MAR IVANIOS COLLEGE (AUTONOMOUS)

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram – 695015

B.Sc. Computer Science Major Project (Phase I) Report

BLOCKCHAIN BASED HYBRID VOTING SYSTEM

A report submitted in partial fulfilment of the requirement for the Fifth Semester BSc. Computer Science

SUBMITTED BY

Akhil Vinod	**(2220813)**
Hari Sankar R S	**(2220819)**
Pradhisha D	**(2220808)**
Renjith M	**(2220824)**

Under the guidance of

Ms. Jisha Isaac

DEPARTMENT OF COMPUTER SCIENCE

BSc Computer Science

2024

MAR IVANIOS COLLEGE (AUTONOMOUS)

Mar Ivanios Vidya Nagar, Nalanchira

Thiruvananthapuram – 695015

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled "**BLOCKCHAIN BASED HYBRID VOTING SYSTEM**" is a bonafide record of the work done by **Akhil Vinod**(2220813), **Hari Sankar R S** (2220819), **Pradhisha D** (2220808), **Renjith**(2220824) in partial fulfilment of the requirements for the award of the Degree of Bachelor of Science in Computer Science by the University of Kerala.

INTERNAL GUIDE HEAD OF THE DEPARTMENT

EXTERNAL EXAMINERS

1.

2.

ACKNOWLEDGEMENT

With profound gratitude to the **ALMIGHTY,** we take this chance to thank people who helped us to complete this project.

We take this as a right opportunity to say **THANKS** to our **PARENTS** who are there to stand with us always with the words “YOU CAN”.

We are thankful to Principal **DR. MEERA GEORGE**, **Mar Ivanios College** who gave us the platform to establish ourselves to reach greater heights.

We earnestly thank **DR. K. OOMMACHAN**, Director, who always encourages us to do novel things and provides all facilities.

We express our sincere gratitude to **MS. TINU C PHILIP**, Head, Department of Computer Science, for her valuable guidance and support to execute all inclines in learning.

It is our delight to thank our project guide **MS. JISHA ISAAC**, Assistant Professor, Department of Computer Science for her help, support, encouragement, suggestions and guidance throughout the development phases of the project.

We convey our gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

A great note of gratitude to friends and people who are known and unknown to us and helped in carrying out this project work a successful one.

Akhil Vinod

Hari Sankar R S

Pradisha D

Renjith M

ABSTRACT

The Blockchain Based Hybrid Voting System introduces a transformative approach to secure, tamper-proof and efficient voting by introducing blockchain technology. Traditional voting methods face several challenges, including inefficiencies, lack of transparency, and risks of manipulation, which undermine trust in the democratic process. This project proposes a hybrid architecture to make a tamper proof record of votes using blockchain technology while giving consideration to scalability of the system. By combining the strengths of a public blockchain network (Solana) with a permissioned blockchain network (Hyperledger Fabric), the system ensures both transparency and scalability. It enhances accessibility, allowing voters to participate from any location with an internet connection through a smartphone, while ensuring security through advanced verification mechanisms. The system also prioritises transparency, offering an immutable and auditable record of election results. This initiative is a significant step toward redefining how elections are conducted, aiming to establish a scalable, secure, and globally trustworthy framework that upholds the integrity of democratic processes in the digital age.

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTML:	Hypertext Markup Language
SSD:	Solid State Drive
RAM:	Random Access Memory
CPU:	Central Processing Unit
GUI:	Graphical User Interface

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The voting process is an integral part of a democratic system, yet traditional methods have inefficiencies, security concerns, and accessibility barriers. From long queues at polling stations to concerns about tampering and manipulation of results, these challenges threaten the integrity and trust in electoral systems. In response to these issues, the **Blockchain Based Hybrid Voting System** leverages the transformative power of blockchain technology to establish a more secure, transparent, and accessible voting infrastructure.

At the core of this project lies a hybrid blockchain architecture that combines the scalability of a permissioned blockchain (Hyperledger Fabric) with the transparency of a public blockchain (Solana). This architecture ensures that individual votes are securely stored in a private network, while aggregate results are made publicly auditable. Key features such as biometric verification for voter authentication, and mobile application support make the system robust, user-friendly, and scalable to accommodate large voter populations.

By seamlessly integrating these advanced technologies, this project addresses critical shortcomings in existing voting systems. It aims to enhance trust in democratic processes by providing a transparent, tamper-proof, and efficient voting mechanism. The proposed solution eliminates geographical and physical barriers, making voting more inclusive while ensuring the security and anonymity of each vote. This initiative paves the way for a modernised electoral framework capable of meeting the demands of a digitally connected world.

1.1 PURPOSE OF THE PROJECT

The project aims to strengthen the security and trust in a voting process by integrating blockchain technology, and biometric verification, preventing fraud such as double voting, impersonation, or tampering of votes. This ensures that only authenticated individuals can cast their vote and votes cannot be altered after the election. By utilising blockchain technology, the system guarantees an immutable and auditable ledger of votes. This transparency allows for public verification of election results, fostering greater trust in the electoral process.

The project enables voters to participate in elections from any location with internet access, removing geographical barriers. This makes voting more convenient and inclusive, particularly for individuals who face difficulties in attending physical polling stations. The project aims to build a hybrid architecture by the use of both a public blockchain (Solana) and a permissioned blockchain (Hyperledger Fabric) ensures that voting data is securely recorded and cannot be altered. This reduces the risks of vote tampering or result manipulation, safeguarding the integrity of elections.

The hybrid system is designed to handle large volumes of votes efficiently, ensuring scalability for national-level elections. In doing so, it can cater to millions of voters without compromising on performance or security. By combining advanced technologies, this project aims to rebuild trust in electoral systems, ensuring that the election results reflect the will of the people. This helps restore faith in the democratic process and encourages wider civic participation.

1.2 OBJECTIVE

The primary objective of this project is to create a secure, transparent, and accessible voting platform that addresses the issues and inefficiencies of traditional voting methods. By utilizing blockchain technology, the system ensures that election results are immutable and tamper-proof, which is essential for maintaining the integrity and trust in the electoral process. In doing so, the system aims to eliminate issues such as vote manipulation, fraud, and inefficiencies that currently hinder the effectiveness of conventional voting systems.

Another key objective is to enhance voter accessibility, allowing individuals to cast their votes from any location with an internet connection. This eliminates geographical barriers and long queues at polling stations, making the voting process more inclusive and convenient. In doing so, the system ensures that all eligible voters, including those with mobility issues or in remote areas, can participate in the electoral process without facing undue challenges.

Additionally, the project focuses on improving the security of the voting process by integrating biometric verification mechanisms. Through face recognition technology, the system ensures that only authenticated voters can participate, which significantly reduces the risk of fraudulent activities such as impersonation or double voting. This enhances the overall security of the system and instil confidence in voters, knowing that their vote will be securely cast and counted.

Lastly, scalability is a critical objective, as the system is designed to accommodate large-scale elections with millions of voters. By leveraging a hybrid blockchain architecture that combines a public blockchain (Solana) for transparent results and a permissioned blockchain (Hyperledger Fabric) for secure vote recording, the system ensures both scalability and efficiency. Individual votes are recorded in the permissioned blockchain network and only a tally of total votes in specific checkpoints are added to the public blockchain network. In doing so, the solution is capable of handling high volumes of votes without compromising performance or security, making it suitable for both local and national-level elections.

CHAPTER 2: LITERATURE SURVEY

Blockchain-based e-voting systems have been widely researched for their potential to enhance security, transparency, and decentralisation. A comprehensive review examined 252 papers, highlighting critical challenges such as accessibility, compatibility, and usability in existing systems. The study emphasised the importance of hybrid blockchain models to improve transparency and scalability while ensuring auditability, making them a viable solution for modern voting systems [1].

Another study explored a decentralised voting system that utilises Aadhaar and OTP-based verification for voter authentication. The system employs a peer-to-peer network for decentralised ledger management, ensuring secure and traceable vote records. Despite these advancements, the lack of biometric authentication remains a limitation, which highlights the need for further integration of advanced security measures to enhance voter verification [2].

Research into privacy-preserving score voting systems has utilised Ethereum-based smart contracts to ensure voter anonymity and decentralised voting. While these systems effectively maintain privacy, vulnerabilities in cryptographic techniques were noted, which could compromise vote integrity. Incorporating advanced cryptographic mechanisms, such as Zero-Knowledge Proofs, has been proposed to address these challenges and ensure robust voter anonymity and verification [3].

The EtherVote system, built on the Ethereum blockchain, eliminates the need for central authority servers and focuses on security, privacy, and transparency. By decentralising the voting process, it addresses key issues in traditional voting systems. However, the system's reliance on a single blockchain network introduces limitations in scalability, which necessitates the exploration of hybrid blockchain architectures for handling larger populations efficiently [4].

The Bie Vote framework combines biometric identification with blockchain technology to enhance security and transparency in voting systems. This study utilises Hyperledger Fabric to integrate biometric authentication, addressing the limitations of traditional e-voting systems. However, concerns regarding scalability and usability remain,

suggesting the need for further research to adapt such frameworks for broader applications and diverse voting environments [5].

A study discussing open issues in blockchain-based e-voting systems highlights challenges such as scalability, usability, and privacy concerns. The research provides a roadmap for overcoming these challenges, but practical implementations remain limited. Advancements in hybrid blockchain architectures and security frameworks are essential for addressing these issues and achieving scalable, secure, and transparent e-voting systems [6].

CHAPTER 3: SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The existing voting systems, particularly in India, are based on traditional ballot methods and electronic voting machines (EVMs). While these systems have been in place for decades, they suffer from several issues that undermine their efficiency, security, and accessibility

- **Inefficiency in Voter Participation:** Traditional voting systems often involve long queues, resulting in delays and inconvenience for voters. This inefficiency discourages some voters from participating in elections, particularly in remote areas.
- **Risk of Vote Tampering and Manipulation:** Electronic voting machines, while designed to be secure, are still vulnerable to hacking, tampering, and manipulation. In some cases, security breaches have raised concerns about the integrity of the election results. In recent times there are cases in which votes are tampered after the election by the officials, raising concerns to the integrity of the current electoral system.
- **Limited Transparency and Accountability:** The current systems lack complete transparency, as the process of vote counting and result tallying remains opaque. This absence of transparency can lead to mistrust among the electorate, especially if there are allegations of vote manipulation or inaccuracies in the results.
- **Inaccessibility for Disabled or Remote Voters:** Many individuals, especially those with physical disabilities or those living in geographically remote areas, face significant barriers in accessing polling stations. The existing system does not offer adequate solutions to ensure that these groups can participate in elections without undue difficulty.
- **Vulnerability to Fraud and Multiple Voting:** Traditional systems are prone to issues like multiple voting, voter impersonation, and other fraudulent activities. Although efforts have been made to curb such practices, the current system lacks an effective, foolproof mechanism for ensuring that only eligible voters participate.

3.2 PROPOSED SYSTEM

The proposed system aims to revolutionize the current voting process by including cutting edge technologies like blockchain technology, cryptography, biometric verification, a user friendly mobile application to increase the security, accessibility and build a trustless system.

- **Hybrid Blockchain Architecture:** The proposed system utilises a combination of a public blockchain (Solana) and a permissioned blockchain (Hyperledger Fabric). This hybrid approach ensures both transparency and privacy, with the public blockchain tracking the final results for public auditability, while the permissioned blockchain securely stores individual votes to prevent tampering and unauthorised access.
- **Biometric Voter Authentication:** To enhance security and prevent fraudulent activities like voter impersonation or double voting, the system integrates biometric verification. Voters will authenticate themselves through face recognition, ensuring that only eligible individuals can cast their votes, thereby reducing the chances of fraud or impersonation.
- **Enhanced Vote Privacy and Integrity:** The system ensures voter privacy and vote integrity by employing advanced cryptographic techniques. This guarantees anonymity while upholding the transparency and correctness of the electoral process.
- **Mobile App for Accessibility:** The system includes a mobile application developed with Flutter, enabling voters to cast their votes conveniently from any location with internet access. The app ensures the voting process is user-friendly and accessible to a wide range of individuals, overcoming barriers related to physical polling stations and enabling greater participation.

3.3 FEASIBILITY STUDY

A feasibility study is carried out before the project is developed to assess the likelihood that the suggested system will be successful. To ascertain whether developing a new or enhanced system is compatible with cost, benefits, operation and technology, a feasibility study is required.

3.3.1 Technical Feasibility

The architecture of combining Solana (a public blockchain) and Hyperledger Fabric (a permissioned blockchain) is inherently scalable and secure. Solana's high throughput capability (with 50,000+ transactions per second) ensures that the system can efficiently handle large-scale voting, such as national elections, while maintaining transparency. Hyperledger Fabric, on the other hand, allows for private transactions, where individual votes are securely recorded, ensuring confidentiality. Hyperledger Fabric's modular architecture supports smart contracts (Chaincode written in Go), which can automate vote validation and record-keeping. Integration of these two blockchain types ensures that the system is both scalable and secure, a feature that has been successfully implemented in other blockchain-based voting systems for large populations.

For voter authentication, the system employs DeepFace, a Python-based library for facial recognition, which ensures that only eligible voters can cast their ballots. In addition, Google ML Kit is integrated to perform liveness detection, ensuring that the face being recognized belongs to a live person and not a static image, further enhancing security. This combination of biometric verification and liveness detection reduces the risk of impersonation or fraud. These technologies are widely used in security applications and have been proven effective in similar contexts.

A Flutter-based mobile application will allow users to participate in elections from their mobile devices, ensuring accessibility and scalability. Flutter's cross-platform capabilities make it ideal for building apps that run on Android, iOS, and other platforms from a single codebase. The app will interact with the blockchain using REST APIs to communicate with Hyperledger Fabric for vote recording and Solana for vote tallying. This ensures seamless mobile interaction with a hybrid blockchain system, combining blockchain's security with the flexibility and accessibility of a mobile platform.

The system employs advanced cryptographic techniques, including hashing and digital signatures, to ensure the integrity and security of each vote. Hashing ensures that the

votes are immutable once recorded, while digital signatures verify the authenticity of each vote. These cryptographic methods are integrated into Hyperledger Fabric chaincode, which ensures that votes are securely stored and cannot be altered. Additionally, blockchain's inherent design allows for transparent auditability without revealing voter identities, addressing concerns related to privacy in elections. The use of these technologies has been demonstrated in several blockchain-based e-voting systems, confirming the technical viability of this approach for secure elections

3.3.2 Social Feasibility

By allowing voting through a mobile app, the system can significantly increase voter turnout, especially among younger, tech-savvy voters. The convenience of voting from anywhere with an internet connection removes barriers for those who find it difficult to visit polling stations, such as people with disabilities or individuals living in remote areas.

The decentralised nature of blockchain means that no single entity controls the voting process, which could lead to greater confidence among voters. Knowing that the election results are recorded on a public, tamper-proof ledger can help diminish concerns about government manipulation or external interference in the election process. This sense of security can increase overall participation and trust in the democratic system.

A blockchain-based voting system can create a more engaging and interactive voting experience. By providing real-time results and fostering transparency, it encourages voters to stay involved throughout the election process. This could potentially lead to increased interest in elections and greater civic participation, as people see firsthand how their votes contribute to the outcome.

3.3.3 Operational Feasibility

The mobile app developed with Flutter is designed to be intuitive and easy to use, making it accessible for voters of all technical levels. With clear navigation, instructions, and real-time prompts, users can easily authenticate, vote, and track their vote status without facing complex processes. This simplicity encourages widespread adoption, as users don't need specialised knowledge to participate, ensuring the system can be used efficiently even by those with minimal technological experience.

By utilising Hyperledger Fabric as the permissioned blockchain for vote recording, operational costs can be significantly reduced. Unlike traditional voting systems that require extensive physical infrastructure for ballot collection, counting, and result verification, a blockchain-based system automates much of this process. This decreases the need for manual intervention and reduces administrative overhead, especially in large-scale elections. Furthermore, the permissioned blockchain restricts access to authorised nodes, ensuring efficient and controlled operations while minimising the risk of errors or fraud.

The system's architecture ensures that the voting process is streamlined and automated, reducing the time required for vote tallying and result validation. By recording votes securely on a permissioned blockchain, the need for complex manual counting and verification is eliminated. Additionally, the integration of the Solana blockchain allows for transparent and near-instantaneous public reporting of results, reducing delays and increasing operational efficiency compared to traditional systems.

3.3.4 Economic Feasibility

The proposed blockchain-based voting system is highly cost-effective compared to traditional methods. Traditional voting systems require significant investments in physical infrastructure, such as polling stations, voting machines, and personnel for vote counting. In contrast, the proposed system primarily relies on mobile application and blockchain infrastructure, which significantly reduces the need for physical resources. By automating vote recording and tallying, operational costs, including staffing and logistics, are minimised. This makes the system an affordable solution, especially for large-scale elections.

The mobile app developed using **Flutter** ensures that voters can participate in elections using their smartphones, which are common in current times and are very much affordable. Since the system doesn't require voters to purchase any additional hardware or attend physical polling stations, it significantly lowers the cost of participation. This is especially beneficial for individuals in rural or remote areas who may face high transportation costs to reach polling stations. The app is free to download and easy to access, making it an affordable option for a wide range of users.

While there is an initial setup cost for integrating blockchain technology and developing the mobile app, the long-term operational savings are substantial. By eliminating the need for physical ballots, EVMs, and extensive manual labour for vote counting, the

system reduces recurrent costs significantly. Blockchain's ability to automate and secure vote transactions also minimises errors and reduces the need for post-election audits or recounts, leading to further cost savings.

CHAPTER 4: SYSTEM REQUIREMENTS SPECIFICATION

4.1 SOFTWARE REQUIREMENTS

4.1.1 Flutter

Flutter is an open-source UI toolkit developed by Google, enabling developers to build cross-platform applications from a single codebase. It is used to create the mobile application for this project, allowing voters to participate conveniently from their smartphones or other devices. Flutter's ability to compile natively ensures high performance, which is essential for a responsive and smooth user experience during critical election processes.

The choice of Flutter stems from its rich widget library and hot-reload feature, which allows for rapid development and testing. The platform supports the creation of visually appealing and intuitive interfaces, crucial for making the voting application accessible to a diverse user base, including individuals with minimal technical expertise. Its seamless compatibility with multiple platforms, such as Android, iOS, and web browsers, ensures that the system can reach a wide audience without requiring separate development efforts for each platform.

In this project, Flutter interacts with the blockchain infrastructure via REST APIs. These APIs enable secure communication between the mobile app and blockchain networks, allowing voters to authenticate themselves, cast votes, and track their status. The app's integration with Google ML Kit for liveness detection further enhances its usability by ensuring secure and real-time voter authentication. By combining Flutter's flexibility with blockchain capabilities, the system achieves both user accessibility and operational efficiency.

4.1.2 DeepFace

DeepFace is an open-source Python library for deep learning-based face recognition. It is utilised in the voting system to verify the identity of voters by matching their facial features with stored data. DeepFace supports multiple pre-trained models, such as VGG-Face,

OpenFace, and Facenet, which provide high accuracy in face matching tasks. Its integration ensures robust voter authentication, reducing the risk of impersonation or fraudulent voting.

The library is chosen for its flexibility and ease of integration with backend systems, making it ideal for this project. DeepFace operates on real-time input from the voter's device, processing their facial data to confirm identity before allowing access to the voting process. It supports multi-model comparisons, which enhances reliability, and its Python-based design ensures compatibility with FastAPI, the framework used for backend services in the system.

DeepFace also works alongside Google ML Kit for liveness detection, ensuring that the face being matched belongs to a live individual rather than an image or video. This combination guarantees that only legitimate users can participate in the election, significantly enhancing the system's security. Together, these technologies form the backbone of the biometric authentication module, which plays a critical role in preventing unauthorised voting while maintaining ease of use for voters.

4.1.3 Hyperledger Fabric

Hyperledger Fabric is a permissioned blockchain framework developed under the Linux Foundation's Hyperledger project. It is tailored for enterprise-grade applications, where privacy, scalability, and modularity are essential. In this voting system, Hyperledger Fabric serves as the backbone for storing and managing private vote data, ensuring that individual votes remain secure and immutable.

The framework is selected for its support for private transactions and controlled network access, which ensures that only authorised nodes can access sensitive voting information. Fabric's modular architecture allows for the customization of consensus mechanisms and membership services, which is essential for meeting the specific security and scalability requirements of this voting system. Smart contracts (Chaincode) written for Hyperledger Fabric automate vote validation and recording, reducing manual intervention and ensuring error-free processes.

Hyperledger Fabric also integrates seamlessly with Solana, the public blockchain used for publishing the final election results. This dual-layer architecture combines the privacy benefits of Hyperledger Fabric with the transparency of Solana, enabling a secure yet

auditable voting process. By utilising Fabric, the system achieves scalability for large-scale elections while maintaining strict confidentiality for individual votes.

4.1.4 Solana Blockchain

Solana is a high-performance, open-source public blockchain designed for scalability and speed. It is used in the voting system to store and publish the final election results, ensuring transparency and immutability. Solana's Proof of History (PoH) consensus mechanism allows it to process thousands of transactions per second while maintaining low transaction costs, making it ideal for large-scale elections.

This blockchain is chosen for its ability to handle high throughput efficiently, ensuring that election results are publicly available in near real-time without delays. Solana's low-cost transaction model makes it economically viable to record and verify the large number of votes associated with national elections. By ensuring that results are immutable and tamper-proof, it builds public trust in the electoral process, as anyone can verify the accuracy of the published data.

In the system architecture, Solana interacts with the Hyperledger Fabric blockchain. While individual votes are securely stored in Hyperledger Fabric for privacy, the aggregated results are transferred to Solana for public auditability. This integration ensures a clear separation of privacy and transparency, combining the strengths of both public and permissioned blockchain technologies to create a trustworthy voting framework.

4.1.5 FastAPI

FastAPI is a modern, high-performance web framework built with Python, optimised for building APIs. In the voting system, it acts as the backend layer, facilitating communication between the mobile application, the blockchain networks, and the biometric authentication module. FastAPI's asynchronous capabilities ensure it can handle multiple concurrent requests efficiently, which is essential for a system that serves a large voter base during high-demand periods.

FastAPI is chosen for its developer-friendly features, such as automatic data validation and interactive API documentation, which speed up development and reduce errors. It also integrates seamlessly with Python libraries like DeepFace for biometric authentication and blockchain SDKs like Fabric SDK for GO and Solana Web3.js, enabling smooth interaction with blockchain networks.

In the system, FastAPI enables functionalities like voter registration, vote submission, and real-time result retrieval. It bridges the gap between the user-facing mobile application and the backend blockchain infrastructure, ensuring that all interactions are secure, efficient, and user-friendly. FastAPI's lightweight architecture also ensures scalability, making it suitable for handling the high traffic typical of national elections.

4.1.6 Firebase

Firebase is a modern, cloud-hosted database that stores data in a flexible and efficient manner. Unlike traditional relational databases that organize data into tables with rows and columns, Firebase utilizes a JSON-like structure to manage data as collections and documents. This design allows each document to contain different types of data—from text and numbers to arrays and nested objects—which makes it easier to handle complex and changing information. Its adaptable structure is particularly useful for web applications where data requirements may evolve over time.

One of the main reasons developers opt for Firebase is its seamless scalability. The platform can effortlessly distribute data across multiple servers to manage larger workloads, making it an ideal choice for applications with high traffic or extensive datasets. Firebase also offers fast query performance and integrates well with modern programming languages and web frameworks. In our prototype, Firebase is used to store off-chain data, which includes details about the voter, information about the candidate, and other essential data required for the creation and management of an election.

4.2 HARDWARE REQUIREMENTS

- **CPU:** A multi-core processor (e.g., Intel Xeon or AMD Ryzen) with sufficient processing power to handle concurrent user requests. At least 8 cores and 16 threads to ensure seamless operation of blockchain nodes, API servers, and biometric verification modules.
- **RAM:** A minimum of 16 GB of RAM to ensure smooth performance, but the actual requirement may increase based on usage patterns. Additional memory might be required for large-scale elections involving millions of voters
- **Storage:** SSD (Solid-State Drive) storage for fast data access.
- **Network:** Gigabit Ethernet or higher for reliable data transfer between servers.

4.3 LANGUAGE DESCRIPTION

4.3.1 HTML

HTML is the standard markup language for creating web pages and web applications. It defines the structure and layout of web content using elements and tags. HTML is not a programming language but rather a markup language for structuring web documents. HTML documents consist of a collection of elements, each represented by tags that denote the beginning and end of the element. These elements can include headings, paragraphs, images, links, forms, lists, and more. Tags are crucial in defining the relationships between different parts of the content and specifying how they should be rendered on the user's browser. Web developers use HTML as a foundation for creating responsive and engaging websites and web applications. The language's simplicity, along with its ability to integrate seamlessly with other technologies, makes it an essential tool for building a wide range of digital experiences. As the internet continues to evolve, HTML remains a cornerstone for shaping the structure and presentation of online content.

4.3.2 CSS (Cascading Style Sheets)

CSS is used to style web pages and manage the presentation of HTML elements, allowing developers to control aspects like colours, fonts, spacing, and layout. It plays a vital role in creating aesthetically pleasing and responsive web designs. A core feature of CSS is selectors, which define which HTML elements are affected by a specific style rule. CSS

offers various types of selectors, such as element, class, and ID selectors, giving developers precise control over how content is styled. Media queries in CSS support responsive design by applying different styles based on factors like screen size, resolution, or device orientation. This helps ensure a seamless and optimised experience for users, regardless of the device or screen size they are using.

4.3.3 JavaScript

JavaScript is a versatile programming language primarily used for adding interactivity and behaviour to web pages. It can be executed in web browsers, making it ideal for creating dynamic web applications. JavaScript is an essential component of modern web development. Being an interpreted language, JavaScript executes code on-the-fly, without the need for a compilation step. This characteristic allows for quick development cycles and immediate feedback during the coding process. Additionally, JavaScript is dynamically typed, meaning variables can hold values of any type without explicit type declarations. While this flexibility can lead to concise code, developers need to be mindful of potential type-related issues.

JavaScript's event-driven and asynchronous nature is another notable feature. It responds to various events triggered by user interactions, such as clicks or keyboard input. The language's asynchronous capabilities, facilitated by features like callbacks and promises, enable the execution of non-blocking code. This is particularly valuable for handling tasks like fetching data from servers without freezing the user interface.

Beyond the browser, JavaScript has expanded its reach through environments like Web3.js, enabling developers to interact with blockchain networks directly from their applications. This capability allows developers to build decentralised applications (dApps) by handling blockchain interactions such as smart contract execution, transaction management, and data retrieval. Web3.js's role in server-side and client-side programming provides a seamless bridge between traditional web technologies and decentralised ecosystems.

This versatility has made JavaScript a prominent language in full-stack and blockchain development, where it can be employed to manage both user interfaces and blockchain integrations, providing a unified language stack for modern decentralised web applications. Overall, JavaScript's adaptability, coupled with its vibrant ecosystem of libraries

and frameworks like Web3.js, has solidified its position as a foundational technology in the worlds of web and blockchain development.

4.3.4 Python

Python is a versatile, high-level programming language widely appreciated for its simplicity, readability, and flexibility. It is extensively used across various fields, such as web development, data analysis, artificial intelligence, machine learning, automation, and scientific research. Designed with an emphasis on clean and readable code, Python helps developers create maintainable and efficient programs with ease. Its syntax relies on indentation rather than braces or semicolons, promoting clarity and reducing common coding errors.

Supporting multiple programming paradigms, including object-oriented, functional, imperative, and procedural programming, Python caters to diverse development needs. Its comprehensive standard library simplifies tasks like file handling, system operations, and web development by providing pre-built modules and packages. Additionally, Python's ecosystem boasts a vast array of third-party libraries and frameworks that extend its functionality. For instance, in projects involving blockchain and facial recognition, frameworks like FastAPI enable secure backend operations, while libraries such as DeepFace enhance biometric authentication.

As a dynamically typed language, Python allows developers to write code without declaring variable types explicitly, which speeds up development but may demand more rigorous testing. Its interpreted nature, which executes code line by line, supports iterative development but can impact performance in resource-intensive scenarios. Despite these limitations, Python's extensive library support, efficient memory management, and ease of use make it ideal for both rapid prototyping and large-scale applications. Its user-friendly design, active community, and widespread adoption have made it a popular choice for projects ranging from simple scripts to advanced AI systems.

4.3.4 Rust

Rust is a modern programming language known for its safety and speed. It helps developers write code that is less prone to errors by checking for potential mistakes while the program is running. The language gives more memory safety, meaning it carefully manages how data is used, which helps prevent common bugs like crashes or security issues. This focus on safe coding practices makes Rust a popular choice for many different projects.

When it comes to writing smart contracts, Rust offers a strong foundation because of its performance and reliability. Smart contracts are small programs that run on a blockchain, and they need to be both efficient and secure. Developers use Rust to write these contracts because it can handle the complex tasks required by blockchain technology. With Rust, smart contracts run quickly and without unexpected problems, which is crucial for maintaining trust and stability on the network.

Solana, a fast and scalable blockchain platform, uses Rust as one of its main languages for smart contract development. On Solana, smart contracts are known as programs, and writing them in Rust lets developers take full advantage of the platform's speed and efficiency. The combination of Rust's robust features and Solana's advanced technology allows for the creation of reliable and high-performance decentralized applications, making the blockchain experience smoother for everyone involved.

4.3.4 Go

Go, also known as Golang, is a statically typed, compiled programming language designed by Google. It is used extensively in Hyperledger Fabric, a key component of the Blockchain Based Hybrid Voting System. Go's simple syntax, strong concurrency support, and performance make it ideal for developing chaincode (smart contracts) for Hyperledger Fabric, which is used to manage private vote transactions and ensure secure, transparent election processes.

One of Go's primary benefits is its efficiency in handling multiple tasks concurrently, which is crucial for large-scale systems that require real-time processing, such as voting systems. Its concurrency model, built around goroutines and channels, allows Hyperledger Fabric to scale efficiently and handle many simultaneous transactions without performance

degradation. This makes Go essential in ensuring that the voting system can accommodate millions of voters without experiencing slowdowns or bottlenecks during peak usage times.

Go is also highly compatible with Hyperledger Fabric's architecture, where it is used to write chaincode that governs the private vote recording process. The ability to quickly process transactions and manage secure communications within the network ensures that the integrity of each vote is maintained. Go's focus on simplicity and performance, combined with its ability to handle complex systems efficiently, makes it a valuable language for ensuring the system's scalability and reliability in this project.

CHAPTER 5: METHODOLOGY

5.1 BLOCKCHAIN ARCHITECTURE

In this project, the architecture of the voting system is based on a hybrid blockchain model, combining Solana (a public blockchain) and Hyperledger Fabric (a permissioned blockchain). Blockchain technology is central to the system's design. Solana and Hyperledger Fabric are integrated to handle different aspects of the voting process. The purpose of this hybrid model is to provide a high level of transparency while maintaining the privacy and security of individual votes. In the system, Solana is used to store the final vote tally, ensuring that the results are publicly available and immutable. Meanwhile, Hyperledger Fabric stores the private transaction data (votes), accessible only to authorised participants.

The design is scalable, ensuring that the system can handle millions of voters without compromising performance. This hybrid structure reduces the load on each blockchain while enabling secure and real-time result tracking, providing a transparent and reliable voting process. Smart contracts, written in Solidity for Solana, are used to automate vote tallying and ensure that all transactions follow predefined rules. These contracts ensure that each vote is securely recorded and counted, minimising the potential for human error or manipulation. In addition, Hyperledger Fabric's chaincode is used to manage the private data, including vote recording and validation, ensuring that only authorised nodes can access sensitive data.

5.2 BIOMETRIC AUTHENTICATION AND SECURITY

Biometric authentication plays a crucial role in securing the voting process and ensuring that only eligible voters can participate. The system uses DeepFace, a Python library, for facial recognition to verify the voter's identity. Additionally, Google ML Kit is used for liveness detection, confirming that the voter is a live person and preventing the use of photos or videos for impersonation.

This combination of biometric features strengthens the system's security by ensuring accurate and tamper-proof identification, minimising the risk of fraud such as double voting or impersonation. The integration of liveness detection further ensures that the voting process is secure by verifying the legitimacy of the voter in real time, making it harder for fraudulent activities to occur.

Importantly, the system prioritises data security and user privacy. It does not store the facial images or the verification data once the process is completed. After successfully verifying the voter's identity, all biometric data is instantly deleted, ensuring that no sensitive information is retained. This approach minimises the risk of data breaches and complies with privacy standards such as GDPR, ensuring that voters' personal information is protected throughout the process.

5.3 FLUTTER MOBILE APP

The mobile app is developed using Flutter, a cross-platform framework that allows the system to be accessible to both Android and iOS users. Flutter enables the creation of a high-performance, responsive, and user-friendly application. Voters can use the app to authenticate themselves, cast their votes, and track the status of their votes in real time. Additionally, the mobile app ensures a smooth experience by providing real-time notifications to voters about the status of their votes. Users receive immediate feedback upon successful authentication and vote submission, reducing uncertainty and increasing transparency in the voting process

The app is designed with simplicity and security in mind, ensuring that users can easily navigate the process of voting. Its seamless integration with the backend API (developed using FastAPI) ensures smooth communication between the user interface and the blockchain networks, ensuring that vote data is securely recorded and results are processed efficiently. The app is optimised for performance, using efficient memory and data management practices, ensuring that it runs smoothly even on lower-end devices. This ensures accessibility for a wide range of users, particularly in areas with limited access to high-end smartphones. By combining Flutter's rich UI capabilities, FastAPI's real-time data handling, and secure blockchain interactions, the mobile app provides a comprehensive and secure solution for modern digital voting

5.4 ELECTION COMMISSION

The election commission module is a core component of our blockchain voting project that plays a central role in managing the election process from start to finish. This module is designed to allow authorized election officials to create new elections, set the election parameters, and assign eligible voters to each election. Once the election details are

set, the commission can also start the election at the designated time and ensure it runs smoothly throughout the voting period. At the end of the election, the module allows the commission to formally close the election and publish all necessary details, such as vote counts and results, to maintain a clear and transparent process. The module's integration with the blockchain guarantees that every action is recorded immutably, enhancing security and trust in the election process.

In addition to managing the overall election workflow, the election commission module includes important verification functions to ensure that every voter is accurately authenticated. In situations where automatic biometric verification does not succeed, the commission is tasked with manually verifying the voter's identity, ensuring that only eligible voters participate. This manual check adds an extra layer of security and helps prevent fraud. The module also provides tools for continuous monitoring and updating of voter assignments, as well as for handling any disputes that may arise during the election process. By incorporating these measures, the election commission module helps create a robust, secure, and transparent system that upholds the integrity of the voting process and supports the democratic values that our project stands for.

CHAPTER 6: SYSTEM DESIGN

6.1 ARCHITECTURE DIAGRAM

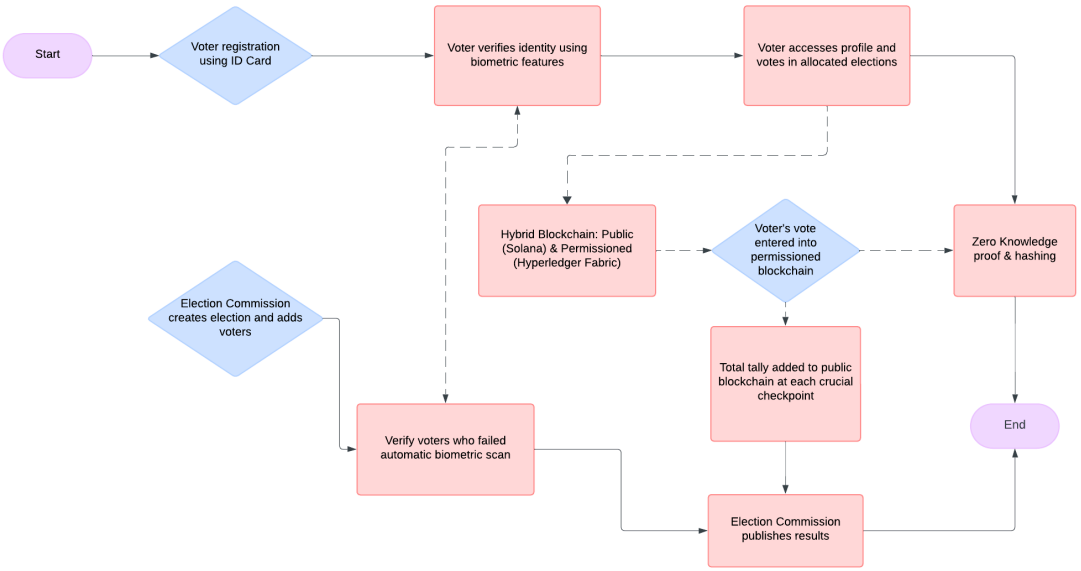

Figure 6.1 Architecture Diagram

6.2 USE CASE DIAGRAM

6.2.1 Voter

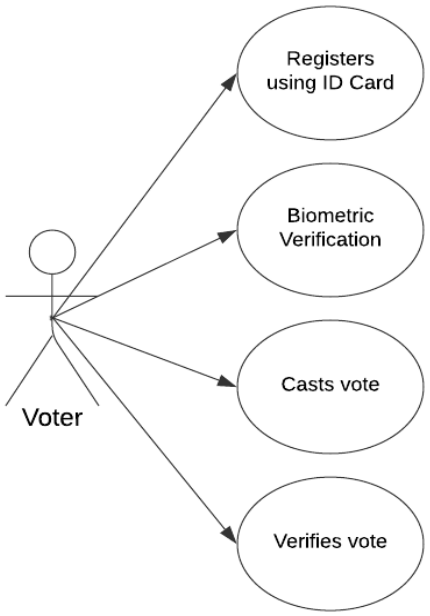

Figure 6.2 Use Case Diagram

6.2.1 Election Commission

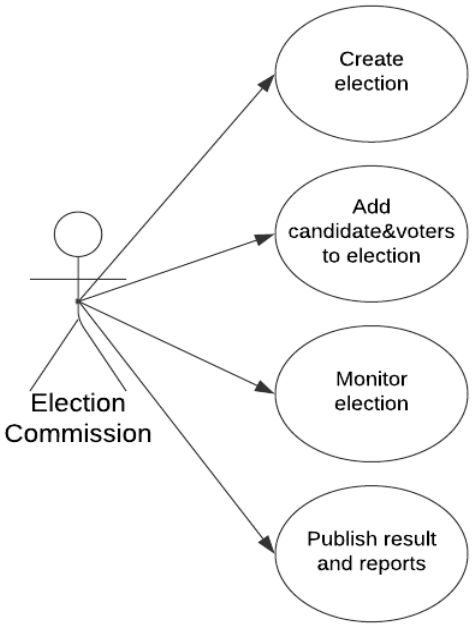

Figure 6.3 Use Case Diagram

CHAPTER 7: SYSTEM IMPLEMENTATION

7.1 HYPERLEDGER FABRIC CHAINCODE

```
package main
import (
	"crypto/sha256"
	"encoding/hex"
	"encoding/json"
	"fmt"
	"time"
	"github.com/hyperledger/fabric-contract-api-go/contractapi"
)
type SmartContract struct {
	contractapi.Contract
}
type Election struct {
	ID          string          `json:"id"`
	Name        string          `json:"name"`
	Candidates  []string        `json:"candidates"`
	VoteCounts  map[string]int  `json:"voteCounts"`
	VoterStatus map[string]string `json:"voterStatus"`
}
func (s *SmartContract) CreateElection(ctx contractapi.TransactionContextInterface,
electionID string, name string) error {
	exists, err := s.ElectionExists(ctx, electionID)
```

```
        if err != nil {
                return err
        }
        if exists {
                return fmt.Errorf("election %s already exists", electionID)
        }
        election := Election{
                ID:          electionID,
                Name:        name,
                Candidates:  []string{},
                VoteCounts:  make(map[string]int),
                VoterStatus: make(map[string]string),
        }
        electionJSON, err := json.Marshal(election)
        if err != nil {
                return err
        }
        return ctx.GetStub().PutState(electionID, electionJSON)
}
func (s *SmartContract) ElectionExists(ctx contractapi.TransactionContextInterface,
electionID string) (bool, error) {
        data, err := ctx.GetStub().GetState(electionID)
        if err != nil {
                return false, err
        }
```

```
	return data != nil, nil
}

func (s *SmartContract) AddCandidate(ctx contractapi.TransactionContextInterface,
electionID string, candidateID string) error {
	election, err := s.GetElection(ctx, electionID)
	if err != nil {
		return err
	}
	for _, c := range election.Candidates {
		if c == candidateID {
			return fmt.Errorf("candidate %s already exists in election %s",
candidateID, electionID)
		}
	}
	election.Candidates = append(election.Candidates, candidateID)
	election.VoteCounts[candidateID] = 0
	electionJSON, err := json.Marshal(election)
	if err != nil {
		return err
	}
	return ctx.GetStub().PutState(electionID, electionJSON)
}

func (s *SmartContract) AssignVoter(ctx contractapi.TransactionContextInterface, electionID
string, voterID string) error {
	election, err := s.GetElection(ctx, electionID)
	if err != nil {
```

```
		return err
	}
	if _, exists := election.VoterStatus[voterID]; exists {
		return fmt.Errorf("voter %s is already registered (or has voted) in election
%s", voterID, electionID)
	}
	election.VoterStatus[voterID] = ""
	electionJSON, err := json.Marshal(election)
	if err != nil {
		return err
	}
	return ctx.GetStub().PutState(electionID, electionJSON)
}
func (s *SmartContract) CastVote(ctx contractapi.TransactionContextInterface, electionID
string, voterID string, candidateID string) (string, error) {
	election, err := s.GetElection(ctx, electionID)
	if err != nil {
		return "", err
	}
	receipt, exists := election.VoterStatus[voterID]
	if !exists {
		return "", fmt.Errorf("voter %s is not registered for election %s", voterID,
electionID)
	}
	if receipt != "" {
		return "", fmt.Errorf("voter %s has already voted", voterID)
```

```
	}
	validCandidate := false
	for _, c := range election.Candidates {
		if c == candidateID {
			validCandidate = true
			break
		}
	}
	if !validCandidate {
		return "", fmt.Errorf("candidate %s is not valid for election %s", candidateID,
electionID)
	}
	election.VoteCounts[candidateID]++
	txTime, err := ctx.GetStub().GetTxTimestamp()
	if err != nil {
		return "", err
	}
	timestamp := time.Unix(txTime.Seconds, int64(txTime.Nanos)).Unix()
	data := fmt.Sprintf("%s:%s:%s:%d", electionID, voterID, candidateID, timestamp)
	hash := sha256.Sum256([]byte(data))
	voteReceipt := hex.EncodeToString(hash[:])

	election.VoterStatus[voterID] = voteReceipt

	electionJSON, err := json.Marshal(election)
```

```
	if err != nil {
		return "", err
	}
	if err = ctx.GetStub().PutState(electionID, electionJSON); err != nil {
		return "", err
	}
	return voteReceipt, nil
}

func (s *SmartContract) VerifyVote(ctx contractapi.TransactionContextInterface, electionID
string, voterID string) (string, error) {
	election, err := s.GetElection(ctx, electionID)
	if err != nil {
		return "", err
	}
	receipt, exists := election.VoterStatus[voterID]
	if !exists || receipt == "" {
		return "", fmt.Errorf("no vote recorded for voter %s in election %s", voterID,
electionID)
	}
	return receipt, nil
}

func (s *SmartContract) GetElection(ctx contractapi.TransactionContextInterface, electionID
string) (*Election, error) {
	electionJSON, err := ctx.GetStub().GetState(electionID)
	if err != nil {
		return nil, err
```

```
        }
        if electionJSON == nil {
                return nil, fmt.Errorf("election %s does not exist", electionID)
        }
        var election Election
        if err = json.Unmarshal(electionJSON, &election); err != nil {
                return nil, err
        }
        return &election, nil
}
func main() {
        chaincode, err := contractapi.NewChaincode(new(SmartContract))
        if err != nil {
                fmt.Printf("Error creating chaincode: %s", err.Error())
                return
        }
        if err = chaincode.Start(); err != nil {
                fmt.Printf("Error starting chaincode: %s", err.Error())
        }
}
```

7.2 SOLANA SMART CONTRACT

```
use solana_program::{
   account_info::{next_account_info, AccountInfo},
   entrypoint,
```

```
    entrypoint::ProgramResult,
    msg,
    program_error::ProgramError,
    pubkey::Pubkey,
};
pub enum VotingInstruction {
    UpdateTally { tally: u64 },
}
impl VotingInstruction {
    pub fn unpack(input: &[u8]) -> Result<Self, ProgramError> {
        let (&tag, rest) = input
            .split_first()
            .ok_or(ProgramError::InvalidInstructionData)?;
        match tag {
            0 => {
                if rest.len() != 8 {
                    return Err(ProgramError::InvalidInstructionData);
                }
                let tally = rest
                    .get(..8)
                    .and_then(|slice| slice.try_into().ok())
                    .map(u64::from_le_bytes)
                    .ok_or(ProgramError::InvalidInstructionData)?;
                Ok(Self::UpdateTally { tally })
            }
```

```
            _ => Err(ProgramError::InvalidInstructionData),
        }
    }
}
entrypoint!(process_instruction);
pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    let instruction = VotingInstruction::unpack(instruction_data)?;
    match instruction {
        VotingInstruction::UpdateTally { tally } => {
            process_update_tally(program_id, accounts, tally)
        }
    }
}
pub fn process_update_tally(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    tally: u64,
) -> ProgramResult {
    let account_info_iter = &mut accounts.iter();
    let tally_account = next_account_info(account_info_iter)?;
    if tally_account.owner != program_id {
```

```
        msg!("Tally account is not owned by the program");
        return Err(ProgramError::IncorrectProgramId);
    }
    let mut data = tally_account.try_borrow_mut_data()?;
    if data.len() < 8 {
        return Err(ProgramError::AccountDataTooSmall);
    }
    data[..8].copy_from_slice(&tally.to_le_bytes());
    msg!("Tally updated to {}", tally);
    Ok(())
}
```

7.3 BACKEND FAST API CODE

```
from fastapi import APIRouter, HTTPException
from typing import List, Optional
from pydantic import BaseModel
router = APIRouter()
class Election(BaseModel):
    id: str
    name: str
    description: Optional[str] = None
    status: str = "pending"  # ("pending", "started", "stopped")
    candidates: List[dict] = []
    voters: List[dict] = []
class Candidate(BaseModel):
```

```
    id: str
    name: str
    party: Optional[str] = None
class ElectionReport(BaseModel):
    election_id: str
    results: dict
    analytics: dict
elections_db = {}
@router.get("/elections", response_model=List[Election])
def list_elections():
    return list(elections_db.values())
def get_election(id: str):
    election = elections_db.get(id)
    if not election:
        raise HTTPException(status_code=404, detail="Election not found")
    return election
def create_election(election: Election):
    if election.id in elections_db:
        raise HTTPException(status_code=400, detail="Election already exists")
    elections_db[election.id] = election
    return election
def update_election(id: str, updated_data: Election):
    if id not in elections_db:
        raise HTTPException(status_code=404, detail="Election not found")
    elections_db[id] = updated_data
```

```
    return updated_data
class User(BaseModel):
    verification_id: str
    dob: str
    name: str
    phone: str
    email: str
    photo: str
    verified: bool
@router.get("/users/unverified", response_model=List[User])
def get_unverified_users():
    users = []
    try:
        users = get_all_users()
    except:
        print("Error occured")

    return [user for user in users if not user["verified"]]
@router.post("/users/{verification_id}/verify_request")
def verify_request(verification_id: str, uploaded_file: UploadFile):
    try:
        upload_result = cloudinary.uploader.upload(uploaded_file)
        cloudinary_url = upload_result.get("secure_url")
        if not cloudinary_url:
            print("[Error] Cloudinary did not return a secure URL for the file")
```

```
            return
        update_field_by_verification_id(verification_id, "selfie", cloudinary_url)
    except:
        print("[Error] Cloudinary upload failed for file")
@router.post("/users/{verification_id}/verify")
def verify_user(verification_id: str):
    if not verification_id.startswith("S"):
        raise HTTPException(status_code=400, detail="Not valid verification")
    try:
        update_field_by_verification_id(verification_id, "verified", True)
        return {"message": f"User {verification_id} verified"}
    except:
        raise HTTPException(status_code=404, detail="User not found")
@router.get("/users/{verification_id}", response_model=User)
def get_user(verification_id: str):
    if not verification_id.startswith("S"):
        raise HTTPException(status_code=400, detail="Not valid verification")
    users = get_all_users()
    for user in users:
        if user["verification_id"] == verification_id:
            return user
class RejectRequest(BaseModel):
    reason: str
@router.post("/users/{verification_id}/reject")
def reject_user(verification_id: str, request: RejectRequest):
```

```
    try:
        update_field_by_verification_id(verification_id, "selfie", "")
        return {"message": f"User {verification_id} rejected. Reason {request.reason}"}
    except:
        raise HTTPException(status_code=404, detail="User not found")
executor = ThreadPoolExecutor()
@router.post("/send_otp")
async def send_otp(request: SendOTPRequest):
    mobile = get_user_mobile(request.verification_id)
    if not mobile:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND, detail="User not found"
        )
    if request.verification_id in otp_util.otp_store:
        try:
            otp_entry = otp_util.update_otp_entry(request.verification_id)
        except Exception as e:
            raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail=str(e))
    else:
        otp_entry = otp_util.create_otp_entry(request.verification_id)
    return {
        "message": "OTP sent successfully",
        "mobile": mobile,
        "otp": otp_entry["otp"],
    }
```

```
@router.post("/face_verification")
async def face_verification(
    verification_id: str = Form(...),
    image: UploadFile = File(...),
):
    try:
        loop = asyncio.get_running_loop()
        result = await loop.run_in_executor(
            executor, verify_face, verification_id, image
        )
        return result
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=str(e),
        )
from solana.rpc.async_api import AsyncClient
from solana.transaction import Transaction, TransactionInstruction
from solana.publickey import PublicKey
from solana.keypair import Keypair
SOLANA_RPC_URL = "https://api.devnet.solana.com"
PROGRAM_ID =
PublicKey("4Nd1mJxZmLQ5zUzgLVX7qhZQUSex4eLWx5gd34ZWkPoH")
async def update_tally(tally: int, tally_account_pubkey: str, payer: Keypair):
    """
```

```
    Update the tally on Solana.

    Args:
        tally: New tally value (u64)
        tally_account_pubkey: Public key of the account to update (as string)
        payer: A Keypair instance used to pay for the transaction fees.

    Returns:
        The RPC response from Solana.
    """
    async with AsyncClient(SOLANA_RPC_URL) as client:
        data = bytes([0]) + struct.pack("<Q", tally)
        tally_account = PublicKey(tally_account_pubkey)
        instruction = TransactionInstruction(
            keys=[
                {"pubkey": tally_account, "is_signer": False, "is_writable": True},
            ],
            program_id=PROGRAM_ID,
            data=data,
        )
        txn = Transaction()
        txn.add(instruction)
        response = await client.send_transaction(txn, payer)
        return response
@router.post("/vote")
```

```
def cast_vote():
    """
    Receives JSON with 'election_id', 'voter_id', and 'candidate_id'
    and calls the Microfab RPC endpoint to invoke the 'CastVote' function.
    """
    data = request.get_json()
    election_id = data.get("election_id")
    voter_id = data.get("voter_id")
    candidate_id = data.get("candidate_id")

    if not all([election_id, voter_id, candidate_id]):
        return (
            jsonify(
                {
                    "error": "Missing required parameters: election_id, voter_id, candidate_id"
                }
            ),
            400,
        )
    payload = {
        "jsonrpc": "2.0",
        "method": "invokeChaincode",
        "params": {
            "channel": "mychannel",
            "chaincode": "voting_chaincode",
```

```
            "fcn": "CastVote",
            "args": [election_id, voter_id, candidate_id],
        },
        "id": 1,
    }
    rpc_url = "http://localhost:7051/"
    try:
        rpc_response = requests.post(rpc_url, json=payload)
        rpc_response.raise_for_status()
        result = rpc_response.json()
        if "error" in result:
            return jsonify({"error": result["error"]}), 500
        vote_receipt = result.get("result")
        return jsonify({"vote_receipt": vote_receipt}), 200

    except Exception as e:
        return jsonify({"error": str(e)}), 500
@router.post("/solana/update_tally")
async def update_solana_tally(update: TallyUpdate):
    payer = Keypair()
    try:
        response = await update_tally(update.tally, update.tally_account, payer)
        return {"result": response}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

7.4 ELECTION COMMISSION FRONTEND

7.4.1 Dashboard

```
import React, { useEffect, useState } from 'react';
import { Users, Vote, BarChart3 } from 'lucide-react';
import { api } from '../api';
import type { Election, User } from '../types';

export default function Dashboard() {
  const [stats, setStats] = useState({
    totalUsers: 0,
    pendingVerifications: 0,
    activeElections: 0,
    completedElections: 0
  });

  useEffect(() => {
    Promise.all([
      api.getUnverifiedUsers(),
      api.getElections()
    ]).then(([users, elections]) => {
      setStats({
        totalUsers: users.length,
        pendingVerifications: users.filter((u: User) => !u.verified).length,
        activeElections: elections.filter((e: Election) => e.status === 'active').length,
```

```
      completedElections: elections.filter((e: Election) => e.status === 'completed').length
    });
  });
}, []);

return (
  <div>
    <h1 className="text-2xl font-semibold text-gray-900">Dashboard</h1>

    <div className="mt-6 grid grid-cols-1 gap-5 sm:grid-cols-2 lg:grid-cols-3">
      <div className="bg-white overflow-hidden shadow rounded-lg">
        <div className="p-5">
          <div className="flex items-center">
            <div className="flex-shrink-0">
              <Users className="h-6 w-6 text-gray-400" />
            </div>
            <div className="ml-5 w-0 flex-1">
              <dl>
                <dt className="text-sm font-medium text-gray-500 truncate">
                  Pending Verifications
                </dt>
                <dd className="text-lg font-semibold text-gray-900">
                  {stats.pendingVerifications}
                </dd>
              </dl>
```

```
      </div>
    </div>
  </div>
</div>

<div className="bg-white overflow-hidden shadow rounded-lg">
  <div className="p-5">
    <div className="flex items-center">
      <div className="flex-shrink-0">
        <Vote className="h-6 w-6 text-gray-400" />
      </div>
      <div className="ml-5 w-0 flex-1">
        <dl>
          <dt className="text-sm font-medium text-gray-500 truncate">
            Active Elections
          </dt>
          <dd className="text-lg font-semibold text-gray-900">
            {stats.activeElections}
          </dd>
        </dl>
      </div>
    </div>
  </div>
</div>
```

```
      <div className="bg-white overflow-hidden shadow rounded-lg">
        <div className="p-5">
          <div className="flex items-center">
            <div className="flex-shrink-0">
              <BarChart3 className="h-6 w-6 text-gray-400" />
            </div>
            <div className="ml-5 w-0 flex-1">
              <dl>
                <dt className="text-sm font-medium text-gray-500 truncate">
                  Completed Elections
                </dt>
                <dd className="text-lg font-semibold text-gray-900">
                  {stats.completedElections}
                </dd>
              </dl>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```

7.4.1 User Verification

```
import React, { useEffect, useState } from 'react';
import { Check, X } from 'lucide-react';
import toast from 'react-hot-toast';
import { api } from '../api';
import type { User } from '../types';

export default function UserVerification() {
  const [users, setUsers] = useState<User[]>([]);

  useEffect(() => {
    loadUsers();
  }, []);

  const loadUsers = () => {
    api.getUnverifiedUsers()
      .then(setUsers)
      .catch(() => toast.error('Failed to load users'));
  };

  const handleVerify = async (userId: string) => {
    try {
      await api.verifyUser(userId);
      toast.success('User verified successfully');
```

```
    loadUsers();
  } catch (error) {
    toast.error('Failed to verify user');
  }
};

return (
  <div>
    <h1 className="text-2xl font-semibold text-gray-900">User Verification</h1>

    <div className="mt-6 bg-white shadow overflow-hidden sm:rounded-md">
      <ul role="list" className="divide-y divide-gray-200">
        {users.map((user) => (
          <li key={user.verification_id}>
            <div className="px-4 py-4 sm:px-6">
              <div className="flex items-center justify-between">
                <div className="flex items-center">
                  <img
                    className="h-12 w-12 rounded-full"
                    src={user.photo}
                    alt={user.name}
                  />
                  <div className="ml-4">
                    <p className="text-sm font-medium text-gray-900">{user.name}</p>
                    <p className="text-sm text-gray-500">{user.email}</p>
```

```
                </div>
              </div>
              <div className="flex space-x-2">
                <button
                  onClick={() => handleVerify(user.verification_id)}
                  className="inline-flex items-center px-3 py-2 border border-transparent
text-sm leading-4 font-medium rounded-md text-white bg-green-600 hover:bg-green-700
focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-green-500"
                >
                  <Check className="h-4 w-4 mr-2" />
                  Verify
                </button>
                <button
                  className="inline-flex items-center px-3 py-2 border border-transparent
text-sm leading-4 font-medium rounded-md text-white bg-red-600 hover:bg-red-700
focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-red-500"
                >
                  <X className="h-4 w-4 mr-2" />
                  Reject
                </button>
              </div>
            </div>
          </div>
        </li>
      ))}
    </ul>
  </div>
```

```
    </div>
  );
}
```

7.4.1 Manage Election

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { Calendar } from 'lucide-react';
import toast from 'react-hot-toast';
import type { Election } from '../types';
import { mockElections } from '../mockData';

export default function CreateElection() {
  const navigate = useNavigate();
  const [formData, setFormData] = useState({
    title: '',
    description: '',
    startDate: '',
    endDate: ''
  });
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    const newElection: Election = {
      id: String(mockElections.length + 1),
```

```
    title: formData.title,
    description: formData.description,
    startDate: formData.startDate,
    endDate: formData.endDate,
    status: 'draft',
    candidates: [],
    voters: []
  };
  toast.success('Election created successfully');
  navigate(`/elections/${newElection.id}`);
};

const handleChange = (e: React.ChangeEvent<HTMLInputElement | HTMLTextAreaElement>) => {
  const { name, value } = e.target;
  setFormData(prev => ({ ...prev, [name]: value }));
};
return (
  <div>
    <div className="md:grid md:grid-cols-3 md:gap-6">
      <div className="md:col-span-1">
        <div className="px-4 sm:px-0">
          <h3 className="text-lg font-medium leading-6 text-gray-900">Create New Election</h3>
          <p className="mt-1 text-sm text-gray-600">
```

```
        Provide the basic information for the new election. You can add candidates and
voters after creation.

          </p>

         </div>

        </div>

        <div className="mt-5 md:mt-0 md:col-span-2">

          <form onSubmit={handleSubmit}>

            <div className="shadow sm:rounded-md sm:overflow-hidden">

              <div className="px-4 py-5 bg-white space-y-6 sm:p-6">

                <div>

                  <label htmlFor="title" className="block text-sm font-medium text-gray-700">

                    Title

                  </label>

                  <div className="mt-1">

                    <input

                      type="text"

                      name="title"

                      id="title"

                      value={formData.title}

                      onChange={handleChange}

                      className="shadow-sm focus:ring-blue-500 focus:border-blue-500 block
w-full sm:text-sm border-gray-300 rounded-md"

                      required

                    />

                  </div>

                </div>
```

```
          <div>
            <label htmlFor="description" className="block text-sm font-medium
text-gray-700">
              Description
            </label>
            <div className="mt-1">
              <textarea
                id="description"
                name="description"
                rows={3}
                value={formData.description}
                onChange={handleChange}
                className="shadow-sm focus:ring-blue-500 focus:border-blue-500 block
w-full sm:text-sm border-gray-300 rounded-md"
              />
            </div>
          </div>

          <div className="grid grid-cols-2 gap-6">
            <div>
              <label htmlFor="startDate" className="block text-sm font-medium
text-gray-700">
                Start Date
              </label>
              <div className="mt-1">
```

```
                <input
                  type="datetime-local"
                  name="startDate"
                  id="startDate"
                  value={formData.startDate}
                  onChange={handleChange}
                  className="shadow-sm focus:ring-blue-500 focus:border-blue-500 block
w-full sm:text-sm border-gray-300 rounded-md"
                  required
                />
              </div>
            </div>

            <div>
              <label htmlFor="endDate" className="block text-sm font-medium
text-gray-700">
                End Date
              </label>
              <div className="mt-1">
                <input
                  type="datetime-local"
                  name="endDate"
                  id="endDate"
                  value={formData.endDate}
                  onChange={handleChange}
```

```
                    className="shadow-sm focus:ring-blue-500 focus:border-blue-500 block
w-full sm:text-sm border-gray-300 rounded-md"
                    required
                   />
                  </div>
                 </div>
                </div>
               </div>
               <div className="px-4 py-3 bg-gray-50 text-right sm:px-6">
                <button
                 type="submit"
                 className="inline-flex justify-center py-2 px-4 border border-transparent
shadow-sm text-sm font-medium rounded-md text-white bg-blue-600 hover:bg-blue-700
focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500"
                >
                 Create Election
                </button>
               </div>
              </div>
             </form>
            </div>
           </div>
          );
}
```

7.5 FLUTTER APP CODE

7.5.1 Home Screen

```
import 'package:flutter/material.dart';
import 'package:votex/screens/voting_screen.dart';
class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Profile."),
        titleTextStyle: TextStyle(
          color: Colors.green,
          fontSize: 30,
          fontWeight: FontWeight.w900,
        ),
      ),
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Row(
                crossAxisAlignment: CrossAxisAlignment.start,
```

```
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              CircleAvatar(
                radius: 40,
                backgroundImage: NetworkImage(
"https://res.cloudinary.com/demazndgq/image/upload/v1739307747/s7wi1wox1hvls6ipe0mz.jpg"),
              ),
              Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Hari Sankar. R S',
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  SizedBox(height: 8),
                  Text(
                    'DOB:      14/06/2005',
                    style: TextStyle(fontSize: 16),
                  ),
                  SizedBox(height: 3),
                  Text(
```

```
          'Phone:   +91*****8599',
          style: TextStyle(fontSize: 16),
        ),
      ],
    ),
  ],
),
SizedBox(height: 22),
Text(
  'Eligible to vote for the following elections:',
  style: TextStyle(
    fontSize: 22,
    color: Colors.green,
    fontWeight: FontWeight.w900,
  ),
),
SizedBox(height: 30),
Container(
  decoration: BoxDecoration(color: Colors.green),
  child: ListTile(
    title: Text(
      'Class elections 2025',
      style: TextStyle(
        color: Color(0xFF101010),
        fontSize: 16,
```

```
        fontWeight: FontWeight.bold,
      ),
    ),
    trailing: ElevatedButton(
      style: ElevatedButton.styleFrom(
        shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.zero),
      ),
      onPressed: () {
        Navigator.of(context).push(
          MaterialPageRoute(
            builder: (context) => VotingScreen(),
          ),
        );
      },
      child: Text(
        "Vote",
        style: TextStyle(
          color: Colors.white,
          fontSize: 20,
          fontWeight: FontWeight.w900,
        ),
      ),
    ),
  ),
```

```
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

7.5.2 Voting Screen

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:votex/screens/end_screen.dart';
class VotingScreen extends StatefulWidget {
  const VotingScreen({super.key});
  @override
  State<VotingScreen> createState() => _VotingScreenState();
}
class _VotingScreenState extends State<VotingScreen> {
  int? _selectedCandidate;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Padding(
          padding: EdgeInsets.all(50),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Text(
                "VOTE NOW",
                style: TextStyle(
                  color: Colors.green,
```

```
    fontWeight: FontWeight.w900,
    fontSize: 30,
  ),
),
SizedBox(height: 16),
Text(
  "You can vote for only one candidate.",
  style: TextStyle(
      color: Colors.green,
      fontWeight: FontWeight.w900,
      fontSize: 14),
),
SizedBox(height: 50),
Column(
  children: [
    _buildCandidateButton(1, 'Candidate 1', Colors.redAccent),
    SizedBox(height: 16),
    _buildCandidateButton(2, 'Candidate 2', Colors.blueAccent),
    SizedBox(height: 16),
    _buildCandidateButton(3, 'Candidate 3', Colors.white),
    SizedBox(height: 32),
    Text(
      _selectedCandidate == null
          ? 'Select Candidate'
          : 'Selected Candidate: Candidate $_selectedCandidate',
```

```
  style: TextStyle(
    color: Colors.green,
    fontSize: 16,
    fontWeight: FontWeight.bold,
  ),
),
SizedBox(height: 24),
ElevatedButton(
  onPressed: () {
    if (_selectedCandidate == null) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Please select a candidate!'),
        ),
      );
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            'Proceeding with Candidate $_selectedCandidate',
          ),
        ),
      );
      Navigator.of(context).push(
        MaterialPageRoute(
```

```
                builder: (context) => EndScreen(),
              ),
            );
          }
        },
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.green,
          minimumSize: Size(double.infinity, 50),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.zero,
          ),
        ),
        child: Text(
          'Proceed',
          style: TextStyle(
            color: Color(0xFF101010),
            fontSize: 16,
            fontWeight: FontWeight.bold,
          ),
        ),
      )
    ],
  )
 ],
),
```

```
      ),
    ),
  );
}

Widget _buildCandidateButton(
    int candidateNumber, String candidateName, Color candidateColor) {
  bool isSelected = _selectedCandidate == candidateNumber;
  return ElevatedButton(
    onPressed: () {
      setState(() {
        _selectedCandidate = candidateNumber;
      });
    },
    style: ElevatedButton.styleFrom(
      minimumSize: Size(double.infinity, 50),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.all(Radius.circular(5)),
      ),
      backgroundColor: isSelected ? Colors.grey : candidateColor,
      foregroundColor: Colors.black,
    ),
    child: Text(
      candidateName,
      style: TextStyle(
```

```
          fontWeight: FontWeight.bold,
          fontSize: 16,
        ),
      ),
    );
  }
}
```

7.5.3 QR Screen

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:qr_code_scanner_plus/qr_code_scanner_plus.dart';
class QRScannerPage extends StatefulWidget {
  const QRScannerPage({super.key});
  @override
  State<QRScannerPage> createState() => _QRScannerPageState();
}
class _QRScannerPageState extends State<QRScannerPage> {
  final GlobalKey qrKey = GlobalKey(debugLabel: 'QR');
  QRViewController? controller;
  bool scanned = false;
  @override
  void reassemble() {
    super.reassemble();
```

```
  if (Platform.isAndroid) {
    controller?.pauseCamera();
  }
  controller?.resumeCamera();
}

void _onQRViewCreated(QRViewController controller) {
  this.controller = controller;
  controller.scannedDataStream.listen((scanData) {
    if (!scanned) {
      scanned = true;
      if (mounted) {
        Navigator.of(context).pop(scanData.code);
      }
    }
  });
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text("ID Verification"),
    ),
    body: Stack(
      children: [
```

```
          QRView(
            key: qrKey,
            onQRViewCreated: _onQRViewCreated,
            overlay: QrScannerOverlayShape(
              borderColor: Colors.lightGreen,
              borderRadius: 10,
              borderLength: 30,
              borderWidth: 10,
              cutOutSize: 250,
            ),
          ),
          Positioned(
            child: ElevatedButton(
              onPressed: () => Navigator.of(context).pop(null),
              child: const Text("Cancel"),
            ),
          ),
        ],
      ),
    );
  }
}
```

CHAPTER 8: SYSTEM TESTING

8.1 LEVELS OF TESTING

Testing is the process of checking if something works as expected. It helps find mistakes early, so they can be fixed before they cause bigger problems. In software, testing makes sure programs run correctly and do what they are supposed to do. It can be done manually by a person or automatically using tools. Good testing improves quality and saves time in the long run.

8.1.1 Unit Testing

Unit testing is an essential part of software testing where developers focus on checking individual parts of a program to make sure each piece works exactly as it should. By isolating small sections of the code, unit testing helps identify bugs early in the development process, which can save time and effort later when problems are harder to fix in a larger system. It involves writing specific test cases for each function or module to verify that it behaves correctly under various conditions. This not only improves the reliability of the software but also provides a safety net when changes or updates are made, ensuring that existing functionality is not unintentionally broken. With unit tests in place, developers can be more confident that their code is robust and that future enhancements will integrate smoothly with the current system. Overall, unit testing is a valuable practice that supports building high-quality software, making it easier to maintain and extend over time.

8.1.2 System Testing

System testing is a process where the whole system is examined as one complete unit. It covers several types of tests such as security, performance, and recovery checks to ensure everything runs smoothly. After initial output tests, the system is evaluated on different platforms and browsers to see if its overall functions and user interface work well in various settings. Essentially, this stage is the final phase of testing, carried out after all other tests are completed.

8.1.3 Validation Testing

The system managed incorrect inputs well by displaying helpful error messages and keeping everything stable. In validation testing, it also checked that the input data was in the right format and sequence.

8.2 TEST CASES

Test ID	Test Scenario	Test Steps	Test Data	Expected Result	Actual Result	Status
1	Valid QR code scanning	1. Select id card type 2. Scan qr code	Valid ID card	Fetch user id and redirects to OTP verification screen	Fetch user id and redirects to OTP verification screen	Pass
2	Invalid QR code Scanning	1. Select id card type 2. Scan an invalid qr code	Invalid ID card	Displays error message	Displays error message	Pass
3	OTP Verification	1. Enter valid otp received to mobile no.	Valid OTP	Redirects to biometric verification screen	Redirects to biometric verification screen	Pass
4	Biometric Verification	1. Verify face of the respective user	N/A	Redirects to user dashboard		Pass
5	Valid Vote Casting on Hyperledger Fabric	1. Log in with a registered voter account. 2. Navigate to the vote casting screen. 3. Select a valid candidate and submit the vote.	Valid election ID, Registered voter ID, Valid candidate ID	Vote is successfully recorded; a vote receipt is generated and displayed.	Vote is successfully recorded; a vote receipt is generated and displayed.	Pass

6	Duplicate Vote Attempt on Hyperledger Fabric	1. Log in with a registered voter account that has already voted. 2. Attempt to cast another vote for the same or a different candidate.	Same voter ID (already voted) Valid election and candidate IDs	error message indicating that the voter has already voted	error message indicating that the voter has already voted	Pass
7	Valid Tally Update on Solana	1. Trigger a tally update from the backend at a checkpoint. 2. Call the Solana update endpoint with the current tally value.	Valid tally value, Correct tally account public key	Solana program updates the tally	Solana program updates the tally	Pass
8	Election Creation	1. Navigate to the election commission interface. 2. Enter election details and create a new election.	Unique election ID, Valid election name and candidate list	Election is successfully created and available for vote registration.	Election is successfully created and available for vote registration.	Pass

CHAPTER 9: CONCLUSION

In conclusion, this project represents a significant step toward modernising electoral processes. By leveraging a hybrid blockchain model, biometric authentication, and a user-friendly mobile application, this project addresses critical challenges in traditional voting systems, including security, accessibility, and transparency. The system ensures tamper-proof vote recording, public auditability of results, and real-time voter authentication, fostering trust in the democratic process. Additionally, its scalability and cost-effectiveness make it suitable for elections of varying sizes, providing a robust framework for conducting secure and inclusive digital voting

CHAPTER 10: FUTURE ENHANCEMENT

In the future, the focus will be on improving the system and fixing any issues found during testing. One of the main updates will be to update the election commission interface to dynamically generate the required files and chaincode, making it easier to set up and run elections in the hyperledger fabric network smoothly. The system will also include more advanced biometric verification techniques like fingerprint scan, retina scan, etc. that could be fetched from the Aadhaar, ensuring more trust and security in the system.

CHAPTER 11: REFERENCES

[1] Hajian Berenjestanaki, M.; Barzegar, H.R.; El Ioini, N.; Pahl, C. Blockchain-Based E-Voting Systems: A Technology Review. Electronics 2024, 13, 17. https://doi.org/10.3390/electronics13010017

[2] Singh, Jagbeer, et al. Blockchain-Based Decentralized Voting System Security Perspective: Safe and Secure for Digital Voting System. 2023, https://arxiv.org/abs/2303.06306

[3] Alshehri, A.; Baza, M.; Srivastava, G.; Rajeh, W.; Alrowaily, M.; Almusali, M. Privacy-Preserving E-Voting System Supporting Score Voting Using Blockchain. Appl. Sci. 2023, 13, 1096. https://doi.org/10.3390/app13021096

[4] Spanos, Achilleas, and Ioanna Kantzavelou. A Blockchain-Based Electronic Voting System: EtherVote. 2023, https://arxiv.org/abs/2307.10726.

[5] Faruk, Md Jobair Hossain, et al. "Bie Vote: A Biometric Identification Enabled Blockchain-Based Secure and Transparent Voting Framework." 2022 Fourth International Conference on Blockchain Computing and Applications (BCCA), 2022, pp. 253–58, https://doi.org/10.1109/BCCA55292.2022.9922588.

[6] Khudoykulov, Z., Tojiakbarova, U., Bozorov, S., & Ourbonalieva, D. (2021). Blockchain Based E-Voting System: Open Issues and Challenges. 2021 International Conference on Information Science and Communications Technologies (ICISCT), 1–5. https://doi.org/10.1109/ICISCT52966.2021.9670245

CHAPTER 12: APPENDIX

Figure 12.1 Landing Page

Figure 12.1 Registration Page

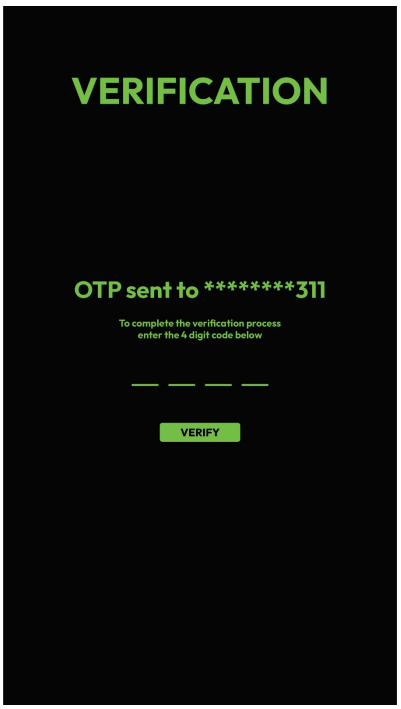

Figure 12.3 Verification Page

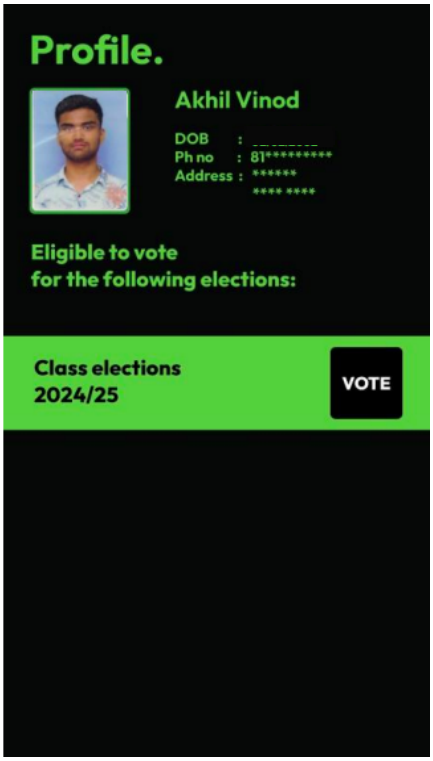

Figure 12.4 Profile Page

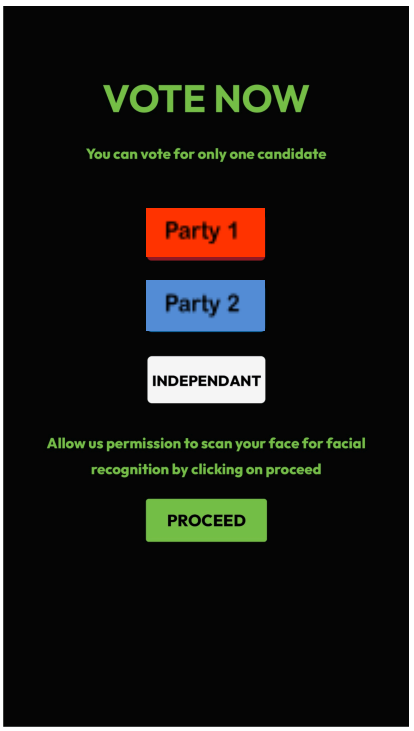

Figure 12.5 Voting Page

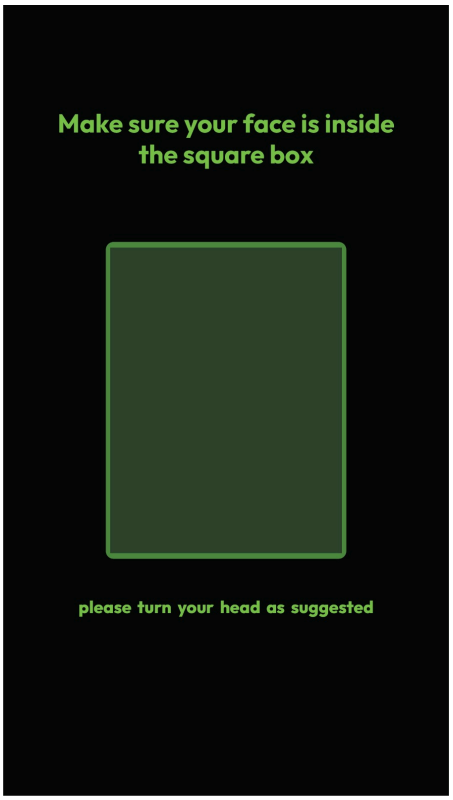

Figure 12.6 Liveness detection Page

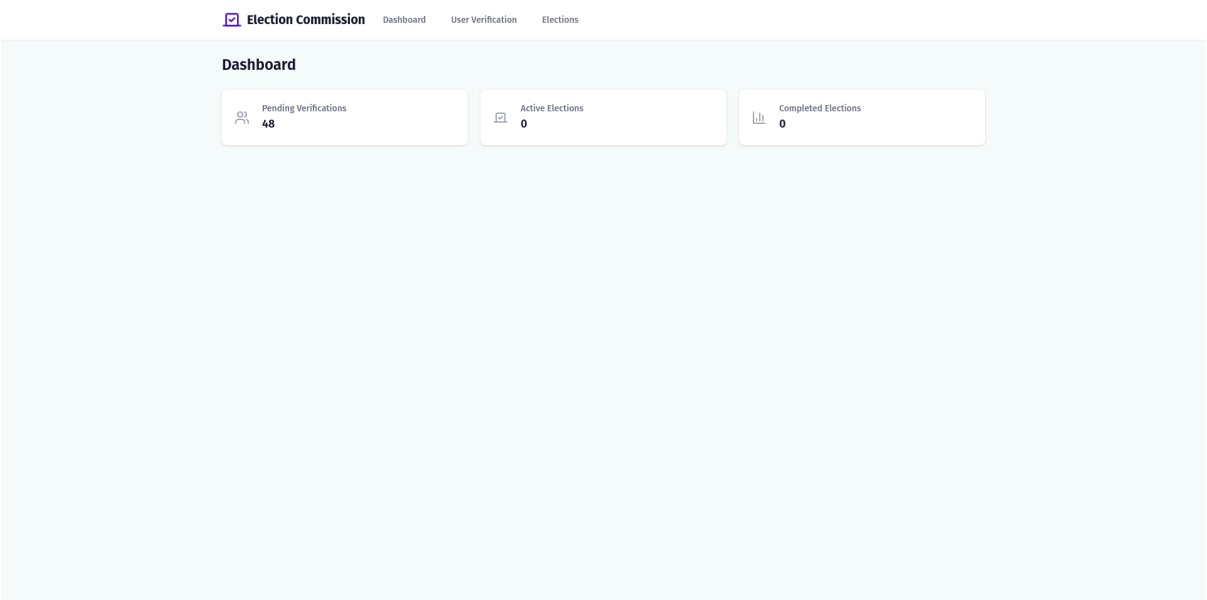

Figure 12.7 Election Commission Dashboard

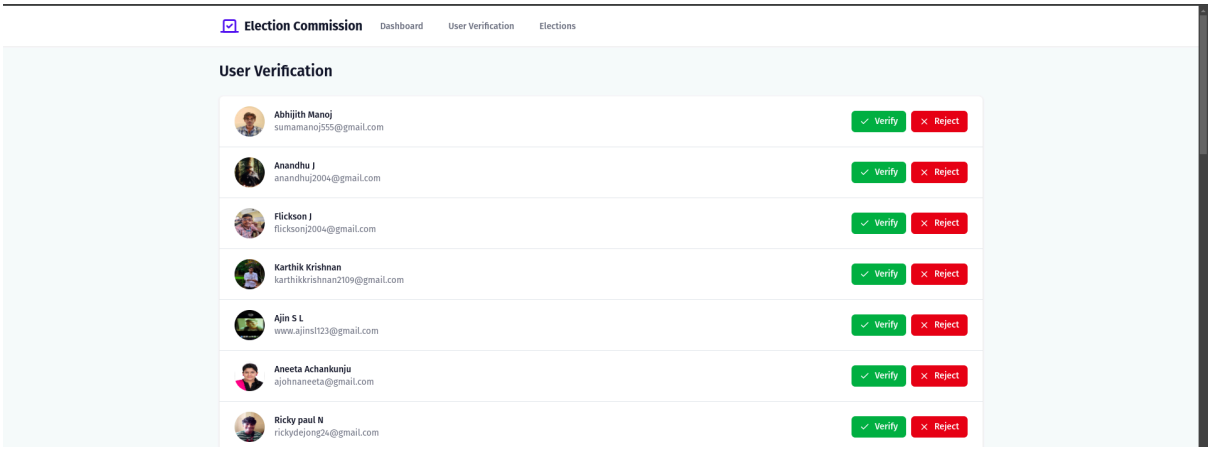

Figure 12.8 Election Commission Manual User Verification

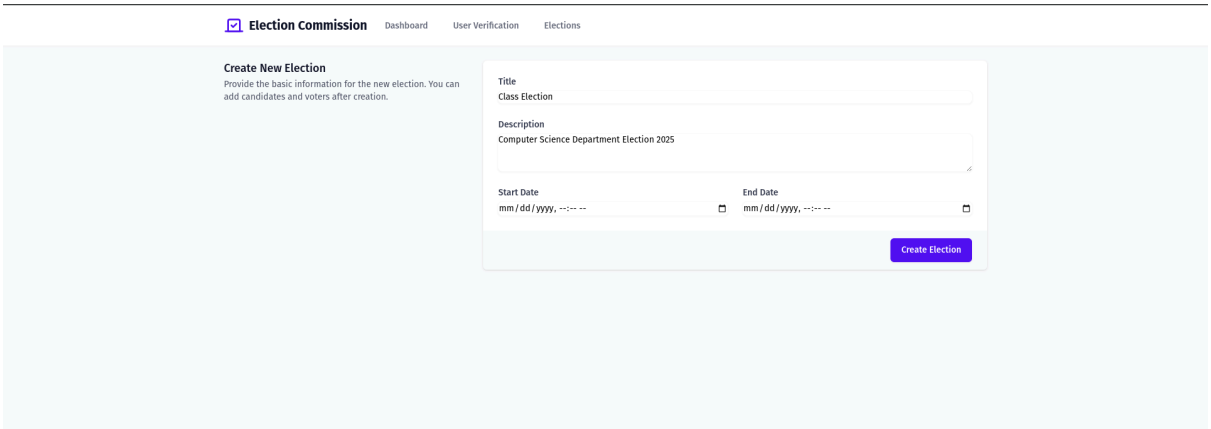

Figure 12.9 Create Election

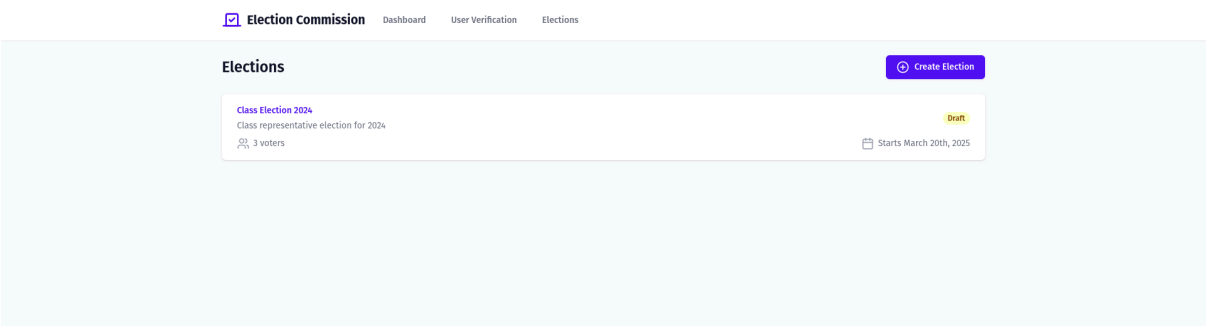

Figure 12.10 Elections View Page

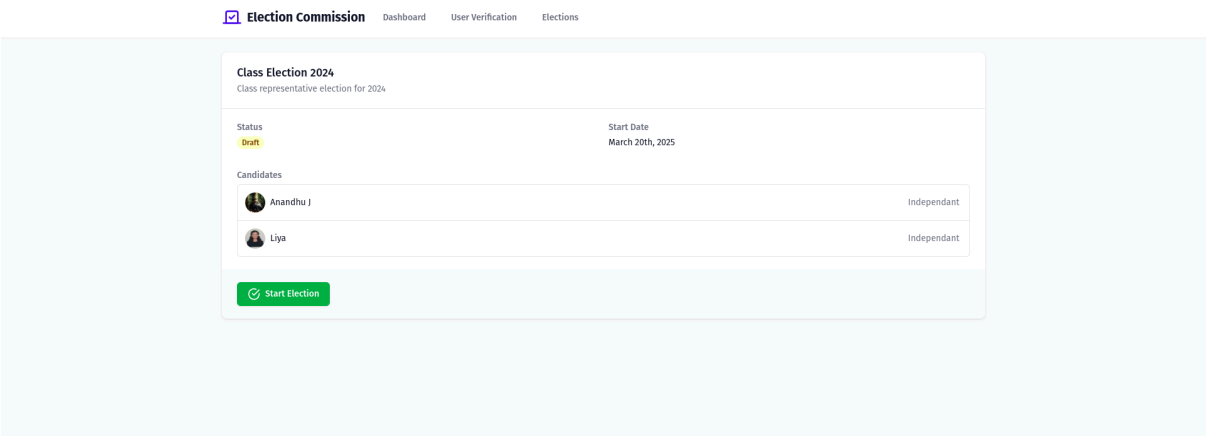

Figure 12.11 Start Election Page

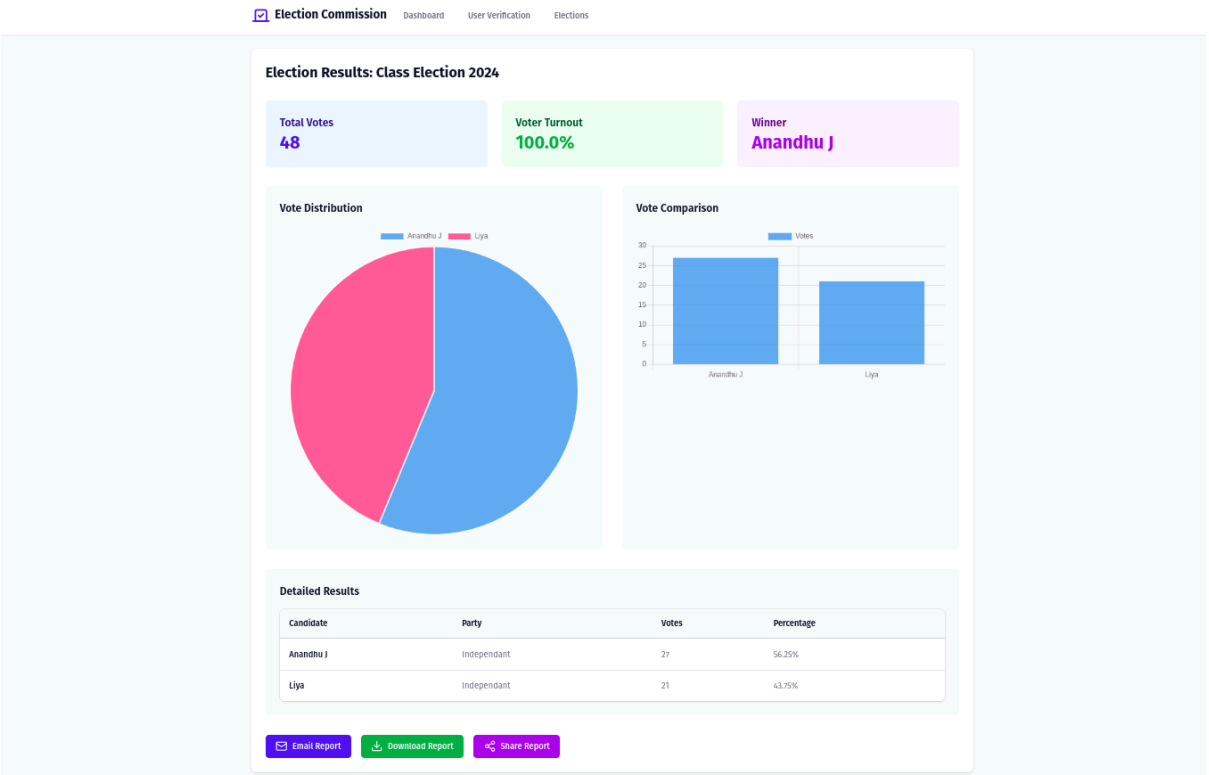

Figure 12.12 Election Report

CHAPTER 13: USER MANUAL

The Blockchain Based Hybrid Voting System comes with both a mobile app for the voters and a web interface for the election commission to create and manage elections. It is designed for better user accessibility and simpler and straightforward user experience.

Voter

1. Voter Registration:

a. Install the VoteX mobile application on the user's device.
b. Select the ID Card type.
c. Scan the QR Code present in the user's ID Card.
d. Enter OTP received to the user's phone number in the input.
e. Scan the user's face for automatic biometric verification. (If automatic biometric verification fails, opt for manual verification which will be done by election commission).
f. Upon successful verification users can access their profile page.

2. Vote Casting:

a. Enter the voting page by clicking the "Vote" button that will be shown in the user's profile page.
b. Select the candidate the user wants to vote.
c. Run liveness detection test.
d. Users will receive a receipt upon successful vote.

Election Commission

1. Manual User Verification:

a. Navigate to the "User Verification" page to view all the pending verification requests.
b. Select "Verify" if the captured image and the stored image of the user matches.
c. Select "Reject" if the images doesn't match

2. Create Election:

a. Navigate to the "Elections" page.
b. Click the "Create Election" button.
c. Enter details like Title, Description, Start and End dates.
d. Select the candidates and enter "Create Election" button
e. Assign voters to the created election.

3. Manage Election:

a. Navigate to the "Elections" page where we can see the created elections.
b. Select the election and press "Start Election" to start the election.
c. Press "Stop Election" to end the election.
d. Press "View Full Report" to see a detailed report of the election and publish the results.