VOTING SYSTEM USING FILE HANDLING

Project submitted to the

SRM University – AP, Andhra Pradesh

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

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

G . POORNA NAGA SAI

AP23110010321

G. sri teja

AP23110010265

B. SIDDARTHA

AP23110010314

M.LEELA VARDHAN

AP23110010303

Under the Guidance

Mrs.Kavitha Rani Karnena

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[November, 2024]

Certificate

Date: 16-Nov-22

This is to certify that the work present in this Project entitled "**VOTING SYSTEM USING FILE HANDLING**" has been carried out by **Poorna, sriteja, Siddartha, leela vardhan** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Prof. / Dr. **Kavitha Rani Karnena**

Acknowledgements

This project **"Voting System Using File Handling"** would not have been possible without the guidance and support of Mrs. Kavita rani karnena Much love to my fellow teammates for their collaboration, and a special thanks to my family for always cheering me on. This has been an eye-opening and rewarding experience for me and I want to thank all of you who made this happen.

Their guidance and encouragement have been pivotal in overcoming challenges and achieving the objectives of this project. I also acknowledge the constructive feedback and suggestions provided during various stages, which helped enhance the quality of the work. This project has been a rewarding learning experience, and I am truly appreciative of the support and collaboration from everyone involved.

Table of Contents

Abstract :

This study delves into creating a voting system, in C++ that utilizes file processing to handle data without relying on a database infrastructure. The project aims to explore the use of file handling methods in developing an well structured voting application with features including voter registration management tools and candidate directory setup for casting votes and tally calculations.

Every aspect of the system is managed through file manipulations whereby voter details and candidate profiles are stored separately in files, for organization and convenient retrieval purposes. The research also looks into ways to confirm data accuracy and protect against voting to maintain the integrity of the voting procedure effectively.Progress made by the study includes highlighting the advantages and drawbacks of file management, in scenarios involving data usage.The study suggests that this method works well for voting setups but would require scalability and improved security measures, for implementations.

Abbreviations

1. ifstream - Input File Stream

2. ofstream - Output File Stream

3. vector - Dynamic Array Container

4.leaders - Leader Vector

5. candidates - Candidate Vector

6. voterName - Voter Name Variable

7. role - Role of Person

8. voteCount - Vote Count Map

9. leaderName - Leader Name Variable

10. map - Associative Array

Introduction

Designing an efficient and user-friendly voting system is crucial for facilitating a seamless democratic process. This presentation will provide an overview of a C++-based voting system, highlighting its key objectives, features, and a step-by-step guide on how to utilize its functionality.

Voting systems play a crucial role in democratic decision-making, requiring reliability, security, and user-friendliness to ensure fair outcomes. This project outlines the creation of a voting system using file handling in C++, specifically designed for situations where traditional databases might not be feasible. By utilizing file handling, the system effectively manages essential functions—voter registration, candidate listing, vote recording, and result calculation—entirely through structured files, providing a lightweight yet effective solution for small-scale voting needs.

The system emphasizes simplicity and ease of use, allowing users to cast their votes and check results effortlessly while ensuring that each vote is unique and accurately recorded. File handling operations, such as reading, writing, and updating files, help maintain data integrity and enable persistent storage without relying on a database. Important security measures, including voter authentication and data validation, are put in place to protect the integrity of the voting process and prevent duplication or unauthorized access

Objective:

1.Accessibility

Enabling voters to easily access and participate in the electoral process from any device, anywhere.

2 .Security

Ensuring the integrity of the voting system through robust security measures and data encryption.

3.Transparency

Providing voters with clear and comprehensive information about the voting process and election results.

Key Features and Functionality :

1.Voter Registration

A secure and user-friendly process that allows voters to register and update their information easily.

2.Ballot Selection

An efficient interface for voters to examine candidates, choose their preferences, and submit their ballots.

3.Results Tracking

A real-time display of election results, ensuring transparency and providing voters with the latest information.

Methodology

1. User details (name, ID, password).

Store user data in a file (e.g., users.txt).

2. Login Module:

Authenticate users from the stored file.

3. Voting Module:

Record votes in a separate file (e.g., votes.txt).

Ensure a user cannot vote multiple times.

4. Result Processing Module:

Read vote data and calculate results.

5. Admin Module (optional):

View results and manage elections.

File Handling:

Utilize file reading (r), writing (w), and appending (a) modes.

Data storage examples:

User data: Name, ID, Password

Vote data: UserID, CandidateID

---

4. Implementation Steps

Step 1: Create the registration and login functionality.

Write user credentials to a file.

Implement password validation for login.

Step 2: Develop voting functionality.

Allow authenticated users to vote.

Append the vote to a votes.txt file.

Step 3: Ensure data integrity.

Check for duplicate votes by storing user IDs of those who voted.

Step 4: Implement result tallying.

Read vote data and count votes for each candidate.

Step 5: Add file encryption (if necessary) for security.

---

5. Testing

Test individual modules:

Registration: Verify if user details are correct.

Step-by-Step Guide on How to Use the Voting system :

Register

Visit it and complete the voter registration process, providing the necessary personal and residency information.

Review Candidates

Explore the candidate profiles, their platforms, and make informed decisions about your selections.

Cast Your Ballot

Navigate to the voting section, review your choices, and securely submit your ballot.

Code Implementation :

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <map>
#include <set>
using namespace std;

class Person {
private:
   string name;
   string role;
public:
   Person(const string& name = "", const string& role = "") :
name(name), role(role) {}

   void registerPerson(const string& name, const string& role) {
      this->name = name;
      this->role = role;
      ofstream file("registration.txt", ios::app);
      if (file.is_open()) {
         file << name << " " << role << endl;
         file.close();
```

```
        }
        cout << name << " registered as a " << role << "." << endl;
    }

    string getName() const { return name; }
    string getRole() const { return role; }
};

void loadRegisteredUsers(vector<Person>& leaders,
vector<Person>& candidates) {
    ifstream file("registration.txt");
    if (!file.is_open()) {
        cout << "No registration data found." << endl;
        return;
    }

    string name, role;
    while (file >> name >> role) {
        Person person(name, role);
        if (role == "leader") {
            leaders.push_back(person);
        } else if (role == "candidate") {
            candidates.push_back(person);
        }
    }
    file.close();
```

```
}

bool isRegistered(const string& name) {
    ifstream file("registration.txt");
    string regName, regRole;
    while (file >> regName >> regRole) {
        if (regName == name) {
            return true;
        }
    }
    return false;
}

bool hasVoted(const string& name) {
    ifstream voteFile("votes.txt");
    string voterName, vote, leaderName;
    while (voteFile >> voterName >> vote >> leaderName) {
        if (voterName == name) {
            return true;
        }
    }
    return false;
}

void castVote(vector<Person>& leaders) {
```

```
string voterName, leaderName;
cin.ignore();
cout << "Enter your name to vote: ";
getline(cin, voterName);

if (!isRegistered(voterName)) {
   cout << "Your name is not registered to vote." << endl;
   return;
}


if (hasVoted(voterName)) {
   cout << "You have already cast your vote." << endl;
   return;
}


cout << "\nAvailable leaders to vote for:\n";
for (int i = 0; i < leaders.size(); i++) {
   cout << "- " << leaders[i].getName() << endl;
}

cout << "Enter the name of the leader you want to vote for: ";
getline(cin, leaderName);
```

```
    ofstream voteFile("votes.txt", ios::app);
    if (voteFile.is_open()) {
        voteFile << voterName << " voted for " << leaderName <<
endl;
        voteFile.close();
        cout << "Vote cast successfully for " << leaderName << "." <<
endl;
    } else {
        cout << "Unable to record vote." << endl;
    }
}

void declareResults() {
    ifstream voteFile("votes.txt");
    if (!voteFile.is_open()) {
        cout << "No votes recorded yet." << endl;
        return;
    }

    map<string, int> voteCount;
    string voterName, vote, leaderName;

    while (voteFile >> voterName >> vote >> vote >> leaderName)
{
        voteCount[leaderName]++;
    }
```

```
    voteFile.close();


    cout << "\n======= Election Results =======\n";
    int maxVotes = 0;



    for (map<string, int>::iterator it = voteCount.begin(); it !=
voteCount.end(); ++it) {
        cout << it->first << " received " << it->second << " votes." <<
endl;
        if (it->second > maxVotes) {
            maxVotes = it->second;
        }
    }



    vector<string> winners;
    for (map<string, int>::iterator it = voteCount.begin(); it !=
voteCount.end(); ++it) {
        if (it->second == maxVotes) {
            winners.push_back(it->first);
        }
    }


    if (!winners.empty()) {
        if (winners.size() == 1) {
```

```
            cout << "\nThe winner is " << winners[0] << " with " <<
maxVotes << " votes!" << endl;
        } else {
            cout << "\nIt's a tie! The winners with " << maxVotes << "
votes each are:\n";
            for (size_t i = 0; i < winners.size(); ++i) {
                cout << "- " << winners[i] << endl;
            }
        }
    } else {
        cout << "No votes cast." << endl;
    }
}
```

```
void registerUser(vector<Person>& leaders, vector<Person>&
candidates) {
    string name, role;
    cout << "Enter name to register: ";
    cin.ignore();
    getline(cin, name);

    cout << "Enter role (leader/candidate): ";
```

```
    getline(cin, role);

    Person person(name, role);
    person.registerPerson(name, role);

    if (role == "leader") {
        leaders.push_back(person);
    } else if (role == "candidate") {
        candidates.push_back(person);
    } else {
        cout << "Invalid role. Registration failed." << endl;
    }
}

int main() {
    vector<Person> leaders;
    vector<Person> candidates;



    loadRegisteredUsers(leaders, candidates);

    int choice;
    do {
        cout << "\n========== Election System
==========\n";
        cout << "1. Register\n";
```

```
        cout << "2. Cast Vote\n";
        cout << "3. Declare Results\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                registerUser(leaders, candidates);
                break;
            case 2:
                if (leaders.empty()) {
                    cout << "No leaders available to vote for. Register
leaders first." << endl;
                } else {
                    castVote(leaders);
                }
                break;
            case 3:
                declareResults();
                break;
            case 4:
                cout << "Exiting the election system..." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
```

```
        }
    } while (choice != 4);

    return 0;
}
```

output of the code:

1. **Option 1: Register**
    - Prompts the user to enter a name and role (`leader` or `candidate`).
    - Adds the registered person to the `registration.txt` file.
    - Expected Output:

```
Enter name to register: John
Enter role (leader/candidate): leader
John registered as a leader.
```

    - If an invalid role is entered:

```
Invalid role. Registration failed.
```

2. **Option 2: Cast Vote**
    - Asks the user to enter their name.
    - Checks if the name is registered.
        - If not registered:

```
Your name is not registered to vote.
```

        - If the user has already voted:

```
You have already cast your vote.
```

    - Displays available leaders to vote for and asks the user to select one.

```
Available leaders to vote for:
- Alice
- Bob
Enter the name of the leader you want to v
Vote cast successfully for Bob.
```

3. **Option 3: Declare Results**

- Reads votes from `votes.txt`, counts votes for each leader, and displays the results.
- Expected output if votes have been recorded:

```
======= Election Results =======
Alice received 3 votes.
Bob received 5 votes.

The winner is Bob with 5 votes!
```

- If it's a tie:

```
======= Election Results =======
Alice received 4 votes.
Bob received 4 votes.

It's a tie! The winners with 4 votes each
- Alice
- Bob
```

4. **Option 4: Exit**
    - Exits the system with the following output:

```
Exiting the election system...
```

summary and conclusion :

Empowering Voters:

=> The voting website empowers zens to actively participate in the nocratic process, making their voices heard.

Secure and Transparent

=>Advanced security measures and real-time result tracking ensure the integrity and transparency of the voting system.

Future work:

1. Improving System Security

User Authentication: Introduce secure authentication mechanisms like:

Multi-factor authentication.

Unique voter IDs and passwords.

OTP-based verification.

Data Encryption: Encrypt all stored data, including registration and voting details, to ensure data integrity and confidentiality.

Audit Trails: Log every action performed in the system to detect and prevent tampering.

2. Scalability and Performance

Database Integration: Migrate from file-based storage to a relational database (e.g., MySQL, PostgreSQL) or NoSQL database (e.g., MongoDB) to handle larger datasets efficiently.

Distributed Systems: Develop a distributed system for conducting elections in multiple locations simultaneously.

Load Testing: Test the system under heavy user loads to ensure performance doesn't degrade.

3. Advanced Voting Mechanisms

Ranked Choice Voting: Implement preferential voting systems like ranked-choice or single transferable votes.

Weighted Voting: Allow voting weights based on specific criteria (e.g., role or stakeholder importance).

Proxy Voting: Introduce a feature for voters to delegate their vote to a trusted individual.

4. Multi-Platform Support

Mobile App: Create a mobile version of the voting system for Android and iOS.

Web-Based System: Develop a web-based platform with responsive design for universal access.

Cross-Platform Synchronization: Ensure real-time data synchronization between platforms.

5. Election Management Features

Election Scheduling: Add functionality for creating and managing multiple elections with defined start and end times.

Automatic Result Declaration: Automate result generation and display at the end of the election period.

Candidate Management: Allow administrators to manage candidate profiles and manifestos.

Reference :

1. "C++Primer" (5th Edition, 2012) by Stanley B. Lippman et al. ○ Covers C++ programming fundamentals, useful for understanding loops, functions, and classes

2. "Object-Oriented Programming in C++" (4th Edition, 2002) by Robert Lafore. ○ Explains object-oriented concepts like encapsulation, constructors, and methods used in your robot class

3. "Data Structures and Algorithm Analysis in C++" (4th Edition, 2013) by Mark A. Weiss. ○ Offers insights into grids and algorithms that could optimize movement and decision-making in your robot simulation.

4. "Programming: Principles and Practice Using C++" (2nd Edition, 2014) by Bjarne Stroustrup. ○

Provides foundational knowledge and practical programming practices for C++.

5. "Introduction to Autonomous Robots" (2nd Edition, 2017) by Nikolaus Correll et al. ○ Covers the basics of robot navigation and decision-making, similar to the logic in your code


Thank You
minions