Secure Online Voting System

Author: Denis Chipirliu
Group: 30431

January 7, 2025

Abstract

The Secure Online Voting System project aims to create a robust, secure, and user-friendly voting platform leveraging modern web technologies, cryptographic protocols, and formal verification. This report outlines the design, implementation, and verification of the system, including experimental results and key insights.

Contents

1 Final Project: Design **2**

1.1 Overview . . . 2

1.2 Actors and Functionalities . . . 2

1.3 Use Case Design . . . 2

1.4 Database Design . . . 3

1.5 Architectural Design . . . 6

1.6 Security Design . . . 7

1.7 Workflow Design . . . 7

1.8 Design Rationale . . . 8

1.9 Formal Models . . . 9

1.9.1 PRISM Model . . . 9

2 Final Project - Implementation **11**

2.1 Source Code . . . 11

3 Appendix: Mini Project **98**

4 References **102**

Chapter 1

Final Project: Design

1.1 Overview

The design of the online voting system is structured to ensure security, reliability, and user-friendly interaction. The system comprises three primary components: the front-end interface for user interaction, the back-end for processing and business logic, and the database for persistent storage. The design emphasizes the principles of modularity, scalability, and security to achieve an efficient and trustworthy voting process.

1.2 Actors and Functionalities

The system identifies two main actors:

- **Voter**: Responsible for logging in, casting a vote, and confirming their choice.
- **Admin**: Manages elections, oversees candidate registration, monitors the voting process, and generates results.

The functionalities for each actor were derived from real-world voting scenarios, focusing on simplicity and adherence to the PRISM security protocol.

1.3 Use Case Design

The use case design revolves around critical functionalities:

- **Login**: Both voters and admins must authenticate themselves securely using unique credentials.

- **Token Validation**: A unique token is issued to each voter to ensure a single vote per user.
- **Vote Casting**: Voters can cast a vote for their preferred candidate, and the vote is securely recorded.
- **Result Generation**: Admins can view and analyze results after the voting period ends.

The use case diagram illustrates these interactions.

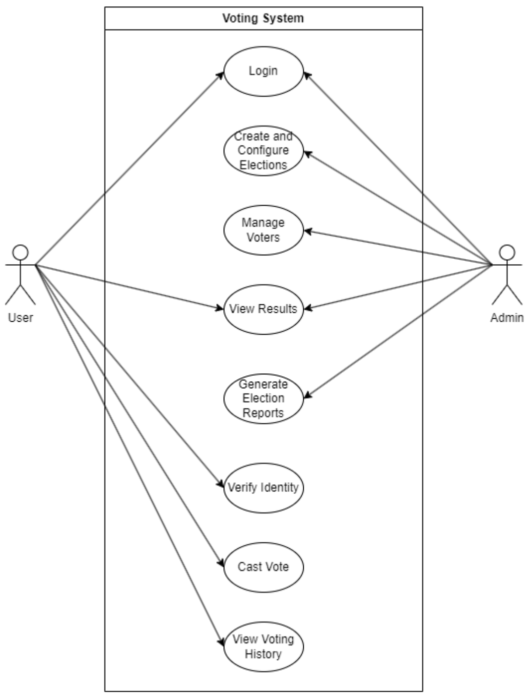


Figure 1.1: Use Case Diagram

1.4 Database Design

The database is designed to store and manage the following key entities:

- **Voter**: Includes details such as voter ID, name, and voting history.
- **Election**: Contains information about elections, including start and end dates and associated candidates.
- **Candidate**: Stores candidate details and their associated elections.

- **Vote**: Tracks each vote cast, associating it with a voter, a candidate, and an election.
- **Token**: Manages unique voting tokens issued to voters.

Each entity is normalized to minimize redundancy and ensure data integrity. The relationships between entities (e.g., voters, votes, and elections) are represented in the class diagram

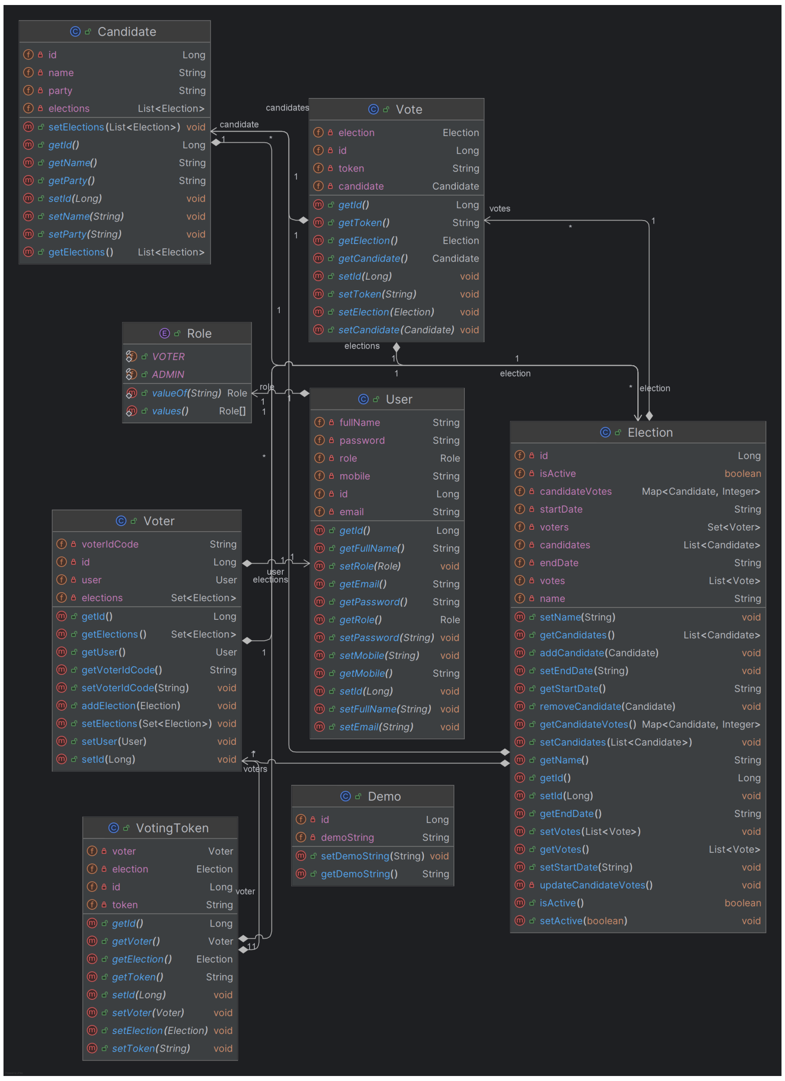


Figure 1.2: Model Layer Class Diagram

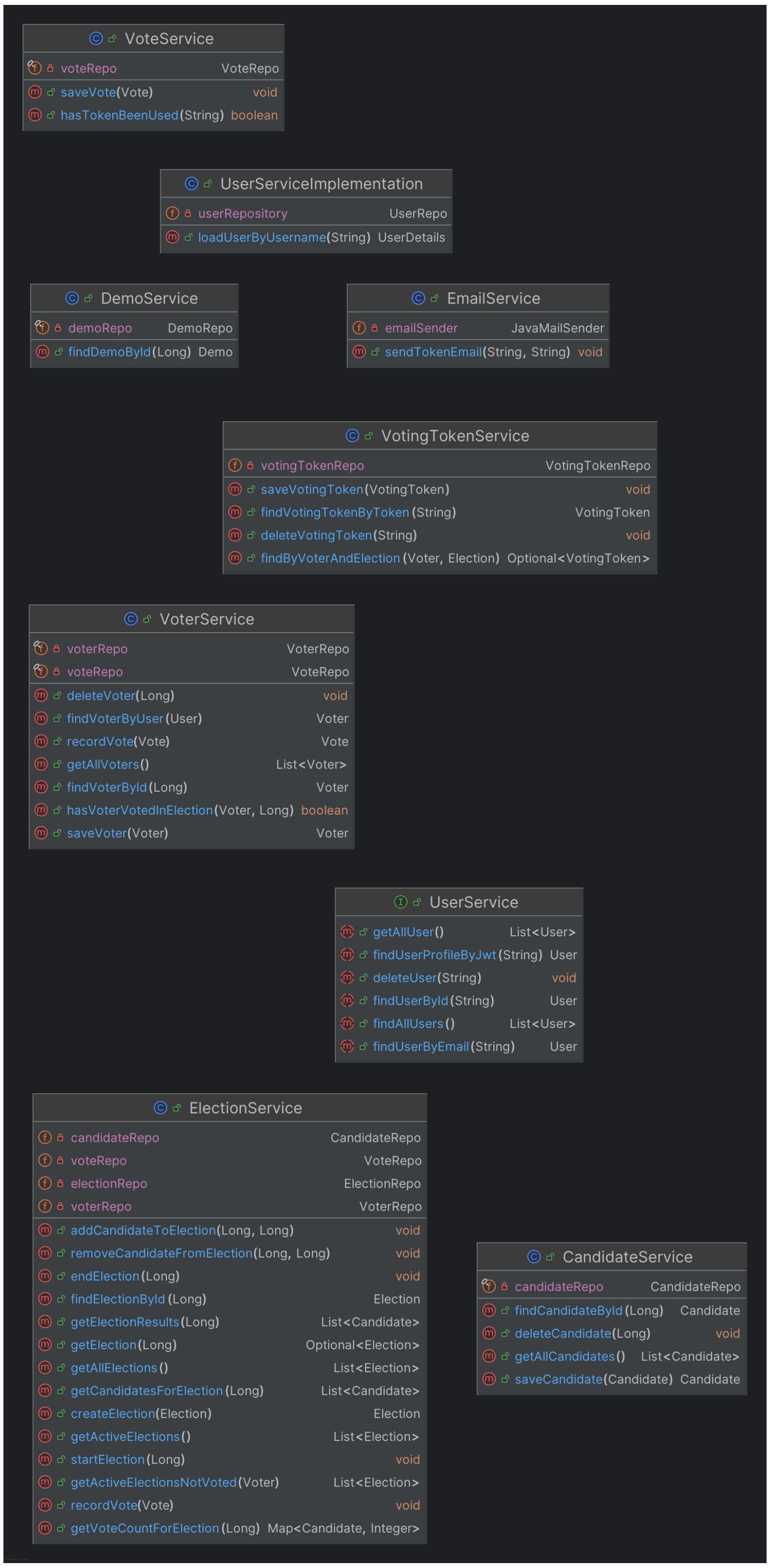
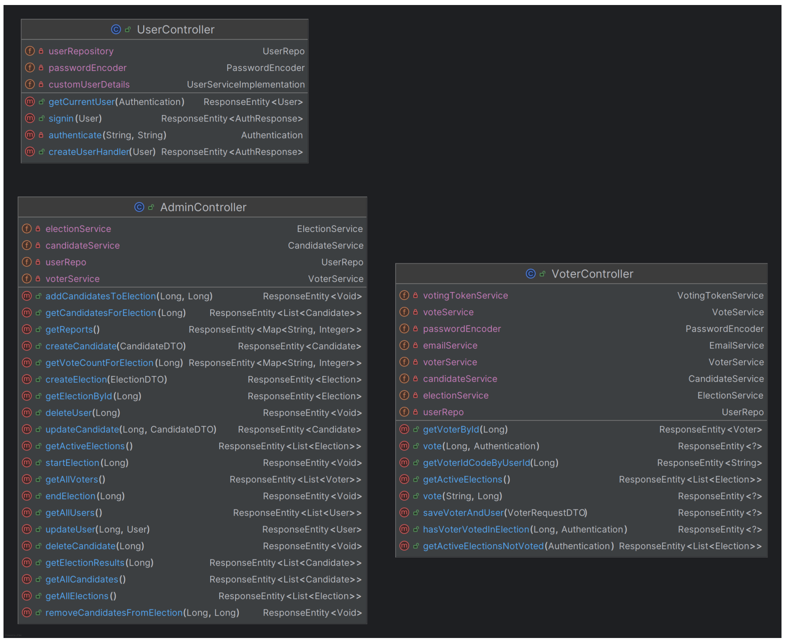

VoteService
voteRepo VoteRepo
saveVote(Vote) void
hasTokenBeenUsed(String) boolean
UserServiceImplementation
userRepository UserRepo
loadUserByUsername(String) UserDetails
DemoService
demoRepo DemoRepo
findDemoById(Long) Demo
EmailService
emailSender JavaMailSender
sendTokenEmail(String, String) void
VotingTokenService
votingTokenRepo VotingTokenRepo
saveVotingToken(VotingToken) void
findVotingTokenByToken(String) VotingToken
deleteVotingToken(String) void
findByVoterAndElection(Voter, Election) Optional<VotingToken>
VoterService
voterRepo VoterRepo
voteRepo VoteRepo
deleteVoter(Long) void
findVoterByUser(User) Voter
recordVote(Vote) Vote
getAllVoters() List<Voter>
findVoterById(Long) Voter
hasVoterVotedInElection(Voter, Long) boolean
saveVoter(Voter) Voter
UserService
getAllUser() List<User>
findUserProfileByJwt(String) User
deleteUser(String) void
findUserById(String) User
findAllUsers() List<User>
findUserByEmail(String) User
ElectionService
candidateRepo CandidateRepo
voteRepo VoteRepo
electionRepo ElectionRepo
voterRepo VoterRepo
addCandidateToElection(Long, Long) void
removeCandidateFromElection(Long, Long) void
endElection(Long) void
findElectionById(Long) Election
getElectionResults(Long) List<Candidate>
getElection(Long) Optional<Election>
getAllElections() List<Election>
getCandidatesForElection(Long) List<Candidate>
createElection(Election) Election
getActiveElections() List<Election>
startElection(Long) void
getActiveElectionsNotVoted(Voter) List<Election>
recordVote(Vote) void
getVoteCountForElection(Long) Map<Candidate, Integer>
CandidateService
candidateRepo CandidateRepo
findCandidateById(Long) Candidate
deleteCandidate(Long) void
getAllCandidates() List<Candidate>
saveCandidate(Candidate) Candidate

Figure 1.3: Service Layer Class Diagram


```
UserController
userRepository                          UserRepo
passwordEncoder                         PasswordEncoder
customUserDetails                       UserServiceImplementation
getCurrentUser(Authentication)          ResponseEntity<User>
signin(User)                            ResponseEntity<AuthResponse>
authenticate(String, String)            Authentication
createUserHandler(User)                 ResponseEntity<AuthResponse>

AdminController
electionService                         ElectionService
candidateService                        CandidateService
userRepo                                UserRepo
voterService                            VoterService
addCandidatesToElection(Long, Long)     ResponseEntity<Void>
getCandidatesForElection(Long)          ResponseEntity<List<Candidate>>
getReports()                            ResponseEntity<Map<String, Integer>>
createCandidate(CandidateDTO)           ResponseEntity<Candidate>
getVoteCountForElection(Long)           ResponseEntity<Map<String, Integer>>
createElection(ElectionDTO)             ResponseEntity<Election>
getElectionById(Long)                   ResponseEntity<Election>
deleteUser(Long)                        ResponseEntity<Void>
updateCandidate(Long, CandidateDTO)     ResponseEntity<Candidate>
getActiveElections()                    ResponseEntity<List<Election>>
startElection(Long)                     ResponseEntity<Void>
getAllVoters()                          ResponseEntity<List<Voter>>
endElection(Long)                       ResponseEntity<Void>
getAllUsers()                           ResponseEntity<List<User>>
updateUser(Long, User)                  ResponseEntity<User>
deleteCandidate(Long)                   ResponseEntity<Void>
getElectionResults(Long)                ResponseEntity<List<Candidate>>
getAllCandidates()                      ResponseEntity<List<Candidate>>
getAllElections()                       ResponseEntity<List<Election>>
removeCandidatesFromElection(Long, Long) ResponseEntity<Void>

VoterController
votingTokenService                      VotingTokenService
voteService                             VoteService
passwordEncoder                         PasswordEncoder
emailService                            EmailService
voterService                            VoterService
candidateService                        CandidateService
electionService                         ElectionService
userRepo                                UserRepo
getVoterById(Long)                      ResponseEntity<Voter>
vote(Long, Authentication)              ResponseEntity<?>
getVoterIdCodeByUserId(Long)            ResponseEntity<String>
getActiveElections()                    ResponseEntity<List<Election>>
vote(String, Long)                      ResponseEntity<?>
saveVoterAndUser(VoterRequestDTO)       ResponseEntity<?>
hasVoterVotedInElection(Long, Authentication) ResponseEntity<?>
getActiveElectionsNotVoted(Authentication) ResponseEntity<List<Election>>
```

Figure 1.4: Controller Layer Class Diagram

1.5 Architectural Design

The system employs a three-tier architecture:

1. **Presentation Layer**: A web-based interface for voters and admins, developed with React, ensures ease of use and responsiveness.

2. **Application Layer**: A Spring Boot backend handles all business logic, including token validation, vote recording, and result computation.

3. **Data Layer**: A PostgreSQL database stores all persistent data securely, with encryption used for sensitive information.

The deployment diagram showcases the system's physical structure, including user devices, web servers, and database servers.

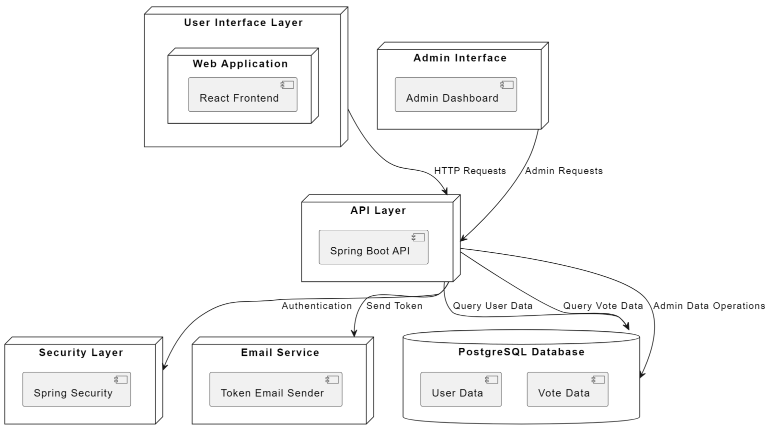


Figure 1.5: Deployment Diagram

1.6 Security Design

Security is a cornerstone of the design, with measures implemented at multiple levels:

- **Authentication**: Ensures that only authorized users can access the system.
- **Token Validation**: Prevents double voting by issuing unique tokens.
- **Data Integrity**: Encrypts sensitive data such as tokens and votes to prevent tampering.
- **Formal Verification**: Utilizes PRISM to model and verify critical properties like token consumption and vote integrity.

The PRISM model (Listing 1.1) represents these security mechanisms formally, ensuring the system meets its requirements.

1.7 Workflow Design

The system's workflow can be summarized as:

1. **Login Phase**:
   - Voter/Admin enters credentials.
   - The system validates credentials and grants access.

2. **Voting Phase**:
    - Voter receives a token and selects a candidate.
    - Token validation ensures the voter can cast only one vote.
    - The vote is recorded in the database and associated with the token.
3. **Result Generation**:
    - Admin ends the election.
    - The system computes and displays results.

The activity diagram outlines this workflow.

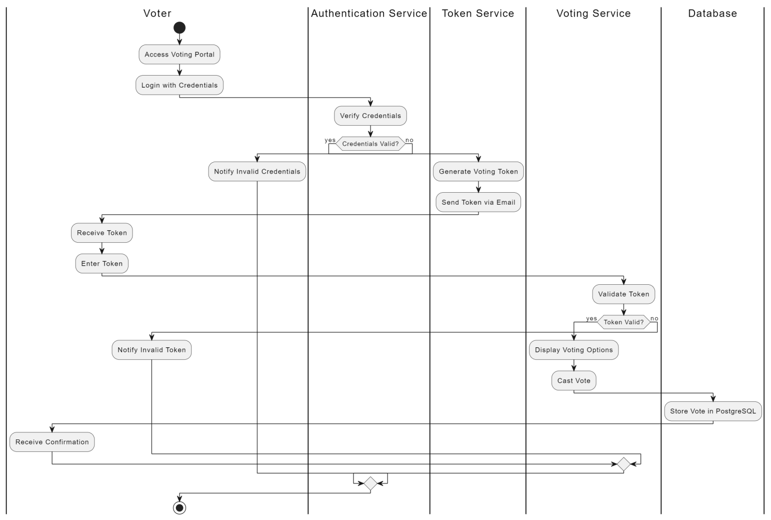


Figure 1.6: Activity Diagram

1.8 Design Rationale

The design decisions were guided by:

- **User Experience**: Ensuring the system is intuitive for both voters and admins.
- **Security**: Protecting voter anonymity and preventing tampering with election results.
- **Modularity**: Allowing future extensions, such as multi-language support or additional analytics features.

1.9 Formal Models

1.9.1 PRISM Model

Listing 1.1: PRISM Model for Voting System

```
dtmc

// Constants
const int MAX_VOTERS = 1;  // Number of voters
const int MAX_TOKENS = 1;  // Number of tokens per voter

// Modules

// Voter module
module Voter
    v : [0..1] init 0;  // 0: Not voted, 1: Voted

    // Logging in
    [login] v=0 -> (v'=0);

    // Casting a vote
    [vote] v=0 -> (v'=1);

    // Voter remains in the "voted" state
    [] v=1 -> (v'=1);
endmodule

// Token module
module Token
    t : [0..1] init 1;  // 1: Valid, 0: Invalid/Consumed

    // Token is consumed when a vote is cast
    [vote] t=1 -> (t'=0);

    // Token remains consumed
    [] t=0 -> (t'=0);
endmodule

// Election module
module Election
    e : [0..1] init 0;  // 0: Election ongoing, 1:
       Election finished
    // Ending the election
    [end] e=0 -> (e'=1);

    // Election remains finished
    [] e=1 -> (e'=1);
endmodule

// Rewards
rewards "votes_cast"
    [vote] true: 1;
endrewards

// Labels
label "all_voted" = (v=1 & t=0);  // All voters have
   voted and tokens are consumed
label "ongoing_election" = (e=0); // Election is still
   ongoing
```

Chapter 2

Final Project - Implementation

2.1 Source Code

```java
package com.example.votingSystem.model;

public enum Role {
    ADMIN, VOTER
}

package com.example.votingSystem.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "app_user")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO,
       generator = "user_seq")
    private Long id;

    private String fullName;
    private String email;

    @JsonProperty(access = JsonProperty.Access.
        WRITE_ONLY)
    private String password;

    @Enumerated(EnumType.STRING)
    private Role role;
    private String mobile;

}

package com.example.votingSystem.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

@Entity
@Getter
@Setter
public class Vote {
    @Id
    @GeneratedValue(strategy =  GenerationType.AUTO)
    private Long id;

    private String token;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "election_id")
    @JsonIgnore  // Prevent cyclic reference from the
        Vote to Election direction
    private Election election;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "candidate_id")
    private Candidate candidate;

    public Vote() {

    }
```

```java

    public Vote(String token,Candidate candidate, Election election) {
         this.token = token;
         this.candidate = candidate;
         this.election = election;
    }

}

package com.example.votingSystem.model;

import com.fasterxml.jackson.annotation.JsonManagedReference;
import jakarta.persistence.*;

import java.util.HashSet;
import java.util.Set;

@Entity
public class Voter {
    @Id
    @GeneratedValue(strategy =  GenerationType.AUTO)
    private Long id;

    @OneToOne
    private User user;

    private String voterIdCode;

    @ManyToMany
    @JoinTable(
            name = "voter_election",
            joinColumns = @JoinColumn(name = "voter_id"),
            inverseJoinColumns = @JoinColumn(name = "election_id")
    )
    @JsonManagedReference  // Prevent circular reference in the Voter to Election direction
    private Set<Election> elections = new HashSet<>();

    public Voter() {

    }

    public Voter(User user, String voterIdCode) {
        this.user = user;
        this.voterIdCode = voterIdCode;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getVoterIdCode() {
        return voterIdCode;
    }

    public void setVoterIdCode(String voterIdCode) {
        this.voterIdCode = voterIdCode;
    }

    public Set<Election> getElections() {
        return elections;
    }

    public void setElections(Set<Election> elections) {
        this.elections = elections;
    }

    public void addElection(Election election) {
        this.elections.add(election);
    }
```

```java
}

package com.example.votingSystem.model;

import com.example.votingSystem.service.VoteService;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class VotingToken {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "voter_id")
    private Voter voter;

    @ManyToOne
    @JoinColumn(name = "election_id")
    private Election election;

    private String token;
}

package com.example.votingSystem.model;

import com.fasterxml.jackson.annotation.
   JsonBackReference;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.ArrayList;
import java.util.List;

```

```

@Entity
@Getter
@Setter
public class Candidate {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    private String party;

    @ManyToMany(mappedBy = "candidates")
    @JsonBackReference
    private List<Election> elections;

    public Candidate() {
    }

    public Candidate(String name, String party) {
        this.name = name;
        this.party = party;
        this.elections = new ArrayList<>();
    }

    // Getters and setters for elections
    public List<Election> getElections() {
        return elections;
    }

    public void setElections(List<Election> elections) {
        this.elections = elections;
    }
}

package com.example.votingSystem.model;

import com.fasterxml.jackson.annotation.
   JsonBackReference;
import com.fasterxml.jackson.annotation.
   JsonManagedReference;
```

```java
import jakarta.persistence.*;
import java.util.*;

@Entity
public class Election {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    private String startDate;

    private String endDate;

    private boolean isActive;

    @ManyToMany
    @JoinTable(
            name = "candidate_election",
            joinColumns = @JoinColumn(name = "election_id"),
            inverseJoinColumns = @JoinColumn(name = "candidate_id")
    )
    @JsonManagedReference  // Prevent circular reference in the Election to Candidate direction
    private List<Candidate> candidates;

    @OneToMany(fetch = FetchType.LAZY)
    @JoinColumn(name = "election_id")
    @JsonBackReference  // Prevent cyclic reference from the Election to Vote direction
    private List<Vote> votes;

    @ManyToMany(mappedBy = "elections")
    @JsonBackReference  // Prevent circular reference in the Election to Voter direction
    private Set<Voter> voters = new HashSet<>();




    // Map to store vote count for each candidate
    @Transient
    private Map<Candidate, Integer> candidateVotes;

    public Election() {
        this.candidates = new ArrayList<>();
        this.votes = new ArrayList<>();
        this.candidateVotes = new HashMap<>();
    }

    public Election(String name, String startDate,
       String endDate, boolean isActive) {
        this.name = name;
        this.startDate = startDate;
        this.endDate = endDate;
        this.isActive = isActive;
        this.candidates = new ArrayList<>();
        this.votes = new ArrayList<>();
        this.candidateVotes = new HashMap<>();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getStartDate() {
        return startDate;
    }

    public void setStartDate(String startDate) {
        this.startDate = startDate;
    }

    public String getEndDate() {
        return endDate;
    }

    public void setEndDate(String endDate) {
        this.endDate = endDate;
    }

    public boolean isActive() {
        return isActive;
    }

    public void setActive(boolean active) {
        isActive = active;
    }

    public List<Candidate> getCandidates() {
        return candidates;
    }

    public void setCandidates(List<Candidate> candidates
       ) {
        this.candidates = candidates;
        updateCandidateVotes();
    }

    public List<Vote> getVotes() {
        return votes;
    }

    public void setVotes(List<Vote> votes) {
        this.votes = votes;
        updateCandidateVotes();
    }

    // Method to count votes and update candidateVotes
       map
    private void updateCandidateVotes() {
        candidateVotes.clear();
        for (Candidate candidate : candidates) {
            candidateVotes.put(candidate, 0);  // Initialize vote count for each candidate
        }

        for (Vote vote : votes) {
            Candidate votedCandidate = vote.getCandidate();
            if (candidateVotes.containsKey(votedCandidate)) {
                candidateVotes.put(votedCandidate, candidateVotes.get(votedCandidate) + 1);
            }
        }
    }

    // Getter for candidate vote count
    public Map<Candidate, Integer> getCandidateVotes() {
        return candidateVotes;
    }

    public void addCandidate(Candidate candidate) {
        this.candidates.add(candidate);
        candidateVotes.put(candidate, 0);  // Initialize vote count when a candidate is added
    }

    public void removeCandidate(Candidate candidate) {
        this.candidates.remove(candidate);
        candidateVotes.remove(candidate);  // Remove vote count when a candidate is removed
    }
}
```

Listing 2.1: Model Layer Code

```java
package com.example.votingSystem.repo;

import com.example.votingSystem.model.Candidate;
import org.springframework.data.jpa.repository.JpaRepository;


public interface CandidateRepo extends JpaRepository<
```

```java
    Candidate, Long> {
}

package com.example.votingSystem.repo;

import com.example.votingSystem.model.Election;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ElectionRepo extends JpaRepository<Election, Long> {
    List<Election> findAllByIsActive(boolean isActive);
}

package com.example.votingSystem.repo;

import com.example.votingSystem.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepo extends JpaRepository<User, Long> {
    User findByEmail(String email);
}

package com.example.votingSystem.repo;

import com.example.votingSystem.model.Election;
import com.example.votingSystem.model.Vote;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface VoteRepo extends JpaRepository<Vote,
    Long> {
    List<Vote> findVotesByElection(Election election);

    boolean existsVoteByToken(String token);
}

package com.example.votingSystem.repo;

import com.example.votingSystem.model.User;
import com.example.votingSystem.model.Voter;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface VoterRepo extends JpaRepository<Voter, Long> {
    Optional<Voter> findByUserId(Long userId);
    Voter findByUser(User user);
    @Query("SELECT CASE WHEN COUNT(v) > 0 THEN TRUE ELSE FALSE END FROM Voter v JOIN v.elections e WHERE v = ?1 AND e.id = ?2")
    boolean hasVoterVotedInElection(Voter voter, Long electionId);
}

package com.example.votingSystem.repo;

import com.example.votingSystem.model.Election;
import com.example.votingSystem.model.Voter;
import com.example.votingSystem.model.VotingToken;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface VotingTokenRepo extends JpaRepository<VotingToken, Long> {

    VotingToken findByToken(String token);

    void deleteByToken(String token);

    Optional<VotingToken> findByVoterAndElection(Voter
       voter, Election election);
}
```

Listing 2.2: Repository Layer Code

```java
package com.example.votingSystem.service;

import com.example.votingSystem.model.Candidate;
import com.example.votingSystem.repo.CandidateRepo;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class CandidateService {
    private final CandidateRepo candidateRepo;

    public CandidateService(CandidateRepo candidateRepo)
        {
        this.candidateRepo = candidateRepo;
    }

    public List<Candidate> getAllCandidates() {
        return candidateRepo.findAll();
    }

    public Candidate findCandidateById(Long id) {
        return candidateRepo.findById(id)
                .orElseThrow(() -> new RuntimeException(
                   "Candidate␣not␣found"));
    }

    public Candidate saveCandidate(Candidate candidate)
       {
        return candidateRepo.save(candidate);
    }

    public void deleteCandidate(Long id) {
        candidateRepo.deleteById(id);
    }
}

package com.example.votingSystem.service;


import com.example.votingSystem.model.Candidate;
import com.example.votingSystem.model.Election;
import com.example.votingSystem.model.Vote;
import com.example.votingSystem.model.Voter;
import com.example.votingSystem.repo.CandidateRepo;
import com.example.votingSystem.repo.ElectionRepo;
import com.example.votingSystem.repo.VoteRepo;
import com.example.votingSystem.repo.VoterRepo;
import org.springframework.beans.factory.annotation.
   Autowired;
import org.springframework.stereotype.Service;

import java.util.*;

@Service
public class ElectionService {

    @Autowired
    private ElectionRepo electionRepo;

    @Autowired
    private CandidateRepo candidateRepo;

    @Autowired
    private VoteRepo voteRepo;
    @Autowired
    private VoterRepo voterRepo;

    // Create a new election
    public Election createElection(Election election) {
        return electionRepo.save(election);
    }

    // Get an election by its ID
    public Optional<Election> getElection(Long
       electionId) {
```

```java
        return electionRepo.findById(electionId);
    }

    // Get all elections
    public List<Election> getAllElections() {
        return electionRepo.findAll();
    }

    // Add candidates to an election
    public void addCandidateToElection(Long electionId, Long candidateId) {
        Optional<Election> electionOptional = electionRepo.findById(electionId);
        if (electionOptional.isPresent()) {
            Election election = electionOptional.get();
            election.addCandidate(candidateRepo.findById(candidateId).orElseThrow());
            electionRepo.save(election);
        }

    }

    // Record a vote
    public void recordVote(Vote vote) {
        voteRepo.save(vote);
    }

    // Get vote count for each candidate in a given election
    public Map<Candidate, Integer> getVoteCountForElection(Long electionId) {
        Optional<Election> electionOptional = electionRepo.findById(electionId);
        if (electionOptional.isPresent()) {
            Election election = electionOptional.get();
            Map<Candidate, Integer> voteCountMap = new HashMap<>();

            // Iterate through the votes and count the votes for each candidate
            List<Vote> votes = voteRepo.findVotesByElection(election);
            for (Vote vote : votes) {
                Candidate candidate = vote.getCandidate();
                voteCountMap.put(candidate, voteCountMap.getOrDefault(candidate, 0) + 1);
            }
            return voteCountMap;
        }
        return null;
    }

    // Get sorted results for candidates in a given election by vote count
    public List<Candidate> getElectionResults(Long electionId) {
        Map<Candidate, Integer> voteCountMap = getVoteCountForElection(electionId);
        if (voteCountMap != null) {
            return voteCountMap.entrySet().stream()
                    .sorted((entry1, entry2) -> entry2.getValue() - entry1.getValue()) // Sort by vote count in descending order
                    .map(Map.Entry::getKey)
                    .toList();
        }
        return null;
    }

    // Get active elections
    public List<Election> getActiveElections() {
        return electionRepo.findAllByIsActive(true);
    }

    public List<Candidate> getCandidatesForElection(Long electionId) {
        Optional<Election> electionOptional = electionRepo.findById(electionId);
        if (electionOptional.isPresent()) {
            Election election = electionOptional.get();
            return election.getCandidates();
        }
        return null;
    }

    public Election findElectionById(Long electionId) {
        return electionRepo.findById(electionId)
                .orElseThrow(() -> new RuntimeException("Election␣not␣found"));
    }

    public List<Election> getActiveElectionsNotVoted(Voter voter) {
        List<Election> activeElections = electionRepo.findAllByIsActive(true);
        Iterator<Election> iterator = activeElections.iterator();
        while (iterator.hasNext()) {
            Election election = iterator.next();
            if (voterRepo.hasVoterVotedInElection(voter, election.getId())) {
                iterator.remove();
            }
        }
        return activeElections;
    }

    public void removeCandidateFromElection(Long electionId, Long candidateId) {
        Optional<Election> electionOptional = electionRepo.findById(electionId);
        if (electionOptional.isPresent()) {
            Election election = electionOptional.get();
            election.removeCandidate(candidateRepo.findById(candidateId).orElseThrow());
            electionRepo.save(election);
        }
    }

    public void startElection(Long electionId) {
        Optional<Election> electionOptional = electionRepo.findById(electionId);
        if (electionOptional.isPresent()) {
            Election election = electionOptional.get();
            election.setActive(true);
            electionRepo.save(election);
        }
    }

    public void endElection(Long electionId) {
        Optional<Election> electionOptional = electionRepo.findById(electionId);
        if (electionOptional.isPresent()) {
            Election election = electionOptional.get();
            election.setActive(false);
            electionRepo.save(election);
        }
    }
}

package com.example.votingSystem.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
public class EmailService {

    @Autowired
    private JavaMailSender emailSender;

    public void sendTokenEmail(String to, String token) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject("Your Voting Token");
        message.setText("Hello, \n\n" +
                "Here is your voting token: " + token + "\n\n" +
                "Please use this token to cast your vote.\n\n" +
                "Thank you.");

        emailSender.send(message);
    }
}
```

```java

package com.example.votingSystem.service;

import com.example.votingSystem.model.User;

import java.util.List;


public interface UserService {


    List<User> getAllUser()  ;

    User findUserProfileByJwt(String jwt);

    User findUserByEmail(String email) ;

    User findUserById(String userId) ;

    List<User> findAllUsers();

    void deleteUser(String userId) ;


}

package com.example.votingSystem.service;

import com.example.votingSystem.model.User;
import com.example.votingSystem.repo.UserRepo;
import org.springframework.beans.factory.annotation.
    Autowired;
import org.springframework.security.core.
    GrantedAuthority;
import org.springframework.security.core.userdetails.
    UserDetails;
import org.springframework.security.core.userdetails.
    UserDetailsService;
import org.springframework.security.core.userdetails.
    UsernameNotFoundException;
import org.springframework.stereotype.Service;


import java.util.ArrayList;
import java.util.List;
import java.util.Optional;


@Service
public class UserServiceImplementation implements UserDetailsService {

    @Autowired
    private UserRepo userRepository;

    public UserServiceImplementation(UserRepo userRepository) {
        this.userRepository=userRepository;
    }


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(username);
        System.out.println(user);

        if(user==null) {
            throw new UsernameNotFoundException("User not found with this email"+username);

        }

        System.out.println("Loaded user: " + user.getEmail() + ", Role: " + user.getRole());
        List<GrantedAuthority> authorities = new ArrayList<>();
        return new org.springframework.security.core.userdetails.User(
                user.getEmail(),
                user.getPassword(),
                authorities);
    }
}

```

```java
package com.example.votingSystem.service;

import com.example.votingSystem.model.User;
import com.example.votingSystem.model.Vote;
import com.example.votingSystem.model.Voter;
import com.example.votingSystem.repo.VoteRepo;
import com.example.votingSystem.repo.VoterRepo;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class VoterService {
    private final VoterRepo voterRepo;
    private final VoteRepo voteRepo;

    public VoterService(VoterRepo voterRepo, VoteRepo voteRepo) {
        this.voterRepo = voterRepo;
        this.voteRepo = voteRepo;
    }

    public Voter findVoterById(Long id) {
        return voterRepo.findById(id)
                .orElseThrow(() -> new RuntimeException("Voter␣not␣found"));

    }

    public Voter saveVoter(Voter voter) {
        return voterRepo.save(voter);
    }

    public Voter findVoterByUser(User user) {
        return voterRepo.findByUser(user);
    }

    public boolean hasVoterVotedInElection(Voter voter, Long electionId) {
        return voterRepo.hasVoterVotedInElection(voter, electionId);
    }

    public Vote recordVote(Vote vote) {
        return voteRepo.save(vote);
    }


    public List<Voter> getAllVoters() {
        return voterRepo.findAll();
    }

    public void deleteVoter(Long id) {
        voterRepo.deleteById(id);
    }
}

package com.example.votingSystem.service;

import com.example.votingSystem.repo.VoteRepo;
import org.springframework.stereotype.Service;


@Service
public class VoteService {
    private final VoteRepo voteRepo;

    public VoteService(VoteRepo voteRepo) {
        this.voteRepo = voteRepo;
    }

    public boolean hasTokenBeenUsed(String token) {
        return voteRepo.existsVoteByToken(token);
    }
}

package com.example.votingSystem.service;

import com.example.votingSystem.model.Election;
import com.example.votingSystem.model.Voter;
import com.example.votingSystem.model.VotingToken;
import com.example.votingSystem.repo.VotingTokenRepo;
import org.springframework.beans.factory.annotation.
   Autowired;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
public class VotingTokenService {
    @Autowired
    private VotingTokenRepo votingTokenRepo;

    public void saveVotingToken(VotingToken votingToken)
        {
        votingTokenRepo.save(votingToken);
    }

    public VotingToken findVotingTokenByToken(String
       token) {
        return votingTokenRepo.findByToken(token);
    }

    public void deleteVotingToken(String token) {
        votingTokenRepo.deleteByToken(token);
    }

    public Optional<VotingToken> findByVoterAndElection(
       Voter voter, Election election) {
        return votingTokenRepo.findByVoterAndElection(
           voter, election);
    }
}
```

Listing 2.3: Service Layer Code

```java
package com.example.votingSystem.controller;

import com.example.votingSystem.dto.CandidateDTO;
import com.example.votingSystem.dto.ElectionDTO;
import com.example.votingSystem.model.*;
import com.example.votingSystem.repo.UserRepo;
import com.example.votingSystem.service.ElectionService;
import com.example.votingSystem.service.CandidateService
   ;
import com.example.votingSystem.service.VoterService;
import org.springframework.beans.factory.annotation.
   Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    @Autowired
    private ElectionService electionService;

    @Autowired
    private CandidateService candidateService;

    @Autowired
    private VoterService voterService;

    @Autowired
    private UserRepo userRepo;


    // Create a new candidate
    @PostMapping("/candidates/create")
    public ResponseEntity<Candidate> createCandidate(
       @RequestBody CandidateDTO candidateDTO) {
        Candidate candidate = new Candidate();
        candidate.setName(candidateDTO.getName());
        candidate.setParty(candidateDTO.getParty());
        Candidate createdCandidate = candidateService.
           saveCandidate(candidate);
        return ResponseEntity.ok(createdCandidate);
    }

    // Get all candidates
    @GetMapping("/candidates")
    public ResponseEntity<List<Candidate>>
       getAllCandidates() {
        List<Candidate> candidates = candidateService.
           getAllCandidates();
        return ResponseEntity.ok(candidates);
    }

    //Update a candidate
    @PutMapping("/candidates/{id}")
    public ResponseEntity<Candidate> updateCandidate(
       @PathVariable Long id, @RequestBody CandidateDTO
       candidateDTO) {
        Candidate candidate = new Candidate();
        candidate.setId(id);
        candidate.setName(candidateDTO.getName());
        candidate.setParty(candidateDTO.getParty());
        Candidate updatedCandidate = candidateService.
           saveCandidate(candidate);
        return ResponseEntity.ok(updatedCandidate);
    }

    // Delete a candidate
    @DeleteMapping("/candidates/{id}")
    public ResponseEntity<Void> deleteCandidate(
       @PathVariable Long id) {
        candidateService.deleteCandidate(id);
        return ResponseEntity.ok().build();
    }

    // Create a new election
    @PostMapping("/elections/create")
    public ResponseEntity<Election> createElection(
       @RequestBody ElectionDTO electionDTO) {
        Election election = new Election();
        election.setName(electionDTO.getName());
        election.setActive(electionDTO.isActive());
        election.setStartDate(electionDTO.getStartDate()
           );
        election.setEndDate(electionDTO.getEndDate());
        Election createdElection = electionService.
           createElection(election);
        return ResponseEntity.ok(createdElection);
    }

    // Get all elections
    @GetMapping("/elections")
    public ResponseEntity<List<Election>>
       getAllElections() {
        List<Election> elections = electionService.
           getAllElections();
```

```java
        return new ResponseEntity<>(elections, org.springframework.http.HttpStatus.OK);
    }

    // Get election by ID
    @GetMapping("/elections/{id}")
    public ResponseEntity<Election> getElectionById(@PathVariable Long id) {
        return electionService.getElection(id)
                .map(ResponseEntity::ok)
                .orElseGet(() -> ResponseEntity.notFound().build());
    }

    // Get active elections
    @GetMapping("/elections/active")
    public ResponseEntity<List<Election>> getActiveElections() {
        List<Election> activeElections = electionService.getActiveElections();
        return ResponseEntity.ok(activeElections);
    }

    // Add candidates to an election
    @PostMapping("/elections/{electionId}/candidates/{candidateId}")
    public ResponseEntity<Void> addCandidatesToElection(@PathVariable Long electionId, @PathVariable Long candidateId) {
        electionService.addCandidateToElection(electionId, candidateId);
        return ResponseEntity.ok().build();
    }

    // Remove candidates from an election
    @DeleteMapping("/elections/{electionId}/candidates/{candidateId}")
    public ResponseEntity<Void> removeCandidatesFromElection(@PathVariable Long electionId, @PathVariable Long candidateId) {
        electionService.removeCandidateFromElection(electionId, candidateId);
        return ResponseEntity.ok().build();
    }

    // Get candidates for an election
    @GetMapping("/elections/{electionId}/candidates")
    public ResponseEntity<List<Candidate>>
       getCandidatesForElection(@PathVariable Long
       electionId) {
        List<Candidate> candidates = electionService.
           getCandidatesForElection(electionId);
        return ResponseEntity.ok(candidates);
    }

    // Get the vote count for each candidate in a given
       election
    @GetMapping("/elections/{electionId}/vote-counts")
    public ResponseEntity<Map<String, Integer>>
       getVoteCountForElection(@PathVariable Long
       electionId) {
        Map<Candidate, Integer> voteCounts =
           electionService.getVoteCountForElection(
           electionId);
        Map<String, Integer> formattedVoteCounts = new
           HashMap<>();
        for (Map.Entry<Candidate, Integer> entry :
           voteCounts.entrySet()) {
            formattedVoteCounts.put(entry.getKey().
               getName() + "␣(" + entry.getKey().
               getParty() + ")", entry.getValue());
        }
        return ResponseEntity.ok(formattedVoteCounts);
    }

    // Get the election results (sorted by vote count)
    @GetMapping("/elections/{electionId}/results")
    public ResponseEntity<List<Candidate>>
       getElectionResults(@PathVariable Long electionId)
        {
        List<Candidate> sortedCandidates =
           electionService.getElectionResults(electionId
           );
        return ResponseEntity.ok(sortedCandidates);
    }

    // Start an election
    @PutMapping("/elections/{electionId}/start")
    public ResponseEntity<Void> startElection(
       @PathVariable Long electionId) {
        electionService.startElection(electionId);
        return ResponseEntity.ok().build();
    }

    // End an election
    @PutMapping("/elections/{electionId}/end")
    public ResponseEntity<Void> endElection(
       @PathVariable Long electionId) {
        electionService.endElection(electionId);
        return ResponseEntity.ok().build();
    }

    // Get all voters
    @GetMapping("/voters")
    public ResponseEntity<List<Voter>> getAllVoters() {
        List<Voter> voters = voterService.getAllVoters()
           ;
        return ResponseEntity.ok(voters);
    }

    // Get all users
    @GetMapping("/users")
    public ResponseEntity<List<User>> getAllUsers() {
        List<User> users = userRepo.findAll();
        return ResponseEntity.ok(users);
    }

    // Update a user
    @PutMapping("/users/{id}")
    public ResponseEntity<User> updateUser(@PathVariable
        Long id, @RequestBody User user) {
        user.setId(id);
        User updatedUser = userRepo.save(user);
        return ResponseEntity.ok(updatedUser);
    }

    // Delete a user
    @DeleteMapping("/users/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable
```

```java
        Long id) {
        User user = userRepo.findById(id).orElseThrow();
        if (user.getRole() == Role.VOTER) {
            Voter voter = voterService.findVoterByUser(
               user);
            voterService.deleteVoter(voter.getId());
        }
        userRepo.deleteById(id);
        return ResponseEntity.ok().build();
    }

}

package com.example.votingSystem.controller;

import com.example.votingSystem.config.JwtProvider;
import com.example.votingSystem.model.Role;
import com.example.votingSystem.model.User;
import com.example.votingSystem.repo.UserRepo;
import com.example.votingSystem.response.AuthResponse;
import com.example.votingSystem.service.UserService;
import com.example.votingSystem.service.
   UserServiceImplementation;
import org.springframework.beans.factory.annotation.
   Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.
   BadCredentialsException;
import org.springframework.security.authentication.
   UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.
   SecurityContextHolder;

import org.springframework.security.core.userdetails.
   UserDetails;
import org.springframework.security.crypto.password.
   PasswordEncoder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
```

```java
public class UserController {

    @Autowired
    private UserRepo userRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;


    @Autowired
    private UserServiceImplementation customUserDetails;


    @PostMapping("/signup")
    public ResponseEntity<AuthResponse>
       createUserHandler(@RequestBody User user)  {
        String email = user.getEmail();
        String password = user.getPassword();
        String fullName = user.getFullName();
        String mobile = user.getMobile();
        String role = String.valueOf(user.getRole());

        User existingUser = userRepository.findByEmail(
           email);
        if(existingUser != null) {
            throw new BadCredentialsException("User␣
               already␣exists");
        }

        User createdUser = new User();
        createdUser.setEmail(email);
        createdUser.setFullName(fullName);
        createdUser.setMobile(mobile);
        createdUser.setRole(Role.valueOf(role));
        createdUser.setPassword(passwordEncoder.encode(
           password));

        User savedUser = userRepository.save(createdUser
           );
        userRepository.save(savedUser);
        Authentication authentication = new
           UsernamePasswordAuthenticationToken(email,
           password);
        SecurityContextHolder.getContext().
```

```
           setAuthentication(authentication);
        String token = JwtProvider.generateToken(
           authentication);


        AuthResponse authResponse = new AuthResponse();
        authResponse.setJwt(token);
        authResponse.setMessage("Register␣Success");
        authResponse.setStatus(true);
        return new ResponseEntity<AuthResponse>(
           authResponse, HttpStatus.OK);
    }





    @PostMapping("/signin")
    public ResponseEntity<AuthResponse> signin(
       @RequestBody User loginRequest) {
        String username = loginRequest.getEmail();
        String password = loginRequest.getPassword();

        System.out.println(username+"-------"+password);

        Authentication authentication = authenticate(
           username,password);
        SecurityContextHolder.getContext().
           setAuthentication(authentication);

        String token = JwtProvider.generateToken(
           authentication);
        AuthResponse authResponse = new AuthResponse();

        authResponse.setMessage("Login␣success");
        authResponse.setJwt(token);
        authResponse.setStatus(true);

        return new ResponseEntity<>(authResponse,
           HttpStatus.OK);
    }


```

```java


    private Authentication authenticate(String username, String password) {

        System.out.println(username+"---++----"+password);

        UserDetails userDetails = customUserDetails.loadUserByUsername(username);

        System.out.println("Sign in in user details"+userDetails);

        if(userDetails == null) {
            System.out.println("Sign in details - null" + userDetails);

            throw new BadCredentialsException("Invalid username and password");
        }
        if(!passwordEncoder.matches(password,userDetails.getPassword())) {
            System.out.println("Sign in userDetails - password mismatch"+userDetails);

            throw new BadCredentialsException("Invalid password");

        }
        return new UsernamePasswordAuthenticationToken(userDetails,null,userDetails.getAuthorities());

    }

    @GetMapping("/me")
    public ResponseEntity<User> getCurrentUser(Authentication authentication) {
        if (authentication == null || !authentication.isAuthenticated()) {
            return new ResponseEntity<>(HttpStatus.UNAUTHORIZED);
        }

        String email = authentication.getName(); // Retrieves the username/email from the authenticated context
        User user = userRepository.findByEmail(email);
        if (user == null) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }

        return new ResponseEntity<>(user, HttpStatus.OK);
    }
}

package com.example.votingSystem.controller;

import com.example.votingSystem.dto.VoterRequestDTO;
import com.example.votingSystem.model.*;
import com.example.votingSystem.repo.UserRepo;
import com.example.votingSystem.service.*;
import jakarta.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

@RestController
@RequestMapping("/api/voter")
public class VoterController {
    @Autowired
    private VoterService voterService;
    @Autowired
    private ElectionService electionService;
    @Autowired
    private UserRepo userRepo;
    @Autowired
    private PasswordEncoder passwordEncoder;
    @Autowired
    private CandidateService candidateService;
    @Autowired
    private VotingTokenService votingTokenService;
    @Autowired
    private EmailService emailService;
    @Autowired
    private VoteService voteService;


    @GetMapping("/{id}")
    public ResponseEntity<Voter> getVoterById(
       @PathVariable Long id) {
        Voter voter = voterService.findVoterById(id);
        return new ResponseEntity<>(voter, HttpStatus.OK
           );
    }

    @PostMapping("/register")
    public ResponseEntity<?> saveVoterAndUser(
       @RequestBody VoterRequestDTO voterRequestDTO) {
        // Check if user already exists
        String email = voterRequestDTO.getEmail();
        User existingUser = userRepo.findByEmail(email);
        if (existingUser != null) {
            return new ResponseEntity<>("User␣already␣
               exists", HttpStatus.BAD_REQUEST);
        }

        // Create new user
        User user = new User();
        user.setEmail(voterRequestDTO.getEmail());
        user.setFullName(voterRequestDTO.getFullName());
        user.setMobile(voterRequestDTO.getMobile());
        user.setPassword(passwordEncoder.encode(
           voterRequestDTO.getPassword())); // Encrypt
           password
        user.setRole(Role.VOTER); // Set role to Voter

```

```java
        User savedUser = userRepo.save(user);

        // Create Voter and associate with the newly created User
        Voter voter = new Voter();
        voter.setUser(savedUser);
        voter.setVoterIdCode(voterRequestDTO.getVoterIdCode());

        Voter savedVoter = voterService.saveVoter(voter);
        return new ResponseEntity<>(savedVoter, HttpStatus.CREATED);
    }

    @GetMapping("/voterIdCode/{id}")
    public ResponseEntity<String> getVoterIdCodeByUserId(@PathVariable Long id) {
        User user = userRepo.findById(id).orElse(null);
        if (user == null) {
            return new ResponseEntity<>("User␣not␣found", HttpStatus.NOT_FOUND);
        }

        Voter voter = voterService.findVoterByUser(user);
        if (voter == null) {
            return new ResponseEntity<>("Voter␣not␣found", HttpStatus.NOT_FOUND);
        }

        return new ResponseEntity<>(voter.getVoterIdCode(), HttpStatus.OK);
    }

    @GetMapping("/active-elections")
    public ResponseEntity<List<Election>> getActiveElections() {
        List<Election> activeElections = electionService.getActiveElections();
        return ResponseEntity.ok(activeElections);
    }

    @GetMapping("/active-elections-not-voted")
    public ResponseEntity<List<Election>>
       getActiveElectionsNotVoted(Authentication
       authentication) {
        // Check if the user is authenticated
        if (authentication == null) {
            return new ResponseEntity<>(HttpStatus.
               UNAUTHORIZED);
        }

        // Get the User object
        String email = authentication.getName();
        User user = userRepo.findByEmail(email);

        if (user == null) {
            return new ResponseEntity<>(HttpStatus.
               NOT_FOUND);
        }

        // Ensure the user is a Voter
        if (user.getRole() != Role.VOTER) {
            return new ResponseEntity<>(HttpStatus.
               FORBIDDEN);
        }

        // Get the Voter object associated with the User
        Voter voter = voterService.findVoterByUser(user)
           ;
        if (voter == null) {
            return new ResponseEntity<>(HttpStatus.
               NOT_FOUND);
        }

        List<Election> activeElections = electionService
           .getActiveElectionsNotVoted(voter);
        return ResponseEntity.ok(activeElections);
    }

    @PostMapping("/generate-token/{electionId}")
    public ResponseEntity<?> vote(@PathVariable Long
       electionId, Authentication authentication) {
        // Check if the user is authenticated
        if (authentication == null) {
            return new ResponseEntity<>("User not logged in", HttpStatus.UNAUTHORIZED);
        }

        // Get the User object
        String email = authentication.getName();
        User user = userRepo.findByEmail(email);

        if (user == null) {
            return new ResponseEntity<>("User not found", HttpStatus.NOT_FOUND);
        }

        // Ensure the user is a Voter
        if (user.getRole() != Role.VOTER) {
            return new ResponseEntity<>("User is not a voter", HttpStatus.FORBIDDEN);
        }

        // Get the Voter object associated with the User
        Voter voter = voterService.findVoterByUser(user);
        if (voter == null) {
            return new ResponseEntity<>("Voter not found", HttpStatus.NOT_FOUND);
        }

        // Check if the voter has already voted in this election
        if (voterService.hasVoterVotedInElection(voter, electionId)) {
            return new ResponseEntity<>("Voter has already voted in this election", HttpStatus.BAD_REQUEST);
        }

        Election election = electionService.findElectionById(electionId);

        Optional<VotingToken> existingToken = votingTokenService.findByVoterAndElection(voter, election);
        if (existingToken.isPresent()) {
            return new ResponseEntity<>("You have already voted or have a pending vote for this election", HttpStatus.BAD_REQUEST);
        }

        String token = UUID.randomUUID().toString();

        VotingToken votingToken = new VotingToken();
        votingToken.setVoter(voter);
        votingToken.setElection(election);
        votingToken.setToken(token);

        votingTokenService.saveVotingToken(votingToken);

        emailService.sendTokenEmail(voter.getUser().getEmail(), token);

        return new ResponseEntity<>("Token generated, please confirm your vote.", HttpStatus.OK);
    }

    @Transactional
    @PostMapping("/vote/{token}/{candidateId}")
    public ResponseEntity<?> vote(@PathVariable String token, @PathVariable Long candidateId) {
        VotingToken votingToken = votingTokenService.findVotingTokenByToken(token);
        if (votingToken == null) {
            return new ResponseEntity<>("Invalid token", HttpStatus.BAD_REQUEST);
        }

        if (voteService.hasTokenBeenUsed(token)) {
            return new ResponseEntity<>("Token has already been used", HttpStatus.BAD_REQUEST);
        }

        Election election = votingToken.getElection();

        Candidate candidate = candidateService.findCandidateById(candidateId);
        if (candidate == null) {
            return new ResponseEntity<>("Candidate␣not␣found", HttpStatus.NOT_FOUND);
        }

        Voter voter = votingToken.getVoter();

        votingTokenService.deleteVotingToken(token);

        Vote vote = new Vote();
        vote.setCandidate(candidate);
        vote.setElection(election);
        vote.setToken(token);

        electionService.recordVote(vote);

        voter.addElection(election);
        voterService.saveVoter(voter);

        return new ResponseEntity<>("Vote␣recorded", HttpStatus.OK);
    }

    @GetMapping("/active-elections/{electionId}")
    public ResponseEntity<?> hasVoterVotedInElection(@PathVariable Long electionId, Authentication authentication) {
        // Check if the user is authenticated
        if (authentication == null) {
            return new ResponseEntity<>("User␣not␣logged␣in", HttpStatus.UNAUTHORIZED);
        }

        // Get the User object
        String email = authentication.getName();
        User user = userRepo.findByEmail(email);

        if (user == null) {
            return new ResponseEntity<>("User␣not␣found", HttpStatus.NOT_FOUND);
        }

        // Ensure the user is a Voter
        if (user.getRole() != Role.VOTER) {
            return new ResponseEntity<>("User is not a voter", HttpStatus.FORBIDDEN);
        }

        // Get the Voter object associated with the User
        Voter voter = voterService.findVoterByUser(user);
        if (voter == null) {
            return new ResponseEntity<>("Voter not found", HttpStatus.NOT_FOUND);
        }

        boolean hasVoted = voterService.hasVoterVotedInElection(voter, electionId);
        return new ResponseEntity<>(hasVoted, HttpStatus.OK);
    }
}
```

Listing 2.4: Controller Layer Code

```java
package com.example.votingSystem.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class CandidateDTO {
    private String name;
    private String party;
}

package com.example.votingSystem.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

```

```java
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class ElectionDTO {
    private String name;
    private boolean isActive;
    private String startDate;
    private String endDate;

}

package com.example.votingSystem.dto;

import com.example.votingSystem.model.Role;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class VoterRequestDTO {

    private String email;
    private String password;
    private String fullName;
    private String mobile;
    private Role role;
    private String voterIdCode;

}
```

Listing 2.5: Data Transfer Objects

```java
package com.example.votingSystem.config;

import jakarta.servlet.http.HttpServletRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.
   Configuration;
import org.springframework.security.config.annotation.
```

```java
    web.builders.HttpSecurity;
import org.springframework.security.config.http.
    SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.
    BCryptPasswordEncoder;
import org.springframework.security.crypto.password.
    PasswordEncoder;
import org.springframework.security.web.
    SecurityFilterChain;
import org.springframework.security.web.authentication.
    www.BasicAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.
    CorsConfigurationSource;

import java.util.Collections;
import java.util.List;

@Configuration
public class ApplicationConfig {

    @SuppressWarnings("deprecation")
    @Bean
    SecurityFilterChain filterChain(HttpSecurity http)
        throws Exception {
        http.sessionManagement(management -> management.
            sessionCreationPolicy(SessionCreationPolicy.
            STATELESS))
                .authorizeRequests(
                        authorize -> authorize
                                .requestMatchers("/api/
                                    voter/register").
                                    permitAll()  // Allow
                                     public access to
                                    register endpoint
                                .requestMatchers("/api/
                                    auth/signin").
                                    permitAll()  // Allow
                                     public access to
                                    login endpoint
                                .requestMatchers("/api
                                    /**").authenticated()
                                      // Secure other API
                                    endpoints
                                .anyRequest().permitAll())
                .addFilterBefore(new JwtTokenValidator(), BasicAuthenticationFilter.class)
                .csrf(csrf -> csrf.disable())
                .cors(cors -> cors.configurationSource(corsConfigurationSource()));

        return http.build();
    }

    private CorsConfigurationSource corsConfigurationSource() {
        return new CorsConfigurationSource() {
            @Override
            public CorsConfiguration getCorsConfiguration(HttpServletRequest request) {
                CorsConfiguration ccfg = new CorsConfiguration();
                ccfg.setAllowedOrigins(List.of("http://localhost:3000"));
                ccfg.setAllowedMethods(Collections.singletonList("*"));
                ccfg.setAllowCredentials(true);
                ccfg.setAllowedHeaders(Collections.singletonList("*"));
                ccfg.setExposedHeaders(List.of("Authorization"));
                ccfg.setMaxAge(3600L);
                return ccfg;
            }
        };
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

package com.example.votingSystem.config;
```

```java

public class JwtConstant {
    public static final String SECRET_KEY = "wpembytrwcvnryxksdbqwjebruyGHyudqgwveytrtrCSnwifoesarjbwe";
    public static final String JWT_HEADER = "Authorization";
}

package com.example.votingSystem.config;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;

import javax.crypto.SecretKey;
import java.util.Collection;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

public class JwtProvider {
    static SecretKey key = Keys.hmacShaKeyFor(JwtConstant.SECRET_KEY.getBytes());

    public static String generateToken(Authentication auth) {
        Collection<? extends GrantedAuthority> authorities = auth.getAuthorities();
        String roles = populateAuthorities(authorities);
        @SuppressWarnings("deprecation")
        String jwt = Jwts.builder()
                .setIssuedAt(new Date())
                .setExpiration(new Date(new Date().getTime()+86400000))
                .claim("email", auth.getName())
                .claim( "authorities",roles)
                .signWith(key)
                .compact();
        System.out.println("Token␣for␣parsing␣in␣
JwtProvider:␣" + jwt);
        return jwt;

    }

    private static String populateAuthorities(Collection<? extends GrantedAuthority> authorities) {
        Set<String> auths = new HashSet<>();
        for(GrantedAuthority authority: authorities) {
            auths.add(authority.getAuthority());
        }
        return String.join(",",auths);
    }


    @SuppressWarnings("deprecation")
    public static String getEmailFromJwtToken(String jwt) {
        jwt = jwt.substring(7); // Assuming "Bearer " is removed from the token
        try {
            //Claims claims=Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(jwt).getBody();
            Claims claims = Jwts.parser().setSigningKey(key).build().parseClaimsJws(jwt).getBody();
            String email = String.valueOf(claims.get("email"));
            System.out.println("Email␣extracted␣from␣JWT:␣" + claims);
            return email;
        } catch (Exception e) {
            System.err.println("Error␣extracting␣email␣from␣JWT:␣" + e.getMessage());
            e.printStackTrace();
            return null;
        }
    }

}

package com.example.votingSystem.config;
```

```java


import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;


import javax.crypto.SecretKey;
import java.io.IOException;
import java.util.List;

public class JwtTokenValidator extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        String jwt = request.getHeader(JwtConstant.JWT_HEADER);
        System.out.println("JWT␣Token␣in␣JwtTokenValidator:␣" + jwt);
        if (jwt != null && jwt.startsWith("Bearer␣")) {
            jwt = jwt.substring(7);

            System.out.println("JWT␣Token␣in␣JwtTokenValidator:␣" + jwt);
            try {
                SecretKey key = Keys.hmacShaKeyFor(JwtConstant.SECRET_KEY.getBytes());
                @SuppressWarnings("deprecation")
                Claims claims = Jwts.parser().setSigningKey(key).build().parseClaimsJws(jwt).getBody();
                System.out.print(claims);

                String email = String.valueOf(claims.get("email"));
                System.out.print(email);
                String authorities = String.valueOf(claims.get("authorities"));
                List<GrantedAuthority> auth = AuthorityUtils.commaSeparatedStringToAuthorityList(authorities);
                Authentication authentication = new UsernamePasswordAuthenticationToken(email, null, auth);
                SecurityContextHolder.getContext().setAuthentication(authentication);

            } catch (Exception e) {
                throw new BadCredentialsException("Invalid␣token", e);
            }
        }

        filterChain.doFilter(request, response);
    }
}

package com.example.votingSystem.response;
import lombok.Data;


@Data
public class ApiResponse {
    private String message;
```

```java
    private boolean status;
    public ApiResponse(String string, boolean b) {

    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public boolean isStatus() {
        return status;
    }
    public void setStatus(boolean status) {
        this.status = status;
    }

}

package com.example.votingSystem.response;


import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
public class AuthResponse {
    private String jwt;
    private String message;
    private Boolean status;

    public String getJwt() {
        return jwt;
    }

    public void setJwt(String jwt) {
        this.jwt = jwt;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public Boolean getStatus() {
        return status;
    }

    public void setStatus(Boolean status) {
        this.status = status;
    }
}

package com.example.votingSystem.exceptions;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.
   ControllerAdvice;
import org.springframework.web.bind.annotation.
   ExceptionHandler;

@ControllerAdvice
public class GlobalExceptionsHandler {
    @ExceptionHandler(RuntimeException.class)
    public ResponseEntity<String> handleRuntimeException
       (RuntimeException ex) {
        return new ResponseEntity<>(ex.getMessage(),
           HttpStatus.BAD_REQUEST);
    }
}

package com.example.votingSystem;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.
   SpringBootApplication;

@SpringBootApplication
public class VotingSystemApplication {

        public static void main(String[] args) {
                SpringApplication.run(
                   VotingSystemApplication.class, args);
        }

}
```

Listing 2.6: Config Code

```
spring.application.name=votingSystem
spring.datasource.url=jdbc:postgresql://localhost:5432/
   votingSystem
spring.datasource.username=postgres
spring.datasource.password=admin
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.
   dialect.PostgreSQLDialect
# Session Management Configuration
spring.security.filter.chain.content-negotiation.
   parameter-strategy=ignore
spring.security.filter.chain.any-request.authorized=
   permitAll
spring.security.filter.chain.request-matcher.path.
   pattern=/api/**
spring.security.filter.chain.request-matcher.path.
   authenticated=true
# CSRF Configuration
spring.security.csrf.disabled=true
# CORS Configuration
spring.security.cors.configurationSource.allowedOrigins=
   http://localhost:3000
spring.security.cors.configurationSource.allowedMethods
   =*
spring.security.cors.configurationSource.allowedHeaders
   =*
spring.security.cors.configurationSource.
   allowCredentials=true
spring.security.cors.configurationSource.exposedHeaders=
   Authorization
spring.security.cors.configurationSource.maxAge=3600
# Mail Configuration
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=
spring.mail.password=
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.com
```

Listing 2.7: Application Properties

Listing 2.8: View Layer Code

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import LoginPage from './components/LoginPage';
import SignupPage from './components/SignupPage';
import Dashboard from './components/Dashboard';
import ProfilePage from './components/ProfilePage';
import AdminDashboard from './components/AdminDashboard';
import ManageElections from './components/ManageElections';
import ManageCandidates from './components/ManageCandidates';
import './App.css'; // Import your CSS file here
import AdminRoute from './components/AdminRoute';
import ManageUsers from './components/ManageUsers';


function App() {
  return (
      <div className="App">
      <Router>
            <Routes>
                <Route path="/" element={<LoginPage />} />
                <Route path="/signup" element={<SignupPage />} />
                <Route path="/dashboard" element={<Dashboard />} />
                <Route path="/profile" element={<ProfilePage />} />
                <Route path="/admin/dashboard" element={<AdminRoute element={<AdminDashboard />} />} />
                <Route path="/admin/manage-elections" element={<AdminRoute element={<ManageElections />} />} />
                <Route path="/admin/manage-candidates" element={<AdminRoute element={<ManageCandidates />} />} />
                <Route path="/admin/manage-users" element
```

```
                   ={<AdminRoute element={<ManageUsers />} />} />
            </Routes>
      </Router>
      </div>
  );
}

export default App;

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function AdminDashboard() {
    const [elections, setElections] = useState([]);
    const [selectedElectionResults, setSelectedElectionResults] = useState([]);
    const [selectedElectionId, setSelectedElectionId] = useState(null);
    const navigate = useNavigate();

    useEffect(() => {
        // Fetch all elections
        fetchElections();
    }, []);

    const fetchElections = async () => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const response = await axios.get("http://localhost:8080/api/admin/elections", {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            setElections(response.data);
        } catch (error) {
            console.error("Error fetching elections:", error);
        }
    };

    const fetchElectionResults = async (electionId) => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const response = await axios.get(`http://
               localhost:8080/api/admin/elections/${
               electionId}/vote-counts`, {
                headers: {
                    'Authorization': `Bearer ${token}`  //
                        Pass JWT token
                }
            });
            console.log("Election results fetched:",
               response.data);
            const results = Object.entries(response.data).
               map(([candidate, voteCount]) => ({
                candidate,
                voteCount
            }));
            setSelectedElectionResults(results);
        } catch (error) {
            console.error("Error fetching election
               results:", error);
            setSelectedElectionResults([]); // Ensure
               selectedElectionResults is an array even if
                there's an error
        }
    };

    const handleElectionClick = (electionId) => {
        setSelectedElectionId(electionId);
        fetchElectionResults(electionId);
    };

    return (
        <div>
            <nav className="navbar navbar-expand-lg navbar
               -light bg-light">
                <div className="container-fluid">
                    <a className="navbar-brand" href="#">
                       Admin Dashboard</a>
                    <button className="navbar-toggler"
                       type="button" data-bs-toggle="
                       collapse" data-bs-target="#
                       navbarNav" aria-controls="navbarNav
                       " aria-expanded="false" aria-label=
                       "Toggle navigation">
                        <span className="navbar-toggler-
                           icon"></span>
                    </button>
                    <div className="collapse navbar-
                       collapse" id="navbarNav">
                        <ul className="navbar-nav ms-auto"
```

```
                                >
                                <li className="nav-item">
                                    <button className="btn btn-link nav-link" onClick={() => navigate('/admin/manage-users')}>Manage Users</button>
                                </li>
                                <li className="nav-item">
                                    <button className="btn btn-link nav-link" onClick={() => navigate('/admin/manage-elections')}>Manage Elections</button>
                                </li>
                                <li className="nav-item">
                                    <button className="btn btn-link nav-link" onClick={() => navigate('/admin/manage-candidates')}>Manage Candidates</button>
                                </li>
                                <li className="nav-item">
                                    <button className="btn btn-link nav-link" onClick={() => {
                                        localStorage.removeItem('token');
                                        navigate('/');
                                    }}>Logout</button>
                                </li>
                            </ul>
                        </div>
                    </div>
                </nav>
                <div className="container mt-5">
                    <h1>Welcome to the Admin Dashboard</h1>
                    <p>Here you can manage elections, users, and view reports.</p>
                    <div className="mt-5">
                        <h2>All Elections</h2>
                        <ul className="list-group">
                            {elections.map((election) => (
                                <li key={election.id} className="list-group-item" onClick={() => handleElectionClick(
                                election.id)}>
                                {election.name} (from {
                                    election.startDate} to
                                    {election.endDate})
                                {election.id ===
                                    selectedElectionId &&
                                    selectedElectionResults.length
                                     > 0 && (
                                     <ul>
                                         {
                                             selectedElectionResults.map
                                             ((result, index
                                             ) => (
                                              <li key={index
                                                  }>
                                                  {
                                                      result.candidate
                                                      } - {
                                                      result.voteCount
                                                      } votes
                                              </li>
                                         ))}
                                     </ul>
                                )}
                            </li>
                        ))}
                    </ul>
                </div>
            </div>
        </div>
    );
}

export default AdminDashboard;

import React, { useEffect, useState } from 'react';
import { Navigate } from 'react-router-dom';
import { getCurrentUser } from './UserService';

function AdminRoute({ element: Component, ...rest }) {
    const [isAdmin, setIsAdmin] = useState(false);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const checkUserRole = async () => {
            try {
                const user = await getCurrentUser();
                setIsAdmin(user.role === 'ADMIN');
            } catch (error) {
                setIsAdmin(false);
            } finally {
                setLoading(false);
            }
        };

        checkUserRole();
    }, []);

    if (loading) {
        return <div>Loading...</div>; // or a loading spinner
    }

    return isAdmin ? Component : <Navigate to="/dashboard" />;
}

export default AdminRoute;

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function Dashboard() {
    const [activeElections, setActiveElections] = useState([]);
    const [filter, setFilter] = useState('all'); // 'all' or 'notVoted'
    const [error, setError] = useState('');
    const navigate = useNavigate();

    useEffect(() => {
        // Fetch active elections
        fetchActiveElections();
    }, [filter]);

    const fetchActiveElections = async () => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const endpoint = filter === 'all' ? "http://localhost:8080/api/voter/active-elections" : "http://localhost:8080/api/voter/active-elections-not-voted";
            const response = await axios.get(endpoint, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
```

```
          }
        });
        console.log("Active elections fetched:", response.data);
        const electionsWithVoteStatus = await Promise.all(response.data.map(async (election) => {
          const electionDetails = await axios.get(`http://localhost:8080/api/voter/active-elections/${election.id}`, {
            headers: {
              'Authorization': `Bearer ${token}` // Pass JWT token
            }
          });
          return { ...election, hasVoted: electionDetails.data };
        }));
        setActiveElections(electionsWithVoteStatus);
        console.log("Active elections with vote status:", electionsWithVoteStatus);
      } catch (error) {
        console.error("Error fetching active elections:", error);
        setActiveElections([]); // Ensure activeElections is an array even if there's an error
      }
    };

    const handleVote = async (electionId, candidateId) => {
      try {
        const token = localStorage.getItem('token'); // Retrieve the token from local storage
        const response = await axios.post(`http://localhost:8080/api/voter/generate-token/${electionId}`, {}, {
          headers: {
            'Authorization': `Bearer ${token}`  // Pass JWT token
          }
        });
        alert(response.data);
        const voteToken = prompt("Enter the vote token:");
        const voteResponse = await axios.post(`http://localhost:8080/api/voter/vote/${voteToken}/${candidateId}`, {}, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Vote successful:", voteResponse.data);
            setError("");
            alert(voteResponse.data);
            fetchActiveElections(); // Refresh the elections after voting
        } catch (error) {
            console.error("Error casting vote:", error);
            setError("Failed to cast vote.");
        }
    };

    const toggleFilter = () => {
        setFilter(filter === 'all' ? 'notVoted' : 'all');
    };

    return (
        <div>
            <nav className="navbar navbar-expand-lg navbar-light bg-light">
                <div className="container-fluid">
                    <a className="navbar-brand" href="#">Voter Dashboard</a>
                    <button className="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                        <span className="navbar-toggler-icon"></span>
                    </button>
                    <div className="collapse navbar-collapse" id="navbarNav">
                        <ul className="navbar-nav ms-auto">
                            <li className="nav-item">
                                <button className="btn btn-link nav-link" onClick={() => navigate('/profile')}>Profile</button>
                                <button className="btn btn-link nav-link" onClick
```

```
                       ={() => {
                        localStorage.removeItem
                           ('token');
                        navigate('/');
                    }}>Logout</button>
                </li>
            </ul>
        </div>
    </div>
</nav>
<div className="container mt-5">
    <h1>Active Elections</h1>
    {error && <p style={{ color: 'red' }}>{
       error}</p>}
    <button className="btn btn-secondary mb-3"
        onClick={toggleFilter}>
        {filter === 'all' ? 'Show Elections
           Not Voted' : 'Show All Elections'}
    </button>
    <ul className="list-group">
        {activeElections.map((election) => (
            <li key={election.id} className="
               list-group-item">
                <h3>{election.name} (from {
                   election.startDate} to {
                   election.endDate})</h3>
                <ul>
                    {election.candidates.map((
                       candidate) => (
                        <li key={candidate.id
                           }>
                            {candidate.name}
                               ({
                               candidate.party
                               })
                            {!
                               election.hasVoted
                                && (
                                <button
                                   className="
                                   btn btn-
                                   primary ml
                                   -3" onClick
                                   ={() =>
                                   handleVote(
                                   election.id
                                   ,
                                   candidate.id
                                   )}>Vote</
```

```
                                               button>
                                        )}
                                    </li>
                                ))}
                            </ul>
                        </li>
                    ))}
                </ul>
            </div>
        </div>
    );
}

export default Dashboard;

import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import {
    MDBContainer,
    MDBInput,
    MDBBtn,
} from 'mdb-react-ui-kit';
import { getCurrentUser } from './UserService';

function LoginPage() {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');
    const navigate = useNavigate();

    const handleLogin = async () => {
        try {
            if (!email || !password) {
                setError('Please enter both email and password.');
                return;
            }
            const response = await axios.post('http://localhost:8080/api/auth/signin', { email, password });
            console.log('Login successful:', response.data);

            // Store JWT token in localStorage
            localStorage.setItem('token', response.data.jwt);

            // Fetch the current user after successful
```

```
               login
            const userResponse = await getCurrentUser(); // Await the user data
            console.log('User:', userResponse);

            // Navigate based on user role
            if (userResponse.role === 'ADMIN') {
                navigate('/admin/dashboard');
            } else {
                navigate('/dashboard');
            }
        } catch (error) {
            console.error('Login failed:', error.response ? error.response.data : error.message);
            setError('Invalid email or password.');
        }
    };
    return (
        <div className="d-flex justify-content-center align-items-start vh-100">
            <div className="border rounded-lg p-4" style={{ maxWidth: '400px', width: '100%' }}>
                <MDBContainer className="p-3">
                    <h2 className="mb-4 text-center">Login</h2>
                    <MDBInput
                        wrapperClass='mb-4'
                        placeholder='Email'
                        id='email'
                        value={email}
                        type='email'
                        onChange={(e) => setEmail(e.target.value)}
                    />
                    <MDBInput
                        wrapperClass='mb-4'
                        placeholder='Password'
                        id='password'
                        type='password'
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                    />
                    {error && <p className="text-danger text-center">{error}</p>}
                    <MDBBtn
                        className="mb-4 w-100"
                        color="primary"
                        size="lg"
                        onClick={handleLogin}
                    >
                        Sign In
                    </MDBBtn>
                    <div className="text-center">
                        <p>Not a member? <a href="/signup"
                            >Register</a></p>
                    </div>
                </MDBContainer>
            </div>
        </div>
    );
}

export default LoginPage;

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function ManageCandidates() {
    const [candidates, setCandidates] = useState([]);
    const [newCandidateName, setNewCandidateName] =
       useState("");
    const [newCandidateParty, setNewCandidateParty] =
       useState("");
    const [editCandidateId, setEditCandidateId] = useState
       (null);
    const [editCandidateName, setEditCandidateName] =
       useState("");
    const [editCandidateParty, setEditCandidateParty] =
       useState("");
    const [error, setError] = useState("");
    const navigate = useNavigate();

    useEffect(() => {
        // Fetch all candidates
        fetchCandidates();
    }, []);

    const fetchCandidates = async () => {
        try {
            const token = localStorage.getItem('token');
               // Retrieve the token from local storage
            const response = await axios.get("http://
               localhost:8080/api/admin/candidates", {
                headers: {
                    'Authorization': `Bearer ${token}`  //
                        Pass JWT token
                }
            });
            setCandidates(response.data);
        } catch (error) {
            console.error("Error fetching candidates:", error);
        }
    };

    const handleCreateCandidate = async () => {
        if (!newCandidateName || !newCandidateParty) {
            setError("All fields are required.");
            return;
        }
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const candidateData = {
                name: newCandidateName,
                party: newCandidateParty
            };
            console.log("Creating candidate:", candidateData);
            const response = await axios.post("http://localhost:8080/api/admin/candidates/create", candidateData, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Candidate created successfully:", response.data);
            setCandidates([...candidates, response.data]);
            setNewCandidateName("");
            setNewCandidateParty("");
            setError("");
        } catch (error) {
            console.error("Error creating candidate:", error);
            setError("Failed to create candidate.");
        }
    };

    const handleDeleteCandidate = async (candidateId) => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            await axios.delete(`http://localhost:8080/api/
                admin/candidates/${candidateId}`, {
                 headers: {
                     'Authorization': `Bearer ${token}`  //
                         Pass JWT token
                 }
            });
            setCandidates(candidates.filter(candidate =>
                candidate.id !== candidateId));
        } catch (error) {
            console.error("Error deleting candidate:",
                error);
            alert("Failed to delete candidate. ");
            setError("Failed to delete candidate.");
        }
    };

    const handleEditCandidate = async () => {
        if (!editCandidateName || !editCandidateParty) {
            setError("All fields are required.");
            return;
        }
        try {
            const token = localStorage.getItem('token');
                // Retrieve the token from local storage
            const candidateData = {
                 name: editCandidateName,
                 party: editCandidateParty
            };
            console.log("Editing candidate:",
                candidateData);
            const response = await axios.put(`http://
                localhost:8080/api/admin/candidates/${
                editCandidateId}`, candidateData, {
                 headers: {
                     'Authorization': `Bearer ${token}`  //
                         Pass JWT token
                 }
            });
            console.log("Candidate edited successfully:",
                response.data);
            setCandidates(candidates.map(candidate =>
                candidate.id === editCandidateId ?
                response.data : candidate));
            setEditCandidateId(null);
            setEditCandidateName("");
            setEditCandidateParty("");
            setError("");
        } catch (error) {
            console.error("Error editing candidate:",
                error);
            setError("Failed to edit candidate.");
        }
    };

    const startEditCandidate = (candidate) => {
        setEditCandidateId(candidate.id);
        setEditCandidateName(candidate.name);
        setEditCandidateParty(candidate.party);
    };

    return (
        <div>
            <nav className="navbar navbar-expand-lg navbar-light bg-light">
                <div className="container-fluid">
                    <a className="navbar-brand" href="#">Manage Candidates</a>
                    <button className="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                        <span className="navbar-toggler-icon"></span>
                    </button>
                    <div className="collapse navbar-collapse" id="navbarNav">
                        <ul className="navbar-nav ms-auto">
                            <li className="nav-item">
                                <button className="btn btn-link nav-link" onClick={() => navigate('/admin/dashboard')}>Back to Dashboard</button>
                            </li>
                        </ul>
                    </div>
                </div>
            </nav>
            <div className="container mt-5">
                <h1>Manage Candidates</h1>
                <div className="mt-4">
                    <h2>Add New Candidate</h2>
                    {error && <p style={{ color: 'red' }}>{error}</p>}
                    <form onSubmit={(e) => {
                        e.preventDefault();
                        handleCreateCandidate(); }}>
                         <div className="mb-3">
                             <label className="form-label">
                                Candidate Name</label>
                             <input
                                 type="text"
                                 className="form-control"
                                 value={newCandidateName}
                                 onChange={(e) =>
                                    setNewCandidateName(
                                    e.target.value)}
                             />
                         </div>
                         <div className="mb-3">
                             <label className="form-label">
                                Party</label>
                             <input
                                 type="text"
                                 className="form-control"
                                 value={newCandidateParty}
                                 onChange={(e) =>
                                    setNewCandidateParty(
                                    e.target.value)}
                             />
                         </div>
                         <button type="submit" className="
                            btn btn-primary">Add Candidate
                            </button>
                     </form>
                 </div>
                 <div className="mt-5">
                     <h2>All Candidates</h2>
                     <ul className="list-group">
                         {candidates.map((candidate) => (
                             <li key={candidate.id}
                                className="list-group-item
                                d-flex justify-content-
                                between align-items-center"
                                >
                                 <span>{candidate.name} ({
                                    candidate.party})</span
                                    >
                                 <div>
                                     <button className="btn
                                        btn-secondary btn-
                                        sm me-2" onClick
                                        ={() =>
                                        startEditCandidate(
                                        candidate)}>Edit</
                                        button>
                                     <button className="btn
                                         btn-danger btn-sm"
                                         onClick={() =>
                                        handleDeleteCandidate
                                        (candidate.id)}>
                                        Delete</button>
                                </div>
                            </li>
                        ))}
                    </ul>
                </div>
                {editCandidateId && (
                    <div className="mt-5">
                        <h2>Edit Candidate</h2>
                        <form onSubmit={(e) => {
                            e.preventDefault();
                            handleEditCandidate(); }}>
                            <div className="mb-3">
                                 <label className="form-
                                    label">Candidate Name</
                                    label>
                                 <input
                                     type="text"
                                     className="form-
                                        control"
                                     value={
                                        editCandidateName}
                                     onChange={(e) =>
                                        setEditCandidateName
                                        (e.target.value)}
                                 />
                            </div>
                            <div className="mb-3">
                                 <label className="form-
                                    label">Party</label>
                                 <input
                                     type="text"
                                     className="form-
                                        control"
                                     value={
                                        editCandidateParty}
                                     onChange={(e) =>
                                        setEditCandidateParty
                                        (e.target.value)}
                                 />
                            </div>
                            <button type="submit"
                                className="btn btn-primary">Save</button>
                            <button type="button" className="btn btn-secondary ms-2" onClick={() => setEditCandidateId(null)}>Cancel</button>
                        </form>
                    </div>
                )}
            </div>
        </div>
    );
}

export default ManageCandidates;

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function ManageElections() {
    const [elections, setElections] = useState([]);
    const [candidates, setCandidates] = useState([]);
    const [newElectionName, setNewElectionName] = useState('');
    const [newElectionStartDate, setNewElectionStartDate] = useState('');
    const [newElectionEndDate, setNewElectionEndDate] = useState('');
    const [selectedElectionId, setSelectedElectionId] = useState('');
    const [selectedCandidateId, setSelectedCandidateId] = useState('');
    const [error, setError] = useState('');
    const navigate = useNavigate();

    useEffect(() => {
        fetchElections();
        fetchCandidates();
    }, []);

    const fetchElections = async () => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const response = await axios.get("http://localhost:8080/api/admin/elections", {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            setElections(response.data);
        } catch (error) {
            console.error("Error fetching elections:", error);
        }
    };

    const fetchCandidates = async () => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const response = await axios.get("http://localhost:8080/api/admin/candidates", {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            setCandidates(response.data);
        } catch (error) {
            console.error("Error fetching candidates:", error);
        }
    };

    const handleCreateElection = async () => {
        if (!newElectionName || !newElectionStartDate || !newElectionEndDate) {
            setError("All fields are required.");
            return;
        }
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const electionData = {
                name: newElectionName,
                startDate: newElectionStartDate,
                endDate: newElectionEndDate
            };
            console.log("Creating election:", electionData);
            const response = await axios.post("http://localhost:8080/api/admin/elections/create", electionData, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Election created successfully:", response.data);
            setElections([...elections, response.data]);
            setNewElectionName("");
            setNewElectionStartDate("");
            setNewElectionEndDate("");
            setError("");
        } catch (error) {
            console.error("Error creating election:", error);
            setError("Failed to create election.");
        }
    };

    const handleAddCandidateToElection = async () => {
        if (!selectedElectionId || !selectedCandidateId) {
            setError("All fields are required.");
            return;
        }
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const response = await axios.post(`http://localhost:8080/api/admin/elections/${selectedElectionId}/add-candidate`, { candidateId: selectedCandidateId }, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Candidate added to election successfully:", response.data);
            fetchElectionCandidates(selectedElectionId);
            setSelectedCandidateId("");
            setError("");
        } catch (error) {
            console.error("Error adding candidate to election:", error);
            setError("Failed to add candidate to election.");
        }
    };

    const handleRemoveCandidateFromElection = async (
candidateId) => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            await axios.delete(`http://localhost:8080/api/admin/elections/${selectedElectionId}/remove-candidate/${candidateId}`, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Candidate removed from election successfully");
            fetchElectionCandidates(selectedElectionId);
        } catch (error) {
            console.error("Error removing candidate from election:", error);
            setError("Failed to remove candidate from election.");
        }
    };

    const handleStartElection = async (electionId) => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            await axios.put(`http://localhost:8080/api/admin/elections/${electionId}/start`, {}, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Election started successfully");
            fetchElections();
        } catch (error) {
            console.error("Error starting election:", error);
            setError("Failed to start election.");
        }
    };

    const handleEndElection = async (electionId) => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            await axios.put(`http://localhost:8080/api/admin/elections/${electionId}/end`, {}, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Election ended successfully");
            fetchElections();
        } catch (error) {
            console.error("Error ending election:", error);
            setError("Failed to end election.");
        }
    };

    const fetchElectionCandidates = async (electionId) => {
        try {
            const token = localStorage.getItem('token');  // Retrieve the token from local storage
            const response = await axios.get(`http://localhost:8080/api/admin/elections/${electionId}/candidates`, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            setCandidates(response.data);
        } catch (error) {
            console.error("Error fetching election candidates:", error);
        }
    };

    const getAvailableCandidates = () => {
        return candidates.filter(candidate => !candidate.elections || !candidate.elections.includes(selectedElectionId));
    };

    return (
        <div>
            <nav className="navbar navbar-expand-lg navbar-light bg-light">
                <div className="container-fluid">
                    <a className="navbar-brand" href="#">Manage Elections</a>
                    <button className="navbar-toggler"
                type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                    <span className="navbar-toggler-icon"></span>
                </button>
                <div className="collapse navbar-collapse" id="navbarNav">
                    <ul className="navbar-nav ms-auto">
                        <li className="nav-item">
                            <button className="btn btn-link nav-link" onClick={() => navigate('/admin/dashboard')}>Back to Dashboard</button>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
        <div className="container mt-5">
            <h1>Manage Elections</h1>
            <div className="mt-4">
                <h2>Create New Election</h2>
                {error && <p style={{ color: 'red' }}>{error}</p>}
                <form onSubmit={(e) => { e.preventDefault(); handleCreateElection(); }}>
                    <div className="mb-3">
                        <label className="form-label">Election Name</label>
                        <input
                            type="text"
                            className="form-control"
                            value={newElectionName}
                            onChange={(e) => setNewElectionName(e.target.value)}
                        />
                    </div>
                    <div className="mb-3">
                        <label className="form-label">Start Date</label>
                        <input
                            type="date"
```

```
                                    className="form-control"
                                    value={
                                       newElectionStartDate}
                                    onChange={(e) =>
                                       setNewElectionStartDate
                                       (e.target.value)}
                                />
                            </div>
                            <div className="mb-3">
                                <label className="form-label">
                                   End Date</label>
                                <input
                                    type="date"
                                    className="form-control"
                                    value={newElectionEndDate}
                                    onChange={(e) =>
                                       setNewElectionEndDate(
                                       e.target.value)}
                                />
                            </div>
                            <button type="submit" className="
                               btn btn-primary">Create
                               Election</button>
                        </form>
                    </div>
                    <div className="mt-5">
                        <h2>Add Candidate to Election</h2>
                        {error && <p style={{ color: 'red'
                           }}>{error}</p>}
                        <form onSubmit={(e) => {
                           e.preventDefault();
                           handleAddCandidateToElection(); }}>
                            <div className="mb-3">
                                <label className="form-label">
                                   Select Election</label>
                                <select
                                    className="form-control"
                                    value={selectedElectionId}
                                    onChange={(e) => {
                                        setSelectedElectionId(
                                           e.target.value);
                                        fetchElectionCandidates
                                           (e.target.value);
                                    }}
                                >
                                    <option value="">Select an
                                        election</option>
                                    {Array.isArray(elections)
                                       && elections.map((
                                        election) => (
                                         <option key={
                                            election.id} value
                                            ={election.id}>
                                             {election.name} (
                                                from {
                                                election.startDate
                                                } to {
                                                election.endDate
                                                })
                                         </option>
                                    ))}
                                </select>
                            </div>
                            <div className="mb-3">
                                <label className="form-label">
                                   Select Candidate</label>
                                <select
                                    className="form-control"
                                    value={selectedCandidateId
                                       }
                                    onChange={(e) =>
                                       setSelectedCandidateId(
                                       e.target.value)}
                                >
                                    <option value="">Select a
                                       candidate</option>
                                    {Array.isArray(
                                       getAvailableCandidates
                                       ()) &&
                                       getAvailableCandidates
                                       ().map((candidate) => (
                                         <option key={
                                            candidate.id} value
                                            ={candidate.id}>
                                             {candidate.name}
                                                ({
                                                candidate.party
                                                })
                                         </option>
                                    ))}
                                </select>
                            </div>
                            <button type="submit" className="
                               btn btn-primary">Add Candidate
                               </button>
                        </form>
                    </div>
                    <div className="mt-5">
```

```
                        <h2>Manage Elections</h2>
                        <ul className="list-group">
                            {elections.map((election) => (
                                <li key={election.id}
                                   className="list-group-item"
                                   >
                                    <h3>{election.name} (from
                                       {election.startDate} to
                                        {election.endDate})</
                                       h3>
                                    <button className="btn btn
                                       -success me-2" onClick
                                       ={() =>
                                       handleStartElection(
                                       election.id)}>Start
                                       Election</button>
                                    <button className="btn btn
                                       -danger me-2" onClick
                                       ={() =>
                                       handleEndElection(
                                       election.id)}>End
                                       Election</button>
                                    <h4>Candidates</h4>
                                    <ul className="list-group"
                                       >
                                        {
                                           election.candidates.map
                                           ((candidate) => (
                                            <li key={
                                               candidate.id}
                                               className="list
                                               -group-item d-
                                               flex justify-
                                               content-between
                                                align-items-
                                               center">
                                                <span>{
                                                   candidate.name
                                                   } ({
                                                   candidate.party
                                                   })</span>
                                                <button
                                                   className="
                                                   btn btn-
                                                   danger btn-
                                                   sm" onClick
                                                   ={() =>
                                                   handleRemoveCandidateFromEle
                                                   (
                                               candidate.id)}>Remove</button>
                                        </li>
                                    ))}
                                </ul>
                            </li>
                        ))}
                    </ul>
                </div>
            </div>
        </div>
    );
}

export default ManageElections;

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function ManageUsers() {
    const [users, setUsers] = useState([]);
    const [editUserId, setEditUserId] = useState(null);
    const [editUserFullName, setEditUserFullName] = useState('');
    const [editUserEmail, setEditUserEmail] = useState('');
    const [editUserMobile, setEditUserMobile] = useState('');
    const [editUserRole, setEditUserRole] = useState('VOTER');
    const [error, setError] = useState('');
    const navigate = useNavigate();

    useEffect(() => {
        fetchUsers();
    }, []);

    const fetchUsers = async () => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const response = await axios.get("http://localhost:8080/api/admin/users", {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("Users fetched:", response.data);
            setUsers(response.data);
        } catch (error) {
            console.error("Error fetching users:", error);
        }
    };

    const handleDeleteUser = async (userId) => {
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            await axios.delete(`http://localhost:8080/api/admin/users/${userId}`, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            setUsers(users.filter(user => user.id !== userId));
        } catch (error) {
            console.error("Error deleting user:", error);
            setError("Failed to delete user.");
        }
    };

    const handleEditUser = async () => {
        if (!editUserFullName || !editUserEmail || !editUserRole) {
            setError("All fields are required.");
            return;
        }
        try {
            const token = localStorage.getItem('token'); // Retrieve the token from local storage
            const userData = {
                fullName: editUserFullName,
                email: editUserEmail,
                mobile: editUserMobile,
                role: editUserRole
            };
            console.log("Editing user:", userData);
            const response = await axios.put(`http://localhost:8080/api/admin/users/${editUserId}`, userData, {
                headers: {
                    'Authorization': `Bearer ${token}`  // Pass JWT token
                }
            });
            console.log("User edited successfully:", response.data);
            setUsers(users.map(user => user.id === editUserId ? response.data : user));
            setEditUserId(null);
            setEditUserFullName("");
            setEditUserEmail("");
            setEditUserRole("VOTER");
            setError("");
        } catch (error) {
            console.error("Error editing user:", error);
            setError("Failed to edit user.");
        }
    };

    const startEditUser = (user) => {
        setEditUserId(user.id);
        setEditUserFullName(user.fullName);
        setEditUserEmail(user.email);
        setEditUserMobile(user.mobile);
        setEditUserRole(user.role);
    };

    return (
        <div>
            <nav className="navbar navbar-expand-lg navbar-light bg-light">
                <div className="container-fluid">
                    <a className="navbar-brand" href="#">Manage Users</a>
                    <button className="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                        <span className="navbar-toggler-icon"></span>
                    </button>
                    <div className="collapse navbar-collapse" id="navbarNav">
                        <ul className="navbar-nav ms-auto">
                            <li className="nav-item">
                                <button className="btn btn-link nav-link" onClick={() => navigate('/
                                        admin/dashboard')}>Back
                                         to Dashboard</button>
                            </li>
                        </ul>
                    </div>
                </div>
            </nav>
            <div className="container mt-5">
                <h1>Manage Users</h1>
                <div className="mt-5">
                    <h2>All Users</h2>
                    {error && <p style={{ color: 'red'
                        }}>{error}</p>}
                    <ul className="list-group">
                        {users.map((user) => (
                            <li key={user.id} className="
                               list-group-item d-flex
                               justify-content-between
                               align-items-center">
                                <span>{user.fullName} ({
                                   user.email}) - {
                                   user.role}</span>
                                <div>
                                    <button className="btn
                                        btn-secondary btn-
                                       sm me-2" onClick
                                       ={() =>
                                       startEditUser(user)
                                       }>Edit</button>
                                    <button className="btn
                                        btn-danger btn-sm"
                                        onClick={() =>
                                       handleDeleteUser(
                                       user.id)}>Delete</
                                       button>
                                </div>
                            </li>
                        ))}
                    </ul>
                </div>
                {editUserId && (
                    <div className="mt-5">
                        <h2>Edit User</h2>
                        <form onSubmit={(e) => {
                            e.preventDefault();
                            handleEditUser(); }}>
                            <div className="mb-3">
                                <label className="form-
                                   label">Full Name</label
```

```
                                        >
                                    <input
                                        type="text"
                                        className="form-
                                           control"
                                        value={
                                           editUserFullName}
                                        onChange={(e) =>
                                           setEditUserFullName
                                           (e.target.value)}
                                    />
                                </div>
                                <div className="mb-3">
                                    <label className="form-
                                         label">Email</label>
                                    <input
                                        type="email"
                                        className="form-
                                           control"
                                        value={editUserEmail}
                                        onChange={(e) =>
                                           setEditUserEmail(
                                           e.target.value)}
                                    />
                                </div>
                                <div className="mb-3">
                                    <label className="form-
                                         label">Mobile</label>
                                    <input
                                        type="text"
                                        className="form-
                                           control"
                                        value={editUserMobile}
                                        onChange={(e) =>
                                           setEditUserMobile(
                                           e.target.value)}
                                    />
                                </div>
                                <div className="mb-3">
                                    <label className="form-
                                         label">Role</label>
                                    <select
                                        className="form-
                                           control"
                                        value={editUserRole}
                                        onChange={(e) =>
                                           setEditUserRole(
                                           e.target.value)}
                                    >
                                    <option value="VOTER">VOTER</option>
                                    <option value="ADMIN">ADMIN</option>
                                </select>
                            </div>
                            <button type="submit" className="btn btn-primary">Save</button>
                            <button type="button" className="btn btn-secondary ms-2" onClick={() => setEditUserId(null)}>Cancel</button>
                        </form>
                    </div>
                )}
            </div>
        </div>
    );
}

export default ManageUsers;

import React, { useState, useEffect } from 'react';
import { getCurrentUser } from './UserService';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';

function ProfilePage() {
    const navigate = useNavigate();
    const [userData, setUserData] = useState({
        id: '',
        fullName: '',
        email: '',
        mobile: '',
        voterIdCode: '',
    });
    const [error, setError] = useState('');

    useEffect(() => {
        fetchUserProfile();
    }, []);

    const fetchUserProfile = async () => {
        try {
            const data = await getCurrentUser();
            console.log('User data:', data);
            const token = localStorage.getItem('token');
```

```
                // Retrieve the token from local storage
            const response = await axios.get(`http://localhost:8080/api/voter/voterIdCode/${data.id}`, {

                headers: {
                'Authorization': `Bearer ${token}`  // Pass JWT token
                }});

            console.log('Voter ID Code:', response.data);
            data.voterIdCode = response.data;
            setUserData(data);

        } catch (err) {
            console.error(err);
            setError('Failed to fetch profile');
        }
    };

    const handleBackToDashboard = () => {
        navigate('/dashboard');
    };

    return (
        <div>
            <h1>Profile Page</h1>
            {error && <p style={{ color: 'red' }}>{error}</p>}
            <div>
                <p><strong>Full Name:</strong> {userData.fullName}</p>
                <p><strong>Email:</strong> {userData.email}</p>
                <p><strong>Mobile:</strong> {userData.mobile}</p>
                <p><strong>Voter ID Code:</strong> {userData.voterIdCode}</p>
                <button onClick={handleBackToDashboard}>Back to Dashboard</button>
            </div>
        </div>
    );
}

export default ProfilePage;

import React, { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { MDBContainer, MDBInput } from 'mdb-react-ui-kit';

function SignupPage() {
    const [fullName, setFullName] = useState('');
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [confirmPassword, setConfirmPassword] = useState
        ('');
    const [mobile, setMobile] = useState(''); // renamed
        state
    const [role] = useState('VOTER'); // role remains
        fixed as 'VOTER'
    const [voterIdCode, setVoterIdCode] = useState('');
    const [error, setError] = useState(''); // State to
        manage error messages
    const history = useNavigate(); // Navigation hook

    const handleSignup = async () => {
        try {
            // Check for empty fields
            if (!fullName || !email || !password || !
                confirmPassword || !mobile || !voterIdCode)
                 {
                setError('Please fill in all fields.');
                return;
            }

            if (password !== confirmPassword) {
                setError("Passwords do not match");
                return;
            }

            // Send data to the backend to create both
                user and voter
            const response = await axios.post('http://
                localhost:8080/api/voter/register', {
                fullName,
                email,
                password,
                mobile,
                role,
                voterIdCode
            });

            console.log("Voter and user registered
                successfully:", response.data);

            localStorage.setItem('token',
                response.data.jwt);

            //Show success message
            alert('Signup successful!');

            // Redirect to login page after successful
                signup
            history('/'); // Redirect after successful
                signup

        } catch (error) {
            // Handle signup error
            console.error('Signup failed:', error.response
                ? error.response.data : error.message);
            setError(error.response ? error.response.data
                : error.message);
        }
    };

    return (
        <div className="d-flex justify-content-center
            align-items-center vh-100">
            <div className="border rounded-lg p-4" style
                ={{ width: '600px', height: 'auto' }}>
                <MDBContainer className="p-3">
                    <h2 className="mb-4 text-center">Sign
                        Up as a Voter</h2>
                    {error && <p className="text-danger">{
                        error}</p>}
                    <MDBInput
                        wrapperClass="mb-3"
                        id="fullName"
                        placeholder="Full Name"
                        value={fullName}
                        type="text"
                        onChange={(e) => setFullName(
                            e.target.value)}
                    />
                    <MDBInput
                        wrapperClass="mb-3"
                        placeholder="Email Address"
                        id="email"
                        value={email}
                        type="email"
                        onChange={(e) => setEmail(
                            e.target.value)}
                    />
                    <MDBInput
                        wrapperClass="mb-3"
```

```
    placeholder="Password"
    id="password"
    type="password"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
/>
<MDBInput
    wrapperClass="mb-3"
    placeholder="Confirm Password"
    id="confirmPassword"
    type="password"
    value={confirmPassword}
    onChange={(e) => setConfirmPassword(e.target.value)}
/>
<MDBInput
    wrapperClass="mb-2"
    placeholder="Mobile Number"
    id="mobile"
    value={mobile}
    type="text"  // Changed to 'tel' type for better validation
    onChange={(e) => setMobile(e.target.value)}
/>
<MDBInput
    wrapperClass="mb-2"
    placeholder="Voter ID Code"
    id="voterIdCode"
    value={voterIdCode}
    type="text"
    onChange={(e) => setVoterIdCode(e.target.value)}
/>
<button
    className="mb-4 d-block mx-auto fixed-action-btn btn-primary"
    style={{ height: '40px', width: '100%' }}
    onClick={handleSignup}
>
    Sign Up
</button>
<div className="text-center">
    <p>Already registered? <a href="/">Login</a></p>
</div>
```

```
                </MDBContainer>
            </div>
        </div>
    );
}

export default SignupPage;

import axios from 'axios';

export const getCurrentUser = async () => {
    const token = localStorage.getItem('token');
    try {
        const response = await axios.get('http://
            localhost:8080/api/auth/me', {
            headers: {
                'Authorization': `Bearer ${token}`,
            },
        });
        return response.data;
    } catch (error) {
        console.error('Error fetching current user:',
            error);
        throw error;
    }
};
```

Chapter 3

Appendix: Mini Project

```java
package com.example.votingSystem.model;

import jakarta.persistence.*;

import java.io.Serializable;

@Entity
public class Demo implements Serializable {
    @Id
    @GeneratedValue(strategy =  GenerationType.AUTO)
    private Long id;
    private String demoString;

    public Demo() {

    }

    public Demo(String demoString) {
        this.demoString = demoString;
    }

    public String getDemoString() {
        return demoString;
    }

    public void setDemoString(String demoString) {
        this.demoString = demoString;
    }
}
```

```java

package com.example.votingSystem.repo;

import com.example.votingSystem.model.Demo;
import org.springframework.data.jpa.repository.
   JpaRepository;

public interface DemoRepo extends JpaRepository<Demo,
   Long> {
}
package com.example.votingSystem.service;

import com.example.votingSystem.model.Demo;
import com.example.votingSystem.repo.DemoRepo;
import org.springframework.beans.factory.annotation.
   Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.
   Transactional;


@Service
public class DemoService {
    private final DemoRepo demoRepo;

    @Autowired
    public DemoService(DemoRepo demoRepo) {
        this.demoRepo = demoRepo;
    }

    @Transactional(readOnly = true)
    public Demo findDemoById(Long demoId) {
        return demoRepo.findById(demoId).orElse(null);
    }
}

package com.example.votingSystem.controller;

import com.example.votingSystem.model.Demo;
import com.example.votingSystem.service.DemoService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;


@RestController
public class DemoController {
    private final DemoService demoService;

    public DemoController(DemoService demoService) {
        this.demoService = demoService;
    }

    @GetMapping("/demo")
    @CrossOrigin(origins = "http://localhost:3000")
    public ResponseEntity<Demo> getDemoById() {
        Demo demo = demoService.findDemoById(1L);
        return new ResponseEntity<>(demo, HttpStatus.OK);
    }

}

import React, { useEffect, useState } from 'react';

function App() {
    const [message, setMessage] = useState("");

    useEffect(() => {
        fetch("http://localhost:8080/demo")
            .then((response) => response.json())
            .then((data) => setMessage(data.demoString))
            .catch((error) => console.error("Error␣fetching␣message:", error));
    }, []);

    return (
        <div>
            <h1>Backend Message:</h1>
            <p>{message}</p>
            </div>
      );
}

export default App;
```

Listing 3.1: Mini Project Code

Chapter 4

References

- Official Spring Boot Documentation: `https://spring.io/projects/spring-boot`.
- PRISM Model Checker: `https://www.prismmodelchecker.org/`.