Dimitru Nirca M00888146 (SCRUM Master), Osee Mbiya M00871715 (Developer), Yasin Lester M00809695 (Developer), Ethan Mebrahtu M00759052 (Secretary), Taf Mushamba M00579249 (Tester).

**Report on Voting Systems**

Table of Contents:

**Introduction1**

Design2

Testing3

Conclusion**4**

Referencing5

## 

## Introduction:

Our team will concentrate on choosing and putting into practice a suitable data structure to hold voter data details for quick searches as part of the voting systems project. We must prioritise both speed and reliability in our data structure selection since voter information is crucial and needs to be accessed quickly throughout voting operations. In order to support our choice, we will use this data structure to perform a thorough analysis of time complexity for important operations, making sure that it satisfies the project's needs. In order to evaluate the software's search capabilities, our team will also be in charge of generating a sample data file. It's important to note that we won't be using any third-party libraries or code in our solution, instead writing everything ourselves in standard C++. We will develop and implement the data structures ourselves, rather than relying on standard template libraries (STLs). This strategy enables us to use the knowledge gathered during the module to design our solution particularly to the Voting Systems project's requirements, assuring optimal performance and dependability.

In the Design section, we justify our selected data structures and algorithms in terms of efficiency and ease of implementation. We also examine key algorithms using pseudo-code representations to demonstrate their functionality without going into the specific C++ source code. The Testing section describes testing approaches, emphasising overall strategy rather than showing actual test code, and is supplemented by a table of test cases outlining various situations to check project performance. The conclusion summarises project accomplishments, reflects on recognised constraints, and recommends future approaches to comparable jobs to avoid making the same mistakes. Finally, the References section lists all sources mentioned in Harvard style, which ensures academic integrity and legitimacy by matching in-text citations.

## Design:

The electoral system is comprised on several major components:   
  
Candidate Management Module: Adds and removes candidates from the candidate list.   
Voter Registration Module: Manages voter registration and generates unique voter IDs.   
Voting Module: Allows authenticated voters to cast their votes more easily.   
Vote Counting Module: Tallies the votes and selects the winner based on the highest vote total.   
Security and Integrity Module: Implements a variety of security measures to safeguard the system from unauthorised access and preserve the integrity of the voting process.   
The Auditing and Verification Module generates audit trails, voter receipts, and conducts post-election audits to ensure openness and accountability.

Each component of the system serves distinct functions:   
  
Candidate Management: Allows you to input new candidates and delete them based on their IDs.   
Voter Registration: Creates unique voter IDs and maintains voter information in a database.   
Voting: Allows verified voters to vote for certain candidates while ensuring that each voter only votes once.   
Vote Counting: Totals the votes for each candidate to determine the winner.   
Security and Integrity: Uses authentication techniques, data encryption, and vote integrity checking to ensure election integrity.   
Auditing and Verification: Creates audit trails to record system actions, issues unique receipts to voters for verification, and conducts post-election audits to guarantee results are accurate.

The components work together flawlessly to facilitate the electoral process:   
  
Candidates are managed using the Candidate Management Module, which changes the candidate list as needed.   
Voters register using the Voter Registration Module and receive unique IDs for authentication.   
Authenticated voters use the Voting Module to cast their votes, which are securely stored.   
The Vote Counting Module tallies the votes to decide the winner.   
Security measures are implemented throughout the process to keep the system safe from threats.   
Auditing and verification procedures promote transparency and accountability.

## Testing:

**Statement of Testing Approach**

The testing approach for the [Project Name] involved a comprehensive strategy aimed at ensuring the correctness, reliability, and functionality of the system. The following methodologies were employed:

1. **Unit Testing**: Individual components, including classes and functions, were tested in isolation to verify their correctness and behaviours.
2. **Integration Testing**: Interaction between different components/modules was tested to ensure they function correctly together. This included testing data flow and communication between modules.
3. **System Testing**: The entire system was tested as a whole to evaluate its performance, reliability, and security aspects. This included testing against specified requirements to ensure it functions as expected.
4. **Acceptance Testing**: The system was validated against user requirements and specifications to ensure it meets user expectations and fulfills its intended purpose.

**Table of Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Description** | **Expected Outcome** | **Reproduction Steps** |
| Ballot Class Initialization | Test if Ballot constructor initializes voter and candidate pointers correctly. | Ballot object is created with correct voter and candidate pointers. | Create a Ballot object with known voter and candidate objects. Verify the pointers. |
| Candidate Class Initialization | Test if Candidate constructor initializes candidate attributes correctly. | Candidate object is created with correct attributes. | Create a Candidate object with known attributes. Verify each attribute. |
| Setting Votes in Candidate Class | Test if the setVotes() function updates the number of votes correctly. | Candidate's number of votes is updated to the new count. | Create a Candidate object with initial vote count. Call setVotes() with a new count. Verify the count. |
| Election Class Initialization | Test if the Election constructor initializes candidates, voters, and ballots containers correctly. | Election object is created with empty containers. | Create an Election object. Verify that containers are empty. |
| Loading Data in Election Class | Test if the loadData() function loads candidate and voter data correctly from files. | Candidate and voter containers are populated with data from files. | Call loadData() with known files. Verify containers are populated. |
| Voter Class Initialization | Test if the Voter constructor initializes voter attributes correctly. | Voter object is created with correct attributes. | Create a Voter object with known attributes. Verify each attribute. |

Conclusion:

An overview of the completed work:   
The code that is provided includes a C++ voting system. Voter education, ballots, candidates, and the election itself are all included. Voters can register, authenticate, and cast ballots for candidates using this method. Loading voter and candidate data from CSV files, voter authentication, voter registration for the first time, voting, candidate information viewing, and election results viewing are among the primary functionalities.   
  
The code has a modular structure, with distinct functions assigned to each class. While the Map class implements a key-value store based on a tree structure, the Vector class offers a dynamic array solution. The Election class manages the entire voting process, including data loading, voter authentication, vote casting, and result computation.

Limitations and Critical Reflection:

Lack of Error Handling: The code lacks comprehensive error handling, which can lead to unexpected behavior or crashes in certain scenarios. For example, file input/output operations and memory allocation are not adequately protected against errors.

Lack of Input Validation: While the code performs basic input validation, such as checking for valid voter names and dates of birth, it may not handle all edge cases or malicious inputs. More robust input validation mechanisms should be implemented to ensure data integrity and security.

Code Duplication: Some functionality, such as file reading and writing operations, is duplicated across multiple classes, leading to code redundancy and maintenance overhead.

Security Issues: Potential security flaws like SQL injection and buffer overflow attacks are not addressed by the system. To reduce these hazards, proper input validation and sanitization are crucial.  
  
How Would I Approach a Similar Task Differently in the Future?

In future versions, I would think about implementing the following modifications to solve the aforementioned restrictions and enhance the general quality of the code:   
  
Use exception handling techniques to implement thorough error handling so that unforeseen circumstances can be handled with grace and users can receive helpful error messages.   
To improve input validation and avoid typical security vulnerabilities, use regular expressions or specialised input validation libraries to make sure that input data satisfies predetermined requirements.

Refactor code by encapsulating common functionality into reusable utility functions or classes to reduce redundancy and encourage code reuse.   
Make security a top priority by employing secure coding techniques, carrying out in-depth security assessments, and making use of security tools like vulnerability scanners and static code analysers.   
To build cleaner, safer, and more maintainable code, embrace contemporary C++ features and best practices like smart pointers, RAII (Resource Acquisition Is Initialization), and the Standard Library.   
Conduct comprehensive testing, such as security, integration, and unit tests, early in the development lifecycle to find and fix possible problems.   
Future iterations of comparable activities can attain more dependability, security, and maintainability by implementing these modifications, which will eventually improve user experience and lower the risk of significant failures or security breaches.

## References :

W3Schools. C++ Tutorial. Available via this link: <https://www.w3schools.com/> (Accessed throughout the creation of the voting system)

Stack Overflow. Available via this link: <https://try.stackoverflow.co/> (Accessed throughout the creation of the voting system)

Learn C++. Used for the tutorials. Available via this link: <https://learncpp.com/> Accessed throughout the creation of the voting system)