**FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY**

DEPARTMENT OF COMPUTING AND TECHNOLOGY

ADVENT 2024 SEMESTER OOP COURSEWORK PROJECT REPORT

PROGRAM: BSCS, BSDS 2:1

COURSE: Object-Oriented Programming

COURSE LECTURER: Ian Raymond Osolo

E-VOTING SYSTEM

*Submitted by*

|  |  |  |  |
| --- | --- | --- | --- |
| S/N | Reg Number | Name | Signature |
| 1. | S23B38/022 | AWOR JOY FLORENCE |  |
| 2. | S23B38/025 | ANITA NAMAGANDA |  |
| 3. | S23B38/015 | NAMATOVU LIZ TREASURE |  |
| 4. | S23B38/019 | RWOTHOMIO CALVIN DIEGO |  |
| 5. | S23B38/006 | OMOYA CHARLES KIDEGA |  |
| 6. | S23B38/016 | RUGOGAMU NOELA |  |

Date Submitted: 12th November 2024

**1.0 Abstract**   
This report presents the design and implementation of an E-voting system in Python, structured to provide a secure, efficient, and user-friendly platform for digital voting. Developed as part of an Object-Oriented Programming (OOP) coursework, the project incorporates core OOP principles to organize and manage voting operations effectively.

The system is composed of four main classes: **Participant**, **Voter**, **candidate**, and **Election**. The participant class serves as the foundation, providing common attributes and methods for both voters and candidates. The Election class coordinates the voting process itself, tracking each candidate’s vote count and maintaining the election's overall status.

Key features of the system include secure user authentication, role-based permissions, single-vote enforcement, and dynamic result tracking. Together, these elements demonstrate how a modular, object-oriented structure can streamline the digital voting process, ensuring accuracy, security, and an intuitive experience for all users.

**2.0 Introduction, problem statement, and project objectives**

**2.1 Introduction**

Voting is a fundamental process in democratic societies, but traditional voting systems face numerous challenges, including logistical difficulties, human errors, and limited accessibility for voters. As technological advancements become more integrated into our daily lives, the need for digital voting solutions has grown, aiming to offer a secure, accessible, and efficient alternative to traditional methods.

The E-voting system developed in this project aims to create a reliable platform where individuals can participate in elections digitally. This system leverages Object-Oriented Programming (OOP) to organize the code into specific roles and responsibilities, ensuring that the application remains modular, secure, and easy to maintain. Using Python, this project applies OOP principles to create a system where different types of users (voters and administrators) can interact with the platform according to their roles, contributing to an organized and user-friendly experience.

**2.2 Problem Statement**

Traditional voting systems are often limited by several issues that affect both the efficiency and security of elections. These issues include:

1. **Manual Errors and Inefficiencies**: Traditional systems require substantial manual effort to set up, manage, and count votes, increasing the risk of human errors and slowing down the election process.
2. **Accessibility Challenges**: Physical voting systems may limit participation due to location constraints or accessibility issues, potentially leading to lower voter turnout.
3. **Data Integrity and Security Risks**: Ensuring the accuracy and security of votes is challenging in traditional systems, with risks such as unauthorized access, multiple voting, and vote tampering.

This project seeks to address these challenges by developing a Python-based E-voting system that provides an accessible, secure, and efficient platform for digital voting. By organizing the code using OOP principles, this system ensures that roles and permissions are clearly defined, sensitive data is protected, and the election process is streamlined for both voters and administrators.

**2.3 Project Objectives**

The main objectives of this project are as follows:

1. **Create a Secure Voting Platform**: The E-voting system is designed to ensure data security at each step of the voting process. Using secure user authentication, each participant can access the system according to their role (voter or administrator), with restricted permissions based on their assigned responsibilities. For instance, the Voter class restricts users to a single vote per election, while the Admin class grants administrators the ability to start and end elections and view results, all protected against unauthorized access.
2. **Ensure Single-Vote Integrity**: A core objective is to prevent multiple votes by the same user. The Voter class incorporates a flag (has\_voted) to track whether a user has already cast a vote, ensuring that each user can vote only once per election. This feature is crucial to maintain the integrity of the election process, preventing duplicate votes and potential manipulation.
3. **Provide Real-Time Voting Results**: The system is designed to offer dynamic tracking and display of results through the Election class. By maintaining a count of votes for each candidate, the system allows administrators to view real-time or final results once the election period ends. This objective ensures transparency and allows administrators to monitor election progress without requiring manual counting.
4. **Maintain System Flexibility and Scalability**: By organizing the system into modular classes, the project aims to make the codebase easy to maintain, modify, and expand. New features, such as additional user roles or enhanced security layers, can be integrated with minimal disruption to the existing structure. This modular design demonstrates the practical benefits of using OOP for developing scalable applications that can adapt to future needs.

**3.0 Methods, tools, and designs used for the project**

**3.1 Methods**

The E-voting system is structured with a modular design, consisting of four core classes, **Participant**, **Voter**, **Candidate, vote** , and **Election**—each tailored to handle a specific part of the voting process:

1. **Participant**:
   * **Purpose**: Acts as a base class to define shared properties and authentication methods for both voters and candidates.
   * **Attributes**:
     + age\_validity and location\_validity: These are fundamental attributes used for authentication. Both Voter and candidate take these attributes embedded in the base class.
2. **Voter Class**:
   * **Purpose**: Manages voter-specific actions and ensures each voter can participate in the election according to set rules.
   * **Methods**:
     + **register\_vote**: Registers a new voter, capturing necessary user data to create a new User profile.
     + **cast\_vote()**: This method allows a voter to submit a vote for a chosen candidate. Before casting, it checks has\_voted to ensure the user hasn’t already voted. If the vote is successfully cast, has\_voted is updated to True.
     + **age\_validity;** checks for voters age and verifies to ensure it’s the right age as specified in the code
     + **location\_validity;** checks for voters location and verifies to ensure it’s the right location as specified in the code
     + **voting\_status;** checks if a voter has voted or not and outputs the necessary output
   * **Role in System**: The Voter class defines the main interactions a candidate has with the election, ensuring secure and single-instance voting.
3. **Candidate class**
   * Has same methods as the voter class but the only difference is, for age\_validity, candidates between 25 and 70 years are allowed to take part
   * Added method include receive\_vote and get\_vote
4. **Election Class**:
   * **Purpose**: Acts as the central manager of the election process, handling candidate data, vote counting, and the election’s active status.
   * **Attributes**:
     + candidates: Stores each candidate and their vote count in a dictionary, where each candidate’s name is a key, and their corresponding vote count is a value.
     + status: A string or Boolean attribute that indicates whether the election is open or closed, controlled by start\_election() and end\_election() methods.
   * **Methods**:
     + add\_candidate(): Allows addition of candidates to the candidate list.
     + add\_voter(): IAllows for addition of votes to the voter list
     + show\_results(): Returns a tally of votes for each candidate, enabling the viewing of the current results.
   * **Role in System**: The Election class manages all aspects of vote tracking and candidate data, making it a centralized component for overseeing the election's progress and final outcomes.

**3.2 Tools**

The primary tool for developing this project is **Python**, which enables the structured use of classes and methods to implement a modular voting system.

The system also employs the following OOP principles to create a modular, scalable, and secure application:

1. Encapsulation: The system uses encapsulation to restrict direct access to sensitive data and operations within the Voter and Admin classes. For example, private attributes for user credentials are securely managed.
2. Abstraction: Abstraction is implemented by focusing on essential features while hiding the underlying implementation details. The Election class, for instance, abstracts complex operations related to managing votes.
3. Inheritance: Inheritance enables code reuse across user roles, with a base User class providing common attributes and methods shared by Voter and Admin.  
   Polymorphism: Polymorphism in this project allows methods in Voter and Admin to be called in the same way, even if they behave differently.

**4.0 Results**

Project Description:

This project simulates a voting system designed for a fictional election process. It provides a comprehensive framework for managing participants (voters and candidates), casting votes, and calculating election results. The project leverages key object-oriented programming (OOP) principles such as inheritance, encapsulation, abstraction, and polymorphism.

---

**Core Components**

1. Participant (Abstract Base Class)

- Represents a general participant in the election (either a voter or candidate).

- Defines shared properties such as name, gender, age, and location.

- Includes abstract methods:

- age\_validity(): Determines if the participant's age qualifies for their role.

- location\_validity(): Validates if the participant resides in the allowed election location.

2. Voter (Inherits from Participant)

- Represents an individual eligible to vote.

- Additional attributes:

- voter\_id: Unique identifier for the voter.

- `has\_voted`: Boolean indicating if the voter has cast their vote.

- Implements:

- age\_validity(): Confirms if the voter is at least 18.

- location\_validity(): Checks if the voter resides in the designated voting area.

- register\_voter(): Registers the voter if eligible.

- cast\_vote(): Allows the voter to cast their vote, preventing multiple votes.

- voting\_status(): Displays if the voter has voted.

3. Candidate (Inherits from Participant):

- Represents an individual running for election.

- Additional attributes:

- candidate\_id: Unique identifier for the candidate.

- votes: Number of votes received by the candidate.

**- Implements:**

- age\_validity(): Validates that the candidate is between 25 and 70 years old.

- location\_validity(): Ensures the candidate resides in the election location.

- register\_candidate(): Registers the candidate if eligible.

- receive\_vote(): Increments the candidate's vote count.

4. Vote:

- Represents the process of casting a vote.

- Attributes:

- voter: Instance of the `Voter` class.

- candidate: Instance of the `Candidate` class.

- Method:

- cast\_vote(): Ensures the voter is eligible and records their vote for the candidate.

5. Election:

- Manages the election process.

- Attributes:

- election\_name: Name of the election.

- \_voters: List of registered voters.

- \_candidates: List of registered candidates.

- Methods:

- add\_voter(voter): Registers and adds a voter to the voter list.

- add\_candidate(candidate): Registers and adds a candidate to the candidate list.

- show\_results(): Displays the election results and announces the winner.

**Key Features**

1. Encapsulation:

- Sensitive data, such as voter IDs and vote status, are protected through private attributes.

2. Abstraction:

- The Participant class defines abstract methods that are implemented differently by `Voter` and `Candidate`.

3. Polymorphism:

- The age\_validity() and location\_validity() methods are overridden in both Voter and Candidate classes to suit their specific rules.

4. Inheritance:

- Voter and Candidate, inherit from Participant to reduce redundancy and promote code reuse.

5. Dynamic Vote Casting:

- Votes are cast dynamically through the Vote class, updating the voting status and vote count.

**Justification: How the Project Solves the Problem and Achieves Objectives**

This project addresses the challenges of traditional voting systems and meets the outlined objectives through a well-structured, secure, and efficient E-voting system. Here's how:

**1. Addressing Manual Errors and Inefficiencies**

Problem: Traditional voting systems rely heavily on manual processes, which can lead to human errors and slow vote counting.

solution:

Automated Processes; The system automates voter registration, candidate registration, and vote counting.

- The Vote class ensures votes are immediately recorded, eliminating the need for manual tallying.

- The show\_results() method in the Election class dynamically calculates and displays results, reducing delays and errors.

Objective Achieved: The system improves efficiency by automating vote management and result computation, minimizing the risk of manual errors.

**2. Tackling Accessibility Challenges**

Problem: Physical voting systems may limit participation due to location constraints or lack of accessibility for some voters.

Solution:

Location-Based Validation: The system uses location\_validity() methods in both the Voter and Candidate classes to ensure participants are from allowed locations.

- This feature could be adapted for broader accessibility, such as enabling remote voting for different regions in future iterations.

Flexibility for Expansion:

- The modular design allows for easy integration of new accessibility features, such as remote voting or integration with assistive technologies.

Objective Achieved:

The system lays the foundation for overcoming accessibility challenges by ensuring participation from valid locations and supporting future scalability.

**3. Ensuring Data Integrity and Security**

Problem: Traditional systems are vulnerable to unauthorized access, multiple voting, and vote tampering.

Solution:

Secure Voting Platform:

- The Voter and Candidate classes encapsulate sensitive data such as `\_has\_voted` and votes, preventing unauthorized access.

- The `cast\_vote()` method enforces strict controls, ensuring a voter can cast a vote only once.

Single-Vote Integrity:

- The `\_has\_voted` flag in the `Voter` class tracks whether a user has voted, ensuring that no user can vote multiple times.

- Any attempt to vote again results in an error message, preserving election integrity.

Restricted Permissions:

- While this implementation focuses on voters and candidates, the design allows for an `Admin` class with elevated privileges, such as starting or ending the election, ensuring role-based access control.

Objective Achieve: The system ensures that every vote is legitimate and secure, preventing vote duplication or tampering.

**4. Providing Real-Time Voting Results**

Problem: Traditional systems require manual counting, delaying result announcements and risking human errors.

Solution:

Dynamic Results Tracking:

- The `Election` class keeps a live tally of votes for each candidate through the `get\_votes()` method.

- Administrators can call the `show\_results()` method at any time to display up-to-date results, ensuring transparency and real-time monitoring.

Transparency and Fairness:

- By instantly showing vote counts and declaring the winner, the system eliminates delays and fosters trust in the election process.

Objective Achieved:

The system ensures quick, accurate, and transparent result computation, improving stakeholder confidence.

**5. Ensuring Flexibility and Scalability**

Problem: Traditional systems lack flexibility, making it difficult to adapt to changing requirements or add new features.

Solution:

- Modular Design:

- The project is organized into distinct classes: `Participant`, `Voter`, `Candidate`, `Vote`, and `Election`.

- This modularity allows for the easy addition of new features or roles (e.g., `Admin`, `Observer`) without disrupting existing functionality.

5.0 Conclusion & Recommendation (1 Page)

**Conclusion**

The development of this E-voting system successfully addresses the challenges inherent in traditional voting processes, including inefficiencies, accessibility constraints, and security risks. Through a structured, object-oriented design, the system ensures:

**Efficiency**: Automating voter and candidate registration, vote casting, and result computation minimizes manual errors and speeds up the election process.

**Security**: Data integrity is preserved through strict controls on voting actions, ensuring that only eligible participants can vote and preventing duplicate voting.

**Transparency**: Real-time vote tracking and instant result computation foster trust in the electoral process by providing timely and accurate outcomes.

**Scalability and Flexibility**: The modular design allows for easy adaptation and future enhancements, ensuring the system remains relevant as requirements evolve.

**Recommendation**

To further enhance the system's capabilities and applicability, the following recommendations are proposed:

1. Implement Advanced Security Features:

- Introduce secure user authentication (e.g., two-factor authentication) to prevent unauthorized access.

- Encrypt sensitive data such as voter information and voting results to enhance security.

2. Enhance Accessibility:

- Develop a web-based or mobile application to allow remote voting, ensuring greater voter participation.

- Integrate assistive technologies to support voters with disabilities.

3. Introduce Role-Based Access Control:

- Add an `Admin` role to manage election configurations, such as starting and ending elections and accessing detailed analytics.

- Implement an `Observer` role for independent monitoring of the election process.

4. Support Multi-Location Elections:

- Expand the system to handle elections across multiple locations or constituencies.

- Allow for configurable location validations to accommodate different electoral jurisdictions.

5. Real-Time Analytics and Reporting:

- Provide detailed insights into voter turnout, voting patterns, and candidate performance through a real-time dashboard.

- Generate comprehensive reports post-election for archival and auditing purposes.

By implementing these enhancements, the system can further improve its effectiveness, security, and user experience, making it suitable for larger-scale elections and diverse electoral needs.