PROJECT REPORT

TRUEVOTE

E VOTING MACHINE

MAY 2025

ELECTRONICS AND COMMUNICATION ENGINEERING

ACKNOWLEDGEMENT

We would like to place on record our sincere gratitude and heartfelt appreciation for the continuous encouragement, invaluable supervision, timely suggestions, and inspired guidance provided by our Project Advisor, Ms. Anju V. Sathyan, OOPS Faculty, Cochin University of Science and Technology (CUSAT). Her expert guidance and persistent support were instrumental in the successful completion of this project.

Team Pentagon

Aaromal A

Abhijith R S

Deeraj P Menon

Hithek Siva

Navaneeth N

ABSTRACT

Objectives:

TrueVote is a secure, user-friendly electronic voting application built with C++/Qt and cryptographic primitives to modernize and safeguard small-scale elections.
It emphasizes integrity, privacy, and auditability by combining SHA-256 password hashing, file-based data storage, and a simple graphical user interface.

System Architecture:

1. Presentation Layer (Qt GUI)
   - Main Menu: Entry point offering "Admin Login," "Vote," and "Exit" options.
2. Admin Workflow:
   - Authentication: Static credentials ("admin"/"admin123") gate access to the Admin Panel.
   - Admin Panel Features:
     - Register Voter: Prompt for Voter ID, name, and password; stores SHA-256 hash in data/voters.csv.
     - Register Candidate: Prompt for Candidate ID and name; appends to data/candidates.csv.
     - Start/Close Poll: Toggles voting availability via an in-memory flag.
     - View Results: Tallies vote from data/votes.log, displays counts per candidate, and writes summary to data/result.txt.
     - Clear Logs: Empties vote and result files, resetting the system.

Voter Workflow:

- Authentication: Verifies Voter ID and SHA-256 hashed password against voters.csv. Rejects repeat voting by scanning votes.log.
- Casting Vote: Presents a dropdown of registered candidates; records "VoterID,CandidateID,Timestamp" in votes.log.

Data Layer (File System):

- voters.csv — CSV stores VoterID,Name,SHA256(Password)
- candidates.csv — stores CandidateID,Name
- votes.log — sequential entries of VoterID,CandidateID,YYYY-MM-DD hh:mm:ss
- result.txt — human-readable tally of final vote counts

Security & Integrity:

- Password Hashing: All voter passwords are hashed with SHA-256 before storage or comparison.
- Single-Vote Enforcement: The system scans votes.log on each login to prevent duplicate voting.
- Immutable Audit Trail: Vote entries are append-only; admins cannot retroactively modify cast votes.

Usability & Deployment:

- Cross-platform Qt application with minimal dependencies.
- Intuitive dialogs (QInputDialog, QMessageBox) guide users through operations.
- Data folder (/data) auto-created on first run; files human-readable for audits.

INDEX

INTRODUCTION

The purpose of this digital voting platform is to streamline voter registration, candidate management, and polling operations with maximum efficiency. Its primary objective is to provide a secure, eco-friendly, and user-friendly voting experience that ensures accuracy, transparency, and trust. Unlike traditional electronic voting machines (EVMs), which require physical infrastructure and are costly to maintain and audit, this platform leverages digital technology to simplify election processes while offering real-time monitoring and easy verification. It is designed to modernize voting, making it more accessible and adaptable for a wide range of institutions and communities.

Why is this relevant?

- Growing need for remote, contactless voting in emergencies (e.g., pandemics).
- Rising awareness about digital trust and transparency in governance.
- Paper-based systems are time-consuming, error-prone, and expensive.
- Governments worldwide are experimenting with digital voting pilots.

Why is not evms?

- EVMs (Electronic Voting Machines) are hardware-limited:
- Require physical presence at polling stations.
- Expensive to manufacture, transport, and store.
- No easy integration with identity verification or secure digital records.
- Hard to audit EVM results in real-time without physical access.
- Limited flexibility for low-budget or small-scale elections (e.g., schools, societies).

Key Features:

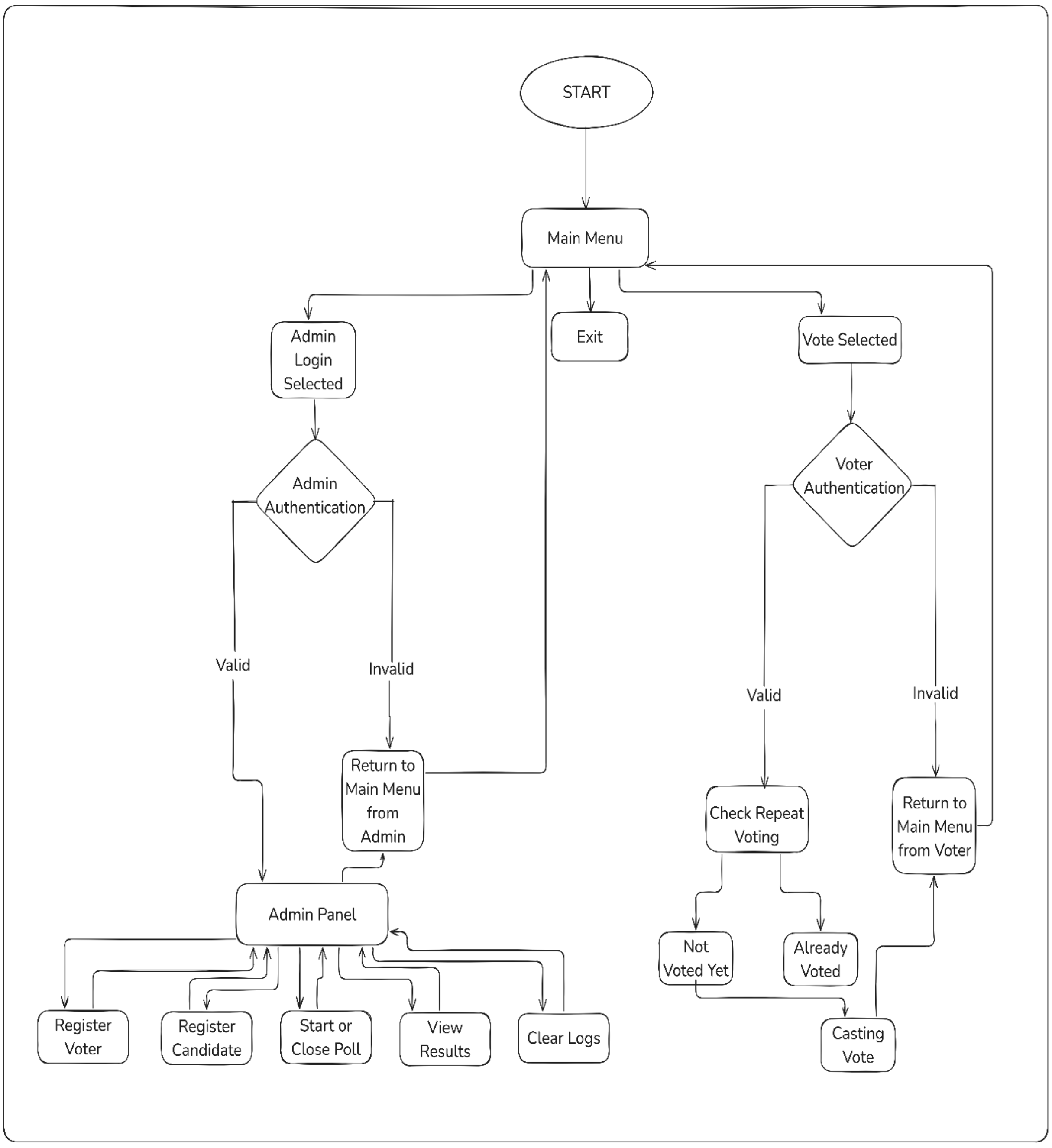
- Admin Dashboard – Full control over voter lists, candidate profiles, and election settings.
- Secure Authentication – Multi-layered login verification to prevent unauthorized access, including storing hashed passwords using SHA-256 hashing with salting for enhanced security.
- Live Voting & Instant Results – Real-time ballot casting and automated tallying for quick outcomes.
- Advanced Data Protection – End-to-end encryption and strict privacy protocols to safeguard sensitive information.

System Design

Algorithm:

```
START
  1. Display Start screen and navigate to Main Menu:
     [Admin Login] [Vote] [Exit]
  2. If Admin Login is selected (admin authentication):
     ○ Prompt for credentials and hash password.
     ○ Validate against stored hash.
     ○ If valid, show Admin Panel. Else, return to Main Menu.
  3. Admin Panel Options:
     ○ Register Voter: Input voter details, hash password, and
     store in voters.csv.
     ○ Register Candidate: Input and store candidate data in
     candidates.csv.
     ○ Start/Close Poll: Update poll status flag.
     ○ View Results: Tally votes from votes.csv and display.
     ○ Clear Logs: Wipe relevant log files.
     ○ Return to Main Menu when done.
  4. If Vote is selected (voter authentication & voting):
     ○ Prompt for Voter ID and password, hash and validate
     using voters.csv.
     ○ If valid, check if vote already cast in votes.csv.
     ○ If not voted yet, display candidates from candidates.csv
     and save vote.
     ○ Else, deny repeat voting and return to Main Menu.
  5. If Exit selected, terminate the application.
END
```

Flowchart:


START
Main Menu
Admin Login Selected
Exit
Vote Selected
Admin Authentication
Voter Authentication
Valid
Invalid
Valid
Invalid
Return to Main Menu from Admin
Check Repeat Voting
Return to Main Menu from Voter
Admin Panel
Not Voted Yet
Already Voted
Register Voter
Register Candidate
Start or Close Poll
View Results
Clear Logs
Casting Vote

TECHNOLOGY DESCRIPTION

1. C++:

   C++ is a powerful, high-performance programming language widely used for system and application development. In this project, C++ serves as the backbone of the application, handling the core logic, data processing, and security operations efficiently. Its speed and reliability make it ideal for building a robust voting system where performance and stability are critical.

2. Qt GUI:

   Qt is a popular cross-platform framework for developing graphical user interfaces (GUIs). We used Qt to create an intuitive and visually appealing interface. Qt's flexibility allows the application to run smoothly on different operating systems (such as Windows, macOS, and Linux) without significant changes to the codebase.

3. SHA-256 Hashing Algorithm:

   SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that generates a unique, fixed-size 256-bit (32-byte) hash for any given input. We implemented SHA-256 to secure the voting data, ensuring that each vote is hashed and stored in a tamper-proof way. This algorithm plays a crucial role in maintaining data integrity and trust: once a vote is cast and hashed, it cannot be altered without detection.

4. CSV File Management:

   We incorporated CSV (Comma-Separated Values) file management to handle data storage and reporting efficiently. All critical data—such as voter lists, candidate details, and voting logs—are stored in CSV files.

OOPS CONCEPTS INVOLVED

1. Inheritance
   - Utilized to establish hierarchical relationships between classes (e.g., User as a base class for Voter, Candidate, and Admin).
2. Polymorphism
   - Function Overloading
   - Method Overriding
   - Operator Overloading
3. Abstract Classes
   - Designed a base User class with pure virtual functions.
4. Friend Functions
   - Securely granted external functions/classes access to private members.
5. Constructors & Destructors
   - Manage object lifecycle.
6. File Handling
   - Store and retrieve data efficiently.
7. Vectors (Dynamic Storage)
   - Scalable containers for managing variable-sized datasets.

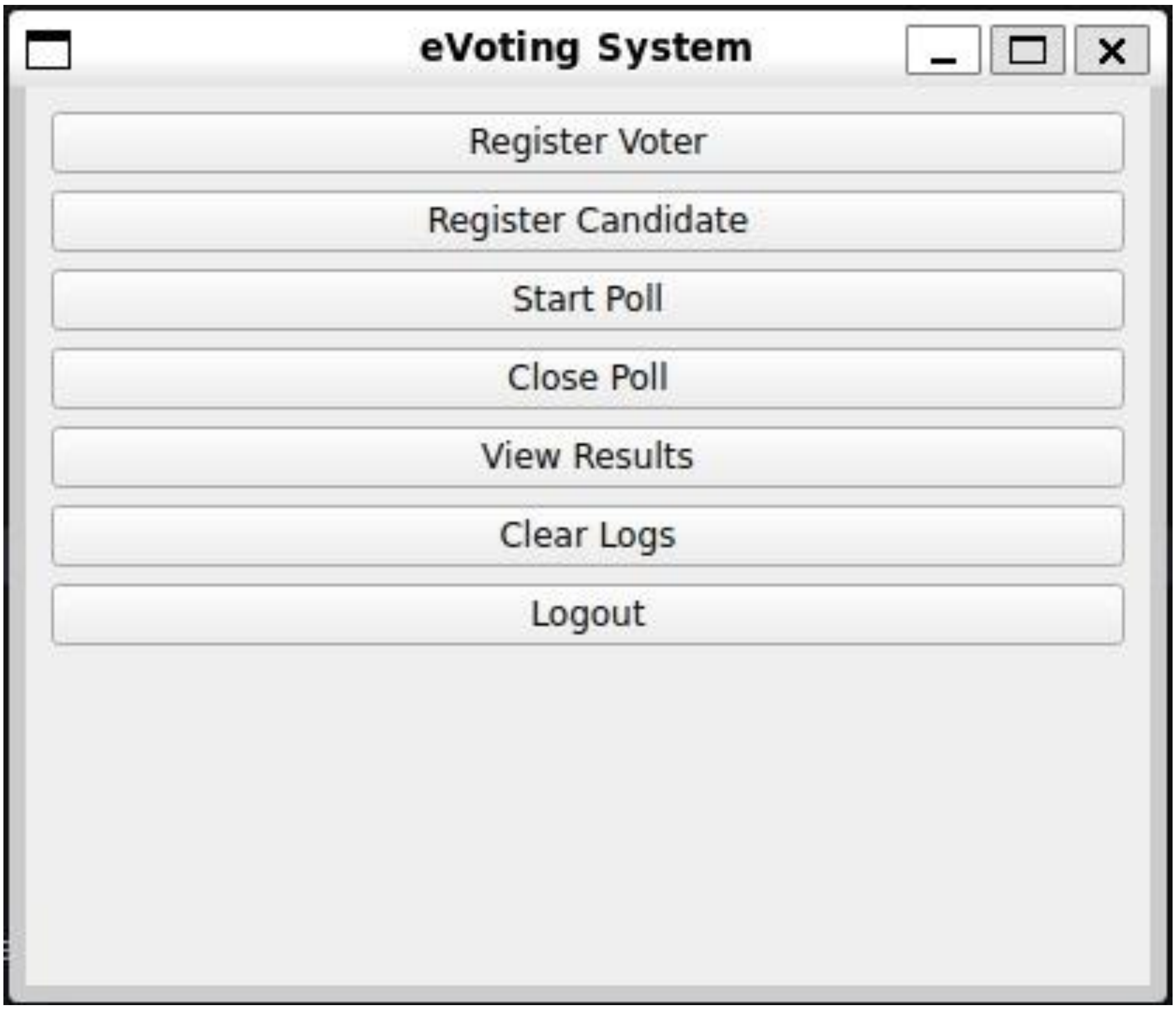
CODE

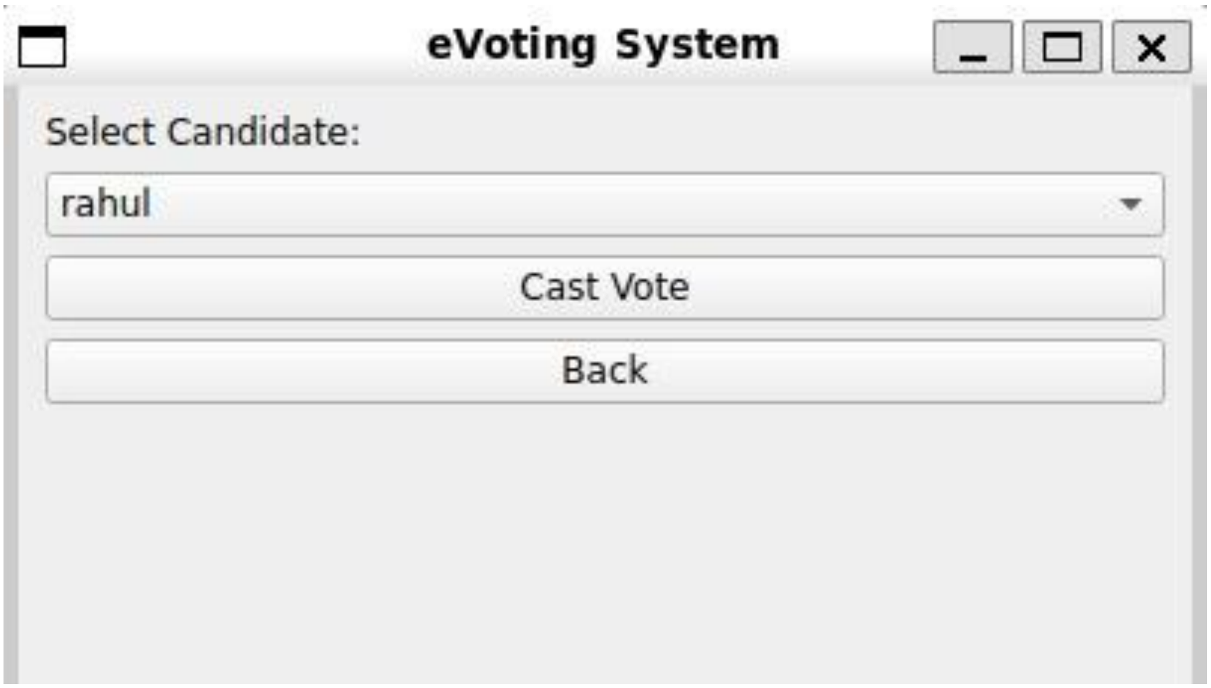
```cpp
//main.cpp
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}
//mainwindow.cpp
#include "mainwindow.h"
#include <QApplication>
#include <QGridLayout>
#include <QFile>
#include <QTextStream>
#include <QDir>
#include <QInputDialog>
#include <QDateTime>
#include <QMap>
#include <QMessageBox>
#include <string>
#include <sstream>
#include <iomanip>
#include <openssl/sha.h>
// SHA256 implementation
inline std::string sha256(const std::string &str) {
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256((unsigned char*)str.c_str(), str.size(), hash);
    std::stringstream ss;
    for (int i = 0; i < SHA256_DIGEST_LENGTH; ++i)
        ss << std::hex << std::setw(2) << std::setfill('0') << (int)hash[i];
    return ss.str();}
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), pollOpen(false) {
    // Create data directory if it doesn't exist
    QDir().mkpath("./data");
    stackedWidget = new QStackedWidget(this);
    setCentralWidget(stackedWidget);
    // Create all pages
    stackedWidget->addWidget(createMainMenu());
    stackedWidget->addWidget(createAdminLogin());
    stackedWidget->addWidget(createVoterLogin());
    stackedWidget->addWidget(createAdminPanel());
    stackedWidget->addWidget(createVotingPage());
    showMainMenu();
    setWindowTitle("eVoting System");
    resize(400, 300);
}
QWidget* MainWindow::createMainMenu() {
    QWidget* widget = new QWidget;
    QVBoxLayout* layout = new QVBoxLayout;
    QPushButton* adminButton = new QPushButton("Admin Login");
    QPushButton* voterButton = new QPushButton("Vote");
    QPushButton* exitButton = new QPushButton("Exit");
    connect(adminButton, &QPushButton::clicked, this, &MainWindow::showAdminLogin);
    connect(voterButton, &QPushButton::clicked, this, &MainWindow::showVoterLogin);
    connect(exitButton, &QPushButton::clicked, qApp, &QApplication::quit);
    layout->addWidget(new QLabel("Welcome to eVoting System"));
    layout->addWidget(adminButton);
    layout->addWidget(voterButton);
    layout->addWidget(exitButton);
    layout->addStretch();
    widget->setLayout(layout);
    return widget;
}
QWidget* MainWindow::createAdminLogin() {
    QWidget* widget = new QWidget;
    QVBoxLayout* layout = new QVBoxLayout;
    adminUserEdit = new QLineEdit;
    adminPassEdit = new QLineEdit;
    adminPassEdit->setEchoMode(QLineEdit::Password);
    QPushButton* loginButton = new QPushButton("Login");
    QPushButton* backButton = new QPushButton("Back");
    connect(loginButton, &QPushButton::clicked, this, &MainWindow::handleAdminLogin);
    connect(backButton, &QPushButton::clicked, this, &MainWindow::showMainMenu);
    layout->addWidget(new QLabel("Username:"));
    layout->addWidget(adminUserEdit);
    layout->addWidget(new QLabel("Password:"));
    layout->addWidget(adminPassEdit);
    layout->addWidget(loginButton);
    layout->addWidget(backButton);
    layout->addStretch();
    widget->setLayout(layout);
    return widget;
}
QWidget* MainWindow::createVoterLogin() {
    QWidget* widget = new QWidget;
    QVBoxLayout* layout = new QVBoxLayout;
    voterIdEdit = new QLineEdit;
    voterPassEdit = new QLineEdit;
    voterPassEdit->setEchoMode(QLineEdit::Password);
    QPushButton* loginButton = new QPushButton("Login");
    QPushButton* backButton = new QPushButton("Back");
    connect(loginButton, &QPushButton::clicked, this, &MainWindow::handleVoterLogin);
    connect(backButton, &QPushButton::clicked, this, &MainWindow::showMainMenu);
    layout->addWidget(new QLabel("Voter ID:"));
    layout->addWidget(voterIdEdit);
    layout->addWidget(new QLabel("Password:"));
    layout->addWidget(voterPassEdit);
    layout->addWidget(loginButton);
    layout->addWidget(backButton);
    layout->addStretch();
    widget->setLayout(layout);
    return widget;
}

QWidget* MainWindow::createAdminPanel() {
    QWidget* widget = new QWidget;
    QVBoxLayout* layout = new QVBoxLayout;
    QPushButton* registerVoterBtn = new QPushButton("Register Voter");
    QPushButton* registerCandidateBtn = new QPushButton("Register Candidate");
    QPushButton* startPollBtn = new QPushButton("Start Poll");
    QPushButton* closePollBtn = new QPushButton("Close Poll");
    QPushButton* viewResultsBtn = new QPushButton("View Results");
    QPushButton* clearLogsBtn = new QPushButton("Clear Logs");
    QPushButton* logoutBtn = new QPushButton("Logout");
    connect(registerVoterBtn, &QPushButton::clicked, this, &MainWindow::registerVoter);
    connect(registerCandidateBtn, &QPushButton::clicked, this,
&MainWindow::registerCandidate);
    connect(startPollBtn, &QPushButton::clicked, this, &MainWindow::startPoll);
    connect(closePollBtn, &QPushButton::clicked, this, &MainWindow::closePoll);
    connect(viewResultsBtn, &QPushButton::clicked, this, &MainWindow::viewResults);
    connect(clearLogsBtn, &QPushButton::clicked, this, &MainWindow::clearLogs);
    connect(logoutBtn, &QPushButton::clicked, this, &MainWindow::logout);
    layout->addWidget(registerVoterBtn);
    layout->addWidget(registerCandidateBtn);
    layout->addWidget(startPollBtn);
    layout->addWidget(closePollBtn);
    layout->addWidget(viewResultsBtn);
    layout->addWidget(clearLogsBtn);
    layout->addWidget(logoutBtn);
    layout->addStretch();
    widget->setLayout(layout);
    return widget;
}
QWidget* MainWindow::createVotingPage() {
    QWidget* widget = new QWidget;
    QVBoxLayout* layout = new QVBoxLayout;
    candidateCombo = new QComboBox;
    QPushButton* voteButton = new QPushButton("Cast Vote");
    QPushButton* backButton = new QPushButton("Back");
    connect(voteButton, &QPushButton::clicked, this, &MainWindow::handleVote);
    connect(backButton, &QPushButton::clicked, this, &MainWindow::showMainMenu);
    layout->addWidget(new QLabel("Select Candidate:"));
    layout->addWidget(candidateCombo);
    layout->addWidget(voteButton);
    layout->addWidget(backButton);
    layout->addStretch();
    widget->setLayout(layout);
    return widget;
}
void MainWindow::showMainMenu() {
    stackedWidget->setCurrentIndex(0);
}
void MainWindow::showAdminLogin() {
    adminUserEdit->clear();
    adminPassEdit->clear();
    stackedWidget->setCurrentIndex(1);
}
void MainWindow::showVoterLogin() {
    if (!pollOpen) {
        QMessageBox::warning(this, "Error", "Poll is not open.");
        return;
    }
    voterIdEdit->clear();
    voterPassEdit->clear();
    stackedWidget->setCurrentIndex(2);
}
void MainWindow::handleAdminLogin() {
    QString username = adminUserEdit->text();
    QString password = adminPassEdit->text();
    if (username == "admin" && password == "admin123") {
        stackedWidget->setCurrentIndex(3);  // Show admin panel
    } else {
        QMessageBox::warning(this, "Error", "Invalid credentials");
    }
}
void MainWindow::handleVoterLogin() {
    QString id = voterIdEdit->text();
    QString password = voterPassEdit->text();
    QFile votersFile("./data/voters.csv");
    if (!votersFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QMessageBox::warning(this, "Error", "Could not open voters file");
        return;
    }
    QTextStream in(&votersFile);
    bool found = false;
    while (!in.atEnd()) {
        QString line = in.readLine();
        QStringList fields = line.split(",");
        if (fields.size() < 3) continue;
        if (fields[0] == id && fields[2] ==
QString::fromStdString(sha256(password.toStdString()))) {
```

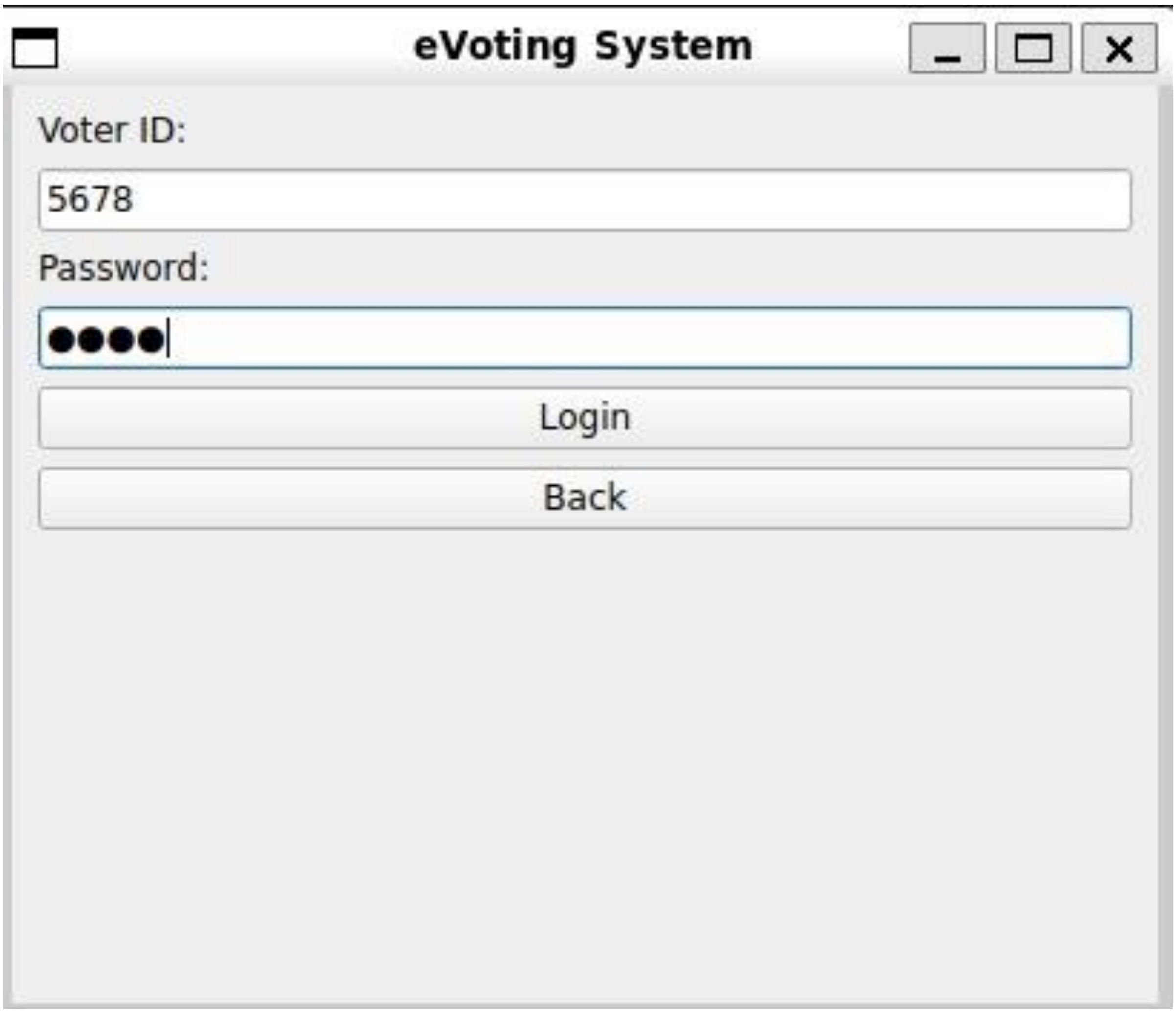
```
            // Check if already voted
            QFile logFile("./data/votes.log");
            if (logFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
                QTextStream logStream(&logFile);
                bool hasVoted = false;
                while (!logStream.atEnd()) {
                    QString logLine = logStream.readLine();
                    if (logLine.startsWith(id + ",")) {
                        hasVoted = true;
                        break;
                    }
                }
                logFile.close();

                if (hasVoted) {
                    QMessageBox::warning(this, "Error", "You have already voted.");
                    votersFile.close();
                    return;
                }
            }
            found = true;
            break;
        }
    }
    votersFile.close();

    if (!found) {
        QMessageBox::warning(this, "Error", "Invalid credentials");
        return;
    }

    // Load candidates
    candidateCombo->clear();
    QFile candidatesFile("./data/candidates.csv");
    if (candidatesFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream candStream(&candidatesFile);
        while (!candStream.atEnd()) {
            QString line = candStream.readLine();
            QStringList fields = line.split(",");
            if (fields.size() >= 2) {
                candidateCombo->addItem(fields[1], fields[0]);
            }
        }
        candidatesFile.close();
    }

    stackedWidget->setCurrentIndex(4);  // Show voting page
}

void MainWindow::registerVoter() {
    bool ok;
    QString id = QInputDialog::getText(this, "Register Voter", "Enter Voter ID:",
QLineEdit::Normal, "", &ok);
    if (!ok || id.isEmpty()) return;

    QString name = QInputDialog::getText(this, "Register Voter", "Enter Voter Name:",
QLineEdit::Normal, "", &ok);
    if (!ok || name.isEmpty()) return;

    QString password = QInputDialog::getText(this, "Register Voter", "Set Password:",
QLineEdit::Password, "", &ok);
    if (!ok || password.isEmpty()) return;

    QFile file("./data/voters.csv");
    if (file.open(QIODevice::Append | QIODevice::Text)) {
        QTextStream out(&file);
        out << id << "," << name << "," <<
QString::fromStdString(sha256(password.toStdString())) << "\n";
        file.close();
        QMessageBox::information(this, "Success", "Voter registered successfully");
    }
}

void MainWindow::registerCandidate() {
    bool ok;
    QString id = QInputDialog::getText(this, "Register Candidate", "Enter Candidate ID:",
QLineEdit::Normal, "", &ok);
    if (!ok || id.isEmpty()) return;

    QString name = QInputDialog::getText(this, "Register Candidate", "Enter Candidate
Name:", QLineEdit::Normal, "", &ok);
    if (!ok || name.isEmpty()) return;

    QFile file("./data/candidates.csv");
    if (file.open(QIODevice::Append | QIODevice::Text)) {
        QTextStream out(&file);
        out << id << "," << name << "\n";
        file.close();
        QMessageBox::information(this, "Success", "Candidate registered successfully");
    }
}

void MainWindow::startPoll() {
    if (pollOpen) {
        QMessageBox::warning(this, "Error", "Poll is already open");
        return;
    }
    pollOpen = true;
    QMessageBox::information(this, "Success", "Poll started successfully");
}

void MainWindow::closePoll() {
    if (!pollOpen) {
        QMessageBox::warning(this, "Error", "Poll is not currently open");
        return;
    }
    pollOpen = false;
    QMessageBox::information(this, "Success", "Poll closed successfully");
}

void MainWindow::viewResults() {
    if (pollOpen) {
        QMessageBox::warning(this, "Error", "Cannot view results while poll is open");
        return;
    }

    QMap<QString, QString> candidates;
    QMap<QString, int> voteCount;

    // Load candidates
    QFile candidatesFile("./data/candidates.csv");
    if (candidatesFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream in(&candidatesFile);
        while (!in.atEnd()) {
            QString line = in.readLine();
            QStringList fields = line.split(",");
            if (fields.size() >= 2) {
                candidates[fields[0]] = fields[1];
                voteCount[fields[0]] = 0;
            }
        }
        candidatesFile.close();
    }

    // Count votes
    QFile logFile("./data/votes.log");
    if (logFile.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream in(&logFile);
        while (!in.atEnd()) {
            QString line = in.readLine();
            QStringList fields = line.split(",");
            if (fields.size() >= 2) {
                QString candidateId = fields[1];
                voteCount[candidateId]++;
            }
        }
        logFile.close();
    }

    // Display results
    QString results = "Election Results:\n\n";
    QMapIterator<QString, QString> i(candidates);
    while (i.hasNext()) {
        i.next();
        results += i.value() + ": " + QString::number(voteCount[i.key()]) + " votes\n";
    }

    // Save to file
    QFile resultFile("./data/result.txt");
    if (resultFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&resultFile);
        out << "Election Results\n================\n\n" << results;
        resultFile.close();
    }

    QMessageBox::information(this, "Results", results);
}

void MainWindow::clearLogs() {
    QFile logFile("./data/votes.log");
    if (logFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
        logFile.close();
    }

    QFile resultFile("./data/result.txt");
    if (resultFile.exists()) {
        resultFile.remove();
    }

    QMessageBox::information(this, "Success", "Logs cleared successfully");
}

void MainWindow::handleVote() {
    if (!pollOpen) {
        QMessageBox::warning(this, "Error", "Poll is not open");
        return;
    }

    QString candidateId = candidateCombo->currentData().toString();
    QString voterId = voterIdEdit->text();

    QFile file("./data/votes.log");
    if (file.open(QIODevice::Append | QIODevice::Text)) {
        QTextStream out(&file);
        out << voterId << "," << candidateId << ","
```

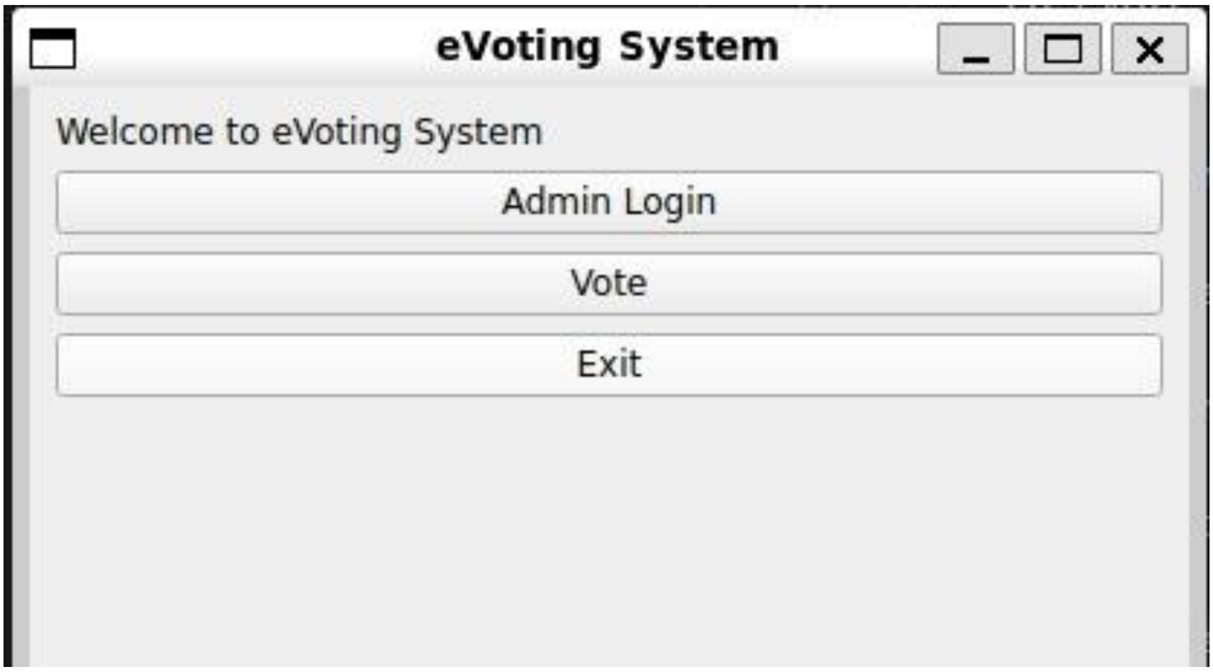
```
            << QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss") << "\n";
        file.close();
    }

    QMessageBox::information(this, "Success", "Vote cast successfully");
    showMainMenu();
}

void MainWindow::logout() {
    showMainMenu();
}
//mainwindow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QStackedWidget>
#include <QPushButton>
#include <QLineEdit>
#include <QLabel>
#include <QVBoxLayout>
#include <QMessageBox>
#include <QComboBox>

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);

private slots:
    void showAdminLogin();
    void showVoterLogin();
    void handleAdminLogin();
    void handleVoterLogin();
    void showMainMenu();
    void registerVoter();
    void registerCandidate();
    void startPoll();
    void closePoll();
    void viewResults();
    void clearLogs();
    void handleVote();
    void logout();

private:
    QStackedWidget *stackedWidget;
    QWidget *createMainMenu();
    QWidget *createAdminLogin();
    QWidget *createVoterLogin();
    QWidget *createAdminPanel();
    QWidget *createVotingPage();

    QLineEdit *adminUserEdit;
    QLineEdit *adminPassEdit;
    QLineEdit *voterIdEdit;
    QLineEdit *voterPassEdit;
    QComboBox *candidateCombo;
    bool pollOpen;
};

#endif
```

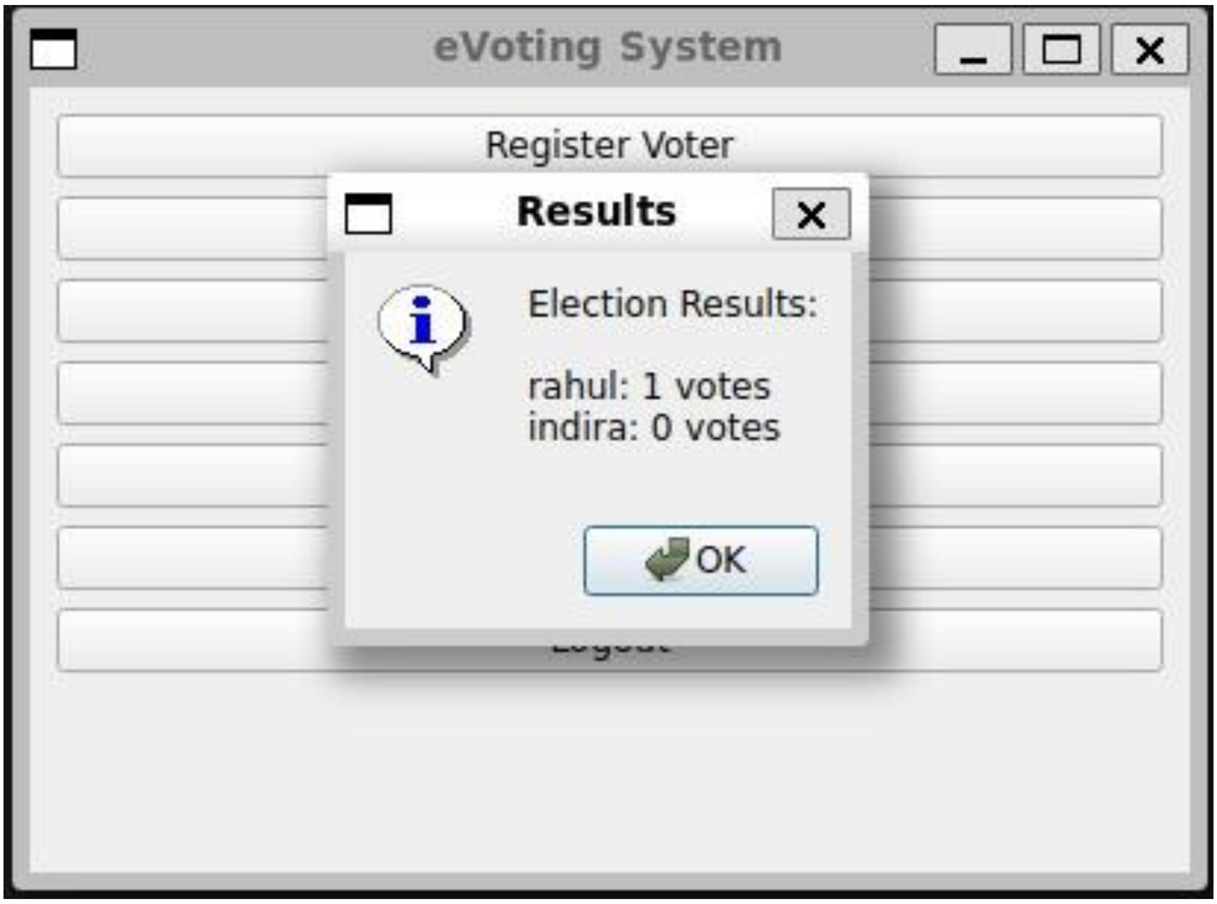
SNAPSHOTS


eVoting System
Register Voter
Register Candidate
Start Poll
Close Poll
View Results
Clear Logs
Logout


eVoting System
Select Candidate:
rahul
Cast Vote
Back


eVoting System
Voter ID:
5678
Password:
●●●●
Login
Back





Election Results:

rahul: 1 votes
indira: 0 votes

CONCLUSION

TrueVote delivers a lightweight, yet robust, e-voting solution ideal for small institutions, colleges, clubs, or student elections.

Built with C++/Qt, it ensures independence, cryptographic hashing, and clear UI, it empowers fair and secure voting.

Admins can easily register voters and candidates, control poll state, and obtain verifiable results; voters enjoy straightforward authentication and casting processes.

Future enhancements may include database integration, digital signatures, and public verification portals.