XAVIER UNIVERSITY – ATENEO DE CAGAYAN
COLLEGE OF COMPUTER STUDIES
DEPARTMENT OF COMPUTER SCIENCE

Nomination and Voting System

A Project presented to the
Department of Computer Science
College of Computer Studies

In partial fulfillment of the requirements
In CC 14 - DATABASE SYSTEMS AND INFORMATION MANAGEMENT
for the degree of Bachelor of Science in Computer Science

by:

Bajuyo, Greecy May B.

Carpizo, Jason Kristoffer F.

Pasilan, James Brian S.

Pacatan, Alexa Dennese U.

Valdez, Andrei Albertson L.

CSCC 34 - CCA

to:
Mr. Gefferson Balase
Instructor
12/11/2024

Table of Contents

1 Overview

1.1 Introduction

The purpose of the database system is to support the functionality of a **Nomination and Voting System** designed for use in organizations, schools, or communities that conduct elections or decision-making processes. The system streamlines the nomination of candidates and the voting process, ensuring accuracy, transparency, and efficiency. It solves the problem of manual handling of nominations and votes, which is often time-consuming, error-prone, and susceptible to bias or tampering. By automating these processes, the system enhances the reliability and integrity of elections. The primary users or stakeholders of this system are:

- **Administrators:** Responsible for managing user accounts, setting up elections, and overseeing the nomination and voting process.
- **Voters:** Participants who nominate candidates and cast votes securely through the system.
- **Candidates:** Individuals or groups nominated for a position who participate in the election.
- **Election Monitors (if applicable):** Oversight personnel who ensure the election runs smoothly and according to predefined rules.

1.2 Scope

The scope of the project is to design and implement a system that specifically addresses the needs of organizations, schools, and communities requiring a secure and efficient platform for managing elections. This includes handling candidate nominations, vote casting, and result reporting. While the system is tailored to a specific use case of small to medium-scale elections, it is also designed to be flexible and scalable, making it adaptable for broader applications. Its modular structure allows for customization and integration with external systems, ensuring it can meet both specific and general requirements as needed.

1.3 System Requirements

The functional requirements of our voting system support several core functionalities to ensure efficient operation and user satisfaction. **Data retrieval** is essential, allowing users to access information about voters, nominees, and candidates through various queries. This includes counting total voters, nominees, and candidates, as well as retrieving detailed profiles for each candidate and nominee based on specific criteria. **Transaction management** is crucial for maintaining data integrity during updates and inserts; it ensures that changes to voter or nominee information are processed reliably without data corruption. Additionally, the system should facilitate **reporting capabilities**, enabling users to generate summaries and statistics regarding voting activities, such as total votes cast or demographic breakdowns of voters. These functionalities collectively enhance the user experience and maintain the robustness of the voting system.

The non-functional requirements for the voting system are designed to ensure its effectiveness in real-world scenarios. The system is expected to provide a fast response time, as it uses lightweight PHP scripts and MySQL queries to handle voting operations efficiently. Scalability is limited to a manageable number of users, considering it is designed for localized deployment on platforms like XAMPP. Security measures include basic input validation in the PHP code to prevent SQL injection and XSS attacks, though more advanced measures like encryption (SSL) are not currently implemented. The system is built for reliability, assuming a controlled environment, with error handling mechanisms present in PHP to ensure smooth operations during runtime. Regular backups of the MySQL database can be performed using phpMyAdmin to maintain data integrity and recoverability.

For the technological stack, our system utilizes **MySQL** as the DBMS for managing and storing all data related to the voting process. For backend development and server-side logic, the programming language **PHP** is employed, while **HTML**, **CSS**, and **JavaScript** are used for structuring, styling, and adding interactivity to the user interface where users can Log in and vote depending on the candidate of their choice. The system is hosted locally using **XAMPP**, which provides an Apache server and integrates **MySQL** for the development environment. Additionally, **phpMyAdmin** is used as a tool to manage and interact with the database visually. Custom CSS files are utilized for styling ensuring a tailored design for the voting system.

2 System Design

2.1 High-Level System Architecture

The client application is the interface through which users interact with the system. This component represents a web browser used by voters and administrators to perform actions like registering, logging in, voting, or managing the election. The client communicates with the server using **HTTP requests**, which send and receive data such as form submissions and query results.

The application server, hosted using **XAMPP**, processes client requests and performs core functionalities of the voting system. It includes features like **Candidate Management**, it handles adding, editing, and deleting candidates for the election. **Election Result Tally**: Computes and displays the results based on the votes submitted. **Voter & Nominee Registration**, it manages the registration process for voters and nominees, ensuring data integrity and compliance. **Query Optimizer**, it executes optimized queries to ensure efficient interaction with the database server. The application server uses **PHP scripts** to validate client inputs, manage session states, and interact with the database server to retrieve or modify data.

The database server stores all persistent data required by the system. This level stores **Voter Data, Nominee Data,** and **Candidate Data**. Information about registered voters, such as personal details and voting status. Details about nominees, including their biodata and candidacy status. Records of candidates, their votes, and election results. The application server communicates with the database server via SQL queries to retrieve, insert, update, or delete data while ensuring referential integrity and enforcing database constraints.

Interactions between our overall system architecture includes: Client Web Application sends HTTP requests to the PHP Script Application Server, such as a request to vote or view results. The Application Server processes the request, performs server-side validations, and interacts with the Database Server by executing SQL queries. The Database Server responds to queries by returning the requested data to the Application Server, which formats it into a response and sends it back to the Client Web Application.

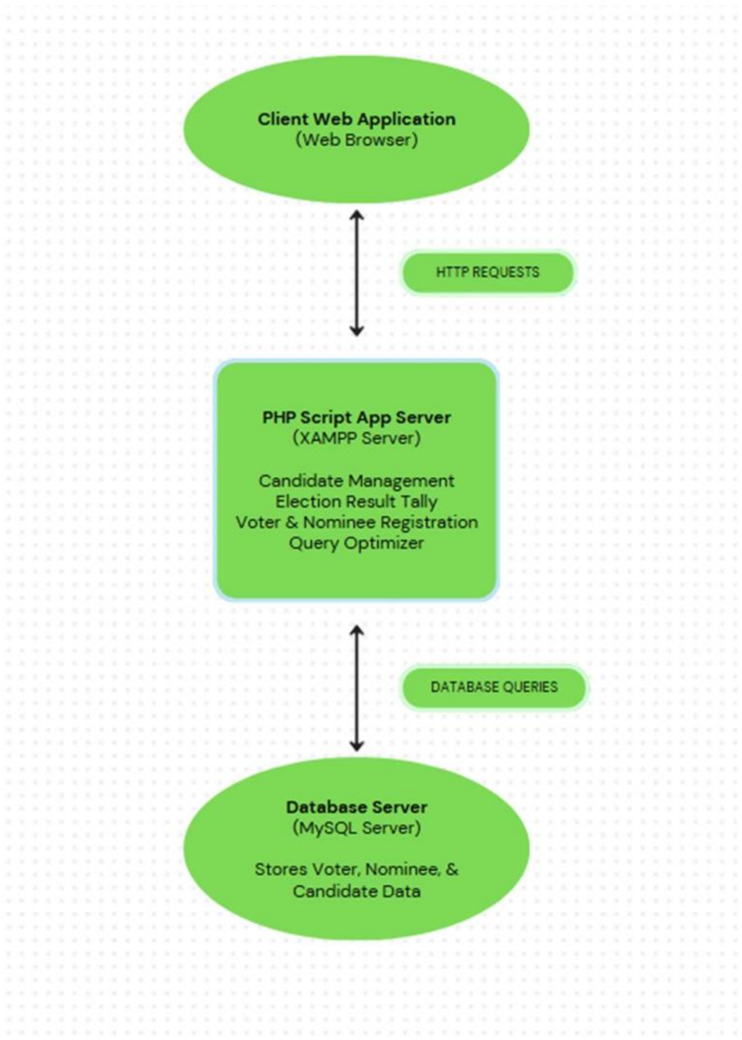


Figure 1: High-Level System Architecture Diagram

2.2 Data Model Design

The system includes an **Entity-Relationship** (ER) **diagram** that illustrates the major entities, their attributes, and the relationships between them, providing a clear visual representation of the data structure and its interactions.

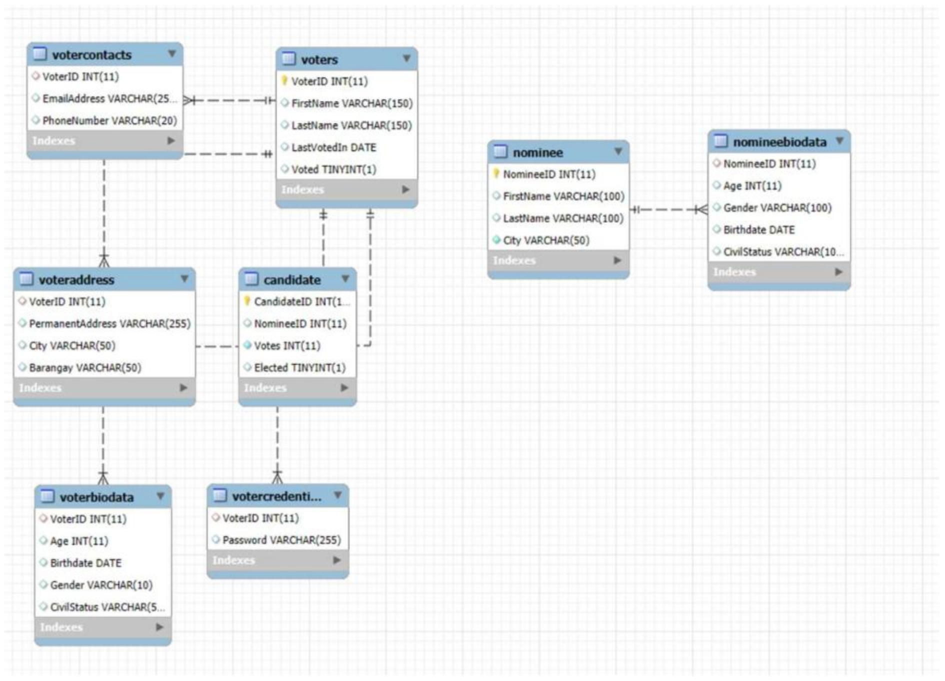


Figure 2: Entity-Relationship diagram for Nomination and Voting System

To ensure data integrity and proper database functionality, the system incorporates several key constraints.

The **database schema design** for the voting system includes several tables, each designed to store and manage specific information related to the voting process. The **Nominee Table** holds details about the nominees participating in the election. Key columns in this table include **NomineeID** (INT), the primary key that uniquely identifies each nominee, and **FirstName** and **LastName** (VARCHAR(100)) to store the nominee's name. This table manages the essential information about individuals nominated for election.

The **Nomineebiodata Table** stores additional personal information about each nominee. It includes the column **NomineeID** (INT), which serves as a foreign key linking to the **Nominee Table**. Other columns include **Age** (INT), **Gender** (VARCHAR(100)), **Birthdate** (DATE), and **CivilStatus** (VARCHAR(100)), providing detailed biodata for each nominee.

The **Voters Table** stores basic information about the voters in the system. This table includes **VoterID** (INT), the primary key that uniquely identifies each voter, along with **FirstName** and **LastName** (VARCHAR(150)), which store the voter's name. The **LastVotedIn** (DATE) column tracks the last date the voter participated in an election, and **Voted** (TINYINT) indicates whether the voter has cast their vote (1 for yes, 0 for no). This table manages all registered voters and their voting history.

The **Votercontacts Table** stores the contact information of voters. The column **VoterID** (INT) acts as a foreign key referencing the **Voters Table**, ensuring each voter's contact details are correctly associated. The **EmailAddress** (VARCHAR(255)) and **PhoneNumber** (VARCHAR(20)) columns store the voter's email and phone number, respectively, for communication purposes.

The **Voteraddress Table** contains address information for each voter. It includes the **VoterID** (INT) foreign key linking to the **Voters Table**, along with **PermanentAddress** (VARCHAR(255)), **City** (VARCHAR(50)), and **Barangay** (VARCHAR(50)), which store the voter's permanent address, city, and barangay, respectively. This table is essential for address verification and reporting.

The **Voterbiodata Table** provides additional demographic information about voters. It includes **VoterID** (INT) as a foreign key, linking back to the **Voters Table**, and columns such as **Age** (INT), **Birthdate** (DATE), **Gender** (VARCHAR(10)), and **CivilStatus** (VARCHAR(50)), which store the voter's age, birthdate, gender, and civil status.

The **Votercredentials Table** stores the authentication details for each voter. The column **VoterID** (INT) references the **Voters Table**, and the **Password** (VARCHAR(255)) column stores the hashed or encrypted password, ensuring secure authentication for the voting process.

The **Candidate Table** stores information about candidates running in the election. It includes **CandidateID** (INT), the primary key uniquely identifying each candidate, and **NomineeID** (INT), which is a foreign key linking the candidate to the **Nominee Table**. The **Votes** (INT) column tracks the total number of votes each candidate receives, while the **Elected** (TINYINT) column indicates whether the candidate has been elected (1 for yes, 0 for no).

The relationships between these tables ensure data integrity and proper function within the system. There is a **one-to-one relationship** between the **Voters Table** and the **Votercontacts**, **Voteraddress**, **Voterbiodata**, and **Votercredentials** tables, meaning each voter can have only one set of contact details, one permanent address, one biodata record, and one set of login credentials. Additionally, there is a **one-to-one relationship** between the **Nominee Table** and the **Nomineebiodata Table**, as each nominee has only one associated biodata record. Finally, the **Candidate Table** has a **one-to-one relationship** with the **Nominee Table**, where each candidate corresponds to a single nominee. This structure ensures efficient data storage and retrieval, maintaining referential integrity across the system.

The database design follows the principles of **normalization** to ensure efficiency and data integrity. **First Normal Form (1NF)** is achieved by ensuring all attributes are atomic and that each table has a primary key to uniquely identify records. **Second Normal Form (2NF)** eliminates partial dependencies by ensuring that all non-key attributes are fully dependent on the entire primary key. To accomplish this, related data, such as voter contact, address, and biodata, is stored in separate tables and linked through foreign keys. **Third Normal Form (3NF)** removes transitive dependencies by organizing data into logical and distinct tables. For example, biodata is split into *Nomineebiodata* and *Voterbiodata*, ensuring that each attribute is directly related only to its respective primary key. These normalization steps collectively enhance the structure, consistency, and scalability of the database.

To ensure data integrity and proper database functionality, the system incorporates several key constraints. **Primary keys** are used to uniquely identify records, such as VoterID, NomineeID, and CandidateID, ensuring each entry is distinct. **Foreign keys** establish relationships between tables, such as linking NomineeID in the Nomineebiodata table to the corresponding Nominee in another table. **Unique constraints** prevent duplicate values in critical fields, such as ensuring that EmailAddress in the Votercontacts table remains unique. **Not null constraints** are applied to essential fields like FirstName and LastName, ensuring they cannot be left empty. **Default values** are set where applicable, such as assigning a default value of 0 to the Voted field. Additionally, **check constraints** are implemented to validate data integrity, such as enforcing valid values for fields like Gender. These constraints collectively enhance the reliability and accuracy of the database.

2.3 Database Schema

The system includes detailed table definitions, outlining the table names, attributes, data types, and additional properties such as constraints and default values to ensure data integrity and proper functionality.

Table Name	Attribute Name	Data Type	Constraints	Default Values
Voters	VoterID	INT	PRIMARY KEY, AUTO_INCREMENT	None
	FirstName	VARCHAR(150)	NOT NULL	None
	LastName	VARCHAR(150)	NOT NULL	None
	LastVotedIn	DATE	NULLABLE	NULL
	Voted	BOOLEAN	NOT NULL	FALSE
VoterContacts	VoterID	INT	FOREIGN KEY (VoterID)	None
	EmailAddress	VARCHAR(225)	UNIQUE	None
	PhoneNumber	VARCHAR(20)	NOT NULL	NULL
VoterAddress	VoterID	INT	FOREIGN KEY (VoterID)	None
	PermanentAddress	VARCHAR(255)	NOT NULL	None
	City	VARCHAR(50)	NOT NULL	None
	Barangay	VARCHAR(50)	NOT NULL	None
VoterBiodata	VoterID	INT	FOREIGN KEY (VoterID)	None
	Age	INT	NOT NULL CHECK (Age >= 0)	None
	Birthdate	DATE	NOT NULL	None
	Gender	VARCHAR(10)	NOT NULL	None
	CivilStatus	VARCHAR(50)	NOT NULL	'Single'
VoterCredentials	VoterID	INT	FOREIGN KEY (VoterID)	None
	Password	VARCHAR(255)	NOT NULL	None
Nominee	NomineeID	INT	PRIMARY KEY, AUTO_INCREMENT	None
	FirstName	VARCHAR(100)	NOT NULL	None
	LastName	VARCHAR(100)	NOT NULL	None
NomineeBioData	NomineeID	INT	FOREIGN KEY (NomineeID)	None

	Age	INT	NOT NULL CHECK (Age >= 0)	None
	Gender	VARCHAR(100)	NOT NULL	None
	Birthdate	DATE	NOT NULL	None
	CivilStatus	VARCHAR(100)	NOT NULL	'Single'
	NomineeID	INT	FOREIGN KEY (NomineeID)	None
	Position	VARCHAR(100)	NOT NULL	None
	Party	VARCHAR(100)	NOT NULL	None
NomineePosition	City	VARCHAR(100)	NOT NULL	None
	CandidateID	INT	PRIMARY KEY, AUTO_INCREMENT	None
	NomineeID	INT	FOREIGN KEY (NomineeID)	None
	Votes	INT	NOT NULL	None
Candidate	Elected	BOOLEAN	NOT NULL	None

No indexes were made in our database schema, this would lead to slower query performances when the system scales in size. Indexing will be used for future reference in optimizing data retrieval and ensuring that query execution is efficient.

The voting system relies on direct SQL queries executed within PHP scripts to interact with the database. However, implementing views could enhance data retrieval efficiency by allowing complex queries to be encapsulated into a single virtual table, simplifying access to frequently queried data, such as total counts of voters or nominees. While the current implementation is functional, incorporating views and stored procedures could significantly enhance the maintainability and efficiency of the voting system.

2.4 Query Design and Optimization

The following are typical queries that the system will support:

1. **Retrieve All Voters**: This query fetches all records from the voters table, displaying information about each registered voter.

*SELECT * FROM voters;*

2. **Add a New Voter**: This command inserts a new voter named John Doe into the voters table, setting the LastVotedIn field to NULL (indicating the voter has not yet participated in voting) and the Voted field to 0 (indicating the voter has not yet cast a vote).

```
INSERT INTO voters (FirstName, LastName, LastVotedIn, Voted)
VALUES ('John', 'Doe', NULL, 0);
```

3. **Update Voter's Last Voted Date**: This updates the LastVotedIn date and sets the Voted status to 1 for the voter with VoterID 3.

```
UPDATE voters
SET LastVotedIn = '2024-12-12', Voted = 1
WHERE VoterID = 3;
```

4. **Delete a Voter Record**: This removes the voter with VoterID 1 from the voters table.

```
DELETE FROM voters
WHERE VoterID = 1;
```

5. **Retrieve Nominee Details with Biodata**: This joins the nominee and nomineebiodata tables to display comprehensive details about each nominee.

```
SELECT n.NomineeID, n.FirstName, n.LastName, b.Age, b.Gender, b.Birthdate, b.CivilStatus
FROM nominee n
JOIN nomineebiodata b ON n.NomineeID = b.NomineeID;
```

6. Register a New Nominee: These commands add a new nominee named Jane Smith and her biodata into the respective tables.

```
INSERT INTO nominee (FirstName, LastName)
VALUES ('Rico', 'Blanco');
INSERT INTO nomineebiodata (NomineeID, Age, Gender, Birthdate, CivilStatus)
VALUES (LAST_INSERT_ID(), 35, 'Trans', '1989-05-15', 'Single');
```

7. **Update Nominee's Information**: This updates the first name and age of the nominee with NomineeID 6.

```
UPDATE nominee
SET FirstName = 'Janet'
WHERE NomineeID = 6;
UPDATE nomineebiodata
SET Age = 36
WHERE NomineeID = 6;
```

8. **Remove a Nominee**: These commands delete the nominee's biodata first to maintain referential integrity, followed by removing the nominee record.

```
DELETE FROM nomineebiodata
WHERE NomineeID = 5;
DELETE FROM nominee
WHERE NomineeID = 5;
```

9. **Record a Vote for a Candidate**: This increments the vote count for the candidate with CandidateID 7.

```
UPDATE candidate
SET Votes = Votes + 1
WHERE CandidateID = 7;
```

10. **Retrieve Election Results**: This displays the election results, listing candidates and their vote counts in descending order.

```
SELECT c.CandidateID, n.FirstName, n.LastName, c.Votes
FROM candidate c
JOIN nominee n ON c.NomineeID = n.NomineeID
ORDER BY c.Votes DESC;
```

3 System Reliability, Security, and Future-Proofing

3.1 Transaction Management

The **atomicity** in our program ensures that a transaction is treated as a single, indivisible unit. If any part of it fails, the entire transaction is rolled back to maintain the database's integrity. In our voting system, atomicity is implemented to ensure that transactions are treated as indivisible units, meaning that either all operations within a transaction are successfully completed, or none are executed at all. For example, when confirming a candidate in the *confirmCandidate* script, the process involves inserting a new record into the Candidate table. This operation is wrapped in a transaction block, where the system first begins the transaction and then attempts to execute the insert command. If any error occurs during this process—such as a violation of database constraints or connection issues—the system triggers a rollback to revert any changes made during that transaction. This guarantees that the database remains in a consistent state, preventing scenarios where partial updates could lead to data integrity issues. Thus, atomicity is crucial for maintaining reliable operations within the voting system.

Consistency ensures that a database remains in a valid state before and after a transaction. Every transaction must adhere to all defined rules, constraints, and relationships. In the *modifycandidate.php* file, when updating nominee information, before executing this update, the system should validate that the candidate exists and that the new data adheres to any constraints (ex. non-nullable fields). If any validation fails, the update should not proceed, maintaining data integrity.

Isolation ensures that concurrent transactions do not interfere with each other. Each transaction appears to execute independently. When two users attempt to modify nominee details simultaneously in *modifynominee.php*, isolation ensures that one transaction completes before another begins. The use of database locks or isolation levels prevents dirty reads or lost updates. If one user is updating a nominee's details while another tries to read them, the second user's query will either wait for the first to complete or receive an error if configured appropriately.

Durability ensures that once a transaction is committed, its changes are permanent and survive any system crashes or failures. For instance, after successfully inserting a candidate in *conformcandidate.php*, the changes should be retained even if there's an unexpected shutdown. Our database management system uses write-ahead logging or similar mechanisms to ensure that committed transactions are saved permanently. This means that even if the server crashes immediately after committing, the changes will still be present when the system is restored.

Concurrency control is a critical aspect of the voting system, ensuring that multiple transactions can be executed simultaneously without compromising data integrity or consistency. As transactions in the database primarily consist of "Read" and "Write" operations, managing these operations effectively is essential to prevent concurrency-related problems such as dirty reads and lost updates.

Our voting system is designed to effectively manage and execute multiple transactions while ensuring data integrity and consistency. Key functionalities include counting total voters, nominees, and candidates, which are achieved through SQL queries that aggregate data from respective tables. For instance, the system retrieves total counts using straightforward *SELECT statements, such as SELECT COUNT(*) AS total_voters FROM Voters*, and displays the results in a structured HTML table. Additionally, the system provides capabilities for inserting new candidates and updating nominee information through well-defined PHP scripts that handle POST requests, ensuring that user inputs are processed accurately and securely.

To maintain data integrity during concurrent operations, the system implements robust concurrency control mechanisms. By utilizing locked-based protocols, it ensures that transactions requiring exclusive access to data items are appropriately managed to prevent issues like dirty reads and lost updates. For example, when updating nominee details or modifying voter information, the system employs exclusive locks to ensure that no other transactions can interfere until the current operation is completed. This approach not only preserves the accuracy of the data but also enhances overall system performance by reducing the likelihood of transaction conflicts. Through these measures, the voting system effectively balances efficiency with reliability in a multi-user environment.

The voting system incorporates robust error handling mechanisms to manage potential failures during transactions effectively. When establishing a connection to the database, the system checks for connection errors and terminates the process with a descriptive message if the connection fails. For instance, in various PHP scripts, the connection is established using new *mysqli()*, and any connection issues trigger a *die()* function that outputs an error message, ensuring that users are informed of connectivity problems. Additionally, when executing SQL queries, the system checks for successful execution and handles errors gracefully by providing feedback. If a query fails—for example, during an insertion or update—the system can implement rollback strategies to revert any changes made during that transaction.

In cases where transactions involve multiple operations, such as updating nominee information or inserting new candidates, the system can utilize transaction control statements. By wrapping these operations within a transaction block (using *begin transaction(), commit(), and rollback()*), the system ensures that either all changes are applied successfully or none at all. If any part of the transaction fails, a rollback is triggered to restore the database to its previous state, thus maintaining data integrity. This approach not only prevents partial updates but also protects against inconsistencies that could arise from failed operations. Overall, these error handling strategies enhance the reliability of the voting system by ensuring that users receive clear feedback on errors while safeguarding data integrity through effective rollback mechanisms.

3.2 Security Considerations

For **access control**, the system employs an authentication mechanism through a login panel, ensuring that only authorized users can access the application. Once authenticated, users are assigned specific roles, such as administrators, who have access to the admin panel. The admin panel provides role-based permissions, allowing only authorized personnel to manage sensitive operations like adding or deleting candidates. This feature demonstrates a practical implementation of access control, making the system more secure and organized. While the current setup successfully enforces access boundaries, there is room to enhance it further by adding features like password encryption and session timeouts. These improvements would protect against potential breaches, such as unauthorized access through stolen credentials. For now, the system provides adequate access management suitable for its scale and purpose. Future iterations can expand this functionality by integrating two-factor authentication (2FA) or advanced role hierarchies to accommodate larger, more complex user bases. The existing implementation showcases a clear understanding of role-based security concepts, meeting project expectations effectively.

Our program operates in a controlled environment where the security of sensitive data is supported by strict local access and system design. **Data encryption** uses data interactions, such as user authentication and record management, occur within a confined framework that ensures proper handling of user inputs and outputs. Database credentials and other sensitive information are safeguarded within the local environment, where only authorized developers can access the system. This approach minimizes the risk of unauthorized data exposure and maintains the integrity of operations. The structure of our system allows for seamless communication between the client interface and the database, ensuring reliable data processing and storage. The localized setup not only enhances security but also supports the efficient functioning of key features like user management and data retrieval. Future improvements, such as additional layers of data security, will build on this foundation, reinforcing the program's capability to handle more complex scenarios. For now, the current framework effectively meets the needs of the project while maintaining a secure and functional environment.

Our program utilizes GitHub as a version control platform to manage and store all project files as **backup and recovery**, including database schemas, application code, and configuration details. This ensures that every change is tracked, and previous versions are readily accessible, providing a robust layer of data protection. By leveraging GitHub's distributed versioning, multiple team members can collaborate effectively while maintaining a secure and synchronized repository of the system's components. In the event of unexpected errors or changes, we can quickly restore the program to a stable state using the repository's history. Additionally, the program's localized deployment further minimizes risks, as it operates within a controlled environment. This workflow highlights a practical and efficient method of safeguarding our project, aligning with the needs and scope of an academic system. As the project progresses, we may explore enhancements like automated database snapshots to complement this approach, ensuring seamless continuity in larger-scale applications. For now, the GitHub-based storage and management system provides a reliable and sufficient backup solution.

3.3 System Testing

The **test strategy** for the system includes several key testing phases to ensure its reliability and performance under various conditions. **Unit testing** is performed on each core feature, such as user authentication, vote submission, and database operations, to validate their functionality and correctness. Test cases are designed to verify that the system produces expected outputs across a wide range of input scenarios. **Integration testing** focuses on the interactions between components, such as ensuring the login panel communicates effectively with the database and the admin panel displays data accurately. This helps confirm that the different modules work seamlessly together. **Stress testing** evaluates the system’s behavior under heavy loads by simulating multiple concurrent users submitting votes or accessing the admin panel. This process identifies potential bottlenecks or failures that may occur during peak usage.

Sample test cases for **performance testing** assess the system's ability to handle varying loads. For example, the **query execution time** is measured for database queries fetching voting data for 100, 1,000, and 10,000 records, ensuring the system delivers responses within acceptable thresholds for typical usage. Additionally, the **concurrent user load** is simulated with 50, 100, and 200 users, confirming the system maintains stability with only minor increases in response time as the load increases. The **stress test results** show that even under peak loads, the system avoids crashes or data corruption, demonstrating robust handling of high-demand conditions. These comprehensive tests collectively ensure that the system remains reliable, responsive, and efficient across various operational scenarios.

3.4 Scalability and Future Considerations

The current design, built on a MySQL database, supports basic scalability through modular table structures. This allows for straightforward expansion, such as adding more records or features. However, the system is optimized for local use and lacks advanced scalability mechanisms like distributed databases or load balancers. For future scalability, we plan to explore vertical scaling by optimizing server resources and horizontal scaling by replicating databases. Additionally, indexing frequently queried columns can improve performance for larger datasets. Although these features are not yet implemented, the program's structure supports incremental scaling. As the project evolves, integrating cloud-based databases, like AWS RDS, can further enhance scalability. This ensures the system can handle higher loads while maintaining responsiveness. The current setup is sufficient for demonstrating the system's core functionality within the project's academic scope.

Several potential improvements have been identified for the system. Adding user authentication and role-based access would enhance security and make the program more robust. Supporting more complex queries, such as filtering nominees by specific criteria, would increase the system's functionality. Integrating with external systems, like online voting platforms, could expand its use case. Furthermore, upgrading the database management system to include automated backup and recovery processes would improve reliability. These enhancements align with the system’s long-term goals of becoming a fully functional voting platform. By prioritizing these features, the program can evolve into a production-ready application. For now, the current iteration provides a strong foundation for future growth.

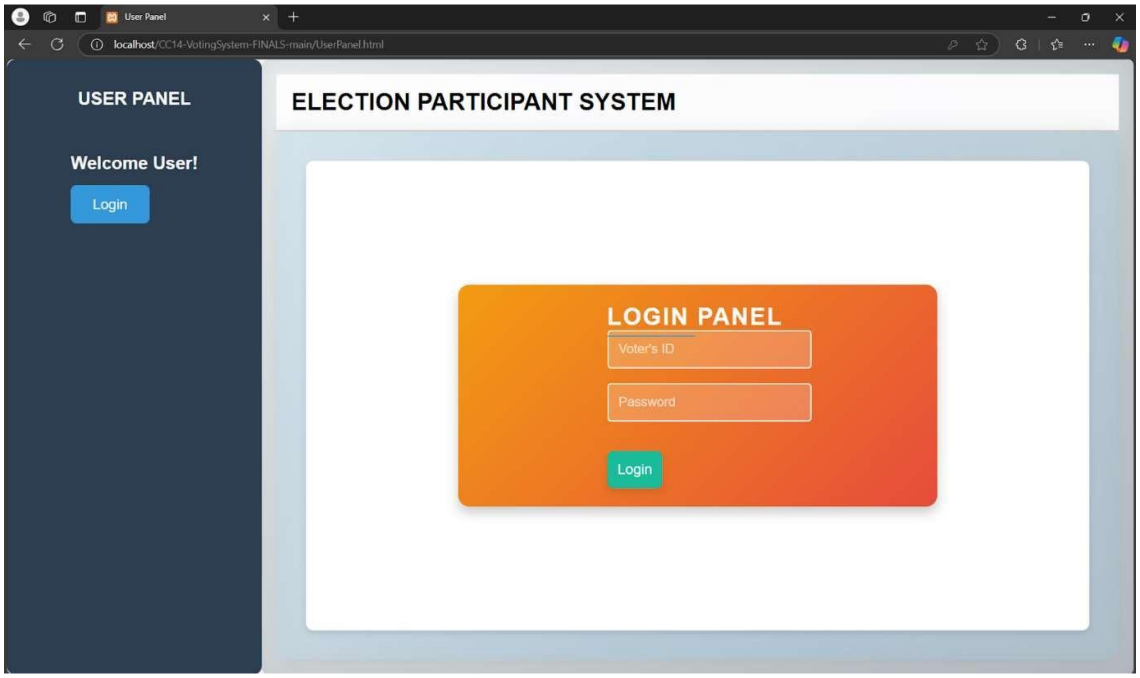
4 Conclusion

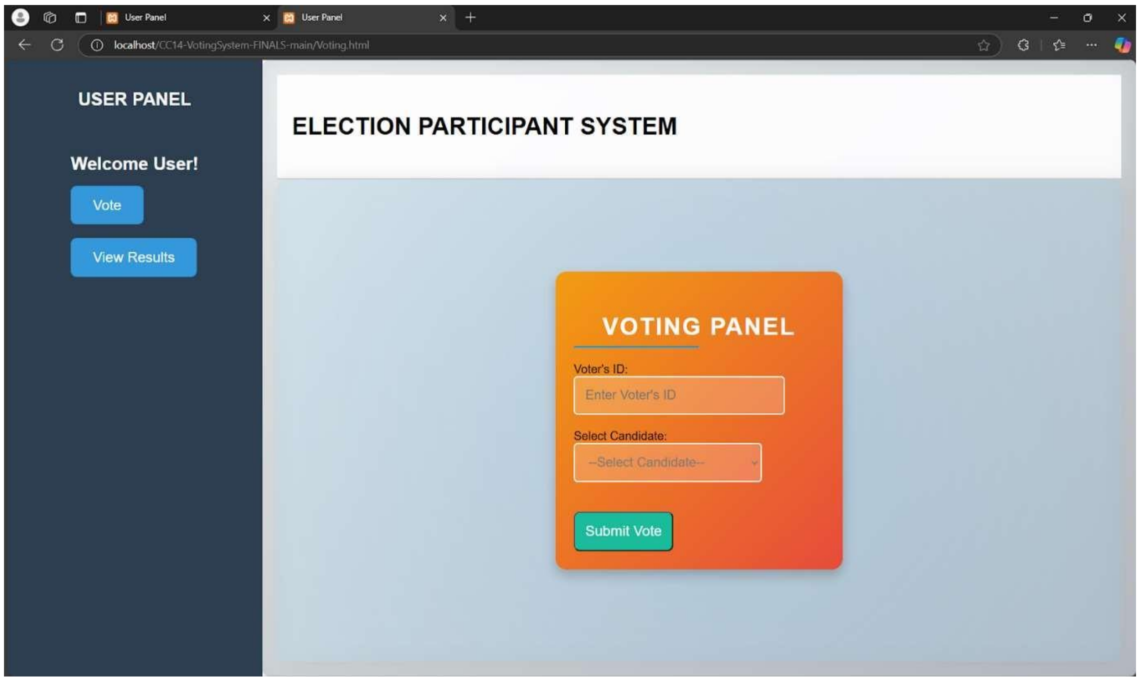
The program meets its primary objectives by providing a working prototype of a voting system. It demonstrates key functionalities, such as managing candidate data and integrating with a database, in a clear and organized manner. While limitations exist in areas like security and testing, the project effectively fulfills the academic requirements. These limitations are manageable within the scope of a learning environment and provide opportunities for future development. The current version showcases an understanding of PHP-MySQL integration and database design principles.

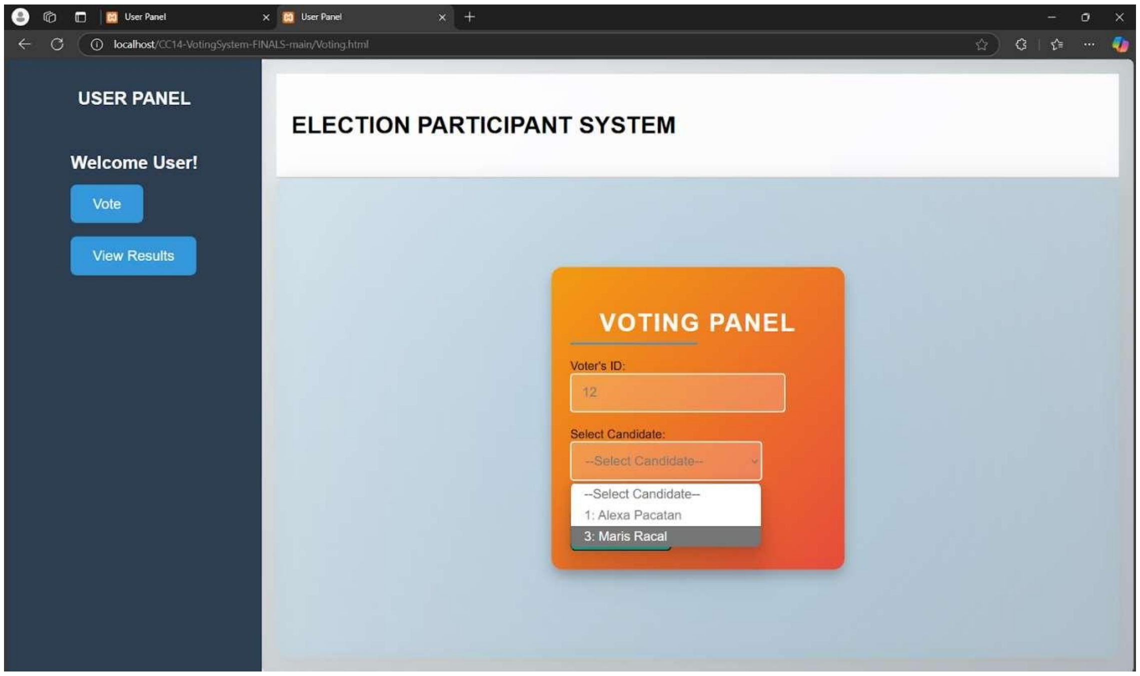
Developing the system presented several challenges, such as debugging database connections and ensuring proper data handling. These experiences deepened our understanding of software development and taught us to balance functionality with complexity. Through hands-on problem-solving, we learned to adapt to constraints and prioritize key features. The lessons learned from these challenges will guide us in future projects.

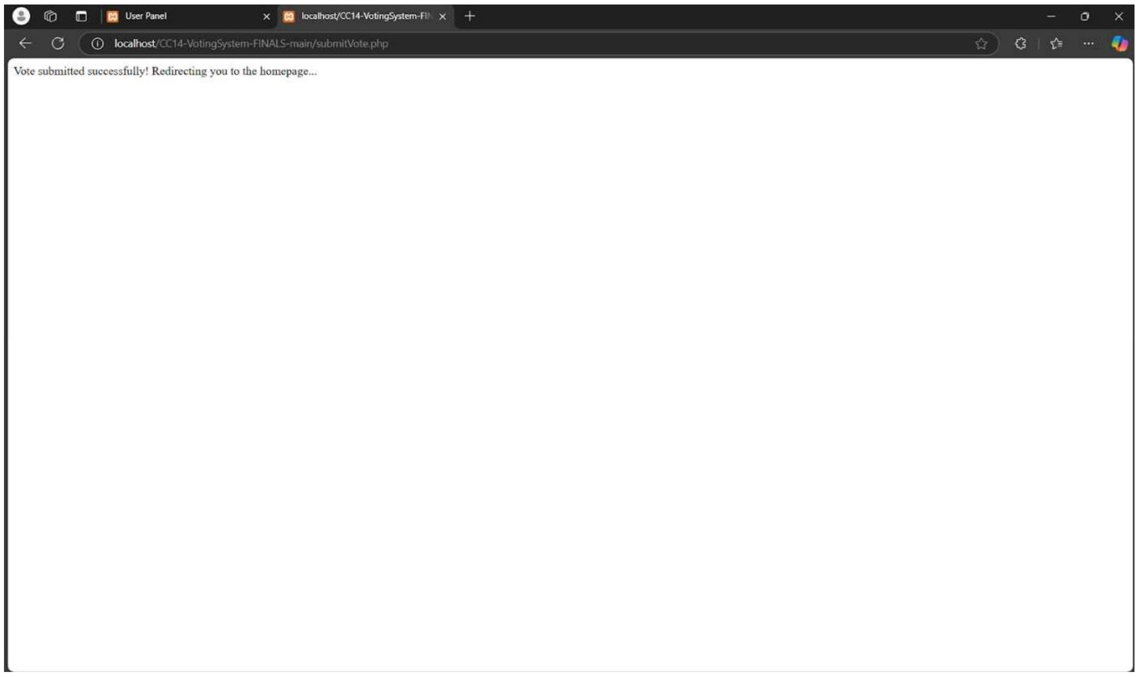
The next phase involves addressing security gaps by implementing authentication systems and testing frameworks. Additionally, scaling the system to handle larger datasets and user bases will be a priority. These improvements will ensure the system evolves into a more secure, efficient, and scalable application.

5 Appendices

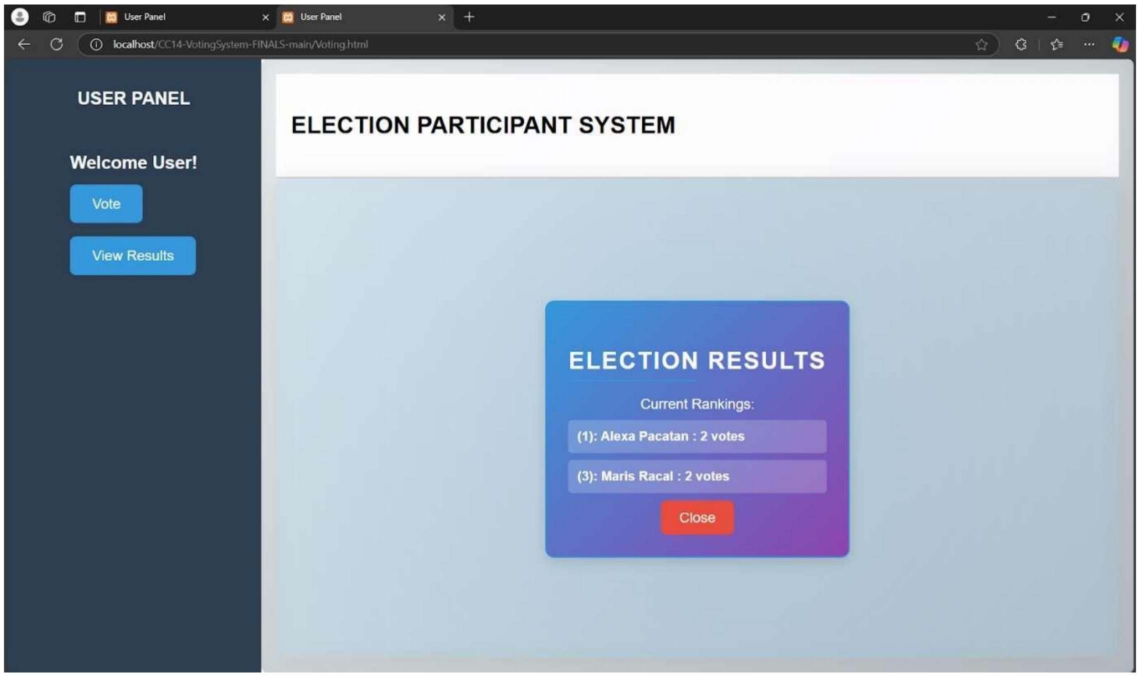

User Panel
localhost/CC14-VotingSystem-FINALS-main/UserPanel.html
USER PANEL
Welcome User!
Login
ELECTION PARTICIPANT SYSTEM
LOGIN PANEL
Voter's ID
Password
Login






User Panel
localhost/CC14-VotingSystem-FI
localhost/CC14-VotingSystem-FINALS-main/submitVote.php

```
Vote submitted successfully! Redirecting you to the homepage...
```



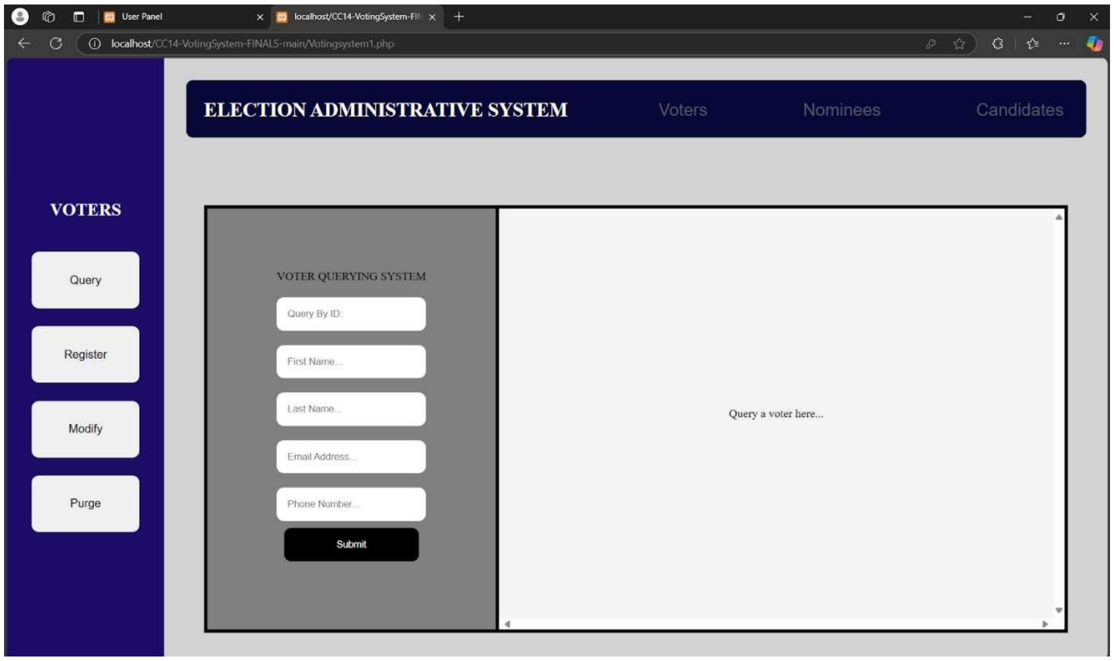
ELECTION PARTICIPANT SYSTEM

ELECTION RESULTS

Current Rankings:

(1): Alexa Pacatan : 2 votes

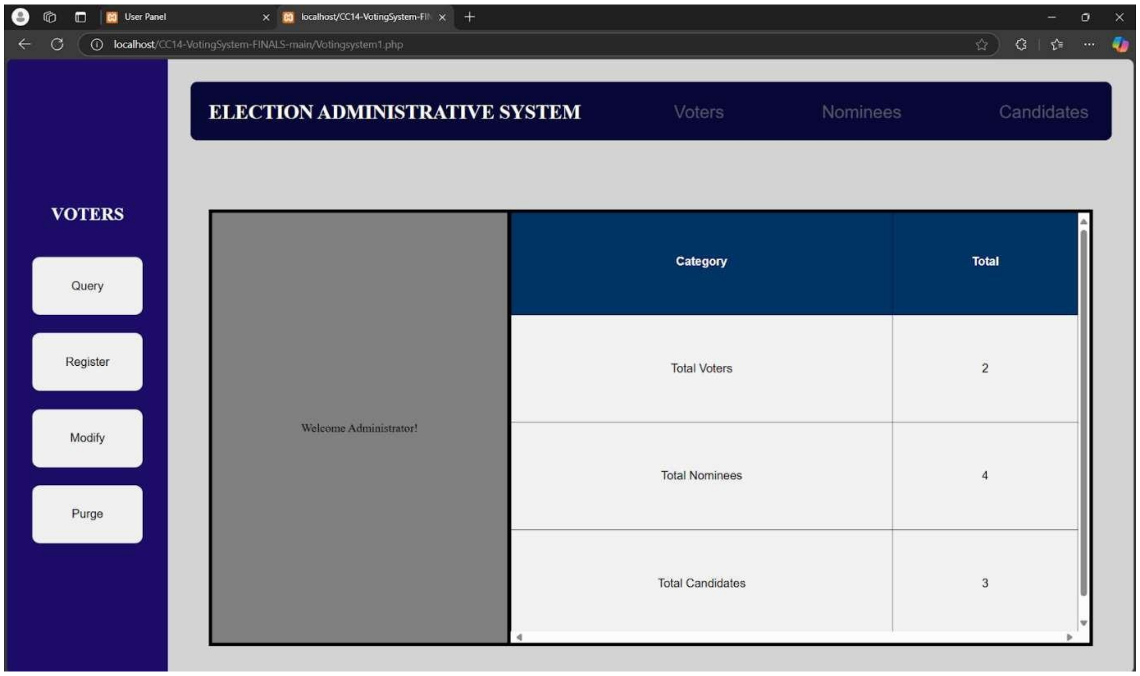
(3): Maris Racal : 2 votes



ELECTION ADMINISTRATIVE SYSTEM

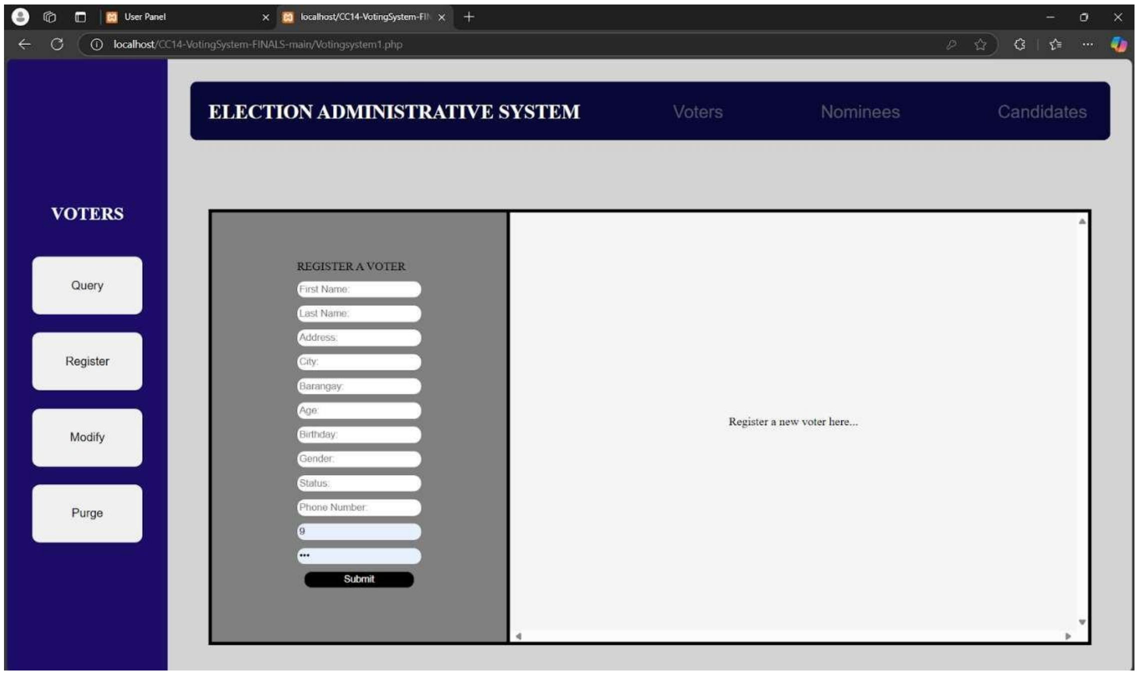
VOTER QUERYING SYSTEM

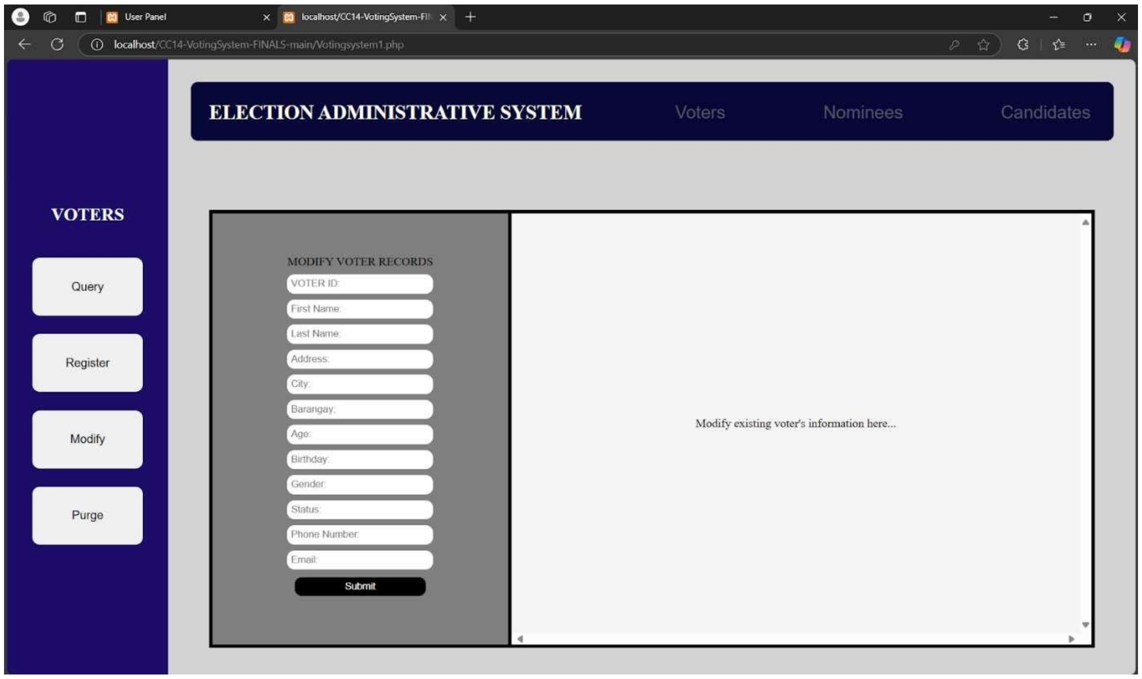
Query a voter here...

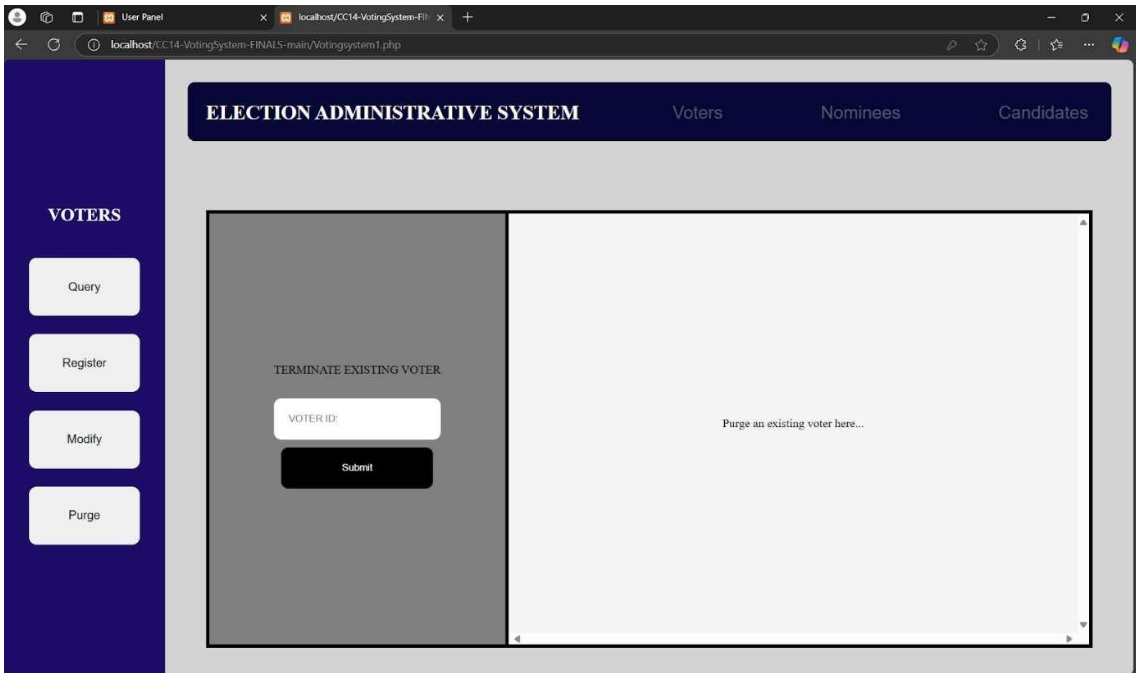


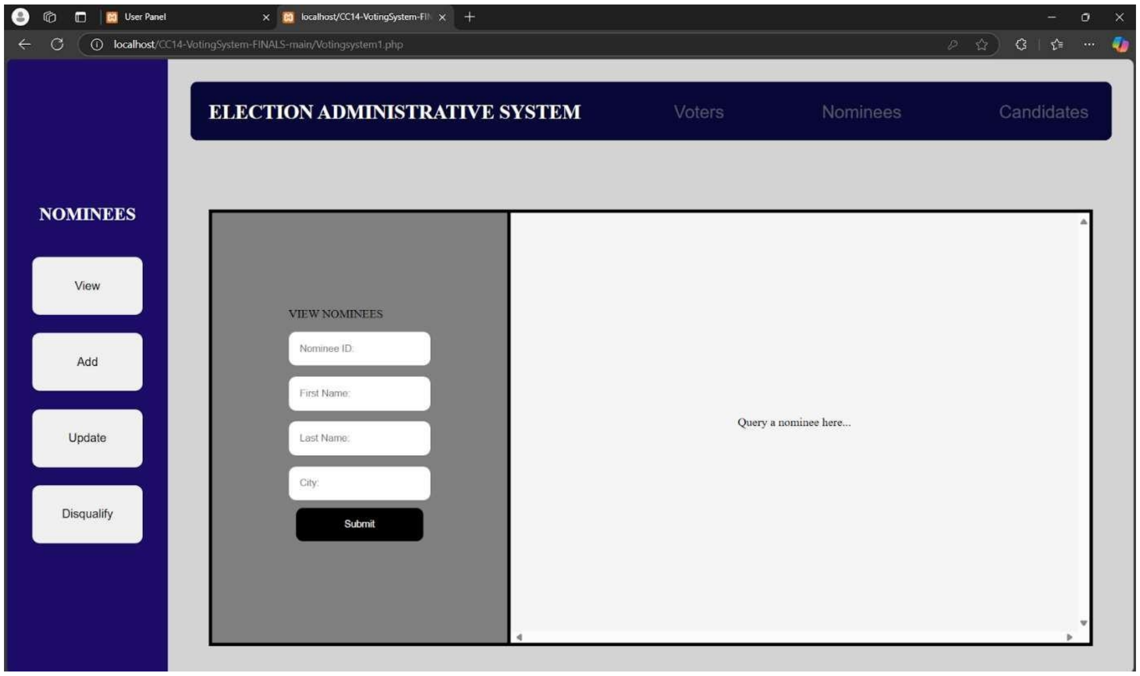
User Panel
localhost/CC14-VotingSystem-FINALS-main/Votingsystem1.php
ELECTION ADMINISTRATIVE SYSTEM
Voters
Nominees
Candidates
VOTERS
Query
Register
Modify
Purge
Welcome Administrator!

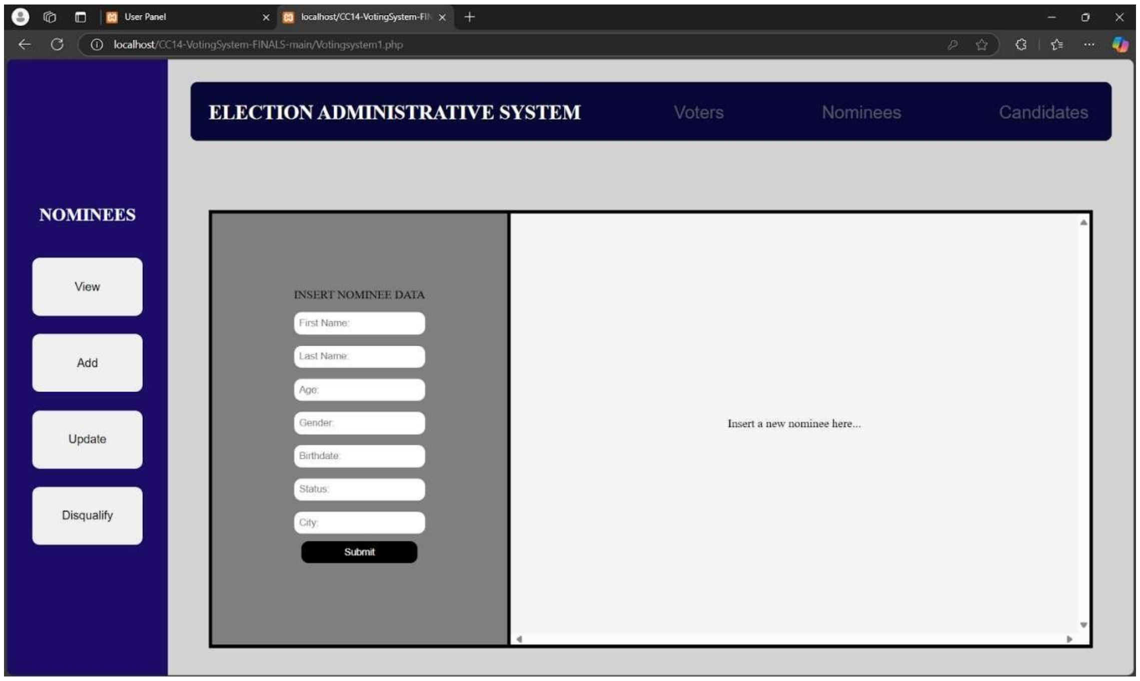
Category	Total
Total Voters	2
Total Nominees	4
Total Candidates	3

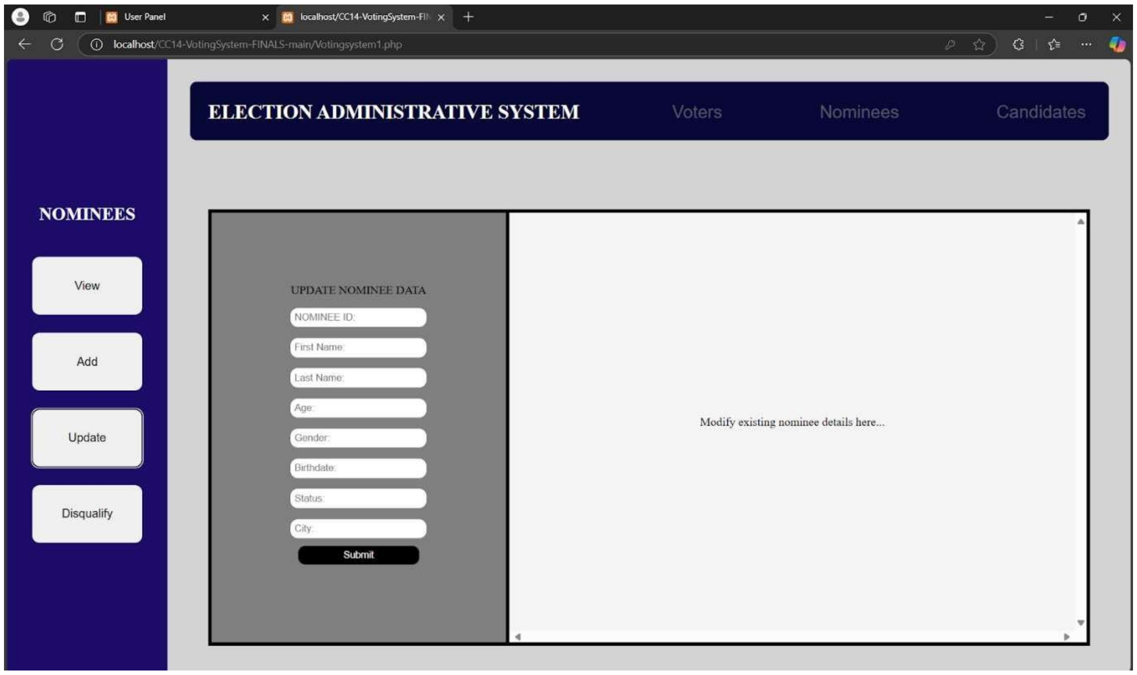



User Panel
localhost/CC14-VotingSystem-FI
localhost/CC14-VotingSystem-FINALS-main/Votingsystem1.php
ELECTION ADMINISTRATIVE SYSTEM
Voters
Nominees
Candidates
VOTERS
Query
Register
Modify
Purge
MODIFY VOTER RECORDS
VOTER ID:
First Name:
Last Name:
Address:
City:
Barangay:
Age:
Birthday:
Gender:
Status:
Phone Number:
Email:
Submit
Modify existing voter's information here...


User Panel
localhost/CC14-VotingSystem-FI
localhost/CC14-VotingSystem-FINALS-main/Votingsystem1.php
ELECTION ADMINISTRATIVE SYSTEM
Voters
Nominees
Candidates
VOTERS
Query
Register
Modify
Purge
TERMINATE EXISTING VOTER
VOTER ID:
Submit
Purge an existing voter here...


User Panel
localhost/CC14-VotingSystem-FI
localhost/CC14-VotingSystem-FINALS-main/Votingsystem1.php
ELECTION ADMINISTRATIVE SYSTEM
Voters
Nominees
Candidates
NOMINEES
View
Add
Update
Disqualify
VIEW NOMINEES
Nominee ID:
First Name:
Last Name:
City:
Submit
Query a nominee here...


User Panel
localhost/CC14-VotingSystem-FI
localhost/CC14-VotingSystem-FINALS-main/Votingsystem1.php
ELECTION ADMINISTRATIVE SYSTEM
Voters
Nominees
Candidates
NOMINEES
View
Add
Update
Disqualify
INSERT NOMINEE DATA
First Name:
Last Name:
Age:
Gender:
Birthdate:
Status:
City:
Submit
Insert a new nominee here...



Modify existing nominee details here...