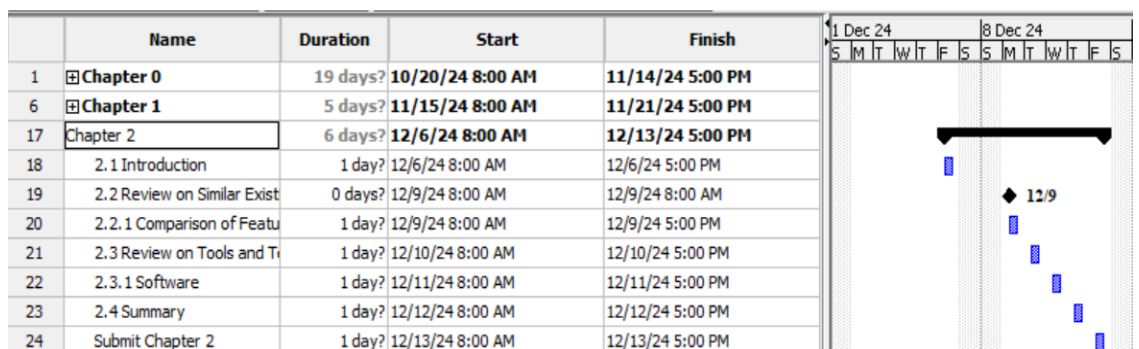
Brief Proposal and Full Proposal:



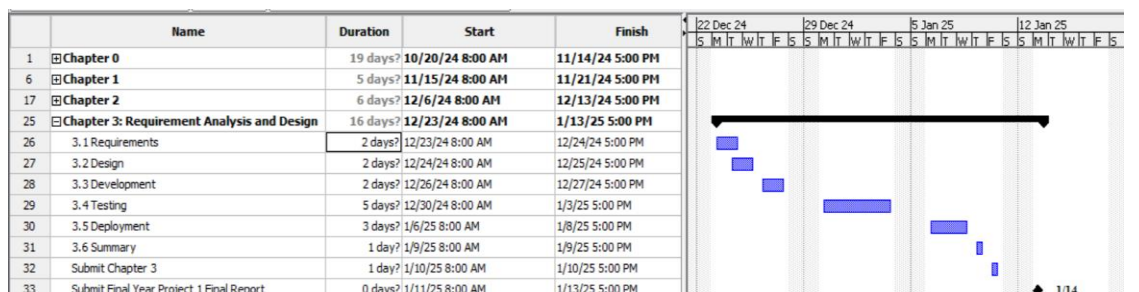
Chapter 1:



Chapter 2:



Chapter 3:



1.8 Expected Outcome

The expected outcome of this project is a mobile application of the blockchain-based voting system that meet all the requirements of a secure vote storage and user friendly interface for conducting UNIMAS MPP elections. This app is expected to serve only for UNIMAS student by implementing a robust account registration process through a One-Time Password (OTP) authentication process. The next key expected outcomes include a user-friendly voting interface for administrators to efficiently manage elections and monitor results. By using blockchain, the app ensures data integrity and eliminates the risks of data tampering and fraud, which is a common weakness in traditional voting systems. Additionally, the integration of blockchain provides an open and transparent ledger for public verification of vote counts, further enhancing the system's credibility.

1.9 Project Outlines

1.9.1 Chapter 1: Introduction

This chapter introduces the project by outlining the motivation behind developing a blockchain-based voting system. It discusses the problems with traditional voting methods, the objectives of the project, and the scope of its implementation. Additionally, it highlights the significance of the system, emphasizing how blockchain can enhance security, transparency, and efficiency in student elections.

1.9.2 Chapter 2: Literature Review

This chapter explores existing electronic voting systems such as Uvotz, Voatz, and Follow My Vote. It examines their features and identifies their weaknesses, including inadequate authentication measures, lack of voter anonymity, and reliance on centralized systems. By analyzing these limitations, this chapter justifies the need for a blockchain-based voting system that enhances security, prevents data tampering, and ensures a more reliable election process.

1.9.3 Chapter 3: System Design and Requirements

This chapter focuses on the technical aspects of the proposed voting system, detailing the necessary requirements for its development. It covers functional and non-functional

requirements, as well as hardware and software specifications. Additionally, it presents the system's design structure, including data flow processes, database models, and user interface wireframes. These elements serve as the blueprint for building a secure and user-friendly blockchain-based voting application.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter focuses on reviewing existing voting platforms and analyzing their procedures and technologies, both in terms of software and hardware. The goal is to identify gaps and limitations in the current systems and explore areas for improvement.

E-voting has long been a focus for cybersecurity developers, as they work to address vulnerabilities and ensure these systems are secure, reliable, and user-friendly. With the rapid pace of digitalization in today's world, online voting is becoming an increasingly relevant solution to electoral system. However, there are still significant challenges to overcome. MIT news (2020) states that, online voting systems remain vulnerable to serious failures, including large-scale attacks that are harder to detect and easier to execute compared to traditional paper-ballot-based voting systems. These vulnerabilities underscore the need for rigorous testing, robust security measures, and transparent processes to ensure the integrity of online elections.

By finding and addressing the gaps in existing voting platform such as ensuring robust authentication process, voter anonymity and preventing data fraud and tampering, we can work towards perfecting a secure and trustworthy e-voting system that meets the demands of modern society while minimizing potential risks.

2.2 Review on Similar Existing Systems

2.2.1 Uvotz

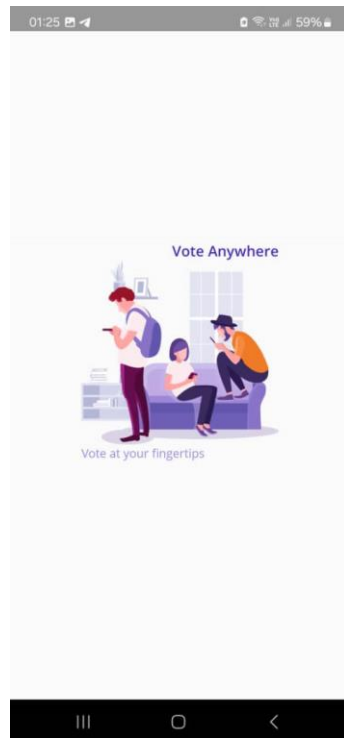


Figure 3.1 Uvotz Splash Screen



Figure 3.2 Uvotz Student Verification

Uvotz is a mobile application designed exclusively for UNIMAS students to serve as a voting platform for MPP elections. When launched, the initial interface of Uvotz is the splash screen in Figure 1.

Voter verification interface is displayed in Figure 2, showcasing the input fields for student matric number and password. To enhance security and streamline the user experience, Uvotz designed the app so that only UNIMAS students can log in directly without the need to create or register an account. This is made possible through pre-existing accounts created by UNIMAS for all students, ensuring that only legitimate students can access the app. This is equivalent to government-based elections where voters need to verify their driver's license or passport. Students simply log in using their matric ID and password, which are also linked to UNIMAS's primary e-learning system, eLeap.



Figure 3.3. Uvotz Home Screen

After login, users are directed to the home screen, where ongoing and upcoming ballots are displayed. The election ballots and candidate lists are managed and published by the app's administrator. The vote counting process occurs automatically in the backend system to ensure accuracy and efficiency.

2.2.2 Voatz

Voatz is a mobile election platform that utilizes blockchain and has been used in a variety of elections across different states in U.S. including Utah, West Virginia, Arizona, South Dakota, and Colorado. Voatz has also been used for student elections at Tufts University in Massachusetts (Harris, 2017).

2.2.2.1 Voatz User Authentication

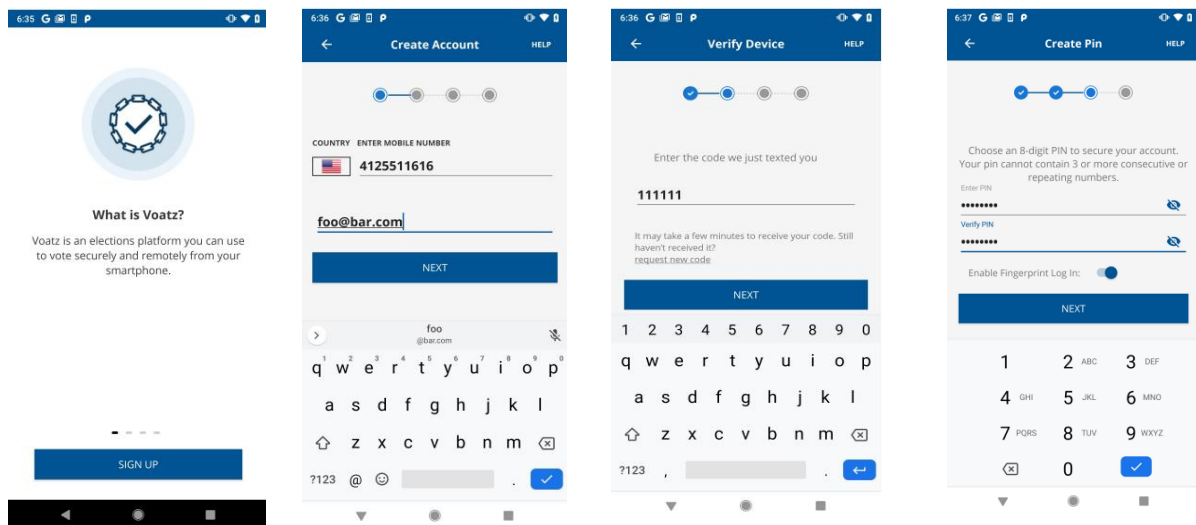


Figure 3.4. Uvotz Voter Account Registration, One-Time Password, and Account PIN number

When the Voatz app is first launched, the splash screen is the initial interface presented to the voter. The voter can swipe through the screen to view an introduction to the app's features and functionality. To begin creating an account, the voter selects the "Sign Up" button available on the splash screen.

The first step in the registration process requires the voter to provide a phone number and email address. These details are used to perform a One-Time Password (OTP) verification via SMS. Following successful verification, the voter is prompted to create a secure 8-digit PIN to safeguard their account. According to Voatz (2020), this PIN is then sent to the server, where it plays a key role in the authentication process.

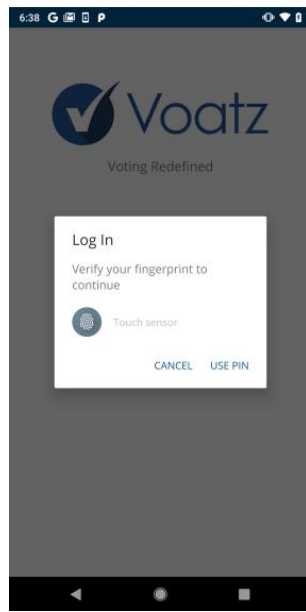


Figure 3.6. Voatz Home

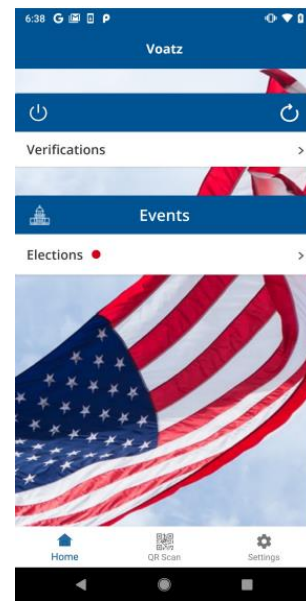


Figure 3.5. Voatz Log

From Figure 4, for devices equipped with fingerprint recognition, voters are offered the option to use their fingerprint as an alternative authentication method. This process works by securely encrypting the 8-digit PIN with a key tied to the voter's fingerprint using the Android Keystore. The encrypted PIN is then stored locally on the device, enabling secure and convenient access.

Once the registration process is complete, the voter is required to log in using either the 8-digit PIN or the registered fingerprint in their device as displayed in Figure 5, ensuring both security and streamline user experience. Upon successful account log in, user is directed to the home page where voter need to go through the next procedure which is voter verification.

2.2.2.2 Voatz User Verification

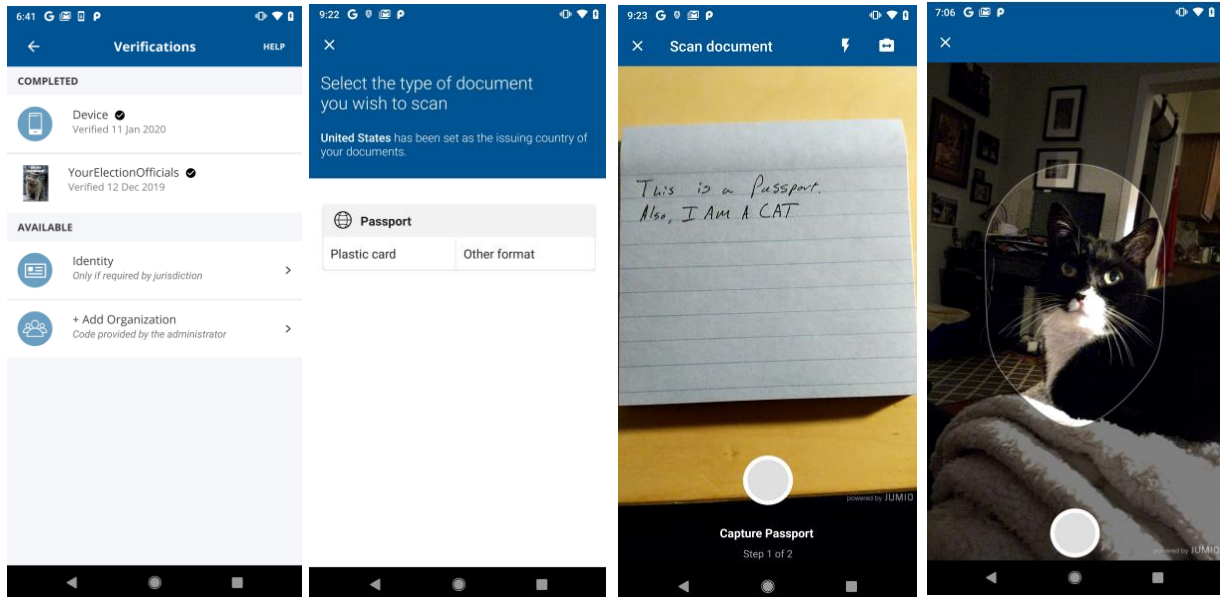


Figure 3.7. Voatz Voter Verification, Government-based ID document Upload, and face scan

After the voter has successfully registered their account, they must still verify their identity within the country or organization. Voatz implements the physical election's identity verification procedure which involves submitting either a state ID or passport. The first screen in Figure 7 shows the initial voter verification interface when clicking the verification tab in the home screen as displayed in Figure 6.

According to Voatz (2020), to facilitate this voter verification process, Voatz integrates with Jumio, a third-party identity proofing service known for its real-time government-based ID verification. Jumio is widely recognized for detecting document tampering and is used by banks like CIMB, HSBC, and Metro to enhance security measures.

When the voter selects the identity verification option on the verification page, Voatz prompts them to provide an official document for identification, typically by scanning their photo ID or passport. The next step involves completing a photo matching process. This step is crucial for comparing the photo on the state ID or passport against the selfie that the voter provides. According to Specter et al. (2020), two facial recognition engines are employed to perform this comparison.

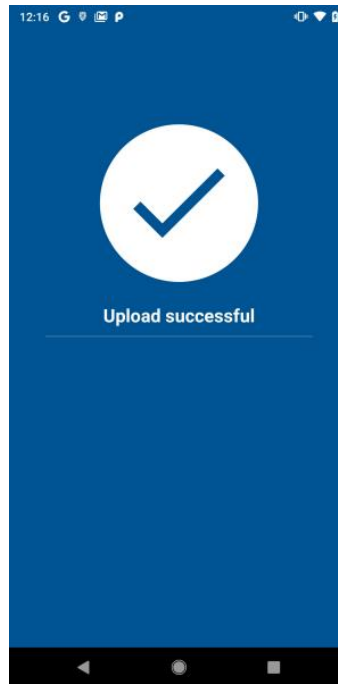


Figure 3.8. Voatz Successful

If the photos match, the voter's identity is successfully verified and the details are securely uploaded to Jumio's server. Figure 8 shows the message that the voter details are uploaded to Jumio's server.

2.2.2.3 Voatz Cast Vote

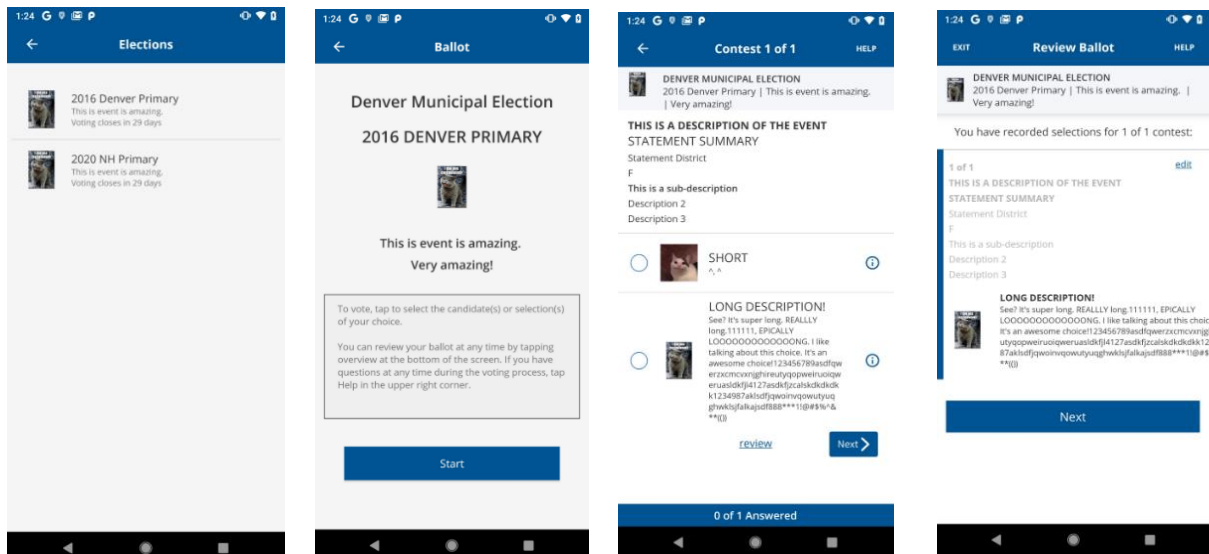


Figure 3.9. Voatz Vote

After the voter's identity has been successfully verified, the app queries the server to retrieve configuration data related to the events the voter is eligible to participate in (Voatz, 2020). This data triggers a menu that allows the voter to select from the available events. The configuration data includes all accessible events, the corresponding ballots for those events, the specific questions on each ballot, and the answer options for those questions. The voter starts by selecting an event from the menu. Once the event is selected, the voter can view the contents tied to that event and make their candidate selections.

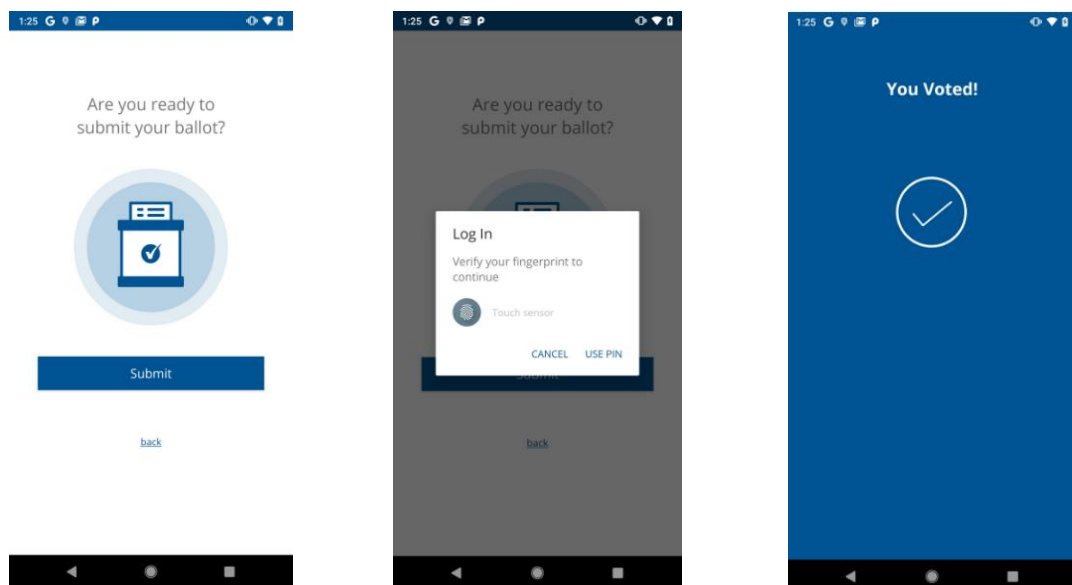


Figure 3.10. Voatz Voter Verifying their Vote

After making their choices, the voter submits their responses to the server. At this point, the app prompts the voter to verify their vote by providing either PIN number or fingerprint as a final authentication step before the ballot is sent to the server. The vote is then submitted to a blockchain.

2.2.3 Follow My Vote

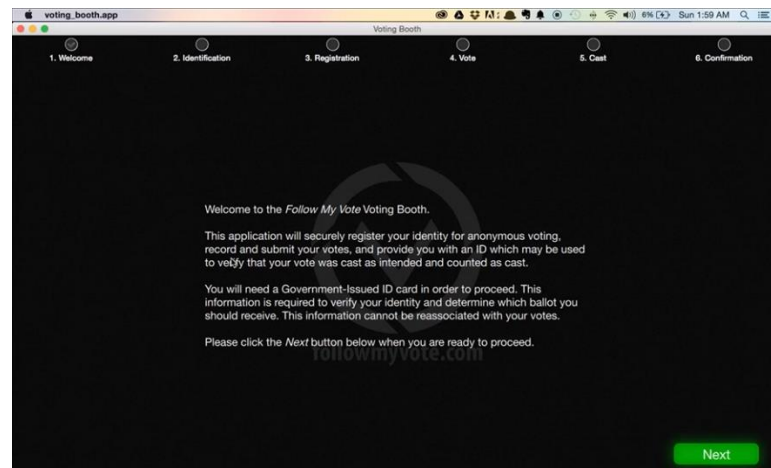


Figure 3.11. Follow My Vote Start Page

Follow My Vote is an activist organization that has created an open source blockchain based voting software. It is an organization that supports the use of online voting and the blockchain technology to develop a verifiable and trackable voting system, hence the name.

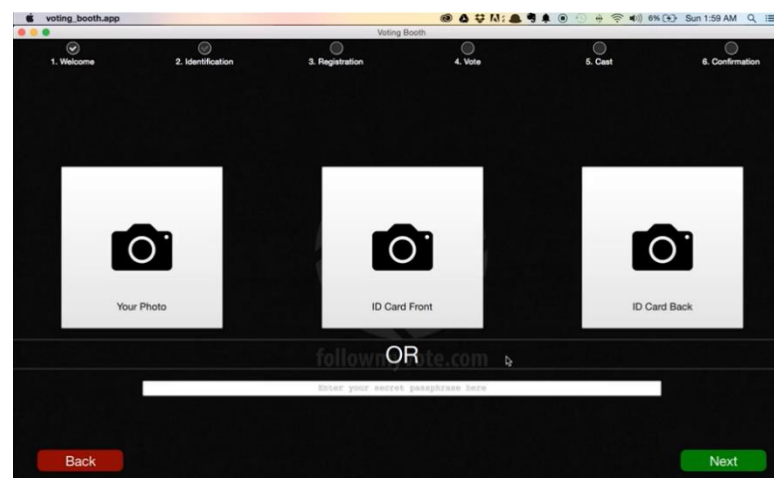


Figure 3.12. Follow My Vote Voter Verification Process

In Follow My Vote's proposed voter verification procedures, they follow the actual process of paper ballot voting procedure. Firstly, the voters will need to upload their selfie to the system. Then following the physical voting procedure, voter need upload the front and back of their state ID.

2.2.3.1 User Verification

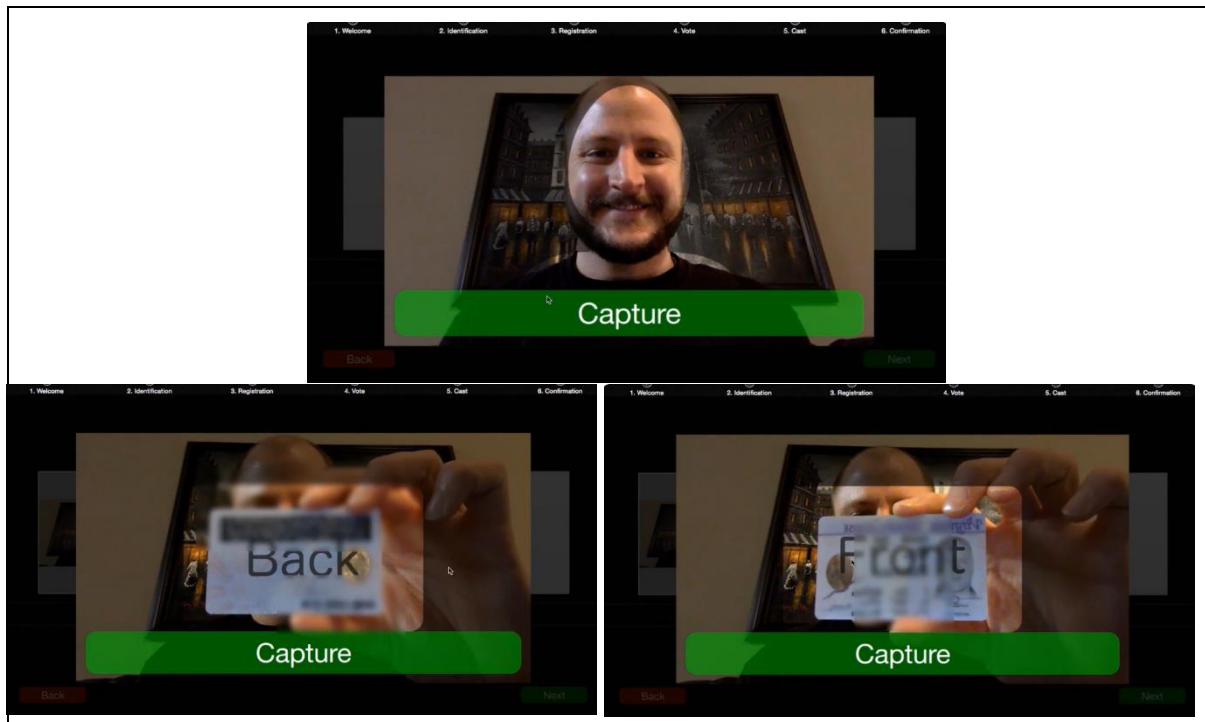


Figure 3.13. Follow My Vote Voter Upload Selfie and State ID

The user verification process begins with the voter submitting their identity details to establish their eligibility to participate in the election. The voter is required to upload a selfie and a scanned copy or clear photo of their government-issued ID card. This submission ensures that the identity of the voter can be authenticated accurately.

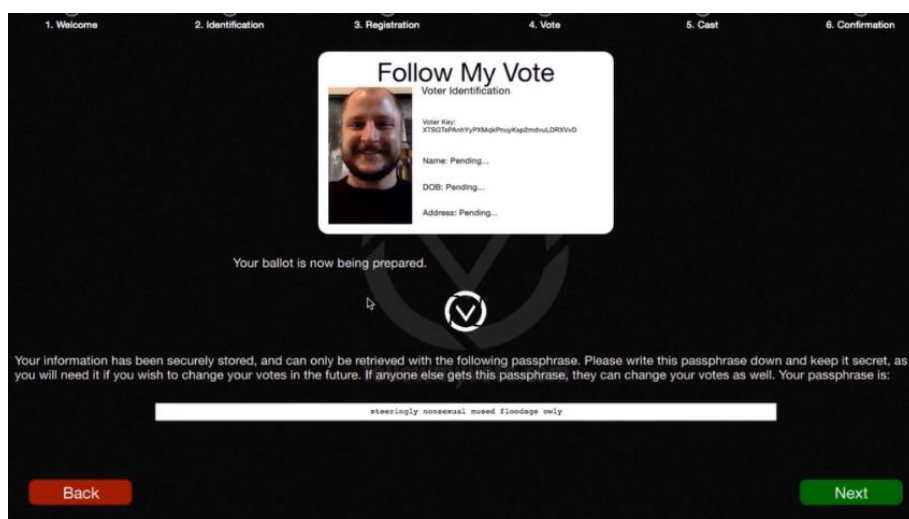


Figure 3.14. Follow My Vote Voter Pending Verification

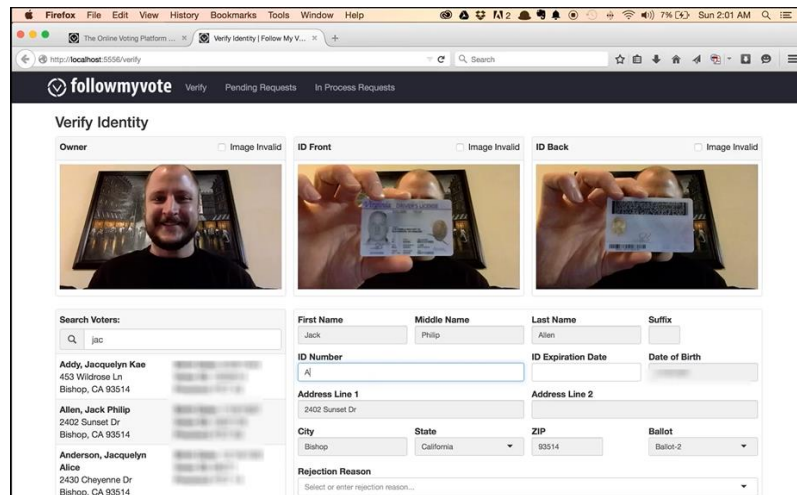


Figure 3.15. Follow My Vote Admin Verifying Voter

Once the voter clicks the "Submit" button, the uploaded information is securely transmitted to a third-party identity verifier, such as Jumio or BlockScore (Follow My Vote, 2016). After submitting their details, the voter is in a pending state as displayed in Figure 14. These verifiers are integrated into the Follow My Vote system and are responsible for manually verifying the authenticity of the submitted documents. At the identity verifier's end, an administrator reviews the selfie and ID card submitted by the user. According to Follow My Vote (2016), the admin searches the voter's name, as listed on the submitted ID to find a match in the pre-approved list of registered voters in the state. This list, which is connected to the third-party integration, serves as a reference to confirm voter eligibility. If the administrator successfully locates the voter's name in the database and determines that the information matches, the admin clicks the "Accept" button to approve the verification.

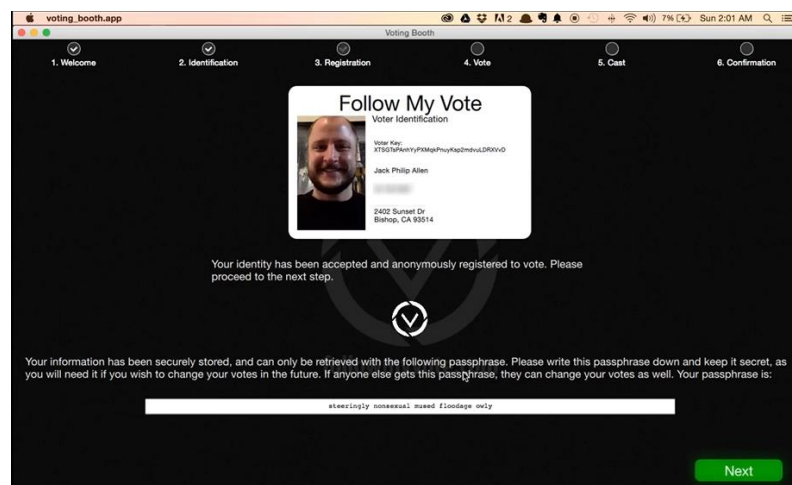


Figure 3.16. Follow My Vote Voter Successful Verification

Once the administrator approves the voter's identity, the user is notified that their account has been successfully verified as shown in Figure 16. This approval grants the voter secure access to proceed to the next stage of the voting process.

2.2.3.2 User Cast Vote

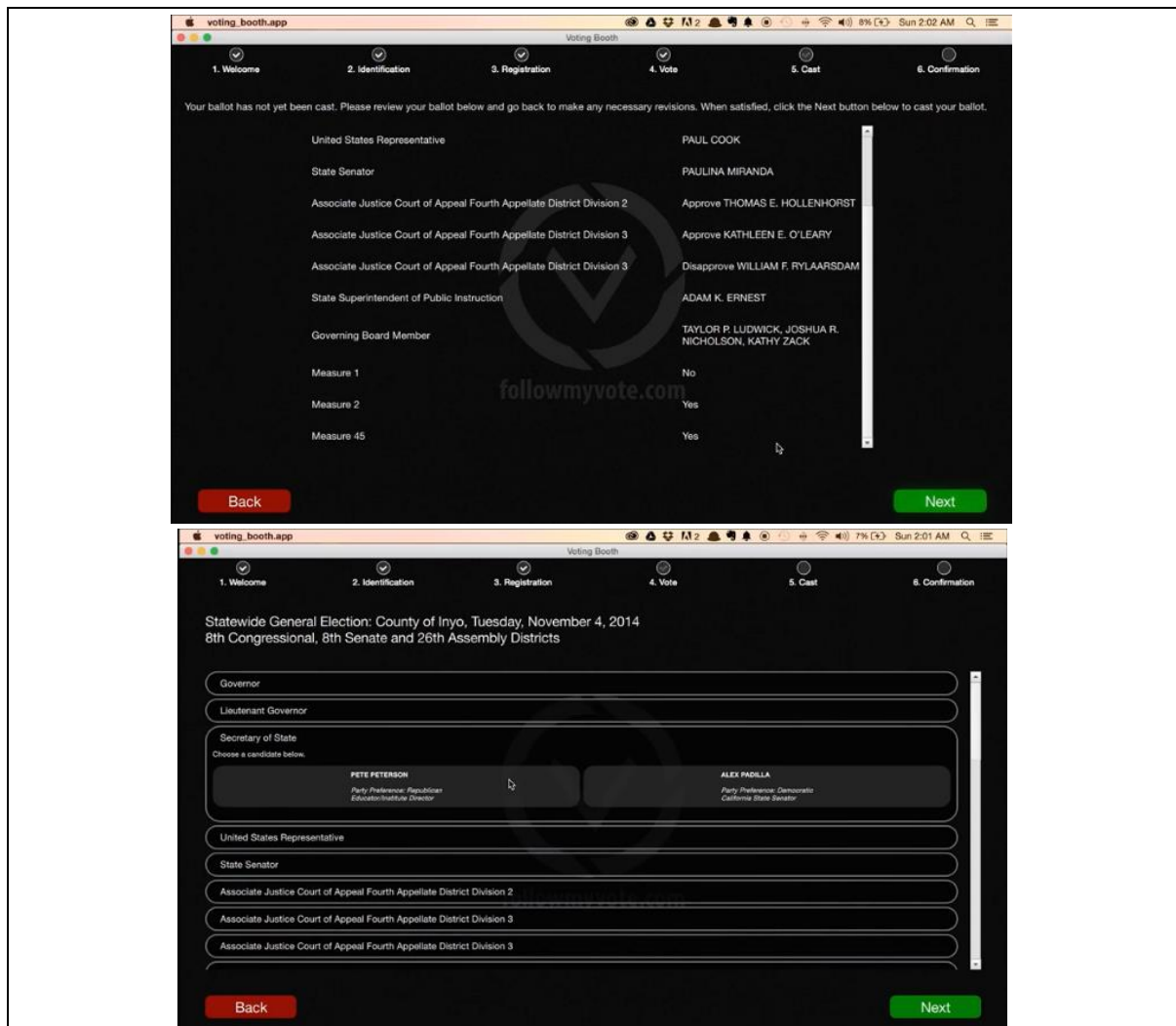


Figure 3.17. Follow My Vote Event Page

After successful verification, the voter can access the ballots or events page. The platform allows verified users to securely interact with the election interface. Upon navigating to the ballots section, the voter is presented with the list of candidates, questions, or options relevant to the election or event in which they are participating.

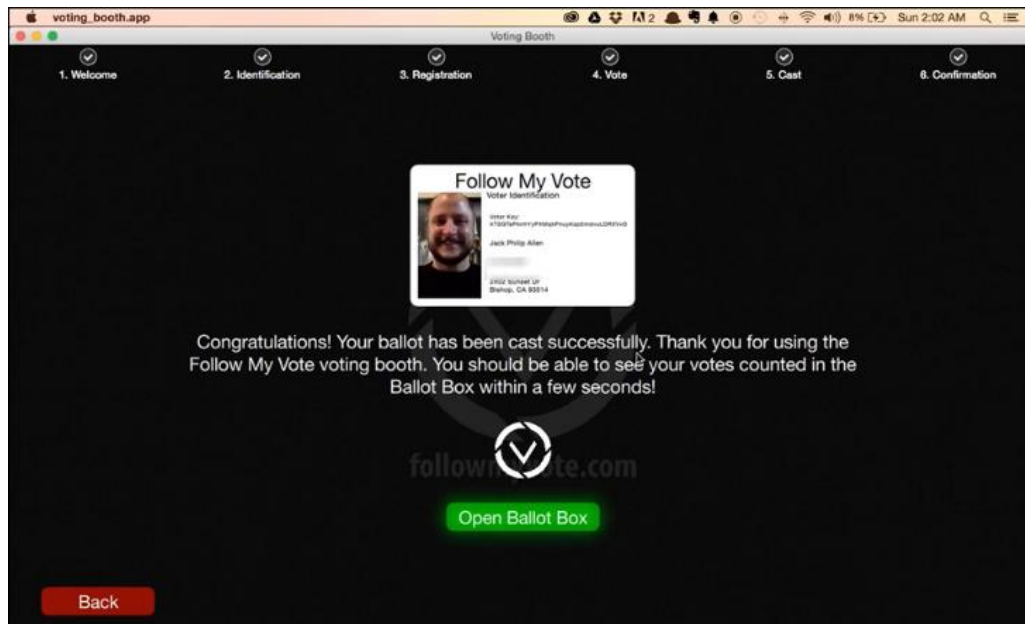


Figure 3.18. Follow My Vote Successful Vote

The voter makes their selections through the intuitive voting interface, which ensures clarity and ease of use. To maintain the integrity of the voting process, the voter's selections are encrypted immediately upon submission and recorded on the blockchain. This ensures that the vote is immutable, meaning it cannot be altered or tampered with after it has been cast.

Importantly, while the system guarantees that the voter’s selection is securely stored and publicly verifiable, it also ensures complete anonymity. The cryptographic mechanisms in place decrypt the voter’s identity from their ballot, preventing anyone from tracing a vote back to the individual who cast it (Follow My Vote, 2026).

Once the vote is cast, the voter receive a receipt or confirmation as shown in Figure 18. This receipt allows the voter to verify that their vote was accurately recorded on the blockchain without revealing their choice. The process concludes with the voter having participated securely and anonymously in the election.

2.2.4 Comparison of Features between Existing System and CryptoVote.

Table 3.1. Features Comparison

Apps Features	UVotz	Voatz	Follow My Vote	CryptoVote (Proposed System)
Double Authentication	X	/	X	/
Voter Verification	/	/	/	/
Transparent and Tamper proof ledger (Blockchain)	X	/	/	/
Fingerprint Authentication	X	/	X	/
Voter Anonymity	X	/	/	/
Vote Receipt	X	/	/	/
Vote Tracking	X	/	/	/

2.3 Proposed System

An analysis of existing voting systems, including Uvotz, Voatz, and Follow My Vote, reveals several key weaknesses. Uvotz and Follow My Vote both lack double authentication, increasing the risk of unauthorized access. They only depend on voter verification process based either on government-based or organization ID. The proposed system aims to leverage Uvotz account registration process whereby only UNIMAS student can enter the app. Additionally, in the proposed system, voter verification process are done completely during the

user registration process where the double authentication step are implemented via the student email. Therefore, the procedure of facial matching between the student ID implemented in Voatz and Follow My Vote can be discarded. In order to be fully dependent on single voter verification process, the verification process need to have a robust security measure. For example, Follow My Vote implements facial scan to ensure a match between the selfie and photo ID uploaded. On the other hand, CryptoVote uses double authentication process via the student email to prevent unauthorized access.

The proposed system, CryptoVote, aims to address these shortcomings by incorporating all the necessary features, as outlined in Table 1. While Voatz and Follow My Vote use blockchain technology to ensure transparency and tamper-proof records, CryptoVote plans to build on these strengths and integrate additional layers of security and features, fingerprint authentication, voter anonymity, vote receipt and vote tracking. By learning from the strengths and weaknesses of existing systems, CryptoVote seeks to provide a more secure, reliable, and transparent voting experience, while also protecting voter anonymity and offering a vote receipt for each user.

In essence, CryptoVote aims to leverage the strengths of Voatz but discarding the facial matching process for voter verification process. CryptoVote aims to leverage the existing voter verification step in the current UNIMAS voting app, Uvotz..By combining robust security measures and with the transparency and immutability of blockchain technology, CryptoVote strives to create a more secure, reliable, and trustworthy voting system.

2.4 Chapter Summary

This chapter discusses the analysis of existing voting system including Uvotz, Voatz, and Follow My Vote. The purpose of analyzing each existing voting system is to highlight its key weaknesses and propose a new system that attempts to fill these gaps. The key weaknesses include the lacking feature of double authentication, fingerprint authentication, voter anonymity, and centralize system.

CHAPTER 3: REQUIREMENT ANALYSIS AND DESIGN

3.1 Introduction

This chapter discusses the requirements and design of the blockchain-based voting application for the UNIMAS's student presidential election. The goal of this application is to provide a secure, transparent, and efficient voting system that overcomes the limitations of traditional centralized methods. Requirement analysis is being done to achieve this goal by specifying and tackling the functional requirements, non-functional requirements, software requirements, and hardware requirements.

The subsequent sections will also outline the overall and technical design of the mobile app. This subsequent section will detail the application's design of data flow, entity-relationship diagrams, data dictionaries and wireframes. These elements form the foundation for building an application that meets the specific needs of the student election process.

3.2 Methodology

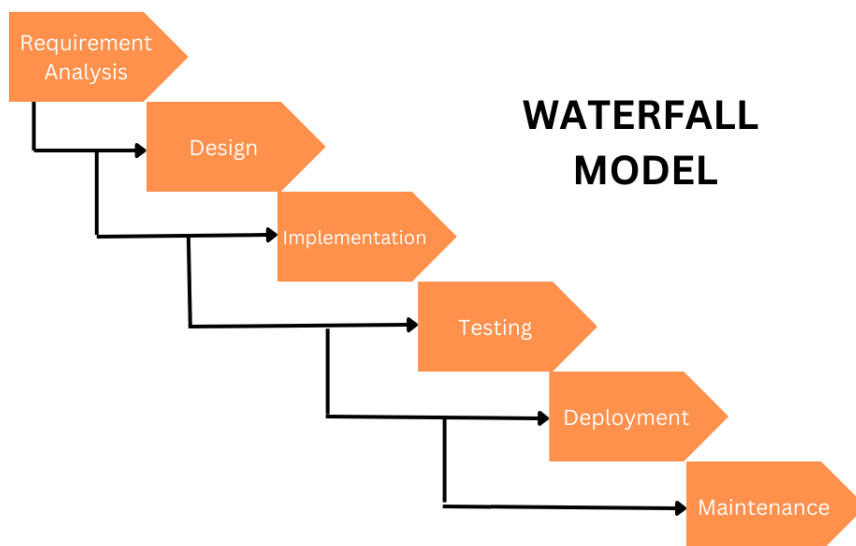


Figure 4.1. Waterfall Model

The Waterfall Model is implemented to develop this mobile app voting platform. The Waterfall model or also referred to as the Linear Sequential Life Cycle Model is chosen due to its linear and sequential nature, where each phase must be completed before proceeding to the next (Khan, 2023). This method provides a clear structure for the software development

process. The key phases include requirement analysis, system design, implementation, testing, deployment, and maintenance. By following to this systematic approach, the Waterfall model ensures that the software is developed in a predictable, controlled environment, helping to meet quality standards, user requirements, and project constraints such as scope, time, and budget. This methodology reduces risks and enhances efficiency, providing a reliable, secure, and user-centric software solution.

3.2.1 Requirement Analysis

Requirement analysis is the initial phase where the project's goals, scope, and functional expectations are defined. This phase involves in understanding user needs and documenting what the application must have. For this blockchain-based voting app, this phase document the essential features like secure voter authentication, anonymous voting, and transparent result tallying.

3.2.2 System Design

The system design phase involves creating a detailed blueprint for the application. This includes defining how the various components interacts such as the frontend, backend, and blockchain. Overall, this phase covers the planning of how data flows, how the entities interact with the application, and designing the user interface.

3.2.3 Implementation

The implementation phase translates the designs and plans into a code. During this phase, the application's codebase is developed, with the frontend built aim to provide a friend user interface, smart contracts written to enable secure and transparent voting processes, and the backend is implemented to facilitate interactions between the user interface and the blockchain.

3.2.4 Testing

The testing phase focuses on ensuring the application functions as intended and meets the required standards. For this app, there are several key areas needed to be tested, such as unit testing for individual components, integration testing to verify interactions between modules, and security testing to protect sensitive voting data, performance testing ensures the application can handle high user loads, and blockchain-specific testing is a critical part of this phase, focusing on validating the functionality, reliability, and security of smart contracts.

3.2.5 Deployment

The deployment phase involves releasing the application to a live environment where users can access and utilize it. Smart contracts are deployed to the Ethereum blockchain, and the application is made available on mobile platforms.

3.2.6 Maintenance

The maintenance phase ensures the application remains functional, secure, and up-to-date over time. This involves monitoring performance, addressing bugs or issues, and incorporating improvements based on user feedback. Maintenance also includes scaling the system to handle future elections and implementing updates to adapt to evolving requirements.

3.3 Requirements

This section outlines the fundamental requirements necessary for developing a software which are functional requirements, non-functional requirements, software requirements and hardware requirements.

3.3.1 Functional Requirements

a) User Registration and Authentication

Voters must log in using their UNIMAS credentials, which include their matric number and student email. The system will verify their eligibility by checking the matric number and sending a one-time password (OTP) code to their student email. The system ensures that there are no duplicate registrations for the same election.

b) Generate voter public key and private key

Upon successful registration, the app will generate a public and private key pair for each voter. The private key is used to sign the vote transaction to the blockchain by creating a digital signature that proves the voter authorized the transaction, while the public key will be shared openly in the blockchain without linking it back to the voter's identity (Hani et al., 2006). According to Z. Liu et al. (2019), private key is used to generate the public key using a one-way cryptographic function making it impossible to reverse the process. Both keys are only assigned to a legitimate UNIMAS student after a successful registration process. Having this public and private key system is crucial to obtain the voter anonymity feature. This is because, the public key confirms and validate that the votes recorded in the blockchain are casted by a

verified voter. Therefore, voter anonymity can be achieved because the public key does not link back to the voter personal information as mentioned by Hani et al., 2006.

c) Election Management

Administrators will have the ability to create new elections and register candidates for each election. The system will store election details such as the election name, dates, and duration. The system will also store candidate information, including their name, manifesto, and the position they are contesting, ensuring that all necessary data is easily accessible for both administrators and voters.

d) Vote Casting

Voters must be able to cast their votes securely using a friendly user interface. The system will ensure that each voter can only cast an allocated number of votes for the election. Additionally, the app will prevent double voting, ensuring that each voter can only vote an allocated number of times.

e) Smart Contracts

A smart contract is deployed on the blockchain (Taherdoost, 2023). It allocates a certain number of votes to each voter and ensures that each voter has not voted yet to prevent double voting. Once the voter casts their vote, the smart contract records it securely on the blockchain, guaranteeing the vote is counted and cannot be altered. It can be said that the contract between the voter and the voting system is to cast an allocated number of votes while also checking whether the voter already voted or not.

f) Blockchain Storage

The blockchain is integrated to store all votes in an open-ledger and decentralized way, making sure that once a vote is recorded, it cannot be altered or deleted either by internal attacks as in a centralized system or outside attacks (Sunny et al., 2022). By storing votes in an open ledger, it allows for vote verification and transparency without compromising voter anonymity using the private and public key system. It provides a transparent and auditable log of all transactions.

g) Vote Receipt

The app will generate a receipt to confirm that the vote has been successfully recorded on the blockchain.

g) Voter Anonymity

The app ensures voter anonymity by using the public and private key system. While the public key is used to verify and track votes recorded on the blockchain, it does not link directly to personal information. This preserves voter anonymity.

h) Auditability and Transparency

The public and private key system allows auditors to verify the integrity of votes without revealing voter identities. The integrity of votes includes that it was cast by a UNIMAS student that has bypass the One-Time password verification process and that the votes are not tampered after being submitted. The public key can be used to track votes on the blockchain while the private key can be used to as a prove that the voter authorized the transaction by creating a digital signature.

3.3.2 Non-Functional Requirements

a) Security

Data security is a top priority to prevent unauthorized access to the app and election data. The application must ensure that only UNIMAS students are allowed to register, safeguarding voter anonymity and maintaining election integrity.

b) Reliability

The application must ensure 100% uptime during election periods. This guarantees that voters can access the app without disruptions and participate in the election process without technical issues.

c) Scalability

The app must be capable of handling simultaneous voting activities during peak periods, ensuring smooth operation even when voter turnout is high. Additionally, the system should be able to support a growing number of users without any degradation in performance degradation

d) Usability

The app should provide an intuitive and user-friendly interface for both voters and administrators, making it easy for everyone to navigate. It is essential to offer clear instructions

and helpful error messages to guide users through the process, ensuring they can complete their tasks without confusion.

e) Maintainability

The system should be modular and easily updated with new features or bug fixes to adapt to changing needs. Comprehensive documentation must be maintained to assist future developers in managing and enhancing the system efficiently.

f) Interoperability

To ensure accessibility, the app must be compatible with major web browsers and mobile devices. It should also support integration with university systems, such as student databases, to verify users and streamline the registration process.

3.3.3 Software Requirements

3.3.3.1 Blockchain Platform

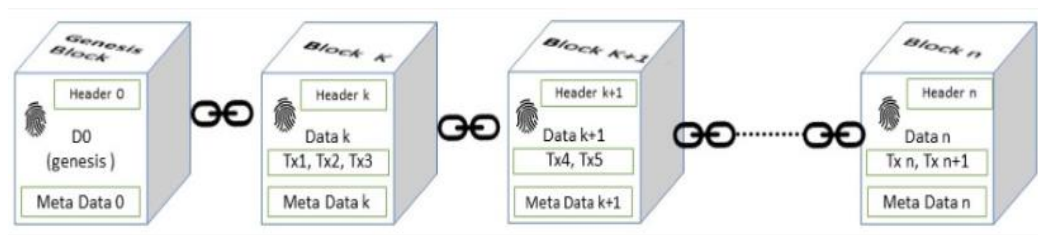


Figure 4.2. A Simplified Structure of Blockchain

Source: (Khan & Byun, 2020)

A blockchain is a type of database that records data in a way that is secure and transparent. In a blockchain, data is grouped into blocks where each block is linked to the one before it, forming a chain of data blocks (Dutta et al., 2020). Every block contains a list of transactions, a timestamp, and a unique identifier called a hash. The hash function to ensure that any attempt to change the data in one block will break the connection to the next block, making it easy to detect tampering (Khan & Byun, 2020). Therefore, this makes blockchain a reliable way to store records, such as votes, financial transactions, or contracts, as it ensures the data cannot be altered without the agreement of most of the network.

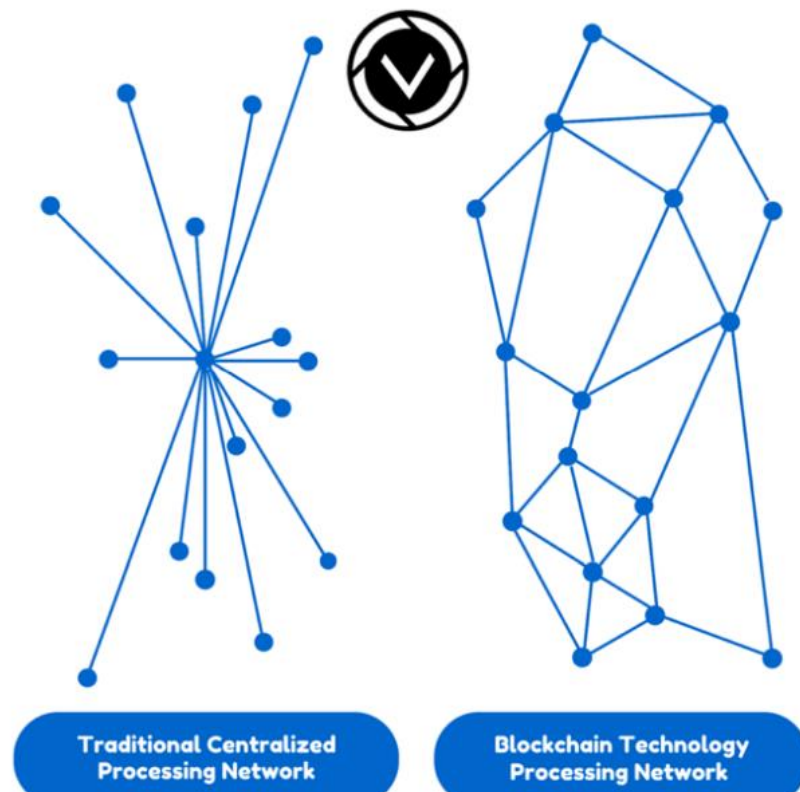


Figure 4.3. Blockchain Decentralize Network

Source: (Follow My Vote, 2020)

Blockchain is a decentralized system, which means it does not rely on a single authority or central server to manage or store data. Instead, it uses a network of computers, called nodes, that work together to validate and record transactions. Each node has a copy of the entire blockchain, ensuring transparency and reducing the risk of data loss or tampering (Jumani et al., 2023). According to Jaoude & Saade (2019) the network functions using a peer-to-peer (P2P) model, where nodes communicate directly with each other rather than through a central server. When a transaction is made, the nodes verify it using a set of rules or smart contracts, and once approved, it is added to a block. These blocks are then linked together in a chain, creating a secure and permanent record of all transactions. This decentralized and P2P nature makes blockchain resistant to tampering and cyberattacks



Figure 4.4. Ethereum Logo

For this voting application, Ethereum blockchain was selected to fulfil the tamper-proof voting storage requirements. Ethereum's decentralized nature ensures that no single party controls the network, eliminating the need for a central authority. This feature significantly reduces the risks of vote storage manipulation and external influence, which are common concerns in centralized or traditional voting systems.

3.3.3.2 FrontEnd



Figure 4.5. React Native Logo

For the frontend of the blockchain-based voting application, React Native is chosen due to its ability to support cross-platform development (Shevtsiv et al., 2019). This means the app can run seamlessly on both iOS and Android devices, making it accessible to a wider audience of UNIMAS students who may be using different platforms. React Native allows for a single codebase to be shared across both platforms, significantly reducing development time while ensuring a consistent user experience across devices.

3.3.3.3 Backend Services

The backend service for the blockchain-based voting application will serve to connect the frontend to the Ethereum blockchain, manages user interactions, and ensures a smooth and secure voting process. Key components of the backend service include:

1. **Node.js**

The backend will use Node.js as the runtime environment, chosen for its efficiency and scalability. Node.js will receive requests from the frontend, such as when users register or cast their votes. It will then process these requests and send responses back to the frontend, like confirming successful registration or vote submission. Node.js will act as the link between the frontend, the blockchain, and other backend processes.

2. **Express.js**

Express.js will be used to build a RESTful API on top of Node.js. A RESTful API (Representational State Transfer) is a client-server communication model that manages data exchange between the client and server. This API will handle essential functionalities especially on user registration.

3. **Solidity**

Solidity programming language will be used to write the smart contracts. The backend will interact with the smart contracts deployed on the Ethereum blockchain. The smart contract will be responsible for crucial functions such as vote casting, and tallying votes. By using Ethereum's smart contracts, the backend ensures that all votes are transparently recorded on the blockchain and cannot be tampered with, providing an added layer of security to the voting process.

4. **Web3.js**

Web3.js will be integrated into the backend service to facilitate communication between the application and the Ethereum blockchain. Web3.js is a JavaScript library that will ensure that the backend can securely interact with the decentralized network, manage vote transactions, and provide access to election results. Web3.js are responsible in generating the private and public key of the voter (Ethereum Foundation, n.d).

3.3.4 Hardware Requirements

Table 4.1. Hardware Requirements

Specification	Requirement
Operating System	iOS 12.0 or upper, Android 8.0 (Oreo) or upper
RAM	2 GB or higher
Storage	100 MB free space (for app installation)
Internet Connectivity	Stable 3G/4G/5G or Wi-Fi

3.4 Design

3.4.1 Context Diagram

A context diagram is a high-level visual representation that outlines the boundaries of this voting app and its interactions with external entities which are the students, admin, and Ethereum blockchain. It essentially provides a bird's-eye view, highlighting who or what the system interacts with.

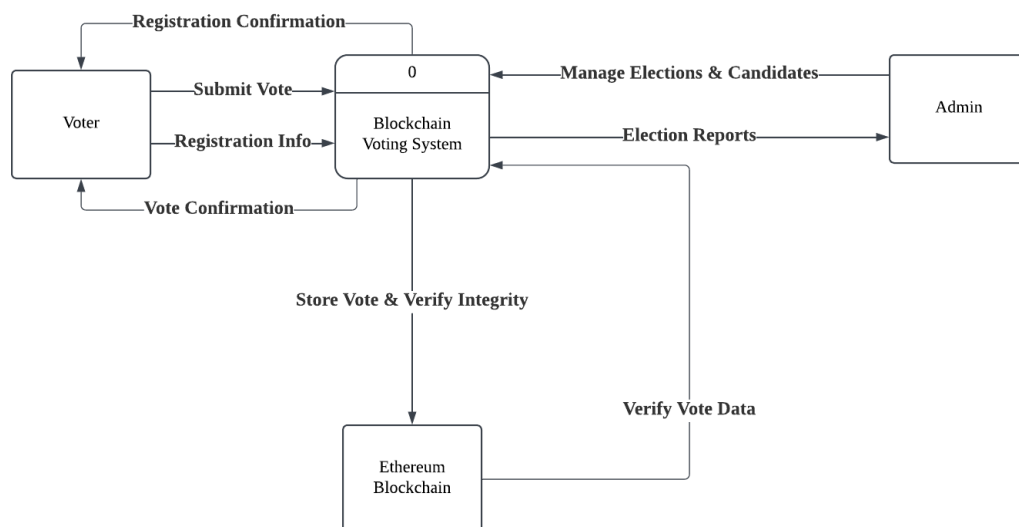


Figure 4.6. Context Diagram

This voting app have three main interactions. These interactions are the interactions between the app and the voter, admin, and Ethereum blockchain. For the interactions between the app and the voter, the app will facilitates voter account registration to verifies their

eligibility as a UNIMAS student to vote. Upon successful vote submission to the blockchain, the system will provide a digital receipt or confirmation to the voter.

Next, the system uses the Ethereum blockchain as a secure and immutable ledger for storing all voting-related data. This integration ensures data integrity and prevents unauthorized modifications. The system interacts with the blockchain to record votes, verify their authenticity, and provide a transparent audit trail.

Lastly, the interaction between the app and the admin are crucial to manage the entire election process. This includes the ability to add or remove candidates, set election details, and obtain election reports. The app provides a centralized platform for administrators to oversee and control the electoral process effectively.

3.4.2 Data Flow Diagrams

A Data Flow Diagram (DFD) is a visual tool that shows how data moves through a system, including inputs, processes, storage, and outputs. DFD Level 0 gives a high-level view of the system as a single process interacting with external entities, while DFD Level 1 breaks this process into smaller parts for more detail.

3.4.2.1 Level 0 DFD

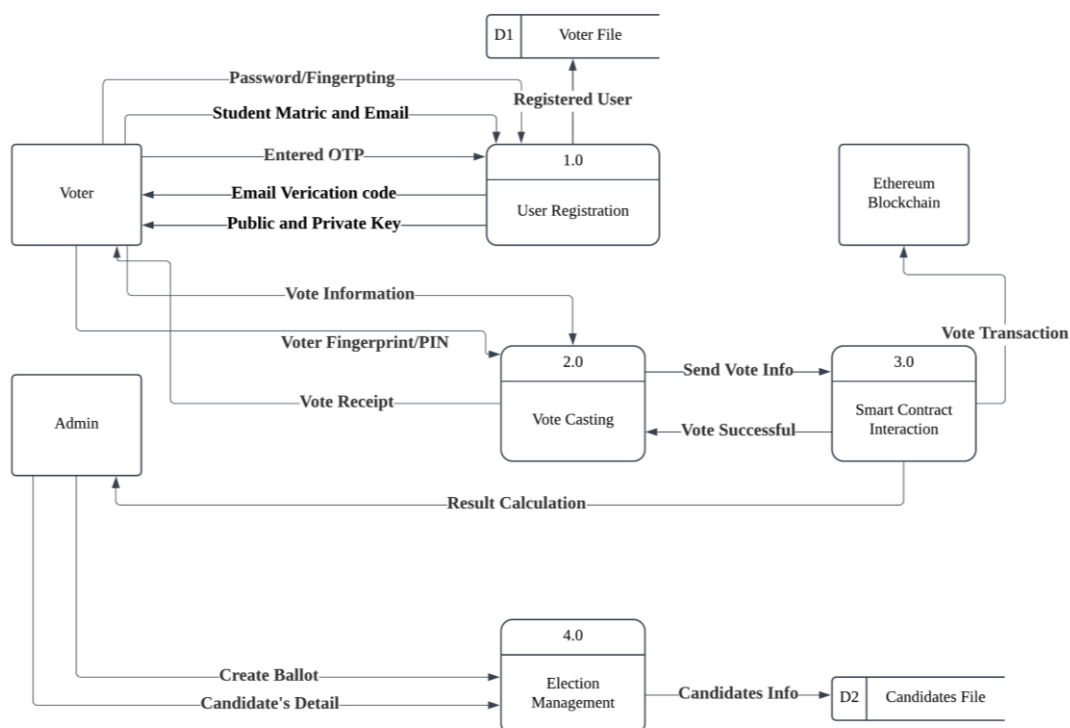


Figure 4.7. Data Flow Diagram Level 0

1. Interaction Between Voter and User Registration (Process 1.0)

The voter begins the registration process by providing their student matric number and UNIMAS email. The system sends a verification code to the provided email, and the voter enters the OTP to confirm their email address. Once verified, the system generates public and private keys for each voter. Voter's details are then stored in the Voter File (D1) as a registered user.

2. Interaction Between Voter and Vote Casting (Process 2.0)

After casting the vote, registered voters will verify the transaction themselves using their fingerprint or PIN. After successful verification, the vote will be submitted to the Ethereum blockchain via the smart contract. As confirmation, the voter receives a vote receipt to ensure their participation recorded

3. Interaction Between Admin and Election Management (Process 4.0)

The admin manages the election by providing details about the ballots and candidates. The system saves the candidates' information in the Candidates File (D2) for reference. Once the voting process is complete, the admin accesses the system to view and manage the results

4. Interaction Between Smart Contract and Ethereum Blockchain

The Smart Contract Interaction (Process 3.0) acts as a secure bridge between the application and the Ethereum blockchain. It records vote transactions on the blockchain to maintain transparency and immutability. The blockchain processes and securely stores the data, making it accessible for auditing and validation.

5. Data Stores

- **Voter File (D1):** Contains information about registered voters, such as matric numbers, emails, and cryptographic keys.
- **Candidates File (D2):** Stores details about the election candidates, including their names and related information.

3.4.3.2 Level 1 DFD

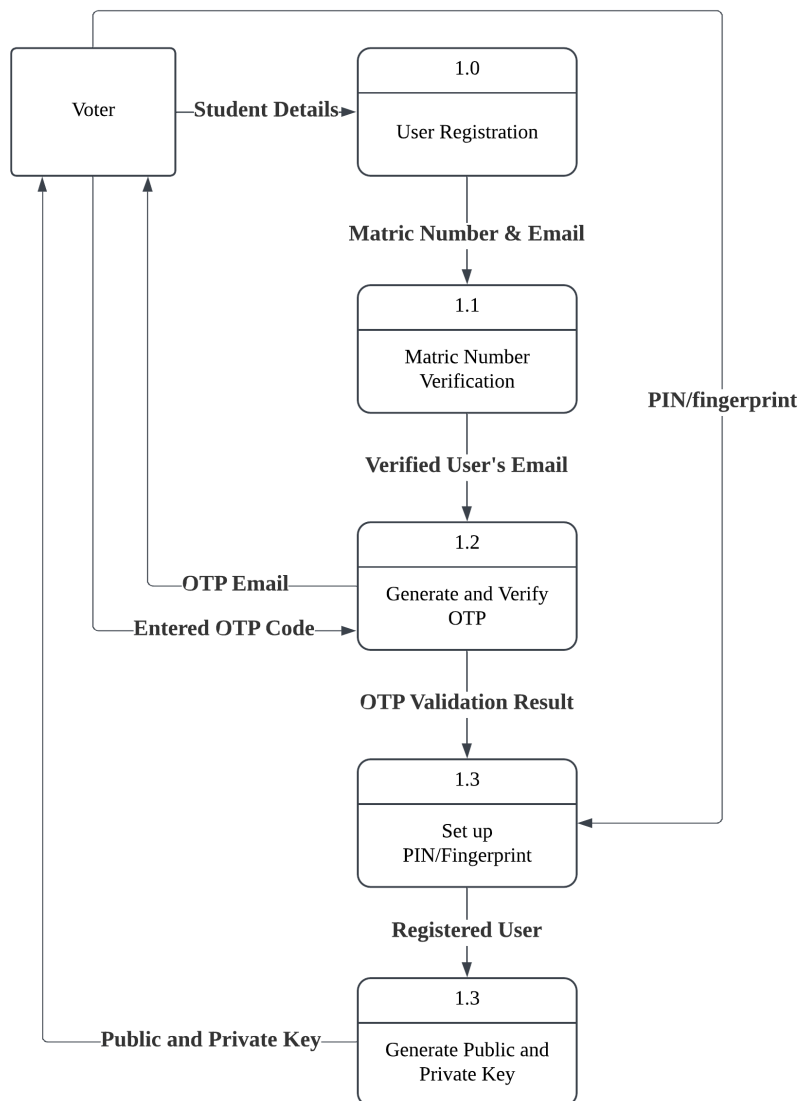


Figure 4.8. Data Flow Diagram Level 1 for User Registration Process

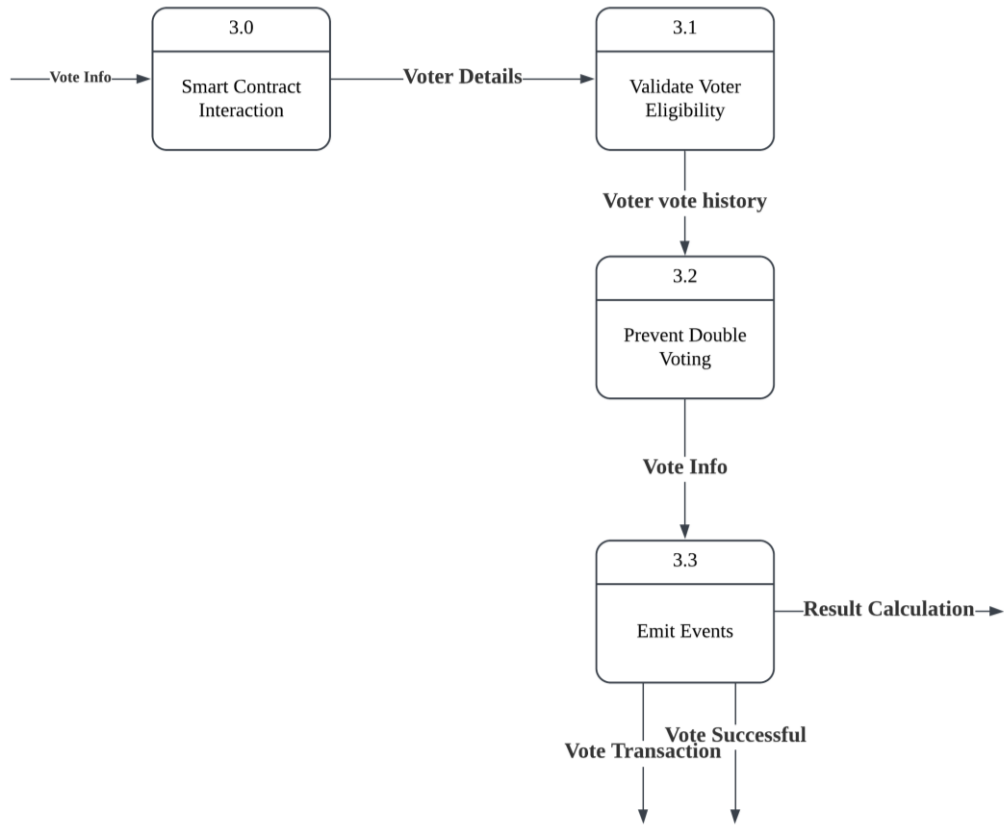


Figure 4.9. Data Flow Level 1 For Smart Contract Interaciton Process

3.4.4 Entity Relationship Diagram

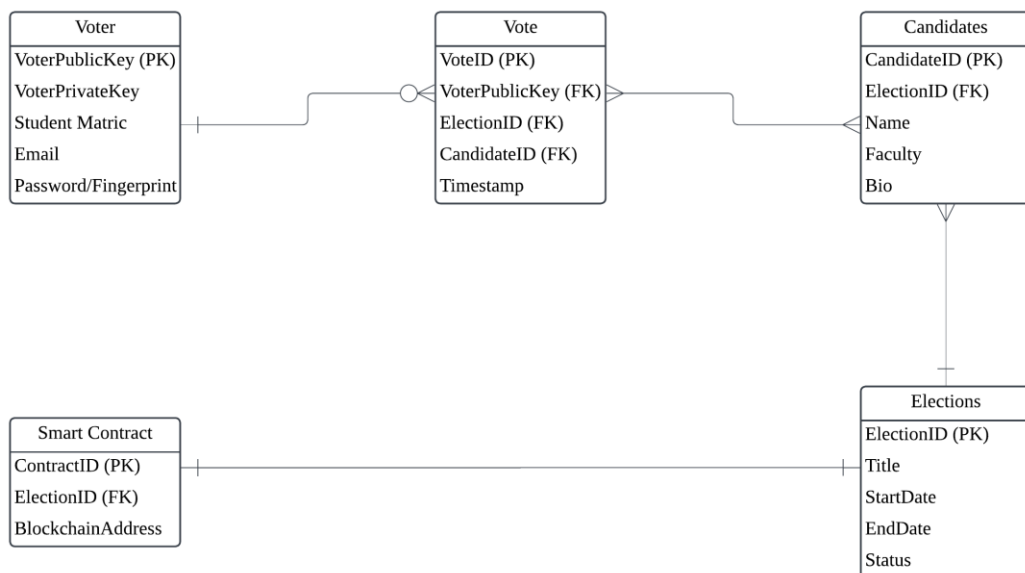


Figure 4.10. Entity Relationship Diagram

The Entity-Relationship Diagram (ERD) for the blockchain-based voting system consists of five main entities: **Voter**, **Vote**, **Candidates**, **Elections**, and **Smart Contract**. Each voter can cast one or more votes (one-to-many relationship), with each vote linked to a single voter. Votes are also associated with specific candidates, where one candidate can receive multiple votes (one-to-many relationship). Each vote belongs to one election, and each election can have multiple votes cast (one-to-many relationship). Candidates are directly linked to elections, where an election can have multiple candidates, but each candidate belongs to a single election (one-to-many relationship). Lastly, each election is managed by a smart contract, and each smart contract corresponds to a single election (one-to-one relationship). This structure ensures data integrity, supports scalability, and leverages blockchain for secure and verifiable voting processes.

3.4.4.1 Data Dictionary

Table 4.2. Data Dictionary for Voter

Field Name	Data Type	Data Size	Description	Constraints
VoterPublicKey (PK)	String	130	Primary key of the voter	Must be unique; cannot be null
VoterPrivateKey	String(Encrypted)	64	Private key of the voter	Confidential; Unique; cannot be null
Student Matric	Int	10	UNIMAS student matric	Must be unique; cannot be null
Email	String	25	UNIMAS student email	Must be unique; cannot be null
Password/Fingerprint	String	64	Hashed version of the voter's password	Cannot be null

Table 4.3. Data Dictionary for Candidate

Field Name	Data Type	Data Size	Description	Constraints
CandidateID (PK)	String	10	Primary key identifier for each candidate	Must be unique; cannot be null
ElectionID (FK)	String	64	Foreign key identifier for the election	Unique; cannot be null
Name	String	10	Name of the candidate	cannot be null
Faculty	String	25	Candidate's UNIMAS faculty	cannot be null
Bio	String	255	Candidates Profile	Cannot be null

Table 4.4. Data Dictionary for Vote

Field Name	Data Type	Data Size	Description	Constraints
voteID (PK)	String	10	Primary key identifier for each vote	Must be unique; cannot be null
VoterPublicKey (FK)	String	130	Foreign key identifier for the for the vote	cannot be null
ElectionID (FK)	String	64	Foreign key identifier for the for election	cannot be null
CandidateID (PK)	String	10	Foreign key identifier for the for candidate	cannot be null

Timestamp	TIMESTAMP	15	Timestamp of when the vote was cast.	Cannot be null
-----------	-----------	----	--------------------------------------	----------------

Table 4.5. Data Dictionary for Election

Field Name	Data Type	Data Size	Description	Constraints
ElectionID (PK)	String	64	Primary key identifier for the for election	cannot be null
Title	String	64	Title or name of the election.	cannot be null
StartDate	Date	10	Start date of the voting period.	cannot be null
EndDate	Date	10	End date of the voting period.	cannot be null
Status	String	15	Status of the election (e.g., 'Open', 'Closed', 'Results Announced').	Cannot be null

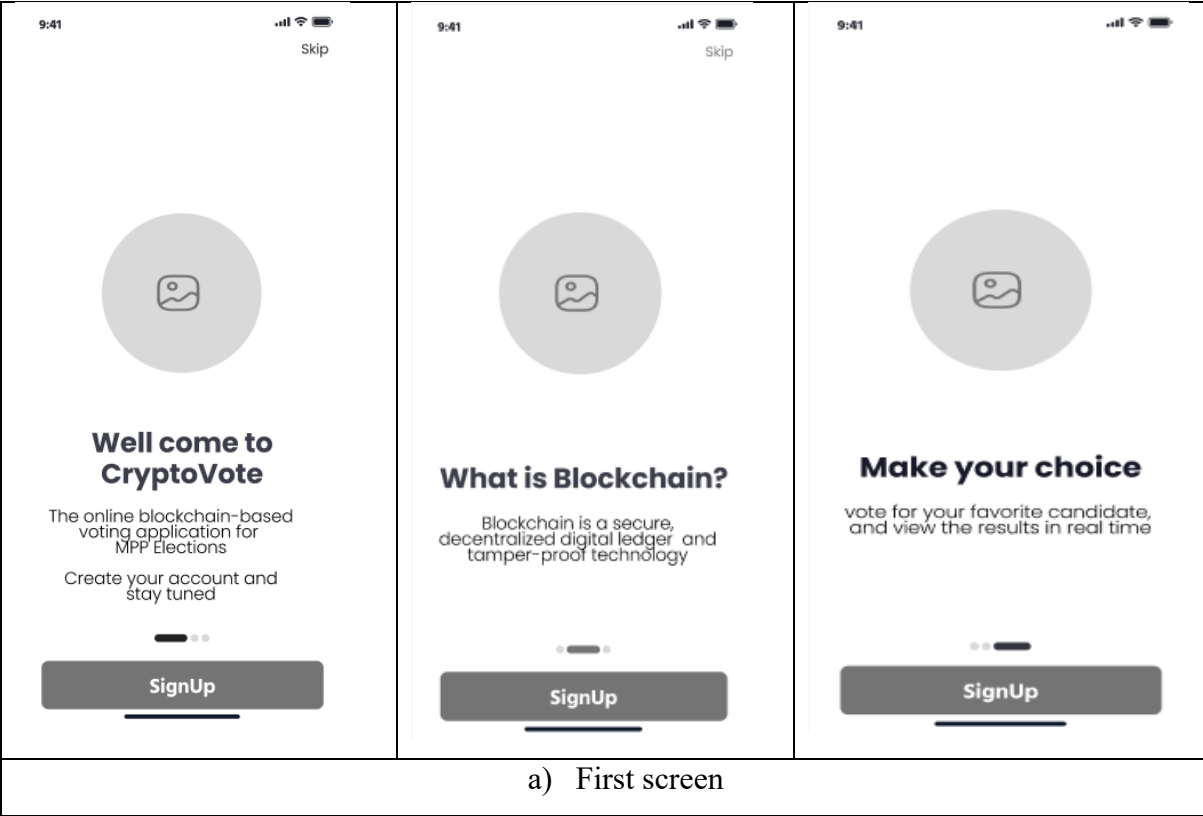
Table 4.6. Data Dictionary for Smart Contract

Field Name	Data Type	Data Size	Description	Constraints
ContractID (PK)	String	64	Primary key identifier for the for smart contract	Unique; cannot be null
ElectionID (PK)	String	64	Primary key identifier for the for election	cannot be null
BlockchainAddress	Date	10	Address of the smart contract	Unique; cannot be null

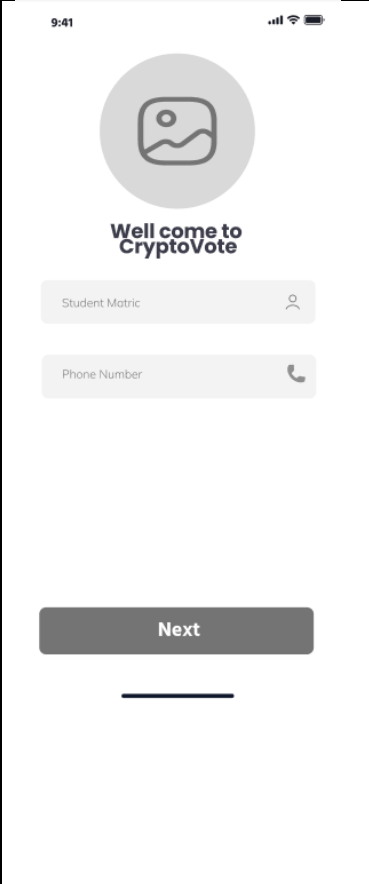
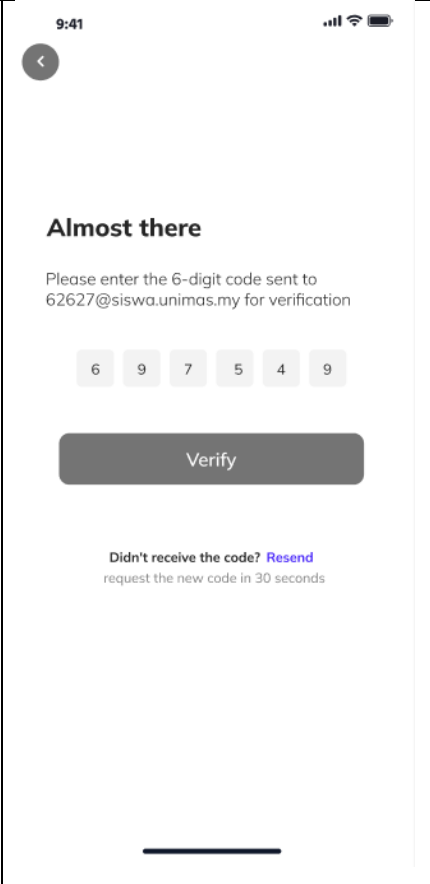
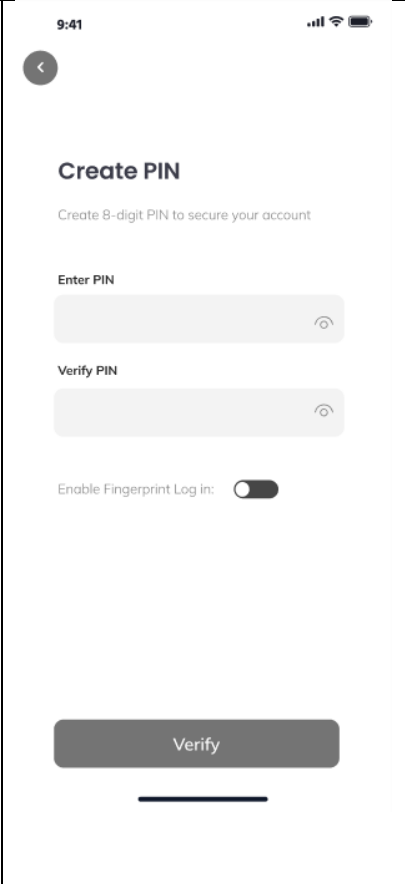
			on the Ethereum blockchain.	
--	--	--	-----------------------------------	--

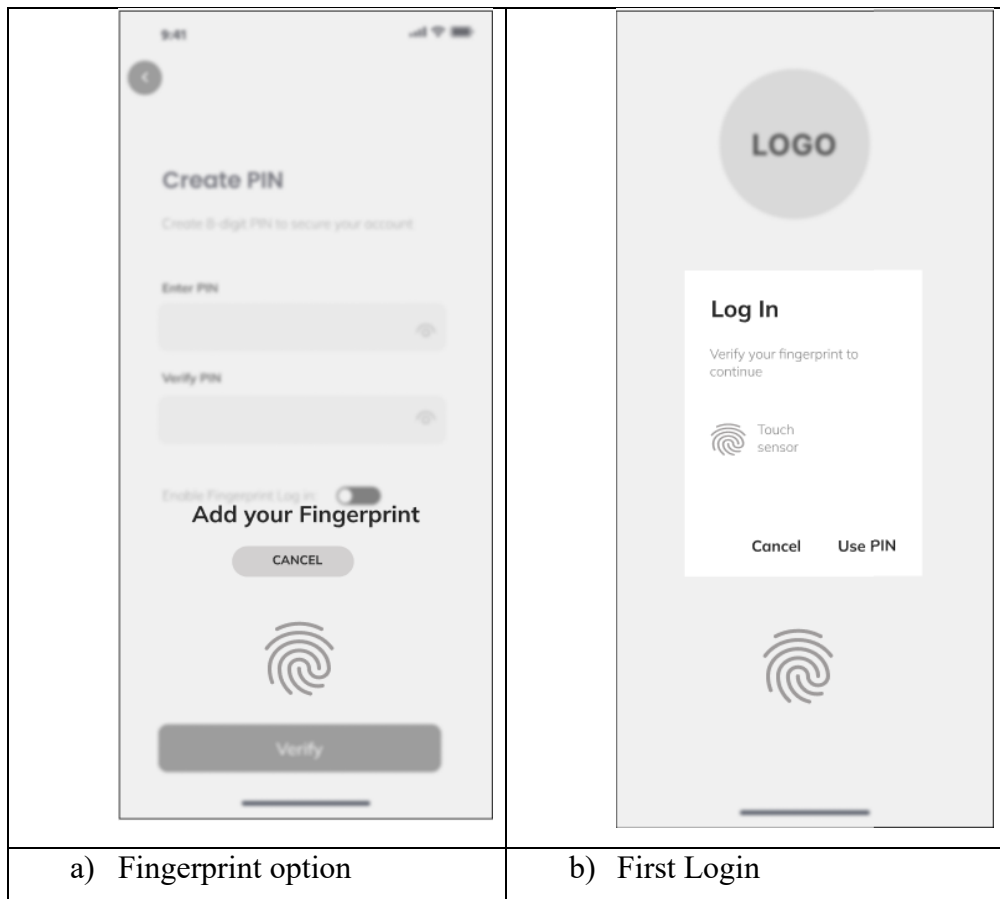
3.4.5 Wireframes

3.4.5.1 Wireframes for Initial Screen

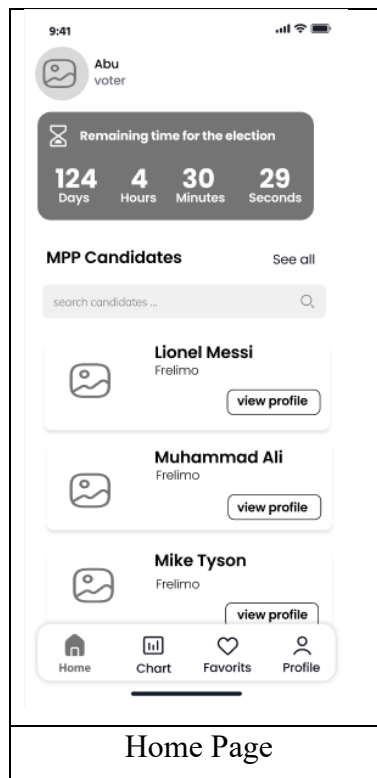


3.4.5.2 Wireframes for User Registration

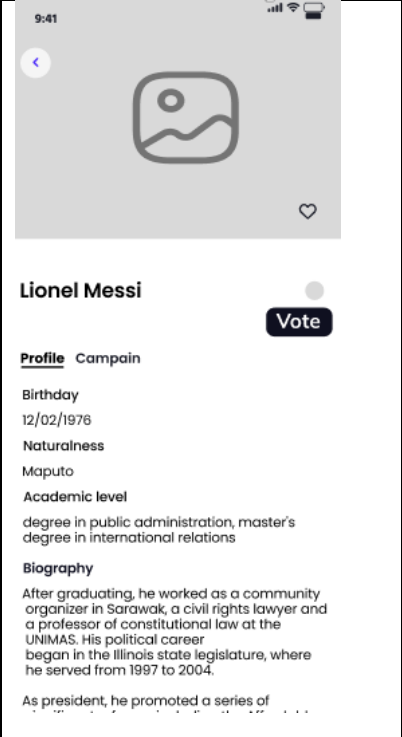
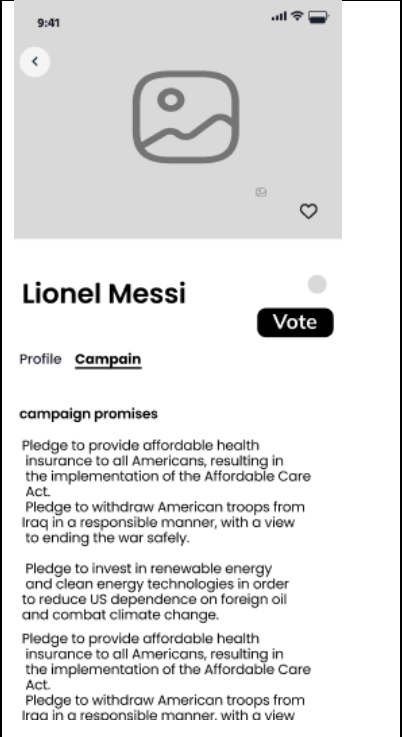
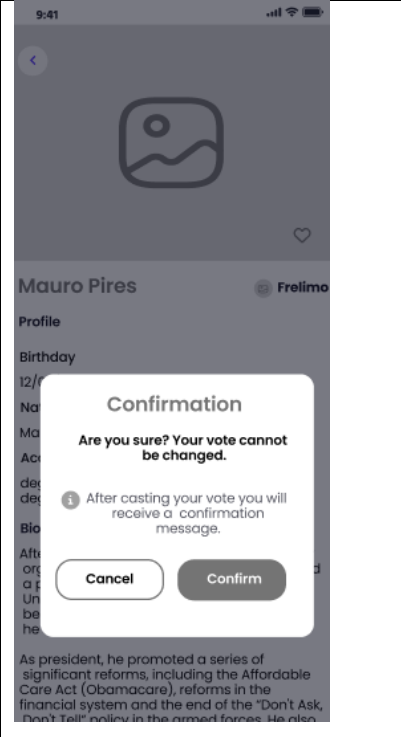
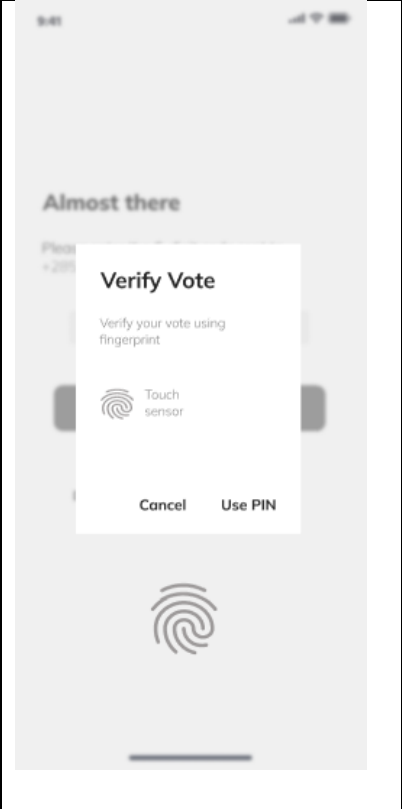
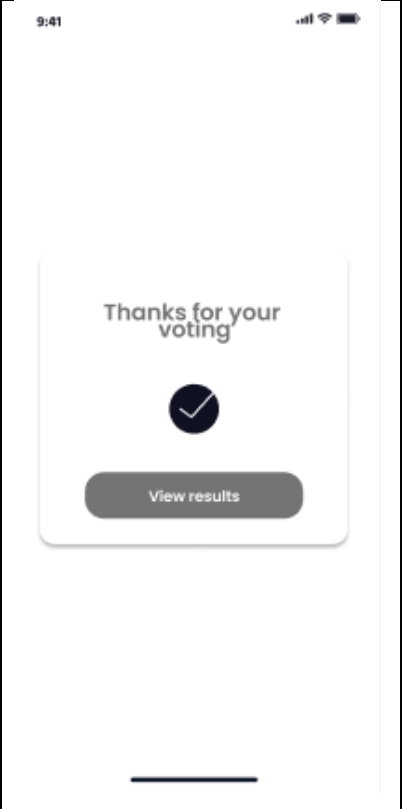
 9:41 Well come to CryptoVote Student Matric Phone Number Next	 9:41 Almost there Please enter the 6-digit code sent to 62627@siswa.unimas.my for verification 6 9 7 5 4 9 Verify Didn't receive the code? Resend request the new code in 30 seconds	 9:41 Create PIN Create 8-digit PIN to secure your account Enter PIN Verify PIN Enable Fingerprint Log in: ☐ Verify
a) Student Matric & Student Email	b) One Time Password send to siswa email	c) Account PIN number



3.4.5.3 Wireframes for Home Page



3.4.5.4 Wireframes for Vote Casting

		
a) Candidate's Profile	b) Candidate's Campaign	c) Vote Confirmation
		
d) Vote Verification	e) Success	

3.5 Summary

The requirement analysis and design of the blockchain-based voting system aim to establish a secure, transparent, and user-friendly platform for conducting university student elections. Key functional requirements include voter registration with UNIMAS credentials, secure vote casting with public and private keys, and the use of smart contracts to record votes on the blockchain. The system ensures data integrity, prevents unauthorized access, and facilitates election management for administrators.

In terms of design, data flow diagrams and entity-relationship diagrams were utilized to outline the interaction between users, processes, and data storage. The application integrates with the university's student database to verify voter eligibility and ensures seamless operation during peak periods through scalable infrastructure. The incorporation of blockchain technology ensures vote immutability and transparency, while off-chain components handle essential tasks like sending OTPs and vote receipts.

Chapter 4: Implementation

4.1 Introduction

The implementation chapter outlines the development process of this blockchain-based voting system. It focuses on the software development tools, technology stack, APIs, frameworks, and core modules used to enable secure, transparent, and verifiable student elections.

This chapter also include the integration of blockchain technology to ensure data integrity, voter anonymity, and tamper-proof vote records. This detailed account of the implementation process aims to offer clear insight into the technical development and decision-making behind the creation of this application.

4.2 Technology Stack

A technology stack refers to the set of programming languages, frameworks, libraries, databases, and tools used to develop and run an application. It includes both frontend and backend technologies, as well as supporting platforms and environments that power the entire development and deployment process.

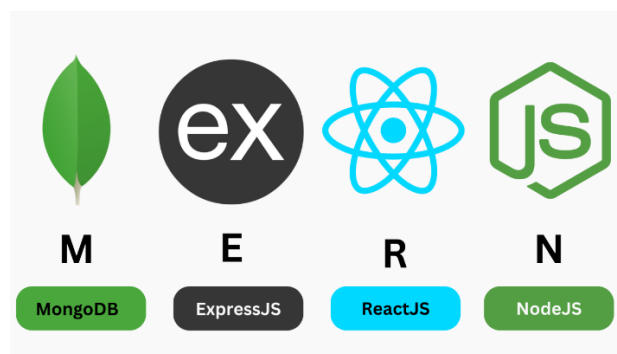


Figure 5.1. Core Technology Stack

This application is developed using the MERN stack, which includes MongoDB for the database, Express.js as the backend framework, React.js for the frontend framework, and Node.js as the runtime environment. This stack provides a unified JavaScript-based development environment, enabling efficient handling of both frontend and backend or server-side logic. Its flexibility and compatibility with blockchain software development tools make it an ideal choice for building a decentralized application.



Figure 5.2. Solidity and Hardhat

In addition to the MERN stack, Solidity is used to develop the smart contract that control the voting logic on the blockchain. Solidity is a high-level programming language specifically designed for writing smart contracts on Ethereum and other EVM-compatible blockchains (Solidity Documentation, n.d.). In addition to Solidity, Hardhat is used specifically for deploying and testing Ethereum smart contracts. It provides tools to compile contract, run tests, debug contract, and deploy to local or public blockchain networks. Hardhat also supports verifying contracts on Etherscan, which is a blockchain explorer that allows users to view and verify Ethereum transactions, smart contracts, and wallet activity in a transparent and accessible way.



Figure 5.3. Sepolia Ethereum Testnet

Sepolia is an official Ethereum test network designed to simulate the Ethereum mainnet environment. It is maintained by the Ethereum Foundation and serves as a stable platform for developers to test their decentralized applications (dApps) and smart contracts before deploying them to the main network. To interact with smart contracts on Sepolia, developers use Sepolia ETH, a test version of Ether (ETH) that has no real-world value. Sepolia ETH is used to pay for gas fees during transactions, simulating the behavior of the Ethereum mainnet.



Figure 5.4. Crypto Wallet MetaMask

To enable users to interact with the smart contract, Metamask app is integrated into the user workflow. MetaMask is a crypto wallet that allows users to connect to the blockchain, sign transactions with their private key, and store digital assets such as crypto currency. It acts as a bridge between this application and the Ethereum blockchain, enabling seamless interaction with the smart contract for actions such as voting, registering, and viewing results.

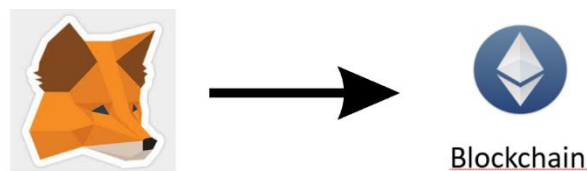


Figure 5.5. MetaMask as a bridge to blockchain

Metamask also handles the public-private key system essential for signing and authorizing blockchain operations, ensuring secure and verifiable interactions between the user and CrptoVote. Every vote cast is a blockchain transaction initiated through MetaMask. A transaction is a piece of data signed by user's private key that changes the state of the blockchain (Capece, Ghiron, & Pasquale, 2020). In the context of this application, state changes refer to the voting actions such as recording a vote on a specific candidate, or adding a poll by admin. Since voting involves updating data on the smart contract, which is deploy to the blockchain, it requires a transaction to be sent to the blockchain network.

4.3 Installation

This section outlines the essential tools and dependencies required to set up the development environment for CryptoVote. It includes core software tools, followed by the dependencies used for developing the smart contract, frontend, and backend.

4.3.1 Software Tools



Figure 5.6. Node.js

- **Node.js**

Node.js is a JavaScript runtime environment used to run the backend of the application and manage dependencies through npm (Node Package Manager). It is essential for running the development server and compiling frontend resources.

Version used: 18.20.8



Figure 5.7. Node Package Manager (NPM)

- **Npm (Node Package Manager)**

Npm is a package manager that comes with Node.js and is responsible for installing, updating, and managing project dependencies. Within the CryptoVote project, npm is used to install essential libraries and tools required for both frontend and backend development. It also facilitates the integration of blockchain functionalities by managing related modules and scripts.

Version used: 10.8.2



Figure 5.8. Visual Studio Code

- **Visual Studio Code**

Visual Studio Code was used as the primary code editor for developing the CryptoVote system. It provided an efficient environment to write, debug, and manage both the frontend and backend code. It offers wide range of extensions and support for javascript, solidity, and other framework.

4.3.2 Smart Contract Dependencies

These dependencies are required to develop, test, and deploy the smart contracts that power the voting logic on the blockchain.

- **@openzeppelin/contracts**

This library provides a set of reliable and secure smart contract components that are used in smart contract developments. It includes many helpful tools for building contracts more efficiently. In CryptoVote, the library is used to handle numeric counters through the counters utility, which helps track values that change as a result of blockchain transactions such as assigning a unique ID each time a poll or candidate is added.

- **@nomicfoundation/hardhat-toolbox**

This is a toolkit that hardhat provides to support for building, testing, and deploying smart contract. Hardhat, a popular smart contract development framework provides essential features that allow developers to write and execute tests, identify and debug errors, simulate contract behavior on a local blockchain, and deploy contracts to test or live Ethereum networks. By combining these tools into one package, it simplifies and streamlines the overall smart contract development workflow.

4.3.3 Backend Dependencies

These packages support the backend server, which handles user authentication, database management, email sending, and security.

Server-side dependencies

- **Express.js**
Express.js is responsible for handling HTTP requests and defining backend routes and middleware for server-side logic.
- **cors**
A middleware package that enables Cross-Origin Resource Sharing (CORS). It allows the frontend application and the backend server to communicate securely.
- **cookie-parser**
This package is used to parse cookies from incoming HTTP requests, which is helpful for managing user sessions or authentication tokens stored in cookies.
- **dotenv**
Used to manage environment variables by loading them from a .env file. This is important for securely handling sensitive information such as MongoDB database connection strings and API key.

Database Dependencies

- **mongoose**
An Object Data Modeling (ODM) library for MongoDB. It provides a structured and simplified approach to defining data models and interacting with the database using JavaScript.

Authentication & Security Dependencies

- **@simplewebauthn/server**
A library that supports the server-side implementation of WebAuthn, a modern standard for secure and passwordless authentication using biometrics or hardware tokens.
- **bcryptjs**
A library used to hash user passwords before storing them in the database. This ensures that plain-text passwords are never saved, enhancing overall system security.

- jsonwebtoken

Jsonwebtoken is used for the creation and verification of JSON Web Tokens (JWTs). These tokens are used to manage user authentication and maintain secure access to protected routes such as home page or create password page.

Email Sender Dependencies

- nodemailer

A module that allows the application to send email messages via SMTP or other transport methods. It is commonly used to handle automated email workflows during user account management processes.

4.3.4 Frontend Dependencies

These dependencies power the frontend of CryptoVote, offering UI interactivity, state management, styling, routing, and blockchain integration.

React Framework

- react & react-dom

Core libraries for building and rendering the user interface. React handles the logic and structure of components, while react-dom is responsible for rendering these components in the browser.

- react-router-dom

Enables frontend navigation within the application. For example, manage navigation between voting pages, the home page, and the login page.

State Management

- react-redux & @reduxjs/toolkit

For global state management across components (e.g., storing user authentication status or connected wallet info).

- zustand

A lightweight alternative to Redux for managing component-level state.

Blockchain Integration

- ethers

A JavaScript library used to connect the frontend to Ethereum smart contracts. It allows users to sign transactions and read blockchain data through wallets like MetaMask.

API Communication

- axios

Handles HTTP requests to backend APIs, such as sign up, logging in, or reset password.

Styling and UI Components

- tailwindcss

A CSS framework used for designing responsive and consistent user interfaces with minimal custom CSS.

- react-icons & lucide-react

Icon libraries that provide scalable icons, used to enhance the visual clarity and design of interface elements.

User Feedback

- react-hot-toast & react-toastify

Used to display notifications to send message to user or confirm user actions. For example, a message like “Vote submitted” or “Wallet connected”.

Biometrics Authentication

- @simplewebauthn/browser

WebAuthn browser library that enables secure user authentication using biometric devices or hardware keys, improving security and user trust.

4.4 Smart Contract Development and Deployment to Blockchain

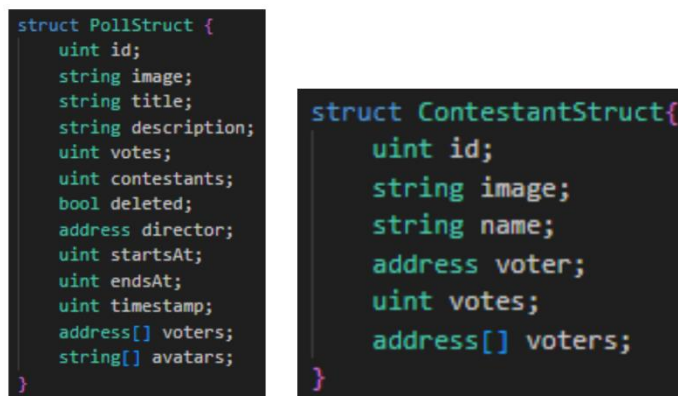
This section outlines the development and deployment process of the smart contract that powers the core functionalities of the CryptoVote system. The contract is written in Solidity and is responsible for managing polls, registering contestants, handling user votes, and maintaining the integrity of the voting process on the blockchain. Subsection 4.4.1 explains the design of the smart contract, including its data structures, state variables, and logical components such as poll creation, contestant registration, and voting mechanisms.

Subsection 4.4.2 focuses on the deployment of the smart contract to the Sepolia test network using the Hardhat development environment. This includes configuration of the Hardhat tool, execution of deployment scripts, and verification of the contract on the blockchain.

4.4.1 Solidity Code

Section 4.4.1 presents the structure and logic of the Solidity smart contract, detailing its data models, variables, and core functions for managing polls, contestants, and the voting process within the CryptoVote system.

4.4.1.1 Smart Contract Data Structure



```
struct PollStruct {
    uint id;
    string image;
    string title;
    string description;
    uint votes;
    uint contestants;
    bool deleted;
    address director;
    uint startsAt;
    uint endsAt;
    uint timestamp;
    address[] voters;
    string[] avatars;
}

struct ContestantStruct{
    uint id;
    string image;
    string name;
    address voter;
    uint votes;
    address[] voters;
}
```

Figure 5.9. Poll & Contestant Struct

The core data structures used in the smart contract are two struct definitions which are PollStruct and ContestantStruct. These structures are used to represent the fundamental entities of the voting system. The PollStruct defines the attributes of a poll. It includes information such as the poll ID, title, image, description, number of votes, number of contestants, poll start

and end times, and the address of the director or creator of the poll. It also keeps a list of voters who have voted in the poll and the image avatars of the contestants. The address type used here stores public wallet addresses on the blockchain. While these addresses are public and used to identify voter participation, they do not expose private keys or personal information. This ensures a secure yet transparent voting process. This struct is essential for storing detailed metadata of each poll in a structured and easily retrievable manner.

Similarly, ContestantStruct defines each contestant participating in a poll. It contains the contestant's ID, name, image, public address of the contestant, total votes received, and an array of voters who voted for that contestant. Like in PollStruct, the public address is stored to identify users, but it is cryptographically impossible to derive their private keys from these addresses. This ensures user privacy while maintaining integrity and verifiability. This struct allows for a transparent tracking of voting outcomes.

4.4.1.2 State Variables and Counters

```
using Counters for Counters.Counter;
Counters.Counter private totalPolls;
Counters.Counter private totalContestants;

mapping(uint => bool) pollExist;
mapping(uint => PollStruct) polls;
mapping(uint => mapping(address => bool)) voted;
mapping(uint => mapping(address => bool)) contested;
mapping(uint => mapping(uint => ContestantStruct)) contestants;

event Voted(address indexed voter, uint timestamp);
```

Figure 5.10. Smart Contract Variables & Counters

The contract declares several state variables and uses OpenZeppelin's Counters utility to manage unique identifiers for polls and contestants. OpenZeppelin is a crucial library for building secure smart contracts on the blockchain. It provides developers with pre-built code components like the Counters utility. The totalPolls and totalContestants counters are used to assign incremental IDs for each new poll and contestant.

Mappings functions as a key-value data store, like a dictionary or hash table. It allows a smart contract to quickly retrieve a specific piece of information from the 'value' by providing its unique identifier the 'key'. Below is the explanation of the key mappings used within the

smart contract, which serve to store and retrieve information related to polls, voters, and contestants.

- **pollExist:** Maps a poll ID to a boolean value, indicating whether a poll exists.
- **polls:** Maps a poll ID to its corresponding PollStruct, allowing direct access to poll details.
- **voted:** A nested mapping of poll ID to voter address, ensuring that each address can only vote once per poll.
- **contested:** Like voted, this mapping ensures that each address can only register as a contestant once per poll.
- **contestants:** A nested mapping of poll IDs to contestant IDs and then to their corresponding ContestantStruct, enabling the retrieval of individual contestant data within a specific poll.

These mappings are foundational to maintaining data consistency and enforcing rules such as one vote per user and single contest entries. They allow the smart contract to efficiently verify eligibility, track participation, and manage data relationships dynamically.

Additionally, the Voted event is declared to emit logs whenever a vote is cast. This Voted event is vital for the frontends to respond in real-time to blockchain changes.

4.4.1.3 Poll Management Functions

```
function createPoll(  
    string memory image,  
    string memory title,  
    string memory description,  
    uint startsAt,  
    uint endsAt  
) public {  
    require(bytes(title).length > 0, 'Title cannot be empty');  
    require(bytes(description).length > 0, 'Description cannot be empty');  
    require(bytes(image).length > 0, 'Image URL cannot be empty');  
    require(startsAt > 0, 'Start date must be greater than 0');  
    require(endsAt > startsAt, 'End date must be greater than start date');  
  
    totalPolls.increment();  
  
    PollStruct memory poll;  
    poll.id = totalPolls.current();  
    poll.title = title;  
    poll.description = description;  
    poll.image = image;  
    poll.startsAt = startsAt;  
    poll.endsAt = endsAt;  
    poll.director = msg.sender;  
    poll.timestamp = currentTime();  
  
    polls[poll.id] = poll;  
    pollExist[poll.id] = true;  
}
```

Figure 5.11. Create Poll Function

The contract includes several functions to manage the poll lifecycle which are creation, updates, deletion, and data retrieval. The createPoll function accepts parameters like title, image, description, and the start and end times for the poll. It performs input validation and ensures logical correctness using the require statement such as the end time must be later than the start time. Upon successful validation, it increments the totalPolls counter, initializes a PollStruct, and stores it in the polls mapping. The poll is marked as active using the pollExist mapping.

```

function updatePoll (
  uint id,
  string memory image,
  string memory title,
  string memory description,
  uint startsAt,
  uint endsAt
) public {
  require(pollExist[id], 'Poll not found');
  require(polls[id].director == msg.sender, 'Unauthorized entity');
  require(bytes(title).length > 0, 'Title cannot be empty');
  require(bytes(description).length > 0, 'Description cannot be empty');
  require(bytes(image).length > 0, 'Image URL cannot be empty');
  require(!polls[id].deleted, 'Polling already deleted');
  require(polls[id].votes < 1, 'Poll has votes already');
  require(endsAt > startsAt, 'End date must be greater than start date');

  polls[id].title = title;
  polls[id].description = description;
  polls[id].startsAt = startsAt;
  polls[id].endsAt = endsAt;
  polls[id].image = image;
}

```

Figure 5.12. Update Poll Function

The `updatePoll` function enforces several conditions to ensure that only valid and authorized updates can be made to an existing poll. It begins by verifying that the specified poll exists and that the caller is the original creator by comparing `msg.sender` to the stored `director` address. Input validation is performed to ensure that the poll's title, description, and image URL are not empty. The function also checks that if the poll has not yet received any votes (`polls[id].votes < 1`) and is not marked as deleted (`!polls[id].deleted`). Lastly, it confirms that the end date is greater than the start date to ensure a valid polling period. Only when all conditions are met will the poll details be updated with the new values.

```

function deletePoll(uint id) public {
    require(pollExist[id], 'Poll not found');
    require(polls[id].director == msg.sender, 'Unauthorized entity');
    require(polls[id].votes < 1, 'Poll has votes already');
    polls[id].deleted = true;
}

function getPoll(uint id) public view returns (PollStruct memory){
    return polls[id];
}

function getPolls() public view returns (PollStruct[] memory Polls) {
    uint available;
    for (uint i = 1; i <= totalPolls.current(); i++){
        if(!polls[i].deleted) available++;
    }

    Polls = new PollStruct[](available);
    uint index;

    for (uint i = 1; i <= totalPolls.current(); i++){
        if(!polls[i].deleted){
            Polls[index++] = polls[i];
        }
    }
}

```

Figure 5.13. Delete & Get Poll Function

The deletePoll function lets the poll creator flag the poll as deleted, provided no votes have been cast. Instead of removing the poll data, it uses a logical delete by assigning deleted to true, which helps retain historical data for transparency while excluding the poll from active listings.

Lastly, the getPoll function retrieves a single poll based on its ID, whereas getPolls returns an array of all active and non-deleted polls. These functions are marked as view, allowing users and external applications to query poll data without consuming gas or pay transactions.

4.4.1.4 Contestant Management Functions

```
function contest(  
    uint id,  
    string memory name,  
    string memory image)  
public {  
    require(pollExist[id], 'Poll not found');  
    require(bytes(name).length > 0, 'name cannot be empty');  
    require(bytes(image).length > 0, 'image cannot be empty');  
    require(polls[id].votes < 1, 'Poll has votes already');  
    require(!contested[id][msg.sender], 'Already contested');  
  
    totalContestants.increment();  
  
    ContestantStruct memory contestant;  
    contestant.name = name;  
    contestant.image = image;  
    contestant.voter = msg.sender;  
    contestant.id = totalContestants.current();  
  
    contestants[id][contestant.id] = contestant;  
    contested[id][msg.sender] = true;  
    polls[id].avatars.push(image);  
    polls[id].contestants++;  
}
```

Figure 5.14. Contest Function

The contest function allows a user to register as a contestant in a specific poll, but only under strict conditions that maintain the validity and fairness of the system. It first checks whether the poll exists using the provided ID. To ensure data completeness, it verifies that both the contestant's name and image URL are not empty. The function also confirms that the poll has not yet received any votes, preserving the integrity of the candidate list once voting begins. Additionally, it prevents duplicate entries by ensuring the caller has not already registered as a contestant in the same poll. Upon passing all checks, the function increments the global contestant counter, creates a new ContestantStruct with the provided details, assigns it a unique ID, and stores it in the corresponding poll. It also updates the poll's list of contestant images and increments the total number of contestants for that poll.

```

function getContestant(
    uint id,
    uint cid
)
public view returns (ContestantStruct memory){
    return contestants[id][cid];
}

function getContestants(
    uint id
)
public view returns (ContestantStruct[] memory Contestants) {
    uint available;
    for (uint i = 1; i <= totalContestants.current(); i++){
        if(contestants[id][i].id == i) available++;
    }

    Contestants = new ContestantStruct[](available);
    uint index;

    for (uint i = 1; i <= totalContestants.current(); i++){
        if(contestants[id][i].id == i){
            Contestants[index++] = contestants[id][i];
        }
    }
}

```

Figure 5.15. Get Contestants Function

The contract also provides `getContestant` to retrieve details of a specific contestant and `getContestants` to fetch all contestants in each poll. These read-only or view functions which are critical for presenting election choices to users on the front-end without having to pay gas or transaction.

4.4.1.5 Voting Logic

```
function vote(uint id, uint cid) public {
    require(pollExist[id], 'Poll not found');
    require(!voted[id][msg.sender], 'Already voted');
    require(!polls[id].deleted, 'Polling not available');
    require(polls[id].contestants > 1, 'Not enough contestants');
    require(
        currentTime() >= polls[id].startsAt && currentTime() < polls[id].endsAt,
        'Voting must be in session'
    );

    polls[id].votes++;
    polls[id].voters.push(msg.sender);

    contestants[id][cid].votes++;
    contestants[id][cid].voters.push(msg.sender);
    voted[id][msg.sender] = true;

    emit Voted(msg.sender, currentTime());
}
```

Figure 5.16. Vote Function

The vote function is central to the contract's functionality. It allows a user to cast a vote for a specific contestant in a particular poll. Before a vote is accepted, several conditions are checked:

- The poll must exist and not be flagged as deleted.
- The user must not have voted in the poll previously.
- The poll must have more than one contestant.
- The current time must fall within the poll's active voting window (between startsAt and endsAt).

Once these conditions are met, the poll's total vote count is incremented, and the voter's address is added to the poll's list of voters. Simultaneously, the selected contestant's vote count is increased, and the voter's address is stored in the contestant's voter list. The mapping voted is updated to reflect that the user has already participated.

Finally, the Voted event is emitted, recording the voting activity on the blockchain for transparency and future reference.

This voting logic guarantees a fair election process with a one-person-one-vote principle, preserves voter privacy by using public addresses, and ensures data immutability after interaction.

4.4.2 Smart Contract Deployment to Sepolia Blockchain

After completing the development of the CryptoVote smart contract, the next step was to deploy it to a live test network to validate its functionality in a real blockchain environment. This section outlines the deployment process to the Sepolia Ethereum testnet, which allows for testing and interaction without needing to use real crypto currency to interact with the blockchain

To deploy the smart contract to the Sepolia test network, several tools and services were utilized, including Hardhat, Alchemy, and Etherscan. Alchemy is a blockchain development platform that provides reliable access to Ethereum blockchain nodes, allowing the deployment script to communicate with the Sepolia Ethereum network. Etherscan, on the other hand, is a widely used Ethereum blockchain explorer that also supports smart contract verification, enabling the public to view the contract's source code and transaction history. The following subsection details the scripting of the deployment logic using these tools.

4.4.2.1 Hardhat Network Configuration

```
require('@nomicfoundation/hardhat-toolbox')
require('dotenv').config()

const ALCHEMY_API_KEY = process.env.ALCHEMY_API_KEY
const SEPOLIA_PRIVATE_KEY = process.env.SEPOLIA_PRIVATE_KEY
const ETHERSCAN_API_KEY = process.env.ETHERSCAN_API_KEY

module.exports = {
  networks: {
    sepolia: {
      url: `https://eth-sepolia.g.alchemy.com/v2/${ALCHEMY_API_KEY}`,
      accounts: [SEPOLIA_PRIVATE_KEY],
    },
  },
  solidity: {
    version: '0.8.17',
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
  etherscan: {
    apiKey: ETHERSCAN_API_KEY,
  },
};
```

Figure 5.17. Hardhat Network Configuration Script

To deploy the CryptoVote smart contract to the Sepolia test network, the Hardhat configuration file was set up with essential network and verification details to successfully deploy the smart contract. The configuration begins by defining three key constants. First is the ALCHEMY_API_KEY, which is used to establish a connection to the Sepolia test network.

Next is `SEPOLIA_PRIVATE_KEY`, which refers to the private key of a crypto wallet account that will be used to deploy the contract. Lastly, `ETHERSCAN_API_KEY` is an API provided by Etherscan to enable access to verification of smart contracts deployment on Etherscan platform.

Within the `module.exports` block, the `networks` object specifies the configuration for Sepolia. This includes an Alchemy RPC endpoint URL that connects to the Sepolia blockchain. An RPC (Remote Procedure Call) endpoint allows the application to send requests to and receive data from the blockchain such as deploying contracts or reading transaction data. The `accounts` array contains the private key responsible for deploying the smart contract and paying for the associated gas fees. The `solidity` field sets the compiler version to 0.8.18, ensuring that the contract is compiled using the correct version of the Solidity language.

Finally, the `etherscan` section allows the deployed contract to be verified on Etherscan. Verification makes the contract's source code publicly available, which increases trust and lets users interact with the contract directly from the Etherscan interface.

4.4.2.2 Deployment Script and Execution

After setting up the Hardhat configuration, the next step is to deploy the smart contract to the Sepolia test network. With the configuration in place, the smart contract was deployed to the Sepolia test network using a Hardhat deploy script. The following is the Hardhat deploy script:

```
const { ethers } = require('hardhat')

async function main() {
  const contract_name = 'cryptoVotes'
  const Contract = await ethers.getContractFactory(contract_name)
  const contract = await Contract.deploy()

  await contract.deployed()

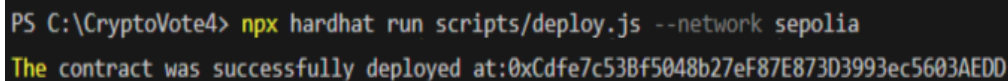
  console.log('The contract was successfully deployed at: ', contract.address)
}

main().catch((error) => {
  console.log(error)
  process.exitCode = 1
})
```

Figure 5.18. Hardhat Deploy Script

The `deploy.js` script is responsible for deploying the smart contract to the Sepolia blockchain network. The `main()` function begins by specifying the name of the contract which

is `cryptoVotes` and retrieving a contract factory using the line `ethers.getContractFactory(contract_name)`. This factory is used to create an instance of the smart contract. The line `const contract = await Contract.deploy()` sends the smart contract to the blockchain for deployment. The script then waits for the deployment to complete using `await contract.deployed()`. Once successful, it prints the contract address to the terminal using `console.log('The contract was successfully deployed at: ', contract.address)`, confirming that the contract is live on the Sepolia network.



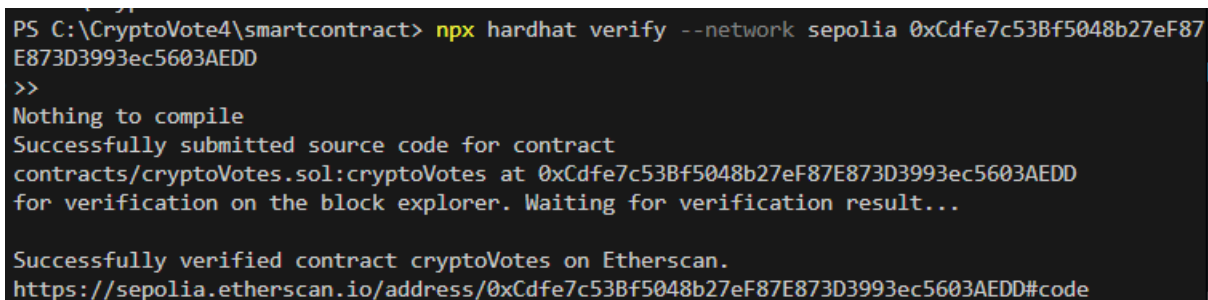
```
PS C:\CryptoVote4> npx hardhat run scripts/deploy.js --network sepolia
The contract was successfully deployed at:0xCdfe7c53Bf5048b27eF87E873D3993ec5603AEDD
```

Figure 5.19. Hardhat Deploy Command

To execute the deployment of the smart contract to the Sepolia test network, the following command was run in the terminal: `npx hardhat run scripts/deploy.js --network sepolia`. This command triggers Hardhat to execute the `deploy.js` script using the previously defined Sepolia network configuration in subsection 4.4.2.1. Upon successful deployment, the terminal outputs the address of the deployed contract on the Sepolia blockchain. This address serves as the unique chain location of the smart contract, allowing it to be accessed by both frontend applications and blockchain explorers such as Etherscan

4.4.2.3 Contract Verification

After successfully deploying the smart contract to the Sepolia test network, the next step involved verifying the contract on Etherscan. Verification allows the source code of the smart contract to be made publicly viewable, ensuring transparency and enabling interaction through the blockchain explorer interface. The verification was performed using Hardhat command plus the deployed contract address which are obtained in subsection 4.4.2.2 Deployment Execution. Figure 48 below shows the terminal output after running the command:



```
PS C:\CryptoVote4\smartcontract> npx hardhat verify --network sepolia 0xCdfe7c53Bf5048b27eF87E873D3993ec5603AEDD
>>
Nothing to compile
Successfully submitted source code for contract
contracts/cryptoVotes.sol:cryptoVotes at 0xCdfe7c53Bf5048b27eF87E873D3993ec5603AEDD
for verification on the block explorer. Waiting for verification result...

Successfully verified contract cryptoVotes on Etherscan.
https://sepolia.etherscan.io/address/0xCdfe7c53Bf5048b27eF87E873D3993ec5603AEDD#code
```

Figure 5.20. Hardhat Contract Verification on Etherscan

This command sends the smart contract's code and metadata to Etherscan using the deployed address as a reference, so it can be checked against the version already deployed at the sepolia blockchain address. If everything matches, Etherscan accepts the verification, and the terminal shows a message confirming that the contract has been successfully verified. The verified contract can be publicly accessed at:

<https://sepolia.etherscan.io/address/0xCdfe7c53Bf5048b27eF87E873D3993ec5603AEDD>

After going to the url above, users are able to view detailed information about the deployed CryptoVote smart contract. This includes the contract address, which uniquely identifies the smart contract on the Ethereum blockchain, and the creator address, which shows the public wallet address of the account that deployed the contract. This level of transparency ensures that the contract's origin and ownership can be independently verified. The screenshots below show the verified CryptoVote smart contract on Etherscan, a blockchain explorer for the Ethereum network which contain the contract address on the blockchain and creator public address.

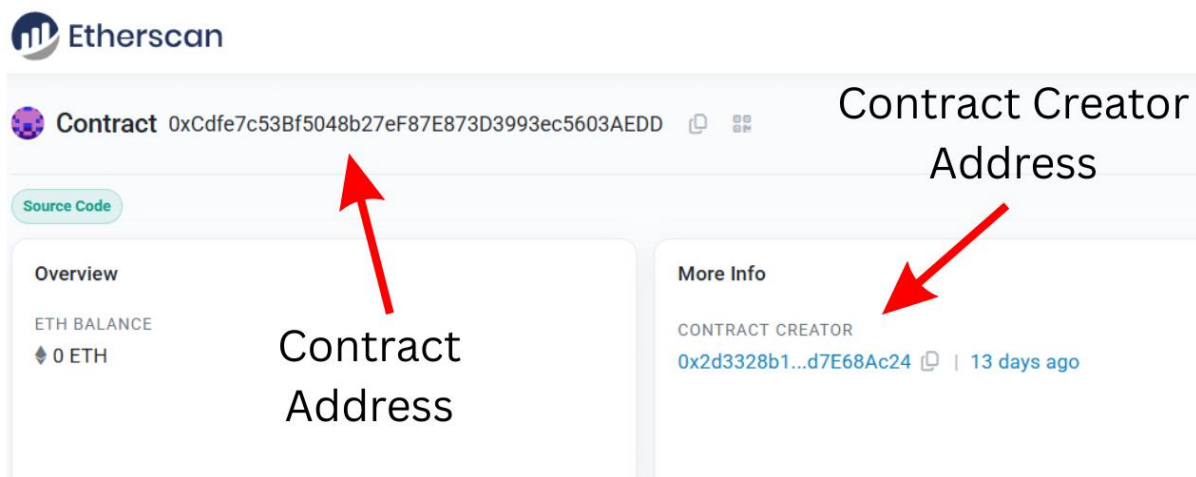


Figure 5.21. Contract Verification on Etherscan

Furthermore, user can also see the transaction history associated with the contract, including interactions such as poll creation, contestant registration, and voting activities in Figure 50 below. These on-chain transactions demonstrate that all contract functions were executed securely and immutably on the blockchain, in line with the system's goal of transparent and verifiable elections. This verification process confirms that the deployed bytecode matches the source code, which enhances user trust and supports external audits or blockchain-based interactions.

in the system, including their methods, purposes, and how they support communication between the frontend and the server.

Table 5.1. List of Backend API Endpoint

HTTP Method	Route	Description
POST	/signup	Handles user registration by accepting sign-up form data.
POST	/verify-email	Verify the OTP code user enter from their UNIMAS Siswa email .
POST	/save-pass	Saves the user's chosen password after email verification.
POST	/init-passkey	Initializes WebAuthn passkey registration if user choose to Enable Biometric during sign up.
POST	/verify-passkey	Verifies the registered WebAuthn passkey.
POST	/login	Processes login requests using email and password.
POST	/logout	Logs the user out and clears their session cookie.
POST	/auth-passkey	Starts the authentication challenge for login using biometric.
POST	/verify-auth-passkey	Verifies the authentication response for WebAuthn login.
POST	/forgot-password	Initiates the password reset process by sending a reset link to their UNIMAS siswa email.

POST	/reset-password/:token	Accepts a new password and resets it using the token from the reset email.
POST	/voteConfirmed	Sends a confirmation email after a successful vote.
POST	/claimToken	Allows users to claim voting tokens from the system.
GET	/check-auth	Verifies if the user is authenticated

4.5.2 Frontend Pages

The frontend of the CryptoVote system is developed using React and is composed of multiple pages that correspond to specific user actions. Each page is linked to a route, and when a user navigates to a route, the appropriate React component is rendered. These pages handle various tasks such as registration, login, password recovery, viewing polls, and voting. Some routes are protected to ensure only authenticated users can access them, while others are accessible to new or unauthenticated users. The table below outlines the main frontend routes and their corresponding components.

Table 5.2. List of Frontend Pages

Route	Component/Page	Description
/	HomePage	Displays the main interface, accessible only to authenticated users.
/signup	SignUpPage	Page for new users to begin the sign-up process.
/verify-email	EmailVerificationPage	Prompts users to enter OTP code send to their UNIMAS email

/create-pass/:token	PassPage	Allows users to create a password after email verification.
/login	LoginPage	Allows existing users to log into their accounts.
/forgot-password	ForgotPasswordPage	Lets users request a password reset link via email.
/reset-password/:toke	ResetPasswordPage	Page for entering a new password using the token from the reset email.
/polls/:id	PollDetailsPage	Displays detailed information about a selected poll and enables voting.

4.6 Sign Up Workflow

The Sign Up workflow outlines the process that every user whether an administrator, contestant, or regular voter must complete to gain access to the CryptoVote system. This workflow ensures a secure and verified registration experience by combining institutional email verification, secure password creation, and optional biometric authentication using WebAuthn library. By applying the same registration process across all user types, the system maintains consistency, strengthens security, and ensures that only authorized individuals can access the platform and participate in its functionalities.

4.6.1 UNIMAS Siswa Email Verification

The sign-up page in the CryptoVote system allows students to register by entering their student id number. The user interface is designed to be simple and focused, containing a single input field and a Sign-Up button.

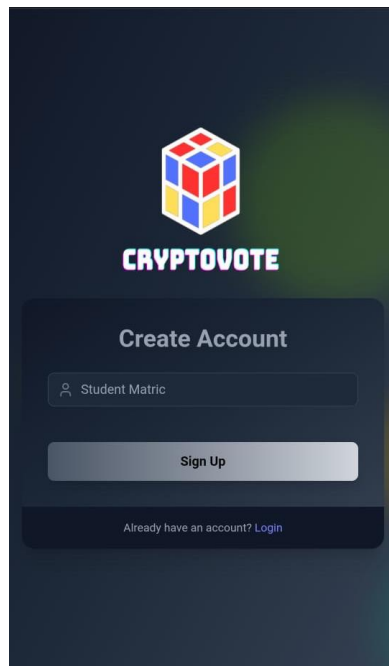


Figure 5.24. Sign In Page

The sign-up process in the CryptoVote system begins when a student enters their matriculation number. Upon clicking the sign-up button, the client side or frontend sends a http post request to the /sign-up API endpoint using the Axios javascripts library. Before this request is made, the frontend performs basic input validation to ensure that the matric number field is not empty as in Figure 51.

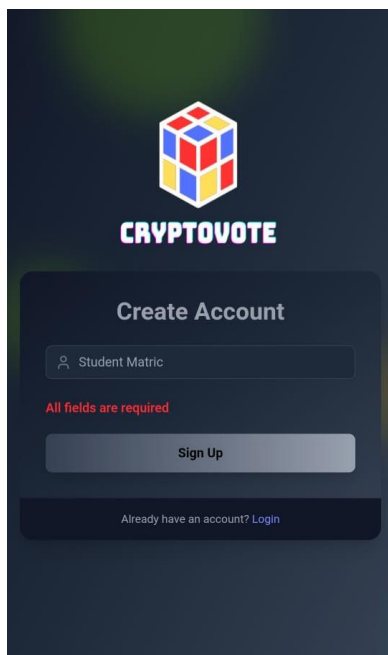


Figure 5.25. Input Validation for Sign In Page

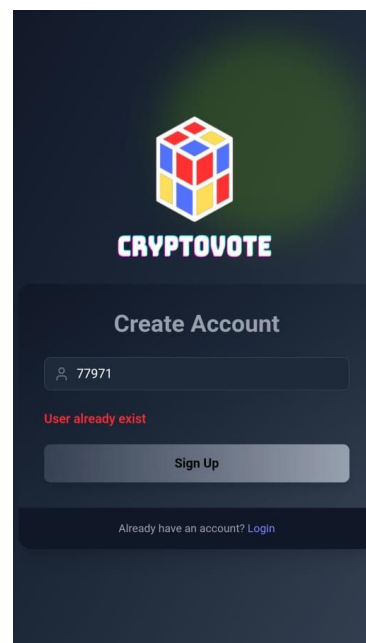


Figure 5.26. User Already Exist Message

Once a valid post request reaches the backend /signup API endpoint, it checks whether a user with the same matric number already exists. If a user with the same matric number is found and their email has already been verified, the server responds with an error message indicating that the user already exists as in Figure 52. Upon success, the server generates a six-digit verification code and sends it to the student's unimas email address to proceed with email verification. If there is any issue receiving the email, the user can simply return to the sign-up page and re-enter their matric number to trigger the process again.

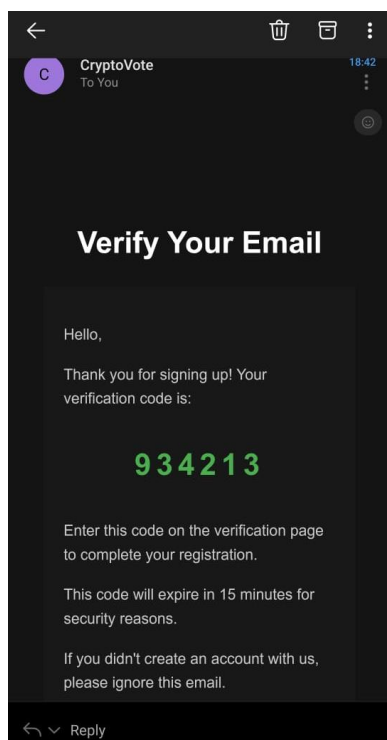


Figure 5.28. Email Verification Code



Figure 5.27. Brevo Email Service

The email is sent using Nodemailer, a Node.js module, which is configured with Brevo's SMTP relay in the /signup backend endpoint to ensure secure and reliable delivery of the verification code. Brevo formerly known as Sendinblue, allows transactional email sending via its SMTP service, which is integrated into the backend. Next, the frontend navigate to Verify Your Email page in Figure 55, prompting the student to enter the verification code they received. This workflow ensures that only legitimate students with access to their unimas siswa email can complete the sign-up process, helping maintain the integrity of the election system.

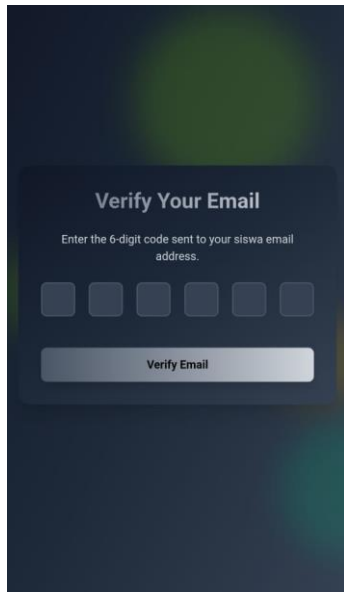


Figure 5.29. Email Verification Page

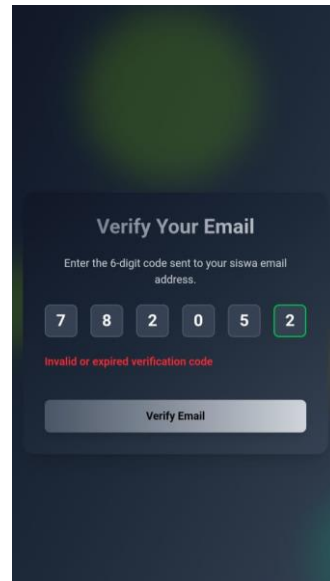


Figure 5.30. Invalid or Expired OTP Message

Once the code is entered and submitted, the frontend sends a http post request to the /verify-email API endpoint. During the signup process, the backend stores the user's information in the MongoDB database, including the verification code and its expiration time. When the verification request is received, the backend checks the database for a user whose verification code matches the submitted code and is still within the valid time window. If a valid match is found, the user's email is marked as verified, and the verification token is removed from the database. If the code is incorrect or expired, error message is displayed to the user as in Figure 56. The next step of user registration is to create password which is detailed in following subsection.

4.6.2 Create Password

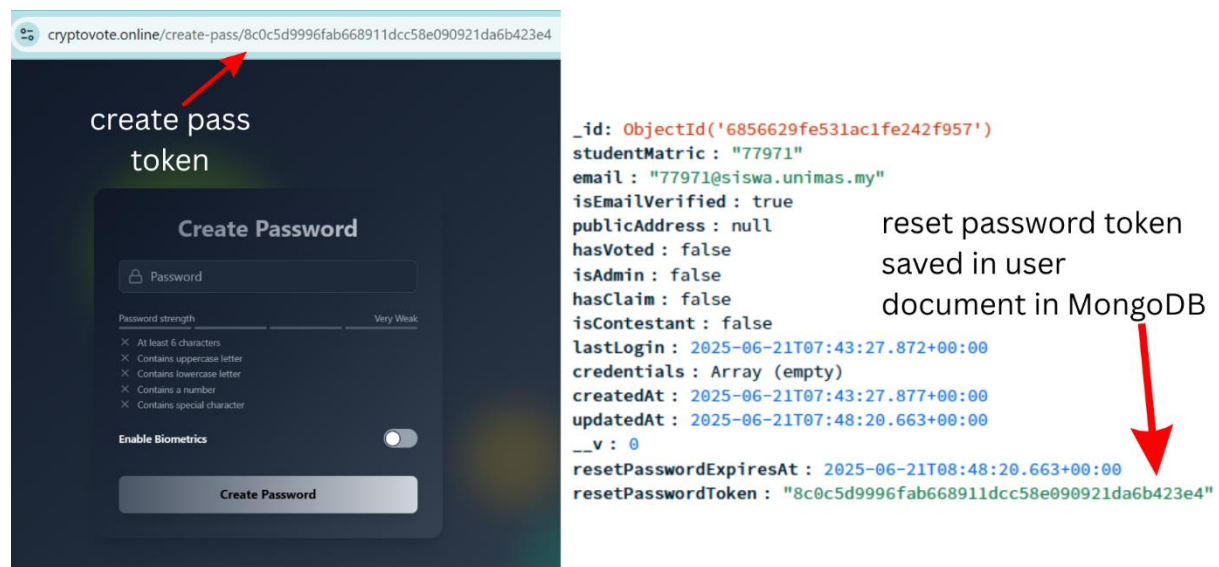


Figure 5.31. Create Password Token

Upon successful email verification, the previous `/verify-email` API endpoint generates a create password token, allowing the user to proceed to the next stage of setting their password or optionally create a passkey using WebAuthn that uses their device's biometric features, like fingerprint or face recognition, for a secure and easy login. This token will be saved temporarily in the MongoDB database as a credential to ensure only user who have successfully verified their email can access the password creation page. Without this token, users are restricted from accessing the page, which prevents unauthorized or unverified individuals from setting a password.

Once the password is submitted, the frontend makes a http post request to the `/save-pass` API endpoint. This request includes the newly created password and the create password token generated in the previous OTP verification stage obtained from the url, which the backend uses to identify the user in the database.

```

_id: ObjectId('6856629fe531ac1fe242f957')
studentMatric: "77971"
email: "77971@siswa.unimas.my"
isEmailVerified: true
publicAddress: null
hasVoted: false
isAdmin: false
hasClaim: false
isContestant: false
lastLogin: 2025-06-21T07:43:27.872+00:00
credentials: Array (1)
createdAt: 2025-06-21T07:43:27.877+00:00
updatedAt: 2025-06-21T08:07:03.100+00:00
__v: 1
password: "$2b$10$Ubw7HviPHN3ENK/jmmqrV./yWqVHR6DfY1UNvZR238PdJSg1L.QFu"

```

Hashed
Password
↓

Figure 5.32. Hashed Password in Database

Next, the password is hashed using a secure algorithm such as Bcryptjs before being stored in the database as in Figure 58. This ensures that even if the database is compromised, the raw passwords are never exposed, aligning with best practices for password management. Although the password creation process is off chain, it plays a crucial role in securing access to the user's account. This off-chain mechanism protects user actions such as linking a wallet, casting votes, and accessing voting history, in which all involve on-chain interactions that require authenticated access. Lastly, upon successful registration, user will be directed to the home page below and receive a welcome email.

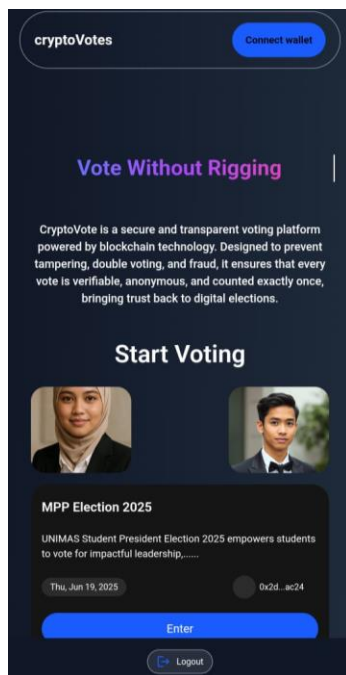


Figure 5.33. Home Page

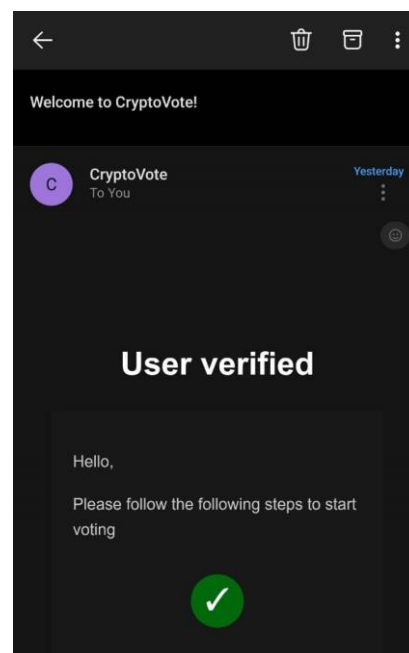


Figure 5.34. Welcome Email

4.6.3 Passkey Registration Using WebAuthn

In addition to traditional password-based authentication, the system also offers an optional method for users to register a passkey using WebAuthn, a modern web standard for secure and passwordless authentication. This feature allows users to use their device's built-in biometric features such as fingerprint or facial recognition to log in. The key idea behind WebAuthn is the use of public-key cryptography in which a user's private key never leaves their device, and only a public key is shared with the server. This makes it significantly harder for attackers to steal login credentials, even in the event of a data breach.

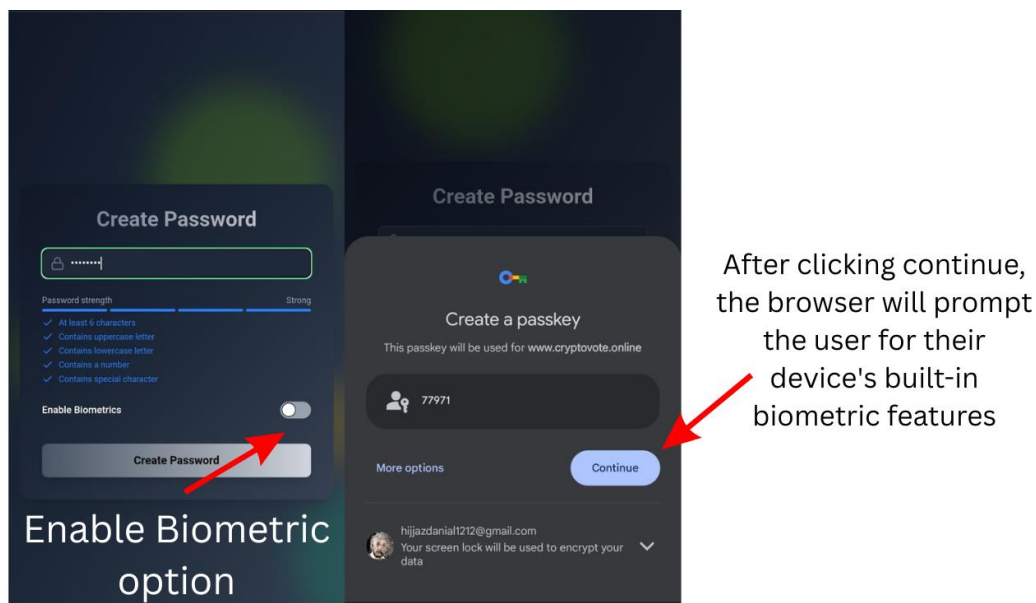


Figure 5.35. WebAuthn Registration

4.7 Login Workflow

4.7.1 Login

The login functionality in the CryptoVote system enables registered students to access their accounts securely using their matriculation number and previously created password. The login interface is designed with simplicity in mind, providing two input fields, one for the matric number and another for the password alongside a Login button.

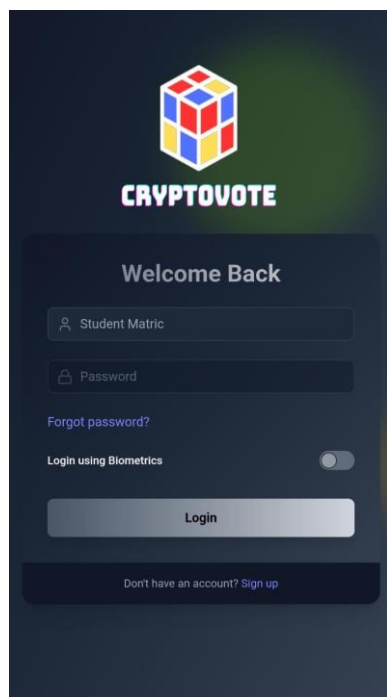


Figure 5.37. Login Page

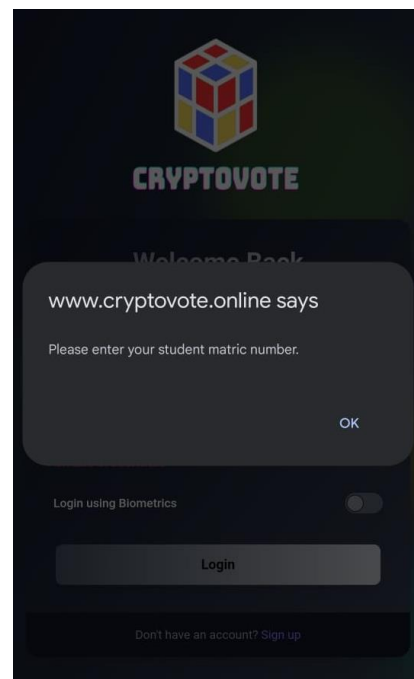


Figure 5.36. Input Validation in Login Page

When a user submits their credentials by clicking the Login button, the frontend uses the Axios JavaScript library to send an HTTP POST request to the /login API endpoint on the backend. Before sending this request, the frontend performs client-side validation to ensure that both the matric number and password fields are filled. If any field is empty, an appropriate error message is displayed immediately.

If the input is valid, the request is sent to the backend with the student's matric number and password. The backend first attempts to find a user record that matches the provided matric number in the MongoDB database. If no such user is found, or if the password is incorrect, the server responds with an error message of "Invalid credentials" to protect against user

enumeration attacks. This login workflow ensures secure, real-time user authentication, maintaining both the integrity and privacy of users accessing the CryptoVote platform.

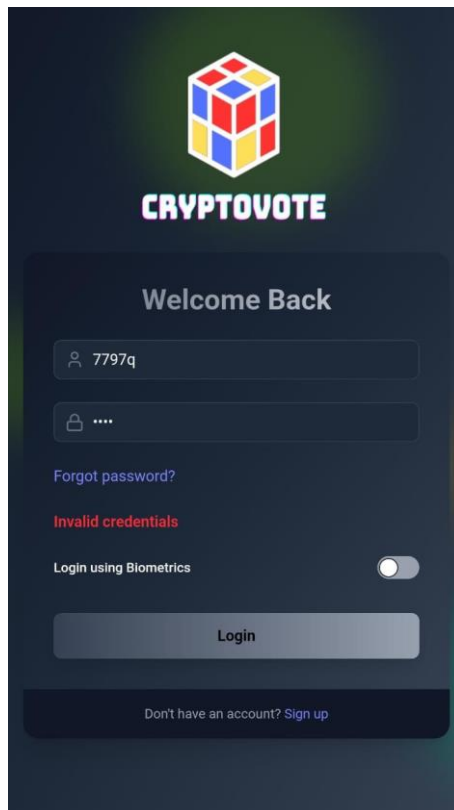


Figure 5.38. Login Page Invalid Credentials

4.7.2 Login Using WebAuthn

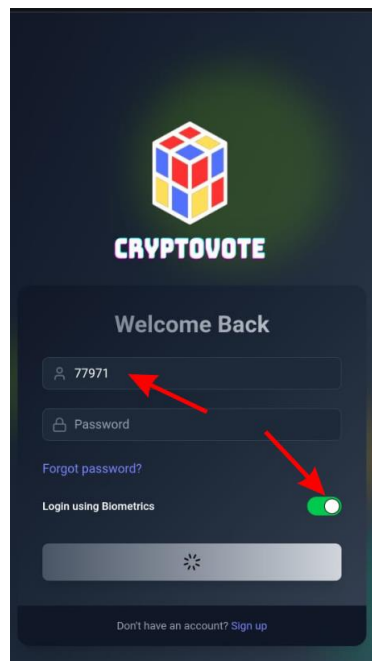
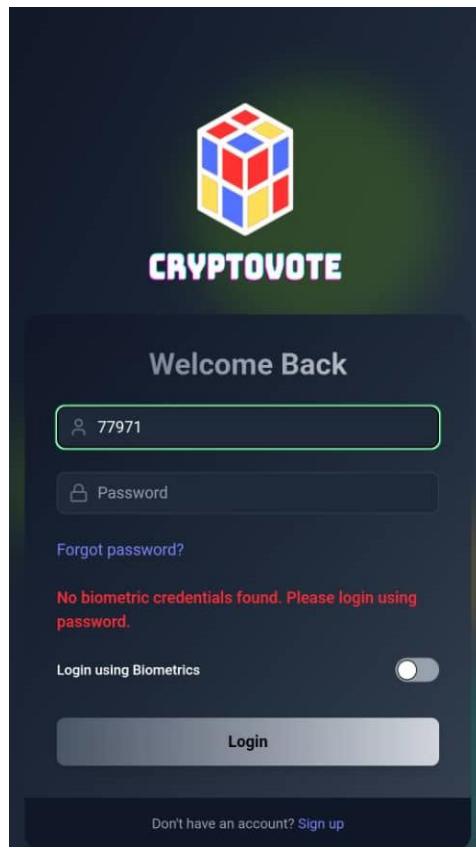


Figure 5.39. Login via WebAuthn

The WebAuthn login flow provides a secure and passwordless authentication method by allowing students to log in using their biometric credentials. To begin the process, the user first enters their student matric number in the login form. Instead of proceeding with a traditional password, they click the “Login with Biometric” button. The system then initiates the WebAuthn authentication request, prompting the browser to trigger a biometric challenge based on the credentials previously registered during the sign-up process.

If the user attempts to log in with biometrics without having registered WebAuthn credentials during sign-up, the system will detect the absence of stored credentials in the database and prevent the authentication process from proceeding. As shown in Figure below, an error message is shown, stating *"No biometric credentials found. Please login using password"*. This feedback informs the student that they must log in using a password instead or re-register with biometric authentication to enable WebAuthn in the future.



4.8 Reset Password Workflow

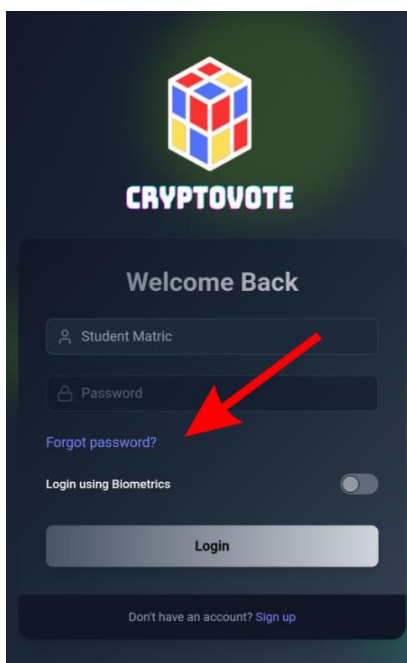


Figure 5.40. Forgot Password Link

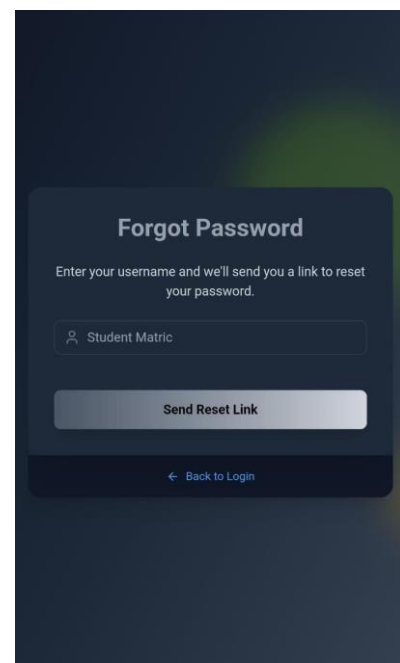


Figure 5.41. Forgot Password Page

The reset password workflow begins when a user clicks on the "Forgot Password?" link from the login form. This action redirects the user to the Forgot Password page, where they are prompted to enter their student matric number. Upon submission, the frontend triggers the forgotPassword function, which sends a POST request to the /forgot-password API endpoint with the student matric as the payload.

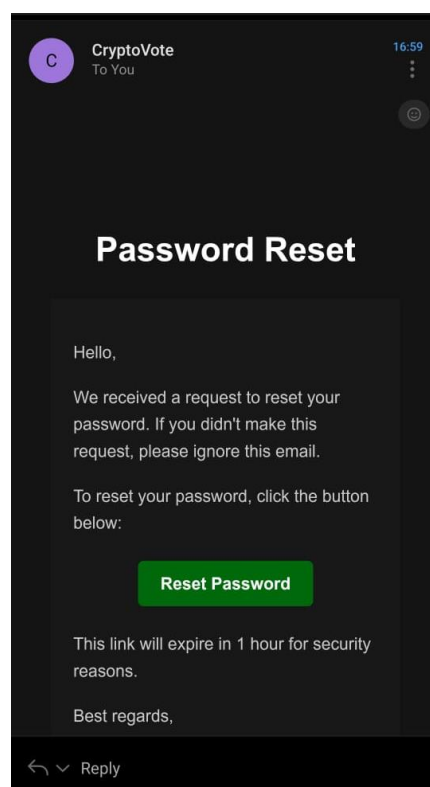


Figure 5.42. Reset Password Email

On the backend, the server first checks whether a user with the given matric exists in the database. If a match is found, it generates a secure reset password token, along with an expiration time set for one hour. This token is then stored in the user's record in MongoDB. Next, the server sends a password reset email to the user's registered email address. This email includes a button that links to a URL embedded with the generated token as in Figure 65. After clicking the Reset Password button, it redirects user to Reset Password page below with matching reset password token in the URL and in user's record in MongoDB.

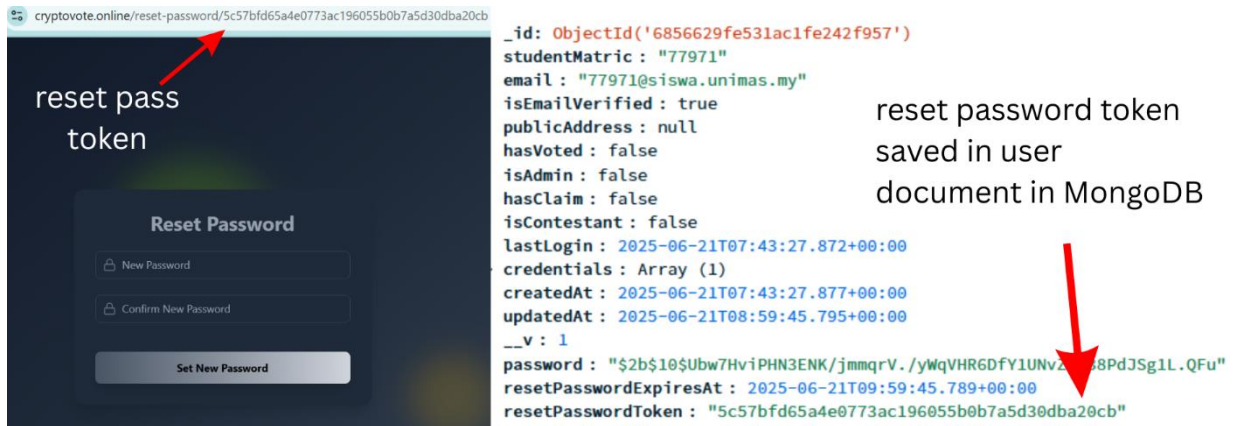


Figure 5.43. Reset Password Page

Once the user accesses the Reset Password page through the link, they are asked to enter and confirm their new password. When the form is submitted, the frontend calls the `resetPassword` function, which sends the token and new password to the `/reset-password/token` API endpoint in a POST request.

The backend verifies the token embedded to the url and check if it exists in the database and whether it is still valid within the time window. If valid, the new password is hashed and updated in the user's record. The reset token and its expiration timestamp are then cleared from the database to prevent reuse. Finally, a confirmation email is sent to notify the user that their password has been successfully changed, and the process concludes with a redirect back to the login page.

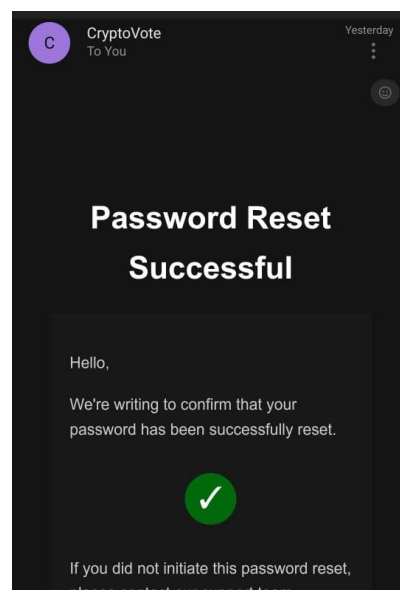


Figure 5.44. Reset Password Success Email

4.9 Connect Wallet Workflow

In order to participate in on-chain activities such as voting, contesting, or claiming tokens, users must first connect their personal crypto wallet to CryptoVote platform. Since all voting activities are performed on the blockchain, users must link their personal wallet to authenticate themselves and authorize transactions. Connecting a wallet allows this system to authenticate and identify users on the blockchain by linking their actions to a specific wallet address. This address acts as the user's unique identity when interacting with the smart contract but not revealing their private key to ensure voter anonymity. By establishing this connection, the platform ensures that all transactions such as casting a vote are securely tied to a verified wallet.

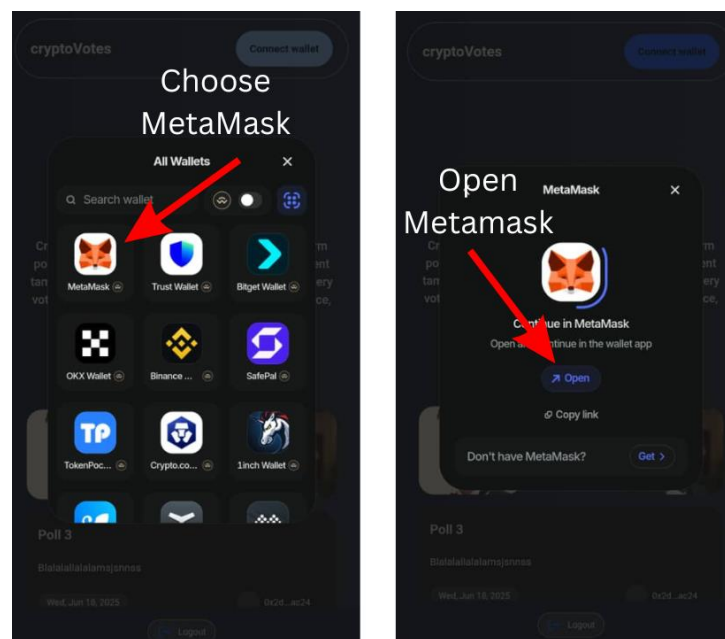


Figure 5.45. Choose Wallet

In this system, MetaMask is used as the primary example for the wallet connection process. Users begin by clicking the “Connect Wallet” button located in the top navigation bar of the application interface. This action opens a wallet selection modal where users are prompted to choose their preferred crypto wallet which can be seen in Figure 71 above. In this implementation, selecting MetaMask triggers a redirect to the MetaMask mobile application.

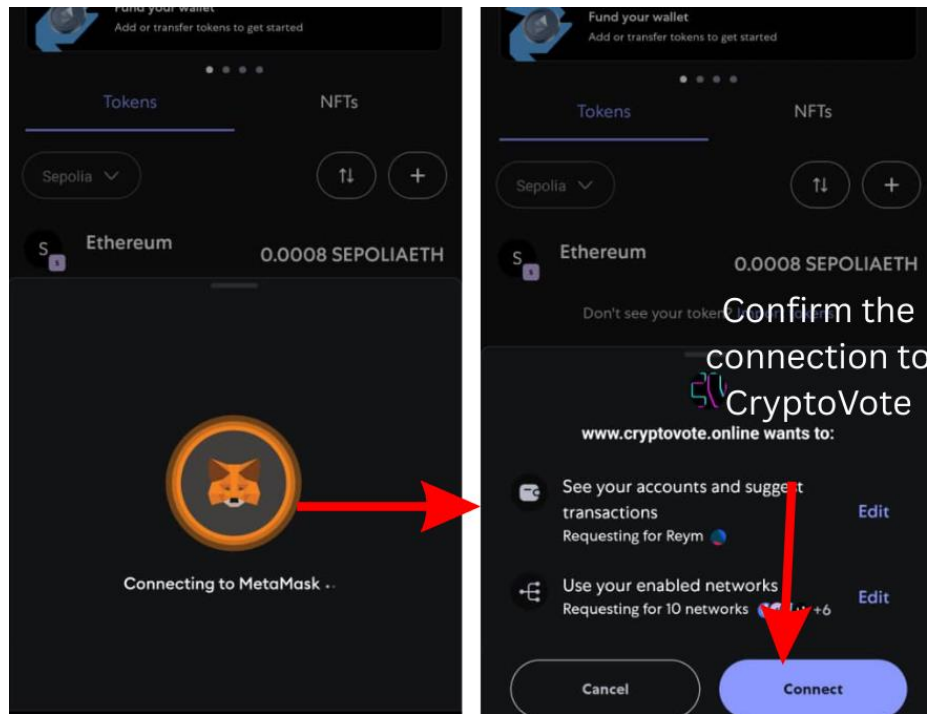


Figure 5.46. Confirm Wallet Connection

MetaMask then prompts the user to confirm the connection by selecting an account and authorizing access. Once the user approves the request, MetaMask establishes the connection and redirects the user back to the CryptoVote platform. At this point, the user's wallet address is stored in the frontend session, displayed on the navbar, and used to identify their interaction with the smart contract as shown in Figure 73 below. From this moment onward, any blockchain-based action performed by the user such as casting a vote or claiming tokens will be linked to this wallet address.



Figure 5.47. Connect Wallet Success

4.10 Claim Token Workflow

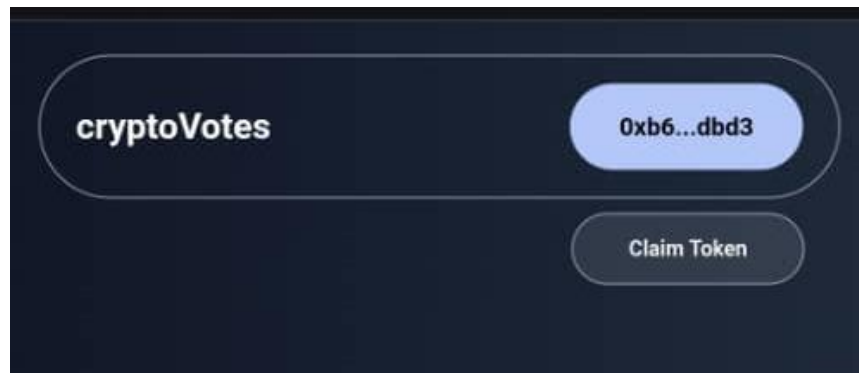


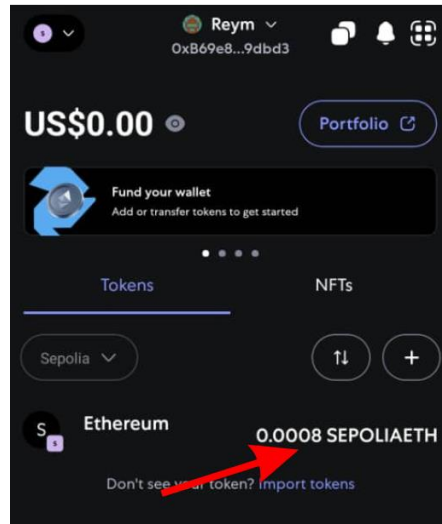
Figure 5.48. Claim Token Button

The Claim Token feature in this application allows users to receive a small amount of Sepolia ETH which is 0.008 from the system’s faucet wallet in order to participate in on-chain activities such as voting. This mechanism ensures that all eligible users can cover the gas fees needed for interacting with the Ethereum blockchain during the election.



Figure 5.49. Claim Token Success Message

When a user clicks the “Claim Token” button on the frontend interface, a POST request is sent to the /claimToken API endpoint. This request includes the user’s wallet address and unique user ID. While the transaction is in progress, the frontend displays a message notification with a spinner labeled “Claiming Token...” to provide feedback to the user. Once the transaction is successfully confirmed, another message appears saying “Token Claimed Successfully” to indicate completion.



After the transfer is confirmed, user receive 0.0008 Sepolia Eth and the system updates the user record in the database by saving the wallet address and setting the hasClaim flag to true. This flag is used on the frontend to ensure that users who have already claimed their tokens no longer see the “Claim Token” button, preventing repeated claims and improving user experience. This process ensures a fair and gas free experience for all participants.

4.11 Logout Workflow

The Logout functionality ensures that a student can securely end their session and prevent unauthorized access to protected routes. Once logged in, the student can click the “Logout” button located in the top navigation bar of the Home page.

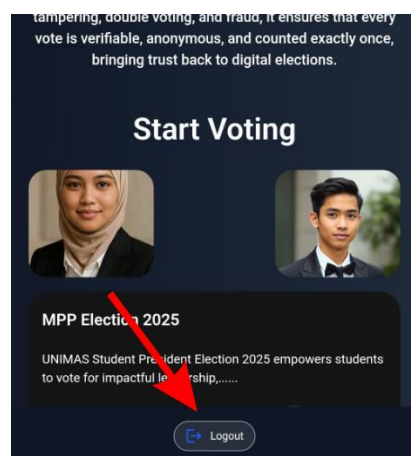


Figure 5.50. Logout Button

When a student clicks the “Logout” button located at the bottom of the Home page, the frontend triggers a HTTP POST request to the /logout API endpoint. On the backend, the server deletes the authentication cookie named ‘token’, which stores a special string called a JSON Web Token (JWT). This token includes the user’s ID and is digitally signed by the server. The JWT is placed inside a cookie when the user logs in, and this cookie is automatically sent to the server on every request. That’s how the server knows who the user is without asking them to log in again.

Once the cookie is deleted, the user’s session is terminated. The frontend then clears any stored user state and redirects the student to the Login page. This workflow helps maintain secure access control and prevents unauthorized access, especially when using shared devices or public networks.

4.12 Admin Workflow

In the CryptoVote system, admins are users with elevated privileges who manage poll-related operations. Their main responsibilities include creating new polls, editing existing ones, and deleting polls when necessary. However, each of these actions is subject to predefined conditions enforced by the smart contract to ensure election integrity. So, admins actions are also depend on the conditions set by smart contract. Further details of these operations are discussed in the following subsections.

4.12.1 Admin Registration

There is no public registration flow for creating admin users within the application. Instead, admin rights are manually assigned at the database level by setting the isAdmin flag to true in the user document. This approach allows the system owner or backend maintainer to control who has administrative privileges.

The figure displays two JSON documents representing user records in a database. The left document shows a user with `isAdmin: false`, and the right document shows the same user with `isAdmin: true`. Red arrows and text annotations explain the change.

```
_id: ObjectId('684ae89814cd92507713151e')
studentMatric: "77971"
email: "77971@siswa.unimas.my"
isEmailVerified: true
publicAddress: null
hasVoted: false
isAdmin: false
hasClaim: false
isContestant: false
lastLogin: 2025-06-14T08:40:58.849+00:00
credentials: Array (empty)
createdAt: 2025-06-12T14:47:52.520+00:00
updatedAt: 2025-06-14T08:40:58.856+00:00
__v: 0
password: "$2b$10$RRq6nMIXCIVciuVN8.Aq9.iwmjcBJlc"
```

default boolean value for isAdmin variable is false

```
_id: ObjectId('684ae89814cd92507713151e')
studentMatric: "77971"
email: "77971@siswa.unimas.my"
isEmailVerified: true
publicAddress: null
hasVoted: false
isAdmin: true
hasClaim: false
isContestant: false
lastLogin: 2025-06-14T08:40:58.849+00:00
credentials: Array (empty)
createdAt: 2025-06-12T14:47:52.520+00:00
updatedAt: 2025-06-14T08:40:58.856+00:00
__v: 0
password: "$2b$10$RRq6nMIXCIVciuVN8.Aq9.iwmjcE"
```

manually assign to true to give user admin role

Figure 5.51. Admin Manual Registration

After manually setting the isAdmin flag to true in the database, the interface will dynamically reveals admin-only controls such as poll creation, editing, and deletion. This method avoids exposing admin registration to the public while still allowing flexible backend control.

4.12.2 Creating Poll

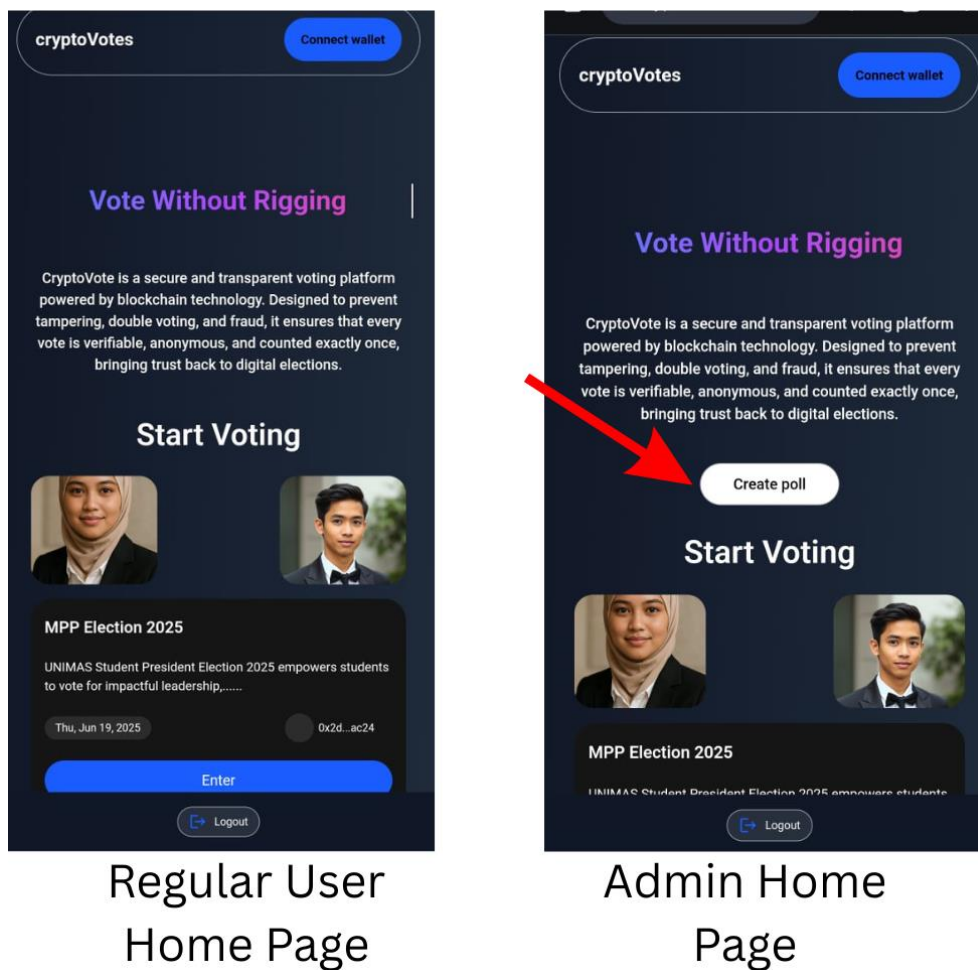


Figure 5.52. Admin Create Poll

Once a user has been assigned admin privileges by setting the `isAdmin` flag to true, the interface dynamically reveals a "Create Poll" button on the home banner. This button is only visible to admin users. As in Figure 70, normal users will not see the button, whereas admin users will.

When an admin clicks the "Create Poll" button, it triggers a modal containing a form interface (see Figure 71 below). This form allows the admin to input essential poll details such as the title, start date, end date, banner url and poll description. For both the start and end dates, the admin can click on the calendar icon, which brings up a calendar pop-up to conveniently select the desired date and time.

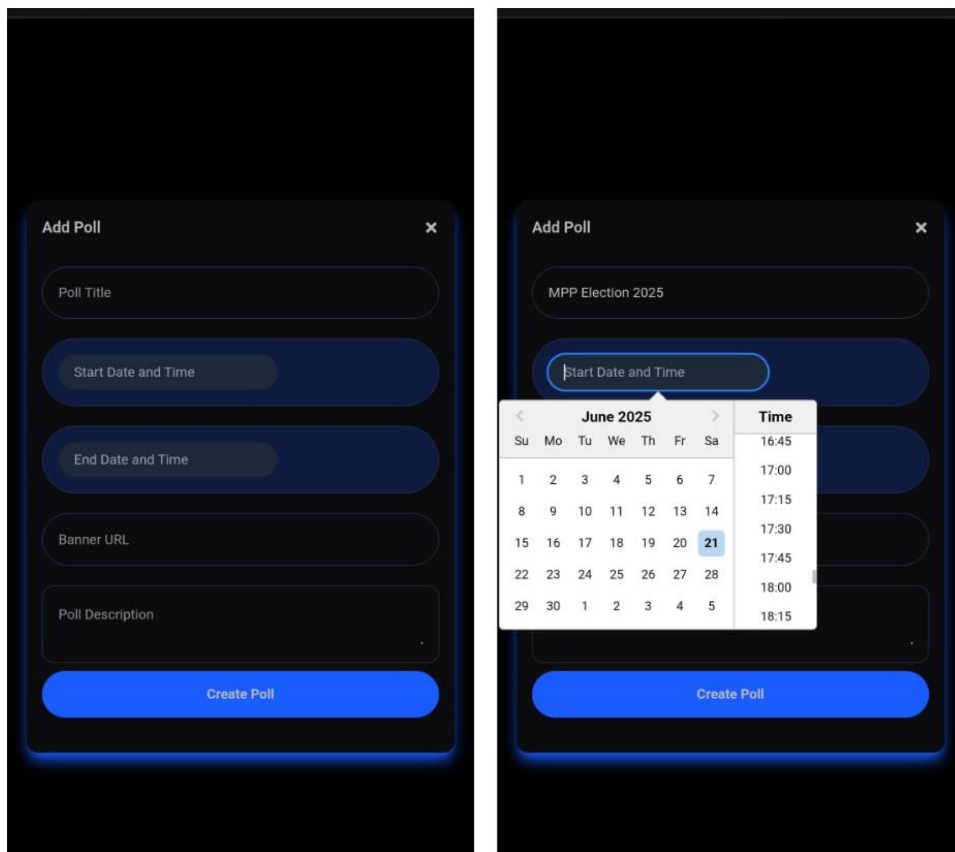


Figure 5.53. Create Poll Modal

For the Banner URL field, the admin is required to provide a direct link to the image that will be displayed as the banner for the poll. In this project, Imgur.com was used as the image hosting platform due to its simplicity and ability to generate direct image links. For example, a valid banner URL would look like: <https://i.imgur.com/BEC72RA.jpeg>. It is important to ensure the link ends with an image file extension such as .jpeg, .png, and .jpg so that the image can be correctly rendered in the poll interface.

After the admin fills in the poll description and clicks the "Create Poll" button, a MetaMask transaction prompt will appear to navigate user to the MetaMask in order to confirm transaction. After navigating to MetaMask, a modal displays the estimated gas fees required to deploy the poll to the blockchain. For example, the transaction or gas fees for this poll creation requires 0.000571 Sepolia ETH.

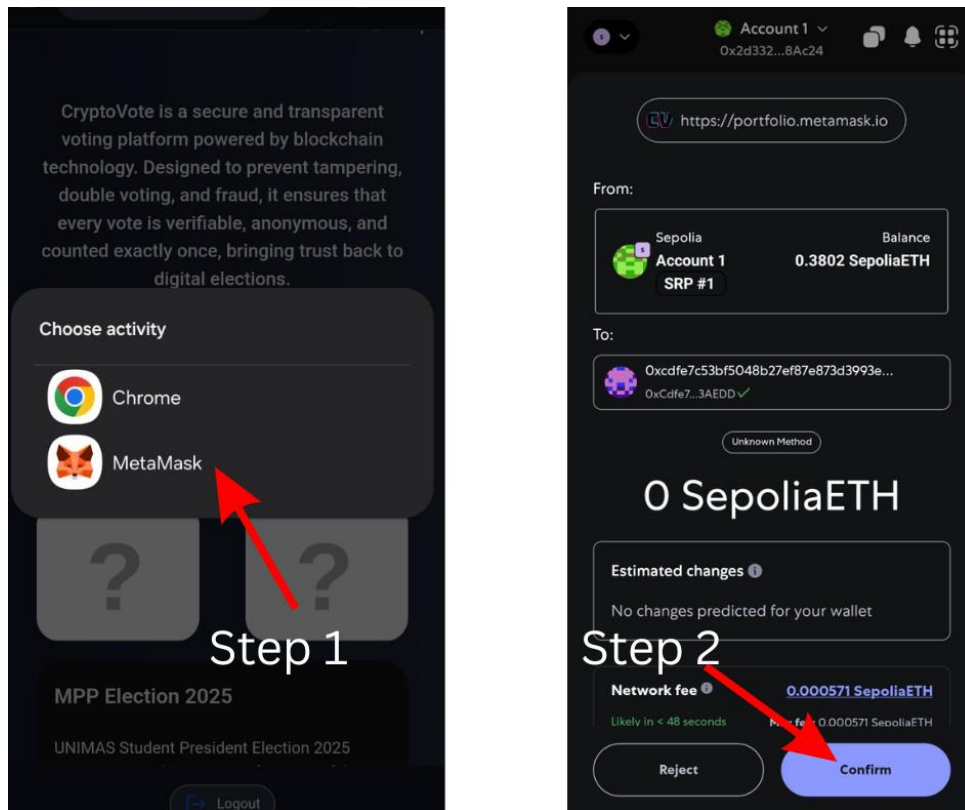


Figure 5.54. Admin Confirm Poll Creation Transaction Request

Once the admin confirms the transaction and it is successfully mined, the poll creation modal automatically closes, and the newly created poll will be displayed on the homepage as in Figure 81 alongside with the poll details and creator public wallet address ensuring that the creator is a valid user. This seamless process ensures that only valid and confirmed polls are recorded on-chain and presented to voters.

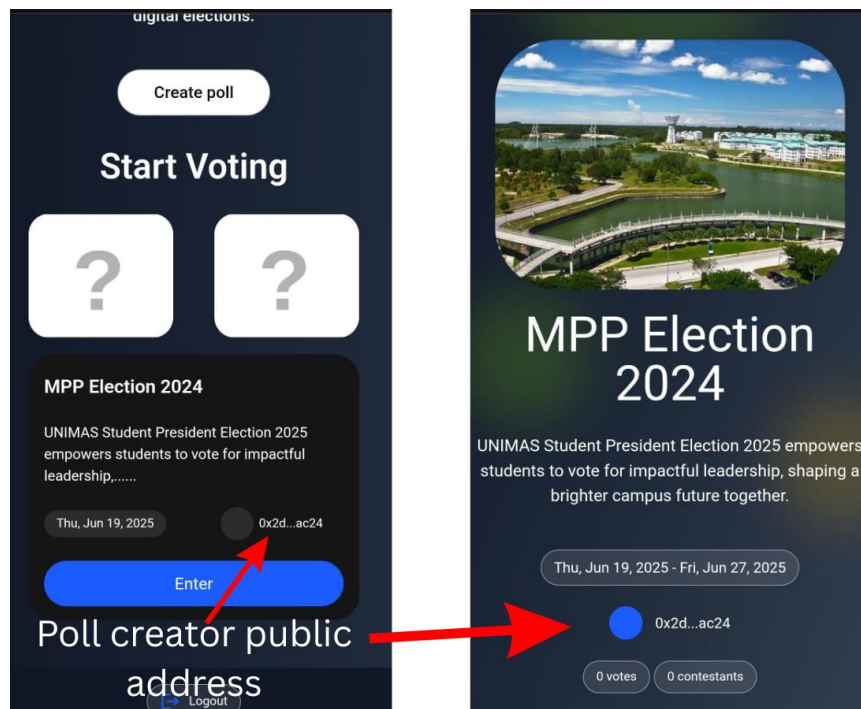


Figure 5.55. Successful Poll Creation

4.12.3 Edit Poll

Editing a poll is a restricted action, only allowed to the original poll creator and under very specific conditions. These conditions are hardcoded into the smart contract to ensure that polls are not tampered with once voting has begun or after deletion.

The important conditions to edit a poll are the person trying to edit must be the same address that created it, and the poll must not have received any votes yet. This prevents any manipulation after voting has started. Lastly, the new end date must be later than the start date to maintain logical poll duration.

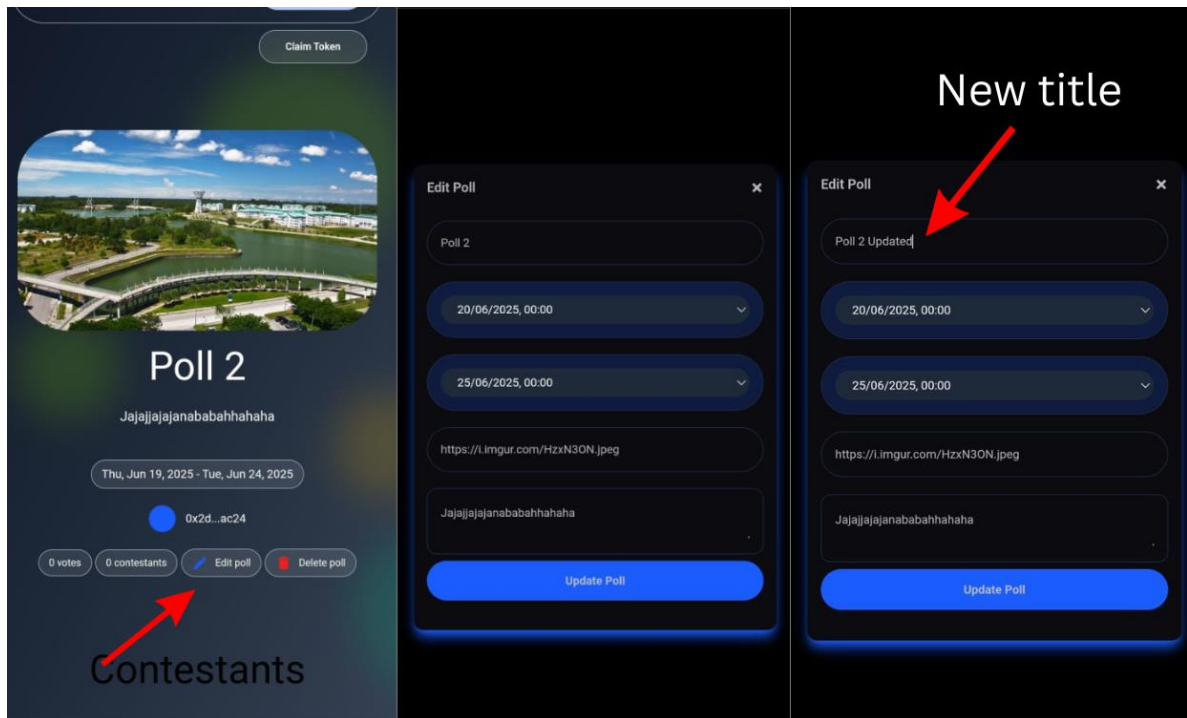


Figure 5.56. Admin Edit Poll

In the user interface, when the creator of a specific poll views their poll card, an Edit Poll button will be visible after. This button is hidden for regular users and other admins who did not create the poll. When clicked, it opens an update poll modal that looks similar to the create poll modal, but all fields are pre-filled with the poll's existing data for easy editing.

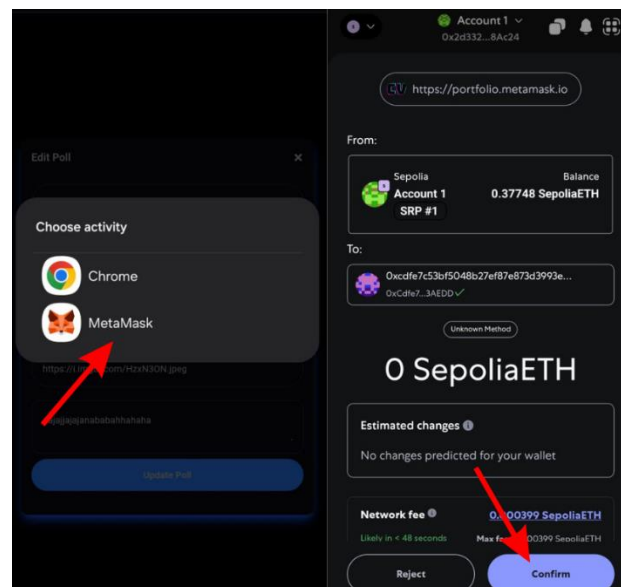


Figure 5.57. Edit Poll Transaction and Confirmation

Admins can modify any of the poll details which are the title, description, start date, end date, or banner image URL. Once done, clicking the Edit Poll button inside the modal triggers a MetaMask transaction confirmation window. The admin is then required to authorize the transaction, which updates the poll data on the blockchain.

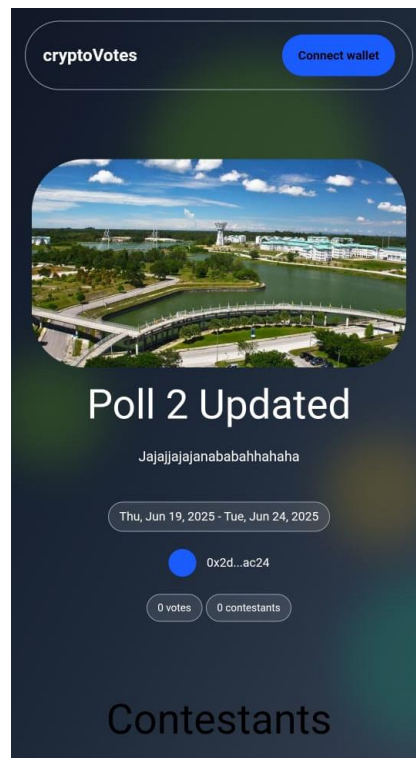


Figure 5.58. Updated poll

After the transaction is confirmed, the modal closes and the updated poll appears on the homepage with the new details. This process ensures that any edits made are verifiable and recorded on-chain, maintaining transparency and preventing any unauthorized or last-minute changes to a poll that may affect the integrity of the election.

4.12.4 Delete Poll

The Delete Poll feature is restricted to the poll creator (director) and is only allowed under specific conditions enforced by the smart contract. These conditions include: the poll must exist, the caller must be the poll's original creator, and the poll must not have received any votes. This ensures the integrity of the election by preventing the deletion of polls that already have voter participation.

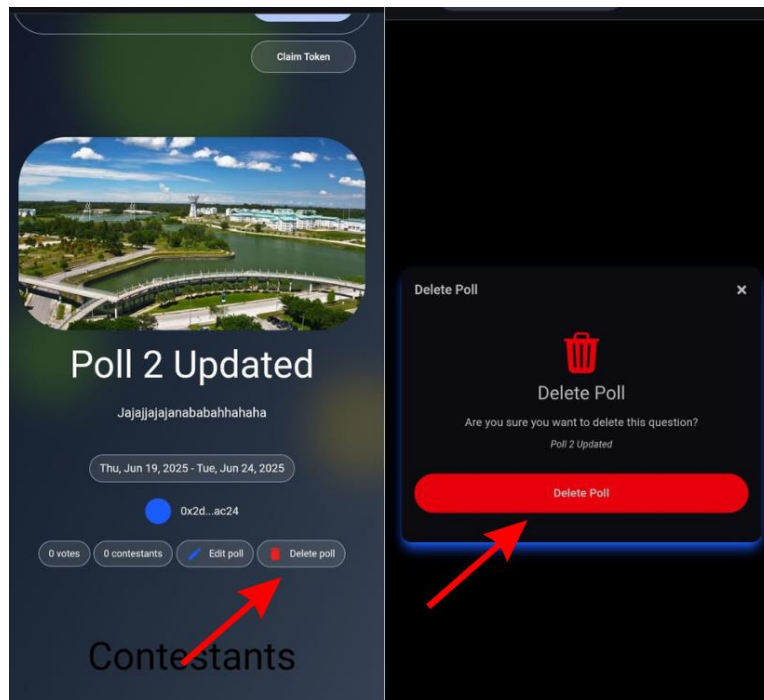


Figure 5.59. Admin Delete Poll

In the user interface, an admin who has created a poll will see a Delete Poll button located next to the Edit Poll button within the poll's card. When the Delete button is clicked, a confirmation modal appears asking the admin to confirm the deletion. This modal ensures that accidental deletions are avoided and gives the user a clear warning

Upon confirming the deletion, MetaMask will prompt the user to authorize the blockchain transaction. Just like with poll creation and editing, a gas fee is required, and the transaction must be signed and approved through MetaMask.

Once the transaction is mined and confirmed, the modal automatically closes, and the deleted poll will be removed from the homepage, ensuring the UI stays in sync with the on-chain data. This workflow provides a secure and verifiable mechanism for poll removal, while ensuring that only authorized, vote-free polls can be deleted.

4.13 Contestants Workflow

This section outlines the workflow of contestants within the CryptoVote system. Contestants are registered users who choose to participate in a poll as candidates. The system provides a structured process for users to register as contestants and associate themselves with specific polls. The following subsections describe how contestants are registered and how they are linked to a poll by updating the smart contract.

4.13.1 Contestants Registration

Similar to the admin registration flow, there is no public-facing registration process for users to become contestants in the CryptoVote system. Instead, contestant status is manually assigned by modifying the user record in the database. Specifically, the system administrator or database maintainer sets the `isContestant` flag to `true` within the user document.

The figure displays two JSON documents representing user records in a database. The left document shows a user with `isContestant: false`. The right document shows the same user after being manually assigned the role of a contestant, with `isContestant: true`. Red arrows and text annotations highlight the change in the `isContestant` field.

```
{
  "_id": "ObjectId('6856629fe531ac1fe242f957')",
  "studentMatric": "77971",
  "email": "77971@siswa.unimas.my",
  "isEmailVerified": true,
  "publicAddress": null,
  "hasVoted": false,
  "isAdmin": true,
  "hasClaim": false,
  "isContestant": false,
  "lastLogin": "2025-06-21T12:23:45.181+00:00",
  "credentials": Array (1),
  "createdAt": "2025-06-21T07:43:27.877+00:00",
  "updatedAt": "2025-06-21T12:23:45.182+00:00",
  "__v": 1,
  "password": "$2b$10$Ubw7HviPHN3ENK/jmmqrV./yWqVHR6DFY1",
  "resetPasswordExpiresAt": "2025-06-21T09:59:45.789+00:00",
  "resetPasswordToken": "5c57bfd65a4e0773ac196055b0b7a5d"
}
```

default boolean value for isContestant variable is false

```
{
  "_id": "ObjectId('6856629fe531ac1fe242f957')",
  "studentMatric": "77971",
  "email": "77971@siswa.unimas.my",
  "isEmailVerified": true,
  "publicAddress": null,
  "hasVoted": false,
  "isAdmin": true,
  "hasClaim": false,
  "isContestant": true,
  "lastLogin": "2025-06-21T12:23:45.181+00:00",
  "credentials": Array (1),
  "createdAt": "2025-06-21T07:43:27.877+00:00",
  "updatedAt": "2025-06-21T12:23:45.182+00:00",
  "__v": 1,
  "password": "$2b$10$Ubw7HviPHN3ENK/jmmqrV./yWqVHR6DFY1",
  "resetPasswordExpiresAt": "2025-06-21T09:59:45.789+00:00",
  "resetPasswordToken": "5c57bfd65a4e0773ac196055b0b7a5d"
}
```

manually assign to true to give user admin role

Figure 5.60. Contestant Registration

Once this `isContestant` flag is updated, the frontend interface dynamically reveals contestant specific functionalities to the user which are a "Contest" button on the poll page as in Figure 78 below. This approach ensures controlled participation, preventing unauthorized users from registering themselves as candidates while still allowing flexibility for the system owner to assign contestant roles as needed.

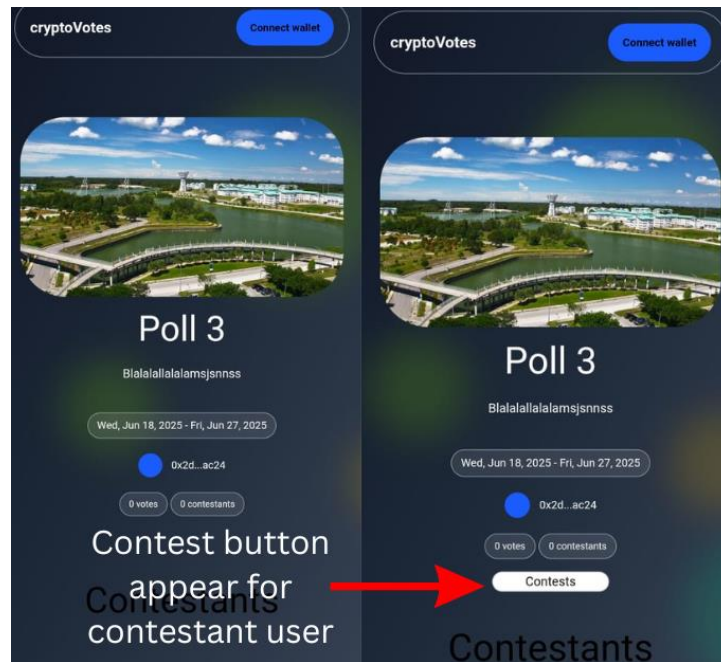


Figure 5.61. Contest Button for Contestants

4.13.2 Contesting Poll

Figure 5.62. Contest Poll Form Modal

Once a user has been granted contestant access, they are able to contest in active polls within the CryptoVote system. On the poll page, a “Contest” button becomes visible to eligible

contestants. When this button is clicked, a form modal appears, prompting the user to enter their name and an image URL to visually represent their candidacy.

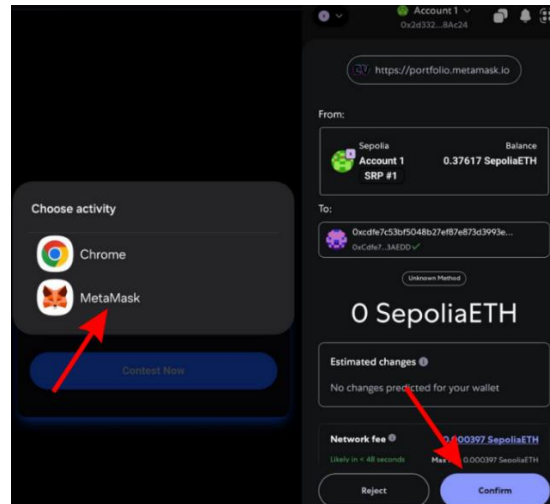


Figure 5.63. Contest Poll Transaction and Confirmation

Once the required fields are filled, clicking the Contest Now button initiates a MetaMask transaction. The user will be redirected to MetaMask to approve and confirm the transaction, which involves writing their contestant information onto the blockchain. Upon successful confirmation and mining of the transaction, the contestant's information will immediately appear on the respective poll's page.

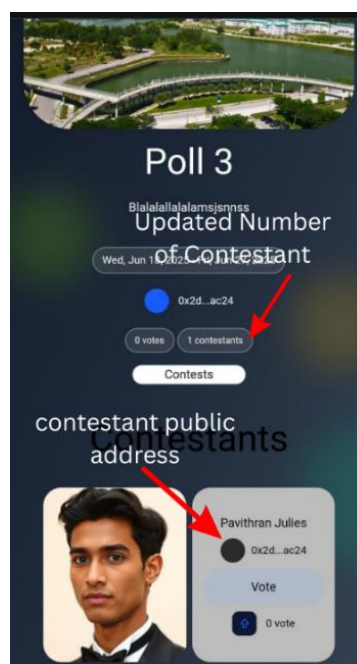


Figure 5.64. New Contestant Displayed

Each contestant is displayed inside a card that includes their name, public wallet address, a Vote button, and the number of votes they have received so far. To ensure transparency, all contestants are listed sequentially based on the number of votes, with the contestant receiving the highest votes appearing at the top of the list. This provides a real-time overview of poll standings and encourages fair participation.

4.14 Voter Workflow

The voting process is a core part of the CryptoVote system, and it is strictly governed by several conditions defined in the smart contract to ensure fairness and prevent abuse. These conditions include

- The poll must exist and must not be marked as deleted.
- The user must not have already voted in that poll.
- The poll must have more than one registered contestant.
- The current time must fall within the poll's active voting period.

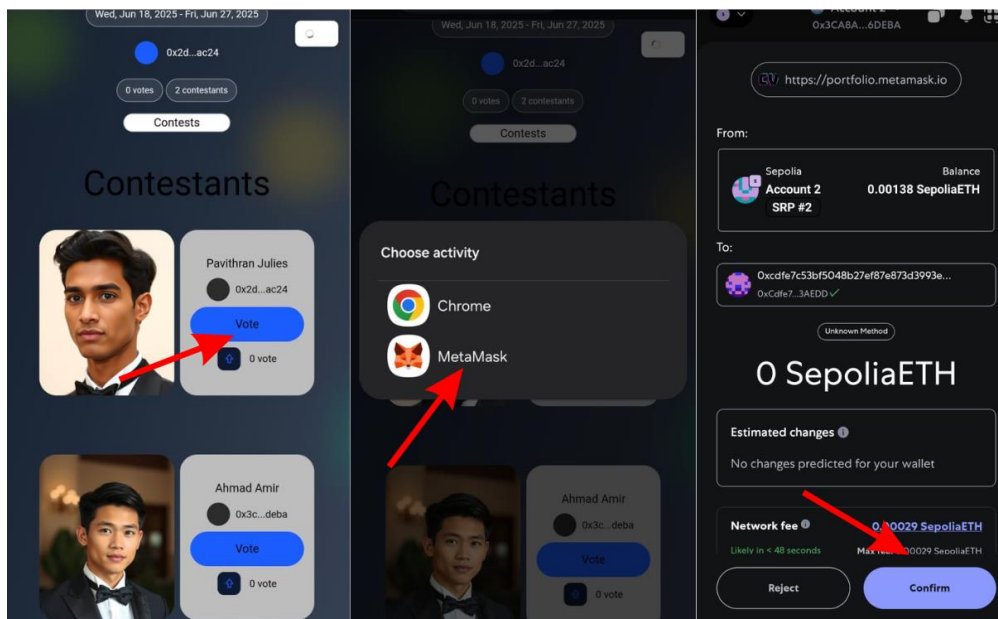


Figure 5.65. Vote flow

If all of these conditions are satisfied, MetaMask prompts the voter to confirm the transaction. Upon approval, the vote is submitted to the blockchain, where it is securely recorded. Once the transaction is confirmed, the frontend updates the poll's details, including the total number of votes and the vote count for the selected contestant as in Figure 84 below.

This update ensures that the voting interface remains synchronized with the blockchain state, providing real-time feedback to users and preserving the integrity of the election process.

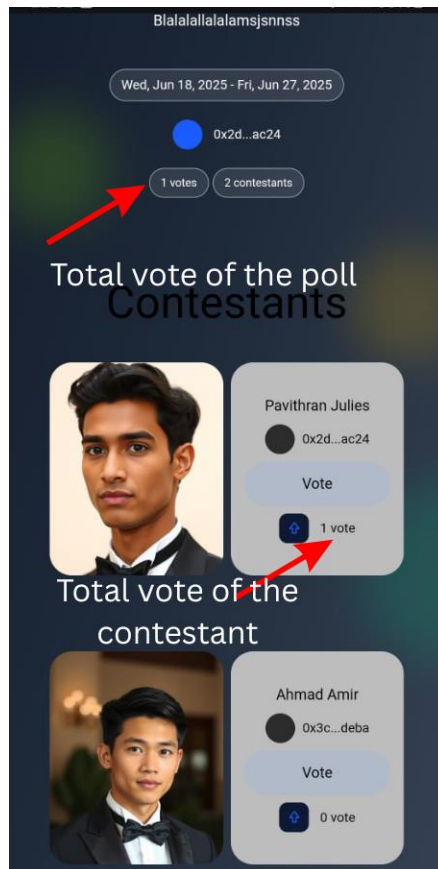


Figure 5.66. Updated Poll Details After Vote

4.15 Summary

Chapter 4 detailed the full implementation of the CryptoVote system, outlining the technologies used, how the framework and dependencies are installed and utilized, and the development and deployment of the smart contract to the Ethereum blockchain. It described the complete user workflows including sign-up, and password reset processes. The chapter also explained the wallet connection workflow demonstrating how users securely link their crypto wallets to interact with the voting system. Role-specific workflows were covered as well, including the admin's ability to manage polls, contestant registration, and the voter's participation in elections, showcasing the system's end-to-end functionality and secure interaction design.

Chapter 5: Testing

5.1 Introduction

This chapter discusses the testing process conducted for the CryptoVote system to ensure that every feature works as intended. Software testing is a crucial step in the development lifecycle, used to identify bugs, system failures, or logical issues before deployment. By thoroughly testing the system's smart contract logic, backend processes, and frontend user interfaces, this chapter ensures that the application behaves correctly and securely in different scenarios.

5.2 Testing Strategy

The testing strategy for CryptoVote is divided into two main categories: Functional Testing and Non-Functional Testing. Functional testing focuses on verifying that the system's features behave as expected according to the requirements. It covers both smart contract functions and user interface flows. Non-functional testing focused specifically on Usability Testing. A small group of real students tested the application and provided feedback on user experience elements such as clarity of the navigation flow, ease of completing tasks like sign-up and voting, and overall visual design.

5.3 Functional Testing

This section covers the functional testing conducted to ensure that all features in the CryptoVote system work as expected. It is divided into two main parts which are smart contract functional testing and authentication functional testing.

5.3.1 Smart Contract Functional Testing

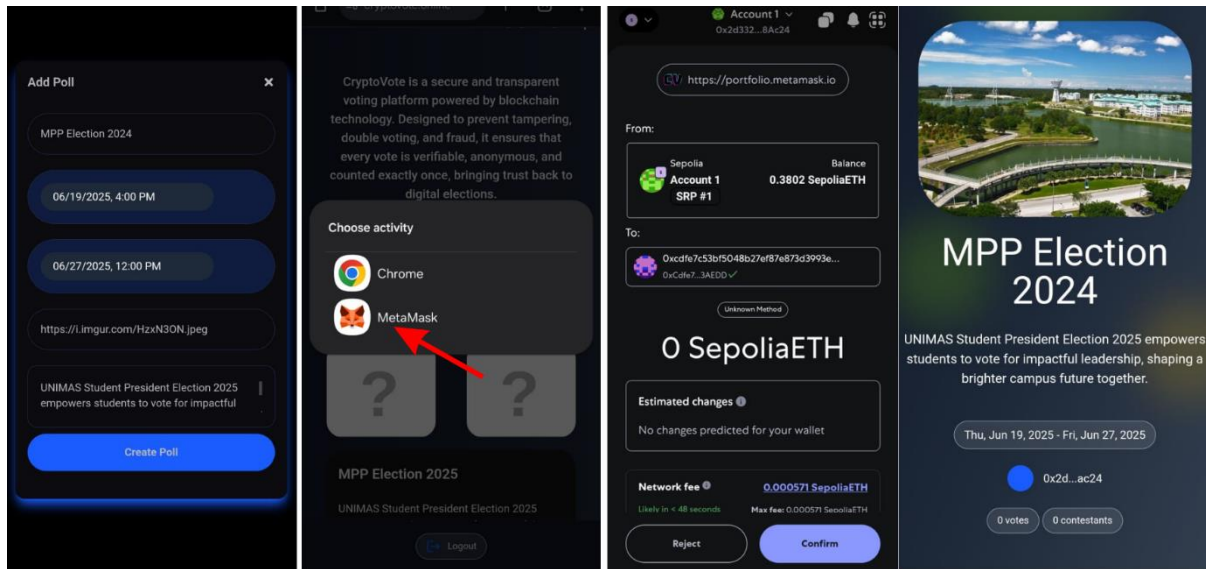
This subsection focuses on testing the core blockchain functionalities handled by the smart contract, including actions performed by admins, contestants, and voters. Each function such as creating polls, registering as a contestant, and casting votes was tested to ensure it executes securely, and updates the blockchain state correctly.

5.3.1.1 Admin Test Case

Admin Test Case ID: A-TC01 Create Poll

<u>Item</u>	<u>Details</u>
Description	Verify that the admin can create a poll with valid inputs. Poll must appear in the frontend after MetaMask transaction is confirmed.
Input	Title: "MPP Election 2024" Start Time: 06/19/2025, 12:00 AM End Time: 06/27/2025, 12:00 AM ImageURL: https://i.imgur.com/HzxN3ON.jpeg Description: UNIMAS Student President Election 2025 empowers students to vote for impactful leadership, shaping a brighter campus future together.
Expected Output	MetaMask transaction prompts admin; after confirmation, poll appears in the list with correct info.
Actual Output	MetaMask transaction completed. Poll displayed correctly in frontend.

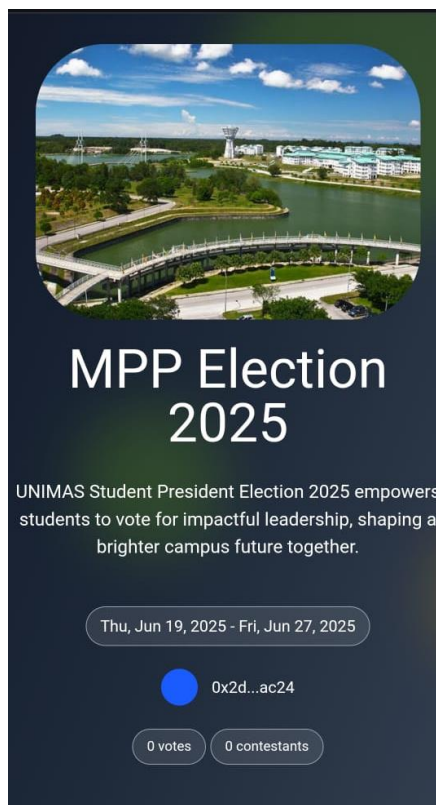
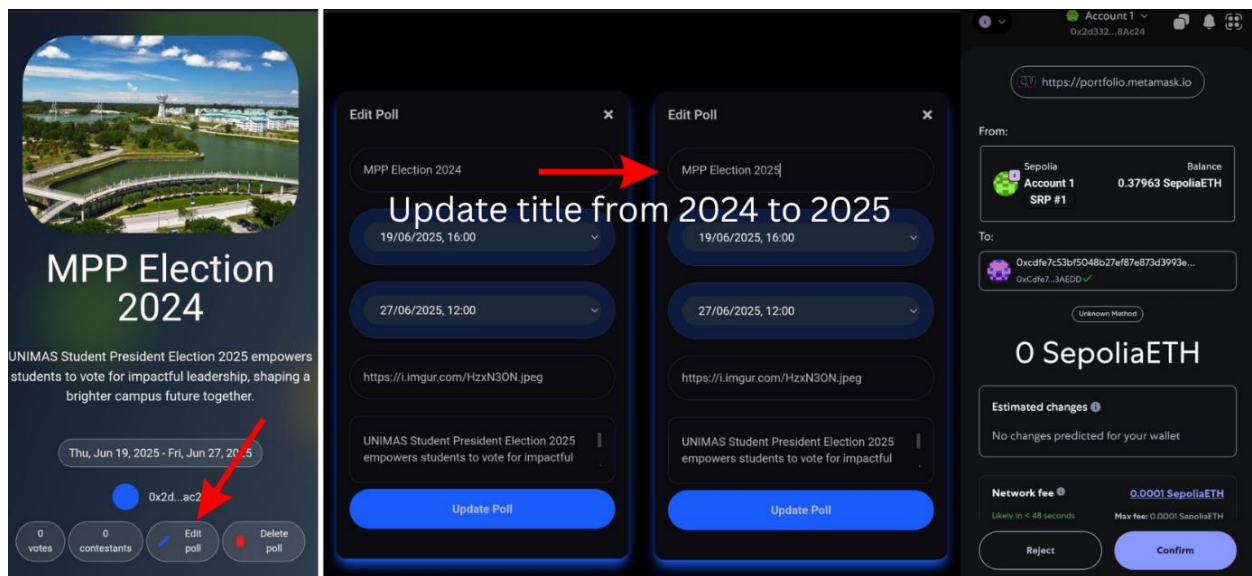
Screenshots:



Admin Test Case ID: TC02 Update Poll

<u>Item</u>	<u>Details</u>
Description	Verifies that the admin can update poll details (title, date, image, description) before any contestants register or votes are cast.
Input	Title: "MPP Election 2025 " (new title) Start Time: 06/19/2025, 12:00 AM End Time: 06/27/2025, 12:00 AM ImageURL: https://i.imgur.com/HzxN3ON.jpeg Description: UNIMAS Student President Election 2025 empowers students to vote for impactful leadership, shaping a brighter campus future together.
Expected Output	After MetaMask confirmation, the updated poll data replaces the original. The poll ID remains the same and the changes reflect in the UI.
Actual Output	MetaMask transaction completed. Poll displayed correctly in frontend.

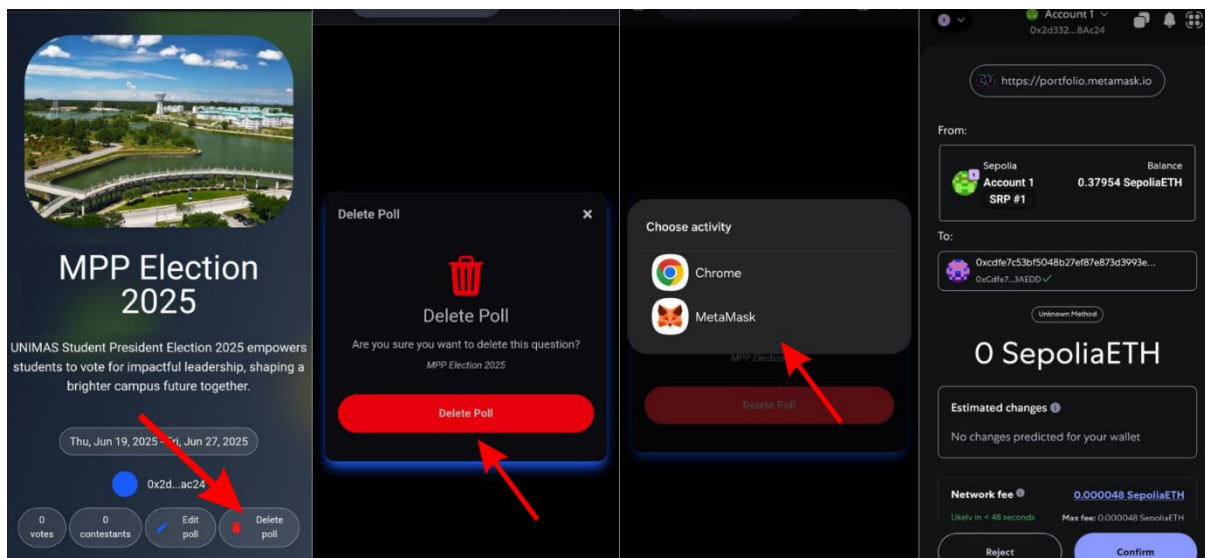
Screenshot:



Admin Test Case ID: TC03 Delete Poll

<u>Item</u>	<u>Details</u>
Description	Verifies that the admin can delete an existing poll, only if no contestants have registered and no votes have been cast.
Input	No form input. Admin clicks “Delete Poll” on the poll detail page, confirms in the pop-up, and approves transaction via MetaMask.
Expected Output	MetaMask prompts for confirmation. Upon approval, the poll is marked as deleted in the smart contract and removed from the frontend UI list.
Actual Output	MetaMask transaction was confirmed. The poll disappeared from the UI poll list and was no longer accessible via direct poll link.

Screenshot:

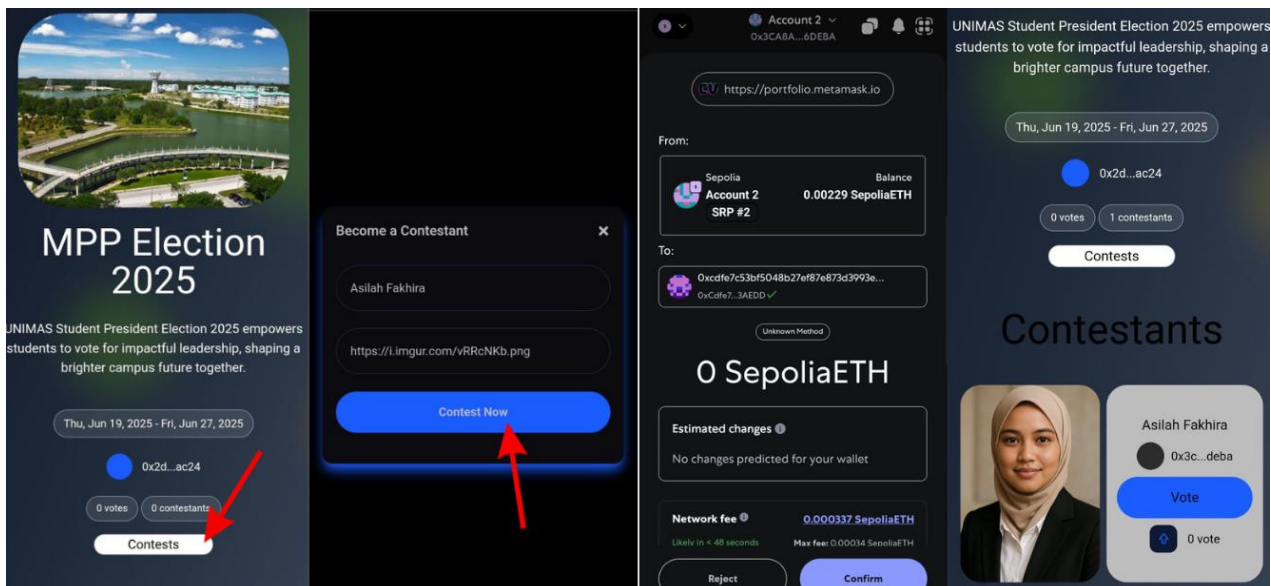


5.3.1.2 Contestant Test Case

Contestant Test Case ID: TC01 Contest Poll

<u>Item</u>	<u>Details</u>
Description	Verifies that a student can successfully register as a contestant in an existing poll by providing their name and image URL.
Input	Name: "MPP Election 2025 " (new title) ImageURL: https://i.imgur.com/vRRcNKb.png
Expected Output	MetaMask prompts the user to confirm the transaction. After confirmation, the contestant is added to the poll with their name, image, and public wallet address.
Actual Output	MetaMask transaction confirmed. Contestant appeared correctly on the poll detail page with matching info and wallet address.

Screenshot:

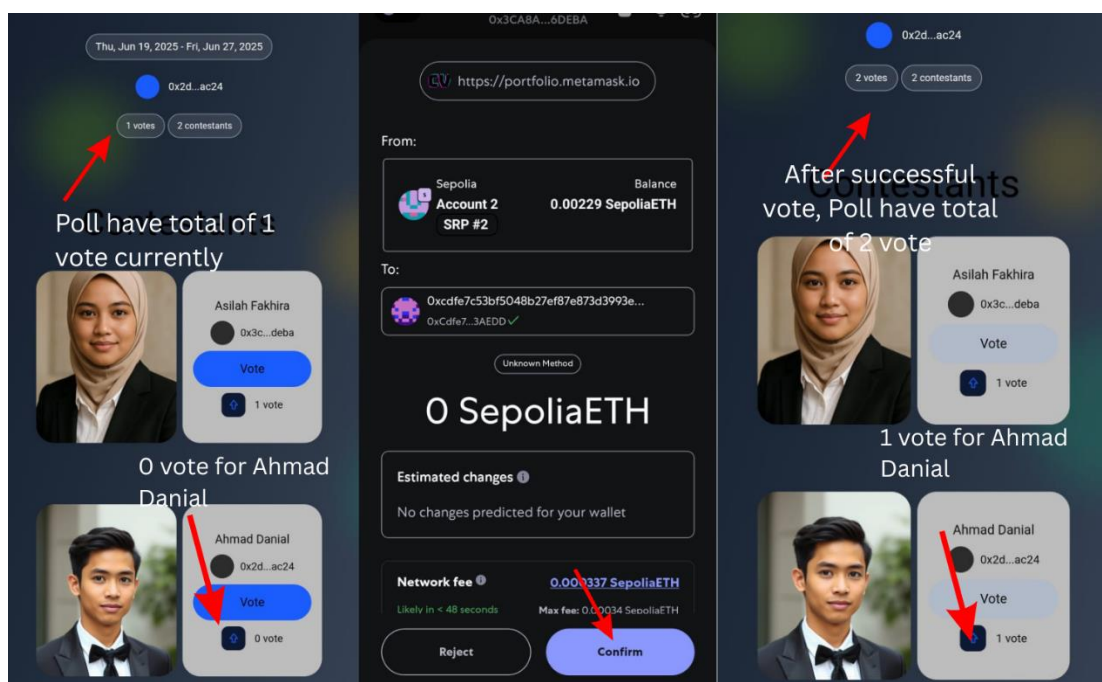


5.3.1.2 Voter Test Case

Voter Test Case ID: TC01 Vote Contestant

<u>Item</u>	<u>Details</u>
Description	Verifies that a registered voter can successfully cast a vote for a contestant during the active voting period. Only one vote per voter per poll is allowed.
Input	Contestant: Ahmad Danial
Expected Output	MetaMask prompts the voter to confirm the transaction. Once confirmed, the vote is recorded on the blockchain, the selected contestant's vote count increases by 1, and the vote button is disabled for that user.
Actual Output	Transaction successfully confirmed via MetaMask. Contestant's vote count increased by 1 in both the frontend and blockchain. The vote button was disabled afterward.

Screenshot:



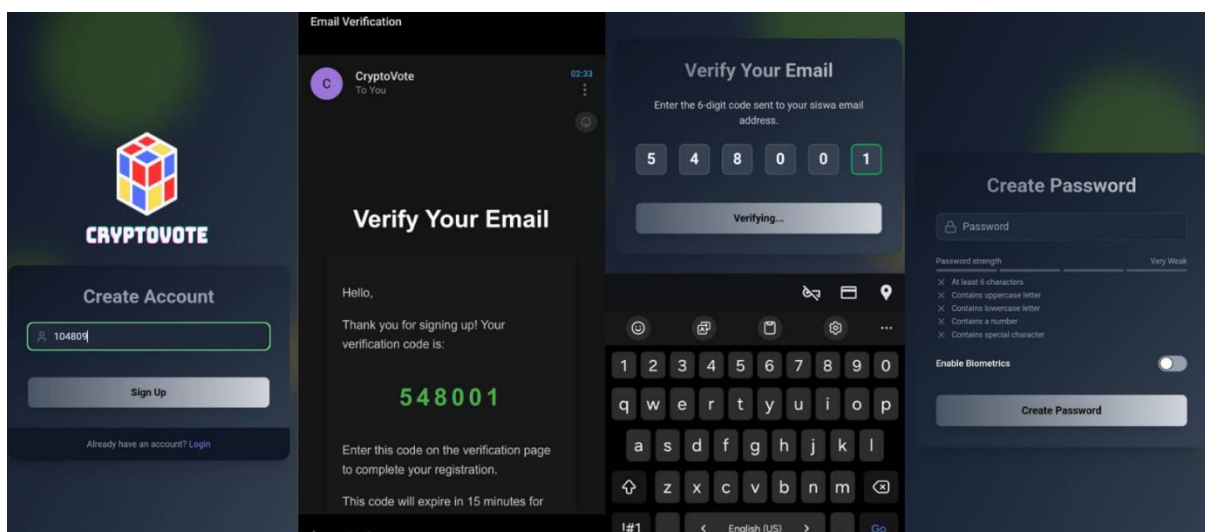
5.3.2 Authentication Functional Testing

5.3.1.2 Sign In Test Case

Test Case ID: TC01 Email Verification

<u>Item</u>	<u>Details</u>
Description	Verifies that a student can receive the OTP via their university email after entering a valid matric number during sign-up.
Input	Matric Number: 104809 OTP: 548001
Expected Output	After submitting a valid matric number, the system sends an OTP to the student's official email. The frontend redirects to the email verification page. Upon entering the correct OTP, the system verifies the user and navigates to the Create Password page.
Actual Output	The OTP was sent successfully using Brevo SMTP service. After entering the correct OTP, the system verified the student and redirected to the password creation page.

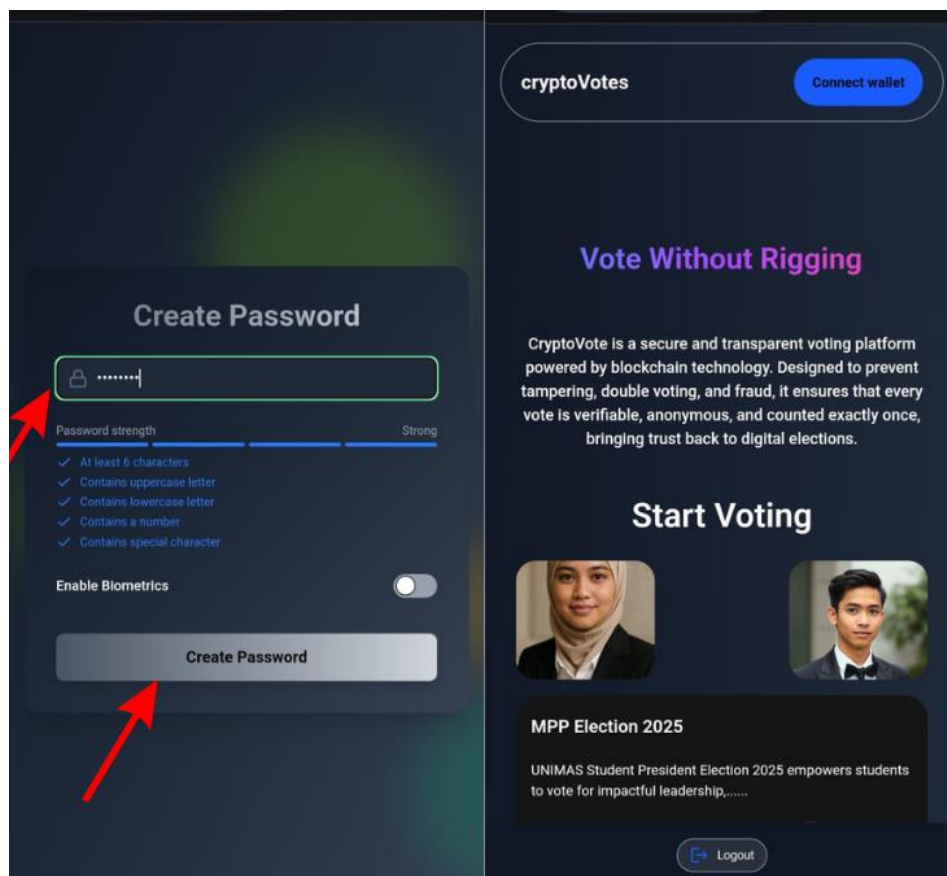
Screenshot:



Test Case ID: TC02 Create Password With No Biometric

<u>Item</u>	<u>Details</u>
Description	Verifies that a student can create a password after successful OTP verification.
Input	Password: Unimas2025!
Expected Output	Once both fields are filled and submitted, the system saves the password securely and redirects the student to the Home page. A success message is displayed.
Actual Output	The password was successfully saved and the interface redirected the student to the Home page..

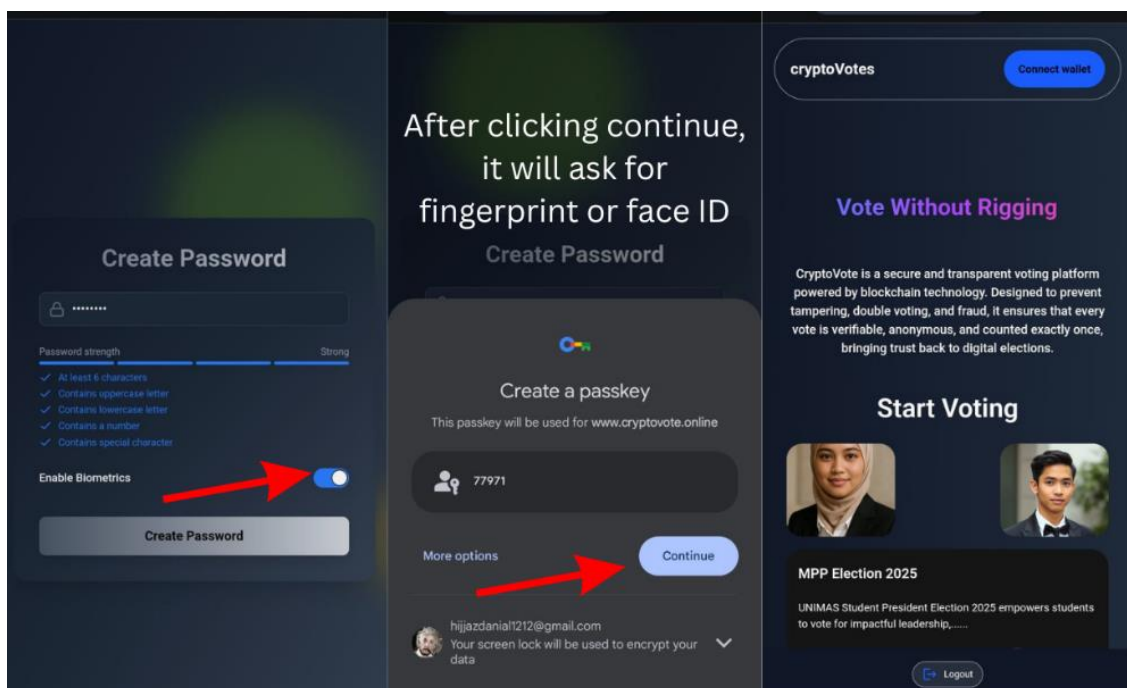
Screenshot:



Test Case ID: TC02 Create Password With Biometric

<u>Item</u>	<u>Details</u>
Description	Verifies that a student can create a password and also register a biometric device using WebAuthn. This allows future logins via fingerprint or other biometric ID.
Input	Password: Unimas2025! Biometric: Enabled by clicking “Enable Biometric” button
Expected Output	After password is submitted, the system triggers the WebAuthn registration prompt. Upon successful biometric verification, user data is saved with biometric credentials and redirected to Home page.
Actual Output	WebAuthn prompt appeared. The student successfully registered a biometric device (fingerprint). Password and credentials saved. Redirected to Home page.

Screenshot:

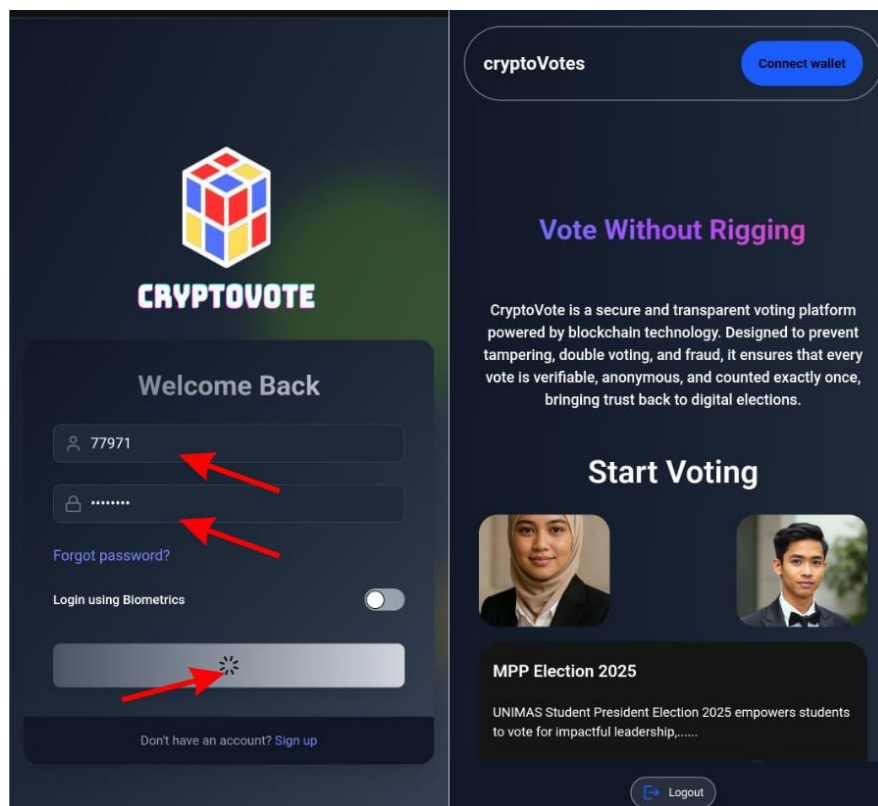


5.3.1.3 Login Test Case

Test Case ID: TC01 Login With Password

<u>Item</u>	<u>Details</u>
Description	Verifies that a registered student can log in using their matric number and password. If the credentials are correct, the system grants access to the Home page.
Input	Matric Number: 77971 Password: Unimas2025!
Expected Output	After submitting valid credentials, the system authenticates the student and redirects them to the Home page
Actual Output	Login succeeded. User was redirected to the Home page

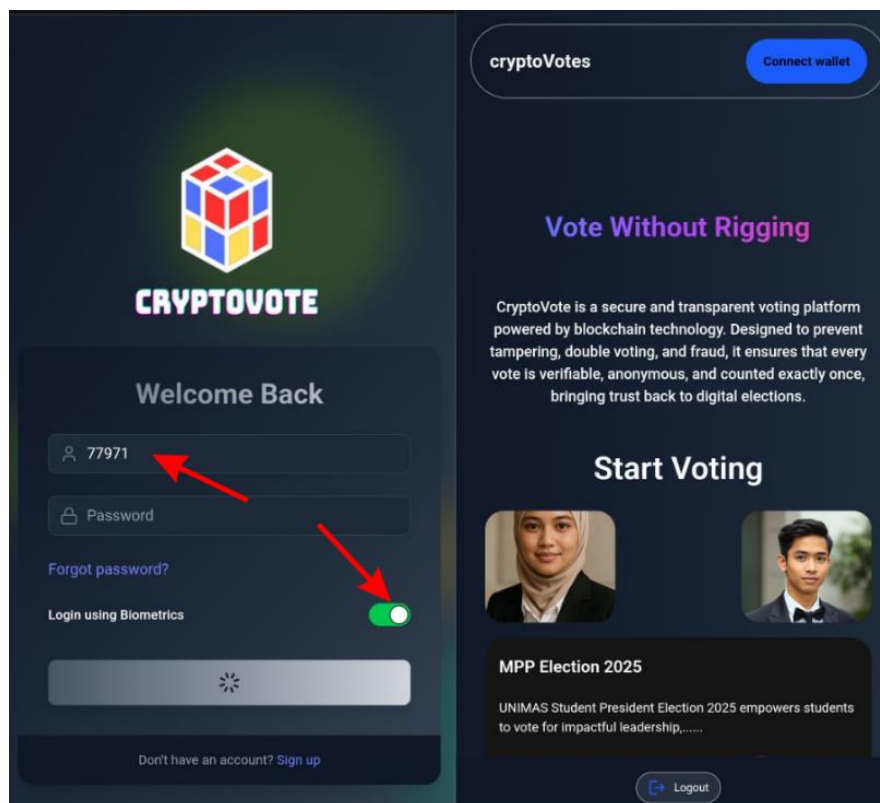
Screenshot:



Test Case ID: TC02 Login With Biometric

<u>Item</u>	<u>Details</u>
Description	This test verifies that a student who has previously registered biometric credentials can log in using only their matric number and biometric authentication. No password input is required
Input	Matric Number: 77971 Biometric: User click “Login with Biometric” button
Expected Output	WebAuthn prompt appears asking the user to verify their identity via biometric (e.g. fingerprint). Upon success, the user is redirected to Home page.
Actual Output	Login succeeded. User was redirected to the Home page

Screenshot:

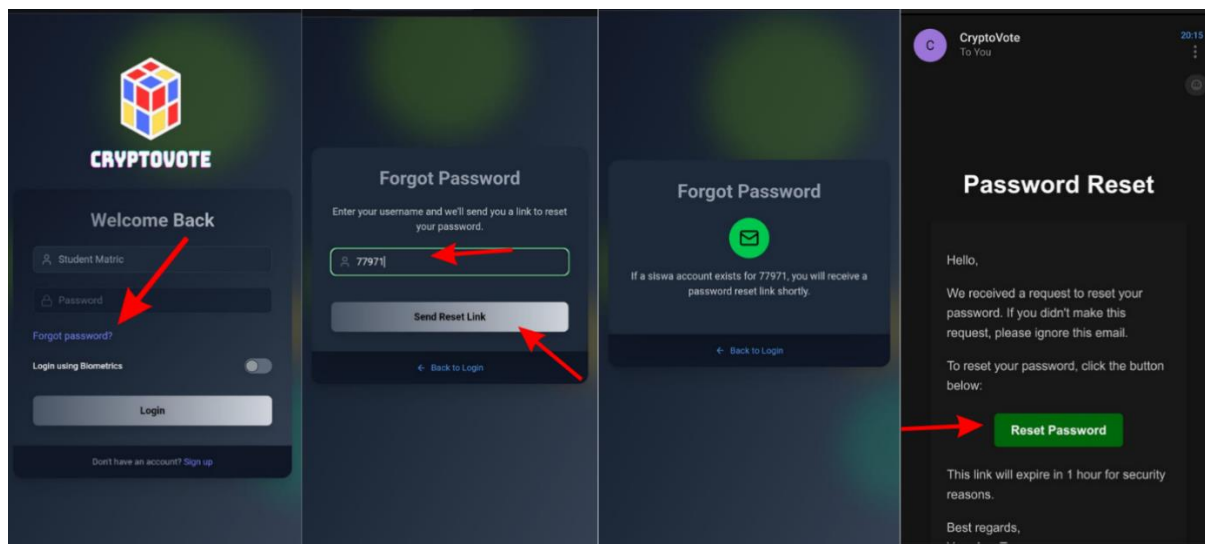


5.3.1.3 Forgot Password Test Case

Test Case ID: TC01 Receive Password Reset Email

<u>Item</u>	<u>Details</u>
Description	This test verifies that a student who has previously registered biometric credentials can log in using only their matric number and biometric authentication. No password input is required
Input	Matric Number: 77971
Expected Output	After entering a valid matric number, the system sends a reset password email to the student's university email address with a link to reset password.
Actual Output	Reset link was successfully sent via email.

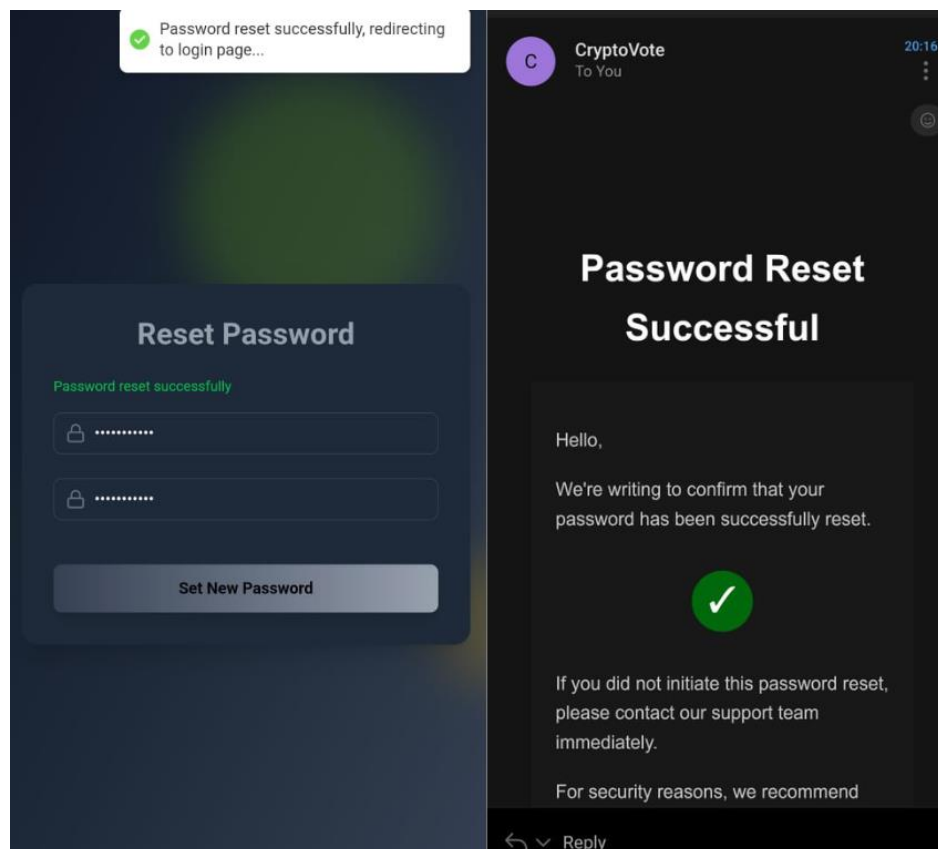
Screenshot:



Test Case ID: TC02 Reset Password

<u>Item</u>	<u>Details</u>
Description	This test verifies that a student can reset their password using the link sent via email. After entering and confirming the new password, the system should update it successfully.
Input	New Password: NewUnimasPass2025! Confirm Password: NewUnimasPass2025!
Expected Output	The system saves the new password, shows a success message on the frontend, and sends a confirmation email to the student's university email address.
Actual Output	The password was updated successfully. A success message appeared on the UI, and a confirmation email was received in the inbox

Screenshot:

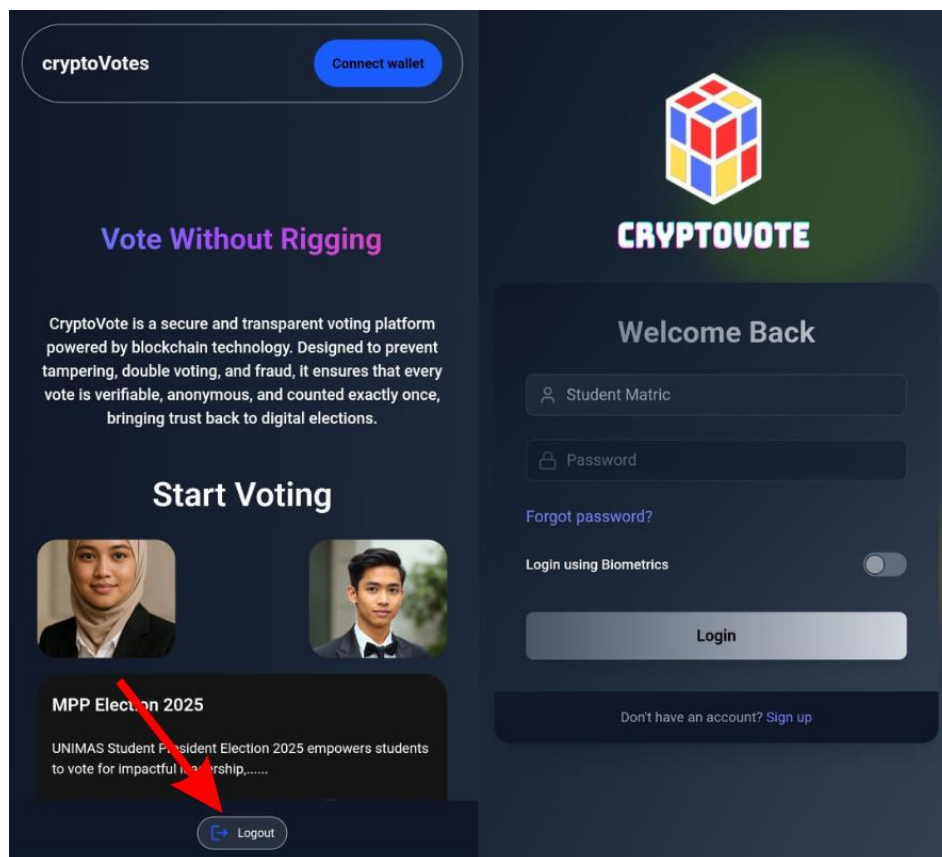


5.3.1.4 Logout Test Case

Test Case ID: TC01 Logout

<u>Item</u>	<u>Details</u>
Description	This test verifies that a logged-in user can securely log out of the system by clicking the “Logout” button from the Home page.
Input	Action: Click on the “Logout” button
Expected Output	The system clears the user session and redirects them to the login page
Actual Output	After clicking the logout button, the session was cleared and the user was redirected to the login page.

Screenshot:



5.4 Non-Functional Testing

Non-functional testing in this project aimed to evaluate the overall user experience, usability, and satisfaction of the CryptoVote system. The assessment was conducted through a structured questionnaire distributed via Google Forms, targeting actual university students across different faculties. This approach focused on understanding how real users interacted with the system in terms of ease of use, reliability, and overall satisfaction. The questionnaire was divided into multiple sections covering authentication, wallet connection, voting activities, and general user feedback. Responses were analyzed both quantitatively and qualitatively to assess how well the system meets usability expectations.

5.4.1 Participants

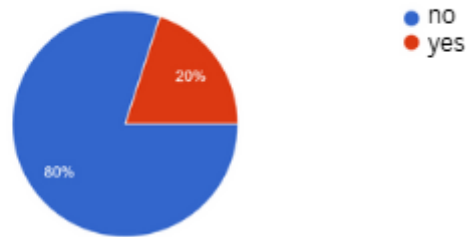
A total of fifteen (15) UNIMAS students participated in the non-functional testing of the CryptoVote system. The majority of the participants (12 students) were from the **Faculty** of Computer Science and Information Technology (FCSIT), while the remaining included two students from the Faculty of Social Sciences and Humanities (FSSH) and one from the Faculty of Economics and Business (FEB). This diverse distribution helped assess the usability of the platform across varying technical backgrounds.

Out of the 15 participants, only two students had prior experience using a crypto wallet such as MetaMask, while the remaining thirteen were first-time users. This demographic provided a meaningful evaluation of the system's learnability, especially in terms of onboarding new users into a blockchain-based voting platform.

5.4.2 Authentication and Registration Experience

All 15 participants successfully completed the sign-up and login process using their UNIMAS student matric number and official @siswa.unimas.my email. This confirmed that the core authentication mechanism—consisting of OTP verification, password setup, and optional biometric login—was functioning as intended.

Did the OTP email arrive in your Inbox (not Junk/Spam)?
15 responses



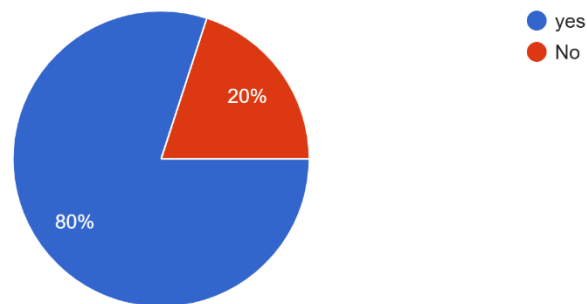
However, 2 participants reported that the OTP verification email was delivered to their spam/junk folder, instead of the inbox. Although this did not block the registration process, it indicates a need to improve email deliverability or prompt users to check their spam folder during registration.

The remaining authentication features, including OTP verification, password creation, login, logout, and optional biometric passkey login, worked without any issues. No participants reported failure in these areas, and those who opted to test the biometric feature were able to create and use it successfully.

5.4.3 Authentication and Registration Experience

As part of the voting process, participants attempted to connect their crypto wallet (MetaMask) to the CryptoVote platform. Out of 15 total respondents, 20% which are 3 participants reported being unable to connect their wallet successfully. As a result, these participants were unable to proceed with the subsequent steps, including claiming the voting token, casting a vote, and receiving a confirmation email. Their experience highlights a usability barrier, especially for users unfamiliar with crypto wallet integration, which led to frustration and early drop-off from the process.

Were you able to connect your crypto wallet (e.g., MetaMask)?
15 responses



The remaining 80% (12 participants) were able to connect their wallet without issues. These users proceeded through the full voting workflow smoothly. They successfully claimed their Sepolia ETH test tokens using the “Claim Token” feature and were able to vote for their preferred contestant. After casting their vote, they received vote confirmation emails, and the system effectively prevented any attempt at double voting, demonstrating that the smart contract’s constraints were functioning correctly.

This feedback indicates that while the blockchain interaction and voting mechanisms were robust for most users, wallet connection remains a critical usability area that may benefit from additional guidance or onboarding support particularly for users with no prior experience using crypto wallets.

5.4.4 Usability and User Satisfaction Ratings

To assess the overall user experience, participants were asked to rate their satisfaction with both the authentication process and the wallet-based voting activities using a 7-point Likert scale (1 = Strongly Disagree, 7 = Strongly Agree). The results are presented in two parts: authentication-related experience and blockchain voting-related experience.

Authentication and Registration Experience

Participants generally reported high satisfaction with the sign-up and login process. As shown in Table 5.5.5.1, the sign-up process, OTP verification, login, logout, and password reset features received strong ratings, all averaging above 6.0. Notably, the sign-up, login, and logout

processes scored an average of 6.5, suggesting that users found these steps intuitive and efficient. Even the optional biometric login feature was positively received, with an average rating of 6.2, indicating that users appreciated the added convenience.

No.	Statement	Avg Rating	Min	Max
1	The sign-up process was straightforward and easy to complete.	6.5	6	7
2	The OTP verification was reliable and timely.	6.2	5	7
3	I found the login experience smooth and easy to understand.	6.5	6	7
4	I found the logout experience smooth and easy to understand.	6.5	6	7
5	Resetting my password was clear and worked without issues.	6.2	5	7
6	I felt the biometric login option added convenience to the process.	6.2	5	7

Wallet Connection and Voting Experience

As shown in Table 5.5.5.2, participants were generally satisfied with the wallet connection and voting process, although the average scores were slightly lower compared to the authentication section. All five items received an average rating of 5.8 or higher. The highest-rated statement was overall satisfaction with the system (6.0), indicating that most users had a positive experience despite some initial technical hurdles. A small portion of participants who encountered wallet connection issues were reflected in the lower minimum rating of 2 across multiple items. This suggests a mixed experience among users unfamiliar with crypto wallets.

No.	Statement	Avg Rating	Min	Max
1	Connecting my wallet was easy and well-guided.	5.8	2	7
2	The token claiming process was smooth and clearly communicated.	5.8	2	7

3	I found the login experience smooth and easy to understand.	5.8	2	7
4	The voting flow was simple, secure, and easy to complete.	5.8	2	7
5	I felt confident that my vote was recorded accurately.	6.0	2	7
6	Overall, I had a positive experience interacting with the voting system.	6.2	4	7

5.5 Summary

Chapter 5 detailed the testing phase of the CryptoVote system, focusing on validating all key functionalities to ensure system reliability and correctness. The test cases were designed and executed by the developer, covering each user role—admin, contestant, and voter—to ensure that all critical paths in the voting process were thoroughly verified. The results showed that all core features performed as expected under both normal and edge-case conditions, and each function could be executed with minimal issues. This testing confirmed that CryptoVote meets its intended functional requirements and provides a secure and robust voting experience.

The non-functional testing of CryptoVote involved 15 participants from various faculties, with only two having prior experience using crypto wallets. All users successfully completed the authentication process, though two reported OTP emails landing in their spam folder. While 80% of users connected their wallets and completed the voting process smoothly, 20% faced wallet connection issues and could not proceed further. Usability ratings were high for the authentication process (average 6.2–6.5/7) and slightly lower for voting-related tasks (around 5.8/7). Overall, the results indicate strong user satisfaction and confirm the system’s readiness for real-world deployment, with minor improvements needed in wallet onboarding.

Chapter 6: Conclusion and Future Work

6.1 Overview

This chapter summarizes the outcomes of the CryptoVote system development and evaluates the extent to which the project's goals were achieved. The chapter also outlines potential directions for future enhancements, with the aim of further improving usability, autonomy, and scalability of the system.

6.2 Conclusion

As stated in Chapter 1, the objectives of the proposed system, CryptoVote, were as follows:

- To design and develop a secure, transparent, and accessible blockchain-based voting system for university student elections.
- To implement a blockchain system that ensures a high level of security, prevents unauthorized access, and maintains data integrity throughout the voting process.
- To test the functionality of the blockchain-based voting system among UNIMAS students.

All of these objectives have been successfully achieved through the development of the CryptoVote platform. The system enables secure poll creation, contestant registration, and verifiable voting using Ethereum smart contracts. By integrating crypto wallet public address, each participant's identity is securely linked to their actions in the smart contract, ensuring transparency and traceability without compromising voter anonymity.

In addition, all functional requirements outlined in Chapter 3 Section 3.3.1 were successfully achieved. The table below maps each requirement to the corresponding feature implemented in CryptoVote:

Table 7.1. Functional Requirements and Implementation Status

Functional Requirement	Description	Status
User Registration and Authentication	Secure sign-up with matric number, email OTP, password setup, and optional biometric login	Achieved

Generate Voter Public/Private Key	Wallet connection (e.g., MetaMask) manages cryptographic keys for secure user identity	Achieved
Election Management	Admin can create, update, and delete polls through the frontend interface	Achieved
Vote Casting	Users vote using their connected wallet and system enforces one vote per user per poll	Achieved
Smart Contracts	Poll logic and voting handled entirely by Ethereum smart contracts	Achieved
Blockchain Storage	All critical data such as polls, votes, contestants stored immutably on the blockchain	Achieved
Vote Receipt	Users receive on-screen and wallet confirmation after voting	Achieved
Voter Anonymity	Users are identified only by wallet address, not by student identity	Achieved
Auditability and Transparency	All transactions are publicly verifiable on Etherscan for full transparency	Achieved

Furthermore, the non-functional testing and User Acceptance Testing (UAT) demonstrated that most users found the system easy to use, particularly in authentication and voting. A majority were able to complete all tasks, including sign-up, wallet connection, and vote casting. However, the UAT also revealed that a small portion of users faced difficulties during the wallet connection process, with some unable to proceed to voting. These insights

validate that the system meets its intended purpose while highlighting real-world usability challenges that must be addressed moving forward.

Although the initial scope proposed a native mobile application, the system was implemented as a mobile-optimized web platform. This decision ensures compatibility with mobile crypto wallets, such as MetaMask Mobile, which are essential for blockchain interaction. The final product works seamlessly across both desktop and mobile browsers, and functional testing among UNIMAS students confirmed that the system performs reliably and as intended in real-use conditions.

6.3 Future Work

While CryptoVote successfully demonstrates the implementation of blockchain to enhance the integrity, transparency, and security of university elections, several opportunities remain to improve the system for broader adoption and real-world deployment.

One important direction is the development of a built-in crypto wallet within the application. Currently, users must connect external wallets such as MetaMask, which may introduce friction especially for non-technical users unfamiliar with blockchain tools. Integrating a native wallet directly into the platform would eliminate this dependency, simplify the user experience, and offer better control over key management while preserving user privacy and security. This concern was supported by findings during User Acceptance Testing (UAT), where several participants faced difficulties connecting their wallets, and some could not proceed with voting as a result. A built-in wallet would reduce technical barriers, simplify onboarding, and allow tighter control over wallet functionality and user flow—making the system more inclusive and seamless.

Next, a fully native mobile application for Android and iOS are also considered in future iterations. While the current web-based system is responsive and functional on mobile devices, a dedicated app could enhance performance, allow tighter integration with native features (such as biometric login), support offline readiness, and provide a more seamless user experience overall.

Lastly, since some participants reported that OTP emails ended up in the junk folder which is likely due to Outlook's strict spam filtering, future iterations will explore on

integrating UNIMAS's official Outlook SMTP service or alternative delivery methods to improve email reliability and ensure prompt OTP delivery during sign-up.

6.4 Summary

Chapter 6 concluded the CryptoVote project by evaluating the system's success in achieving its objectives, discussing completed features, and identifying potential improvements. The system effectively delivered a secure, transparent, and accessible blockchain-based voting platform tailored to UNIMAS MPP elections, with functional features such as smart contract, public and private key system, and a mobile-optimized interface.

Future enhancements were also outlined, including plans to integrate a built-in crypto wallet and develop native mobile applications. These improvements aim to make CryptoVote more seamless, autonomous, and scalable for institutional use, laying the foundation for broader real-world adoption in digital elections

REFERENCE

- Dutta, P., Choi, T., Somani, S., & Butala, R. (2020). Blockchain technology in supply chain operations: Applications, challenges and research opportunities. *Transportation Research Part E Logistics and Transportation Review*, 142, 102067. <https://doi.org/10.1016/j.tre.2020.102067>
- Follow My Vote. (2016). Follow My Vote whitepaper. Follow My Vote. Retrieved from <https://followmyvote.com/wp-content/uploads/2016/09/Follow-My-Vote.pdf>
- Capece, G., Ghiron, N. L., & Pasquale, F. (2020). *Blockchain technology: Redefining trust for digital certificates*. *Sustainability*, 12(21), 8952. <https://doi.org/10.3390/su12218952>
- Harris, L. (2017, September 27). TCU Senate uses new voting process. *The Tufts Daily*. <https://www.tuftsdaily.com/article/2017/09/tcu-senate-uses-new-voting-process>
- Jumani, N. M. A., Du Yujie, N., & Khan, N. M. O. (2023). The Future of Democracy: Exploring the Potential of Block Chain-Based Digital Voting Systems. *International Journal of Scientific Research in Computer Science Engineering and Information Technology*, 221–231. <https://doi.org/10.32628/cseit2390633>
- Khan, P. W., & Byun, Y. (2020). A Blockchain-Based secure image encryption scheme for the industrial internet of things. *Entropy*, 22(2), 175. <https://doi.org/10.3390/e22020175>
- Khan, S. M. A. (2023). *Waterfall model used in software development: Reference software requirements engineering waterfall model*. ResearchGate. https://www.researchgate.net/publication/371902449_Waterfall_Model_Used_in_Software_Development_Reference_Software_Requirements_Engineering_Waterfall_Model
- Liu, Z., Yang, G., Wong, D. S., Nguyen, K., & Wang, H. (2019). Key-Insulated and Privacy-Preserving Signature Scheme with Publicly Derived Public Key. 2019 IEEE European Symposium on Security and Privacy (EuroS&P), 215–230. <https://doi.org/10.1109/eurosp.2019.00025>

- Malkawi, M., Yassein, M. B., & Bataineh, A. (2021). Blockchain based voting system for Jordan parliament elections. *International Journal of Power Electronics and Drive Systems/International Journal of Electrical and Computer Engineering*, 11(5), 4325. <https://doi.org/10.11591/ijece.v11i5.pp4325-4335>
- Mccorry, P., Mehrnezhad, M., Toreini, E., Shahandashti, S. F., & Hao, F. (2021). On Secure E-Voting over Blockchain. *Digital Threats Research and Practice*, 2(4), 1–13. <https://doi.org/10.1145/3461461>
- MIT News. (2020, February 13). MIT researchers uncover security vulnerabilities in voting app. MIT News. Retrieved from <https://news.mit.edu/2020/voting-voatz-app-hack-issues-0213>
- Nakamoto, Satoshi, Bitcoin: A Peer-to-Peer Electronic Cash System (August 21, 2008). Available at SSRN: <https://ssrn.com/abstract=3440802> or <http://dx.doi.org/10.2139/ssrn.3440802>
- Shevtsiv, N., Shvets, D., & Karabut, N. (2019b). Prospects for using React Native for developing cross-platform mobile applications. *Central Ukrainian Scientific Bulletin Technical Sciences*, 2(33), 208–213. [https://doi.org/10.32515/2664-262x.2019.2\(33\).208-213](https://doi.org/10.32515/2664-262x.2019.2(33).208-213)
- Specter, M. A., Koppel, J., & Weitzner, D. (2020). The Ballot is Busted Before the Blockchain: A Security Analysis of Voatz, the First Internet Voting Application Used in {U.S}. *FederalElections*. <https://www.usenix.org/conference/usenixsecurity20/presentation/specter>
- Sunny, F. A., Hajek, P., Munk, M., Abedin, M. Z., Satu, M. S., Efat, M. I. A., & Islam, M. J. (2022). A Systematic Review of Blockchain Applications. *IEEE Access*, 10, 59155–59177. <https://doi.org/10.1109/access.2022.3179690>

Solidity Documentation. (n.d.). *Solidity 0.8.30 documentation*. Soliditylang.org.
<https://docs.soliditylang.org/en/v0.8.30/>

Taherdoost, H. (2023). Smart contracts in Blockchain Technology: A critical review. *Information*, 14(2), 117. <https://doi.org/10.3390/info14020117>

Voatz. (2020). Voatz security whitepaper. Voatz. Retrieved from <https://voatz.com/wp-content/uploads/2020/07/voatz-security-whitepaper.pdf>

Vladucu, M., Dong, Z., Medina, J., & Rojas-Cessa, R. (2023). E-Voting meets Blockchain: a survey. *IEEE Access*, 11, 23293–23308. <https://doi.org/10.1109/access.2023.325368>

APPENDICES

Appendix A: User Evaluation Questionnaire

Section 1: Demographic Data

- Full Name
- Which Faculty are you from?
- Do you have any experience using a crypto wallet before?

☐ Yes

☐ No

Section 2: System Usability Experience

Part A: Authentication and Registration (Sign Up, Login, etc.)

- Were you able to sign up successfully using your matric number and email?
 - ☐ Yes ☐ No
- Did you receive the OTP code email after entering your matric number?
 - ☐ Yes ☐ No
- Did the OTP email arrive in your Inbox (not Junk/Spam)?
 - ☐ Yes ☐ No
- Were you able to create a password after OTP verification?
 - ☐ Yes ☐ No
- *(Optional)* Were you able to create a passkey by clicking 'Enable Biometric' during the password creation stage?
 - ☐ Yes ☐ No
- Were you able to log in using your password?
 - ☐ Yes ☐ No

- *(Optional)* Were you able to log in using biometric authentication (fingerprint/Face ID)?
 - ☐ Yes ☐ No
- Were you able to logout successfully?
 - ☐ Yes ☐ No
- *(Optional)* Did you try using the "Forgot Password" feature?
 - ☐ Yes ☐ No
- *(Optional)* After resetting your password, were you able to log in again successfully?
 - ☐ Yes ☐ No

The following statements are rated on a 7-point Likert scale (1 = Strongly Disagree, 7 = Strongly Agree):

- The sign-up process was straightforward and easy to complete.
- The OTP verification was reliable and timely.
- I found the login experience smooth and easy to understand.
- I found the logout experience smooth and easy to understand.
- *(Optional)* Resetting my password was clear and worked without issues.
- *(Optional)* I felt the biometric login option added convenience to the process.

Part B: Voting Activities

- Is this your first time using a crypto wallet?
 - ☐ Yes ☐ No
- Were you able to connect your crypto wallet (e.g., MetaMask)?
 - ☐ Yes ☐ No
- Did you successfully claim Sepolia ETH using the “Claim Token” button?

☐ Yes ☐ No

- Were you able to vote for a contestant during the test?

☐ Yes ☐ No

- After voting, did the system prevent you from voting again?

☐ Yes ☐ No

- Did you receive the vote confirmation email?

☐ Yes ☐ No

The following statements are rated on a 7-point Likert scale (1 = Strongly Disagree, 7 = Strongly Agree):

- Installing and setting up a crypto wallet was difficult for me.
- Connecting my wallet was easy and well-guided.
- The token claiming process was smooth and clearly communicated.
- The voting flow was simple, secure, and easy to complete.
- I felt confident that my vote was recorded accurately.
- Overall, I had a positive experience interacting with the voting system.

Section 3: User Experience Questionnaire (UEQ)

Please rate your experience based on the following adjective pairs:
(1 = Fully agree with left word, 5 = Fully agree with right word)

Pair	Rating (1–5)
Difficult to learn → Easy to learn	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Boring → Exciting	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Unattractive → Attractive	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Pair	Rating (1–5)
Impractical → Practical	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
Confusing → Clear	☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Section 4: Open Feedback

- What did you find confusing or difficult while using the system?
-
- Do you have any suggestions to improve the sign-up, login, or voting process?
-
- Did you experience any errors or bugs? If yes, please describe them.
-
- Do you have any general feedback or additional comments?