Online Voting System

1.Introduction

The **Online Voting System** is a Java-based console application that allows registered users to cast votes electronically in a secure and controlled environment. The system uses JDBC to connect to a MySQL database and supports separate modules for users (voters) and a single administrator.

1.1 Purpose

- Provide a simple, digital alternative to manual voting processes.
- Ensure that each eligible user can vote only once per election.
- Allow an admin to manage elections, candidates, voters, and results from a central system.

1.2 Scope

- User registration and authentication.
- Admin authentication and management operations.
- Election and candidate management.
- Vote casting, result calculation, and history viewing.
- Command-line interface (CLI) application only (no GUI/web UI in this version).

2. System Overview

2.1 Architecture

The system follows a layered architecture:

- **Presentation Layer (CLI):**
    - Main class provides menus for user and admin operations.
- **Service Layer:**
    - UserService, AdminService handle business logic and user/admin flows.
- **Data Access Layer (DAO):**
    - UserDAO, AdminDAO, ElectionDAO, CandidateDAO, VoteDAO implement JDBC operations.
- **Database Layer:**
    - MySQL database voting_db stores all persistent data.

2.2 Technologies Used

- Java (console application)
- JDBC (Java Database Connectivity)
- MySQL (Relational database)
- MySQL Connector/J driver

3. Functional Requirements

3.1 User (Voter) Functions

- **Sign-up:**
    - Input: name, age, email, phone, password, address (state/district/city/area).
    - Validate: age must be ≥ 18; email must be unique.
- **Login:**
    - Authenticate with email and password.
- **Vote in Elections:**
    - View active elections matching user's address.
    - View candidates per election.
    - Cast vote once per election (enforced via DB constraint on (user_id, election_id)).
- **Update Profile:**
    - Update age and phone with age validation.
- **Manage Address:**
    - Update address string used for election scoping.
- **View History & Results:**
    - See list of votes cast with election name, candidate name, and time.
    - See results of past elections with candidate names and vote counts.

3.2 Admin Functions

- **Login:**
    - Single admin account stored in admin table.

- **Manage Elections:**
    - Add election (year, name, description, dates, status, scope address).
    - Start election (status → RUNNING).
    - Terminate election (status → ENDED).
    - List all elections.
- **Manage Candidates:**
    - Add candidate (name, party, election).
    - Update candidate data.
    - Delete candidate.
    - List all candidates.
- **Manage Voters:**
    - List all registered users.
    - Update user age/phone.
    - Delete user.
- **Manage Results:**
    - Mark election as RESULT_DECLARED.
    - View previous election results with aggregated vote counts.

4. Database Design

4.1 Tables

1. **users**
    - id (INT, PK, AUTO_INCREMENT)
    - name (VARCHAR)
    - age (INT)
    - email (VARCHAR, UNIQUE)
    - phone (VARCHAR)
    - password (VARCHAR)
    - address (VARCHAR) – format: state/district/city/area
2. **admin**
    - id (INT, PK)
    - name (VARCHAR)
    - email (VARCHAR, UNIQUE)
    - password (VARCHAR)
3. **elections**
    - id (INT, PK, AUTO_INCREMENT)
    - year_of_election (INT)
    - name (VARCHAR)
    - description (VARCHAR)
    - election_date (DATE)
    - scheduled_start (DATETIME)
    - scheduled_end (DATETIME)
    - status (VARCHAR) – SCHEDULED, RUNNING, ENDED, RESULT_DECLARED

- scope_address (VARCHAR) – election area or *

4. **candidates**
    - id (INT, PK, AUTO_INCREMENT)
    - name (VARCHAR)
    - party (VARCHAR)
    - election_id (INT, FK → elections.id)
5. **votes**
    - id (INT, PK, AUTO_INCREMENT)
    - user_id (INT, FK → users.id)
    - election_id (INT, FK → elections.id)
    - candidate_id (INT, FK → candidates.id)
    - vote_time (DATETIME)
    - UNIQUE (user_id, election_id) to ensure one vote per election per user.

4.2 Relationships

- One election → many candidates.
- One user → many votes.
- One candidate → many votes.

5. Implementation Details

5.1 Key Classes

- **DBConnection**
    - Provides getConnection() using JDBC URL jdbc:mysql://localhost:3306/voting_db.
    - Loads driver with Class.forName("com.mysql.cj.jdbc.Driver").
- **Model Classes** (com.voting.model)
    - User, Admin, Election, Candidate, Vote – plain POJOs.
- **DAO Classes** (com.voting.dao)
    - UserDAO: CRUD for users, login, update contact, update address.
    - AdminDAO: login for admin.
    - ElectionDAO: add/list/update elections, get active elections by address.
    - CandidateDAO: add/update/delete/list candidates.
    - VoteDAO: check if user has voted, cast vote, show voting history with names (JOIN), show election results.
- **Service Classes** (com.voting.service)
    - UserService: handles user sign-up, login, user menu, voting, profile updates, history & results.
    - AdminService: handles admin menu, manages elections, candidates, voters, and results.
- **Main** (com.voting.app.Main)
    - Entry point of the application.
    - Provides top-level menu: User login, User sign-up, Admin login, Exit.

5.2 Input & Validation

- Uses Scanner to read input from console.
- Age validation: loops until a valid integer ≥ 18 is entered.
- Basic error handling: try-catch for SQLException and NumberFormatException, prints user-friendly messages.

6. System Workflow

6.1 User Workflow

1. User selects **sign-up** from main menu.
2. Enters details; system validates age and unique email, inserts into users table.
3. User logs in with email and password.
4. User dashboard options:
    - Vote → choose election → choose candidate → cast vote.
    - Update age/phone → persist changes.
    - Manage address → update address string.
    - Voting history → see elections and candidates voted for, with timestamps.
    - View results → see all elections and candidate vote counts.
    - Logout → return to main menu.

6.2 Admin Workflow

1. Admin logs in using credentials in admin table.
2. Admin dashboard options:
    - Manage Elections → add/start/terminate/list elections.
    - Manage Candidates → add/update/delete/list candidates.
    - Manage Voters → list/update/delete users.
    - Manage Results → declare results and show previous results.
    - Logout → return to main menu.

7. Testing & Validation

- **Unit-level testing**:
    - Basic manual tests on DAO methods (insert, update, delete, select).
- **Integration testing**:
    - End-to-end tests from CLI: sign-up, login, manage elections, vote, show results.
- **Constraints verified**:
    - Age < 18 registration blocked.
    - Duplicate email prevented.
    - Duplicate vote per election prevented by unique constraint.

Limitations & Future Enhancements

8.1 Limitations

- No graphical or web interface in this version.
- Passwords are stored as plain text in the database.
- Address stored as a single string instead of normalized tables.
- No encryption of votes or advanced security.

8.2 Future Enhancements

- Implement web version using Servlet/JSP or Spring Boot.
- Add password hashing (e.g., BCrypt).
- Add more granular address tables (state, district, city, area) with foreign keys.
- Add role-based access with multiple admins.
- Implement better logging and error reporting.
- Add automated tests (JUnit).