



**UE22CS352B - Object Oriented Analysis & Design**

**Mini Project Report**

**ELECTRONIC VOTING SYSTEM**

*Submitted by:*

*Name : SRN (Team Members)*

|                             |                      |
|-----------------------------|----------------------|
| <b>SOUMYA RANJAN MISHRA</b> | <b>PES2UG22CS571</b> |
| <b>SUHAS VENKATA</b>        | <b>PES2UG22CS590</b> |
| <b>SOHAM SALAUNKE</b>       | <b>PES2UG22CS565</b> |
| <b>SHUKLAV REDDY</b>        | <b>PES2UG22CS557</b> |

Semester Section

**Faculty Name**

**Prof . RUBY DINAKAR**

**January - May 2025**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
FACULTY OF ENGINEERING  
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)  
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

# Problem Statement:

Traditional voting processes, particularly for organizational, community, or institutional elections, often rely on manual methods (paper ballots, manual registration, hand counting) which present several significant challenges. These methods can be resource-intensive, requiring considerable time and personnel for setup, execution, and vote tabulation. Manual counting is inherently prone to human error, potentially affecting the accuracy and legitimacy of the results. Furthermore, ensuring voter verification and preventing fraudulent activities like multiple voting can be complex and logistically demanding in a purely manual system. Accessibility can also be a concern, potentially limiting participation for individuals unable to be physically present.

The administrative overhead involved in managing candidates, voter lists, and election logistics is substantial. There is a clear need for a more efficient, secure, and reliable alternative, especially in contexts where deploying large-scale, government-grade electronic voting infrastructure is not feasible or necessary.

This project addresses these challenges by designing and developing a desktop-based Electronic Voting System. The primary goal is to create a robust, user-friendly application using Java, JavaFX, and MySQL that streamlines the entire election process. The system aims to improve efficiency through digital registration and automated vote tallying.

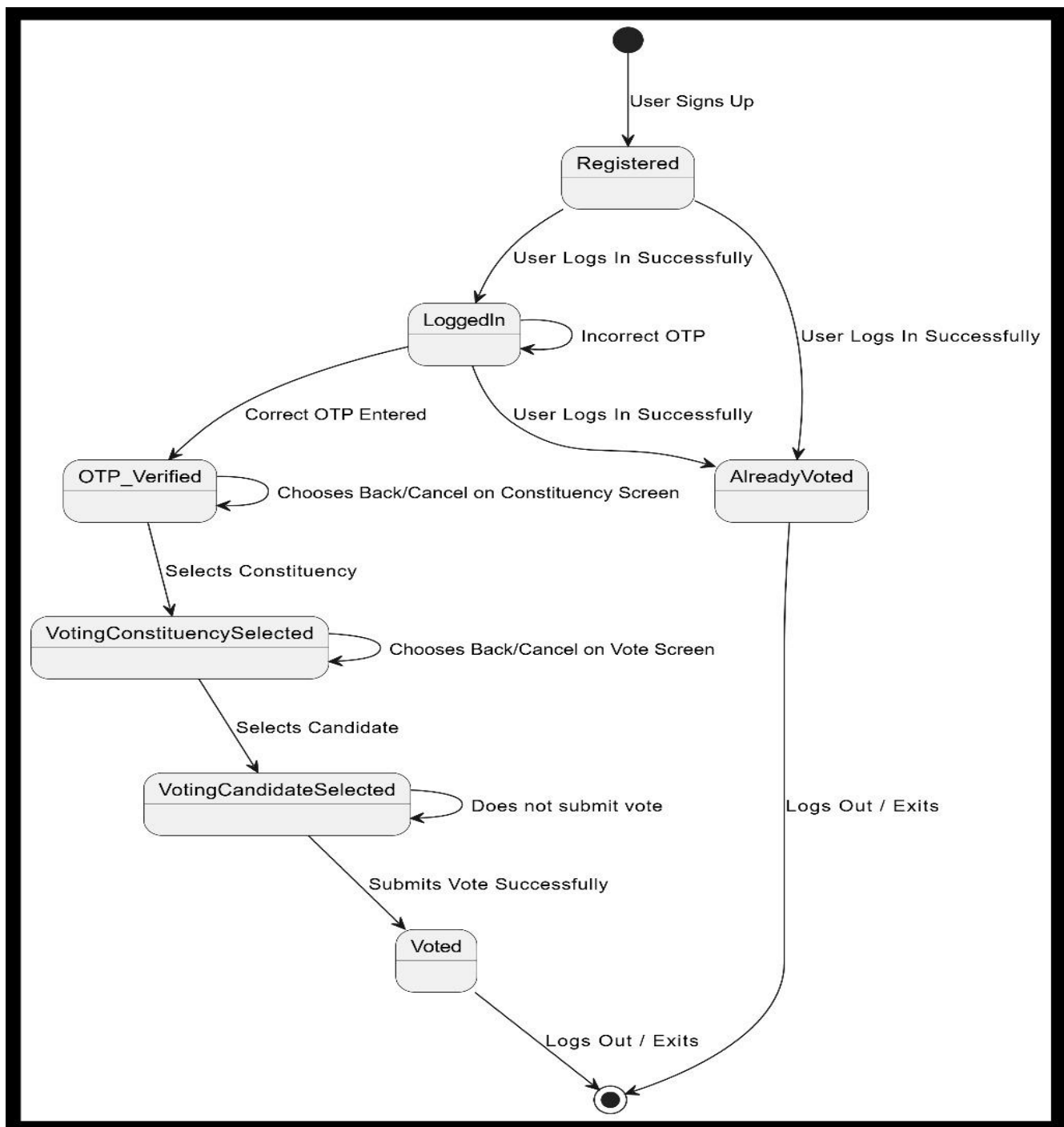
Security is enhanced via user authentication (username/password) and an additional OTP verification step for voters, along with role-based access control distinguishing administrators from regular voters. By automating the counting process, the system minimizes the risk of tabulation errors. The use of a graphical user interface built with JavaFX and support for multiple languages enhances usability and accessibility. The implementation strictly follows the Model-View-Controller (MVC) architectural pattern to ensure a well-structured, maintainable, and scalable solution suitable for managing smaller-scale elections effectively and reliably.

## Key Features:

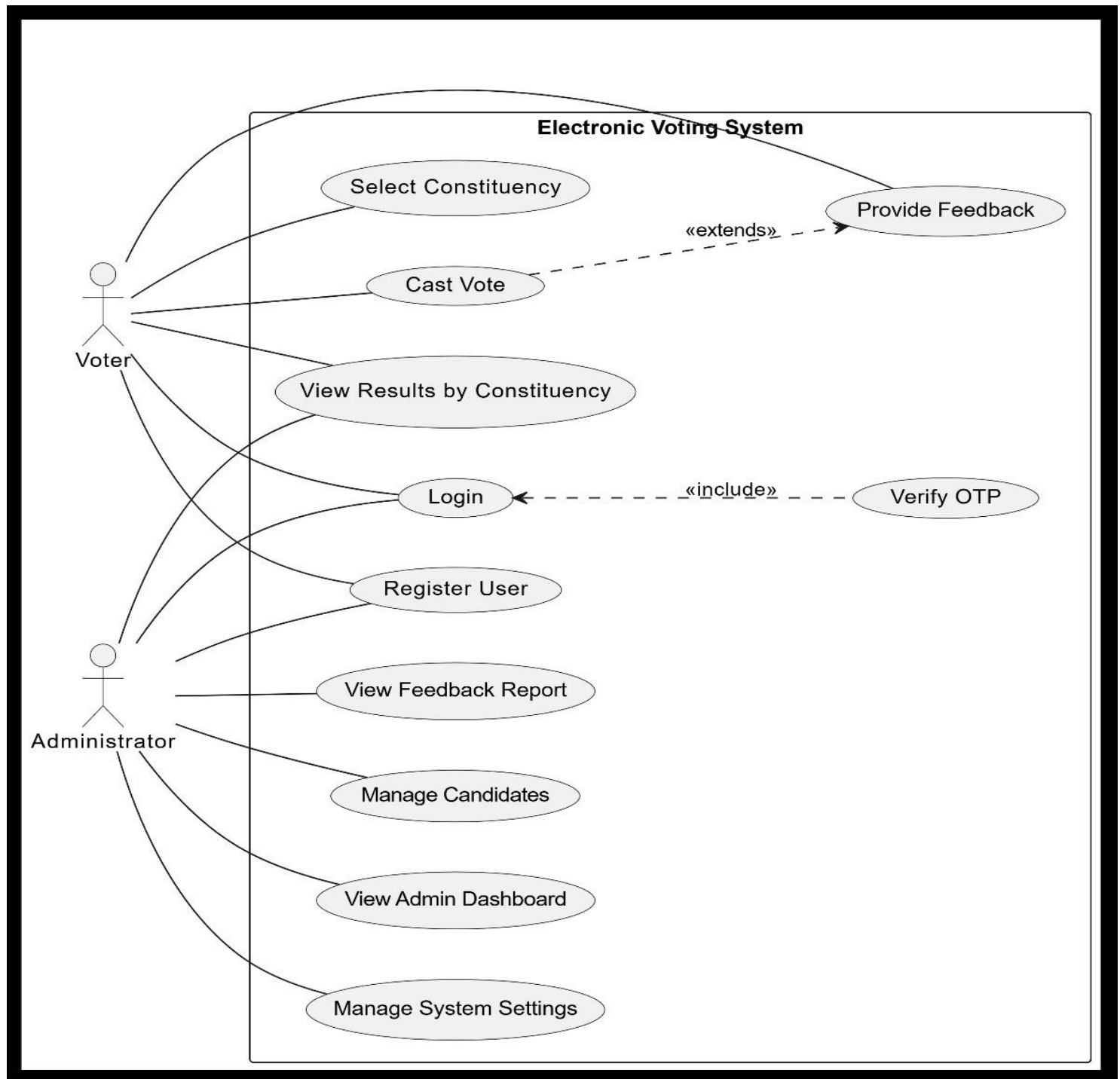
- **User Management:** Supports registration and login for both voters and administrators. Administrators have distinct privileges.
- **Secure Authentication:** Implements username/password authentication. Includes an OTP (One-Time Password) verification step for voters before they can proceed to vote (Note: OTP implementation uses console output and a temporary map).
- **Constituency-Based Voting:** Organizes elections based on constituencies. Voters select their constituency before viewing candidates and casting their vote.
- **Candidate Management:** Allows registration of candidates under specific constituencies. The admin panel provides features to add, view, and delete candidates.
- **Voting Process:** Enables authenticated and verified voters to cast a single vote for a candidate within their selected constituency, including a "None of the Above" (NOTA) option. Prevents users from voting more than once.
- **Result Tabulation:** Calculates and displays election results, viewable per constituency.
- **Feedback Mechanism:** Allows voters to provide feedback on their experience after voting. Feedback can be viewed through the main menu or potentially by administrators.

- **Administrator Panel:** A dedicated JavaFX interface (AdminView.java) for administrators providing:
  - ➔ Dashboard with statistics (total votes, candidates, users, constituencies), vote distribution charts, and voter turnout information.
  - ➔ Management of candidates (add/delete).
  - ➔ System management functions like resetting votes, user vote status, and candidates (requires admin password confirmation; some backend logic is placeholder).
- **Internationalization:** Supports multiple languages (English, Hindi, Kannada) using Java Resource Bundles, selectable at startup.

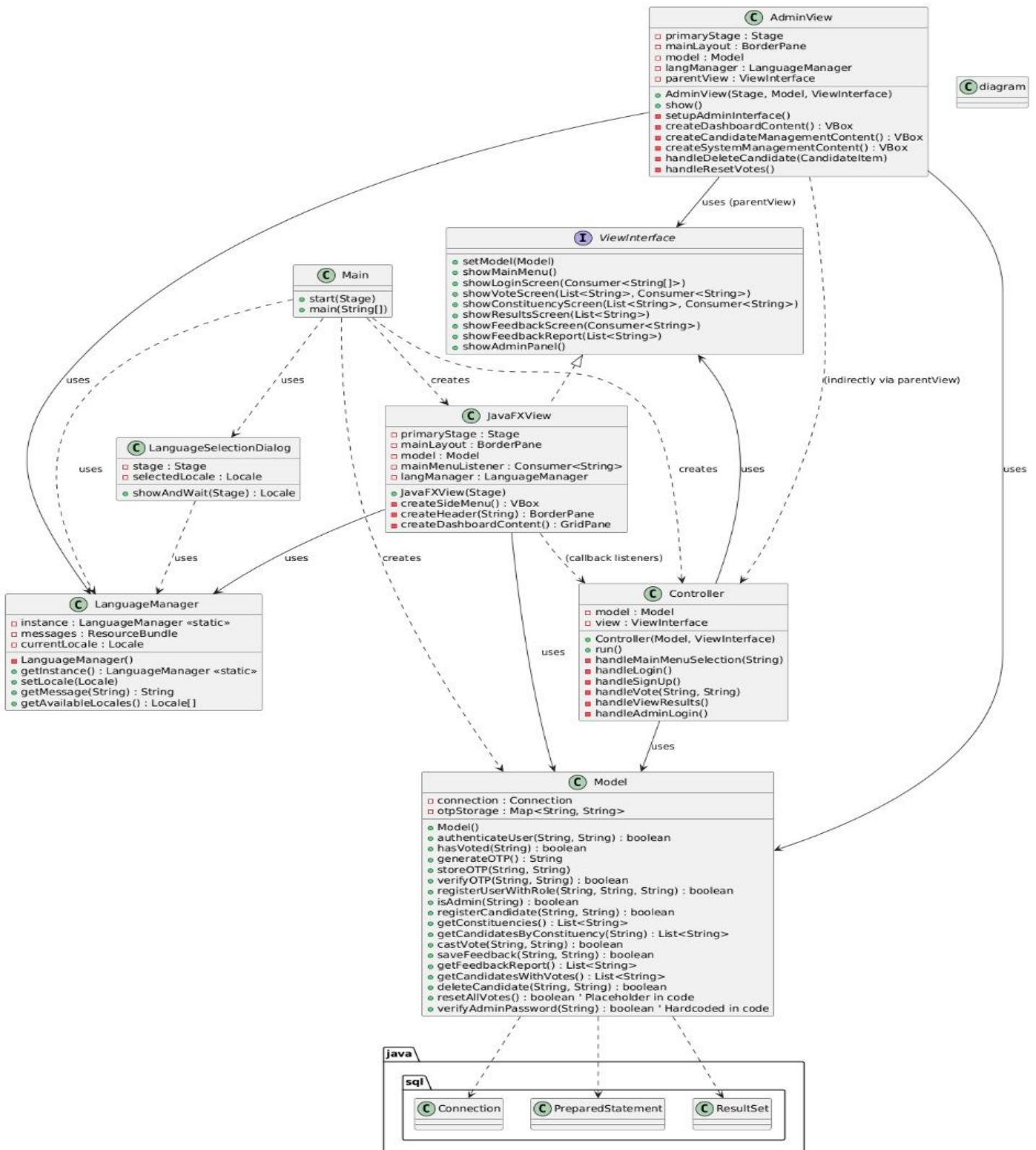
# STATE DIAGRAM:



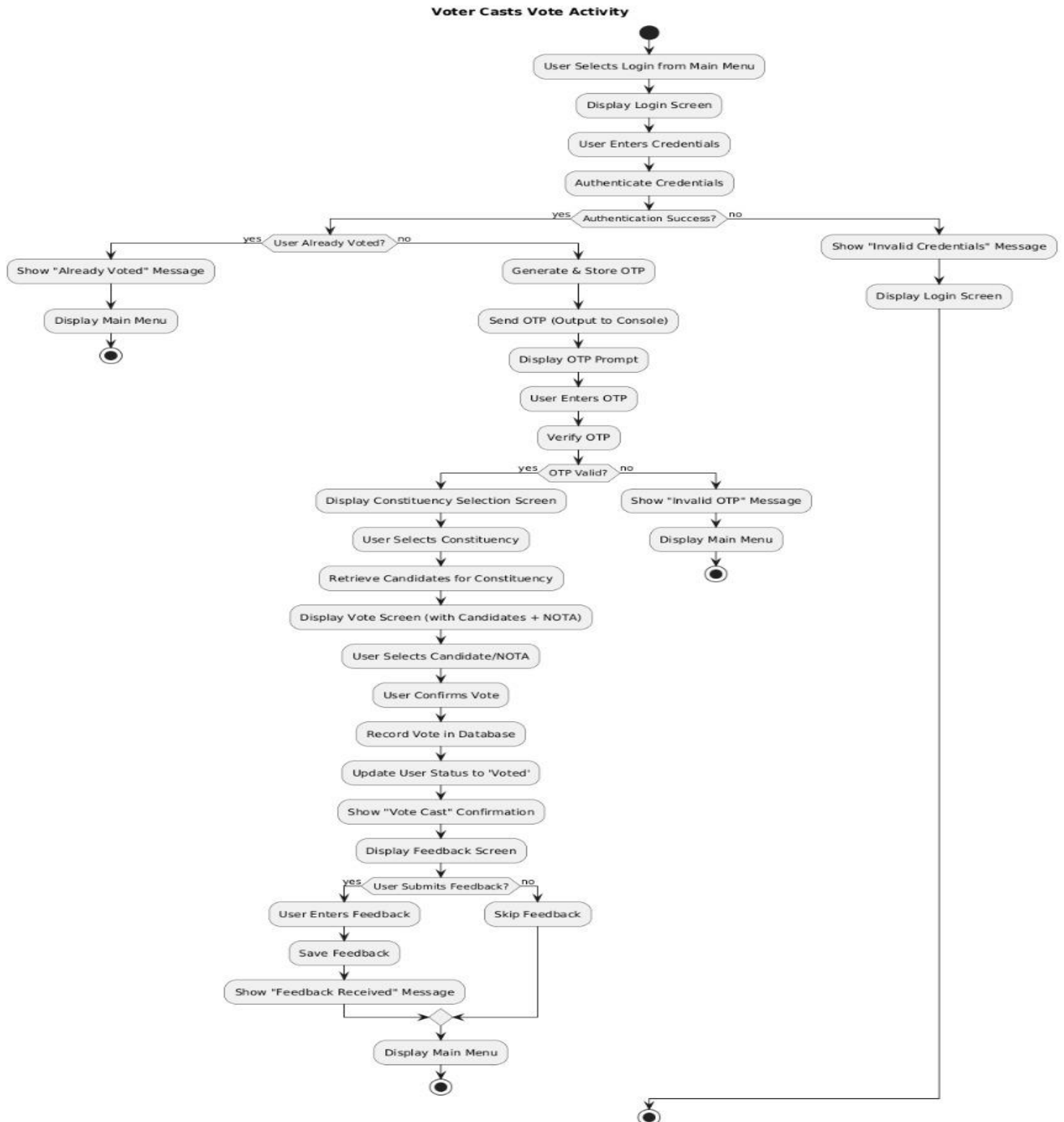
# Use Case Diagram:



## Class Diagram:



# Activity Diagrams:





# Major Use Cases:

- **Cast Vote:** This is arguably the most crucial function for a voter. It involves multiple steps: selecting a constituency, viewing candidates, making a selection (including NOTA), submitting the vote, and updating the user's status.
- **Login & Verify OTP:** While seemingly simple, this is a critical gateway for voters and involves authentication followed by a mandatory OTP step before accessing core voting functions. Its multi-step nature and security importance make it a major process.
- **Register User:** This allows new users (voters or admins) to join the system, a fundamental requirement.
- **Manage Candidates (Admin):** A core responsibility for the administrator, involving viewing, adding, and deleting candidates, which directly impacts the election setup.
- **Manage System Settings (Admin):** Includes critical, high-impact actions like resetting votes or deleting all candidates, often involving password confirmation.

## Minor Use Cases:

- **Provide Feedback:** This typically involves a simple flow: the user types feedback into a text area and submits it. It often follows the Cast Vote use case but is secondary to the voting action itself.
- **View Results by Constituency:** While important, the interaction is relatively simple: select the option, choose a constituency, and view the displayed results.
- **View Feedback Report:** Similar to viewing results, involves selecting the option and viewing the list.
- **Select Language:** A simple interaction at the start of the application handled by LanguageSelectionDialog.
- **Delete Single Candidate (Admin):** This is part of "Manage Candidates" but could be modeled as a simpler, minor flow: select candidate from list, confirm deletion.

# ARCHITECTURE PATTERNS:

The project explicitly uses the MVC pattern to separate concerns:

## ➤ **Model (Model.java)**

- ➔ **Core Function:** As you stated, the Model.java file manages the application's data, logic, and rules. It's the 'brain' behind the operations.
- ➔ **Data Persistence:** It directly interacts with your MySQL database (evoting schema) using JDBC (java.sql.\* classes like Connection, PreparedStatement, ResultSet). This includes:
  - Loading data (e.g., fetching users for authentication, retrieving candidates by constituency, getting feedback reports).
  - Storing data (e.g., registering new users and candidates, saving feedback, updating vote counts and user has\_voted status).
  - It even includes logic to ensure the database schema is correct (e.g., ensureRoleColumnExists adding the 'role' column if missing).
- ➔ **Business Logic & Rules:** It contains the core application logic beyond simple data retrieval:
  - User authentication (authenticateUser) and role checking (isAdmin).
  - Vote casting rules (castVote), including checking the has\_voted flag and the (currently hardcoded) 20-vote limit.
  - OTP generation (generateOTP) and verification (verifyOTP) logic (using a non-persistent otpStorage map for temporary storage).
  - Candidate registration (registerCandidate) and deletion (deleteCandidate).
  - Admin-specific logic like password verification (verifyAdminPassword - currently hardcoded) and system reset functions (some of which are placeholders, e.g., resetAllVotes, deleteAllCandidates).

- **State Encapsulation:** It holds the application's data state, both transient (like the otpStorage map) and persistent (managed via the database connection). It represents entities like Users, Candidates, Constituencies, and Votes.

### ➤ **View (JavaFXView.java, AdminView.java)**

- ➔ **Core Function:** These classes are responsible for rendering the user interface using JavaFX and presenting data to the user. They define *what* the user sees.
- ➔ **UI Rendering:** They use JavaFX components (Stage, Scene, BorderPane, GridPane, Button, Label, TextField, TableView, PieChart, RadioButton etc. ) to build the different screens the user interacts with (Login, Main Menu, Voting, Results, Admin Panel etc. ).
- ➔ **Data Display:** They receive data (e.g., lists of candidates/constituencies, results data, feedback reports, dashboard statistics ) usually passed from the Controller (which gets it from the Model) and display it in appropriate controls like TableView or PieChart. Note: For dashboard/admin stats, the Views (JavaFXView, AdminView) also directly hold a reference to the Model to fetch some display data, which is a common pattern but slightly blends responsibilities.
- ➔ **User Input Capture:** They capture user actions like button clicks (setOnAction), text entry, and selections in ComboBox or RadioButton/ToggleGroup.
- ➔ **Abstraction (ViewInterface):** JavaFXView implements the ViewInterface. This allows the Controller to interact with the main view through this interface, reducing direct dependency on the concrete JavaFXView class.
- ➔ **Multiple Views:** Your project uses distinct View classes for different major parts of the application: JavaFXView for the general user and AdminView for administrator tasks, providing tailored interfaces. It also uses LanguageSelectionDialog for the initial language choice.
- ➔ **Localization:** The views use the LanguageManager to display text labels and messages in the currently selected language (English, Hindi, or Kannada).

## ➤ **Controller (Controller.java)**

- ➔ **Core Function:** This class acts as the intermediary, orchestrating interactions between the Model and the View. It handles the application's flow and logic based on user input.
- ➔ **Input Handling:** It receives notifications of user actions from the View. This is primarily done through listeners or callbacks set on the View components (e.g., the `mainMenuListener` set via `view.setMainMenuListener` or the Consumers passed into methods like `view.showLoginScreen`, `view.showVoteScreen` ).
- ➔ **Processing User Actions:** When an action occurs (like a button click handled by a lambda expression passed to the View), the Controller interprets the request (e.g., inside methods like `handleLogin`, `handleVote`, `handleCandidateRegistration`, `handleAdminLogin`).
- ➔ **Model Interaction:** Based on the user's request, the Controller calls the appropriate methods on the Model instance to:
  - Display different screens (e.g., `view.showMainMenu`, `view.showConstituencyScreen`, `view.showVoteScreen`, `view.showAdminPanel`).
  - Show messages or results (e.g., `view.showMessage`, `view.showResultsScreen`, `view.showFeedbackReport`).
  - Request further input (e.g., `view.showOTPPrompt`).
- ➔ **Application Flow Control:** It manages the sequence of operations, guiding the user through the application's workflow, from login/signup through voting/administration to viewing results or exiting.

# DESIGN PRINCIPLES

## 1. Separation of Concerns (SoC):

- This is perhaps the most prominent principle, primarily achieved through the Model-View-Controller (MVC) architecture.
- Model (Model.java): Concerns itself solely with data (users, candidates, votes), business logic (authentication, voting rules), and persistence (database interaction).
- View (JavaFXView.java, AdminView.java ): Concerns itself with presenting data to the user and capturing input via the JavaFX graphical interface.
- Controller (Controller.java): Concerns itself with handling user input from the View, orchestrating actions by interacting with the Model, and selecting the appropriate View to display next. This separation makes the application easier to develop, test, maintain, and modify.

## 2. Single Responsibility Principle (SRP):

- Classes generally aim to have a single primary responsibility.
- LanguageManager is responsible only for handling internationalization and language resources.
- Controller is responsible for application flow control and mediating between Model and View.
- View classes (JavaFXView, AdminView ) are responsible for specific UI areas (main application vs. admin panel).
- Model handles all data-related logic. While it covers many responsibilities (users, candidates, votes, DB access), within the context of this project size, it acts as the single source for data concerns. In a larger system, this might be further broken down (e.g., using DAOs).

### **3. Encapsulation:**

- Data fields within classes are generally kept private (e.g., connection, otpStorage in Model; UI components and listeners in JavaFXView and AdminView ).
- Access to data and internal state is controlled through public methods (e.g., model.authenticateUser(...), view.showMainMenu() ). This hides internal implementation details.

### **4. Abstraction:**

- The ViewInterface provides an abstraction for the main user interface.
- The Controller interacts with the View via this interface (ViewInterface view field), decoupling it from the concrete JavaFXView implementation. This allows for potential future changes in the View technology without necessarily breaking the Controller, as long as the new view implements the interface.

### **5. Dependency Inversion Principle (DIP) (Partial Application):**

- The principle states that high-level modules should not depend on low-level modules; both should depend on abstractions.
- This is applied partially where the Controller (high-level module for flow) depends on the ViewInterface (abstraction) rather than the concrete JavaFXView (low-level detail).
- However, the Controller directly depends on the concrete Model class. A stricter application might involve a ModelInterface that the Model class implements, and the Controller would depend on that interface instead.

## 6. Modularity

- The use of distinct classes for Model, View, Controller, and specific features like Language Manager promotes modularity.
- If Java Modules were used as initially mentioned (though `module-info.java` isn't present for review), this would further enforce modularity at a higher level by defining explicit dependencies and exports between different parts of the application.



# DESIGN PATTERNS

## 1. SINGLET

## 2. ON PATTERN

- Description: Ensures that a class has only one instance and provides a global point of access to it.
- Implementation: The Language Manager class implements the Singleton pattern. It has a private constructor, a private static instance variable (instance), and a public static getInstance() method that creates the instance if it doesn't exist and returns the single instance.
- Benefit: Guarantees that there is only one manager for language resources and locale settings throughout the application, ensuring consistency.

## 3. DATA ACCESS OBJECT(DAO) PATTERN (CONCEPTUAL APPLICATION)

- Description: Abstracts and encapsulates all access to the data source (database). The DAO manages the connection with the data source to obtain and store data.
- Implementation: While you don't have separate UserDAO, CandidateDAO classes, the Model.java class effectively acts as a DAO Layer or Facade for data persistence. It centralizes all the JDBC code and SQL queries needed to interact with the users and candidates tables, hiding these database interaction details from the Controller. Methods like authenticateUser, registerCandidate, castVote, saveFeedback, getAllCandidates serve as the data access interface.
- Benefit: Decouples the business logic (in Controller and potentially Model itself) from the specific data persistence mechanism. Makes it easier to change the database or persistence strategy later without affecting other parts of the application significantly.

#### **4. OBSERVER PATTERN (IMPLICIT VIA JAVA EFFECTS AND EVENTS-LISTENERS)**

- Description: Defines a one-to-many dependency between objects so that when one object (the subject) changes state, all its dependents (observers) are notified and updated automatically.
- Implementation: While not explicitly implemented using Java's Observer/Observable interfaces in your core MVC structure, the event handling mechanism in JavaFX and the way your View interacts with the Controller heavily relies on this pattern's principles.
  - > UI controls in the View (JavaFXView, AdminView ) act as subjects.
  - > Action handlers (often implemented as lambda expressions in the Controller and passed to the View via methods like showLoginScreen, showVoteScreen, or setMainMenuListener ) act as observers.
  - > When a user interacts with a control (e.g., clicks a button), the control fires an event, notifying the registered listener (the observer code in the Controller) to take action.
- Benefit: Decouples the View components (which generate events) from the Controller logic (which handles events), allowing for flexible UI design and interaction logic.

# Screenshots

## 1. LANGUAGE SELECTION

Select Language / भाषा चुनें / ಭಾಷೆಯನ್ನು ಆಯ್ಕೆಮಾಡಿ

Select Language / भाषा चुनें / ಭಾಷೆಯ...

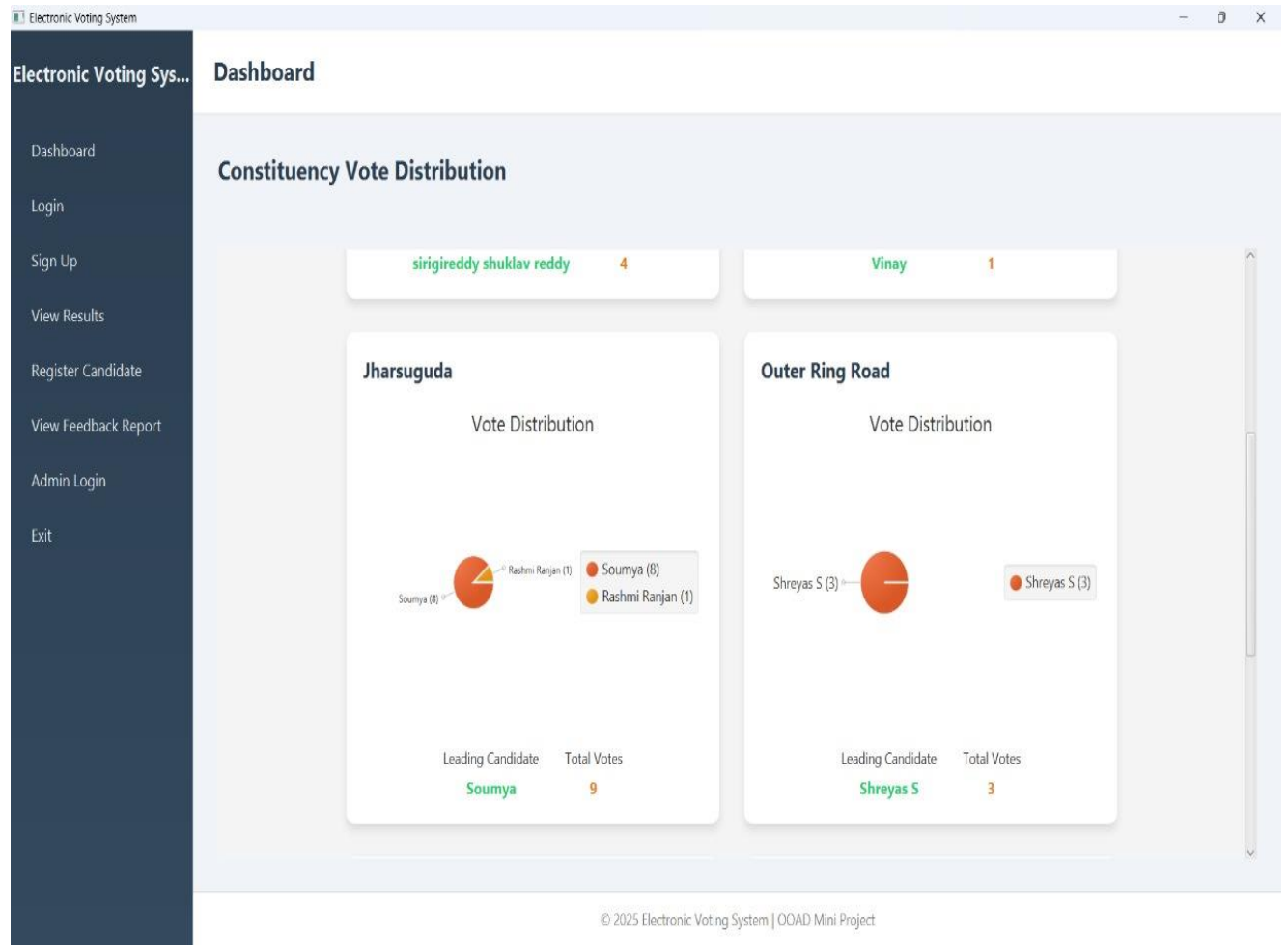
☒ English

☐ Hindi (हिन्दी)

☐ Kannada (ಕನ್ನಡ)

Continue / जारी रखें / ...

## 2. DASHBOARD



### 3. USER REGISTRATION

Electronic Voting System

User Registration

Choose a Username

PESIT

Choose a Password

••••

Select Role

voter

Register

Back to Main Menu

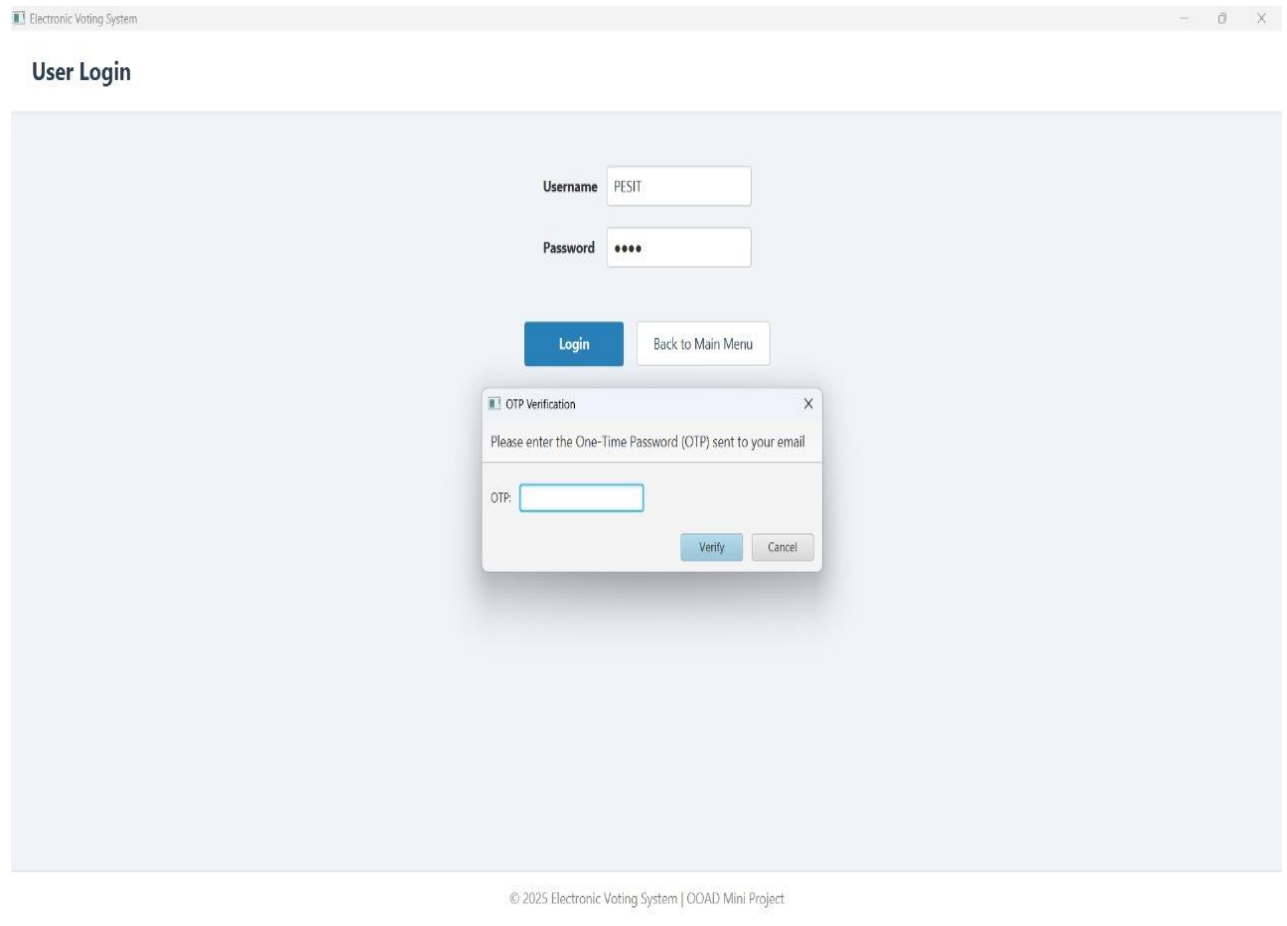
E-Voting System

Sign-up successfull! You can now log in.

OK

© 2025 Electronic Voting System | OOAD Mini Project

## 4. USER LOGIN



```
PS C:\Users\NITRO\OneDrive\Desktop\OOAD Mini Project> .\run.bat
Starting E-Voting Application...
Setting up classpath...
Compiling application...
Note: src\com\example\ voting\LanguageManager.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Copying resources...
CSS resources copied successfully.
Language resources copied successfully.
Running application...
Sending OTP to user: PESIT. Your OTP is: 469397
```

## 5. SELECT CONSTITUENCY

Electronic Voting System

Select Your Constituency

Please select your constituency:

☐

☐ Jharsuguda

☐ Kadapa

☐ Outer Ring Road

☐ Hospete

☐ Belgaum

Continue

Back to Main Menu

© 2025 Electronic Voting System | OOAD Mini Project

## 6. CAST YOUR VOTE

Electronic Voting System

— 0 X

### Cast Your Vote

Please select your preferred candidate:

☐ Soumya

☐ Rashmi Ranjan

☐ None of the Above (NOTA)

Submit Vote

Back to Main Menu

© 2025 Electronic Voting System | OOAD Mini Project



## 7.PROVIDE FEEDBACK

Electronic Voting System

— ◻ ×

Provide Feedback

Please share your voting experience:

Skip

Submit Feedback

## 8.FEEDBACK REPORT

Electronic Voting System

Feedback Report

Kyrie

hi

Vinay

thank you BAIS

Shukla

nice

Sohum Salunke

disqualify him

Shreyasa

bye

Charlie\_Puth

00000000

Sudeep

vv

Back to Main Menu

## 9.REGISTER NEW CANDIDATE


Electronic Voting System

### Register New Candidate

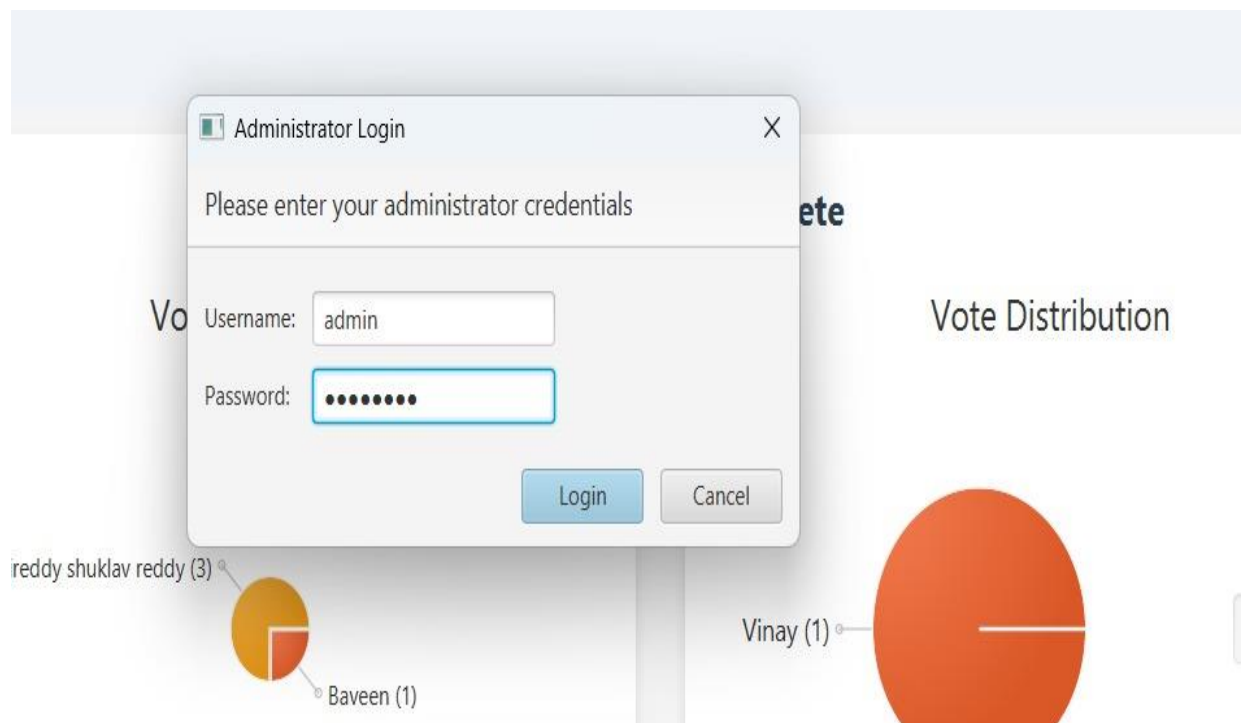
Candidate Name

Constituency

E-Voting System

 Candidate Shreyas N registered successfully!

## 10. ADMINISTRATOR LOGIN



# 11. ADMINISTRATOR PANEL

Administrator Panel

Exit Admin Panel

Dashboard

Candidates

System Management

Manage Candidates

| Candidate Name            | Constituency    | Votes | Actions |
|---------------------------|-----------------|-------|---------|
| Baveen                    |                 | 1     | Delete  |
| sirigireddy shuklav reddy |                 | 3     | Delete  |
| Soumya                    | Jharsuguda      | 8     | Delete  |
| Suhas Venkata             | Kadapa          | 1     | Delete  |
| Shreyas S                 | Outer Ring Road | 4     | Delete  |
| Vinay                     | Hospete         | 1     | Delete  |
| Rashmi Ranjan             | Jharsuguda      | 1     | Delete  |
| Shashank Naikar           | Belgaum         | 2     | Delete  |
| Shreyas N                 | Outer Ring Road | 0     | Delete  |

Add New Candidate

Add Candidate

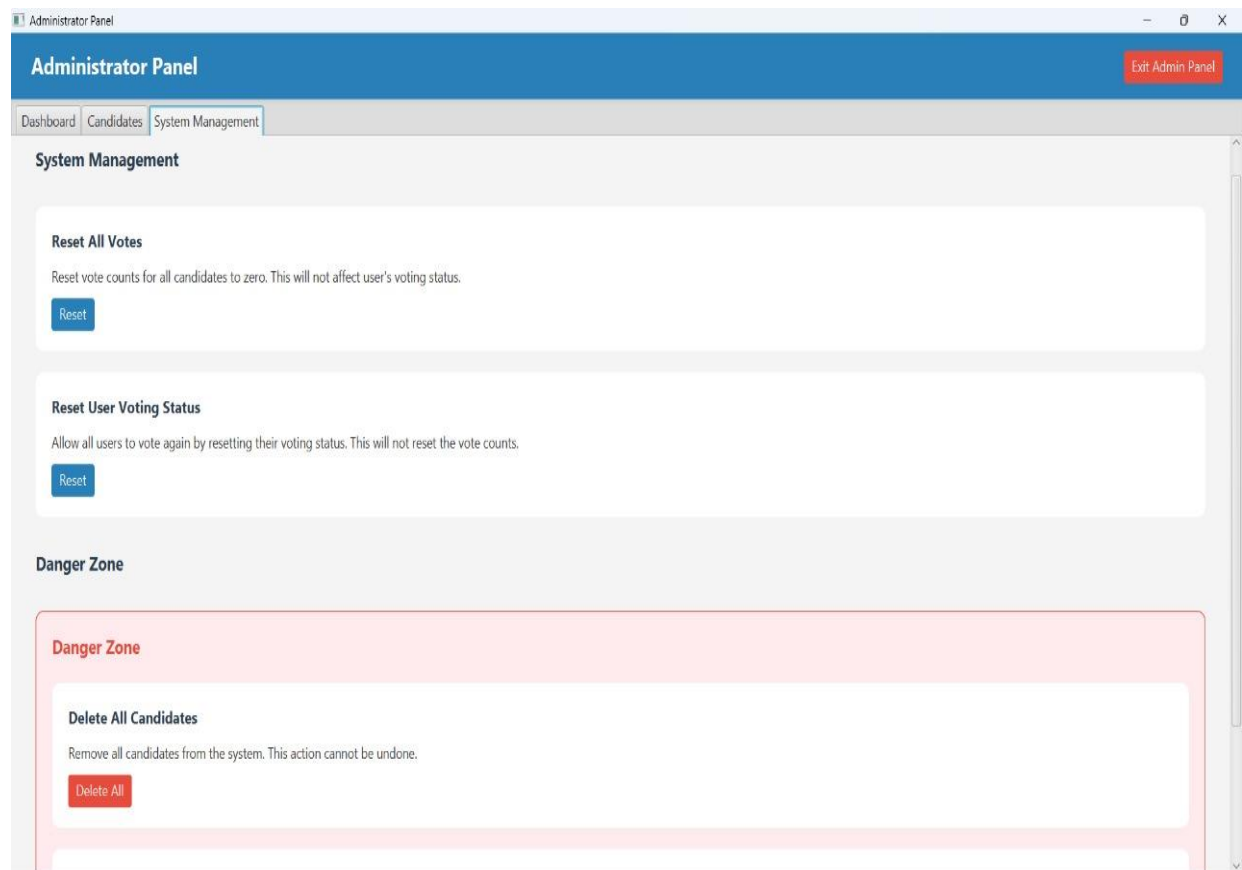
Candidate Name

Enter candidate name

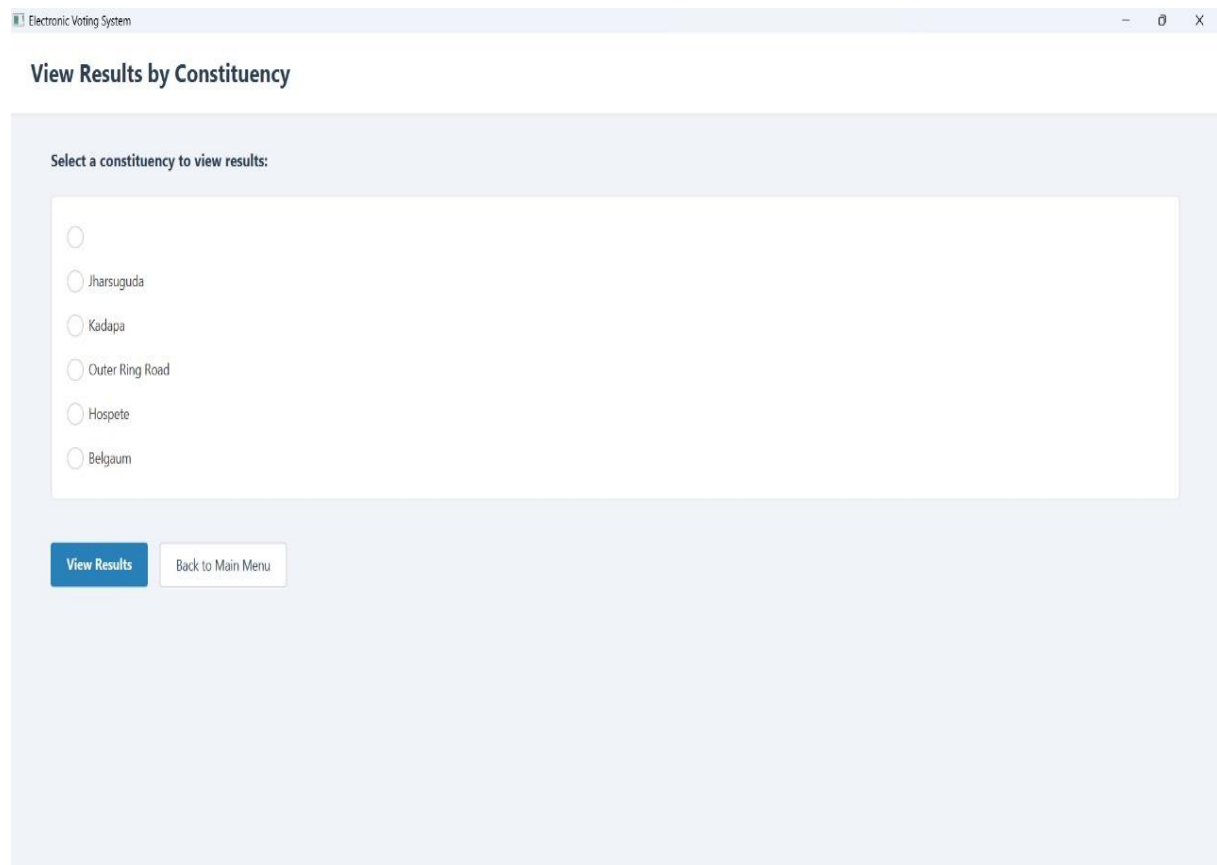
Constituency

Add

Administrator Panel - Restricted Access



Administrator Panel - Restricted Access



## Constituency Results

| Candidate     | Votes |
|---------------|-------|
| Soumya        | 8     |
| Rashmi Ranjan | 1     |
|               |       |
|               |       |
|               |       |
|               |       |
|               |       |
|               |       |
|               |       |
|               |       |
|               |       |
|               |       |

Back to Main Menu

Back to Constituencies

## Individual contributions of the team members

| Name                               | SRN                  | Modules Worked on  |
|------------------------------------|----------------------|--|
| <b>SOUMYA RANJAN MISHRA</b>        | <b>PES2UG22CS571</b> | <b>Backend Core (Model):</b><br>Implemented Model.java including database setup (JDBC), user authentication, OTP logic (backend), candidate data management, voting logic, and data persistence for users, candidates, and feedback.   |
| <b>SUHAS VENKATA KARAMALAPUTTI</b> | <b>PES2UG22CS590</b> | <b>Application Flow &amp; Core UI (Controller/View):</b> Implemented Controller.java handling main application flow, user actions (login, signup, vote initiation). Developed core user workflow screens in JavaFXView.java (Login, Signup, Constituency Selection, Voting, Feedback)      |
| <b>SOHAM PRAVIN SALUNKHE</b>       | <b>PES2UG22CS565</b> | <b>UI Display &amp; Supporting Features (View/Utils):</b> Implemented results display screens and main dashboard UI (including charts) in JavaFXView.java. Developed LanguageManager and LanguageSelectionDialog for internationalization support. Handled application setup in Main.java. |
| <b>SHUKLAV REDDY</b>               | <b>PES2UG22CS557</b> | <b>Admin Module (View/Integration):</b> Implemented the complete AdminView.java UI (Dashboard, Candidate Management, System Management tabs). Integrated admin functionalities with the Model (fetching stats, add/delete  |



|  |  |  |
|--|--|--|
|  |  | <b>candidates, system resets).<br/>Handled Admin login flow<br/>integration<br/>(Controller/JavaFXView).</b> |
|--|--|--|

## GITHUB LINK

[https://github.com/SoumyaMishra03/evoting\\_OOAD](https://github.com/SoumyaMishra03/evoting_OOAD)