Election Voting Machine and Voter Verified Paper Audit Trail System

OOPS End Term Project

Prof Kiran.D.C

Submitted on: 5th May 2025

Vaidhav Adit

Cohort 2, UEN: 136

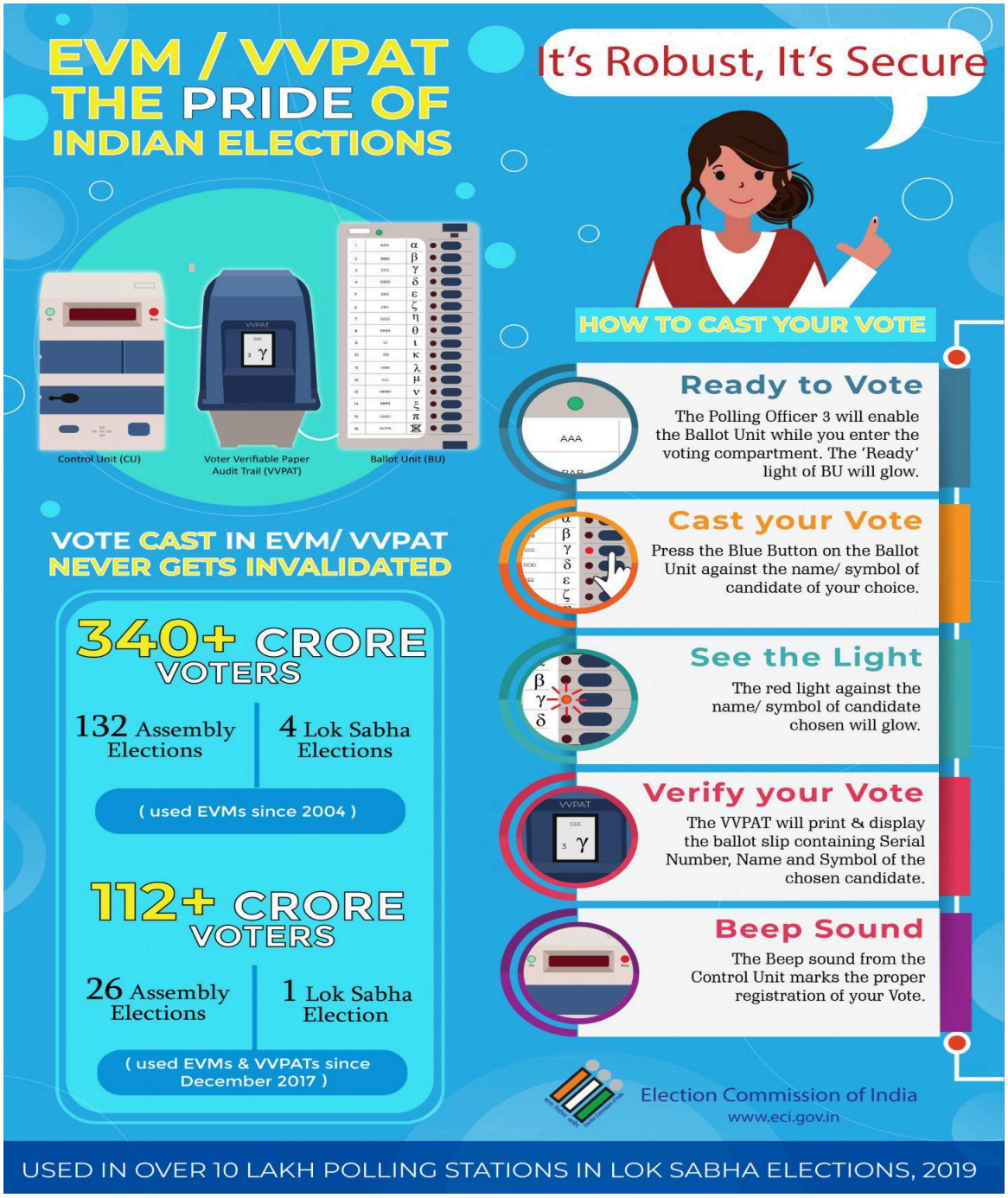
Problem Statement:

India's electronic voting system, using EVMs and VVPATs, is essential for conducting secure and efficient elections. However, the actual system is complex and not easily accessible for public understanding, testing, or demonstration. This creates a need for a digital model that not only simulates the full electoral process but also reflects how such systems could be implemented in real-world environments.

Understanding voting systems is key to building public trust, improving transparency, and ensuring that democratic processes are both secure and inclusive. Beyond testing and education, a well-designed software model can help visualize how the real system functions and serve as a base for developing future, fully functional implementations.

This project delivers a Java-based simulation of India's voting process, organized into distinct packages: Main for GUI and application flow, Users for roles like Admin, Voter, and ElectionOfficer, System for core logic, EVM, VVPAT, and database operations, and exceptions for robust error handling. It supports secure voter verification, vote casting, and paper slip generation, ensuring tamper-proof recording and auditable logs. Designed to reflect real-world voting logic, this simulation serves as an educational tool and a potential foundation for practical deployment in controlled settings.

How does the EVM System work in India: (Source: https://www.eci.gov.in/evm/)



HOW TO CAST YOUR VOTE USING

EVM & VVPAT

1

ENTER THE BOOTH

The Presiding Officer will enable the Ballot Unit while you enter the Polling Compartment.

2

CAST YOUR VOTE

Press the blue button on the Ballot Unit against the name/symbol of candidate of your choice.

3

SEE THE LIGHT

The red light against the name/symbol of candidate chosen will glow.

4

SEE THE PRINT

A Ballot slip having Serial no., Name & Symbol of the chosen candidate will be visible for 7 seconds & will drop in the VVPAT box.

Note

If you do not see Ballot slip and hear the beep sound, please contact the Presiding Officer.

ELECTION COMMISSION OF INDIA

URL : https://eci.nic.in

Requirements:

System Components and Roles

- The system must include packages: Main for GUI and flow, Users for roles, System for core logic, and exceptions for error handling.
- The System package must contain EVM for candidate management, voting, and vote storage, VVPAT for vote confirmation slips, and ElectionCommission for voter oversight.
- The Users package must define roles: Admin (manages candidates/results), Voter (casts votes), and ElectionOfficer (verifies voters), with ElectionRole and DataManager interfaces.
- The Main package's ElectionSystem must orchestrate GUI (Swing), user interaction, and inter-package operations.

Voter Registration

- ElectionCommission in the System package must store voters in a HashMap using unique voter IDs.
- Voter registration must accept name, age, 12-digit Aadhaar, and address, supporting full or half registration via constructor overloading in Voter.
- The system must reject voters under 18 using InvalidAgeException and allow removal based on name, Aadhaar, and reason (e.g., duplicate, demise) with RemoveVoterException.
- ElectionCommission must display all registered voters' details (ID, name, age, Aadhaar, address) or indicate if none are registered.

Voter Verification

- The ElectionOfficer in the Users package must verify voters, checking age (≥18), voter list presence, Aadhaar length (12 digits), and constituency match.
- Successful verification must allow voting; failure must throw VerifyVoterException and block voting with an error message.
- Verification logic must interact with ElectionCommission in the System package for voter data access.
- Errors during verification must be caught and displayed without crashing the system.

Voting Process

- EVM in the System package must list candidates (number, name, party, symbol) using an Iterator.
- Voter must select a candidate by number, with EVM validating the range, rejecting invalid votes with InvalidVoteException.
- Each voter must vote once; duplicate attempts must be rejected with InvalidVoteException.
- If the candidate list is empty, voting must be blocked with an error message.

Valid Vote Actions

- A valid vote must increment the candidate's vote count in EVM.
- EVM must blink a red light and beep to confirm the vote.
- VVPAT in the System package must generate, display, and print a slip with candidate name, party, timestamp, and a newline.
- Votes must be logged anonymously (e.g., "Voter X") with candidate details and timestamp in EVM.

Candidate Management

- Admin in the Users package must add candidates (name, symbol, party), rejecting duplicates from the same party with SamePartyCandidateException.
- Admin must remove candidates by name and party, confirming success or reporting failure with CandidateNotFoundException.
- Candidate data must be managed via EVM in the System package.
- ElectionSystem in Main must provide GUI options for these operations.

Admin Access

- Admin must use a password to access settings or results, with failure throwing AdminFailException.
- The Admin menu in the GUI (Main package) must include options to add/remove candidates, view results, and exit.
- Access logic must interact with EVM in the System package for authentication.
- Errors must be handled using exceptions from the exceptions package.

Vote Storage

- EVM in the System package must store votes in a List, writing them to "votes.csv" using BufferedWriter.
- The CSV must include headers and entries with anonymized ID, candidate name, party, and timestamp, excluding voter personal data.
- Vote writing must confirm completion with a message in the GUI (Main package).
- IOException and custom exceptions from the exceptions package must handle errors during writing.

Election Results

- ElectionSystem in the Main package must display vote counts for all candidates (name, party, votes) via EVM.

- Admin must view results with additional symbol details, interacting with EVM in the System package.
- The system must identify the winner (most votes) or declare a draw, listing tied candidates.
- Results must be presented through the GUI with error handling for edge cases.

System Interaction

- The Main package's GUI must offer a menu: Admin, Voter, Election Commission, View Results, End Election.
- Voter interaction must prompt for ID, verify via ElectionOfficer, and allow voting if successful, using System package components.
- ElectionCommission interaction must offer options to register/remove voters, view all voters, and exit, displayed via GUI.
- Ending the election must save votes to CSV via EVM and stop the process.

Error Handling and Setup

- Exceptions from the exceptions package must display error messages without crashing the system.
- The system must start by prompting for the constituency name in the Main package's GUI.
- Initial setup in ElectionSystem must pre-register four candidates and ten voters via System package classes.
- DatabaseManager in the System package must handle persistent storage and retrieval for voters and candidates.

External Modules Imported:

The ElectionSystem leverages external Java modules for efficient data management, user interaction, file operations, timestamping, GUI functionality, and database connectivity. The Main package uses Swing and AWT for the GUI, while the System package integrates JDBC for database operations. Collections like ArrayList and HashMap in the Users and System packages handle dynamic data, and exceptions package exceptions ensure robust error handling, enhancing the system's reliability and user experience.

1. **java.util.ArrayList**
    - A resizable array implementation for dynamic storage of elements.
    - Supports efficient addition, removal, and random access of items.
    - Ideal for managing variable-sized collections like candidates.
    - Used in System.EVM.candidateList to store and manipulate candidates.
2. **java.util.HashMap**
    - A hash table-based implementation for key-value pair storage.
    - Provides fast lookups, insertions, and deletions with unique keys.
    - Perfect for mapping voter IDs to Voter objects efficiently.
    - Employed in System.ElectionCommission.voterList for voter management.
3. **java.util.Iterator**
    - An interface for traversing collections in a standardized way.
    - Enables sequential access to elements without exposing internals.
    - Useful for iterating over collections like vote buffers or candidate lists.
    - Applied in System.EVM.displayCandidates and writeVotesToCSV to process data.
4. **java.util.List**
    - An interface representing an ordered collection of elements.
    - Allows duplicate elements and provides positional access.
    - Suitable for storing sequences like ballot records dynamically.
    - Utilized in System.EVM.voteBuffer to log and export votes.
5. **java.util.Scanner**

- A utility class for reading input from various sources.
- Simplifies parsing of strings, integers, and other data types.
- Enables user interaction for admin or voter inputs in console mode.
- Integrated in Main.ElectionSystem for fallback console-based operations.

6. **java.time.LocalDateTime**
   - A class for representing date and time without a timezone.
   - Provides precise timestamping for events or records.
   - Essential for tracking ballot creation times accurately.
   - Used in System.EVM.Ballot.timestamp and System.VVPAT for vote timestamps.
7. **java.io.BufferedWriter**
   - A buffered output stream writer for efficient file writing.
   - Reduces I/O operations by buffering data before writing.
   - Optimizes large-scale data export to files.
   - Applied in System.EVM.writeVotesToCSV for vote data export.
8. **java.io.FileWriter**
   - A class for writing character data to files.
   - Handles the low-level file output operations.
   - Supports appending or overwriting file content.
   - Used in System.EVM.writeVotesToCSV to save votes to "votes.csv".
9. **java.io.IOException**
   - An exception class for I/O operation failures.
   - Indicates errors like file access or write issues.
   - Essential for handling potential file operation errors.
   - Employed in System.EVM.writeVotesToCSV for error management.
10. **java.sql.Connection**
    - An interface for establishing a database connection.
    - Facilitates communication between Java and the database server.
    - Essential for executing SQL queries and updates.

- Used in System.DatabaseManager to connect to the MySQL database.

11. **java.sql.DriverManager**
    - A class for managing database driver connections.
    - Registers drivers and establishes database connections.
    - Critical for initiating database access in the application.
    - Utilized in System.DatabaseManager.getConnection for database connectivity.
12. **java.sql.PreparedStatement**
    - An interface for executing parameterized SQL queries safely.
    - Prevents SQL injection by using placeholders for input values.
    - Enhances efficiency for repeated query execution.
    - Applied in System.DatabaseManager for secure database operations.
13. **java.sql.ResultSet**
    - An interface for retrieving and manipulating query results.
    - Provides methods to iterate over database query result rows.
    - Essential for fetching voter, candidate, and vote data.
    - Used in System.DatabaseManager to retrieve database records.
14. **java.sql.SQLException**
    - An exception class for database operation failures.
    - Indicates errors like connection issues or query failures.
    - Crucial for handling database-related errors gracefully.
    - Employed in System.DatabaseManager for error management.
15. **java.sql.Timestamp**
    - A class for representing SQL TIMESTAMP with nanosecond precision.
    - Enables accurate storage and retrieval of date-time in databases.
    - Useful for logging vote timestamps in the database.
    - Used in System.DatabaseManager.VoteRecord to handle vote timestamps.
16. **javax.swing.** (Swing Components)
    - A package providing GUI components for building interactive interfaces.

- Includes classes like JFrame, JPanel, JButton for user interaction.
- Enables a user-friendly graphical interface for the election system.
- Utilized in Main.ElectionSystem to create the GUI for all operations.

17. **java.awt.** (AWT Components)
    - A package for foundational GUI components and event handling.
    - Provides classes like CardLayout, BoxLayout, ActionListener for layout and interaction.
    - Supports the structural and behavioral aspects of the GUI.
    - Applied in Main.ElectionSystem for GUI layout and event management.

Exceptions:

1. **SamePartyCandidateException**:
    - Handles errors when a candidate from the same party is added, ensuring party uniqueness in EVM.addCandidate.
    - Defined in the exceptions package for centralized error handling.
    - Prevents multiple candidates from the same party in a constituency.
    - Triggered during candidate addition in the Users.Admin class.
2. **AdminFailException**:
    - Manages authentication failures for admin actions, triggered in EVM.verifyAdmin if the password is incorrect.
    - Resides in the exceptions package for consistent error management.
    - Alerts the Election Commission of unauthorized access attempts.
    - Used in Users.Admin.accessSystem to secure admin operations.
3. **CandidateNotFoundException**:
    - Occurs when a requested candidate isn't found, used in EVM.castVote or Admin.removeCandidate.
    - Part of the exceptions package for uniform exception handling.

- Ensures robust error reporting during candidate operations.
- Also applies in System.DatabaseManager.removeCandidate for database checks.

4. **InvalidVoteException**:
    - Signals invalid vote attempts, such as duplicate votes, handled in EVM.castVote.
    - Located in the exceptions package for streamlined error processing.
    - Blocks voting if the candidate list is empty or index is invalid.
    - Utilized in Users.Voter.castVote to enforce voting rules.
5. **InvalidAgeException**:
    - Detects underage voters, thrown during ElectionOfficer.verifyVoter age checks.
    - Defined in the exceptions package for cohesive error handling.
    - Ensures compliance with the legal voting age of 18.
    - Also used in System.DatabaseManager.addVoter for database validation.
6. **RemoveVoterException**:
    - Indicates failures in removing voters, used in ElectionCommission.removeVoter.
    - Housed in the exceptions package for consistent exception management.
    - Handles cases like invalid removal reasons or non-existent voters.
    - Supports duplicate voter detection in the System package.
7. **VerifyVoterException**:
    - Manages errors in voter verification, triggered in ElectionCommission.verifyVoter if data is invalid.
    - Part of the exceptions package for standardized error handling.
    - Checks Aadhaar length, constituency match, and voter existence.
    - Used in Users.ElectionOfficer.verifyVoter to ensure voting eligibility.

Interfaces

1. Interface Name: ElectionRole

The ElectionRole interface defines a contract for user roles interacting with the election system. It ensures that all user classes (e.g., Voter, Admin, ElectionOfficer, ElectionCommission) in the Users and System packages implement a standardized method for accessing system functionalities, promoting consistency and modularity across different roles.

a. Attributes:
    i. None
        This interface does not define attributes, focusing solely on behavioral contracts for role-based system access.
b. Methods:
    i. accessSystem(EVM evm, Scanner scan):
        An abstract method that specifies how a role interacts with the system, taking an EVM for voting operations and a Scanner for user input. Each implementing class must define this method based on the role's responsibilities, such as voting for a Voter or managing candidates for an Admin.

2. Interface Name: DataManager

The DataManager interface establishes a contract for classes managing election data, such as voter or candidate information. Implemented by Admin and ElectionCommission in the Users and System packages, it ensures consistent data management practices across the system, enhancing maintainability and scalability.

a. Attributes:

i. None

This interface focuses on defining data management behavior without specifying attributes, leaving implementation details to the classes.

b. Methods:

i. manageData(EVM evm):

An abstract method that defines how a class handles election data, taking an EVM to access candidate or vote data. Implementing classes like Admin use this to add candidates, while ElectionCommission uses it to manage voter data, ensuring a unified approach to data operations.

Classes and Framework:

1. Class Name: User

The User class serves as the abstract base class for all system users, residing in the Users package. It establishes shared functionality and data, such as storing and retrieving user names, ensuring consistency across roles. All specialized user roles like Voter, Admin, and ElectionOfficer inherit from this class.

a. Attributes:

i. userName:

A String that holds the name of the user. This uniquely identifies users during interactions with the system and is inherited by all subclasses.

b. Methods:

i. User(String userName):

Initializes a new user with a given name, setting the userName attribute for identification across the system.

ii. accessSystem(EVM evm, Scanner scan) (abstract):

An abstract method that defines a contract for how different users interact with the system. Each subclass must implement this method based on the role's specific responsibilities such as casting votes for a Voter or managing results for an Admin.

iii. getUserName():

Returns the userName of the user. This method allows any class to retrieve and display the current user's name for logs, UI prompts, or verification.

2. Class Name: ElectionStaff

The ElectionStaff class, in the Users package, serves as an abstract base class for election-related staff roles, providing common functionality for managing staff members. It inherits from the User class, enabling identification and management of staff within the election process.

a. Attributes
   i. staffID: A unique ID given to each staff member, created automatically using a counter.
   ii. sIDcounter: A static counter used to generate unique staff IDs, initialized to 100.
   iii. userName (inherited): The name of the staff user. It helps identify who is using the system.

b. Methods
   i. ElectionStaff(...):

   This method initializes a new staff member with a name and automatically generates a unique staff ID using a counter. It keeps the staff identification consistent.

   ii. getStaffID():

   Used to retrieve the staff member's unique ID. The admin can use this to check or verify staff identities.

   iii. accessSystem(...) (inherited)

   This is the entry point for the staff member. It provides a framework for system interaction, though specific implementation is handled by subclasses.

   iv. getUserName() (inherited):

The staff can use this method to retrieve their name. This is helpful for system logs or showing staff information.

3. Class Name: Admin

The Admin class handles all administrative tasks during the election. It allows the admin to manage candidates, view election results, and access protected parts of the system. This class inherits from ElectionStaff, which extends the base User class.

a. Attributes
   i. adminPassword: Stores the password required for admin login. It helps secure access to admin features.
   ii. staffID (inherited): A unique ID given to each admin, created automatically using a counter.
   iii. userName (inherited): The name of the admin user. It helps identify who is using the system.

b. Methods
   i. addCandidate(...)
      This method lets the admin add a new candidate to the election. It checks if the party already has a candidate and throws an error if a duplicate is found. This keeps the election fair.
   ii. removeCandidate(...)
      Used to remove a candidate from the election. The method looks for a match by name and party. If the candidate isn't found, it shows an error message.
   iii. viewResults(...)
      Displays the current vote count for each candidate. The admin can use this to check or announce results.
   iv. accessSystem(...)

This is the entry point for the admin. It asks for a password and, if correct, gives access to features like adding candidates, removing them, or viewing results.

v. manageData(...)
Allows the admin to manage election data, such as adding a new candidate as an example of data management functionality.

vi. getStaffID() (inherited)
The admin can use this method to retrieve their unique ID. This is helpful for system logs or showing admin information.

vii. getUserName() (inherited)
The admin can use this method to retrieve their name. This is helpful for system logs or showing admin information.

4. Class Name: Voter

The Voter class, in the Users package, represents individuals who cast votes in the election, enforcing voting restrictions like single-vote rules. It inherits from User and implements the ElectionRole interface for system interaction.

a. Attributes
   i. voterID: A unique ID given to each voter, created automatically using a counter.
   ii. vIDcounter: A static counter used to generate unique voter IDs, initialized to 100.
   iii. hasVoted: Indicates whether the voter has already cast a vote.
   iv. voterAge: The age of the voter, used for verification.
   v. aadhaarNumber: The 12-digit Aadhaar number for voter identification.
   vi. address: The address of the voter, used to verify constituency.
   vii. userName (inherited): The name of the voter user. It helps identify who is using the system.

viii. voterDbId: An int storing the voter's database ID, linking the voter to database records for persistence.

b. Methods

i. Voter(...)

This method initializes a new voter with name, age, Aadhaar number, and address. It keeps voter identification and details consistent.

ii. Voter(...)

This overloaded constructor method initializes a new voter with name and Aadhaar number, using default values for age and address.

iii. castVote(...)

This method lets the voter cast a vote interactively using a Scanner, validating the choice and ensuring no duplicate votes.

iv. castVote(...)

This overloaded method allows the voter to cast a vote directly by specifying a candidate index, validating the choice and ensuring no duplicate votes.

v. accessSystem(...)

This is the entry point for the voter. It provides a framework for system interaction, specifically for voting.

vi. getVoteStatus()

Used to retrieve whether the voter has voted. The system can use this to enforce voting restrictions.

vii. getVoterID()

The voter can use this method to retrieve their unique ID. This is helpful for system logs or verification.

viii. getVoterAge()

The voter can use this method to retrieve their age. This is helpful for verification purposes.

ix. getAadhaarNumber()

The voter can use this method to retrieve their Aadhaar number. This is helpful for identification.

x. getAddress()

The voter can use this method to retrieve their address. This is helpful for constituency verification.

xi. getVoterDbId():

Returns the voter's database ID, linking the voter to database records for persistence and updates.

xii. setVoterDbId(int voterDbId):

Sets the voter's database ID after registration, ensuring proper linkage with database records.

xiii. getUserName() (inherited):

Returns the voter's name, inherited from User, helpful for system logs or displaying voter information.

5. Class Name: ElectionOfficer

The ElectionOfficer class, in the Users package, manages voter verification tasks, ensuring voters meet eligibility criteria before voting. It inherits from ElectionStaff, which extends User, and implements the ElectionRole interface for system interaction.

a. Attributes
   i. officerName: The name of the election officer, used to identify the officer performing verifications.
   ii. staffID (inherited): A unique ID given to each election officer, created automatically using a counter.
   iii. userName (inherited): The name of the election officer user. It helps identify who is using the system.

b. Methods
   i. ElectionOfficer(...):

This method initializes a new election officer with a name. It keeps officer identification consistent.

ii. verifyVoter(...):
Used to verify a voter's eligibility by checking age (≥18), presence in the voter list, Aadhaar length (12 digits), and address matching the constituency. It returns true if verified or false with an error message if not.

iii. iii. accessSystem(...):
This is the entry point for the election officer. It provides a framework for system interaction, specifically for voter verification.

iv. iv. getOfficerName():
The election officer can use this method to retrieve their name. This is helpful for system logs or showing officer information.

v. getStaffID() (inherited):
The election officer can use this method to retrieve their unique ID. This is helpful for system logs or verification.

vi. getUserName() (inherited):
The election officer can use this method to retrieve their name. This is helpful for system logs or showing officer information.

6. Class Name: Candidate

The Candidate class, in the Users package, represents individuals contesting in the election. It allows the system to store and manage candidate details, including their vote counts. This class does not inherit from any other class.

a. Attributes
   i. candidateName: The name of the candidate, used to identify them in the election.
   ii. candidateSymbol: The symbol representing the candidate, used for voter recognition.

iii. candidateParty: The party affiliation of the candidate, used to ensure election fairness.

iv. countVotes: The number of votes received by the candidate, tracked during the election.

b. Methods

i. Candidate(String name, String symbol, String candidateParty):
Initializes a new candidate with name, symbol, and party, setting the initial vote count to 0 and candidateId to -1.

ii. Candidate(String name, String symbol, String candidateParty, int candidateId):
An overloaded constructor initializing a candidate with name, symbol, party, and database ID, used when fetching from the database.

iii. getCandidateName():
Returns the candidate's name, used by the system to display candidate details in voting or results.

iv. getCandidateSymbol():
Returns the candidate's symbol, helping voters identify the candidate during voting.

v. getCandidateParty():
Returns the candidate's party, essential for managing party-based restrictions and displaying results.

vi. getVoteCount():
Returns the candidate's vote count, critical for calculating election results or displaying tallies.

vii. addVote():
Increments the candidate's vote count when a vote is cast, keeping the election tally accurate.

viii. setVoteCount(int voteCount):
Sets the candidate's vote count, used when syncing with database records during result retrieval.

ix. getCandidateId():
Returns the candidate's database ID, linking the candidate to database records for persistence.

x. setCandidateId(int candidateId):
Sets the candidate's database ID after registration, ensuring proper linkage with database records.

7. Class Name: EVM

The EVM class, in the System package, manages the core election process, including candidate management, voting, and vote storage. It serves as the central component for conducting the election and ensuring vote integrity. This class does not inherit from any other class.

a. Attributes
   i. candidateList: An ArrayList of Candidate objects, storing all candidates in the election.
   ii. master_admin_password: A static final string storing the password required for admin access.
   iii. constituency: A static string representing the name of the election constituency.
   iv. voteBuffer: A List of Ballot objects, storing vote records anonymously.
   v. voterCounter: An integer counter initialized to 1, used for anonymized voter IDs.

b. Methods
   i. displayCandidates(): This method displays the list of candidates with their numbers, names, parties, and symbols using an Iterator. The system uses this to show voting options.

ii. castVote(...): Used to process a vote by validating the button press and incrementing the selected candidate's vote count. It returns true if successful or false if invalid.
iii. verifyAdmin(...): This static method checks if the provided password matches the master_admin_password. It returns true if valid or false if not.
iv. getCandidateList(): The system can use this method to retrieve the list of candidates. This is helpful for managing or displaying candidate data.
v. shineRedLight(): Simulates a red light blink and beep after a valid vote. This provides voter feedback.
vi. logVote(...): Allows the system to log a vote anonymously with voter ID and candidate details. This ensures vote tracking.
vii. writeVotesToCSV(): Writes all votes from the voteBuffer to "votes.csv" using BufferedWriter, including headers and anonymized data. It confirms completion with a message.
viii. Ballot(...) (inner class constructor):
Initializes a new Ballot with an anonymized ID, candidate ID, and timestamp. This keeps vote records consistent.

Private Inner Class Ballot

The Ballot class represents a single vote record within the EVM class. It stores anonymized voting data to ensure privacy while tracking election results. This class is an inner class of EVM.

a. Attributes
   i. anonymizedID: A string representing an anonymized voter ID (e.g., "VoterX").
   ii. candidateID: The name of the candidate voted for, used to link the vote to a candidate.
   iii. timestamp: A LocalDateTime object recording the time the vote was cast.
b. Methods

i. Ballot(...)

This method initializes a new ballot with an anonymized ID, candidate ID, and the current timestamp. It ensures vote records are accurately timestamped.

8. Class Name: VVPAT

The VVPAT class, in the System package, manages the Voter Verified Paper Audit Trail process, ensuring vote transparency by generating, displaying, and printing confirmation slips. This class does not inherit from any other class.

a. Methods

i. generateSlip(...):

This method generates a confirmation slip with candidate name, party, symbol, and timestamp. The system uses this to provide voter verification.

ii. showSlip(...):

Used to display the confirmation slip with candidate name, party, and timestamp. This allows the voter to verify their vote.

iii. printSlip(...):

Prints the confirmation slip with candidate name, party, and timestamp, simulating the physical drop into a secure box. It ends with a newline and confirms the vote is secured.

9. Class Name: ElectionCommission

The ElectionCommission class, in the System package, manages the overall administration of the election process, including voter registration, verification, and data management. It serves as the central authority for maintaining voter records and ensuring electoral integrity. This class does not inherit from any other class.

a. Attributes

i. voterList: A HashMap storing voters with unique voter IDs as keys, used to manage voter registration and verification.

b. Methods

i. registerVoter(...):

This method registers a new voter with name, age, 12-digit Aadhaar number, and address. It checks for eligibility (age ≥ 18) and adds the voter to the voterList, rejecting invalid registrations with an error message.

ii. removeVoter(...):

Used to remove a voter from the voterList based on name, Aadhaar number, and a reason (e.g., duplicate, demise). It confirms success or shows an error if the voter isn't found.

iii. viewAllVoters():

Displays all registered voters' details (ID, name, age, Aadhaar, address) or indicates if no voters are registered. The system uses this to provide transparency.

iv. getVoterList():

The system can use this method to retrieve the voterList. This is helpful for verification or auditing purposes.

v. accessSystem(...):

This is the entry point for the Election Commission. It provides a framework for system interaction, specifically for voter management.

vi. manageData(...):

Allows the Election Commission to manage voter data, such as registering or removing voters as an example of data management functionality.

10. Class Name: ElectionSystem

The ElectionSystem class serves as the main controller for the election process, coordinating interactions between all system components and roles. It facilitates the initiation, management, and conclusion of the election. This class does not inherit from any other class.

a. Attributes

i. scan:

A static final Scanner object for handling user input throughout the system.

ii. ec:

A static ElectionCommission object, providing access to voter management functionality.

iii. admin:

An Admin object, responsible for managing candidates and results.

iv. evm:

An EVM object, handling voting and candidate management.

v. vvpat:

A VVPAT object, managing vote confirmation slips.

vi. currentVoter:

A Voter object storing the currently verified voter during GUI voting interactions.

vii. frame:

A JFrame object representing the main GUI window for the election system.

viii. cardLayout:

A CardLayout object managing multiple GUI panels for navigation.

ix. cards:

A JPanel containing all GUI panels, switched using CardLayout.

x. welcomePanel:

A JPanel for the initial constituency setup screen in the GUI.

xi. mainMenuPanel:

A JPanel for the main menu, offering role selection options.

xii. adminLoginPanel:

A JPanel for admin login with password input.

xiii. adminPanel:

A JPanel for admin operations like adding/removing candidates.

xiv. voterVerificationPanel:

A JPanel for voter ID verification before voting.

xv. votingPanel:

A JPanel for displaying candidates and casting votes.

xvi. ecPanel:

A JPanel for Election Commission tasks like voter registration.

xvii. resultsPanel:

A JPanel for displaying election results.

xviii. resultsTextArea:

A JTextArea within resultsPanel for showing election results.

b. Methods

i. startElection():

This method initiates the election process by prompting for the constituency name and offering a main menu with options for Admin, Voter, Election Commission, View Results, and End Election. It coordinates role-specific interactions and ends with vote storage.

ii. countVotes():

Used to display the vote count for all candidates (name, party, votes). The system uses this to provide a summary of voting data.

iii. findWinner():

Determines the election winner based on the highest vote count or declares a draw if multiple candidates tie, returning the winner or null if tied.

iv. startGUI():

Initializes and launches the Swing-based GUI, setting up panels for welcome, main menu, admin, voter, Election Commission, and results, handling all user interactions.

v. initializeData(DatabaseManager dbManager):

A private method pre-registering four candidates and ten voters, syncing with the database, and initializing the admin for the election setup.

vi. updateVotingPanel():

A private method dynamically updating the voting panel with candidate buttons, handling vote casting, and simulating EVM and VVPAT actions via GUI dialogs.

vii. updateResultsPanel():

A private method updating the results panel with the latest vote counts from the database, displaying them in the resultsTextArea.

viii. main(String[] args):

The entry point of the program, initializing the system with an admin, EVM, VVPAT, and starting the GUI-based election process.

11. Class Name: DatabaseManager

The DatabaseManager class, in the System package, manages all database operations for the election system, including operations for voters, candidates, votes, and constituencies. It does not inherit from any other class.

a. Attributes:

i. DB_URL:

A static final String storing the database URL ("jdbc:mysql://localhost:3306/ElectionSystem") for MySQL connectivity.

ii. USER:

A static final String storing the database username ("root") for authentication.

iii. PASS:

A static final String storing the database password ("bLACKPANTHER123#") for authentication.

b. Methods:

iv. customHash(String input):

A static method generating a simple hash of a string by summing character values and multiplying by 31, used for password hashing.

v. getConnection():
A static method establishing a connection to the MySQL database, returning a Connection object or throwing SQLException on failure.

vi. initializeAdmin(String userName, String password, String staffId):
Inserts an admin into the database with a hashed password, handling duplicates silently to avoid crashes.

vii. verifyAdmin(String userName, String password):
Verifies an admin's credentials by comparing the hashed input password with the stored hash, returning true if valid or false if not.

viii. addVoter(String voterId, String name, int age, String aadhaarNumber, String address, String constituencyName):
Adds a voter to the database, checking age eligibility, returning the generated database ID or -1 on failure.

ix. updateVoterStatus(int voterDbId, boolean hasVoted):
Updates a voter's has_voted status in the database, ensuring accurate voting records.

x. getVoter(String voterId):
Retrieves a voter from the database by voterId, returning a Voter object or null if not found.

xi. removeVoter(int voterDbId):
Deletes a voter from the database by their database ID, throwing SQLException if not found.

xii. getVoters(String constituencyName):
Retrieves a list of voters for a given constituency from the database, returning an empty list on error.

xiii. addCandidate(String name, String symbol, String party, String constituencyName):
Adds a candidate to the database, returning the generated database ID or -1 on failure (e.g., duplicate party).

xiv. removeCandidate(int candidateId):
Deletes a candidate from the database by their database ID, throwing CandidateNotFoundException if not found.

xv. getCandidates(String constituencyName):
Retrieves a list of candidates for a given constituency from the database, including vote counts, returning an empty list on error.

xvi. updateCandidateVoteCount(int candidateId):
Increments a candidate's vote count in the database, throwing SQLException if the candidate is not found.

xvii. addVote(int voterDbId, int candidateId, String constituencyName):
Records a vote in the database, checking for duplicate votes with InvalidVoteException, associating it with the constituency.

xviii. getVotes(String constituencyName):
Retrieves a list of vote records for a given constituency from the database, including candidate details and timestamps.

xix. addConstituency(String name):
Adds a constituency to the database, returning the generated database ID or -1 on failure.

xx. getConstituencyId(String constituencyName):
A private method retrieving a constituency's database ID, throwing SQLException if not found.

Private Inner Class: VoteRecord

The VoteRecord class, a static inner class of DatabaseManager, represents a single vote record retrieved from the database, used for exporting vote data.

a. Attributes:
    i. voteId:
    An int storing the database ID of the vote record.

ii. candidateId:
An int storing the database ID of the candidate voted for.

iii. candidateName:
A String storing the name of the candidate voted for.

iv. candidateParty:
A String storing the party of the candidate voted for.

v. timestamp:
A LocalDateTime storing the time the vote was cast.

b. Methods:

i. VoteRecord(int voteId, int candidateId, String candidateName, String candidateParty, LocalDateTime timestamp):
Initializes a new vote record with the given vote ID, candidate ID, name, party, and timestamp, used for vote data export.

Justification of Collections used in the Program:

1. ArrayList<Candidate> in EVM

In the EVM class (under the System package), I used an ArrayList<Candidate> for the candidateList because it fits the needs of managing candidates really well.

- Easy to Grow or Shrink: I needed something that could handle adding or removing candidates on the fly, like when the admin runs addCandidate or removeCandidate. ArrayList lets me do that without worrying about a fixed size.
- Keeps Things in Order: When showing candidates to voters with displayCandidates, I want them listed in the order they were added. ArrayList keeps that order naturally, which makes the display straightforward.

- Quick Changes: The admin might add or remove candidates a lot during setup, and ArrayList makes adding at the end super fast (basically instant, O(1)). Removing is a bit slower (O(n)), but it's still manageable for our small candidate lists.
- Fast Lookups by Index: When a voter picks a candidate in castVote, I grab the candidate by their index. ArrayList gives me that in O(1), so voting stays snappy.
- Saves Space: Compared to something like a LinkedList, ArrayList doesn't waste memory on extra pointers, which is great since we're only dealing with a handful of candidates per election.
- Works Well with the Database: Since I'm pulling candidates from the database with DatabaseManager.getCandidates, ArrayList makes it easy to load and sync that data in memory for the election.

2. List<Ballot> in EVM

For the voteBuffer in the EVM class (also in the System package), I went with a List<Ballot> (set up as an ArrayList) to store vote records during the election.

- Handles Any Number of Votes: I don't know how many people will vote, so List lets me keep adding votes to voteBuffer without worrying about running out of space.
- Keeps Votes in Order: When votes get logged with logVote, I want them in the order they happened. List keeps things chronological, which is handy for auditing later.
- Fast to Add Votes: Adding a vote to the end of the list in writeVotesToCSV is quick with List, it's basically instant (O(1)), which keeps things smooth during voting.
- Keeps Data Safe: Since ballots hold sensitive stuff like anonymizedID and timestamp, using a List inside EVM lets me control access and avoid accidental changes from other parts of the code.

- Easy to Loop Through: When I export votes to a CSV file in writeVotesToCSV, I just loop through the voteBuffer. List makes that simple with a basic for loop, and it's not too slow (O(n)).
- Helps with Database Backup: Even though votes now go straight to the database, I still use voteBuffer to keep an in-memory copy for the CSV export. List makes this backup step easy before I commit everything to the database.

3. HashMap<String, Voter> in ElectionCommission

In the ElectionCommission class (in the System package), I used a HashMap<String, Voter> for the voterList to keep track of voters, with their voterID as the key. (This is more of a backup now since I'm mostly using the database.)

- No Duplicate Voters: HashMap lets me use the voterID as a key, so I can make sure there's only one entry per voter when I call registerVoter. No duplicates, which is super important.
- Quick Checks: When verifying voters in ElectionOfficer.verifyVoter, I need to find them fast. HashMap looks up voters by their ID in O(1), so verification doesn't slow things down.
- Matches IDs to Voters: The voterList pairs each voterID with a Voter object, which makes it easy to grab voter details for things like removeVoter or viewAllVoters.
- Easy to Update: Adding or removing voters (like in detectDuplicates) happens a lot, and HashMap makes those updates quick, usually O(1) for both adding and deleting.
- Handles Growth Well: Even if there are tons of voters, HashMap can handle it by adjusting its size automatically, keeping things fast for big elections.

4. List<Candidate> in DatabaseManager

In the DatabaseManager class (also in the System package), I used a List<Candidate> (set up as an ArrayList) in the getCandidates method to pull candidates from the database and pass them around.

- Handles Any Number of Candidates: I don’t know how many candidates there’ll be in a constituency, so List lets me grab them all from the database without worrying about the count.
- Keeps the Database Order: The database gives me candidates in a certain order (like by ID), and List keeps that order when I display them in Admin.viewResults or set up voting buttons in ElectionSystem.updateVotingPanel.
- Easy to Loop Through: When showing results in Admin.viewResults, I loop through the candidates to print their vote counts. List makes that straightforward and not too slow (O(n)).
- Fast for Voting Buttons: In ElectionSystem.updateVotingPanel, I need to grab candidates by their position to make voting buttons. List lets me do that quickly with O(1) access.
- Doesn’t Waste Space: Since I’m just holding candidates temporarily after pulling them from the database, List (via ArrayList) keeps memory use low, which is good for small elections.
- Works with EVM: I use this List to load candidates into EVM.candidateList, so it keeps things consistent between the database and what the EVM uses during voting.

5. List<Voter> in DatabaseManager

In DatabaseManager (in the System package), I used a List<Voter> (as an ArrayList) in the getVoters method to fetch voters from the database for things like displaying them.

- Grabs All Voters: I don’t know how many voters are in a constituency, so List lets me pull them all from the database and handle any number.
- Shows Voters in Order: When I display voters in ElectionCommission.viewAllVoters, I want them in the order the database gives me. List keeps that order, making the output look clean.

- Simple to Loop Through: Looping through voters to show their details in viewAllVoters is easy with List. It's a basic O(n) loop, so it's not a performance issue.
- Temporary Holder: I only need the voter list for a short time after pulling it from the database, and List is a simple way to hold that data without keeping it around forever.
- Safe for Errors: If the database query fails, I return an empty List in getVoters. That way, the rest of the code doesn't crash—it just shows no voters.
- Helps with GUI: In the GUI, like in Election System ecPanel, I can use this List to show voter details. It's lightweight and works well for displaying small to medium voter lists.

6. List<VoteRecord> in DatabaseManager

Still in DatabaseManager (in the System package), I used a List<VoteRecord> (as an ArrayList) in the getVotes method to pull vote records from the database for exporting or auditing.

- Handles All Votes: Elections can have any number of votes, so List lets me grab them all from the database without worrying about the total.
- Keeps Votes in Time Order: The database gives me votes with timestamps, and List keeps them in the order I get them, which is great for showing the voting timeline in EVM.writeVotesToCSV.
- Easy for Exporting: When I export votes to a CSV in writeVotesToCSV, I loop through the List of vote records. It's a simple O(n) loop, so exporting stays efficient.
- Keeps Data Consistent: Each VoteRecord has stuff like candidate names and timestamps, and List makes sure they're all structured the same way for exporting or auditing.
- Handles Errors Gracefully: If the database query fails in getVotes, I return an empty List. That keeps EVM from crashing and lets it handle the error nicely.

- Matches CSV Format: The List<VoteRecord> structure fits perfectly with how I write votes to CSV in EVM.writeVotesToCSV, making it easy to turn vote data into CSV rows for audit trails.

Justifying the Variable Specifiers:

Private:

- Encapsulation for Data Security: Private variables restrict direct access, protecting sensitive data from unauthorized modification. For example, in EVM (in the System package), candidateList (an ArrayList<Candidate>) is private to prevent external classes from altering the candidate list directly, ensuring only addCandidate or removeCandidate methods (via Admin) can modify it.
- Controlled Access via Methods: Private variables force the use of getter/setter methods, allowing validation. In Voter (in the Users package), hasVoted is private to prevent tampering with a voter's voting status; only castVote can update it after checks, ensuring no duplicate votes.
- Internal State Management: Private variables manage internal state without exposing implementation details. In ElectionSystem (in the Main package), cardLayout (a CardLayout) is private since it's an internal GUI component for panel switching, not meant for external use.
- Preventing Unintended Modifications: In DatabaseManager (in the System package), DB_URL is private (and static final) to prevent accidental changes to the database URL, ensuring consistent database connectivity.

Public:

- Accessibility for External Interaction: Public variables allow necessary access for other classes to interact with the system. In EVM, constituency is a public static

String so that classes like ElectionSystem can access the election's constituency name for display or logging in the GUI.

- Shared Constants or Configurations: Public variables are used for constants that need to be shared. In DatabaseManager, USER and PASS are public static final to provide database credentials to methods like getConnection, ensuring consistent access across the system.
- Facilitating System-Wide Access: Public variables enable global access to critical components. In ElectionSystem, ec (an ElectionCommission) is public static so that other classes, like the GUI panels, can access voter management functionality without creating new instances.
- Simplified Access for Non-Sensitive Data: Public variables are used for data that doesn't need protection. In Candidate (in the Users package), candidateId could be public (though it's not currently) to allow direct access to the database ID for display purposes, as it's not sensitive.

What AI Contributed:

Splitting Code into Packages: At first, all my code was in one big file, which was super messy. AI helped me organize it by splitting it into packages like Main, Users, System, and exceptions. It showed me how to design this structure like putting user related classes (e.g, Voter, Admin) in Users and system logic (e.g, EVM, DatabaseManager) in System. AI also taught me how to use the right import statements, like import Users.Voter in ElectionSystem, so my classes could talk to each other without errors. This made my code much cleaner and easier to manage.

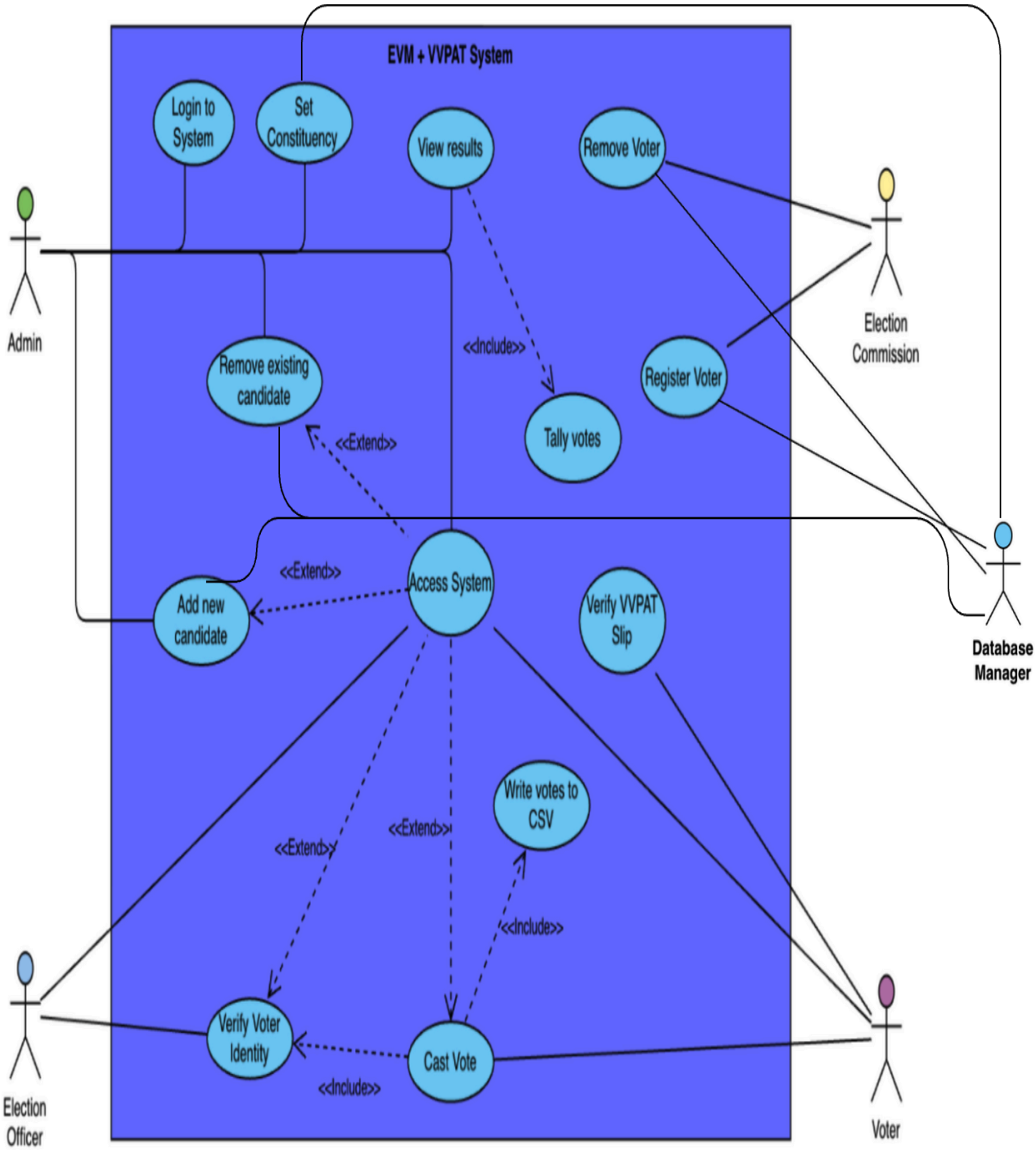
Adding and Improving the GUI: I started with a basic GUI in ElectionSystem just some boxes and labels using Swing. But AI helped me make it better. It showed me how to add more panels, like votingPanel and resultsPanel, and connect them to my methods,

like castVote in EVM for voting or viewResults in Admin for showing results. AI also helped me use CardLayout to switch between screens smoothly, making the GUI more user friendly.

Writing the Project Report: Writing the project report was tough because I wasn't sure how to explain my work clearly. AI helped me a lot here. It took my basic points, like why I used ArrayList in EVM and helped me explain them in more detail, like talking about fast lookups and memory efficiency. It also helped me structure sections like the class descriptions and justifications, making my report sound more professional and detailed while still being easy to understand.

Helping with PlantUML Code for Diagrams: I wanted to create class and sequence diagrams for my project to show how everything connects. I designed the diagrams on paper, like how User connects to Voter and Admin, or the sequence of castVote in EVM. AI helped me turn my designs into proper PlantUML code. For example, it gave me the code for the class diagram to show inheritance (e.g., User <|-- Voter) and for the sequence diagram to show steps like Voter -> EVM: castVote(). This made it easy to generate the diagrams and include them in my report.

Use Case Diagram:


EVM + VVPAT System
Login to System
Set Constituency
View results
Remove Voter
Admin
Election Commission
Remove existing candidate
<<Include>>
Register Voter
Tally votes
<<Extend>>
<<Extend>>
Access System
Add new candidate
Verify VVPAT Slip
Database Manager
Write votes to CSV
<<Extend>>
<<Extend>>
<<Include>>
Verify Voter Identity
Cast Vote
<<Include>>
Election Officer
Voter

Case 1: Secure Admin Verification Before Election Configuration

1. Actors:
   • Admin
   • EVM System
2. Preconditions:
   • Admin must be registered with the correct password.
   • System must not be in voting mode.
3. Main Flow:
   a. Admin attempts to access the system.
   b. System prompts for the master admin password.
   c. Admin inputs the password.
   d. System verifies the password.
   e. If valid, access is granted. Otherwise, access is denied and an alert message is displayed.
4. Postconditions:
   • Admin gains access to sensitive election settings.
   • Unauthorized users are blocked from proceeding.

Case 2: Adding a Candidate with Party Validation

1. Actors:
   • Admin
   • EVM System
   • Database Manager
2. Preconditions:
   • Admin must be authenticated.
   • The party being added must not already have a candidate.
3. Main Flow:
   a. Admin selects the option to add a candidate.
   b. Enter the candidate's name, symbol, and party.

c. System checks if another candidate from the same party already exists.
d. If not, the candidate is added. Otherwise, the system throws an exception.

4. Postconditions:
   • New candidate was added successfully.
   • Party duplication is prevented to maintain fairness.
   • Database Manager adds new entry into database

Use Case 3: Verifying a Voter Before Allowing to Vote

1. Actors:
   • Election Officer
   • Voter
   • Election Commission
2. Preconditions:
   • Voters must be registered.
   • Aadhaar and address must be stored in the system.
3. Main Flow:
   a. Voter presents ID and Aadhaar to the officer.
   b. Officer inputs details into the system.
   c. System checks voter age (18+), Aadhaar format, and constituency match.
   d. If all checks pass, the voter is verified for voting.

Postconditions:
• Voters are granted access to vote.
• Invalid or mismatched data results in denial of access.

Case 4: Casting a Valid Vote and Generating a VVPAT Slip

1. Actors:
   • Voter

- EVM
- VVPAT

2. Preconditions:
   - Voters must be verified and must not have voted before.
3. Main Flow:
   a. Voter selects a candidate from the displayed list.
   b. System checks if the button press is valid and unique.
   c. If valid, vote is recorded, red light and beep are triggered.
   d. VVPAT generates, displays, and prints the slip.
4. Postconditions:
   - Vote is registered securely and anonymously.
   - Paper trail is created for verification.

Case 5: Preventing Duplicate Voting Attempts

1. Actors:
   - Voter
   - EVM
2. Preconditions:
   - Voter has already cast a vote.
3. Main Flow:
   a. Voter re-attempts to vote using their ID.
   b. System checks vote status.
   c. Detects that the voter has already voted.
   d. Blocks access and shows a warning.
4. Postconditions:
   - Duplicate vote attempt is stopped.
   - System integrity is preserved.

Case 6: Handling Invalid Button Press During Voting

1. Actors:
   - Voter
   - EVM
2. Preconditions:
   - Voter is verified and ready to vote.
3. Main Flow:
   a. Voter presses a button outside the valid range or multiple buttons.
   b. System detects invalid or double input.
   c. Voting process is halted and cancelled.
   d. Voter is shown an error message.
4. Postconditions:
   - Vote is not recorded.
   - Voters can be redirected to start again (if allowed).

Case 7: Writing All Votes to CSV for Audit

1. Actors:
   - Admin
   - EVM System
2. Preconditions:
   - At least one vote must be cast.
   - Election must be officially ended.
3. Main Flow:
   a. Admin ends the election.
   b. System goes into results and export mode.
   c. All votes stored in memory are written to a CSV file with an anonymized ID and timestamp.

4. Postconditions:
    • CSV log is saved securely for auditing.
    • Tamper-proof vote trail is preserved.

Case 8: Removing Voter Based on Verified Disqualification

1. Actors:
    • Election Commission
    • Voter
2. Preconditions:
    • Voters must exist in the database.
3. Main Flow:
    a. Commission selects reasons for removal (e.g., moved, deceased, disqualified).
    b. Inputs voter's name and Aadhaar.
    c. System validates that the voter exists.
    d. Voter is removed, and a message is displayed.
4. Postconditions:
    • Voter is cleanly removed from the list.
    • System prevents misuse by disqualified users.

Case 9: Detecting and Deleting Duplicate Voter Records

1. Actors:
    • Election Commission
2. Preconditions:
    • Multiple records with the same name and Aadhaar exist.
3. Main Flow:
    a. Commission initiates “Remove Duplicate Entry”.
    b. System scans the voter list for matches.
    c. Deletes duplicate entries while retaining one.

4. Postconditions:
    • Voter list is cleaned.
    • Prevents multiple votes from the same person.

Case 10: Declaring Election Winner or Tie

1. Actors:
    • EVM
    • Admin
2. Preconditions:
    • Election must be completed.
    • At least one vote must exist.
3. Main Flow:
    a. Admin requests to view the winner.
    b. System counts all votes.
    c. Compare vote totals.
    d. If one has the highest, declare the winner. If tied, display draw.
4. Postconditions:
    • Winner is declared.
    • Public results are shown clearly and fairly.

Case 11: Initial System Setup with Constituency Configuration

1. Actors:
    • Admin
    • ElectionSystem
    • Database Manager
2. Preconditions:
    • System must be newly started.
3. Main Flow:
    a. Admin launches the ElectionSystem.

b. System prompts for the constituency name.
c. Admin enters the constituency name.
d. System sets the constituency and proceeds to initial setup.

4. Postconditions:
   - Constituency is configured for the election.
   - System is ready for candidate and voter registration.
   - Database Manager adds new entry into database

Case 12: Preloading Candidates During System Initialization

1. Actors:
   - Admin
   - ElectionSystem
   - Database Manager
2. Preconditions:
   - System must be in initial setup mode.
3. Main Flow:
   a. Admin triggers the initial setup process.
   b. System automatically pre-registers four predefined candidates with names, symbols, and parties.
   c. Admin confirms the candidate list is loaded.
4. Postconditions:
   - Four candidates are successfully added to the EVM.
   - Election is ready for voter registration.
   - Database Manager adds new entry into database

Case 13: Preloading Voters During System Initialization

1. Actors:
   - ElectionCommission
   - ElectionSystem

• Database Manager

2. Preconditions:
   • System must be in initial setup mode.
3. Main Flow:
   a. ElectionCommission initiates the initial setup process.
   b. System automatically pre-registers ten predefined voters with names, ages, Aadhaar numbers, and addresses.
   c. The Commission verifies the voter list.
4. Postconditions:
   • Ten voters are successfully registered in the voterList.
   • Election is ready to begin.
   • Database Manager adds new entry into database

Case 14: Viewing All Candidates by Admin

1. Actors:
   • Admin
   • EVM
   • Database Manager adds new entry into database
2. Preconditions:
   • Admin must be authenticated.
3. Main Flow:
   a. Admin selects an option to view all candidates.
   b. EVM retrieves and displays the candidate list with names, parties, and symbols. It uses the database manager to retrieve.
   c. Admin reviews the displayed information.
4. Postconditions:
   • Admin has access to the current candidate roster.
   • No changes are made to the election data.

Case 15: Election Officer Rejecting Voter Due to Underage Status

1. Actors:
   • Election Officer
   • Voter
   • Election Commission
2. Preconditions:
   • Voter is registered but under 18 years old.
3. Main Flow:
   a. Voter presents ID and Aadhaar to the officer.
   b. Officer inputs details into the system.
   c. System checks voter age and detects it is under 18.
   d. Officer receives an error message and denies voting access.
4. Postconditions:
   • Voters are blocked from voting.
   • System integrity is maintained by rejecting underage voters.

Case 16: Admin Exiting Configuration Mode

1. Actors:
   • Admin
   • EVM System
2. Preconditions:
   • Admin must be authenticated and in configuration mode.
3. Main Flow:
   a. Admin selects the exit option from the admin menu.
   b. System confirms the exit request.
   c. Admin confirms, and the system returns to the main menu.
4. Postconditions:
   • Admin is logged out of configuration mode.
   • System awaits next user interaction.

Case 17: Election Commission Viewing Voter Statistics

1. Actors:
   • ElectionCommission
   • ElectionSystem
2. Preconditions:
   • At least one voter must be registered.
3. Main Flow:
   a. ElectionCommission selects the option to view all voters.
   b. System retrieves and displays voter details (ID, name, age, Aadhaar, address).
   c. The Commission analyzes the voter statistics.
4. Postconditions:
   • Commission has access to voter data for review.
   • No modifications are made to the voter list.

Case 18: Handling System Crash During Vote Logging

1. Actors:
   • EVM
   • ElectionSystem
2. Preconditions:
   • A vote is being logged, and an IOException occurs.
3. Main Flow:
   a. EVM attempts to log a vote to the voteBuffer.
   b. An IOException is triggered during the process.
   c. System catches the exception and displays an error message.
   d. Voting continues without crashing.
4. Postconditions:
   • Vote logging fails for the affected vote.

• System remains operational for other transactions.

Case 19: Voter Attempting to Vote with Non-Matching Constituency

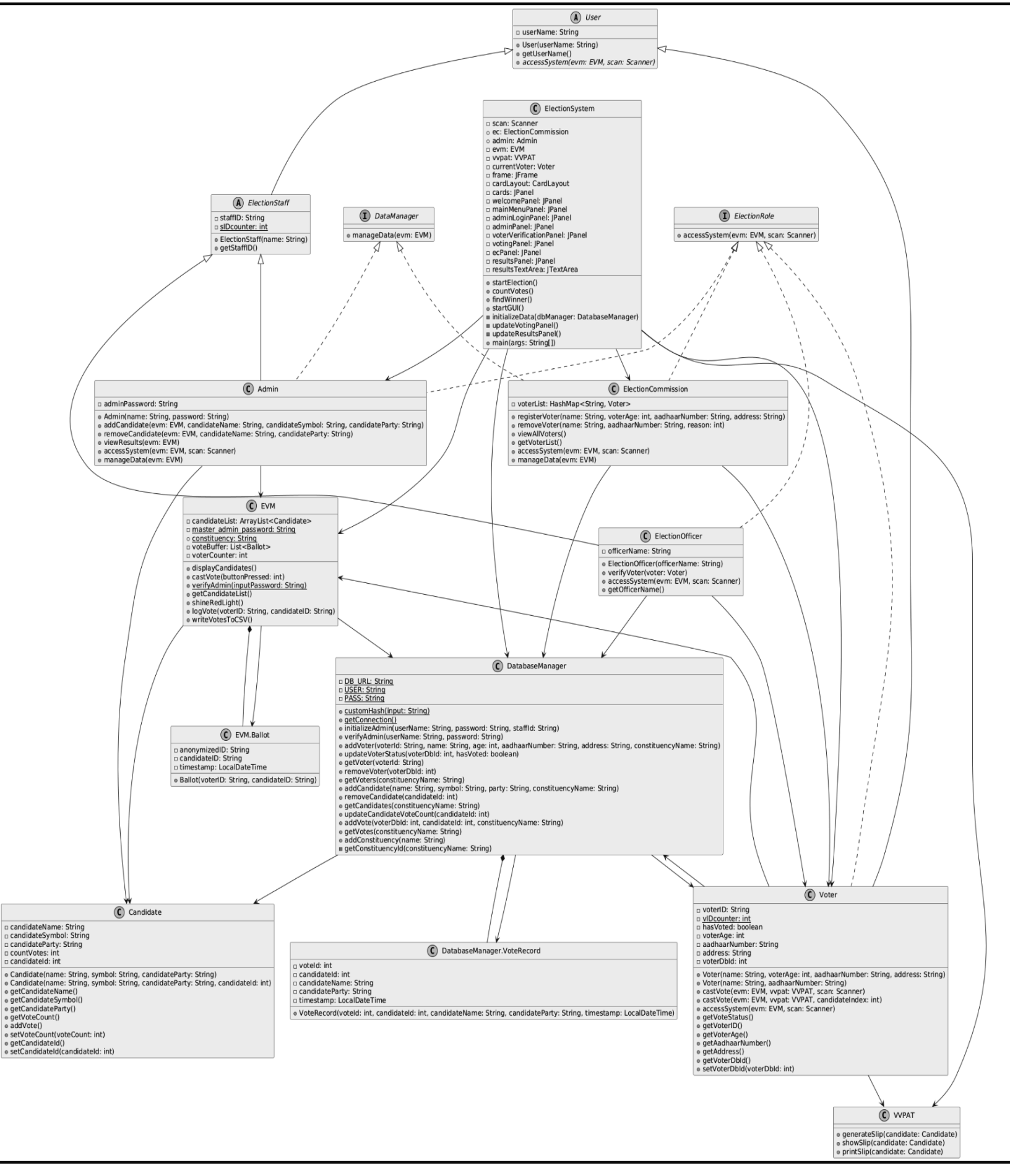
1. Actors:
   • Election Officer
   • Voter
   • Election Commission
2. Preconditions:
   • Voter is registered but from a different constituency.
3. Main Flow:
   a. Voter presents ID and Aadhaar to the officer.
   b. Officer inputs details into the system.
   c. System checks the address and detects a constituency mismatch.
   d. Officer receives an error message and denies voting access.
4. Postconditions:
   • Voters are blocked from voting.
   • System ensures constituency-specific voting.

Case 20: Admin Resetting Candidate List

1. Actors:
   • Admin
   • EVM
2. Preconditions:
   • Admin must be authenticated.
   • At least one candidate must exist.
3. Main Flow:
   a. Admin selects an option to reset the candidate list.
   b. System prompts for confirmation to clear all candidates.
   c. Admin confirms, and the candidateList is emptied.

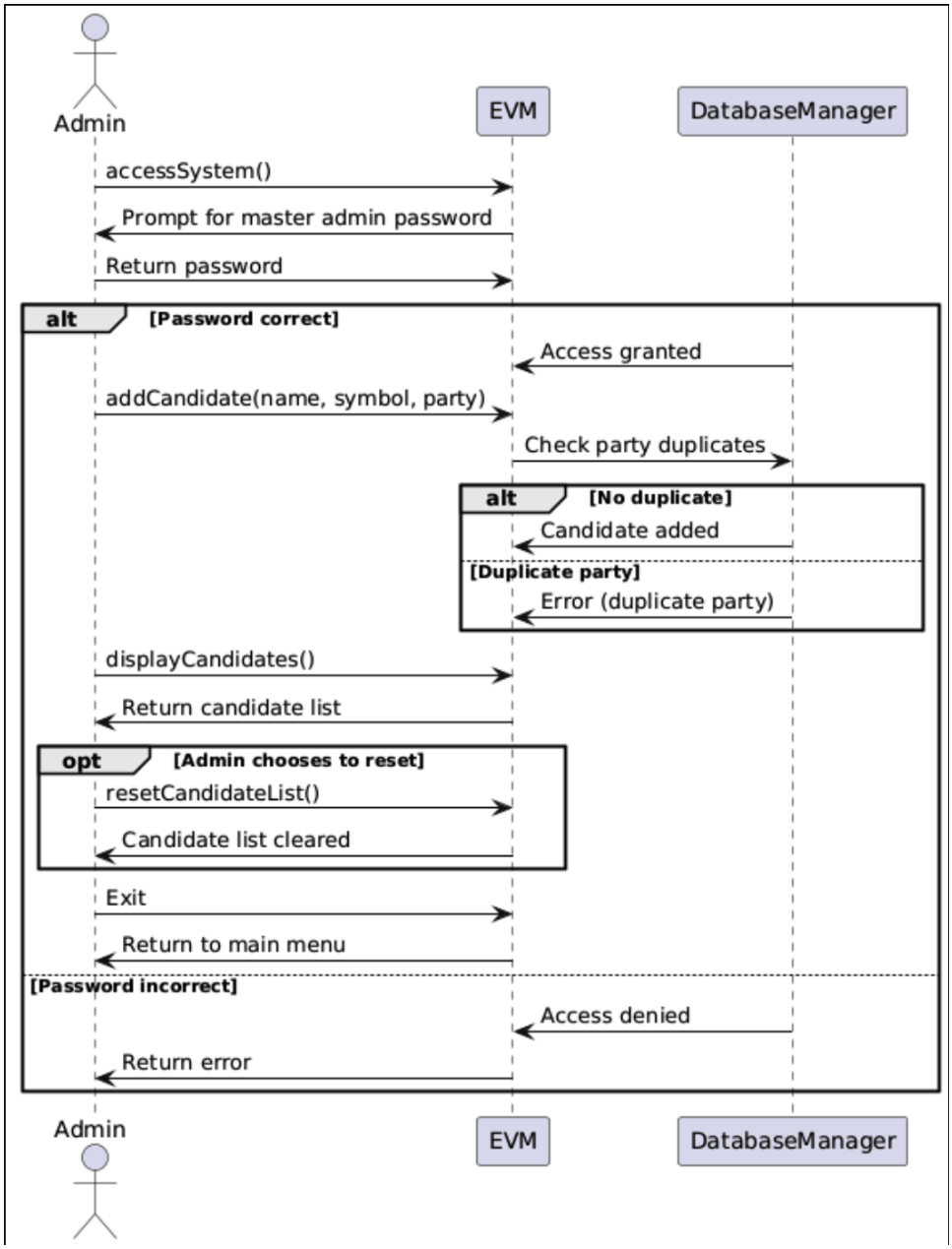
4. Postconditions:
    - All candidates are removed from the EVM.
    - System is reset for a new candidate setup.

Class Diagram:

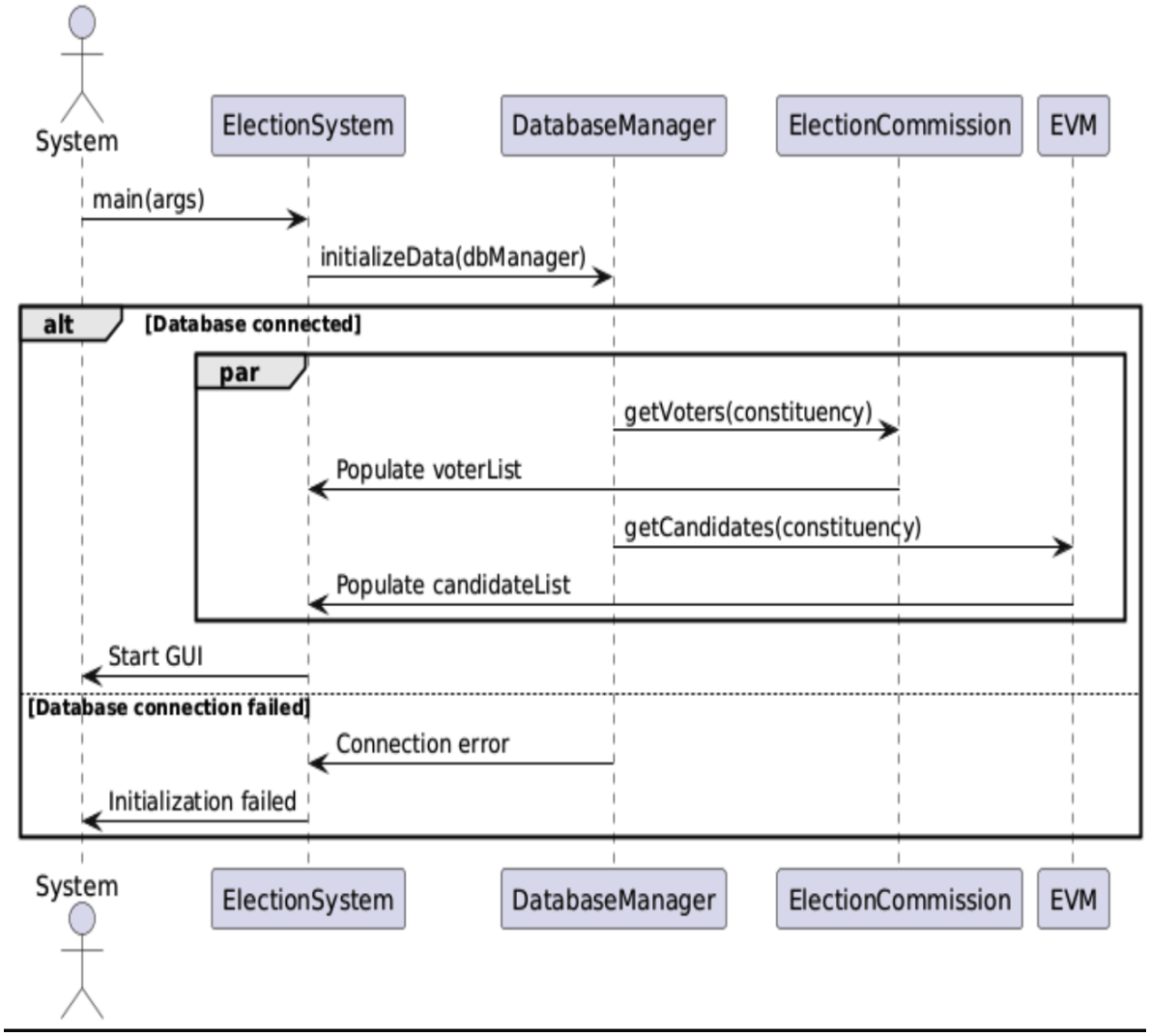

User
- userName: String
+ User(userName: String)
+ getUserName()
+ accessSystem(evm: EVM, scan: Scanner)
ElectionSystem
- scan: Scanner
- ec: ElectionCommission
- admin: Admin
- evm: EVM
- vvpat: VVPAT
- currentVoter: Voter
- frame: JFrame
- cardLayout: CardLayout
- cards: JPanel
- welcomePanel: JPanel
- mainMenuPanel: JPanel
- adminLoginPanel: JPanel
- adminPanel: JPanel
- voterVerificationPanel: JPanel
- votingPanel: JPanel
- ecPanel: JPanel
- resultsPanel: JPanel
- resultsTextArea: JTextArea
+ startElection()
+ countVotes()
+ findWinner()
+ startGUI()
- initializeData(dbManager: DatabaseManager)
- updateVotingPanel()
- updateResultsPanel()
+ main(args: String[])
ElectionStaff
- staffID: String
- sIDcounter: int
+ ElectionStaff(name: String)
+ getStaffID()
DataManager
+ manageData(evm: EVM)
ElectionRole
+ accessSystem(evm: EVM, scan: Scanner)
Admin
- adminPassword: String
+ Admin(name: String, password: String)
+ addCandidate(evm: EVM, candidateName: String, candidateSymbol: String, candidateParty: String)
+ removeCandidate(evm: EVM, candidateName: String, candidateParty: String)
+ viewResults(evm: EVM)
+ accessSystem(evm: EVM, scan: Scanner)
+ manageData(evm: EVM)
ElectionCommission
- voterList: HashMap<String, Voter>
+ registerVoter(name: String, voterAge: int, aadhaarNumber: String, address: String)
+ removeVoter(name: String, aadhaarNumber: String, reason: int)
+ viewAllVoters()
+ getVoterList()
+ accessSystem(evm: EVM, scan: Scanner)
+ manageData(evm: EVM)
EVM
- candidateList: ArrayList<Candidate>
- master_admin_password: String
- constituency: String
- voteBuffer: List<Ballot>
- voterCounter: int
+ displayCandidates()
+ castVote(buttonPressed: int)
+ verifyAdmin(inputPassword: String)
+ getCandidateList()
+ shineRedLight()
+ logVote(voterID: String, candidateID: String)
+ writeVotesToCSV()
ElectionOfficer
- officerName: String
+ ElectionOfficer(officerName: String)
+ verifyVoter(voter: Voter)
+ accessSystem(evm: EVM, scan: Scanner)
+ getOfficerName()
DatabaseManager
- DB_URL: String
- USER: String
- PASS: String
+ customHash(input: String)
+ getConnection()
+ initializeAdmin(userName: String, password: String, staffId: String)
+ verifyAdmin(userName: String, password: String)
+ addVoter(voterId: String, name: String, age: int, aadhaarNumber: String, address: String, constituencyName: String)
+ updateVoterStatus(voterDbId: int, hasVoted: boolean)
+ getVoter(voterId: String)
+ removeVoter(voterDbId: int)
+ getVoters(constituencyName: String)
+ addCandidate(name: String, symbol: String, party: String, constituencyName: String)
+ removeCandidate(candidateId: int)
+ getCandidates(constituencyName: String)
+ updateCandidateVoteCount(candidateId: int)
+ addVote(voterDbId: int, candidateId: int, constituencyName: String)
+ getVotes(constituencyName: String)
+ addConstituency(name: String)
- getConstituencyId(constituencyName: String)
EVM.Ballot
- anonymizedID: String
- candidateID: String
- timestamp: LocalDateTime
+ Ballot(voterID: String, candidateID: String)
Candidate
- candidateName: String
- candidateSymbol: String
- candidateParty: String
- countVotes: int
- candidateId: int
+ Candidate(name: String, symbol: String, candidateParty: String)
+ Candidate(name: String, symbol: String, candidateParty: String, candidateId: int)
+ getCandidateName()
+ getCandidateSymbol()
+ getCandidateParty()
+ getVoteCount()
+ addVote()
+ setVoteCount(voteCount: int)
+ getCandidateId()
+ setCandidateId(candidateId: int)
DatabaseManager.VoteRecord
- voteId: int
- candidateId: int
- candidateName: String
- candidateParty: String
- timestamp: LocalDateTime
+ VoteRecord(voteId: int, candidateId: int, candidateName: String, candidateParty: String, timestamp: LocalDateTime)
Voter
- voterID: String
- vIDcounter: int
- hasVoted: boolean
- voterAge: int
- aadhaarNumber: String
- address: String
- voterDbId: int
+ Voter(name: String, voterAge: int, aadhaarNumber: String, address: String)
+ Voter(name: String, aadhaarNumber: String)
+ castVote(evm: EVM, vvpat: VVPAT, scan: Scanner)
+ castVote(evm: EVM, vvpat: VVPAT, candidateIndex: int)
+ accessSystem(evm: EVM, scan: Scanner)
+ getVoteStatus()
+ getVoterID()
+ getVoterAge()
+ getAadhaarNumber()
+ getAddress()
+ getVoterDbId()
+ setVoterDbId(voterDbId: int)
VVPAT
+ generateSlip(candidate: Candidate)
+ showSlip(candidate: Candidate)
+ printSlip(candidate: Candidate)

Sequence Diagram:

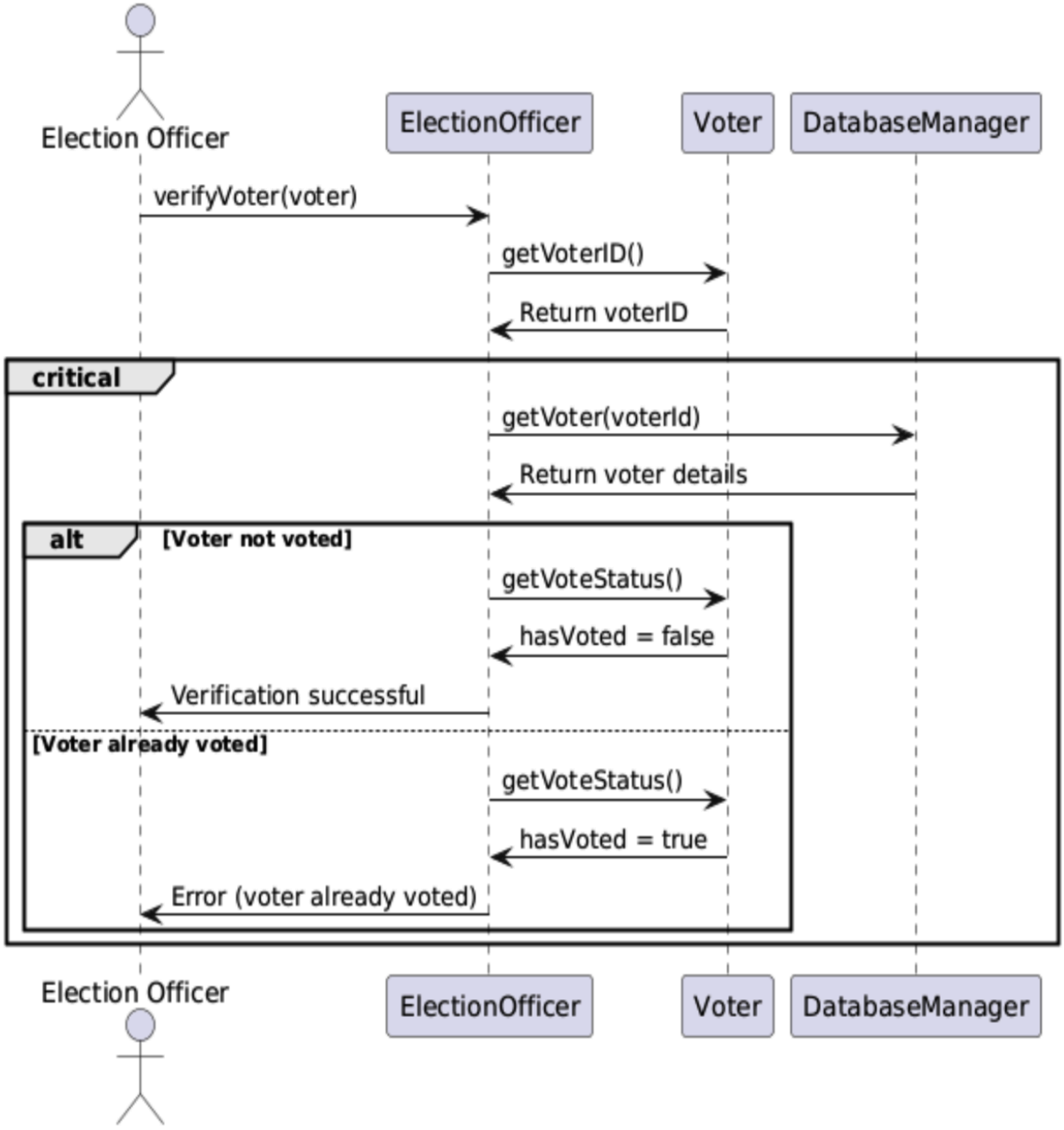
1. Admin Configuration and Management



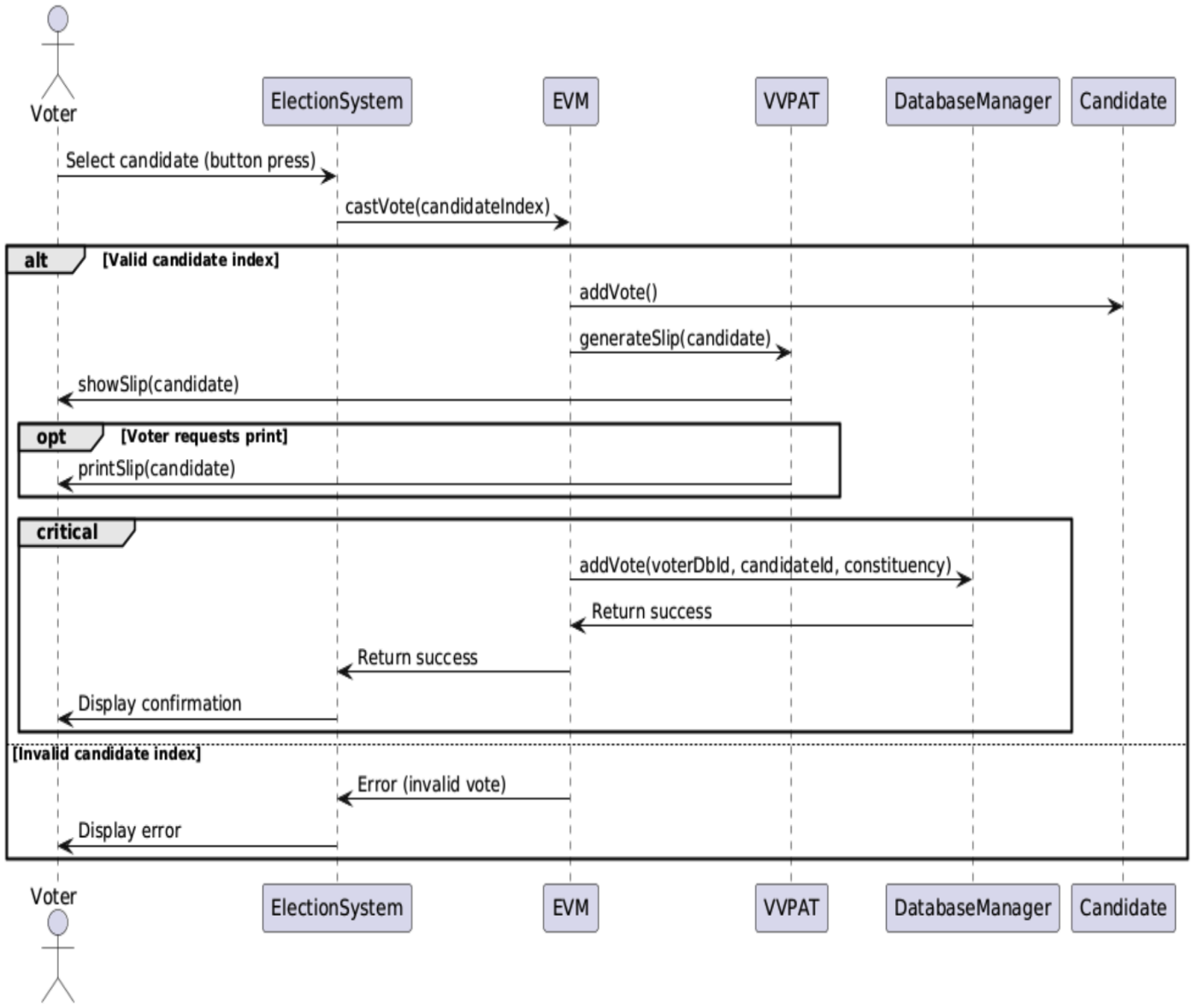
2. System Initialization and Setup



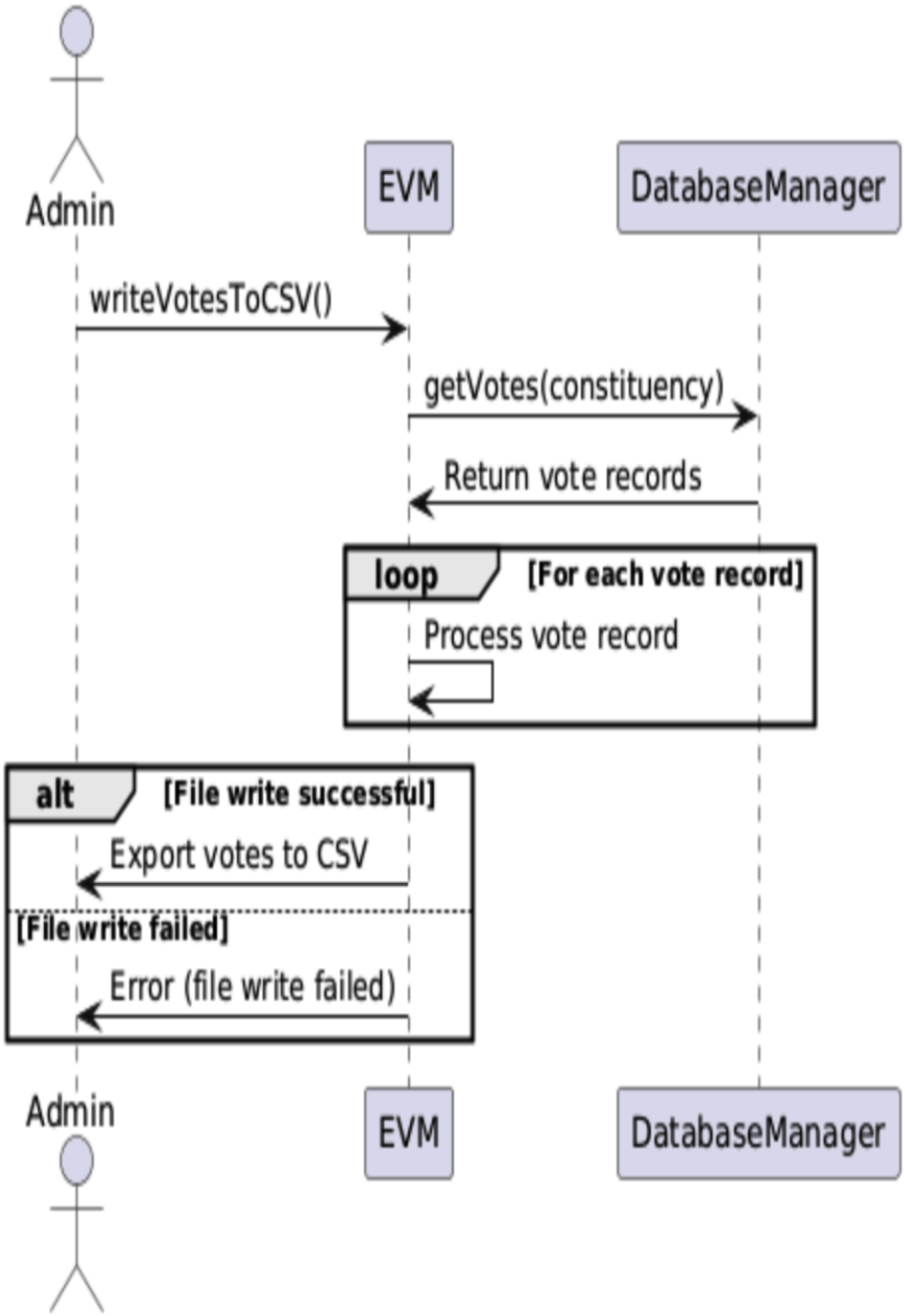
3. Voter Verification and Eligibility Checks



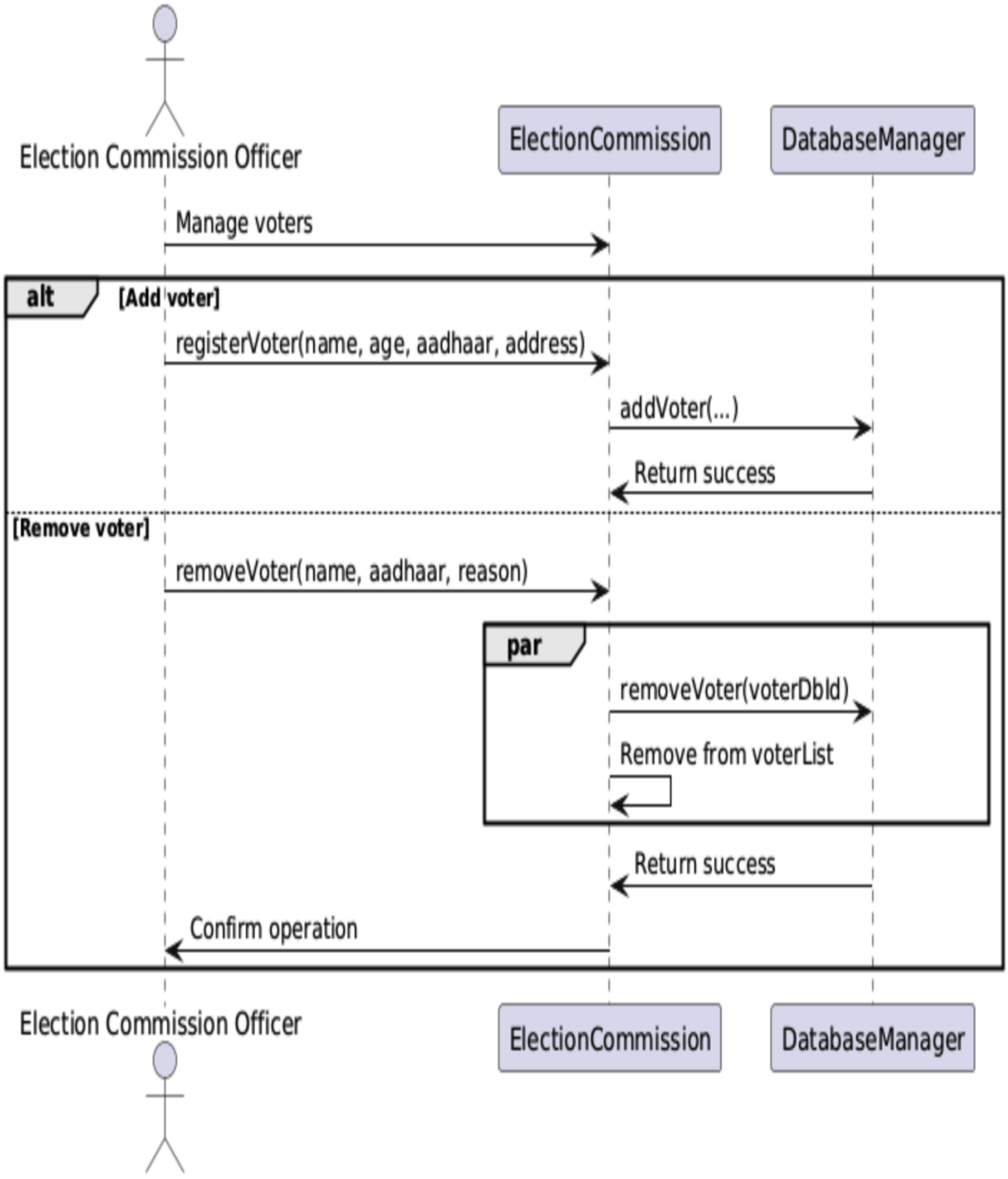
4. Voting Process and VVPAT Generation



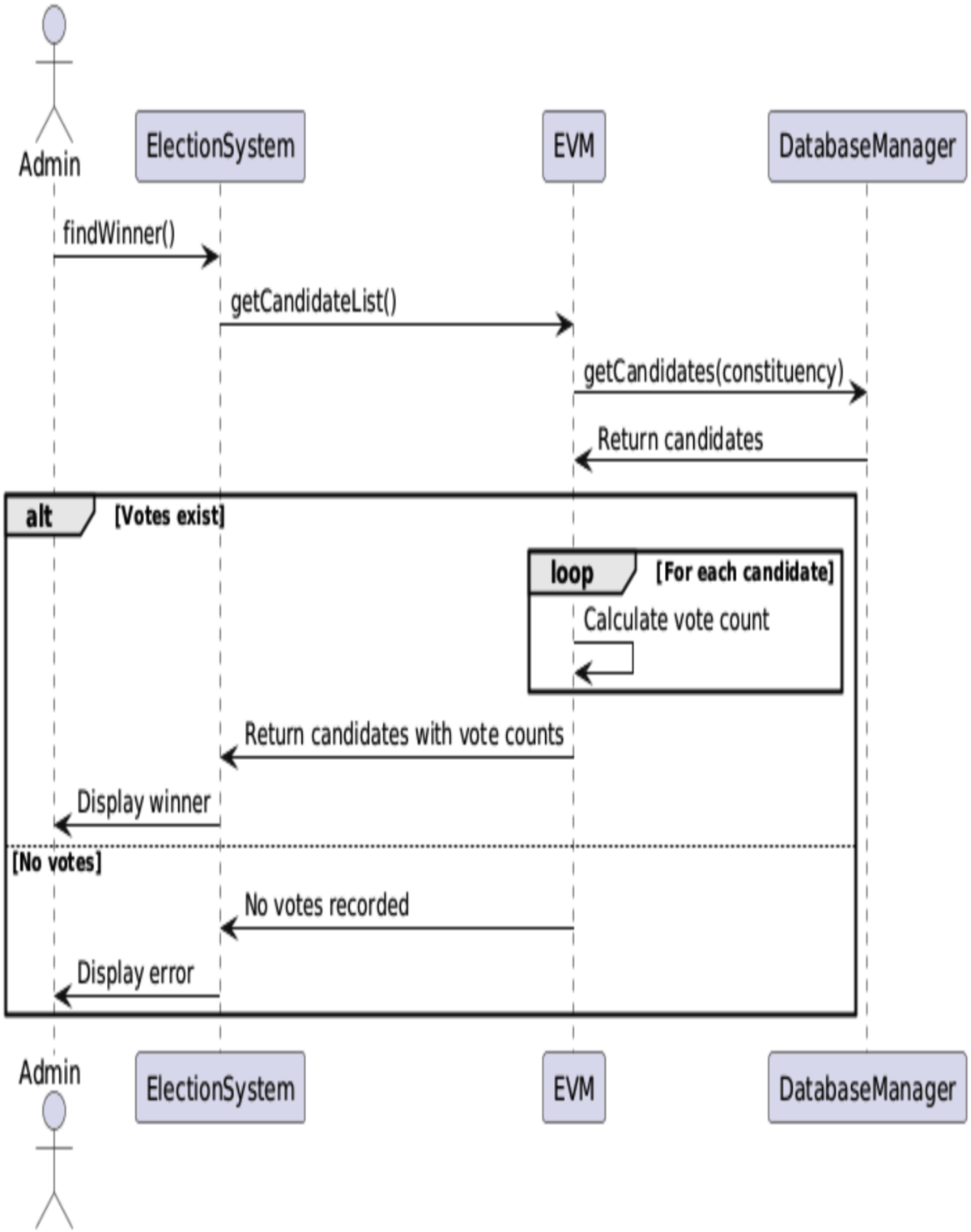
5. Vote Export and Audit



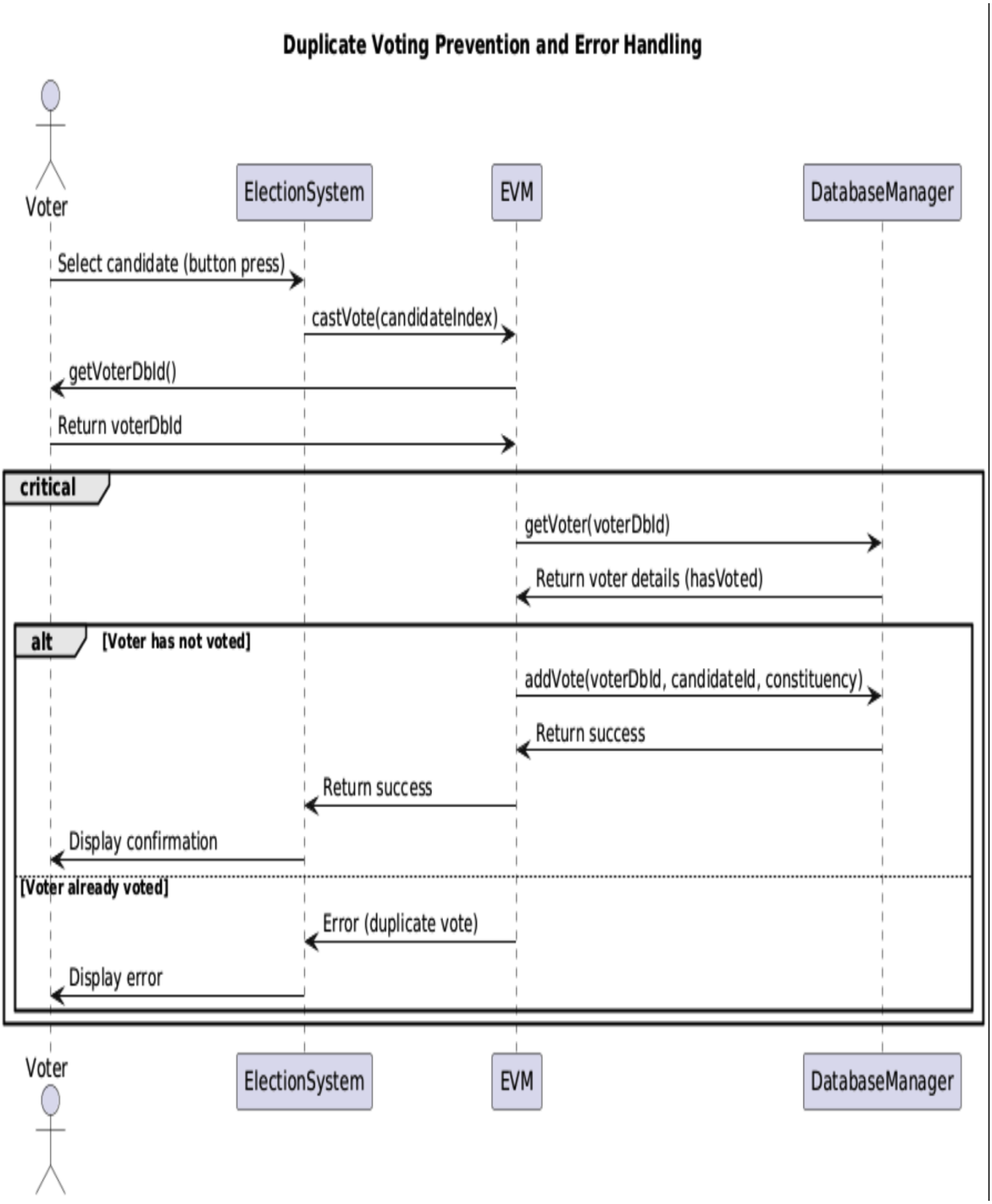
6. Voter Management by Election Commission



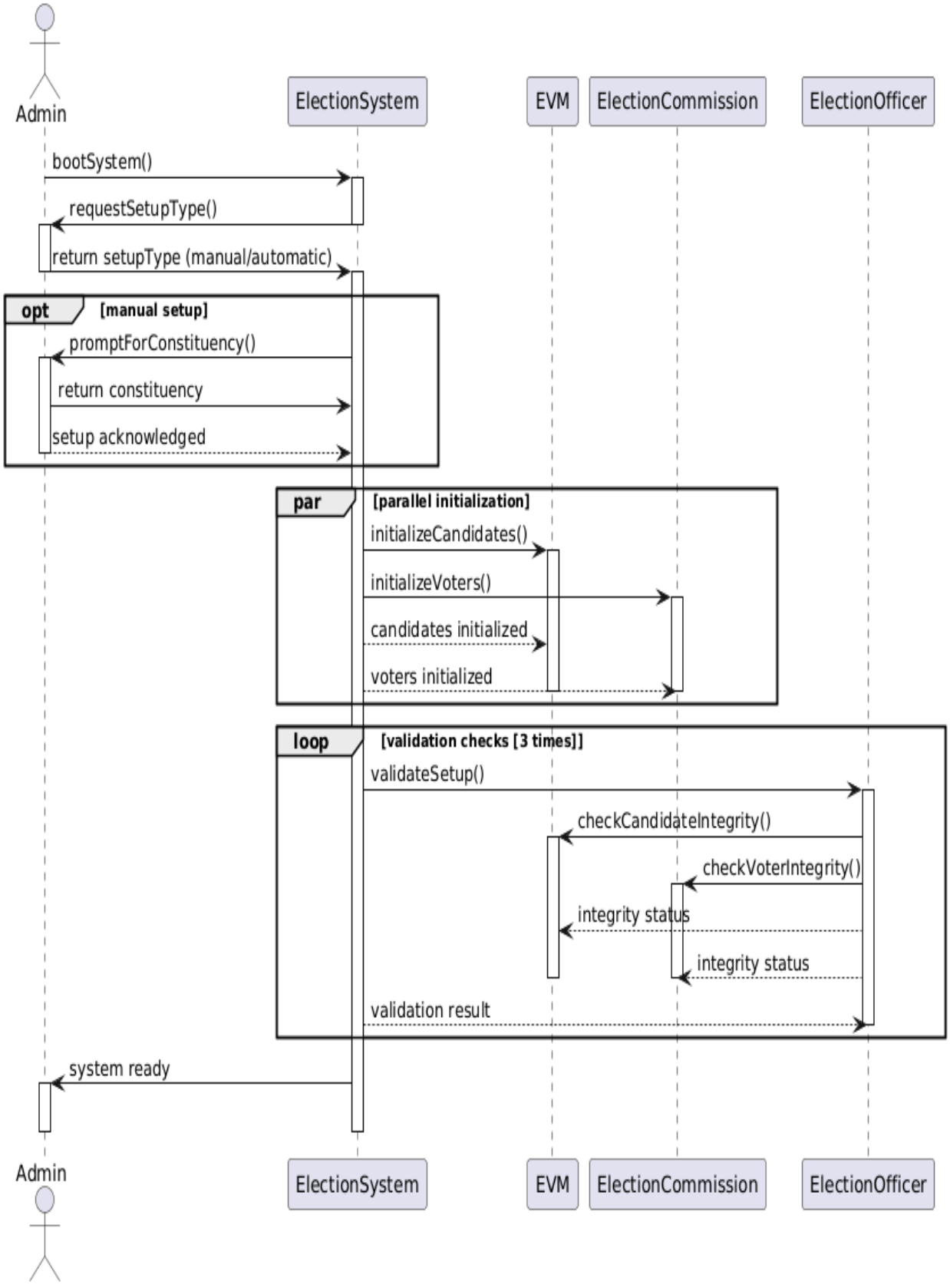
7. Election Result Declaration



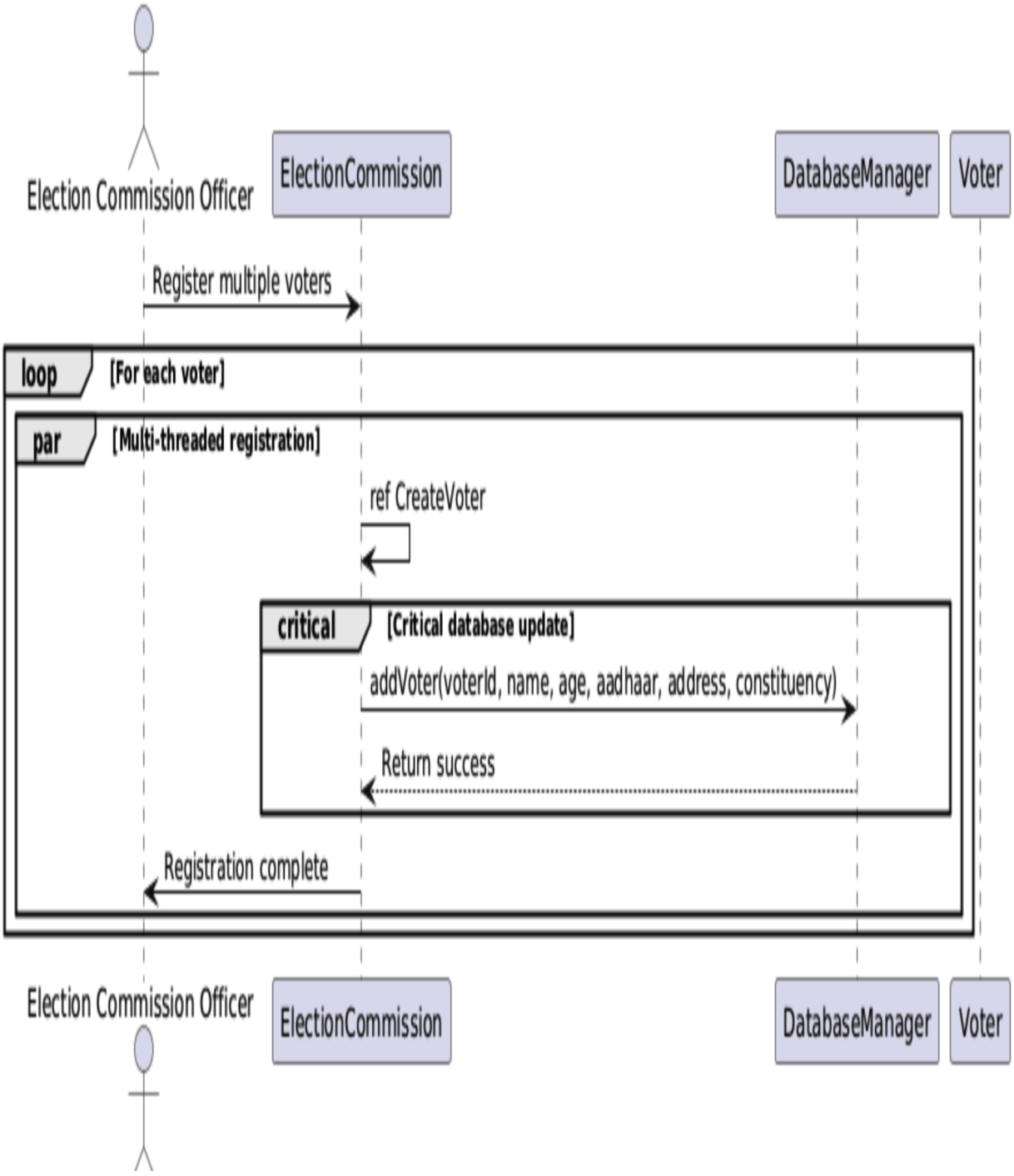
8. Duplicate Voting Prevention and Error Handling



9. Election System Bootstrapping and Multi-Phase Setup


Admin
ElectionSystem
EVM
ElectionCommission
ElectionOfficer
bootSystem()
requestSetupType()
return setupType (manual/automatic)
opt
[manual setup]
promptForConstituency()
return constituency
setup acknowledged
par
[parallel initialization]
initializeCandidates()
initializeVoters()
candidates initialized
voters initialized
loop
[validation checks [3 times]]
validateSetup()
checkCandidateIntegrity()
checkVoterIntegrity()
integrity status
integrity status
validation result
system ready
Admin
ElectionSystem
EVM
ElectionCommission
ElectionOfficer

10. Voter Registration and Multi Threaded Processing



Object Oriented Programming Checklist:

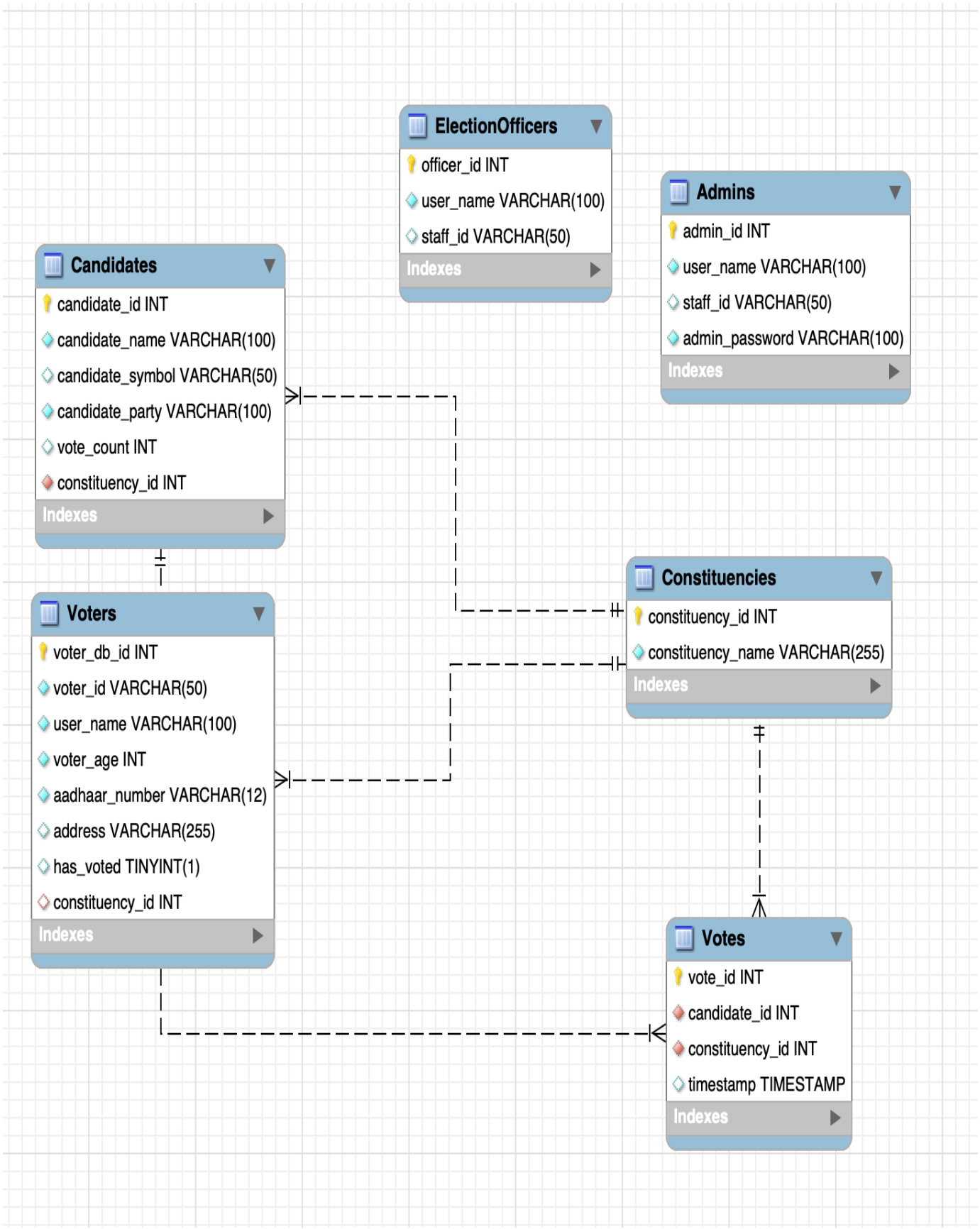
Program	
Feature	**Checklist**
Class (Private, Public, Protected)	YES
Sub Class	YES
Inner Class	YES
Method with Object reference	YES
Methods without Object Reference	YES
Constructors	YES
Overloaded Constructors	YES
Collections: 1 , 2 , 3	YES
Overloaded Methods	YES
Abstraction through Abstract Class	YES
Abstraction through interface	YES
Inheritance: Single	YES
Inheritance Multilevel	YES
Inheritance Hierarchical	YES
Exceptions	YES

User Defined Exceptions	YES
Multiple Inheritance	YES
Design	
Use Case Diagram: Use Cases, Relation, Actors Pre Condition and Post Conditions · Consistency with Method in Java Code	YES
Class Diagram: Class, Attributes, Cardinality, Correct notations · Consistency with Method in Java Code	YES
Sequence Diagram: · Lifeline · Message Flow · Blocks (loop, condition……)	YES

· Consistency with Method in Java Code	

EER Diagram

This is the EER diagram MySQL made for my schema and code:


ElectionOfficers
officer_id INT
user_name VARCHAR(100)
staff_id VARCHAR(50)
Indexes
Admins
admin_id INT
user_name VARCHAR(100)
staff_id VARCHAR(50)
admin_password VARCHAR(100)
Indexes
Candidates
candidate_id INT
candidate_name VARCHAR(100)
candidate_symbol VARCHAR(50)
candidate_party VARCHAR(100)
vote_count INT
constituency_id INT
Indexes
Constituencies
constituency_id INT
constituency_name VARCHAR(255)
Indexes
Voters
voter_db_id INT
voter_id VARCHAR(50)
user_name VARCHAR(100)
voter_age INT
aadhaar_number VARCHAR(12)
address VARCHAR(255)
has_voted TINYINT(1)
constituency_id INT
Indexes
Votes
vote_id INT
candidate_id INT
constituency_id INT
timestamp TIMESTAMP
Indexes

MySQL CODE:

```
-- Create the database if it doesn't exist
CREATE DATABASE IF NOT EXISTS ElectionSystem;

-- Use the created database
USE ElectionSystem;



-- Create Constituencies table
CREATE TABLE Constituencies (
    constituency_id INT AUTO_INCREMENT PRIMARY KEY,
    constituency_name VARCHAR(255) NOT NULL UNIQUE
);

-- Create Admins table (assuming one admin for now, but table structure allows more)
CREATE TABLE Admins (
    admin_id INT AUTO_INCREMENT PRIMARY KEY,
    user_name VARCHAR(100) NOT NULL,
    staff_id VARCHAR(50) UNIQUE, -- Maps to ElectionStaff staffID
    admin_password VARCHAR(100) NOT NULL -- Store hashed passwords in
production
);

-- Create ElectionOfficers table
CREATE TABLE ElectionOfficers (
    officer_id INT AUTO_INCREMENT PRIMARY KEY,
    user_name VARCHAR(100) NOT NULL,
    staff_id VARCHAR(50) UNIQUE -- Maps to ElectionStaff staffID
```

```
);

-- Create Voters table
CREATE TABLE Voters (
    voter_db_id INT AUTO_INCREMENT PRIMARY KEY, -- Internal DB ID
    voter_id VARCHAR(50) NOT NULL UNIQUE, -- Java generated ID (e.g.,
VOTER100)
    user_name VARCHAR(100) NOT NULL,
    voter_age INT NOT NULL,
    aadhaar_number VARCHAR(12) UNIQUE NOT NULL,
    address VARCHAR(255),
    has_voted BOOLEAN DEFAULT FALSE,
    constituency_id INT,
    FOREIGN KEY (constituency_id) REFERENCES Constituencies(constituency_id)
ON DELETE SET NULL
);

-- Create Candidates table
CREATE TABLE Candidates (
    candidate_id INT AUTO_INCREMENT PRIMARY KEY,
    candidate_name VARCHAR(100) NOT NULL,
    candidate_symbol VARCHAR(50),
    candidate_party VARCHAR(100) NOT NULL,
    vote_count INT DEFAULT 0,
    constituency_id INT NOT NULL,
    FOREIGN KEY (constituency_id) REFERENCES Constituencies(constituency_id)
ON DELETE CASCADE,
    UNIQUE KEY unique_candidate_party_constituency (candidate_party,
constituency_id) -- Ensure only one candidate per party per constituency
```

```
);

-- Create Votes table (to log individual votes, maintaining some anonymity as per original logic)
CREATE TABLE Votes (
    vote_id INT AUTO_INCREMENT PRIMARY KEY,
    candidate_id INT NOT NULL,
    constituency_id INT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (candidate_id) REFERENCES Candidates(candidate_id) ON DELETE CASCADE,
    FOREIGN KEY (constituency_id) REFERENCES Constituencies(constituency_id) ON DELETE CASCADE
);
```

PACKAGE CODES:

1. exceptions:

This package has all the custom exceptions in my code as individual JAVA files.

```
class SamePartyCandidateException extends Exception
{
    public SamePartyCandidateException(String message)
    {
        super(message);
    }
}
```

```
class AdminFailException extends Exception
{
   public AdminFailException(String message)
   {
      super(message);
   }
}

class CandidateNotFoundException extends Exception
{
   public CandidateNotFoundException(String message)
   {
      super(message);
   }
}

class InvalidVoteException extends Exception
{
   public InvalidVoteException(String message)
   {
      super(message);
   }
}

class InvalidAgeException extends Exception
{
   public InvalidAgeException(String message)
   {
```

```
        super(message);
    }
}

class RemoveVoterException extends Exception
{
    public RemoveVoterException(String message)
    {
        super(message);
    }
}

class VerifyVoterException extends Exception
{
    public VerifyVoterException(String message)
    {
        super(message);
    }
}
```

2. User:

This package has all the classes that represent users. Voters, Admin, Election Officers, Candidates are put in here. Abstract class user and election role also are present here. The 2 interfaces in my projects are also present in this package

Voter.java

```
package Users;

import System.DatabaseManager;
```

```
import System.EVM;
import System.VVPAT;
import exceptions.InvalidVoteException;
import java.sql.SQLException;

public class Voter extends User implements ElectionRole {
  private String voterID;
  private static int vIDcounter = 100;
  private boolean hasVoted;
  private int voterAge;
  private String aadhaarNumber;
  private String address;
  private int voterDbId;

  public Voter(String name, int voterAge, String aadhaarNumber, String address) {
     super(name);
     this.voterID = "VOTER" + vIDcounter++;
     this.voterAge = voterAge;
     this.aadhaarNumber = aadhaarNumber;
     this.address = address;
     this.hasVoted = false;
  }

  public Voter(String name, String aadhaarNumber) {
     super(name);
     this.aadhaarNumber = aadhaarNumber;
  }

  public void castVote(EVM evm, VVPAT vvpat, java.util.Scanner scan) {
```

```
    try {
        if (hasVoted) {
            throw new InvalidVoteException("You have already cast your vote!");
        }
        evm.displayCandidates();
        System.out.print("Enter the number of your chosen candidate: ");
        int choice = scan.nextInt();
        if (evm.castVote(choice)) {
            vvpat.generateSlip(evm.getCandidateList().get(choice - 1));
            vvpat.showSlip(evm.getCandidateList().get(choice - 1));
            vvpat.printSlip(evm.getCandidateList().get(choice - 1));
            hasVoted = true;
            DatabaseManager dbManager = new DatabaseManager();
            dbManager.updateVoterStatus(voterDbId, true);
            dbManager.addVote(voterDbId, EVM.constituency);
            System.out.println("Vote cast successfully!");
        } else {
            throw new InvalidVoteException("Invalid vote choice!");
        }
    } catch (InvalidVoteException e) {
        System.out.println("Exception Caught: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Database Error: " + e.getMessage());
    }
}

public void castVote(EVM evm, VVPAT vvpat, int candidateIndex) {
    try {
        if (hasVoted) {
```

```
            throw new InvalidVoteException("You have already cast your vote!");
        }
        if (evm.castVote(candidateIndex)) {
            vvpat.generateSlip(evm.getCandidateList().get(candidateIndex - 1));
            vvpat.showSlip(evm.getCandidateList().get(candidateIndex - 1));
            vvpat.printSlip(evm.getCandidateList().get(candidateIndex - 1));
            hasVoted = true;
            DatabaseManager dbManager = new DatabaseManager();
            dbManager.updateVoterStatus(voterDbId, true);
            dbManager.addVote(voterDbId, EVM.constituency);
        } else {
            throw new InvalidVoteException("Invalid vote choice!");
        }
    } catch (InvalidVoteException e) {
        System.out.println("Exception Caught: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Database Error: " + e.getMessage());
    }
}

@Override
public void accessSystem(EVM evm, java.util.Scanner scan) {
    castVote(evm, new VVPAT(), scan);
}

public boolean getVoteStatus() {
    return hasVoted;
}
```

```
    public String getVoterID() {
        return voterID;
    }

    public int getVoterAge() {
        return voterAge;
    }

    public String getAadhaarNumber() {
        return aadhaarNumber;
    }

    public String getAddress() {
        return address;
    }

    public int getVoterDbId() {
        return voterDbId;
    }

    public void setVoterDbId(int voterDbId) {
        this.voterDbId = voterDbId;
    }
}
```

User.java

```
package Users;

import System.EVM;
```

```
public abstract class User {
  private String userName;

  public User(String userName) {
    this.userName = userName;
  }

  abstract void accessSystem(EVM evm, java.util.Scanner scan);

  public String getUserName() {
    return userName;
  }
}
```

ElectionStaff.java

```
package Users;

public abstract class ElectionStaff extends User {
  private String staffID;
  private static int sIDcounter = 100;

  public ElectionStaff(String name) {
    super(name);
    this.staffID = "STAFF" + sIDcounter++;
  }

  public String getStaffID() {
    return staffID;
```

```
  }
}
```

ElectionRole.java

```
package Users;

import System.EVM;

public interface ElectionRole {
  void accessSystem(EVM evm, java.util.Scanner scan);
}
```

ElectionOfficer.java

```
package Users;

import System.DatabaseManager;
import System.EVM;
import exceptions.VerifyVoterException;
import java.sql.SQLException;

public class ElectionOfficer extends ElectionStaff implements ElectionRole {
  private String officerName;

  public ElectionOfficer(String officerName) {
    super(officerName);
    this.officerName = officerName;
  }

  public String getOfficerName() {
```

```
        return officerName;
    }

    public boolean verifyVoter(Voter voter) {
        try {
            DatabaseManager dbManager = new DatabaseManager();
            Voter dbVoter = dbManager.getVoter(voter.getAadhaarNumber());
            if (dbVoter == null) {
                throw new VerifyVoterException("Voter not found in database!");
            }
            if (dbVoter.getVoteStatus()) {
                throw new VerifyVoterException("Voter has already cast their vote!");
            }
            voter.setVoterDbId(dbVoter.getVoterDbId());
            return true;
        } catch (VerifyVoterException e) {
            System.out.println("Exception Caught: " + e.getMessage());
            return false;
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
            return false;
        }
    }

    @Override
    public void accessSystem(EVM evm, java.util.Scanner scan) {
        System.out.println("Election Officer can verify voters and manage the voting
process.");
        System.out.print("Enter voter's Aadhaar number: ");
```

```
        String aadhaarNumber = scan.nextLine();
        Voter voter = new Voter("", aadhaarNumber);
        if (verifyVoter(voter)) {
            System.out.println("Voter verified successfully!");
            voter.accessSystem(evm, scan);
        }
    }
}
```

DataManager.java

```
package Users;

import System.EVM;

public interface DataManager {
    void manageData(EVM evm);
}
```

Candidate.java

```
package Users;

public class Candidate {
    private String candidateName;
    private String candidateSymbol;
    private String candidateParty;
    private int countVotes;
    private int candidateId;

    Candidate(String name, String symbol, String candidateParty) {
```

```
    this.candidateName = name;
    this.candidateSymbol = symbol;
    this.candidateParty = candidateParty;
    this.countVotes = 0;
}

public Candidate(String name, String symbol, String candidateParty, int candidateId) {
    this.candidateName = name;
    this.candidateSymbol = symbol;
    this.candidateParty = candidateParty;
    this.candidateId = candidateId;
    this.countVotes = 0;
}

public String getCandidateName() {
    return candidateName;
}

public String getCandidateSymbol() {
    return candidateSymbol;
}

public String getCandidateParty() {
    return candidateParty;
}

public int getVoteCount() {
    return countVotes;
}
```

```
  public int getCandidateId() {
     return candidateId;
  }

  public void addVote() {
     countVotes++;
  }

  public void setVoteCount(int voteCount) {
     this.countVotes = voteCount;
  }

  public void setCandidateId(int candidateId) {
     this.candidateId = candidateId;
  }
}
```

Admin.java

```
package Users;

import System.EVM;
import System.DatabaseManager;
import exceptions.SamePartyCandidateException;
import exceptions.CandidateNotFoundException;
import exceptions.AdminFailException;
import java.sql.SQLException;

public class Admin extends ElectionStaff implements ElectionRole, DataManager {
```

```
  String adminPassword;

  public Admin(String name, String password) {
     super(name);
     this.adminPassword = password;
  }

  public void addCandidate(EVM evm, String candidateName, String candidateSymbol,
String candidateParty) {
     try {
        for (Candidate c : evm.getCandidateList()) {
           if (c.getCandidateParty().equalsIgnoreCase(candidateParty)) {
              throw new SamePartyCandidateException("There is already another
candidate contesting from this constituency. Only 1 candidate from each party is allowed
per faction!");
           }
        }
        Candidate newCandidate = new Candidate(candidateName, candidateSymbol,
candidateParty);
        evm.getCandidateList().add(newCandidate);
        DatabaseManager dbManager = new DatabaseManager();
        int candidateId = dbManager.addCandidate(candidateName, candidateSymbol,
candidateParty, EVM.constituency);
        newCandidate.setCandidateId(candidateId);
        System.out.println("Candidate " + candidateName + " Representing: " +
candidateParty + " has been added successfully.\n");
     } catch (SamePartyCandidateException e) {
        System.out.println("Exception Caught: " + e.getMessage());
     } catch (SQLException e) {
```

```
            System.out.println("Database Error: " + e.getMessage());
        }
    }

    public void removeCandidate(EVM evm, String candidateName, String candidateParty)
{
        try {
            java.util.Iterator<Candidate> iterator = evm.getCandidateList().iterator();
            int candidateId = -1;
            while (iterator.hasNext()) {
                Candidate c = iterator.next();
                if (c.getCandidateName().equalsIgnoreCase(candidateName) &&
c.getCandidateParty().equalsIgnoreCase(candidateParty)) {
                    candidateId = c.getCandidateId();
                    iterator.remove();
                    System.out.println("Candidate " + c.getCandidateName() + " Representing "
+ c.getCandidateParty() + " has been removed successfully.\n");
                    DatabaseManager dbManager = new DatabaseManager();
                    dbManager.removeCandidate(candidateId);
                    return;
                }
            }
            throw new CandidateNotFoundException("The specified candidate does not exist
in the list. Please check spelling again.");
        } catch (CandidateNotFoundException e) {
            System.out.println("Exception Caught: " + e.getMessage());
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
```

```
    }

    public void viewResults(EVM evm) {
        try {
            System.out.println("\n--- Election Results ---");
            DatabaseManager dbManager = new DatabaseManager();
            java.util.List<Candidate> candidates =
dbManager.getCandidates(EVM.constituency);
            if (candidates.isEmpty()) {
                throw new CandidateNotFoundException("No candidates registered.");
            }
            for (Candidate c : candidates) {
                System.out.println(c.getCandidateName() + " (" + c.getCandidateParty() + ") ("
+ c.getCandidateSymbol() + "): " + c.getVoteCount() + " total votes");
            }
        } catch (CandidateNotFoundException e) {
            System.out.println("Exception Caught: " + e.getMessage());
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
    }

    @Override
    public void accessSystem(EVM evm, java.util.Scanner scan) {
        try {
            System.out.println("Admin can configure system, manage candidates, and view
results.\n");
            System.out.print("Enter admin password: ");
            String inputPassword = scan.nextLine();
```

```
        DatabaseManager dbManager = new DatabaseManager();
        if (!dbManager.verifyAdmin(getUserName(), inputPassword)) {
            throw new AdminFailException("Invalid password. Alerting Electoral
Commission about unauthorized login attempt!\n");
        }
        System.out.println("Admin has been approved, Welcome " + getUserName() +
".");
        while (true) {
            System.out.println("\n1. Add Candidate | 2. Remove Candidate | 3. View
Results | 4. Exit Admin Mode");
            int choice = scan.nextInt();
            scan.nextLine();
            if (choice == 1) {
                System.out.print("Enter candidate name: ");
                String name = scan.nextLine();
                System.out.print("Enter candidate symbol: ");
                String symbol = scan.nextLine();
                System.out.print("Enter candidate party: ");
                String party = scan.nextLine();
                addCandidate(evm, name, symbol, party);
            } else if (choice == 2) {
                System.out.print("Enter candidate name to remove: ");
                String name = scan.nextLine();
                System.out.print("Enter candidate party: ");
                String party = scan.nextLine();
                removeCandidate(evm, name, party);
            } else if (choice == 3) {
                viewResults(evm);
            } else if (choice == 4) {
```

```
                break;
            }
        }
    } catch (AdminFailException e) {
        System.out.println("Exception Caught: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Database Error: " + e.getMessage());
    }
  }

  @Override
  public void manageData(EVM evm) {
    // Implementation of data management for admin
  }
}
```

3. System

VVPAT.java

```
package System;

import Users.Candidate;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;

public class VVPAT {
  public void generateSlip(Candidate candidate) {
```

```
        try (BufferedWriter writer = new BufferedWriter(new FileWriter("vvpat_slip.txt"))) {
            writer.write("Voter Verifiable Paper Audit Trail\n");
            writer.write("------------------------------\n");
            writer.write("Candidate Name: " + candidate.getCandidateName() + "\n");
            writer.write("Party: " + candidate.getCandidateParty() + "\n");
            writer.write("Symbol: " + candidate.getCandidateSymbol() + "\n");
            writer.write("Timestamp: " + LocalDateTime.now() + "\n");
            writer.write("------------------------------\n");
            writer.write("This is your VVPAT slip. Please verify your vote.\n");
        } catch (IOException e) {
            System.out.println("Error generating VVPAT slip: " + e.getMessage());
        }
    }

    public void showSlip(Candidate candidate) {
        System.out.println("\n--- VVPAT Slip ---");
        System.out.println("Candidate Name: " + candidate.getCandidateName());
        System.out.println("Party: " + candidate.getCandidateParty());
        System.out.println("Symbol: " + candidate.getCandidateSymbol());
        System.out.println("Timestamp: " + LocalDateTime.now());
        System.out.println("-------------------");
        System.out.println("Please verify your vote.");
    }

    public void printSlip(Candidate candidate) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter("printed_slips.txt",
true))) {
            writer.write("Voter Verifiable Paper Audit Trail\n");
            writer.write("------------------------------\n");
```

```
            writer.write("Candidate Name: " + candidate.getCandidateName() + "\n");
            writer.write("Party: " + candidate.getCandidateParty() + "\n");
            writer.write("Symbol: " + candidate.getCandidateSymbol() + "\n");
            writer.write("Timestamp: " + LocalDateTime.now() + "\n");
            writer.write("--------------------------------\n\n");
        } catch (IOException e) {
            System.out.println("Error printing VVPAT slip: " + e.getMessage());
        }
    }
}
```

EVM.java

```
package System;

import Users.Candidate;
import java.util.ArrayList;
import java.util.List;
import java.time.LocalDateTime;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class EVM {
    private ArrayList<Candidate> candidateList = new ArrayList<>();
    private static final String master_admin_password = "VUadmin@123";
    public static String constituency;
    private List<Ballot> voteBuffer = new ArrayList<>();
    private int voterCounter = 1;
```

```
    public void displayCandidates() {
        System.out.println("\n--- List of Candidates ---");
        for (int i = 0; i < candidateList.size(); i++) {
            Candidate c = candidateList.get(i);
            System.out.println((i + 1) + ". " + c.getCandidateName() + " (" +
c.getCandidateParty() + ") (" + c.getCandidateSymbol() + ")");
        }
    }

    public boolean castVote(int buttonPressed) {
        if (buttonPressed < 1 || buttonPressed > candidateList.size()) {
            return false;
        }
        Candidate selectedCandidate = candidateList.get(buttonPressed - 1);
        selectedCandidate.addVote();
        String anonymizedID = "V" + voterCounter++;
        Ballot ballot = new Ballot(anonymizedID, selectedCandidate.getCandidateName());
        voteBuffer.add(ballot);
        return true;
    }

    public static boolean verifyAdmin(String inputPassword) {
        return inputPassword.equals(master_admin_password);
    }

    public ArrayList<Candidate> getCandidateList() {
        return candidateList;
    }
```

```
  public void shineRedLight() {
     System.out.println("Red light is shining");
  }

  public void logVote(String voterID, String candidateID) {
     try (BufferedWriter writer = new BufferedWriter(new FileWriter("vote_log.txt",
true))) {
        writer.write(voterID + "," + candidateID + "," + LocalDateTime.now() + "\n");
     } catch (IOException e) {
        System.out.println("Error writing to vote log: " + e.getMessage());
     }
  }

  public void writeVotesToCSV() {
     try (BufferedWriter writer = new BufferedWriter(new FileWriter("votes.csv"))) {
        writer.write("VoterID,Candidate,Timestamp\n");
        for (Ballot ballot : voteBuffer) {
           writer.write(ballot.anonymizedID + "," + ballot.candidateID + "," +
ballot.timestamp + "\n");
        }
     } catch (IOException e) {
        System.out.println("Error writing to CSV: " + e.getMessage());
     }
  }

  private class Ballot {
     String anonymizedID;
     String candidateID;
```

```
    LocalDateTime timestamp;

    Ballot(String voterID, String candidateID) {
        this.anonymizedID = voterID;
        this.candidateID = candidateID;
        this.timestamp = LocalDateTime.now();
    }
  }
}
```

ElectionCommission

```
package System;

import Users.Voter;
import Users.ElectionRole;
import Users.DataManager;
import exceptions.InvalidAgeException;
import exceptions.RemoveVoterException;
import java.util.HashMap;
import java.sql.SQLException;

public class ElectionCommission implements ElectionRole, DataManager {
  private HashMap<String, Voter> voterList = new HashMap<>();

  public void registerVoter(String name, int voterAge, String aadhaarNumber, String
address) {
    try {
        if (voterAge < 18) {
```

```
            throw new InvalidAgeException("Voter must be at least 18 years old to
register!");
        }
        Voter newVoter = new Voter(name, voterAge, aadhaarNumber, address);
        DatabaseManager dbManager = new DatabaseManager();
        int voterDbId = dbManager.addVoter(newVoter.getVoterID(), name, voterAge,
aadhaarNumber, address, EVM.constituency);
        newVoter.setVoterDbId(voterDbId);
        voterList.put(aadhaarNumber, newVoter);
        System.out.println("Voter " + name + " has been registered successfully!");
    } catch (InvalidAgeException e) {
        System.out.println("Exception Caught: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Database Error: " + e.getMessage());
    }
  }

  public void removeVoter(String name, String aadhaarNumber, int reason) {
    try {
        Voter voter = voterList.get(aadhaarNumber);
        if (voter == null) {
            throw new RemoveVoterException("Voter not found in the system!");
        }
        if (!voter.getUserName().equals(name)) {
            throw new RemoveVoterException("Name and Aadhaar number do not
match!");
        }
        DatabaseManager dbManager = new DatabaseManager();
        dbManager.removeVoter(voter.getVoterDbId());
```

```
            voterList.remove(aadhaarNumber);
            System.out.println("Voter " + name + " has been removed from the system.");
        } catch (RemoveVoterException e) {
            System.out.println("Exception Caught: " + e.getMessage());
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
        }
    }

    public void viewAllVoters() {
        try {
            System.out.println("\n--- Registered Voters ---");
            DatabaseManager dbManager = new DatabaseManager();
            java.util.List<Voter> voters = dbManager.getVoters(EVM.constituency);
            if (voters.isEmpty()) {
                System.out.println("No voters registered yet.");
                return;
            }
            for (Voter v : voters) {
                System.out.println("Name: " + v.getUserName());
                System.out.println("Voter ID: " + v.getVoterID());
                System.out.println("Age: " + v.getVoterAge());
                System.out.println("Aadhaar: " + v.getAadhaarNumber());
                System.out.println("Address: " + v.getAddress());
                System.out.println("Voted: " + (v.getVoteStatus() ? "Yes" : "No"));
                System.out.println("--------------------");
            }
        } catch (SQLException e) {
            System.out.println("Database Error: " + e.getMessage());
```

```
    }
  }

  public HashMap<String, Voter> getVoterList() {
    return voterList;
  }

  @Override
  public void accessSystem(EVM evm, java.util.Scanner scan) {
    System.out.println("Election Commission can manage voters and oversee the
election process.");
    while (true) {
      System.out.println("\n1. Register Voter | 2. Remove Voter | 3. View All Voters | 4.
Exit");
      int choice = scan.nextInt();
      scan.nextLine();
      if (choice == 1) {
        System.out.print("Enter voter name: ");
        String name = scan.nextLine();
        System.out.print("Enter voter age: ");
        int age = scan.nextInt();
        scan.nextLine();
        System.out.print("Enter Aadhaar number: ");
        String aadhaar = scan.nextLine();
        System.out.print("Enter address: ");
        String address = scan.nextLine();
        registerVoter(name, age, aadhaar, address);
      } else if (choice == 2) {
        System.out.print("Enter voter name: ");
```

```
            String name = scan.nextLine();
            System.out.print("Enter Aadhaar number: ");
            String aadhaar = scan.nextLine();
            System.out.print("Enter reason (1-3): ");
            int reason = scan.nextInt();
            scan.nextLine();
            removeVoter(name, aadhaar, reason);
        } else if (choice == 3) {
            viewAllVoters();
        } else if (choice == 4) {
            break;
        }
    }
  }

  @Override
  public void manageData(EVM evm) {
      }
}
```

DatabaseManager.java

```
package System;

import Users.Voter;
import Users.Candidate;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class DatabaseManager {
  private static final String DB_URL = "jdbc:mysql://localhost:3306/ElectionSystem";
  private static final String USER = "root";
  private static final String PASS = "bLACKPANTHER123#";

  public static String customHash(String input) {
    StringBuilder result = new StringBuilder();
    for (char c : input.toCharArray()) {
      result.append((int) c);
    }
    return result.toString();
  }

  public static Connection getConnection() throws SQLException {
    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
      return DriverManager.getConnection(DB_URL, USER, PASS);
    } catch (ClassNotFoundException e) {
      throw new SQLException("MySQL JDBC Driver not found", e);
    }
  }
```

```
  public void initializeAdmin(String userName, String password, String staffId) throws
SQLException {
    String sql = "INSERT INTO Admins (user_name, admin_password, staff_id)
VALUES (?, ?, ?) " +
            "ON DUPLICATE KEY UPDATE user_name=user_name";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, userName);
        pstmt.setString(2, customHash(password));
        pstmt.setString(3, staffId);
        pstmt.executeUpdate();
    }
  }

  public boolean verifyAdmin(String userName, String password) throws SQLException
{
    String sql = "SELECT * FROM Admins WHERE user_name = ? AND
admin_password = ?";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, userName);
        pstmt.setString(2, customHash(password));
        try (ResultSet rs = pstmt.executeQuery()) {
            return rs.next();
        }
    }
  }
```

```
  public int addVoter(String voterId, String name, int age, String aadhaarNumber, String address, String constituencyName) throws SQLException {
      String sql = "INSERT INTO Voters (voter_id, user_name, voter_age, aadhaar_number, address, constituency_id) VALUES (?, ?, ?, ?, ?, ?)";
      try (Connection conn = getConnection();
           PreparedStatement pstmt = conn.prepareStatement(sql, PreparedStatement.RETURN_GENERATED_KEYS)) {
          pstmt.setString(1, voterId);
          pstmt.setString(2, name);
          pstmt.setInt(3, age);
          pstmt.setString(4, aadhaarNumber);
          pstmt.setString(5, address);
          pstmt.setInt(6, getConstituencyId(constituencyName));
          pstmt.executeUpdate();
          try (ResultSet rs = pstmt.getGeneratedKeys()) {
              if (rs.next()) {
                  return rs.getInt(1);
              }
          }
      }
      return -1;
  }

  public void updateVoterStatus(int voterDbId, boolean hasVoted) throws SQLException {
      String sql = "UPDATE Voters SET has_voted = ? WHERE voter_db_id = ?";
      try (Connection conn = getConnection();
           PreparedStatement pstmt = conn.prepareStatement(sql)) {
          pstmt.setBoolean(1, hasVoted);
```

```
            pstmt.setInt(2, voterDbId);
            pstmt.executeUpdate();
        }
    }

    public Voter getVoter(String aadhaarNumber) throws SQLException {
        String sql = "SELECT * FROM Voters WHERE aadhaar_number = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, aadhaarNumber);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    Voter voter = new Voter(rs.getString("user_name"), rs.getInt("voter_age"),
rs.getString("aadhaar_number"), rs.getString("address"));
                    voter.setVoterDbId(rs.getInt("voter_db_id"));
                    return voter;
                }
            }
        }
        return null;
    }

    public void removeVoter(int voterDbId) throws SQLException {
        String sql = "DELETE FROM Voters WHERE voter_db_id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, voterDbId);
            pstmt.executeUpdate();
        }
```

```
    }

    public List<Voter> getVoters(String constituencyName) throws SQLException {
        List<Voter> voters = new ArrayList<>();
        String sql = "SELECT * FROM Voters WHERE constituency_id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, getConstituencyId(constituencyName));
            try (ResultSet rs = pstmt.executeQuery()) {
                while (rs.next()) {
                    Voter voter = new Voter(rs.getString("user_name"), rs.getInt("voter_age"),
rs.getString("aadhaar_number"), rs.getString("address"));
                    voter.setVoterDbId(rs.getInt("voter_db_id"));
                    voters.add(voter);
                }
            }
        }
        return voters;
    }

    public int addCandidate(String name, String symbol, String party, String
constituencyName) throws SQLException {
        String sql = "INSERT INTO Candidates (candidate_name, candidate_symbol,
candidate_party, constituency_id) VALUES (?, ?, ?, ?)";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql,
PreparedStatement.RETURN_GENERATED_KEYS)) {
            pstmt.setString(1, name);
            pstmt.setString(2, symbol);
```

```
            pstmt.setString(3, party);
            pstmt.setInt(4, getConstituencyId(constituencyName));
            pstmt.executeUpdate();
            try (ResultSet rs = pstmt.getGeneratedKeys()) {
                if (rs.next()) {
                    return rs.getInt(1);
                }
            }
        }
        return -1;
    }

    public void removeCandidate(int candidateId) throws SQLException {
        String sql = "DELETE FROM Candidates WHERE candidate_id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, candidateId);
            pstmt.executeUpdate();
        }
    }

    public List<Candidate> getCandidates(String constituencyName) throws
SQLException {
        List<Candidate> candidates = new ArrayList<>();
        String sql = "SELECT * FROM Candidates WHERE constituency_id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, getConstituencyId(constituencyName));
            try (ResultSet rs = pstmt.executeQuery()) {
```

```
            while (rs.next()) {
                Candidate candidate = new Candidate(
                    rs.getString("candidate_name"),
                    rs.getString("candidate_symbol"),
                    rs.getString("candidate_party"),
                    rs.getInt("candidate_id")
                );
                candidate.setVoteCount(rs.getInt("vote_count"));
                candidates.add(candidate);
            }
        }
    }
    return candidates;
  }

  public void updateCandidateVoteCount(int candidateId) throws SQLException {
    String sql = "UPDATE Candidates SET vote_count = vote_count + 1 WHERE
candidate_id = ?";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, candidateId);
        pstmt.executeUpdate();
    }
  }

  public void addVote(int candidateId, String constituencyName) throws SQLException {
    String sql = "INSERT INTO Votes (candidate_id, constituency_id, timestamp)
VALUES (?, ?, ?)";
    try (Connection conn = getConnection();
```

```
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, candidateId);
        pstmt.setInt(2, getConstituencyId(constituencyName));
        pstmt.setTimestamp(3, Timestamp.valueOf(LocalDateTime.now()));
        pstmt.executeUpdate();
        updateCandidateVoteCount(candidateId);
    }
  }

  public List<VoteRecord> getVotes(String constituencyName) throws SQLException {
    List<VoteRecord> votes = new ArrayList<>();
    String sql = "SELECT v.vote_id, v.candidate_id, c.candidate_name,
c.candidate_party, v.timestamp " +
                 "FROM Votes v JOIN Candidates c ON v.candidate_id = c.candidate_id " +
                 "WHERE v.constituency_id = ?";
    try (Connection conn = getConnection();
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, getConstituencyId(constituencyName));
        try (ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                votes.add(new VoteRecord(
                    rs.getInt("vote_id"),
                    rs.getInt("candidate_id"),
                    rs.getString("candidate_name"),
                    rs.getString("candidate_party"),
                    rs.getTimestamp("timestamp").toLocalDateTime()
                ));
            }
        }
```

```
        }
        return votes;
    }

    public int addConstituency(String name) throws SQLException {
        String sql = "INSERT INTO Constituencies (constituency_name) VALUES (?)";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql,
PreparedStatement.RETURN_GENERATED_KEYS)) {
            pstmt.setString(1, name);
            pstmt.executeUpdate();
            try (ResultSet rs = pstmt.getGeneratedKeys()) {
                if (rs.next()) {
                    return rs.getInt(1);
                }
            }
        }
        return getConstituencyId(name);
    }

    public int getConstituencyId(String constituencyName) throws SQLException {
        String sql = "SELECT constituency_id FROM Constituencies WHERE
constituency_name = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, constituencyName);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt("constituency_id");
```

```
            }
        }
    }
    return -1;
  }

  public static class VoteRecord {
    int voteId;
    int candidateId;
    String candidateName;
    String candidateParty;
    LocalDateTime timestamp;

    VoteRecord(int voteId, int candidateId, String candidateName, String candidateParty,
LocalDateTime timestamp) {
        this.voteId = voteId;
        this.candidateId = candidateId;
        this.candidateName = candidateName;
        this.candidateParty = candidateParty;
        this.timestamp = timestamp;
    }
  }
}
```

4. Main

ElectionSystem.java

```
package Main;
```

```
import Users.Admin;
import Users.Candidate;
import Users.ElectionOfficer;
import Users.Voter;
import System.EVM;
import System.VVPAT;
import System.ElectionCommission;
import System.DatabaseManager;
import java.util.Scanner;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.util.List;


// ElectionSystem class
class ElectionSystem
{
  private static final Scanner scan = new Scanner(System.in);
  public static ElectionCommission ec = new ElectionCommission();
  Admin admin;
  EVM evm;
  VVPAT vvpat;
  private Voter currentVoter;

  // GUI Components
```

```
    private JFrame frame;
    private CardLayout cardLayout;
    private JPanel cards;
    private JPanel welcomePanel;
    private JPanel mainMenuPanel;
    private JPanel adminLoginPanel;
    private JPanel adminPanel;
    private JPanel voterVerificationPanel;
    private JPanel votingPanel;
    private JPanel ecPanel;
    private JPanel resultsPanel;
    private JTextArea resultsTextArea;

    void startElection()
    {
        System.out.println("\nElection started in constituency: " + EVM.constituency);
        while (true)
        {
            System.out.println("\nWho is interacting with the system?");
            System.out.println("1. Admin | 2. Voter | 3. Election Commission | 4. View Results
| 5. End Election");
            int choice = scan.nextInt();
            scan.nextLine();

            if (choice == 1)
            {
                admin.accessSystem(evm, scan);
            }
            else if (choice == 2)
```

```
{
    System.out.print("Enter Aadhaar number: ");
    String aadhaar = scan.nextLine();
    Voter voter = ec.getVoterList().get(aadhaar);
    if (voter != null)
    {
        voter.accessSystem(evm, scan);
        ElectionOfficer officer = new ElectionOfficer("Officer1");
        if (officer.verifyVoter(voter))
        {
            voter.castVote(evm, vvpat, scan);
        }
        else
        {
            System.out.println("Voter verification failed. Cannot vote.");
        }
    }
    else
    {
        System.out.println("Voter not found.");
    }
}
else if (choice == 3)
{
    ec.accessSystem(evm, scan);
}
else if (choice == 4)
{
    if (admin != null)
```

```
            {
                admin.viewResults(evm);
            }
            else
            {
                System.out.println("Admin not initialized. Please log in as Admin first.");
            }
        }
        else if (choice == 5)
        {
            evm.writeVotesToCSV();
            break;
        }
        else
        {
            System.out.println("Invalid choice. Please select 1, 2, 3, 4, or 5.");
        }
    }
}

void countVotes()
{
    try
    {
        System.out.println("\n--- Vote Count ---");
        DatabaseManager dbManager = new DatabaseManager();
        List<Candidate> candidates = dbManager.getCandidates(EVM.constituency);
        if (candidates.isEmpty())
        {
```

```
            throw new SQLException("No candidates found.");
        }
        for (Candidate c : candidates)
        {
            System.out.println(c.getCandidateName() + " (" + c.getCandidateParty() + "): "
+ c.getVoteCount() + " votes");
        }
    }
    catch (SQLException e)
    {
        System.out.println("Database Error: " + e.getMessage());
    }
}

public Candidate findWinner()
{
    try
    {
        DatabaseManager dbManager = new DatabaseManager();
        List<Candidate> candidates = dbManager.getCandidates(EVM.constituency);
        if (candidates.isEmpty())
        {
            throw new SQLException("No candidates found.");
        }
        Candidate winner = candidates.get(0);
        boolean allTied = true;

        for (Candidate c : candidates)
        {
```

```
            if (c.getVoteCount() > winner.getVoteCount())
            {
                winner = c;
                allTied = false;
            }
            else if (c.getVoteCount() < winner.getVoteCount())
            {
                allTied = false;
            }
        }

        if (allTied)
        {
            throw new SQLException("All candidates have the same number of votes.
Constituency results in a DRAW!");
        }
        return winner;
    }
    catch (SQLException e)
    {
        System.out.println("Database Error: " + e.getMessage());
        return null;
    }
}

public void startGUI()
{
    frame = new JFrame("Election Voting Machine");
    frame.setSize(600, 400);
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);

        cardLayout = new CardLayout();
        cards = new JPanel(cardLayout);
        frame.add(cards);

        // Welcome Panel
        welcomePanel = new JPanel();
        welcomePanel.setLayout(new BoxLayout(welcomePanel, BoxLayout.Y_AXIS));
        welcomePanel.setBackground(Color.LIGHT_GRAY);
        JLabel titleLabel = new JLabel("Election Voting Machine");
        titleLabel.setFont(new Font("Arial", Font.BOLD, 24));
        titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        JLabel constituencyLabel = new JLabel("Enter Constituency:");
        JTextField constituencyField = new JTextField(20);
        JButton setConstituencyButton = new JButton("Set Constituency");
        setConstituencyButton.addActionListener(e -> {
            String constituency = constituencyField.getText().trim();
            if (!constituency.isEmpty()) {
                EVM.constituency = constituency;
                try {
                    DatabaseManager dbManager = new DatabaseManager();
                    dbManager.addConstituency(constituency);
                    initializeData(dbManager);
                    cardLayout.show(cards, "mainMenu");
                } catch (SQLException ex) {
                    JOptionPane.showMessageDialog(frame, "Error adding constituency: " +
ex.getMessage());
```

```
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Please enter a constituency name.");
    }
});
welcomePanel.add(Box.createVerticalGlue());
welcomePanel.add(titleLabel);
welcomePanel.add(Box.createRigidArea(new Dimension(0, 20)));
welcomePanel.add(constituencyLabel);
welcomePanel.add(constituencyField);
welcomePanel.add(setConstituencyButton);
welcomePanel.add(Box.createVerticalGlue());
cards.add(welcomePanel, "welcome");

// Main Menu Panel
mainMenuPanel = new JPanel();
mainMenuPanel.setLayout(new BoxLayout(mainMenuPanel, BoxLayout.Y_AXIS));
mainMenuPanel.setBackground(Color.LIGHT_GRAY);
JLabel roleLabel = new JLabel("Please Select your Role");
roleLabel.setFont(new Font("Arial", Font.BOLD, 18));
roleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
mainMenuPanel.add(roleLabel);
mainMenuPanel.add(Box.createRigidArea(new Dimension(0, 20)));
JButton adminButton = new JButton("Admin");
adminButton.setMaximumSize(new Dimension(200, 30));
adminButton.setAlignmentX(Component.CENTER_ALIGNMENT);
adminButton.addActionListener(e -> cardLayout.show(cards, "adminLogin"));
mainMenuPanel.add(adminButton);
mainMenuPanel.add(Box.createRigidArea(new Dimension(0, 10)));
```

```
JButton voterButton = new JButton("Voter");
voterButton.setMaximumSize(new Dimension(200, 30));
voterButton.setAlignmentX(Component.CENTER_ALIGNMENT);
voterButton.addActionListener(e -> cardLayout.show(cards, "voterVerification"));
mainMenuPanel.add(voterButton);
mainMenuPanel.add(Box.createRigidArea(new Dimension(0, 10)));
JButton ecButton = new JButton("Election Commission");
ecButton.setMaximumSize(new Dimension(200, 30));
ecButton.setAlignmentX(Component.CENTER_ALIGNMENT);
ecButton.addActionListener(e -> cardLayout.show(cards, "ecPanel"));
mainMenuPanel.add(ecButton);
mainMenuPanel.add(Box.createRigidArea(new Dimension(0, 10)));
JButton resultsButton = new JButton("View Results");
resultsButton.setMaximumSize(new Dimension(200, 30));
resultsButton.setAlignmentX(Component.CENTER_ALIGNMENT);
resultsButton.addActionListener(e -> {
    updateResultsPanel();
    cardLayout.show(cards, "resultsPanel");
});
mainMenuPanel.add(resultsButton);
mainMenuPanel.add(Box.createRigidArea(new Dimension(0, 10)));
JButton endElectionButton = new JButton("End Election");
endElectionButton.setMaximumSize(new Dimension(200, 30));
endElectionButton.setAlignmentX(Component.CENTER_ALIGNMENT);
endElectionButton.addActionListener(e -> {
    Candidate winner = findWinner();
    String message;
    if (winner != null) {
```

```
            message = "Election Winner: " + winner.getCandidateName() + " (" +
winner.getCandidateParty() + ") with " + winner.getVoteCount() + " votes!";
        } else {
            message = "All candidates have the same number of votes. Constituency results
in a DRAW!";
        }
        JOptionPane.showMessageDialog(frame, message, "Election Results",
JOptionPane.INFORMATION_MESSAGE);
        evm.writeVotesToCSV();
        System.exit(0);
    });
    mainMenuPanel.add(endElectionButton);
    cards.add(mainMenuPanel, "mainMenu");

    // Admin Login Panel
    adminLoginPanel = new JPanel();
    adminLoginPanel.setLayout(new BoxLayout(adminLoginPanel,
BoxLayout.Y_AXIS));
    adminLoginPanel.setBackground(Color.LIGHT_GRAY);
    JLabel passwordLabel = new JLabel("Enter Admin Password:");
    JPasswordField passwordField = new JPasswordField(20);
    JButton loginButton = new JButton("Login");
    loginButton.addActionListener(e -> {
        String password = new String(passwordField.getPassword());
        try {
            DatabaseManager dbManager = new DatabaseManager();
            if (dbManager.verifyAdmin(admin.getUserName(), password)) {
                cardLayout.show(cards, "adminPanel");
            } else {
```

```
                JOptionPane.showMessageDialog(frame, "Invalid password.");
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(frame, "Error verifying admin: " +
ex.getMessage());
        }
        passwordField.setText("");
    });
    adminLoginPanel.add(Box.createVerticalGlue());
    adminLoginPanel.add(passwordLabel);
    adminLoginPanel.add(passwordField);
    adminLoginPanel.add(loginButton);
    adminLoginPanel.add(Box.createVerticalGlue());
    cards.add(adminLoginPanel, "adminLogin");

    // Admin Panel
    adminPanel = new JPanel();
    adminPanel.setLayout(new BoxLayout(adminPanel, BoxLayout.Y_AXIS));
    adminPanel.setBackground(Color.LIGHT_GRAY);
    JLabel addCandidateLabel = new JLabel("Add Candidate");
    JTextField nameField = new JTextField(20);
    JTextField symbolField = new JTextField(20);
    JTextField partyField = new JTextField(20);
    JButton addButton = new JButton("Add");
    addButton.addActionListener(e -> {
        String name = nameField.getText().trim();
        String symbol = symbolField.getText().trim();
        String party = partyField.getText().trim();
        if (!name.isEmpty() && !symbol.isEmpty() && !party.isEmpty()) {
```

```
            admin.addCandidate(evm, name, symbol, party);
            nameField.setText("");
            symbolField.setText("");
            partyField.setText("");
        } else {
            JOptionPane.showMessageDialog(frame, "Please fill all fields.");
        }
    });
    JLabel removeCandidateLabel = new JLabel("Remove Candidate");
    JTextField removeNameField = new JTextField(20);
    JTextField removePartyField = new JTextField(20);
    JButton removeButton = new JButton("Remove");
    removeButton.addActionListener(e -> {
        String name = removeNameField.getText().trim();
        String party = removePartyField.getText().trim();
        if (!name.isEmpty() && !party.isEmpty()) {
            admin.removeCandidate(evm, name, party);
            removeNameField.setText("");
            removePartyField.setText("");
        } else {
            JOptionPane.showMessageDialog(frame, "Please fill all fields.");
        }
    });
    JButton viewResultsButton = new JButton("View Results");
    viewResultsButton.addActionListener(e -> {
        updateResultsPanel();
        cardLayout.show(cards, "resultsPanel");
    });
    JButton exitAdminButton = new JButton("Exit Admin Mode");
```

```
        exitAdminButton.addActionListener(e -> cardLayout.show(cards, "mainMenu"));
        adminPanel.add(addCandidateLabel);
        adminPanel.add(new JLabel("Name:"));
        adminPanel.add(nameField);
        adminPanel.add(new JLabel("Symbol:"));
        adminPanel.add(symbolField);
        adminPanel.add(new JLabel("Party:"));
        adminPanel.add(partyField);
        adminPanel.add(addButton);
        adminPanel.add(Box.createRigidArea(new Dimension(0, 10)));
        adminPanel.add(removeCandidateLabel);
        adminPanel.add(new JLabel("Name:"));
        adminPanel.add(removeNameField);
        adminPanel.add(new JLabel("Party:"));
        adminPanel.add(removePartyField);
        adminPanel.add(removeButton);
        adminPanel.add(viewResultsButton);
        adminPanel.add(exitAdminButton);
        cards.add(adminPanel, "adminPanel");

        // Voter Verification Panel
        voterVerificationPanel = new JPanel();
        voterVerificationPanel.setLayout(new BoxLayout(voterVerificationPanel,
BoxLayout.Y_AXIS));
        voterVerificationPanel.setBackground(Color.LIGHT_GRAY);
        JLabel aadhaarLabel = new JLabel("Enter Aadhaar number:");
        JTextField aadhaarField = new JTextField(20);
        JButton verifyButton = new JButton("Verify");
        verifyButton.addActionListener(e -> {
```

```
    String aadhaar = aadhaarField.getText().trim();
    Voter voter = ec.getVoterList().get(aadhaar);
    if (voter != null) {
        ElectionOfficer officer = new ElectionOfficer("Officer1");
        if (officer.verifyVoter(voter)) {
            currentVoter = voter;
            updateVotingPanel();
            cardLayout.show(cards, "votingPanel");
        } else {
            JOptionPane.showMessageDialog(frame, "Voter verification failed.");
        }
    } else {
        JOptionPane.showMessageDialog(frame, "Voter not found.");
    }
    aadhaarField.setText("");
});
voterVerificationPanel.add(Box.createVerticalGlue());
voterVerificationPanel.add(aadhaarLabel);
voterVerificationPanel.add(aadhaarField);
voterVerificationPanel.add(verifyButton);
voterVerificationPanel.add(Box.createVerticalGlue());
cards.add(voterVerificationPanel, "voterVerification");

// Voting Panel
votingPanel = new JPanel();
votingPanel.setLayout(new BoxLayout(votingPanel, BoxLayout.Y_AXIS));
votingPanel.setBackground(Color.LIGHT_GRAY);
cards.add(votingPanel, "votingPanel");
```

```
// Election Commission Panel
ecPanel = new JPanel();
ecPanel.setLayout(new BoxLayout(ecPanel, BoxLayout.Y_AXIS));
ecPanel.setBackground(Color.LIGHT_GRAY);
JLabel registerLabel = new JLabel("Register Voter");
JTextField regNameField = new JTextField(20);
JTextField regAgeField = new JTextField(20);
JTextField regAadhaarField = new JTextField(20);
JTextField regAddressField = new JTextField(20);
JButton registerButton = new JButton("Register");
registerButton.addActionListener(e -> {
    String name = regNameField.getText().trim();
    String ageText = regAgeField.getText().trim();
    String aadhaar = regAadhaarField.getText().trim();
    String address = regAddressField.getText().trim();
    try {
        int age = Integer.parseInt(ageText);
        if (!name.isEmpty() && !aadhaar.isEmpty() && !address.isEmpty()) {
            ec.registerVoter(name, age, aadhaar, address);
            regNameField.setText("");
            regAgeField.setText("");
            regAadhaarField.setText("");
            regAddressField.setText("");
            JOptionPane.showMessageDialog(frame, "Voter registered successfully!");
        } else {
            JOptionPane.showMessageDialog(frame, "Please fill all fields.");
        }
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(frame, "Please enter a valid age.");
```

```
        }
    });
    JLabel removeLabel = new JLabel("Remove Voter");
    JTextField remNameField = new JTextField(20);
    JTextField remAadhaarField = new JTextField(20);
    String[] reasons = {"Duplicate Entry", "Voter's Demise", "Change of Address",
"Disqualification", "Non-existent", "NRI Citizenship"};
    JComboBox<String> reasonComboBox = new JComboBox<>(reasons);
    JButton removeEcButton = new JButton("Remove");
    removeEcButton.addActionListener(e -> {
        String name = remNameField.getText().trim();
        String aadhaar = remAadhaarField.getText().trim();
        int reason = reasonComboBox.getSelectedIndex() + 1;
        if (!name.isEmpty() && !aadhaar.isEmpty()) {
            ec.removeVoter(name, aadhaar, reason);
            remNameField.setText("");
            remAadhaarField.setText("");
            JOptionPane.showMessageDialog(frame, "Voter removed successfully!");
        } else {
            JOptionPane.showMessageDialog(frame, "Please fill all fields.");
        }
    });
    JButton viewVotersButton = new JButton("View All Voters");
    viewVotersButton.addActionListener(e -> {
        try {
            DatabaseManager dbManager = new DatabaseManager();
            List<Voter> voters = dbManager.getVoters(EVM.constituency);
            StringBuilder sb = new StringBuilder();
            if (voters.isEmpty()) {
```

```
                sb.append("No voters registered.");
            } else {
                for (Voter v : voters) {
                    sb.append("ID: ").append(v.getVoterID())
                      .append(", Name: ").append(v.getUserName())
                      .append(", Age: ").append(v.getVoterAge())
                      .append(", Aadhaar: ").append(v.getAadhaarNumber())
                      .append(", Address: ").append(v.getAddress())
                      .append("\n");
                }
            }
            JOptionPane.showMessageDialog(frame, sb.toString(), "Registered Voters",
JOptionPane.INFORMATION_MESSAGE);
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(frame, "Database Error: " +
ex.getMessage());
        }
    });
    JButton exitEcButton = new JButton("Exit EC Mode");
    exitEcButton.addActionListener(e -> cardLayout.show(cards, "mainMenu"));
    ecPanel.add(registerLabel);
    ecPanel.add(new JLabel("Name:"));
    ecPanel.add(regNameField);
    ecPanel.add(new JLabel("Age:"));
    ecPanel.add(regAgeField);
    ecPanel.add(new JLabel("Aadhaar:"));
    ecPanel.add(regAadhaarField);
    ecPanel.add(new JLabel("Address:"));
    ecPanel.add(regAddressField);
```

```
    ecPanel.add(registerButton);
    ecPanel.add(Box.createRigidArea(new Dimension(0, 10)));
    ecPanel.add(removeLabel);
    ecPanel.add(new JLabel("Name:"));
    ecPanel.add(remNameField);
    ecPanel.add(new JLabel("Aadhaar:"));
    ecPanel.add(remAadhaarField);
    ecPanel.add(new JLabel("Reason:"));
    ecPanel.add(reasonComboBox);
    ecPanel.add(removeEcButton);
    ecPanel.add(viewVotersButton);
    ecPanel.add(exitEcButton);
    cards.add(ecPanel, "ecPanel");

    // Results Panel
    resultsPanel = new JPanel();
    resultsPanel.setLayout(new BorderLayout());
    resultsPanel.setBackground(Color.LIGHT_GRAY);
    resultsTextArea = new JTextArea(10, 40);
    resultsTextArea.setEditable(false);
    resultsPanel.add(new JScrollPane(resultsTextArea), BorderLayout.CENTER);
    JButton backButton = new JButton("Back to Main Menu");
    backButton.addActionListener(e -> cardLayout.show(cards, "mainMenu"));
    resultsPanel.add(backButton, BorderLayout.SOUTH);
    cards.add(resultsPanel, "resultsPanel");

    frame.setVisible(true);
    cardLayout.show(cards, "welcome");
}
```

```
  private void initializeData(DatabaseManager dbManager) throws SQLException
  {
      dbManager.initializeAdmin("Admin1", "VUadmin@123", "STAFF100");

      // Add predefined candidates
      admin.addCandidate(evm, "Sowymya Reddy", "Hand", "Indian National Congress");
      admin.addCandidate(evm, "Arun Prasad A", "Elephant", "Bahujan Samaj Party");
      admin.addCandidate(evm, "Tejasvi Surya", "Lotus", "Bharatiya Janata Party");
      admin.addCandidate(evm, "Tintisha Hemachandra Sagar", "---", "Independent");

      // Add predefined voters
      ec.registerVoter("Ramesh Kumar", 35, "123456789012", "Jayanagar, Bengaluru
South");
      ec.registerVoter("Priya Sharma", 28, "987654321098", "Basavanagudi, Bengaluru
South");
      ec.registerVoter("Anil Gowda", 45, "456789123456", "Koramangala, Bengaluru
South");
      ec.registerVoter("Lakshmi Devi", 50, "789123456789", "Banashankari, Bengaluru
South");
      ec.registerVoter("Suresh Patil", 32, "321654987123", "JP Nagar, Bengaluru South");
      ec.registerVoter("Meena Rao", 40, "654987321456", "BTM Layout, Bengaluru
South");
      ec.registerVoter("Kiran Shetty", 29, "147258369012", "Vijayanagar, Bengaluru
South");
      ec.registerVoter("Vikram Singh", 38, "258369147012", "Mysore City, Mysore");
      ec.registerVoter("Asha Nair", 33, "369147258012", "Hubli Central, Hubli");
      ec.registerVoter("Rahul Menon", 47, "741852963012", "Mangalore North,
Mangalore");
```

```
        // Sync evm candidate list with database
        List<Candidate> dbCandidates = dbManager.getCandidates(EVM.constituency);
        evm.getCandidateList().clear();
        evm.getCandidateList().addAll(dbCandidates);
    }

    private void updateVotingPanel()
    {
        votingPanel.removeAll();
        JLabel selectLabel = new JLabel("Select Candidate to Vote For:");
        selectLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        votingPanel.add(selectLabel);
        try {
            DatabaseManager dbManager = new DatabaseManager();
            List<Candidate> candidates = dbManager.getCandidates(EVM.constituency);
            if (candidates.isEmpty()) {
                JOptionPane.showMessageDialog(frame, "No candidates available to vote
for.");
                cardLayout.show(cards, "mainMenu");
                return;
            }
            for (int i = 0; i < candidates.size(); i++) {
                Candidate c = candidates.get(i);
                JButton voteButton = new JButton(c.getCandidateName() + " (" +
c.getCandidateParty() + ") - " + c.getCandidateSymbol());
                voteButton.setMaximumSize(new Dimension(400, 30));
                voteButton.setAlignmentX(Component.CENTER_ALIGNMENT);
                final int index = i + 1;
```

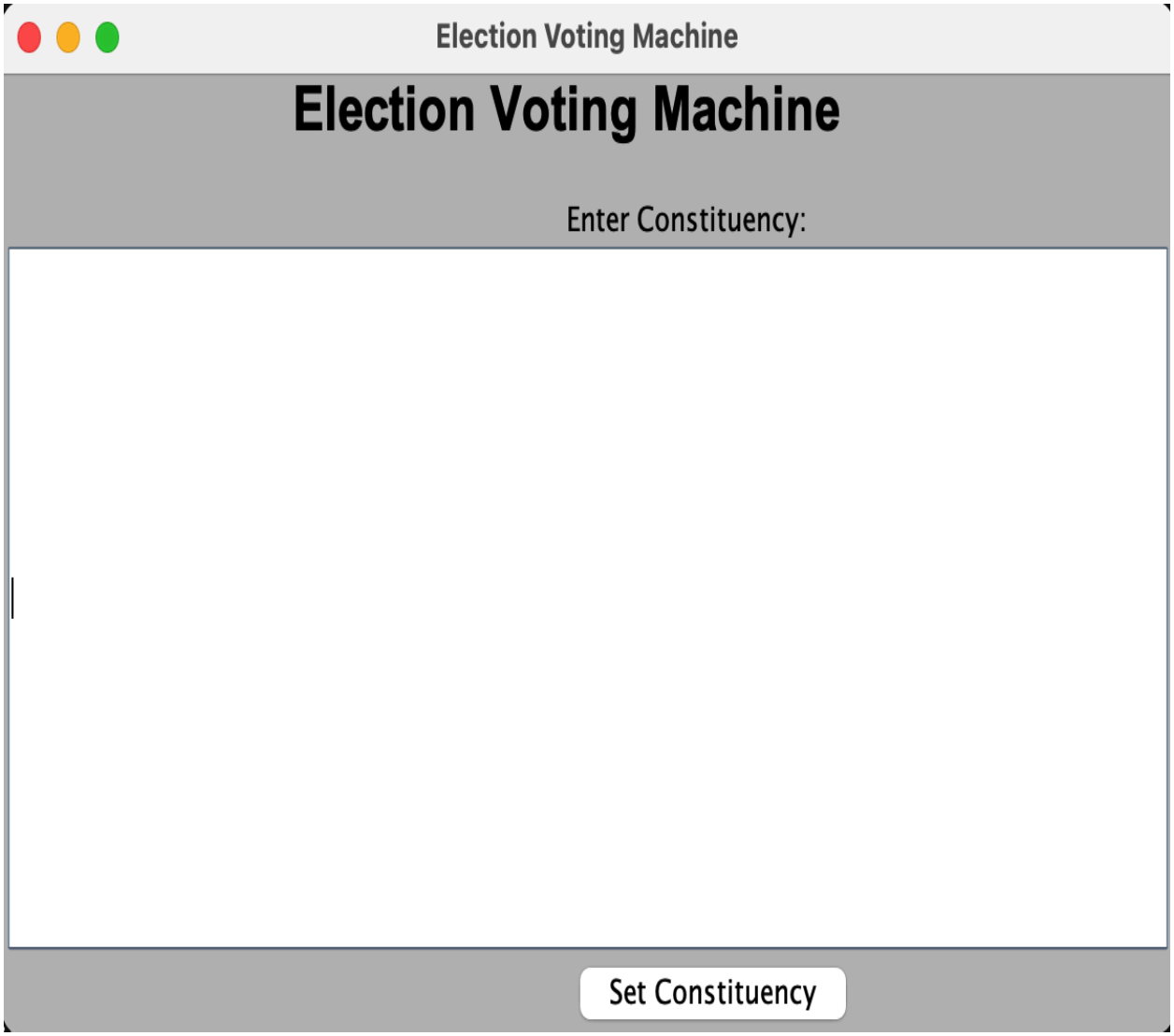
```
        voteButton.addActionListener(e -> {
            try {
                if (currentVoter.getVoteStatus()) {
                    JOptionPane.showMessageDialog(frame, "You have already voted. Duplicate voting is not allowed.");
                    cardLayout.show(cards, "mainMenu");
                    return;
                }
                Candidate selected = candidates.get(index - 1);

                // Simulate EVM and VVPAT actions (display only, do NOT increment in memory)
                evm.shineRedLight(); // Red light logic from EVM
                JOptionPane.showMessageDialog(frame, "🚨🚨🚨🚨 Red light ON. Beep! 🚨🚨🚨🚨", "EVM Status", JOptionPane.INFORMATION_MESSAGE);

                // VVPAT logic
                vvpat.generateSlip(selected);
                vvpat.showSlip(selected);
                vvpat.printSlip(selected);

                // Update database (this increments the vote count)
                dbManager.addVote(selected.getCandidateId(), EVM.constituency);
                dbManager.updateVoterStatus(currentVoter.getVoterDbId(), true);

                // Mark voter as voted in memory
                // currentVoter.castVote(evm, vvpat, index); // REMOVE or comment out this line!
```

```
                JOptionPane.showMessageDialog(frame, "Vote cast successfully!");
                cardLayout.show(cards, "mainMenu");
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(frame, "Database Error: " +
ex.getMessage());
            }
        });
        votingPanel.add(voteButton);
        votingPanel.add(Box.createRigidArea(new Dimension(0, 5)));
      }
      JButton exitButton = new JButton("Exit Voting");
      exitButton.setMaximumSize(new Dimension(400, 30));
      exitButton.setAlignmentX(Component.CENTER_ALIGNMENT);
      exitButton.addActionListener(e -> cardLayout.show(cards, "mainMenu"));
      votingPanel.add(Box.createRigidArea(new Dimension(0, 10)));
      votingPanel.add(exitButton);
    } catch (SQLException e) {
      JOptionPane.showMessageDialog(frame, "Database Error: " + e.getMessage());
    }
    votingPanel.revalidate();
    votingPanel.repaint();
  }

  private void updateResultsPanel()
  {
    try {
      DatabaseManager dbManager = new DatabaseManager();
      List<Candidate> candidates = dbManager.getCandidates(EVM.constituency);
      StringBuilder sb = new StringBuilder("--- Election Results ---\n");
```

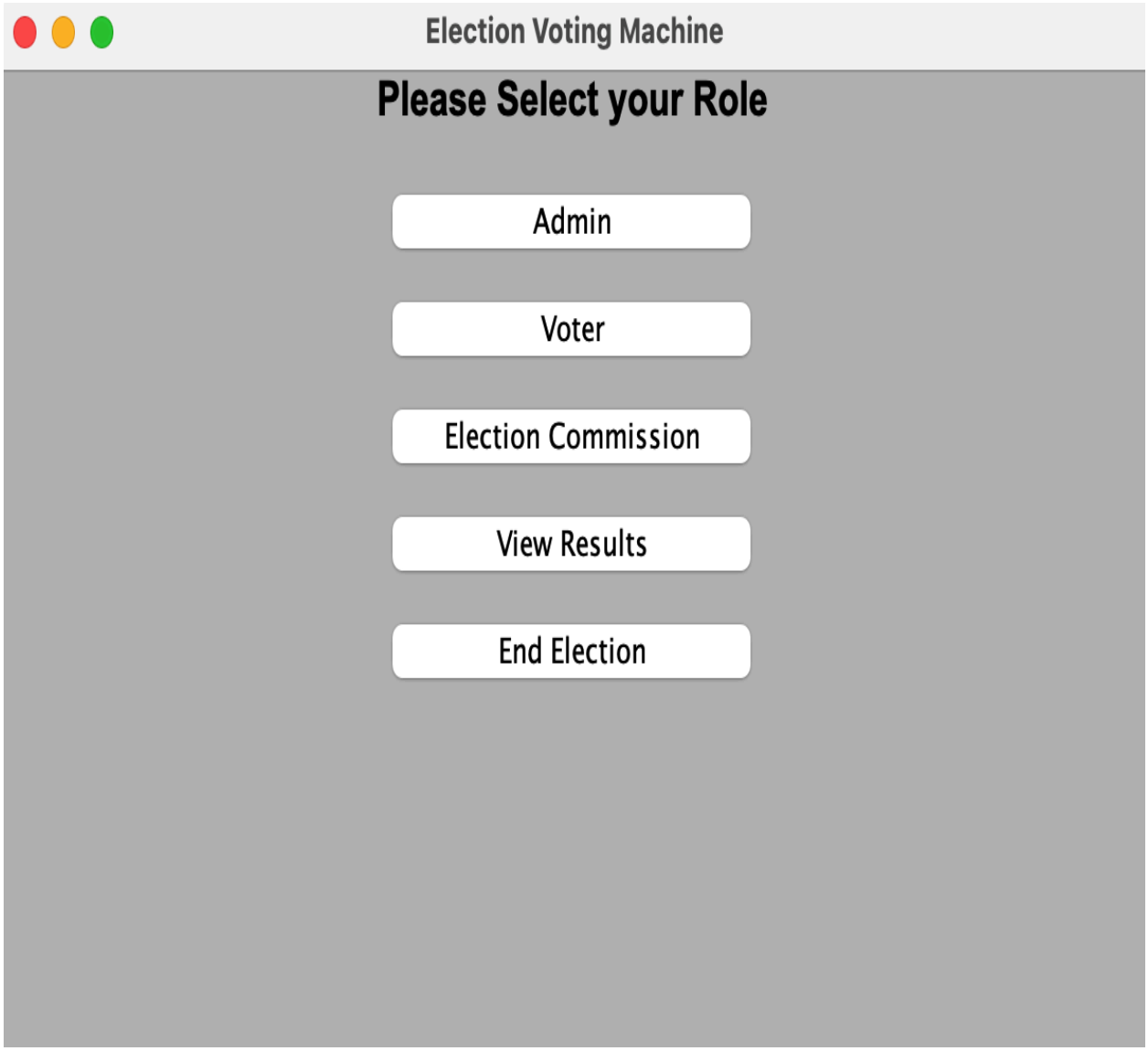
```
        if (candidates.isEmpty()) {
            sb.append("No candidates registered.");
        } else {
            for (Candidate c : candidates) {
                sb.append(c.getCandidateName()).append(" (")
                  .append(c.getCandidateParty()).append(") (")
                  .append(c.getCandidateSymbol()).append("): ")
                  .append(c.getVoteCount()).append(" total votes\n");
            }
        }
        resultsTextArea.setText(sb.toString());
    } catch (SQLException e) {
        resultsTextArea.setText("Database Error: " + e.getMessage());
    }
}

public static void main(String[] args)
{
    try
    {
        ElectionSystem system = new ElectionSystem();
        system.admin = new Admin("Admin1", "VUadmin@123");
        system.evm = new EVM();
        system.vvpat = new VVPAT();
        system.startGUI();
    }
    catch (Exception e)
    {
        System.out.println("Error starting GUI: " + e.getMessage());
```

```
        }
    }
}
```

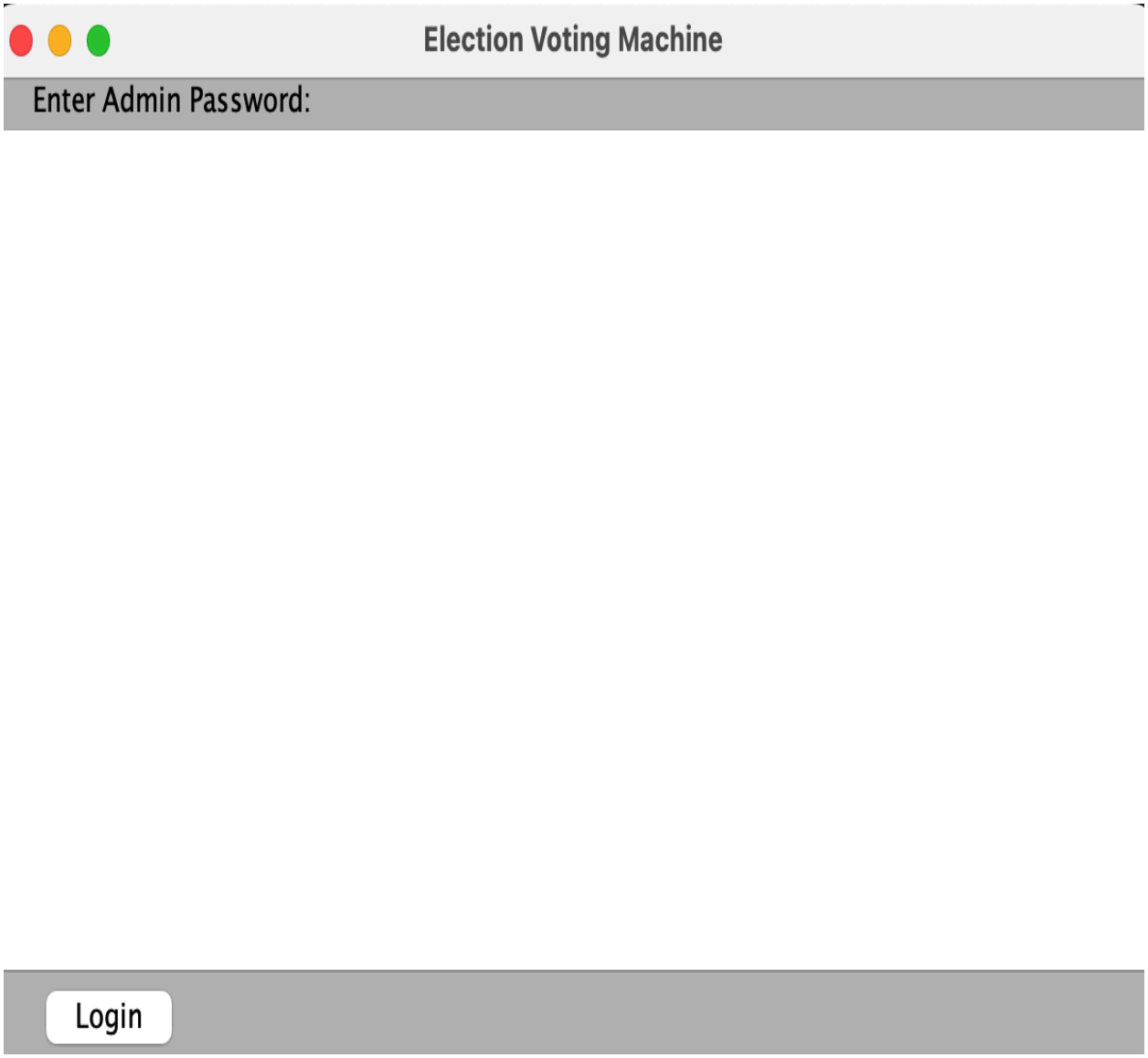
Output:



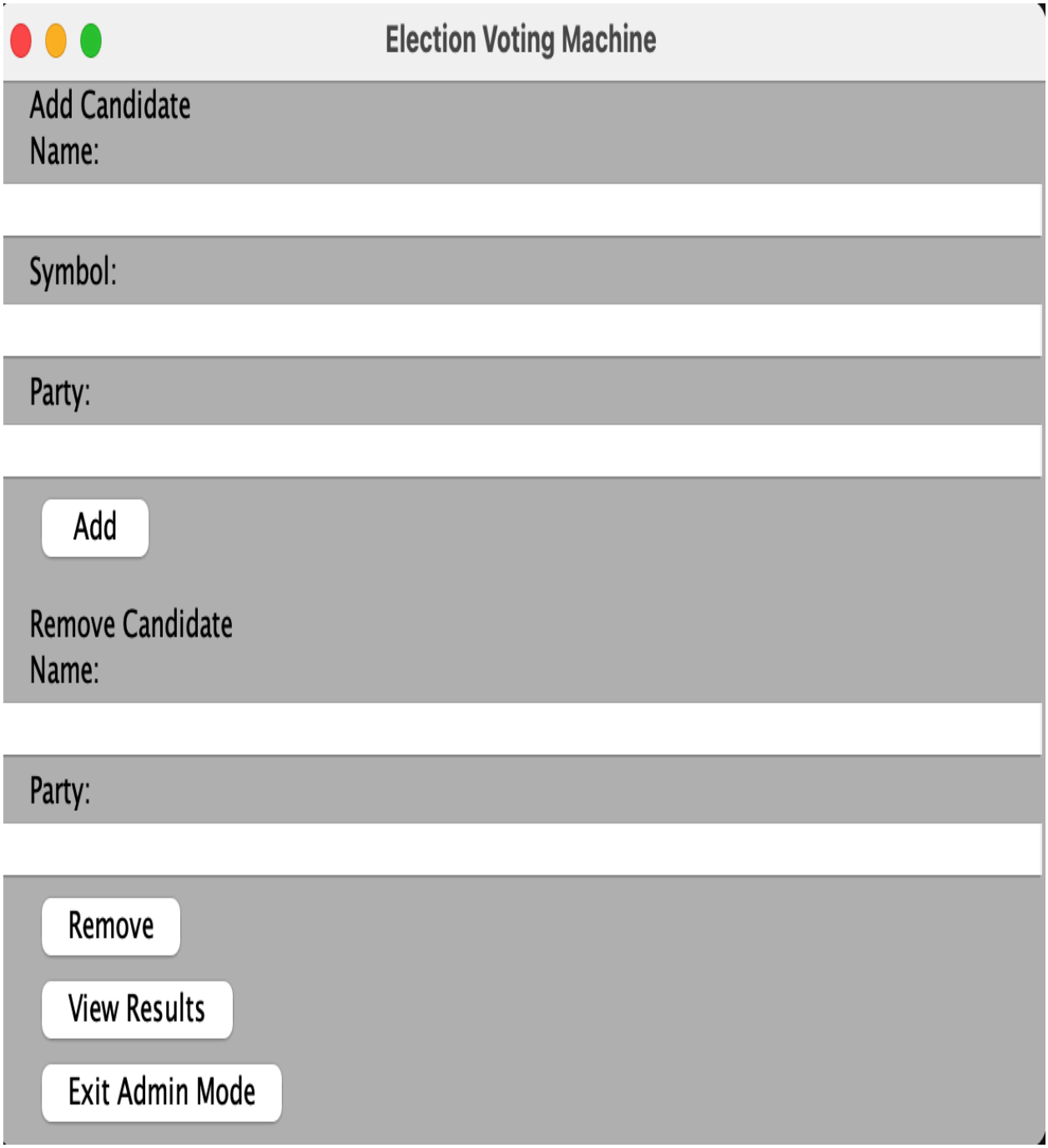
Opening Page



Select Role Page

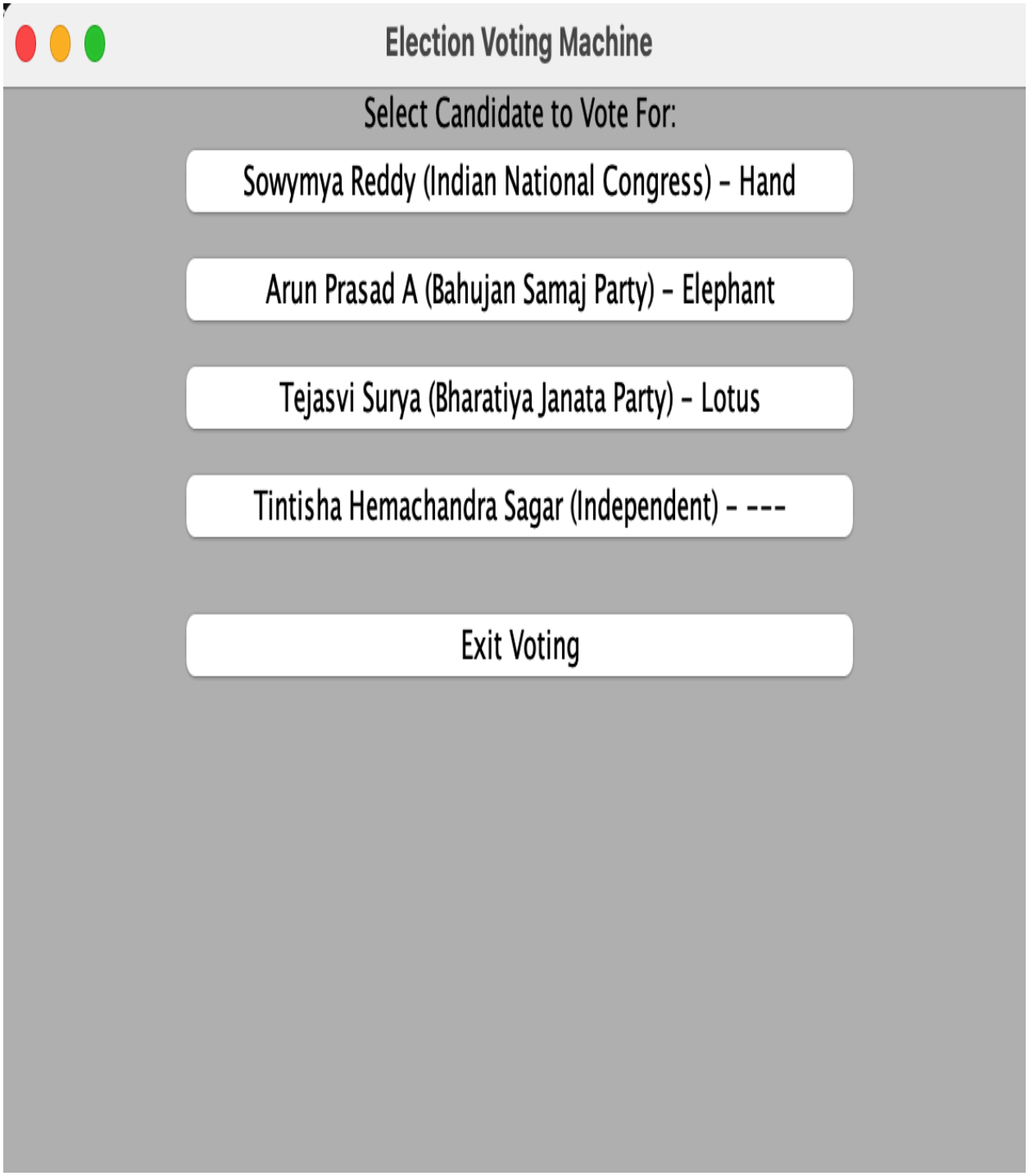


Page After Selecting Admin - We need to enter Admin Password



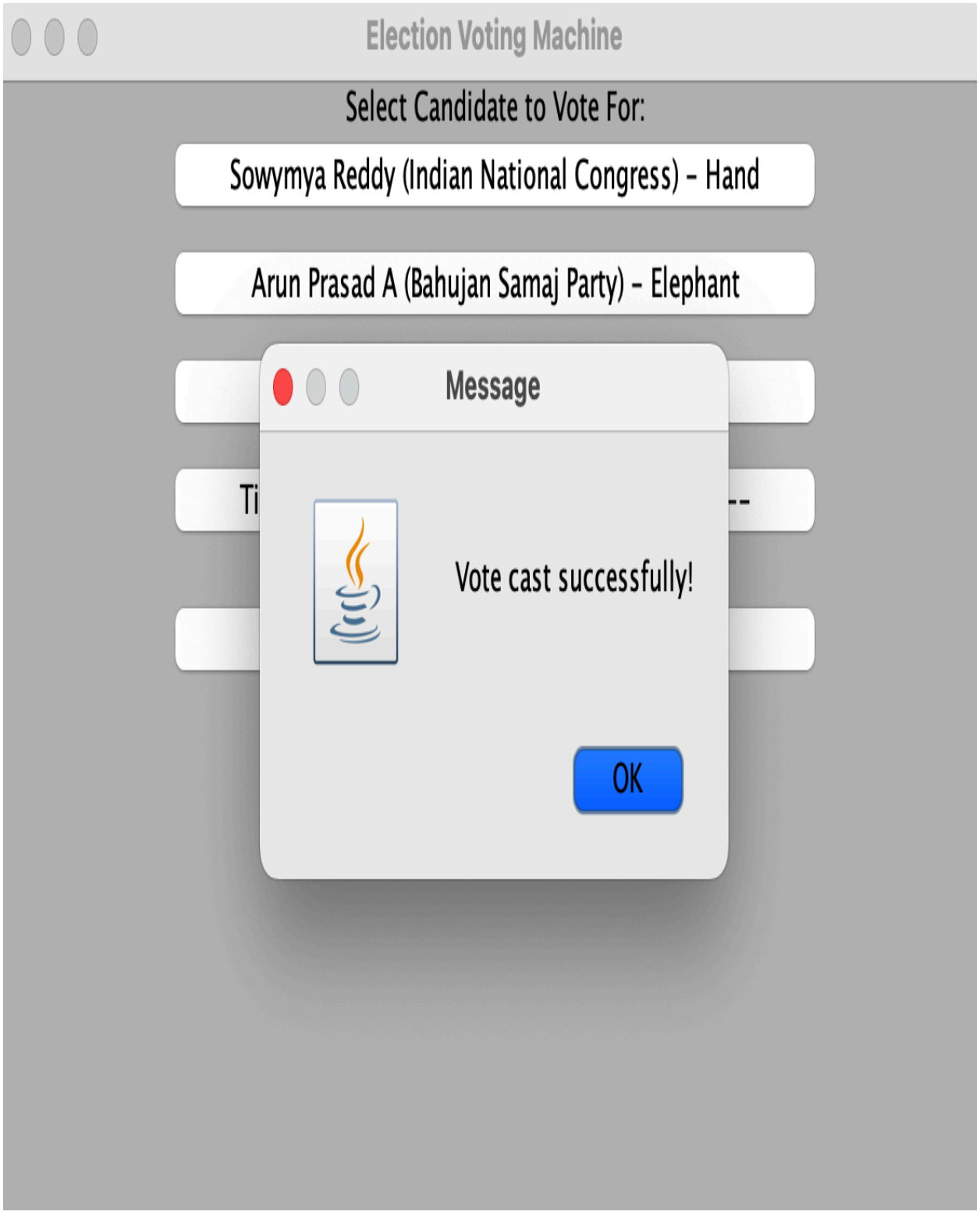
Page After Entering correct Admin Password

Page After choosing Voter in the Role Page - We need to put in Voter Aadhar

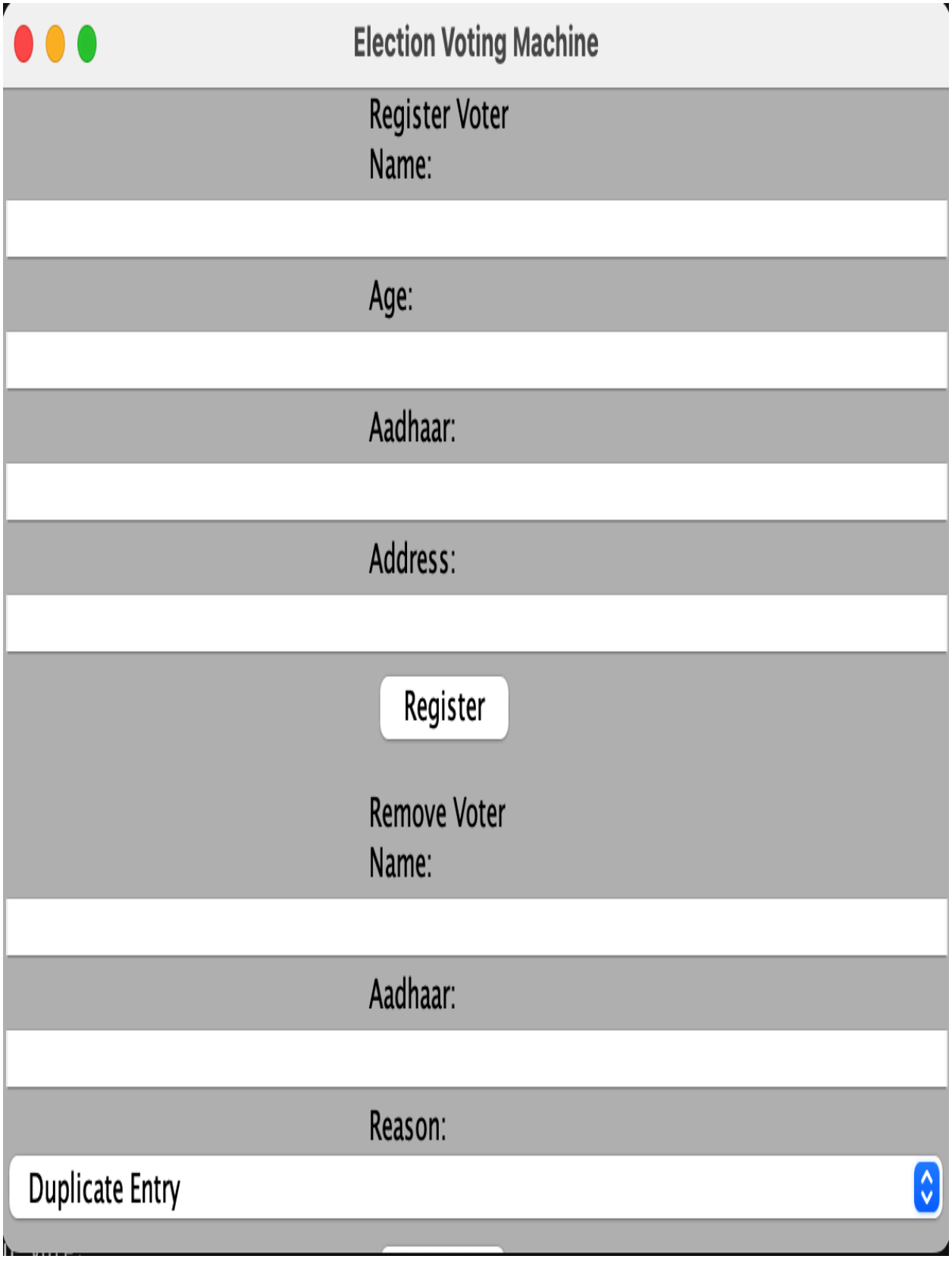


Page After confirming Voter existence and eligibility to vote - The voter gets to choose who to vote for

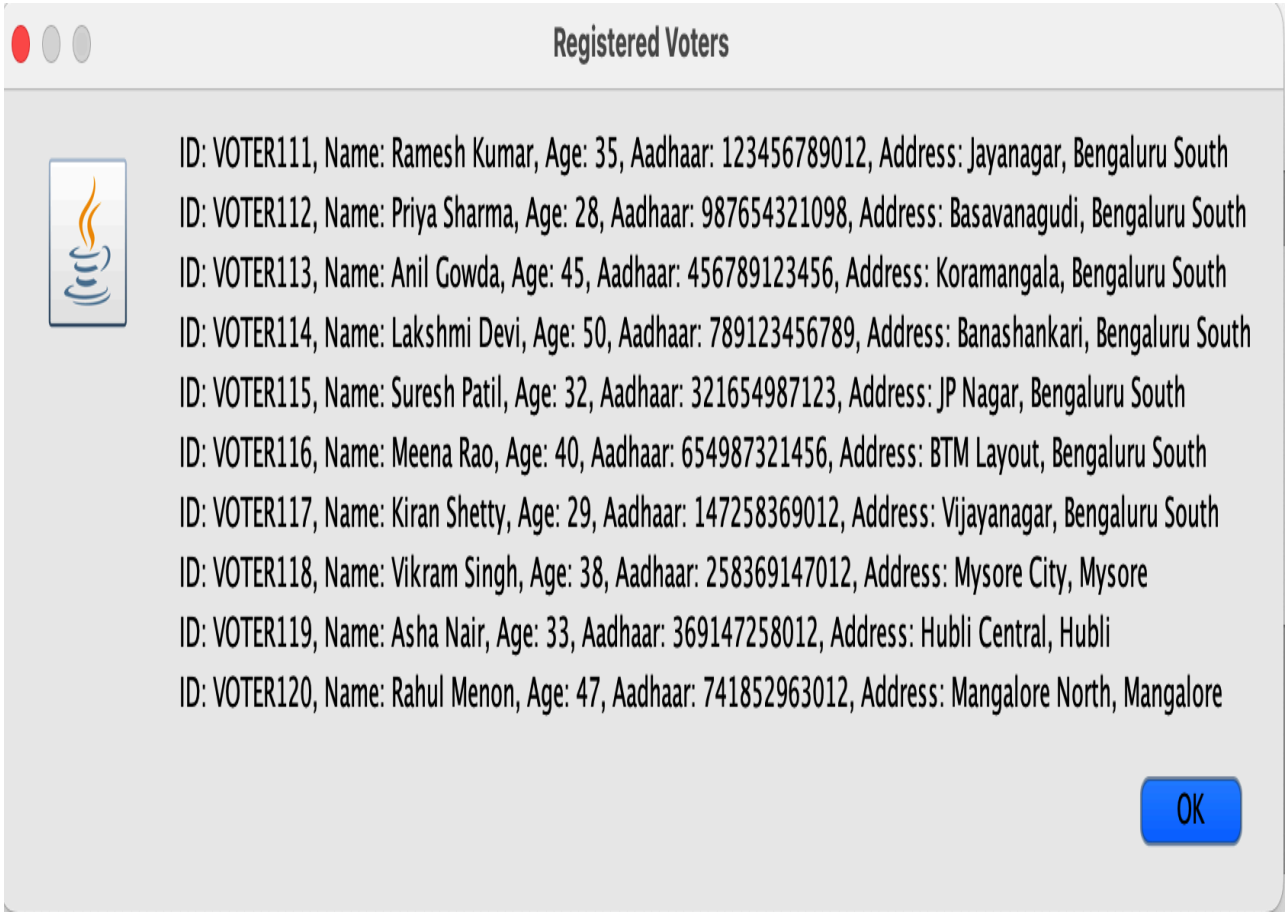
Page After Voter votes for desired Candidate



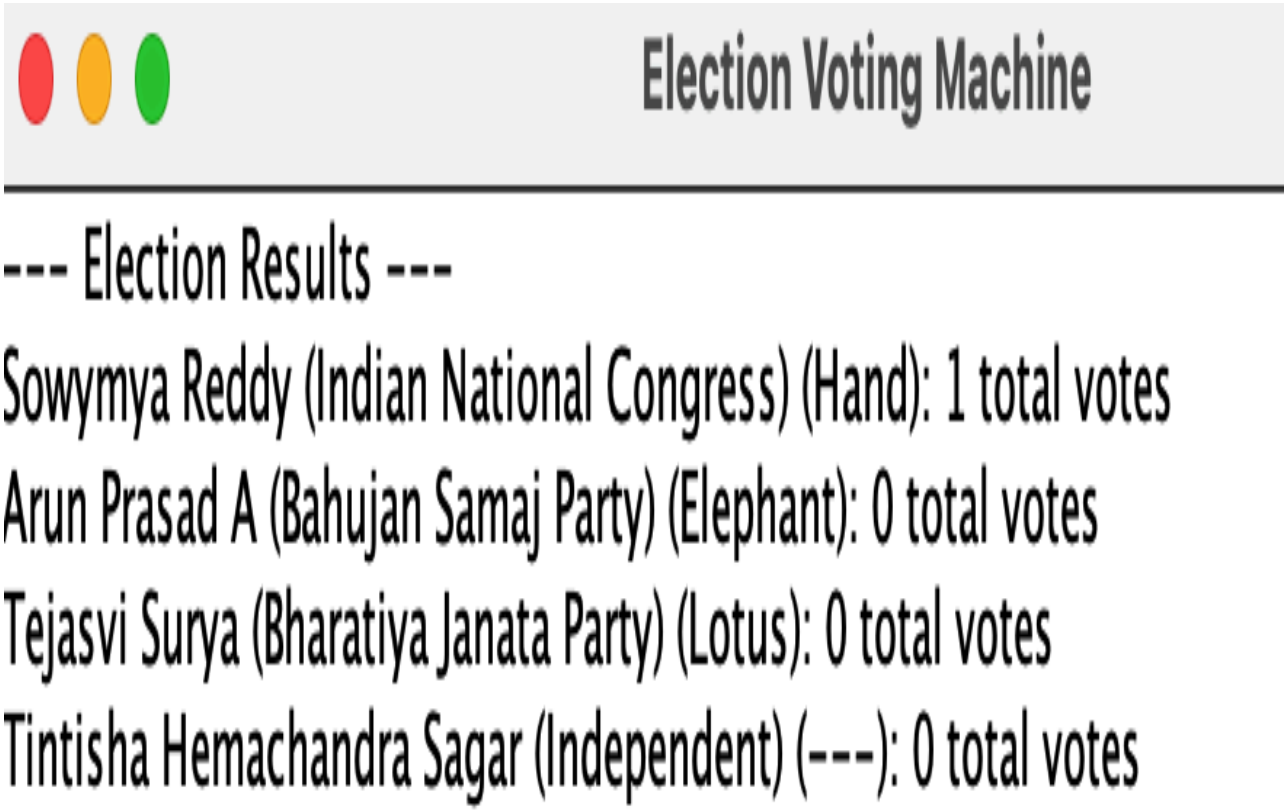
Page to Confirm Vote


Election Voting Machine
Register Voter
Name:
Age:
Aadhaar:
Address:
Register
Remove Voter
Name:
Aadhaar:
Reason:
Duplicate Entry

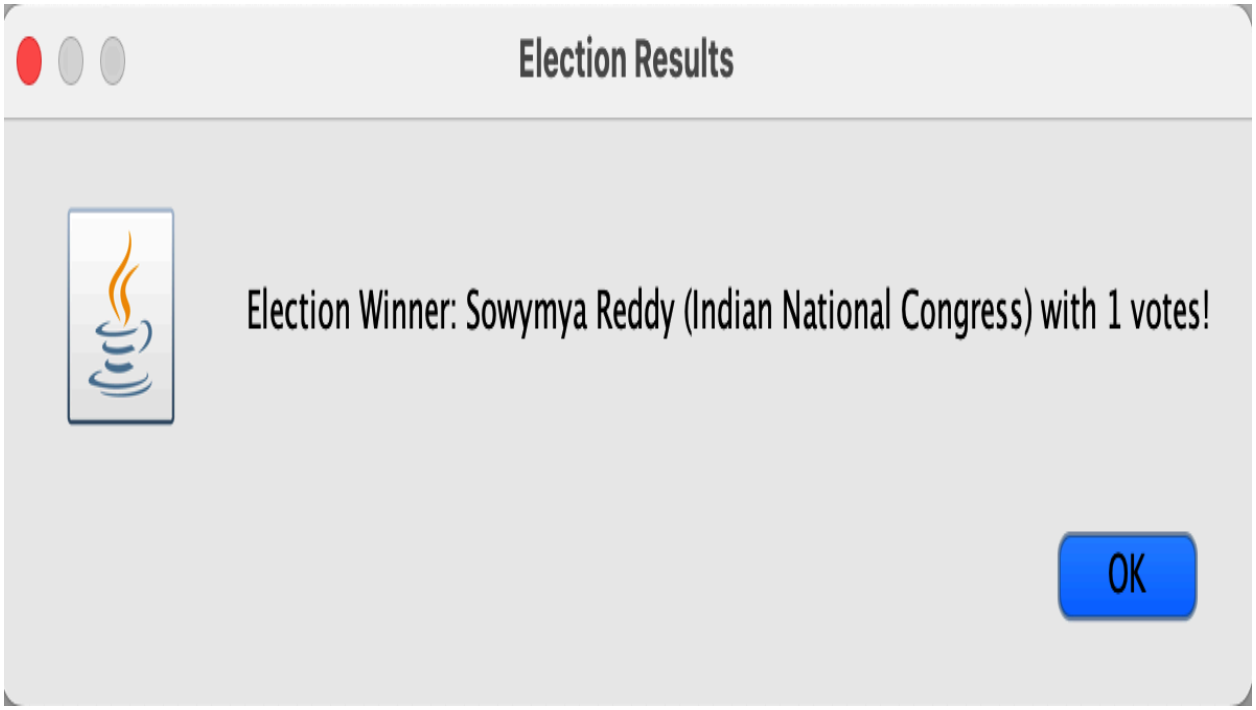
Page we get after choosing Election Commission at Role page



Page we get if ElectionCommission chooses to see All Voters

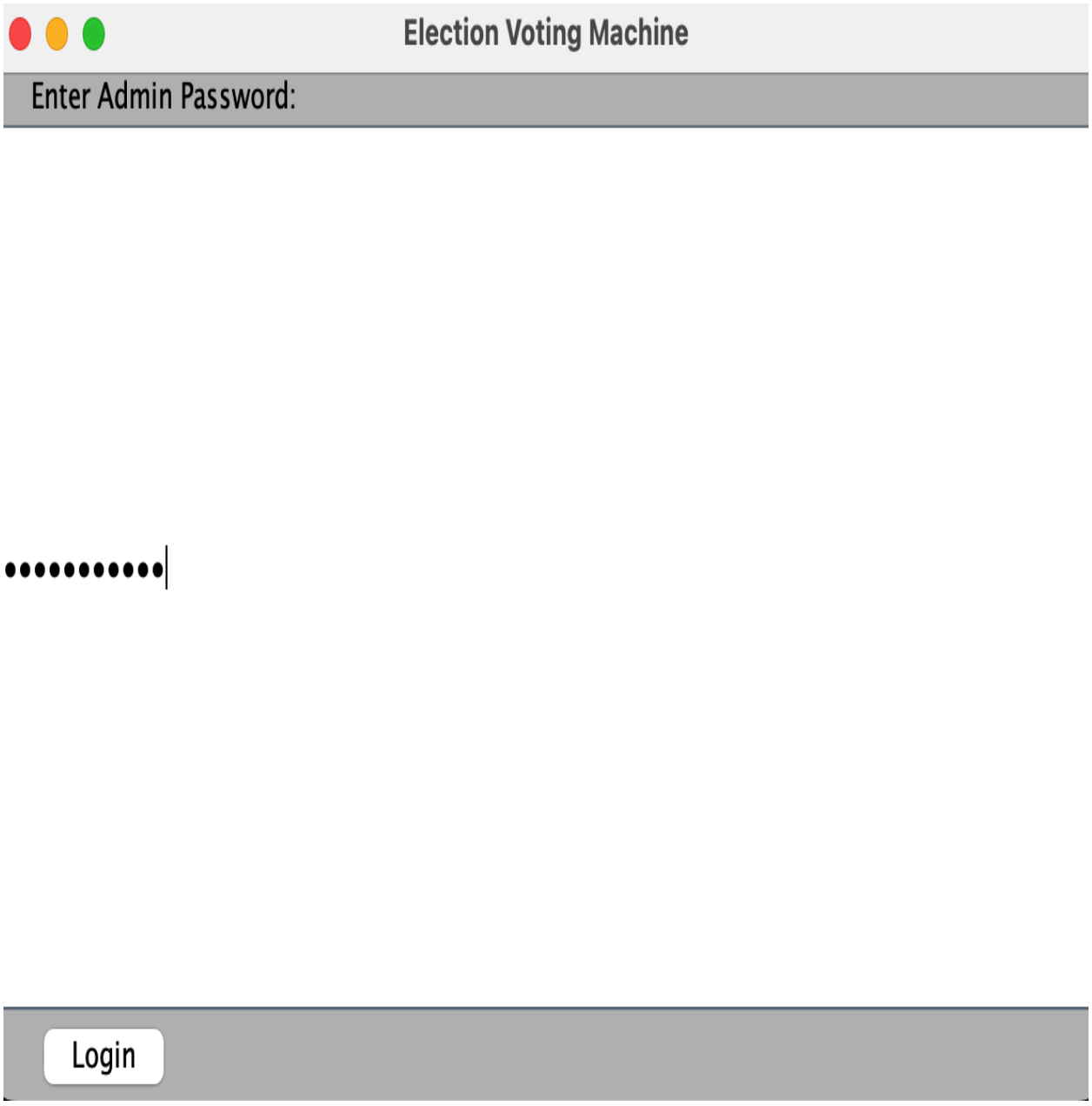


Page we get if we want to see election results

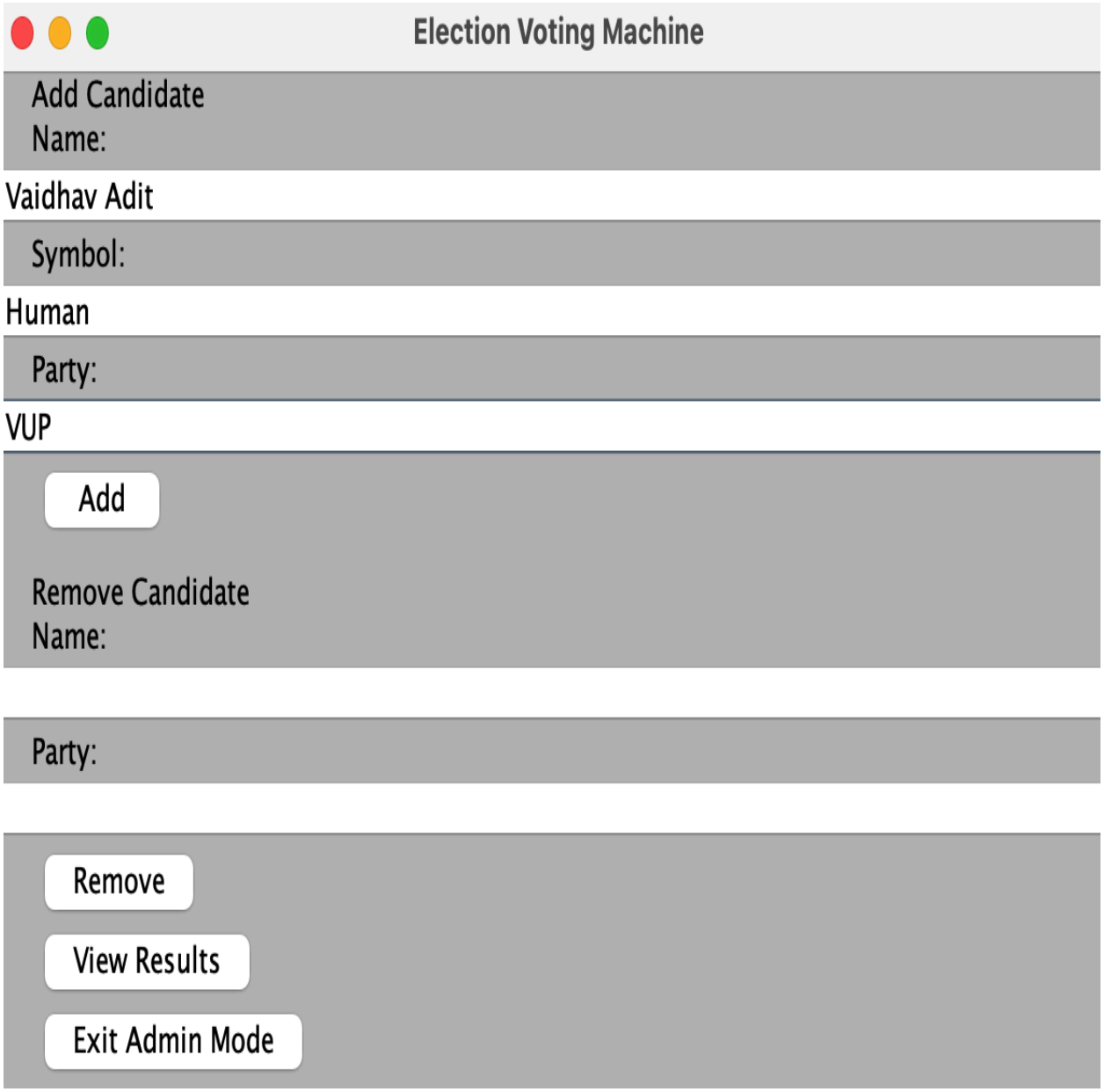


Page Declaring the Winner of the Election and ending the program

Admin Output



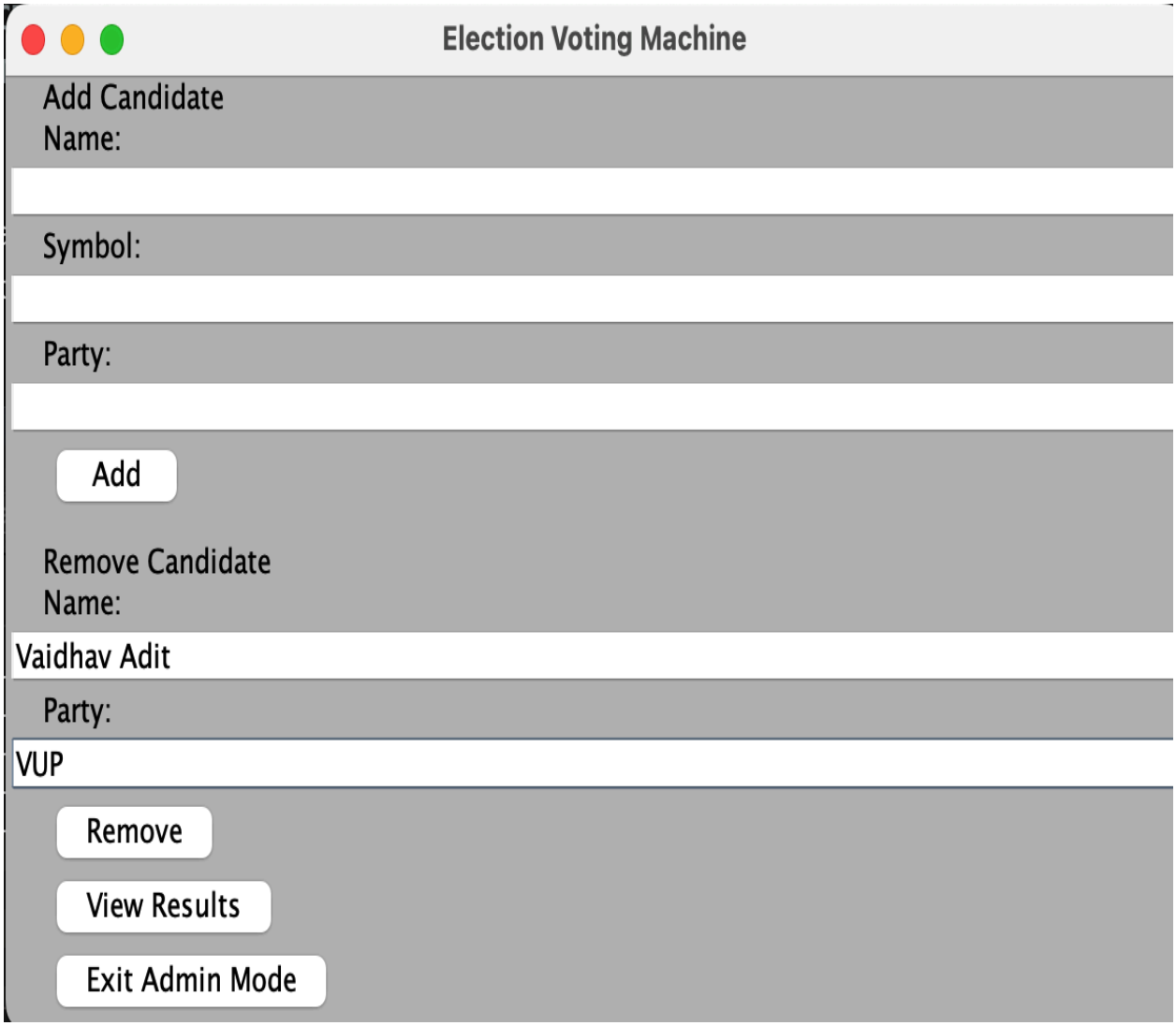
Page When Admin needs to give their password



Admin Adds a new Candidate

candidate_id	candidate_name	candidate_sym...	candidate_party	vote_count	constituency...
1	Sowymya Reddy	Hand	Indian National Congress	0	1
2	Arun Prasad A	Elephant	Bahujan Samaj Party	0	1
3	Tejasvi Surya	Lotus	Bharatiya Janata Party	0	1
4	Tintisha Hemachandra Sagar	---	Independent	0	1
5	Vaidhav Adit	Human	VUP	0	1
NULL	NULL	NULL	NULL	NULL	NULL

New Candidate Appears in the Database

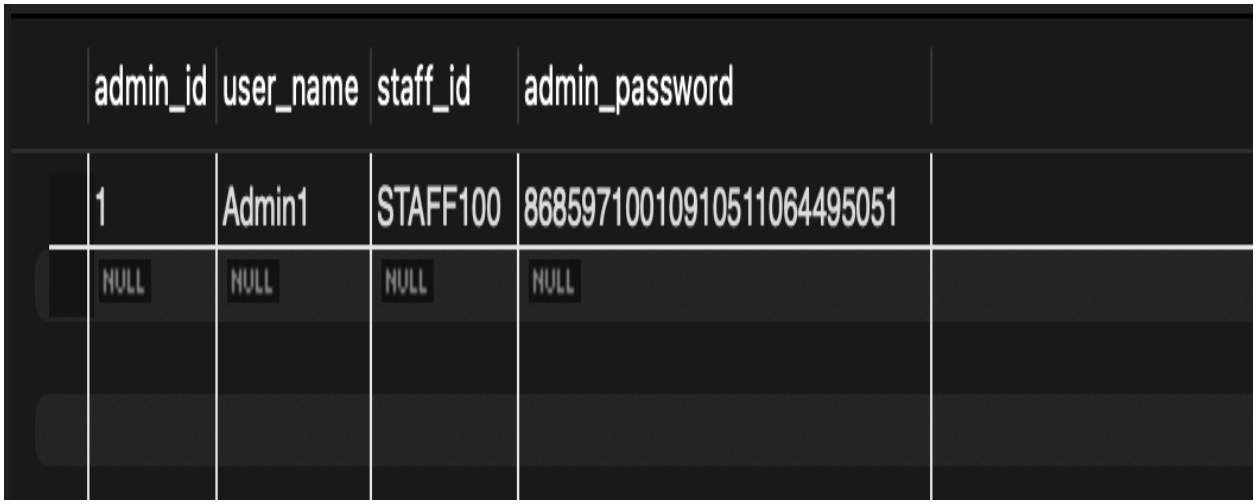


Admin Removes Candidate from Candidate List

candidate_id	candidate_name	candidate_sym...	candidate_party	vote_count	constituency...
1	Sowymya Reddy	Hand	Indian National Congress	0	1
2	Arun Prasad A	Elephant	Bahujan Samaj Party	0	1
3	Tejasvi Surya	Lotus	Bharatiya Janata Party	0	1
4	Tintisha Hemachandra Sagar	---	Independent	0	1
NULL	NULL	NULL	NULL	NULL	NULL

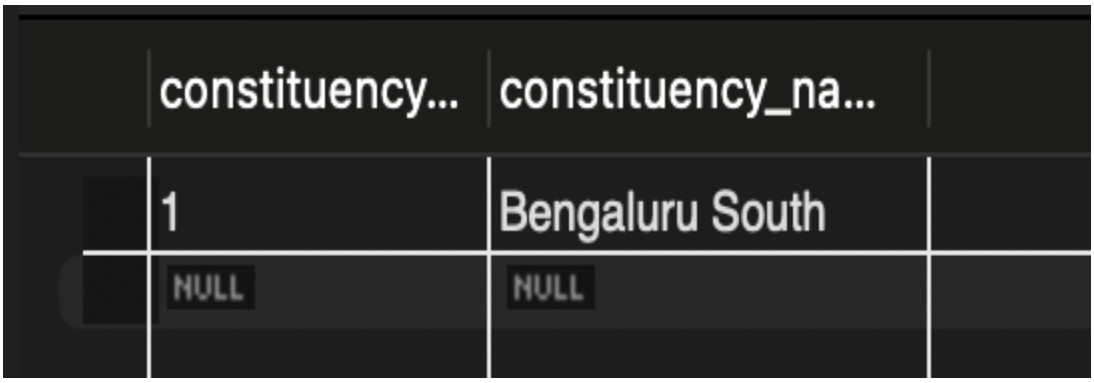
Candidate Has been removed from Database

Admins views the current results of the election



admin_id	user_name	staff_id	admin_password
1	Admin1	STAFF100	868597100109105110644950 51
NULL	NULL	NULL	NULL

Admin Login had been added to the database and their password has been safely encrypted



constituency...	constituency_na...
1	Bengaluru South
NULL	NULL

When admin set the constituency it was registered in the database

CSV Output:

Output for the above screenshots:

AnonymizedID	CandidateID	CandidateName	CandidateParty	Timestamp
VoterX	Vaidhav	Vaidhav	VU Samaj Party	2025-04-09T23:17:15.628007200

Separate Output:

AnonymizedID	CandidateID	CandidateName	CandidateParty	Timestamp
VoterX	Tejasvi Surya	Tejasvi Surya	Bharatiya Janata Party	2025-04-09T23:12:19.123821900
VoterX	Sowymya Reddy	Sowymya Reddy	Indian National Congress	2025-04-09T23:12:30.258957600
VoterX	Arun Prasad A	Arun Prasad A	Bahujan Samaj Party	2025-04-09T23:12:51.778910