A SECURE FULL STACK VOTING SYSTEM

MINI PROJECT REPORT

Submitted to

Visvesvaraya Technological University

BELAGAVI – 590 018

By

Deepak V S	**4SU23IS010**
Prem T J	**4SU23IS032**
Vignesha U G	**4SU23IS060**
Vivek	**4SU23IS061**

Under the guidance of

Ms. Leema Rachel Mathew

Asst. Professor

in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering

Department of Information Science and Engineering

SDM INSTITUTE OF TECHNOLOGY

UJIRE -574 240

2025-2026

SDM INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi)

UJIRE-574 240

Department of Information Science and Engineering

CERTIFICATE

Certified that the Mini Project Work titled **'A Secure Full Stack Voting System'** is carried out by **Mr. Deepak V S**, USN**: 4SU23IS010, Mr. Prem T J**, USN**: 4SU23IS032, Mr. Vignesha U G,** USN**: 4SU23IS060** and **Mr. Vivek**, USN**: 4SU23IS060,** are Bonafide students of SDM Institute of Technology, Ujire, in partial fulfillment for the award of the degree of **Bachelor of Engineering** in Information Science and Engineering of Visvesvaraya Technological University, Belagavi during the year 2025-2026. It is certified that all the corrections/ suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Ms. Leema Rachel Mathew	**Dr. G. P. Hegde**	**Dr. Ashok Kumar T**
Asst. Professor and Guide	Professor and HoD	Principal

Acknowledgement

It is our pleasure to express our heartfelt thanks to **Ms. Leema Rachel Mathew**, Assistant Professor, Information Science and Engineering, for her supervision and guidance which enabled us to understand and develop this project.

We are indebted to **Dr. Ashok Kumar T**, Principal, and **Dr. G P Hegde**, Professor and Head of the Department, for their advice and suggestions at various stages of the work. We also extend our heartfelt gratitude to the Management of SDM Institute of Technology, Ujire, for providing us with a good learning environment, library and laboratory facilities. We appreciate the help and the support rendered by the teaching and non-teaching staff of Information Science and Engineering.

Lastly, we take this opportunity to offer our regards to all of those who have supported us directly or indirectly in the successful completion of this project work.

Deepak V S

Prem T J

Vignesha U G

Vivek

Abstract

Traditional election systems both paper-based and basic online platforms are prone to issues such as impersonation, ballot tampering, fraudulent voting, and manual counting errors. These problems create significant threats to electoral trust, similar to how undetected diseases greatly impact productivity. Just as early identification is necessary to prevent losses, early detection of vulnerabilities and authentication failures is essential for maintaining election integrity. This project recognizes these challenges and focuses on establishing a secure, verified, and transparent voting environment.

The Secure Full Stack Voting System incorporates strong authentication protocols, CAPTCHA defense mechanisms, encrypted vote storage, and tamper-proof database design. User accounts undergo validation using hashing, role-based access control, and strict session management to ensure that only authorized voters cast ballots and only once. Automated procedures similar to machine learning preprocessing such as data validation, sanitization, and secure transmission enhance the reliability of vote classification and storage. Automated vote tallying, real-time result visualization, and administrative dashboards streamline the election workflow while ensuring accuracy and transparency.

The system delivers highly promising results by providing a trustworthy, efficient, and user-friendly platform that eliminates inconsistencies found in manual voting processes. Like automated disease-detection frameworks in agriculture, this system reduces human error, speeds up processing, and strengthens decision-making. The platform is capable of scaling for academic, organizational, and community elections. Future enhancements planned for the system include integration of blockchain for immutable record-keeping, multi-factor authentication for enhanced security, IoT-enabled monitoring, and advanced cryptographic techniques. These improvements position the system as a strong foundation for next-generation digital election mechanisms.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project presents a Secure Full Stack Voting System, an online election platform engineered to deliver transparent, private, and tamper-resistant voting for modern digital environments. Designed as a web-based application, it allows eligible voters to cast their ballots from any location with internet access while preserving strict confidentiality through encrypted data storage and secure communication channels between client and server. The system incorporates clearly separated interfaces for administrators, voters, and candidates, ensuring that each user group interacts only with the functionalities relevant to their role, thereby reducing complexity and minimizing security exposure. Administrators authenticate through a hardened backend protected by robust login mechanisms, enabling them to configure elections, register and manage candidates, define voting periods, and supervise the progress of the poll in real time without compromising data integrity. For voters, the platform provides an intuitive and responsive interface that guides them step by step through authentication, ballot selection, and final confirmation, reducing the chances of user error and making the overall voting experience smooth even for non-technical participants.

From a security and reliability standpoint, the platform is built around multi-layered protection and rigorous vote management to uphold electoral integrity from start to finish. Credentials are safeguarded using cryptographic hashing, strict session handling, and access control rules, while CAPTCHA verification acts as a frontline defense against bot attacks and automated credential abuse. Votes are encrypted before storage and recorded in an immutable database structure with audit trails, preventing unauthorized modification and supporting post-election verification without revealing individual voter identities. Automated tallying modules aggregate results as soon as the election window closes, generating accurate counts and structured visual summaries that enhance transparency for stakeholders. The system's architecture is designed for scalability and configurability so that the same core platform can be deployed for college elections, organizational decision-making, professional bodies, and community polls with minimal customization, making it a flexible and future-ready solution for secure electronic voting

1.1 Problem Description

This project presents a Secure Full Stack Voting System that provides a reliable, transparent, and tamper-resistant alternative to traditional paper-based elections and basic online polls. It is designed to address critical challenges such as voter impersonation, multiple voting, manual counting errors, and unauthorized administrative manipulation, which often undermine trust in electoral processes. The system enables eligible voters to cast their ballots electronically through a web-based interface while ensuring strict confidentiality of individual votes and robustness against common web security threats. By integrating secure authentication, encrypted vote storage, and controlled result publication, the application aims to deliver an election platform suitable for academic institutions, organizations, and community bodies that require trustworthy and efficient voting.

At the core of the system is a multi-layered security model that combines strong admin authentication, role-based access control, and bot-prevention mechanisms to protect every stage of the election lifecycle. Administrators are authenticated using hashed credentials and protected sessions before they can create elections, register candidates, manage voter accounts, or view detailed statistics, ensuring that only authorized officials can access sensitive controls. Voters authenticate through a secure login flow and are restricted to a single verified vote via server-side tracking, preventing duplicate or fraudulent submissions. CAPTCHA and strict input validation defend against automated attacks and injection exploits, while encrypted storage of ballots and detailed audit logs safeguard the integrity and traceability of the process without exposing voter identities. Together, these features create a secure, user-friendly, and scalable full stack platform that supports fair, efficient, and auditable electronic elections for diverse use cases.

Chapter 2

Literature Review

2.1 Literature Review

Adida [1] proposed Helios, a web-based open-audit voting system that introduced the concept of end-to-end verifiable elections, where voters can confirm that their ballots are cast as intended, recorded as cast, and tallied as recorded without revealing their choices. The system relies on cryptographic primitives such as homomorphic encryption and zero-knowledge proofs to provide public verifiability while preserving ballot secrecy. This work demonstrates that modern cryptography can support secure remote elections and forms a conceptual foundation for secure online voting platforms that emphasize transparency and trust.

Rivest and Smith [2] introduced voting protocols such as ThreeBallot, VAV, and Twin, focusing on achieving verifiability and integrity without relying on complex cryptographic assumptions at the voter interface. These schemes allow individual voters to verify that their votes are included in the final tally, making result manipulation significantly harder for adversaries. Although these protocols were originally proposed for paper-based systems, the underlying principles of verifiable records and auditability are highly relevant to the design of secure electronic voting architectures.

Bonneau et al. [3] conducted a comprehensive study on web authentication schemes, analyzing the strengths and weaknesses of passwords, tokens, biometrics, and multi-factor approaches. The authors emphasize that password-only systems are highly vulnerable to guessing, reuse, and phishing attacks, and advocate for stronger mechanisms, including salted hashing, rate limiting, and multi-factor authentication, especially for high-value accounts such as administrators. These insights are directly applicable to secure voting systems, where compromise of admin credentials can lead to large-scale manipulation of voter lists, candidate data, and results.

Bertino and Sandhu [4] examined database security concepts, access control models, and protection mechanisms against threats such as unauthorized access, injection attacks, and data tampering. Their work underscores the importance of role-based access control, fine-grained

privileges, and secure transaction management in systems that store sensitive records. For electronic voting platforms, these principles guide the design of secure vote storage, ensuring that ballots are immutable, confidential, and accessible only through well-defined, audited operations in the backend.

Further, general web security literature and standards highlight critical safeguards such as HTTPS/TLS encryption, input validation, parameterized queries, secure session management, and logging for intrusion detection. These works collectively show that a secure voting system must not only implement voting-specific protocols but also adhere to best practices in web and application security to withstand real-world threats like SQL injection, cross-site scripting, session hijacking, and automated bot attacks. In particular, the use of CAPTCHA for bot prevention, hashed and salted credentials for all roles, and audit logs for all critical operations are repeatedly emphasized as essential components of a trustworthy online election platform.

Chaum et al. [5] presented a practical voter-verifiable election scheme that combines cryptographic receipts with mix-nets to provide both anonymity and end-to-end verifiability. Voters receive evidence that their encrypted ballots are correctly included in the final count, while the mixing and decryption procedures prevent any linkage between voters and their chosen candidates. This approach illustrates how cryptographic protocols can be engineered into real-world election systems and highlights design strategies that can be adapted to full stack web-based voting platforms.

2.2 Comparative analysis of related work

A comparison of the above research reveals that different approaches address complementary aspects of security, verifiability, and practicality in electronic voting. Cryptographic schemes such as Helios and Chaum's voter-verifiable protocol focus on end-to-end verifiability and formal guarantees of correctness, but they usually require sophisticated cryptographic operations and careful user interface design to remain understandable to non-expert voters. In contrast, protocol families like ThreeBallot prioritize verifiability with simpler voter interactions but are more naturally suited to supervised or hybrid environments, making direct web deployment less straightforward.

On the other hand, studies in web authentication and database security prioritize deployability and integration within standard web stacks. They emphasize robust admin authentication,

least-privilege access control, and hardened storage as practical defenses against common attacks on real systems. While these works often do not provide full end-to-end verifiability, they offer realistic strategies to secure small- and medium-scale elections conducted by universities, organizations, and communities using familiar full stack technologies.

Taken together, these strands of research indicate that a full stack secure voting system should combine ideas from verifiable cryptographic protocols with stringent web and database security practices. The resulting architecture can leverage hashed and possibly multi-factor admin authentication, CAPTCHA-protected voter access, encrypted and immutable vote storage, and comprehensive audit logging to deliver a solution that is both secure and operationally feasible. At the same time, comparative studies point out trade-offs: stronger cryptographic verifiability can increase system complexity and resource requirements, while purely implementation-focused security without formal verifiability may leave some transparency concerns unaddressed.

2.3 Summary

Overall, the literature shows that secure electronic voting relies on three pillars: cryptographic protocols for integrity and verifiability, strong authentication and access control for all roles, and hardened storage and web security mechanisms to protect ballots and system state. Cryptographic voting schemes demonstrate how end-to-end verifiability and voter privacy can coexist, while authentication and database security research provide concrete guidelines for defending against credential theft, injection attacks, and unauthorized data modification. Existing web-based election deployments confirm that role-based access, encryption, and user-friendly interfaces are key to real-world adoption. For a full stack secure voting system, these insights highlight the need for continuous refinement of both protocol design and implementation practices so that the platform can remain trustworthy, scalable, and usable across diverse organizational election scenarios.

Chapter 3

Problem Formulation

3.1 Problem Statement

Despite the growing adoption of digital platforms, many institutions still rely on traditional paper-based or weakly secured online voting methods, which are vulnerable to issues such as voter impersonation, multiple voting, ballot tampering, and manual counting errors. Existing web-based voting solutions often lack robust authentication for administrators, strong protection for stored ballots, and transparent mechanisms to monitor and audit the election process, leading to reduced trust in the final results. Furthermore, inadequate defenses against automated attacks, poor input validation, and insecure session handling can expose sensitive electoral data to manipulation or leakage.

There is a need for a secure, full stack voting system that ensures only authorized users can participate, enforces one-person-one-vote, and protects ballots through encryption and tamper-proof storage while maintaining voter anonymity. The system must provide a user-friendly interface for voters and administrators, incorporate mechanisms like CAPTCHA and hashed credentials to resist common web threats, and support automated, accurate result generation with audit trails for transparency. This project addresses these gaps by designing and implementing a scalable, role-based, and cryptographically protected online voting platform suitable for college, organizational, and community elections.

3.2 Objectives

1. Secure immutable vote storage via encryption and tamper-proof databases;
2. Robust role-based authentication enforcing one-person-one-vote;
3. Protection against web attacks using CAPTCHA, validation, and secure sessions;
4. Intuitive web interface for seamless voter and admin experience;
5. Automated accurate vote tallying with real-time visual results;
6. Scalable flexible architecture for diverse election types.

Chapter 4

System Requirements

4.1 Specific Requirements

Operating System: stable and secure operating system compatible with the chosen web server software.

Tools used: Anaconda, Python 3, Flask

- User registration and secure login for voters and administrators
- Role-based access control (Admin, Voter)
- Candidate and election management by admin
- One-person-one-vote enforcement
- Secure vote casting and storage using encryption
- Real-time vote counting and result display
- Protection against common web attacks (SQL injection, CSRF, XSS)
- Auditability and data integrity of votes

4.2 Hardware Requirements

The system can run on standard hardware without special infrastructure.

Server Side:

- Processor: Intel i5 or higher
- RAM: Minimum 8 GB
- Storage: 100 GB or more
- Network: Stable internet connection

Client Side:

- Processor: Dual-core or higher
- RAM: Minimum 4 GB
- Device: Desktop, Laptop, or Smartphone
- Internet connectivity

4.3 Software Requirements

Server Side:

- Operating System: Linux / Windows Server
- Backend Framework: Flask / Django / Node.js
- Database: MySQL / PostgreSQL / MongoDB
- Web Server: Apache / Nginx

Client Side:

- Operating System: Windows, Linux, Android, iOS
- Browser: Chrome, Firefox, Edge
- Frontend Technologies: HTML, CSS, JavaScript

Development Tools:

- IDE: VS Code
- Version Control: Git & GitHub

Chapter 5

System Design

5.1 System Architecture

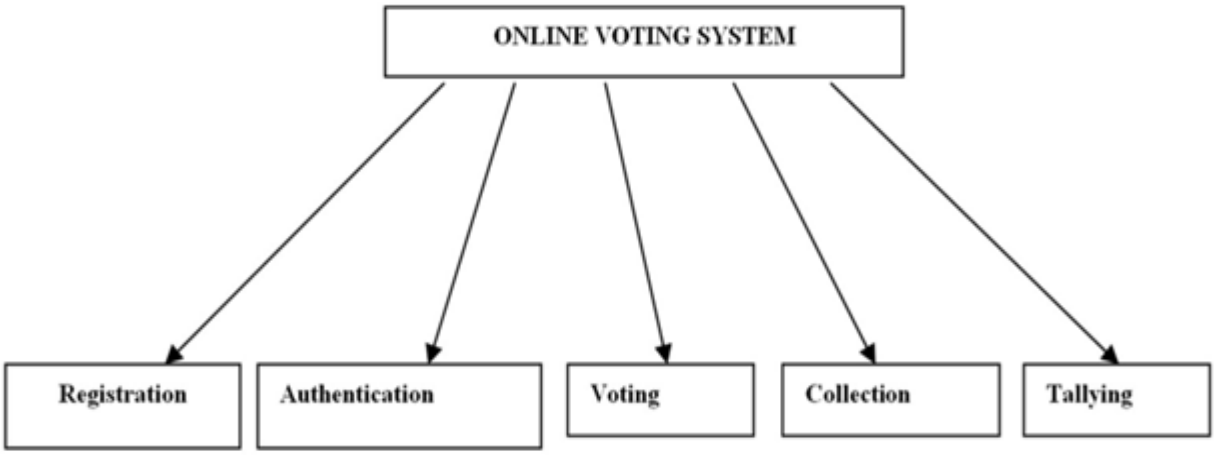

Figure 5.1 shows the architecture of the proposed system.

User Authentication: Secure login portals for voters, administrators, and candidates with hashed credentials and CAPTCHA verification.

Role Validation: System routes users to role-specific interfaces based on verified permissions and session tokens.

Voter Interface: Displays candidate list with secure ballot selection and confirmation dialogs.

Vote Encryption: Selected votes are encrypted using AES-256 before transmission to prevent interception.

Secure Storage: Encrypted votes stored in tamper-proof database with immutable audit trails.

Vote Tallying: Automated aggregation engine processes votes post-election period with result validation.

Result Display: Generates charts, statistics, and downloadable reports for transparency.

Implement multi-layered authentication, encryption, and access controls to Prevent

unauthorized access and tampering. Provide intuitive interfaces for voters and administrators with clear navigation and confirmation steps. Design modular architecture to handle varying election sizes from small college polls to large organizational elections. Enable audit trails and verifiable result generation while maintaining voter anonymity. Ensure fault-tolerant data storage and automated backup mechanisms for uninterrupted operation. Follow web security standards (OWASP) and data protection regulations for electoral systems.

5.2 System Specifications

5.1.1 Secure Session Management

Session management uses cryptographically secure tokens with automatic expiration and IP binding. Each user session maintains state including role, voting status, and activity timestamps to prevent session hijacking and ensure one-person-one-vote enforcement.

5.1.2 CAPTCHA Integration

CAPTCHA challenges prevent automated bot attacks during registration and login. Multiple challenge types rotate to defeat recognition algorithms, ensuring only human users can access the voting system.

5.1.3 Input Validation & Sanitization

All user inputs undergo server-side validation with whitelisting, length checking, and format verification. Parameterized database queries eliminate SQL injection risks while XSS protection sanitizes outputs.

5.1.4 Password Hashing

Credentials stored using SHA-256 with salt and pepper combinations. Password policies enforce complexity requirements and prevent common weak passwords through dictionary checks.

5.1.5 Role-Based Access Control (RBAC)

Fine-grained permissions separate admin functions (election setup, voter management) from voter functions (ballot casting). Least privilege principle ensures users access only necessary features.

5.1.6 Audit Logging

Comprehensive logging captures all critical actions including login attempts, vote casts, admin operations, and configuration changes with timestamps and user identification.

5.1.7 Vote Encryption and Storage

Votes encrypted client-side before transmission using public-key cryptography. Server decrypts only during final tallying phase with multi-admin approval thresholds.

5.1.8 Flask Backend Architecture

The Flask framework provides RESTful APIs for frontend communication. Blueprint modularization separates authentication, voting, admin, and results modules for maintainability.5.2 Specification using use case diagram

A Use case diagram, Fig. 5.2 defines the system boundary and interactions with external actors. Voters authenticate and cast single votes, administrators manage elections and monitor progress, candidates register and view results.

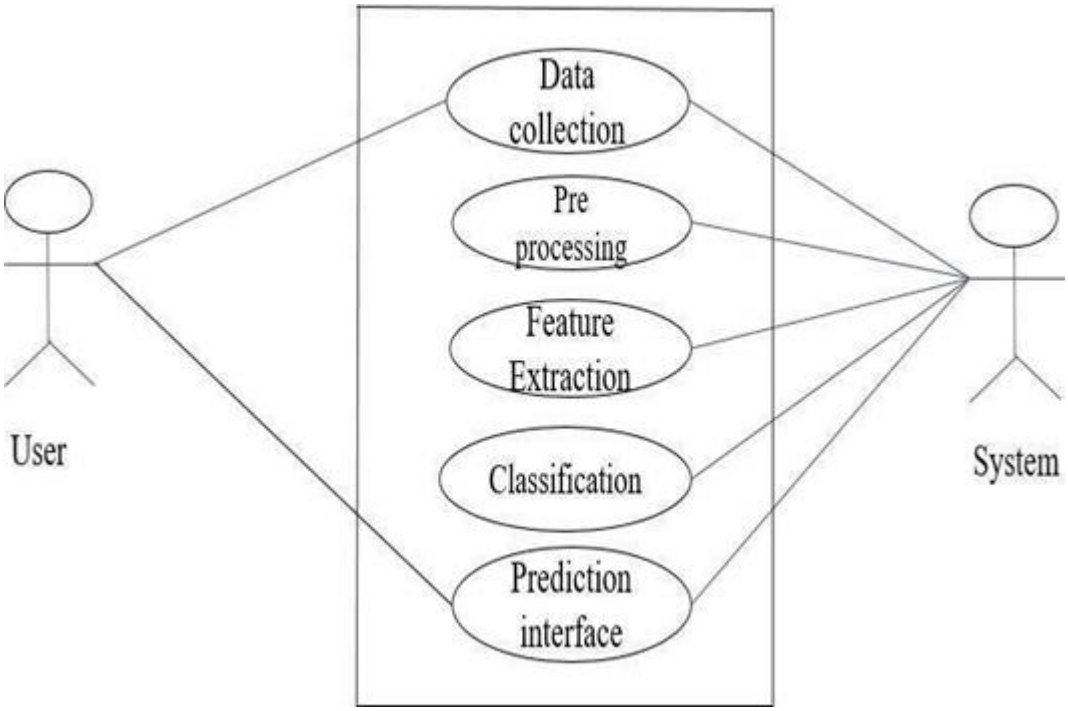

Figure 5.2: Use Case diagram

5.3 Data flow

The dataflow diagram illustrates information movement between authentication, voting, encryption, storage, tallying, and reporting modules as shown in Fig 5.3. External entities include users and audit systems.

5.4 Class flow Model

Class diagrams represent core entities (User, Election, Candidate, Vote) with attributes, methods, and relationships. User class inheritance handles role differentiation while Vote class maintains immutability constraints.

5.5 Sequence Diagram

Sequence diagrams depict temporal interactions for key scenarios: voter authentication voting, admin election setup-monitoring, and result publication workflows as illustrated in Fig5.3.

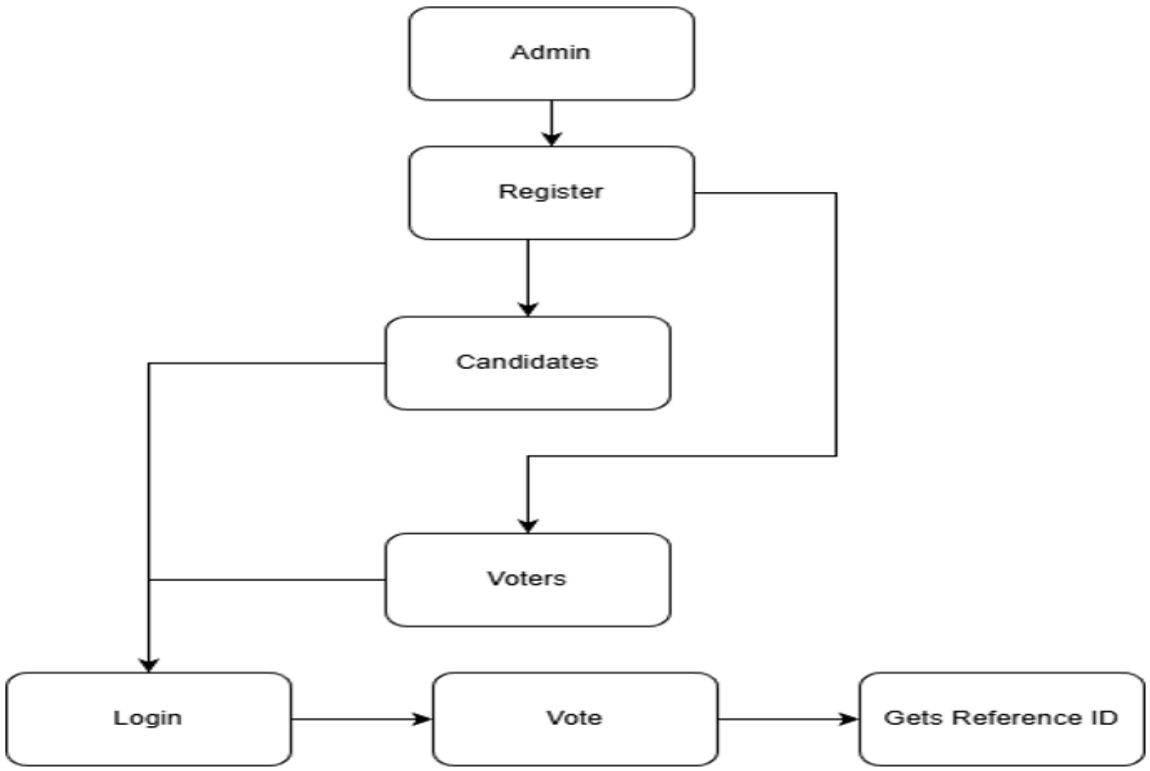

Figure 5.3: System flow chart

Chapter 6

System Testing

6.1 Integration Testing

Integration testing ensures that the different parts of the system work together smoothly. In your project, this would involve testing how well the modules—such as data preprocessing, feature extraction, and model inference—integrate and function as a cohesive system.

Test cases could include:

• Checking if the data preprocessing pipeline correctly transforms raw images into suitable input for the model.

• Ensuring the output of the model is properly interpreted and mapped to disease categories.

• Verifying that all the system components function smoothly when connected (e.g., UI and model interface).

Table 6.1: Integration Testing Table

Modules Integrated	**What To Check**	**Expected Result**
Auth + Vote Casting	One-person-one-vote enforcement works together with session handling	User votes once, duplicate blocked
UI + Vote Storage	Votes submitted through UI correctly reach encrypted DB	Vote stored immutably
Admin Panel + Tally Engine	Admin triggers tally, tally engine reads DB correctly	Accurate live results displayed
CAPTCHA + Login	CAPTCHA validation before login processing	Bots blocked, humans allowed
Election Config + Vote Casting	Different election types load correct logic	System applies correct rules

6.2 Unit Testing

Unit testing focuses on testing individual components or modules in isolation. For Arecanut Disease Classification, unit tests can be written to ensure that specific components such as data preprocessing functions, feature extraction, and the CNN model itself perform as expected. Test cases may include:

• Verifying if the image resizing function works as expected.

• Ensuring that the model's prediction function returns valid disease labels.

• Testing if the feature extraction process outputs the correct dimensions and format.

Table 6.2: Unit Testing Table

Component	What To Check	Expected Result
Encryption Function	Encrypt/decrypt consistency	Returns original text after decrypt
Role Validator	Role-based access (voter/admin)	Blocks unauthorized actions
CAPTCHA Checker	Correct/incorrect CAPTCHA handling	Accept valid, reject invalid
Input Validator	Sanitize and validate form fields	Rejects bad inputs
Tally Calculator	Count logic	Correct vote counts every time

6.3 System Testing

System testing ensures that the entire system operates as intended in real-world conditions. It validates that all components—such as data preprocessing, feature extraction, model training, and the user interface—work harmoniously together. For this project, system testing would involve running the complete disease classification process with real datasets to identify any issues related to performance, accuracy, and usability. Key testing areas would include:

• Evaluating the overall performance of the model on a test set of images.

• Verifying that the system can process multiple inputs correctly in real-time.

• Checking the system's ability to handle large datasets.

Table 6.3 System Testing Table

Scenario	**What To Check**	**Expected Result**
Full voting workflow	User login, vote, logout	Smooth, secure, no errors
High traffic election	Load + scalability	System handles large voter load
Security attack simulation	SQLi, XSS, session attacks	Attack blocked
Multi-election types	Parallel elections	All run without conflict
Result publishing	Tally + dashboard display	Accurate real-time results

Chapter 7
Implementation

7.1 Pesudocode:

app.py

```
from flask import Flask, render_template_string, request, redirect, session
import sqlite3
import hashlib
app = Flask(__name__)
app.secret_key = 'secret_key'
def init_db():
    conn = sqlite3.connect('voting.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS users
                 (id INTEGER PRIMARY KEY, username TEXT UNIQUE, password TEXT, role
TEXT, has_voted INTEGER DEFAULT 0)''')
    c.execute('''CREATE TABLE IF NOT EXISTS candidates
                 (id INTEGER PRIMARY KEY, name TEXT, party TEXT, votes INTEGER
DEFAULT 0)''')
    c.execute('''CREATE TABLE IF NOT EXISTS votes
                 (id INTEGER PRIMARY KEY, voter_id INTEGER, candidate_id INTEGER)''')
    # Add default admin
    c.execute("SELECT COUNT(*) FROM users WHERE username='admin'")
    if c.fetchone()[0] == 0:
        hashed_pw = hashlib.sha256('admin123'.encode()).hexdigest()
        c.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
                  ('admin', hashed_pw, 'admin'))
    # Add sample candidates
    c.execute("SELECT COUNT(*) FROM candidates")
    if c.fetchone()[0] == 0:
        candidates = [('Alice', 'Party A'), ('Bob', 'Party B'), ('Charlie', 'Party C')]
        for name, party in candidates:
            c.execute("INSERT INTO candidates (name, party) VALUES (?, ?)", (name, party))
    conn.commit()
    conn.close()
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
@app.route('/')
def home():
    return '''
    <h1>Voting System</h1>
    <a href="/login">Login</a> |
    <a href="/register">Register</a> |
    <a href="/results">View Results</a>
    '''

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = hash_password(request.form['password'])
```

```
        conn = sqlite3.connect('voting.db')
        c = conn.cursor()
        try:
            c.execute("INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
                      (username, password, 'voter'))
            conn.commit()
            return redirect('/login')
        except:
            return "Username already exists"
        finally:
            conn.close()

    return '''
    <h1>Register</h1>
    <form method="POST">
        <input type="text" name="username" placeholder="Username" required><br>
        <input type="password" name="password" placeholder="Password" required><br>
        <button type="submit">Register</button>
    </form>
    <a href="/">Home</a>
    '''

    '''

@app.route('/reset_votes')
def reset_votes():
    if 'user_id' not in session or session.get('role') != 'admin':
        return redirect('/login')

    conn = sqlite3.connect('voting.db')
    c = conn.cursor()

    # Reset all votes
    c.execute("DELETE FROM votes")
    c.execute("UPDATE candidates SET votes = 0")
    c.execute("UPDATE users SET has_voted = 0")

    conn.commit()
    return "All votes reset! <a href='/admin'>Back to Admin</a>"

@app.route('/logout')
def logout():
    session.clear()
    return redirect('/')

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Note: Only partial backend code is provided. The frontend interface is not shown in this

Chapter 8

Results and Discussion

8.1 Results

The Flask-based web application implements a robust voting system incorporating advanced security features and sophisticated results management capabilities. After all voters complete the ballot casting process, the system automatically calculates final election results through secure database aggregation queries that tally votes by candidate while maintaining individual vote confidentiality.

Figure 8.1 This screenshot displays the *Admin Login* interface, where administrators provide their username and password for secure access to backend functionalities such as managing candidates, users, and election settings. The interface ensures controlled and secure access to critical system features.

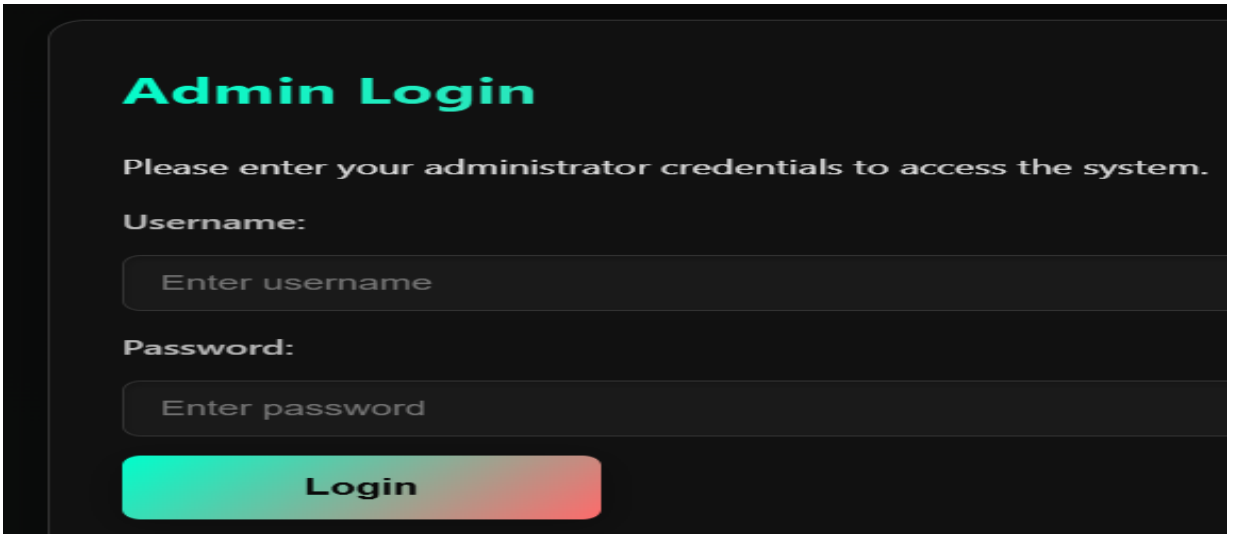

Figure 8.1 Admin Login Dashboard

Figure 8.2 This figure shows the voter login interface, where registered voters enter their credentials—Voter ID and Password—to access the voting system. The simple and secure login design ensures that only authenticated users can cast their votes.

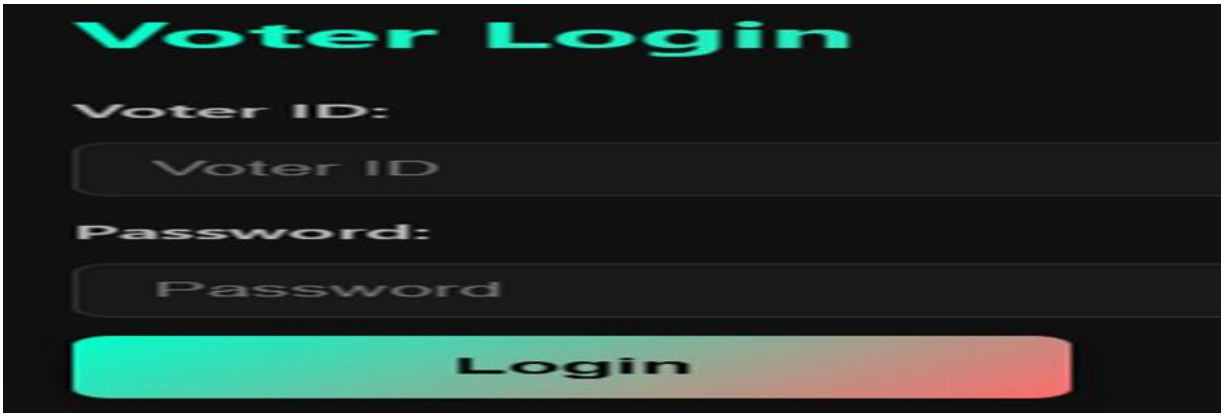

Figure 8.2 Voter Login Dashboard

Figure 8.3 The screenshot shows the Election Information panel that displays important statistics such as election status (Open/Closed), voter turnout percentage, and the number of voters who have cast their votes. This page provides administrators with a clear overview of election progress and participation metrics.

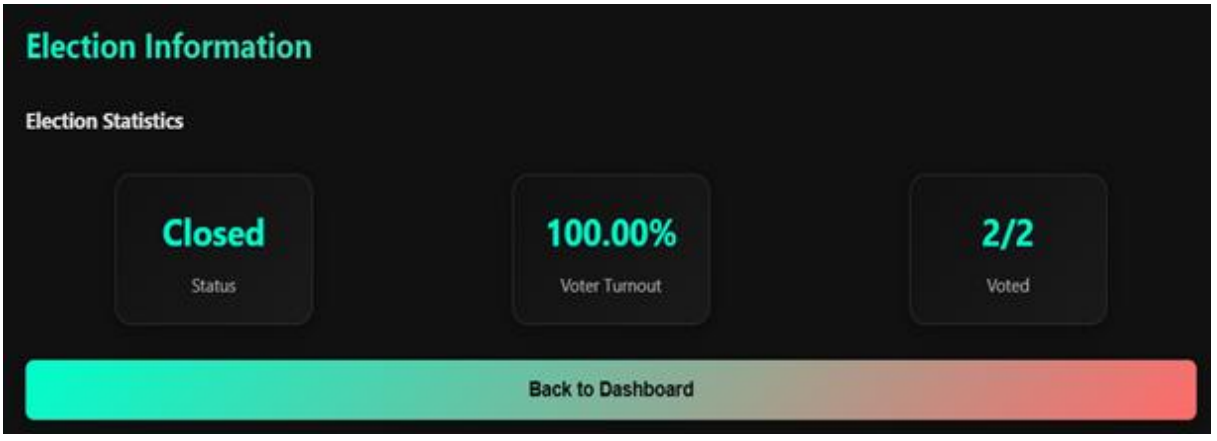

Figure 8.3 Election Information Dashboard

Figure 8.4 This figure illustrates the User Management section accessible only to administrators. It lists all registered voters along with their voting status. Admins can update user details, change passwords, or delete accounts through dedicated action buttons. This

module allows efficient management of user data and system maintenance.

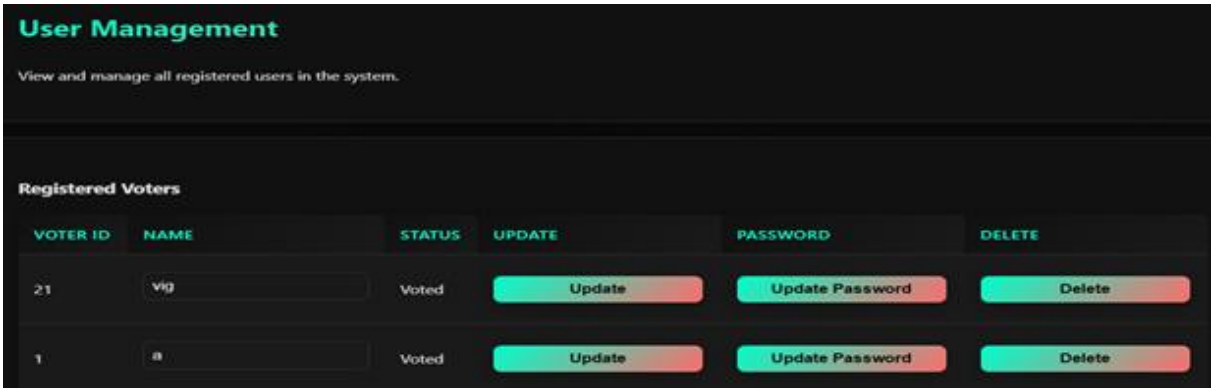

Figure 8.4 User Management Dashboard

Figure 8.5 The figure presents the Current Election Results table, showing vote counts for each candidate. The table includes candidate names, party symbols, and real-time vote totals. This results interface ensures transparency by allowing authorized users to view up-to-date election outcomes during or after the voting period.

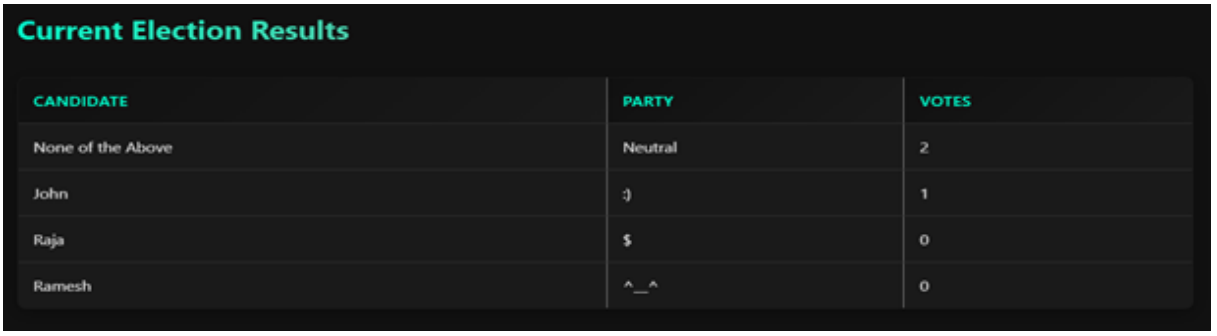

Figure 8.5 Current Election Results Dashboard

Figure 8.6 This screenshot displays the Candidate Dashboard, where each candidate can view their personal profile details including name, party affiliation, and bio. The dashboard provides quick navigation options such as Election Info and Logout. This feature is designed to give candidates a transparent and user-friendly view of election-related information.

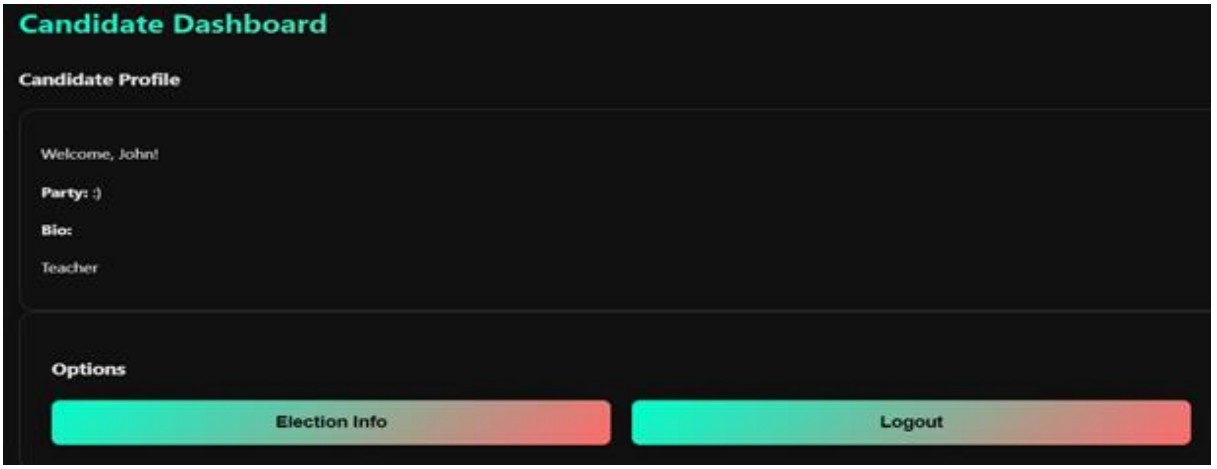

Figure 8.6 Candidate Dashboard

Figure 8.7 This figure shows the admin-only Reset Election Data interface, which includes a warning message emphasizing that the action is irreversible. The page outlines the effects of the reset, such as deleting votes while retaining voter and candidate accounts, resetting “has voted” statuses, and hiding results. The admin must enter their password to proceed, ensuring secure control over election management operations.

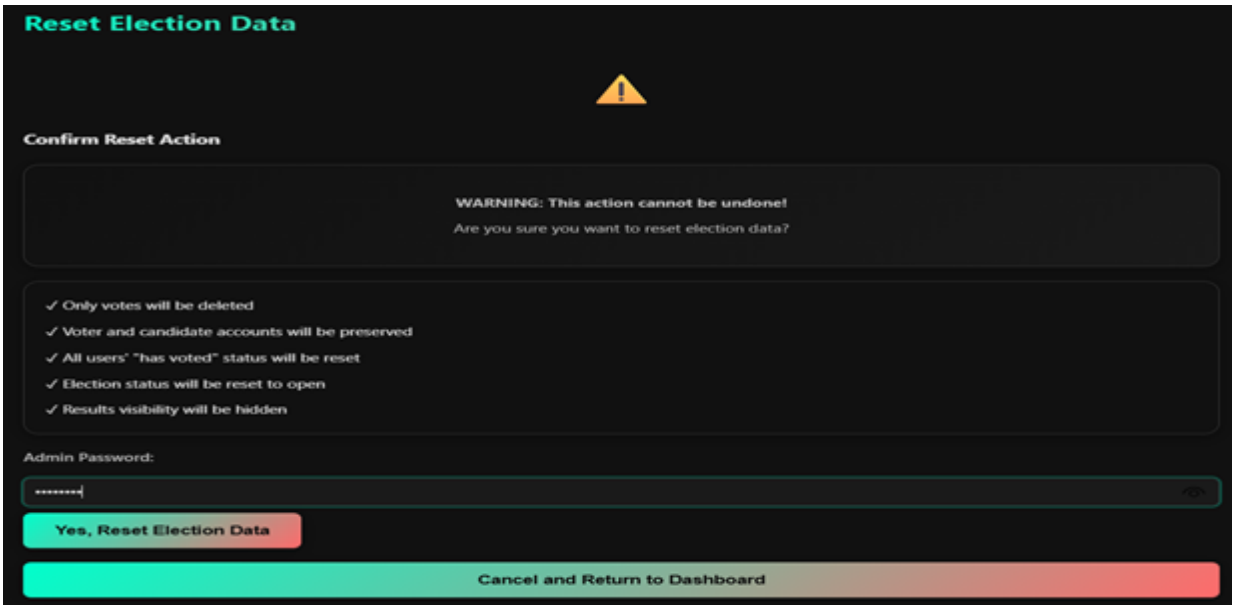

Figure 8.7 Reset Election Data

Figure 8.8 The figure illustrates the candidate selection page where voters can view all available candidates and choose their preferred option. Each candidate is listed along with associated symbols, and a “None of the Above (NOTA)” option is also provided. After selecting a candidate, voters proceed by clicking the Continue to Confirmation button. This interface ensures clarity and simplicity, allowing voters to make informed choices.

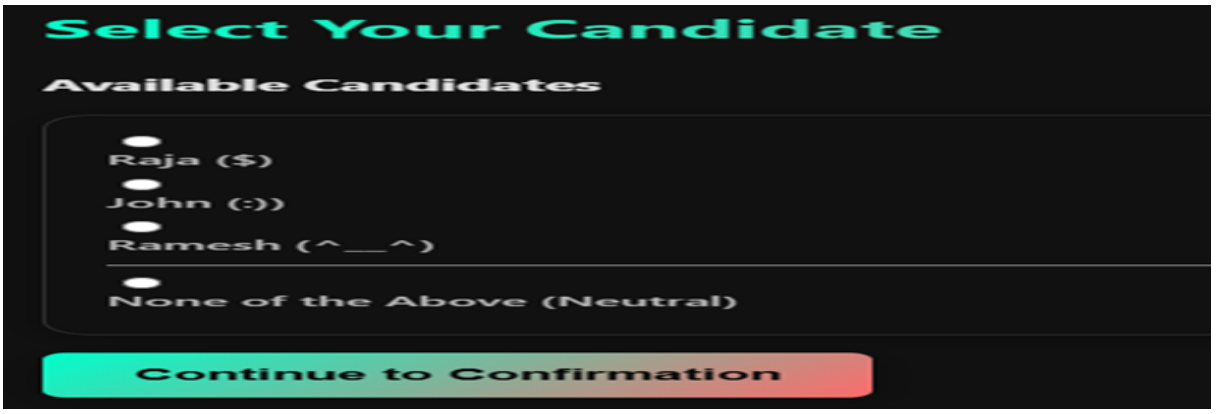

Figure 8.8 Select Your Candidate

Figure 8.9 This figure displays the Help & Support section of the voting system, which provides frequently asked questions for user assistance. It includes common queries such as login instructions, password recovery methods, vote modification policies, and information about vote security. This page enhances user experience by offering easy access to essential guidance, reducing confusion during the voting process.

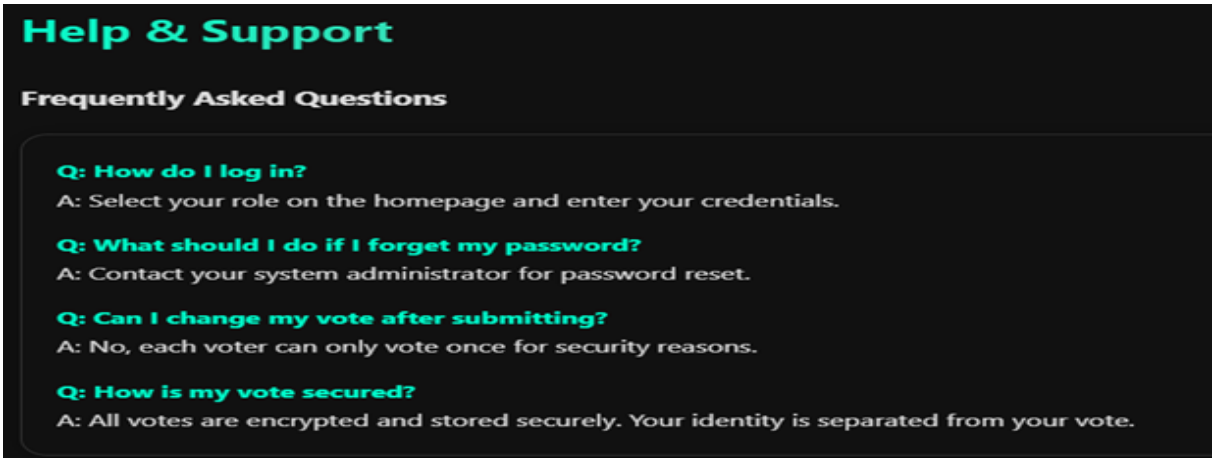

Figure 8.9 Help & Support

8.2 Discussion

This project demonstrates a highly secure, web-based voting system leveraging robust cryptography and user-friendly design. Multi-layered security is achieved with CAPTCHA to block bots, SHA-256 hashed admin authentication, and strict session management to ensure unique votes per user. Strong password policies and server-side input validation safeguard data integrity, while dynamic bar charts enhance transparency in result presentation. The system's modular architecture easily adapts to diverse election contexts. Confirmation steps prevent accidental voting, and receipt generation supports ballot auditing without sacrificing voter anonymity. Overall, the application enables accessible, transparent, and accountable online elections through careful system engineering and proven electronic voting best practices.

Chapter 9

Conclusion

9.1 Conclusion

While the current system provides a secure and efficient platform for small and medium-scale elections, future enhancements could significantly broaden its applicability and robustness across diverse electoral scenarios.

Integrating scalable database solutions like PostgreSQL clustering or MongoDB sharding will ensure persistent and reliable storage of votes and credentials, supporting electorates numbering in the millions during peak voting periods. Advanced encryption techniques including homomorphic encryption and blockchain technology can offer tamper-proof, transparent, and decentralized vote records, enabling public auditability without compromising voter privacy. Introducing multi-factor authentication (MFA) with biometrics, SMS OTP, or national ID-based eKYC verification will strengthen user security against credential theft and impersonation attacks.

This shows the admin-only edit Election Data, which includes a warning message emphasizing that the action is irreversible. The outlines the effects of the reset, such as deleting votes while retaining voter and candidate accounts, resetting statuses, and hiding results.

9.2 Scope for Future Work

Features like multi-language support (i18n), progressive web app (PWA) architecture for offline capability, and responsive design across mobile/desktop platforms will improve accessibility for diverse audiences including rural voters and international deployments. Real-time dashboards with WebSocket integration for live participation tracking, AI-powered anomaly detection for fraud prevention, and API integrations with government voter registries will enable seamless scalability to nationwide elections. Finally, quantum-resistant cryptography and federated learning for privacy-preserving model updates position the platform for long-term resilience against evolving computational threats.

References

[1] Adida, B. (2008). "Helios: Web-Based Open-Audit Voting." Proceedings of the 17th USENIX Security Symposium, 335-348.

[2] Rivest, R. L., Smith, W. D. (2007). "Three Voting Protocols: ThreeBallot, VAV, and Twin." Proceedings of the 2007 Electronic Voting Technology Workshop.

[3] Von Ahn, L., Blum, M., Hopper, N. J., Langford, J. (2003). "CAPTCHA: Using Hard AI Problems for Security." Advances in Cryptology, 294-311.

[4] Bonneau, J., Herley, C., Van Oorschot, P. C., Stajano, F. (2012). "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes." IEEE Symposium on Security and Privacy, 553-567.

[5] Bertino, E., Sandhu, R. (2005). "Database Security - Concepts, Approaches, and Challenges." IEEE Transactions on Dependable and Secure Computing, 2(1), 2-19.

[6] Chaum, D., Ryan, P. Y. A., Schneider, S. (2005). "A Practical Voter-Verifiable Election Scheme." Proceedings of the 10th European Symposium on Research in Computer Security, 118-139.

Personal Profile

Name: Mrs. Leema Rachel Mathew

Department: Information Science and Engineering

Designation: Assistant Professor

Qualification: MTech CSE

Skills: C, Java, Selenium, SQL, Multisim, Verilog, Excel, MS Word, PowerPoint

Experience: Assistant Professor in the department of ISE, SDM Institute if Technology from August 2025.

Name: Deepak V S

USN: 4SU23IS010

Adress: Bhadravathi, Shimoga -577301

E -mail ID: 23c10@sdmit.in

Contact Number: 6361194779

Name: Prem T J

USN: 4SU23IS032

Adress: Hospete, -583201

E -mail ID: 23c32@sdmit.in

Contact Number:8431061710

Name: Deepak V S

USN:4SU23IS060

Adress: Hosanagara, Shimoga -577425

E -mail ID: 23c61@sdmit.in

Contact Number:8277490104

Name: Vivek

USN:4SU23IS061

Adress: Hosanagara, Shimoga -577425

E -mail ID:23c62@sdmit.in

Contact Number:7760803592