VOTING & NOMINATION SYSTEM

Bajuyo, Carpizo, Pasilan, Pacatan, Valdez

INTRODUCTION

Purpose of the System

- Facilitate the nomination and voting process in communities, schools, ect...
- Ensure secure and efficient management of voter and nominee data.
- **Scope**: Administrators, Voters, Candidates, & Election Monitors

Key Features

- Candidate nomination and validation.
- Role-based user authentication (e.g., administrators, voters).
- Secure voting mechanism with real-time vote tracking.
- Election result generation and reporting for transparency.

Scalability - small to medium-scale elections, can be expanded (multi-election management, ect...)

SYSTEM REQUIREMENTS

FUNCTIONAL REQUIREMENTS

Data Retrieval - Access voter, nominee, and candidate information, count totals, and retrieve detailed profiles.

Transaction Management - Ensure reliable updates and inserts to maintain data integrity.

Reporting - Generate summaries and statistics, such as total votes cast and voter demographics.

NON-FUNCTIONAL REQUIREMENTS

Performance: Fast response time using PHP scripts and MySQL queries.

Scalability: Supports localized deployment on XAMPP.

Security: Basic input validation prevents SQL injection and XSS attacks.

Reliability: Includes error handling and regular database backups.

TECHNOLOGICAL STACK

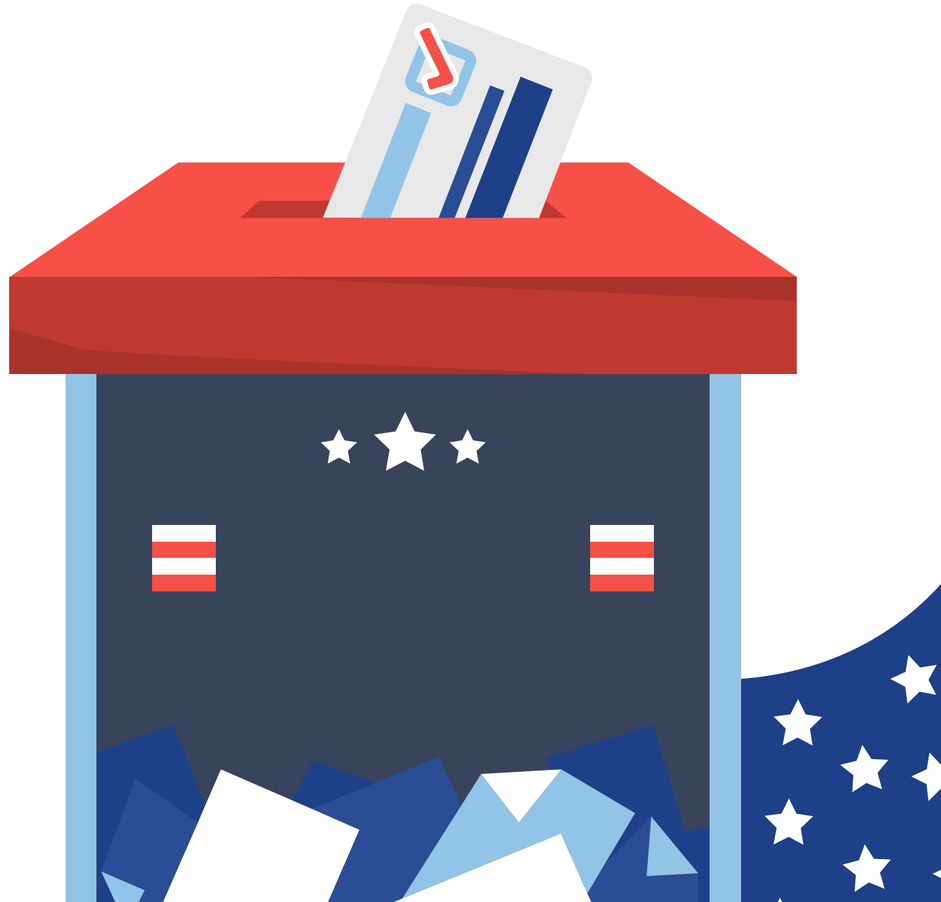
Programming Languages: PHP & SQL for backend, HTML, CSS, and JavaScript for frontend.

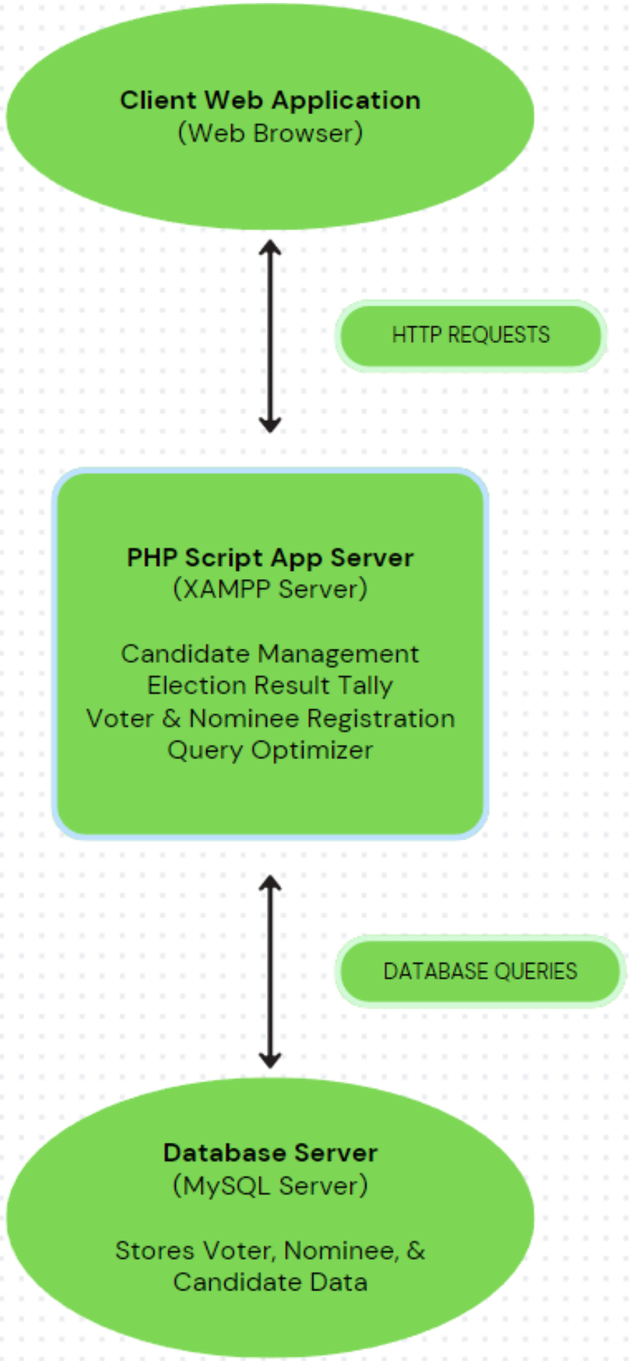
Environment: Hosted locally using XAMPP.

Tools: phpMyAdmin for database management and custom CSS for styling.

SYSTEM ARCHITECTURE OVERVIEW

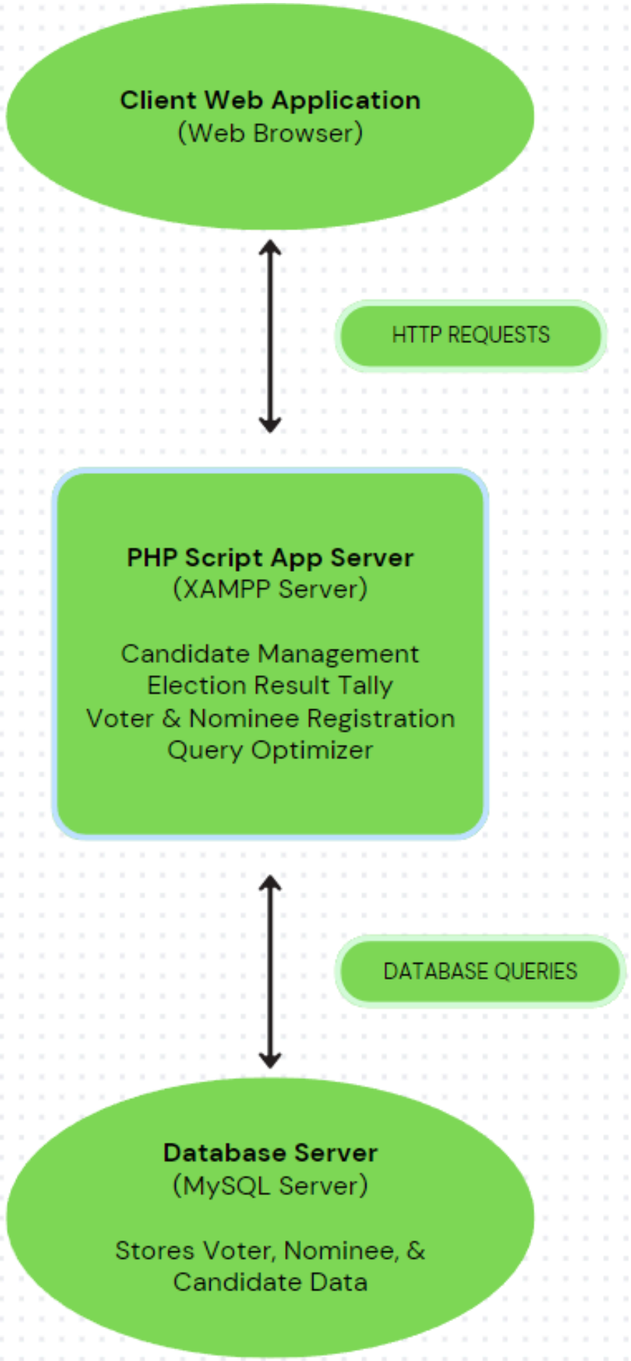
The architecture ensures **efficient communication between clients and servers** while maintaining data integrity and security within the voting system. Each component is interconnected, allowing seamless data flow and interaction throughout the system.





CLIENT WEB APPLICATION

- Interface for users (voters and admins) to register, log in, vote, and manage elections.
- Communicates with the server via HTTP requests to send/receive data like form submissions and query results.

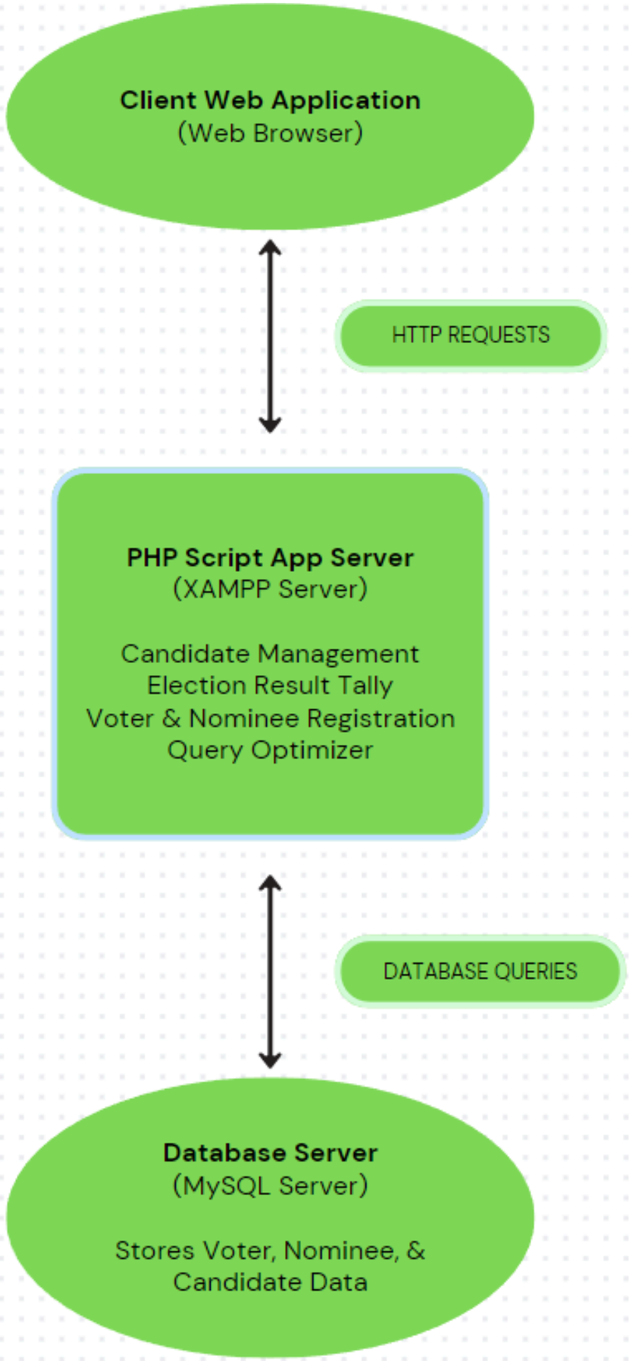


PHP SCRIPT APPLICATION SERVER

Hosted on **XAMPP**, processes client requests and manages core functionalities:

- Candidate Management: Add, edit, and delete candidates.
- Election Result Tally: Compute and display voting results.
- Voter & Nominee Registration: Ensure data integrity during the registration process.
- Query Optimizer: Execute efficient database queries.

Validates inputs, manages sessions, and interacts with the database.



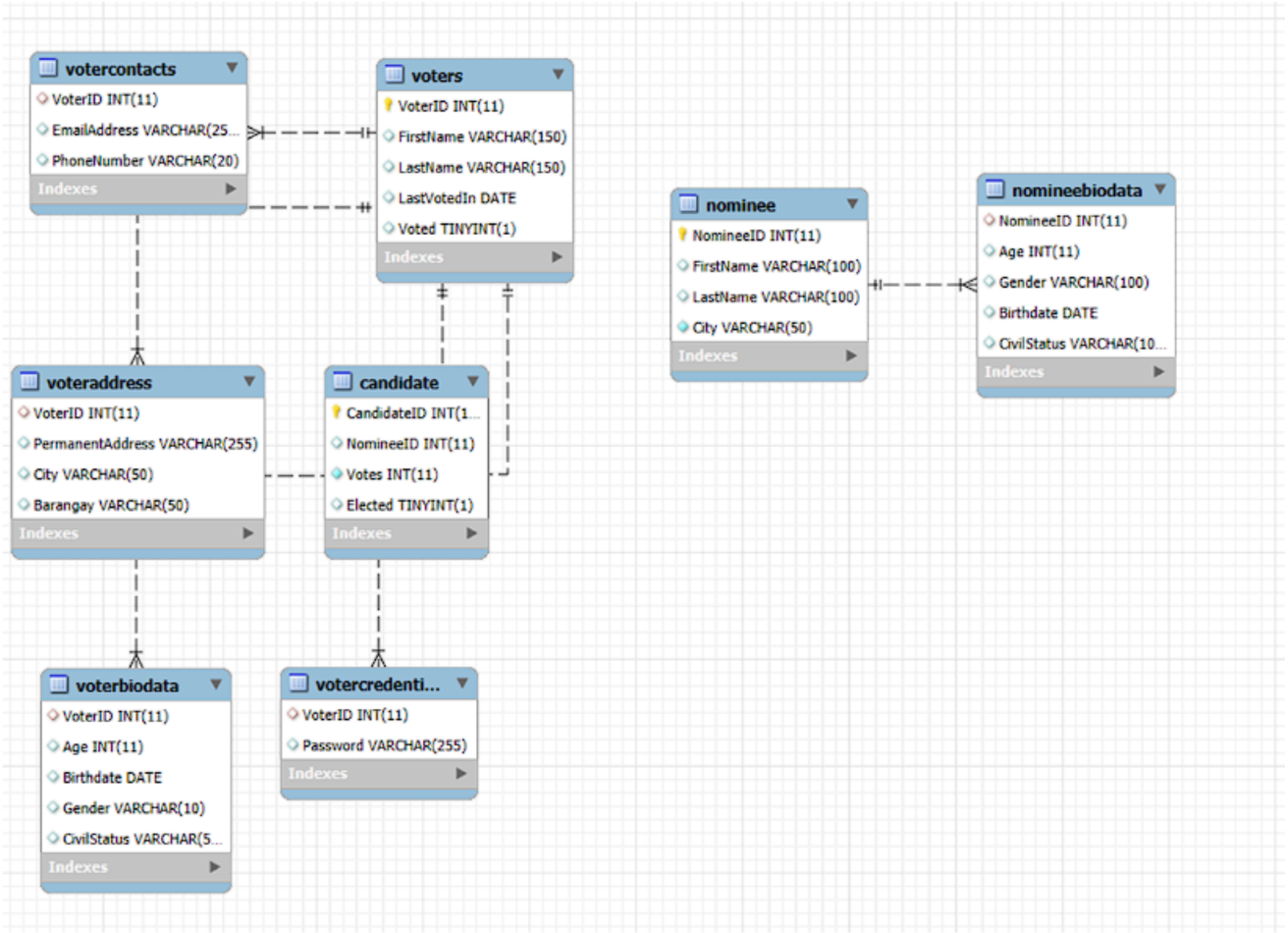
DATABASE SERVER

Stores all system data:

- Voter Data: Personal details and voting status.
- Nominee Data: Biodata and candidacy details.
- Candidate Data: Election results and vote counts.

Ensures referential integrity and enforces constraints during SQL operations.

DATA MODEL DESIGN



ENTITY RELATIONSHIP DIAGRAM

RELATIONSHIPS

Voters to Votercontacts: One-to-one relationship; each voter can have one set of contact details.

Voters to Voteraddress: One-to-one relationship; each voter can have one permanent address.

Voters to Voterbiodata: One-to-one relationship; each voter can have one biodata record.

Voters to Votercredentials: One-to-one relationship; each voter can have one set of login credentials.

Nominee to Nomineebiodata: One-to-one relationship; each nominee has one biodata record.

Candidate to Nominee: One-to-one relationship; each candidate corresponds to one nominee.

NORMALIZATION

1NF

Ensures **all attributes are atomic** (indivisible), and each table has a primary key for unique identification.

2NF

Eliminates partial dependencies by ensuring all non-key attributes are fully dependent on the primary key. Related data (e.g., voter contact, address, and biodata) is stored in separate tables linked by foreign keys.

3NF

Removes transitive dependencies by separating data into logical tables, such as splitting biodata into *Nomineebiodata* and *Voterbiodata*. Each attribute is directly related only to its primary key.

CONSTRAINTS

Primary Keys: Ensure unique identification of records (e.g., **VoterID**, **NomineeID**, **CandidateID**).

Foreign Keys: Maintain relationships between tables (e.g., NomineeID in Nomineebiodata references Nominee).

Unique Constraints: Prevent duplicate values (e.g., EmailAddress in Votercontacts).

Not Null Constraints: Ensure critical fields (e.g., FirstName, LastName) cannot be empty.

Default Values: Apply defaults where needed (e.g., Voted defaults to 0).

Check Constraints: Validate data integrity (e.g., valid values for Gender).

Table Definitions

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'
NomineePosition	NomineeID	INT	FOREIGN KEY (NomineeID)	None
	Position	VARCHAR(100)	NOT NULL	None
	Party	VARCHAR(100)	NOT NULL	None
	City	VARCHAR(100)	NOT NULL	None
Candidate	CandidateID	INT	PRIMARY KEY	None
	NomineeID	INT	FOREIGN KEY (NomineeID)	None
	Votes	INT	NOT NULL	None
	Elected	BOOLEAN	NOT NULL	None

INDEXES, VIEWS & STORED PROCEDURES

INDEXES

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.

VIEWS & STORED PROCEDURES

The voting system **uses direct SQL queries** within PHP scripts to interact with the database. **Implementing views could improve efficiency** by simplifying frequent queries like voter or nominee counts. Adding views and stored procedures would enhance maintainability and performance.

ACID PROPERTIES IN THE VOTING SYSTEM

ATOMICITY

Transactions are treated as indivisible units. *For example*, when confirming a candidate, the system ensures that the entire operation succeeds or fails entirely. If an error occurs during insertion, the transaction is rolled back to maintain data integrity.

CONSISTENCY

The database remains in a valid state before and after a transaction. Updates, such as modifying nominee information, are validated to ensure compliance with rules and constraints, preventing invalid data from being saved.

ACID PROPERTIES IN THE VOTING SYSTEM

ISOLATION

Concurrent transactions are executed independently. When two users modify nominee details simultaneously, isolation ensures one transaction completes before the other begins, avoiding conflicts like dirty reads or lost updates.

DURABILITY

Once a transaction is committed, its changes are permanent. For instance, after inserting a candidate, the system ensures the data persists even after unexpected failures, using mechanisms like write-ahead logging.

CONCURRENCY CONTROL

The voting system ensures **data integrity** and **consistency** during simultaneous transactions using lock-based protocols. For example, when updating nominee information, exclusive locks are applied to prevent other transactions from reading or modifying the data until the update is complete. This prevents issues like dirty reads or lost updates. SQL queries, such as counting voters with SELECT COUNT(*) AS total_voters FROM Voters, are executed efficiently to manage concurrent operations, ensuring that the system remains responsive even with multiple users accessing it simultaneously. This approach balances reliability and system performance in a multi-user environment.

ERROR HANDLING

Robust error handling ensures reliable transactions throughout the voting system. For instance, if a database connection fails, an error message is displayed using die("Connection failed: " . $conn->connect_error);, alerting the user to the issue. Additionally, failed SQL queries trigger rollback mechanisms to maintain data integrity. Transaction blocks are implemented using begin_transaction(), rollback(), and commit(), ensuring that either all changes succeed or none are applied. *For example,* if an insertion of a new candidate fails after some updates have been made, the system will roll back all changes to prevent partial updates and inconsistencies.

SECURITY CONSIDERATIONS

ACCESS CONTROL

Authentication and role-based permissions restrict access to sensitive operations. Administrators manage candidate data, while password encryption and session timeouts enhance security. Future iterations may include two-factor authentication **(2FA)** for added protection.

DATA ENCRYPTION

Sensitive information is **securely stored within the local environment**, minimizing risks of unauthorized access. The controlled setup supports secure user authentication and record management.

BACKUP AND RECOVERY

The system uses GitHub for version control, ensuring all code, configurations, and schemas are securely stored and recoverable. GitHub’s distributed versioning allows for collaboration and quick restoration of stable states. Future enhancements may include automated database snapshots to strengthen recovery options further.

This synthesized content provides a concise overview of key aspects of concurrency control, error handling, security considerations, and backup strategies within the voting system code.

CODE DEMO

CONCLUSION

Overall, our project successfully delivers a functional prototype of a voting system, showcasing key features like candidate data management and PHP-MySQL integration, while addressing academic requirements despite manageable limitations in security and testing. We learnt lessons from overcoming challenges such as debugging and data handling have enhanced our software development skills, setting a foundation for future improvements like implementing better authentication, testing frameworks, and scaling for larger datasets.

THANKS!